# A First Course in Computer Programming Laboratory Manual CG, DS

212 🖷															
2104	■213														
2080211	205	•214													
2064	■209	2034	•215												
198●207	201 <sup>,</sup>	●204	189	●216											
1964	∎199	1904	•202	187	•217										
194 电 197	1914	•200	1857	<b>•</b> 188	182 (	●218									
192	∎195			183	•186	1804	●219								
166 🖶 193			1754	<b>●</b> 184	1784	●181	139●	220							
1644	∎167			173	•176	1404	<b>■</b> 179	1374	∎221						
162 电 165	1594	•168	1714	<b>■</b> 174	1414	●177	135 🖷	138	132 🖷	222					
1604	∎163	1574	●169	142	•172			1334	<b>1</b> 36	130¶	J223				
152 🛛 161	1554	•158	1434	<b>●</b> 170			125 🖷	134	128	131	1164	∎224			
1504	<b>1</b> 53	1444	<b>•</b> 156					1234	∎126	117 🖷	129	114	•225		
148•151	145 (	•154					121•	124	118•	127	1124	<b>•</b> 115	109	•226	
1464	∎149							1194	<b>■</b> 122			1104	•113	1074	•227
66●147							93 🖷	120			1024	<b>1</b> 11	105	•108	228•
644	₿67							914	∎94			1004	•103	1064	●229
62 🛭 65	594	●68					89•	92	86¶	195	984	•101	104/	•230	
60 <b>4</b>	₿63	574	●69					874	₿90	84 🖷	196	994	•231		
52 061	55+	•58	434	∎70			79•	88	82 🖷	185	974	●232			
504	₿53	444	●56	41	•71			774	₿80	83 🖷	233				
48●51	45+	●54	394	∎42	364	∎72	75 🖷	78	81•	1234					
464	∎49			374	●40	344	₽73	764	₽235						
20●47			294	∎38	32+	<b>■</b> 35	74•	236							
184	₽21			271	•30	334	₽237								
16●19	13+	•22	254	₿28	314	₽238									
14	∎17	11•	₽23	26	●239										
6015	94	•12	244	∎240											
44	₽7	104	■241												
2 • 5	84	•242													
34	₽243														
1-244															

# Contents

1 Entrance	1
2 Lab 1: Hello World! Hello You!	7
3 Presentation: Microworlds, Distributed Cognition, Learning to Learn	11
4 Lab 2: Hello Painter! Hello Composer!	13
5 Presentation: Using Objects - Painters and Composers	23
6 Lab 3: Establishing a CS1 Work Site	25
7 Presentation: Command Line OS, Text Editing, The Web	33
8 Lab 4: Expressions and Shapes World Problem Solving	35
9 Presentation: Basic Data Types, Problem Solving Strategies	41
10 Lab 5: An Interpreter Featuring Loop Forever and Selection	43
11 Presentation: Selection (if) and Repetition (while)	49
12 Lab 6: Functions and Commands	51
13 Presentation: Application Architectures	57
14 Lab 7: String Thing	59
15 Presentation: Strings	67
16 Lab 8: Array Play	69
17 Presentation: Arrays, File IO, the for Statement	77
18 Lab 9: Simple List Processing	79
19 Presentation: Generics and Simple List Processing	87
20 Lab 10: Establishing and Using Classes	89
21 Presentation: Class Definition, Interface Implementation	97
22 Lab 11: Modeling Objects with Classes	99
23 Presentation: Data Modeling, Abstract Classes	105
24 Lab 12: Grapheme to Color Synesthesia	107
25 Presentation: Synesthesia, Chromesthesia	113
26 Lab 13: Chromesthesia	115
27 Presentation: L-Systems and Fractals	125
28 Lab 14: Fun with Fractals	127
29 Exit	159

30 Appendix 1: Nonrepresentational Painting World (NPW)	161
31 Appendix 2: Modular Melody World (MMW)	169
32 Resources and References	175

## 1 Entrance

### Marcel Proust on WISDOM

We do not receive wisdom, we must discover it for ourselves, after a journey through the wilderness which no one else can make for us, which no one can spare us, for our wisdom is the point of view from which we come at last to regard the world.

### Marvin Minsky on MICROWORLDS

In doing this (working in the world of children's blocks), we'll try to imitate how Galileo and Newton learned so much by studying the simplest kinds of pendulums and weights, mirrors and prisms. Our study of how to build with blocks will be like focusing a microscope on the simplest objects we can find, to open up a great and unexpected universe. It is the same reason why so many biologists today devote more attention to tiny germs and viruses than to magnificent lions and tigers. For me and a whole generation of students, the world of work with children's blocks has been the prism and the pendulum for studying intelligence. In science, one can learn the most by studying what seems the least.

#### Overview

The text you are currently reading represents a first course in computer programming that features the Java programming language. More precisely, it substantially represents the *laboratory component* of the course, while only nominally representing the *presentational component* of the course, and merely alluding to the *programming component* of the course. What makes this course somewhat distinctive is that the approach is grounded in a number of cognitively oriented thematic threads, namely *educational microworlds*, *distributed cognition*, and elements of *the learning sciences*.

### Orientation of the Text

One of Seymour Papert's many catchy observations is that in order to think about thinking you have to think about thinking about something. The same basic idea applies to the process of learning to program. You need to write programs about something, in the context of some domain, so that you can think about your programming and reflect deeply upon the wide range of phenomena surrounding the programming of computers. In this text a number of domains that are closely associated with creative acts will serve as computational contexts for programming activities and reflections. One of the domains is *music*. Another of the domains is *nonrepresentational art*. A third domain involves objects of chance. Modest computational learning environments associated with each of these domains have been crafted in support of this course. A **computational learning environment** is simply a collection of computational objects that are thematically related, and which collectively afford opportunities to computationally explore domain specific ideas in particularly productive ways. By grounding your study of computer programming in these domains, particularly music and nonrepresentational art, you will be more likely to wind up viewing computer science as a creative activity than you would if your learning were tied to the fields more traditionally associated with first courses in computer programming, the STEM fields of science, technology, engineering, and math. There is a move afoot to add an A to STEM. The A in STEAM is for the arts! There is nothing new about the notion

that the arts can fuel the fire of innovation in science, technology, engineering, and math. The MIT Media lab was founded on this premise. A number of colleges and universities embrace this idea (MIT, NJIT, and Drexel are among the leaders in doing so), and various projects are dedicated to the proposition (for example, EarSketch). Yet there hardly appears to be a tidal wave of enthusiasm for flavoring STEM curricula, or even first computer science courses, with the arts. The course of study determined by this text is clearly sympathetic to those who are championing STEAM in educational circles.

### Structure of the Text

The course for which this text was written is something of an inversion to the norm, in that its cornerstones are *experiences* in the laboratory rather than *explanations* in the lecture. Both are essential, but in a significant sense the laboratory experiences are intended to drive the course. What you do with this laboratory manual will determine, to a large extent, the degree to which you actually realize a *meaningful* first course in computer science. If you merely read it, you won't get much out of the course. If you fully engage in the laboratories, which requires faithfully attending lectures and studiously sorting out the material presented during the lectures, in order to position yourself to be successful with the labs, you will probably learn something substantial about the nature of computer programming. If, in addition, you tackle each programming assignment like you mean it, and craft a program/demo archival work site to be proud of, you could very well find that you will have taken a first step on the road to becoming a computer scientist!

This text is structured as a sequence of *laboratories* separated by references to *presentations* (references to lectures), all preceded by an *entrance* and followed by an *exit*. The laboratory parts, the essential parts of this text, are intended to be held in hand. This will facilitate your engagement in the laboratory activities. In this manual, the references to presentations are meant to remind you that you really will be missing out on essential material should you miss a lecture. The classroom *experience* of declarative lectures laced with procedural demonstrations, framed by all of the cognitive phenomena surrounding these knowledge oriented activities, is simply not something that you can make up.

The bipartite nature of this text, the inclusion of *labs* and the referencing of *presentations*, is represented explicitly in the table of contents. The thematic threads that permeate this text (and the course), on the other hand, find expression in a much more implicit manner. The next three sections of this *Entrance* are presented in an effort to prepare you to appreciate these thematic threads as you work your way through the course, especially when you are doing the labs and the programming challenges.

### Microworlds

A **microworld** is a limited collection of objects in a limited environment that can be manipulated in a limited number of ways. Microworlds, being rather small in scope, can fairly easily be tailored to suit particular needs. Microworlds that are easy to understand without much explicit instruction, yet which are rich enough to be interesting, can be effective educational tools. In this text, the term *microworld* will mean *computational educational microworld*, which simply means that the microworlds considered have been designed with learning in mind and will manifest as virtual worlds. For most practical purposes, the term *microworld* can be used synonymously with the phrase *computational learning environment*. Interestingly, it is considered sport, among some (skeptics of AI, in particular), to criticise microworlds because the don't *scale up*. Yet the critiques generally fail to acknowledge, much less appreciate, the fact that microworlds are not so much intended to scale up as they are intended to help you, and your ideas, scale up!

Two microworlds will take center stage in this course, one featuring simple geometric shapes, and the other featuring basic musical notes. The former is called the Nonrepresentational Painting World, or NPW. The latter is called the Modular Melody World, or MMW. A third microworld that features coins and dice, called the Chance World, will make occasional appearances throughout the course.

## **Distributed** Cognition

According to Roy Pea (1997), **distributed cognition** "is the conception of cognition as something accomplished through collaborative interactions involving people and artifacts, as opposed to something possessed by individuals in isolation." Most definitions of the concept are more or less consistent with this one. Some perspective, and some elaboration, will help to make clear how this notion plays a role in computer science, and how it will be featured in the course for which this text was written.

Cognitive science is closely associated with the premise that cognition can best be understood in terms of representations and transformations of those representations. This is the **computational/representational assumption** that serves to provide a substantial element of cohesion to the field. Cognitive science is also closely associated with the idea that the mind can best be studied by approaching it from a diversity of disciplinary perspectives, and then endeavoring to integrate the findings that accrue. This is the **interdisciplinary assumption** which adds breadth to the field. These two foundational assumptions can be viewed as defining characteristics of **traditional cognitive science**. Almost from the start, a number of individuals within the contributing disciplines to cognitive science have been highly critical of the computational/representational assumption, suggesting that is is much too limiting (Dreyfus, 1979; Searle, 1980; Winograd & Flores, 1987).

In words that I more or less lifted from Salomon (1997), but then heavily edited:  $\ll$ The proponents of distributed cognition don't dispute the computational/representational assumption, per se. Rather, they take issue with the location that traditional cognitive scientists tend to ascribe to the representations and transformations that are said to form the basis of cognition. In contrast to the theoretical stance associated with traditional cognitive science, that cognition is in the head, proponents of distributed cognition adopt the theoretical view that cognition takes place within a system that includes humans and tools. In the traditional cognitive science framework, cognition is considered to be a state of being that tends to be described in terms of a rich mix of mental constructs. In the distributed cognition framework, cognition is viewed as an emergent property of interaction among components of the system. In other words, distributed cognition adopts a perspective for investigating cognitive phenomena according to which cognition is equated with representation based interactions among the people and the artifacts that make up a system.»

The ideas associated with distributed cognition were originally proposed by Edwin Hutchins (1995). Hutchins, a cognitive anthropologist, was a keen observer of navigation. He observed people responsible for navigating both airplanes and ships. He was struck by how information was *distributed* among people and artifacts, and how no *single person* was in a position to navigate the vehicle. One of Hutchins' key insights is that the very *nature* of a problem is changed when it is considered through the lens of distributed cognition. Donald Norman, a colleague of Hutchins, adroitly made his own contributions to the theory of distributed cognition, expertly discussed the framework in popular texts, and succeeded in pointing the way towards reconceiving the foundational elements of Human-Computer Interaction (HCI) in terms of distributed cognition. Among Norman's most significant theoretical contributions was his articulation of the nature and role of *cognitive artifacts*. A **cognitive artifact** is essentially a man-made tool that is designed to enhance cognition. "The power of a cognitive artifact comes from its function as a representational device" wrote Norman (1991). He then went on to define cognitive artifact in *representational* terms as "an artificial device designed to maintain, display, or operate upon information in order to serve a representational function."

A to do list is a cognitive artifact. Such a list effectively enhances your memory. From another perspective, however, the list merely changes the nature of what you do from referencing your memory to maintaining and checking a list. A GPS is a cognitive artifact. It effectively enhances your ability to get from here to there. On the other hand, neither your knowledge of navigation nor your sense of direction are improved by the GPS system. Clearly it changes the nature of your task, not only the rational aspect of the task which amounts to searching for a desired route through a landscape of sometimes hard to find landmarks, but also the emotional aspect of the task which tends to be transformed from somewhat stressful ordeal to rather relaxed journey! LaTex is a cognitive artifact that I used for generating this text. Merely the fact that the table of contents references pages correctly is an indicator that I can do better work with LaTex than without it.

How is any of this relevant to the processes of learning to program computers? You will be using a range of cognitive artifacts in support of your programming activities. IntelliJ, the *integrated development environment (IDE)* that is featured in this course is an excellent example of a cognitive artifact, one that powerfully reflects Norman's representational definition of the concept. It will help you to be a much more productive Java programmer than you would otherwise be. The microworlds (NPW and MMW) are domain specific cognitive artifacts that will help you to do graphics programming and sonic programming. Additionally, you will be using certain information processing tools that will *indirectly* support your programming activities by enhancing your skills with respect to learning. These information processing tools – search tools, site development tools, social media tools – are also cognitive artifacts.

Having said all of this about distributed cognition, it is important to emphasize, as Gavriel Salomon (1997) does in "No distribution without individuals' cognition: a dynamic interactional view," that it is vital to value both *individual cognition* and *distributed cognition*. To effectively engage in computer programming, you want to (1) cultivate a mix of well developed mental models, and (2) develop an ability to become one with powerful tools and systems. Furthermore, it is worth noting that while some interactions with cognitive artifacts may fail to change the way we think, by merely changing the nature of the problem at hand, as Hutchins so insightfully observed, other interactions with cognitive artifacts may significantly change the way we think. In fact, computer programming languages and computational microworlds are cognitive artifacts of this second kind, the kind that leave some sort of *cognitive residue* behind in the wake of intellectual partnership.

### The Learning Sciences

One body of knowledge that is increasingly finding traction among educators, especially those in higher education, goes by the name of *the learning sciences*. The term **learning sciences** refers to a system of principles that pertain to learning, along with investigations into the validity and utility of the principles, and explorations of interactions among the principles. There is no one universally agreed upon set of principles, but here is a list of those learning science principles that figured most prominently in the conception and construction of this text:

- **Constructionist Principle** An extension of Piaget's constructivism the theory that learning involves the building of knowledge structures within the individual mind which adds (1) the idea that this happens especially felicitously in a context in which the learner is consciously engaged in constructing a public entity (Papert, 1980), and (2) a more distributed view of instruction, one where learning and teaching are constructed in interactions between the teacher and students as they are engaging in design and discussion of learning artifacts (Kafai, 2006).
- **Deep Learning Principle** Education is best accomplished by privileging engagement over explanation, uncoverage over coverage, questioning over answering, reflection over reaction, representation over information, and process over product.
- **Project-Based Learning Principle** Deep learning accrues as a side-effect of engagement in an incremental, holistic process of artifact creation in response to the consideration of a substantial problem of interest to the learner.
- Learner-Centered Design Principle Favor bridging the "gulf of expertise" over the "gulf of execution" and the "gulf of evaluation". That is, place emphasis on scaffolding which affords opportunities to enhance understanding by bridging the conceptual distance between a novice and an expert in the domain of interest, rather than on tools or methodologies that merely ease the performance of tasks (Quintana, Shin, Norris, & Soloway, 2006).
- Imagery Principle Educators need to search for ways in which the power of imagery (e.g., effortless structural interpretation) can be used to support learning, creativity, and reasoning (Schwartz & Heiser, 2006).
- Inscription Principle Students learn by doing and by thinking about what they have done. Creating external representations of one's thoughts in some sort of inscription system for reflecting upon one's thinking and sharing one's thoughts with others is of central significance to deep learning (Pea, 1993).
- Distributed Cognition Principle Cognition is something accomplished through collaborative interactions involving people and artifacts rather than something possessed by individuals in isolation (Hutchins, 1995).

You will find unconcealed traces of these principles lurking throughout the course that this text supports, throughout its *labs* and its *presentations* and its *programming challenges*. Where do they come from? They derive from the work of some of the great thinkers about learning and education, including L. S. Vygotsky, Maria Montesori, John Dewey, Jean Piaget, and Seymour Papert. The list that I compiled is just *one possible organization* of a selection of *lasting ideas about learning* into a system of principles. The names that I determined to give the principles are merely intended to make them a bit more sticky in the mind.

### Tips on Learning to Learn

The aforementioned principles informed my crafting of this course, the *labs*, the *presentations*, and the *programming* challenges. That is, they informed my efforts at *teaching* the material, by which I mean, to riff on Einstein, setting up conditions in which my students might learn. On the other side of the teaching/learning equation, here are just a few ideas that you might like to bear in mind as you work through this text, ideas about *learning* and *thinking* that may resonate with you:

- The "3 Rs" of Learning Guy Claxton Claxton (2000) references three potential habits of mind that he believes are the mark of a good learner. The Rs stand for resourcefulnes, resilience, and reflection. Resourcefulness is the ability to deal with challenging problems or situations in inventive ways. Resilience is the ability to persist in pursuit of a goal in spite of uncertainty, confusion, obfuscation, or other difficulties. Reflection is the act of looking at a thought from a strategic point of view with an eye towards confirmation, refutation, or reformulation. (For Dewey (1933), reflective thinking has connotations of being grounded in experience, of evaluating the quality of the thought, and of vigilantly reshaping thoughts in potentially productive ways.)
- *Mindfulness* One potential habit of mind that has been getting quite a bit of press lately is *mindfulness*, as championed by Ellen Langer. A **mindful approach to thinking**, she suggests, involves three things: a search for new ways to classify knowledge, a disposition to appropriate new information, and an appreciation of multiple perspectives (Langer, 1998, p. 4). The proponents of mindfulness believe that approaching cognitive activities mindfully is a key to empowering learning.
- Metacognition The term **metacognition** refers to thinking about thinking. In metaphorically eloquent words that are a good fit for a text on learning to program, M. Martinez (2010, p. 143) asks: "How is it possible to establish higher-order thinking as a habit to build metacognition into our mental software as a background application that runs continuously?" If you are serious about learning, you will do your very best to find an answer that works for you!
- *Mindset* Carol Dweck (2007), based on decades of research, contrasts the **fixed mindset** with the **growth mindset**. By distancing yourself from the former, which tends to stifle your ability to learn by incorporating the notion that having to work hard simply betrays intellectual inadequacies, and cultivating the latter, which champions the belief that hard work is a catalyst for meaningful growth, you are more likely to become the person, thinker, programmer that you would really like to be. (Dweck makes a compelling case for the growth mindset in some of her 10 minute Youtube videos.)

### How to Use this Text

This laboratory manual will help to guide you through the acquisition of some basic knowledge of computer programming in Java using the IntelliJ integrated development environment. The labs are an *integral part* of the course. Some of them introduce new material that will be elaborated during classroom presentations. Some of them serve to clarify ideas presented during the classroom presentations. Some of them constitute the start of a programming assignment.

# $\implies$ The labs are, with the exception of just a few rather short labs, designed to be started during your formal laboratory period, and then completed on your own.

Having the hard copy text of a lab with you will substantially enhance your laboratory experience, compared with trying to read it from on line. So please be sure to bring the lab manual with you to each laboratory class. You

should be answering the occasional questions posed right in the manual. You should be making notes in the manual about questions that arise in your mind as you work through the manual. You should be keeping your place in the sequences and subsequences of tasks that define the laboratory activities by making marks in appropriate ways on the pages of the manual. In short, you should be making your lab manual "your own"!

Not only do the labs tend to extend beyond the temporal scope of a lab period, but they extend beyond the temporal scope of the semester. By saying this I am referring to the fact that there is at least one more lab in the manual than will fit comfortably into the semester. Maybe two. Maybe three. This is by design. No worries. Any "extra" labs will be featured prominently in classroom discussions. Moreover, they will serve as the basis of my first answer to the question frequently asked by students at the end of the course: "What might I do in order to prepare for the subsequent CS2 course?"

### Technical Content Represented in this Text

Lest all of these preliminaries obscure the fact that this text really does support a CS1 course, the following partial list of technical terms and phrases that you will find in this text, presented in no particular order, is intended to serve as a reminder!

$$\label{eq:second} \begin{split} & \begin{subarray}{ll} \label{eq:second} \end{subarray} \en$$

# 2 Lab 1: Hello World! Hello You!

### William James on WISDOM

The art of being wise is the art of knowing what to overlook.

### Overview

In this lab you will establish and run two very simple programs. The first is loosely referred to as the "hello world" program. This opener is purely text-based. The second is a variant of the first which features a widget - a computational component with a graphical representation.

### Why do it?

As you work through this lab you will:

- 1. Get acquainted with IntelliJ, the standard computer programming environment for this particular course.
- 2. Establish and run simple Java programs in IntelliJ.
- 3. Hone your skills with respect to methodically executing sequences of tasks.

### Task 1: Prepare to do some Java programming in IntelliJ

- 1. Log on to a sanctioned machine.
- 2. Get into IntelliJ ...
  - (a) Search for IntelliJ on your machine, unless you can spot it just lying around somewhere.
  - (b) Launch it!
- 3. Establish a new project ...
  - (a) When the Welcome to IntelliJ IDEA window appears, choose Create New Project.
  - (b) On the New Project form that appears on the screen ...
    - i. Be sure that Java is selected on the left side of the screen and click Next.
    - ii. Choose not to use a template for our project and click Next again.
    - iii. Type CS1 into the Project Name field.
    - iv. Click Finish.

### Task 2: Establish and run the traditional starter program

```
1. Create a package ...
```

- (a) On the left side of the screen, click the arrow next to CS1 to expand it.
- (b) Right click on the **src** folder and create a new **Package**. A package is a way to keep parts of your Java program organized.
- (c) In the window that appears, call your package greetings and click OK.

```
2. Create a source program ...
```

- (a) Expand the the src folder if necessary.
- (b) Right click on the greetings package and create a new Java Class.
- (c) On the Create New Class form that appears ...
  - i. Type HelloWorld into the Name field.
    - ii. Select Class from the Kind field.
  - iii. Click OK.
- (d) Modify the source program template so that it matches the following:

```
/*
 * Traditional starter program.
 */
package greetings;
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello world!");
   }
}
```

3. Run the program ...

- (a) Select Run 'HelloWorld.main()' from the menu that appears when you *right click* somewhere in the area that is displaying the source code.
- (b) Observe that Hello World! appears among the information the Output window.

### Task 3: Establish and run the nontraditional variant of the starter program

- 1. Create a source program ...
  - (a) Right click on the greetings package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type HelloYou into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.

(c) Modify the source program template so that it matches the following:

```
/*
 * Variant of the traditional starter program that features a widget.
 */
package greetings;
import javax.swing.JOptionPane;
public class HelloYou {
    public static void main(String[] args) {
        String name = JOptionPane.showInputDialog(null, "Who are you?");
        System.out.println("Hello, " + name + "!");
    }
}
```

- 2. Run the program ...
  - (a) Select Run 'HelloYou.main()' from the menu that appears when you right click somewhere in the area that is displaying the source code.
  - (b) Enter your name into the Input dialog box that appears.
  - (c) Observe that the appropriate text appears among the information the Output window.

### Task 4: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

# 3 Presentation: Microworlds, Distributed Cognition, Learning to Learn

This material is presented in real time during course lectures.

# 4 Lab 2: Hello Painter! Hello Composer!

### Carl Jung on THE CREATIVE MIND

The creation of something new is not accomplished by the intellect but by the play instinct acting from inner necessity. The creative mind plays with the objects it loves.

### Something to think about

Learning can be viewed as a two stage process in which you (1) gain some experience, and (2) endeavor to make sense of the experience. As you engage in this lab, which affords you an opportunity to get acquainted with the *Non-representational Painting World* and the *Modular Melody World*, please do your best to appreciate this perspective on learning.

### Overview

One premise of this course is that it tends to be more fun to take, and more fun to teach, if some interesting computational objects, housed in computational learning environments, are incorporated into the course. With this premise in mind, a graphical microworld and a musical microworld are introduced in this lab.

The Nonrepresentational Painting World (NPW) contains functionality for creating and manipulating a variety of shapes. It also contains functionality for creating painters that can render (draw/paint) the shapes on a virtual canvas. In this lab you are asked to create two images in the context of the NPW.

The Modular Melody World (MMW) affords access to simple note objects, and to composer objects that make use of a dedicated note object to assist you in laying down coherent sequences of musical notes. The note objects, whether stand alone or composer controlled, can be rendered sonically, visually, or chromesthetically. In this lab you will be asked to create a melodic fragment with a stand alone note object, to establish a *listening* program that will enable you to get acquainted with the most basic modular melodic sequences of MMW, and to create a melodic sequence by enlisting the aid of a composer object.

Note: This lab is sure to run a bit long! It is not expected that you will finish it during the lab hour. Please finish this on your own prior to your next week's lab hour. You might think of the task of completing this lab on your own as a preliminary programming assignment, since the programming assignments are activities that your are expected to do on your own.

## Why do it?

As you work through this lab you will:

- 1. Install some .jar files in your CS1 project library, files that contain programs that you will run.
- 2. Create and use computational objects in Java.
- 3. Engage in some creative computational activities.
- 4. Learn how to write a program by starting with an existing program that does something similar to what you want your new program to do.
- 5. Gain familiarity with the Nonrepresentational Painting World.
- 6. Gain familiarity with the Modular Melody World.

# Task 1: Install the software that implements the computational microworlds (NPW and MMW) in the library of your CS1 project

- 1. Get a browser going.
- 2. Find your way to the following page: www.cs.oswego.edu/~blue/software/
- 3. Download three files. These files contain software that you will use to create images and melodic sequences.
  - (a) Download the simple painter code (the SimplePainter.jar file) from the Java Microworld APIs area of the page. Important: Please perform the download in the following way:
    - i. *Right click* on the link.
    - ii. Select the Save Link As ... option.
    - iii. Save the file to your *home* directory, or to a special directory that you create for the purpose of storing these three files.
  - (b) Download the simple composer code (the SimpleComposer.jar file) from the Java Microworld APIs area of the page. Important: Please perform the download in the manner prescribed for the simple painter file.
  - (c) Download *JFugue* (the jfugue-4.0.3-with-musicxml.jar file) from the *Java Microworld APIs* area of the page. **Important**: Please make use of the same downloading procedure.
- 4. Add the files to the *External Libraries* folder of the CS1 project:
  - (a) Return to IntelliJ.
  - (b) Select, if necessary, the CS1 project.
  - (c) From the File menu choose Project Structure...
  - (d) On the left side of the window which appears, choose Modules.
  - (e) Click the Dependencies tab.
  - (f) At the bottom of the window, near the center, click the + button. Select the JARs or directories... option. With the assistance of the widget that appears, find your way to the SimplePainter.jar file that you recently downloaded, and then get it.
  - (g) At the bottom of the window, near the center, click the + button. Select the JARs or directories... option. With the assistance of the widget that appears, find your way to the SimpleComposer.jar file that you recently downloaded, and then get it.
  - (h) At the bottom of the window, near the center, click the + button. Select the JARs or directories... option. With the assistance of the widget that appears, find your way to the jfugue-4.0.3-with-musicxml.jar file that you recently downloaded, and then get it.
  - (i) Click OK in the Project Structure window.
  - (j) If you haven't yet done so, look to see if the External Libraries folder contains the three files.

Task 2: Write and run a program to generate an image consisting of a blue dot



- 1. Create a package ...
  - (a) On the left side of the screen, click the arrow next to CS1 to expand it.
  - (b) Right click on the src folder and create a new Package.
  - (c) In the window that appears, call your package npw and click OK.
- 2. Create a source program ...
  - (a) Right click on the npw package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type BlueDot into the Name field.
      - ii. Select Class from the Kind field.
      - iii. Click OK.
  - (c) Modify the source program template so that it matches the following:

### BlueDot Program

```
/*
 * Program to paint a blue dot in the context of the Nonrepresentational
 * Painting World, NPW.
 */
package npw;
import java.awt.Color;
import javax.swing.SwingUtilities;
import painter.SPainter;
import shapes.SCircle;
public class BlueDot {
    // THE SOLUTION TO THE BLUE DOT PROBLEM
```

```
private void paintTheImage() {
    SPainter klee = new SPainter("Blue Dot",600,600);
    SCircle dot = new SCircle(200);
    klee.setColor(Color.BLUE);
    klee.paint(dot);
}
// REQUIRED INFRASTRUCTURE
public BlueDot() {
    paintTheImage();
}
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new BlueDot();
        }
    });
}
```

3. Run the program ...

}

- (a) Select Run 'BlueDot.main()' from the menu that appears when you right click somewhere in the area that is displaying the source code.
- (b) Observe the output, the blue dot in the window that appears.
- (c) Close the window containing the blue dot.

# Task 3: Write and run a program to generate the first several notes of a well-known melody using a *lone note*

- 1. Create a package...
  - (a) Right click on the src folder and create a new Package.
  - (b) In the window that appears, call your package mmw and click OK.
- 2. Create a source program...
  - (a) Right click on the mmw package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type Dorothy into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.

(c) Modify the source program template so that it matches the following:

```
/*
 * Name that tune!
 */
package mmw;
import note.SNote;
public class Dorothy {
    public static void main(String[] args) {
        SNote note = new SNote();
        note.text();
        note.x2(); note.play();
        note.rp(7); note.play();
        note.s2(); note.lp(); note.play();
        note.lp(2); note.s2(); note.play();
        note.rp(); note.play();
        note.x2(); note.rp(); note.play();
        note.rp(); note.play();
        System.out.println();
    }
}
```

Dorothy Program

- 3. Run the program ...
  - (a) Select Run 'Dorothy.main()' from the menu that appears when you right click somewhere in the area that is displaying the source code.
  - (b) Observe the output, the textual representation of the notes, at least. If you can do so, give a listen. Can you name that tune?

#### Task 4: Write and run a program to listen to the Basic modular melodic sequences

Eight of the modular melodic sequences inherently available in the MMW are classified as *Basic* sequences. This task invites you to write a program that will allow you to listen to the sequences, provided you have a way to hear the machine's sonic output.

- 1. Create a source program...
  - (a) Right click on the mmw package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type BasicsListener into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.

(c) Modify the source program template so that it matches the following:

```
{\tt BasicsListener}\ {\bf Program}
```

```
/*
 * Program to check out (view and possibly listen to) the eight melodic
 * sequences classified as "Basic" sequences in the Modular Melody World
 */
package mmw;
import composer.SComposer;
public class BasicsListener {
    public static void main(String[] args) {
        SComposer c = new SComposer();
        c.text();
        System.out.println("c.mms1 ..."); c.mms1(); space(c);
        System.out.println("c.mms2 ..."); c.mms2(); space(c);
        System.out.println("c.mms3 ..."); c.mms3(); space(c);
        System.out.println("c.mms4 ..."); c.mms4(); space(c);
        System.out.println("c.mms5 ...."); c.mms5(); space(c);
        System.out.println("c.mms6 ..."); c.mms6(); space(c);
        System.out.println("c.mms7 ..."); c.mms7(); space(c);
        System.out.println("c.mms8 ..."); c.mms8(); space(c);
        c.untext();
    }
    private static void space(SComposer c) {
        c.untext(); c.rest(2); c.text();
    }
}
```

- 2. Run the program. If you can hear the sound generated by the machine you are working on, good! If not, perhaps you can imagine it from the textual output provided.
  - (a) Select Run 'BasicsListener.main()' from the menu that appears when you right click somewhere in the area that is displaying the source code.
  - (b) Observe the output, the textual representation of the notes, for sure. The sounds, if possible.

# Task 5: Develop a program to generate a simple melodic sequence from modular melodic sequences using a *simple composer*

- 1. Create a source program...
  - (a) Right click on the mmw package and create a new Java Class.
  - (b) On the Create New Class form that appears  $\ldots$ 
    - i. Type Melody into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.
  - (c) Modify the source program template so that it matches the following:

### Melody Program

```
/*
 * A sequence of simple modular melodic sequences.
 */
package mmw;
import composer.SComposer;
public class Melody {
    public static void main(String[] args) {
        SComposer c = new SComposer();
        c.text();
        c.mms5();
        c.rp(); c.mms5(); c.lp();
        c.lp(); c.mms5(); c.rp();
        c.mms5();
        c.untext();
    }
}
```

- 2. Run the program. If you can't hear the sound on the system you are working on, no worries. The text command causes the notes to be rendered textually, so you should be able to determine whether or not your program is working correctly.
  - (a) Select Run 'Meolody.main()' from the menu that appears when you right click somewhere in the area that is displaying the source code.
  - (b) Observe the output, the textual representation of the notes, for sure. The sounds, if possible.
- 3. Add a bit to the program. Do this by means of a copy-paste-edit operation.
  - (a) Copy the following four lines of the program:

```
c.mms5();
c.rp(); c.mms5(); c.lp();
c.lp(); c.mms5(); c.rp();
c.mms5();
```

- (b) Paste them, right after the four lines that you copied, just before the c.untext() command that constitutes the last statement of the main method. You will then have two adjacent occurrences of the same sequence of four statements.
- (c) Edit the second of the two identical sequences of four statements so that it looks like this:

c.mms7(); c.rp(); c.mms8(); c.lp(); c.lp(); c.mms7(); c.rp(); c.mms8();

4. Run the program once again, and observe the longer sequence of notes.

#### Task 6: Write and run a program to generate the Target icon in the NPW



Please note that the diameter of the large red defining circle is three times that of the small red defining circle, and that the diameter of the white defining circle is twice that of the small red defining circle.

- 1. Create a source program from the extant BlueDot program, but then change it to paint the Target icon ...
  - (a) Right click on the npw package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type Target into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.
  - (c) Using the BlueDot program as a model, and using the specifications of NPW functionality as reference material (see Appendix 1), write a program to paint the Target symbol. In doing so, please proceed in the following manner:
    - i. Replace all of the text in the Target buffer with all of the text in the BlueDot buffer. (Copy and paste is your friend!)
    - ii. Edit the comment at the head of the file to reflect the fact that you will be painting the Target icon rather than a blue dot.
    - iii. Replace all occurrences of BlueDot with Target. (There are three such occurrences.) Also, replace the title of the painter's canvas appropriately.
    - iv. Change the method that actually does the painting (the paintTheImage method) so that it paints the Target icon rather than the blue dot.

2. Run the program. If it does not work, fix it, and repeat this item.

## Task 7: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

5 Presentation: Using Objects - Painters and Composers

This material is presented in real time during the course lectures.

# 6 Lab 3: Establishing a CS1 Work Site

### Samuel Johnson on THE ACTIVE MIND

When the eye or the imagination is struck with any uncommon work, the next transition of an active mind is to the means by which it was performed.

### Overview

This lab is designed to help you to *commence* the activity of building a Web site dedicated to presenting your work in this course. In operational terms, this lab engages you in processes of creating directories, downloading files, creating files, distributing files to appropriate directories, and repeatedly editing/viewing files in a browser.

Please bear in mind that the idea is merely for you to *strive* to complete the tasks specified in this lab during the lab period. It is *not* expected that you will complete all of them during the allocated time. Please complete those that you do not manage to finish during the lab time within the following week.

### Why do it?

To engage in computer programming effectively, it is essential to (1) have a sense of the environment in which you are operating, and (2) have at least a modest command of a selection of tools which are used to operate within, and occasionally tailor, that environment. As you work through this lab you will develop a sense for the environment, and a bit of knowledge pertaining to the operating system, a texted editor, and a Web browser.

You will also commence to build a Web site by:

- 1. Creating a directory structure for your work site.
- 2. Downloading files to particular locations within your newly created directory structure.
- 3. Engaging in a constructionist process of:
  - (a) Editing .html files and a .css file using the Emacs text editor.
  - (b) Viewing .html files in a browser.
  - (c) Saving .java files as .html files to particular locations within your Web site work area.
  - (d) Creating and saving *image files* to particular locations within your Web site work area.
  - (e) Creating and saving Standard IO demo files to particular locations within your Web site work area.

### **Required directory structure**

Please do your best to make the following image sticky in your mind. The image suggests the directory structure that you should use to house files associated with this course.



### Task 1: Get ready to do some work

- 1. Log on to a sanctioned machine.
- 2. Open a Terminal window.
- 3. Open the Emacs text editor. If faced with more than one, be sure to choose the GUI version.
- 4. Open IntelliJ.
- 5. Open a Web browser, like Firefox or Chrome.

### Task 2: Add some folders to your directory structure

As you work through this task, it will benefit you to be mindful of the relationship between the commands that your are issuing and the directory structure depicted.

- 1. In the *Terminal* widow, make sure that you are in your home directory by issuing two operating system commands, and observing the operating system response.
  - (a) Issue the "go home" version of the change directory command by typing cd followed by the enter key.

- (b) Issue the print working directory command by typing pwd followed by the enter key.
- 2. Create a folder for your Web work. This folder must have a particular name in order for the CS servers to, eventually, allow you to make your work available for all to see on the Web.
  - (a) Issue the make directory command to create the public\_html folder within your home directory by mindfully thinking underscore (rather than dash) and typing: mkdir public\_html (followed by the enter key)
  - (b) Issue the *list files* command by typing 1s (that is "elle" "esss") followed by the enter key. Observe that your folder was created!
- 3. Change directories so that you are working within your public\_html directory.
  - (a) Type: cd public\_html (followed by the enter key)
  - (b) Just to be sure that you are where you want to be, issue the command to print your working directory, pwd, and observe.
- 4. Create a folder within your public\_html directory for your course work site. Be careful to name it as specified.
  - (a) Issue the make directory command by typing: mkdir CS1WorkSite
  - (b) Issue the *list files* command by typing 1s (that is "elle" "esss") followed by the enter key. Observe that your folder was created!
- 5. Create a folder within your *home directory* for the storage of miscellaneous course related files. Be careful to name it as specified.
  - (a) Issue the "go home" version of the change directory command by typing cd followed by the enter key.
  - (b) Just to be sure that you are where you want to be, issue the command to print your working directory, pwd, and observe.
  - (c) Issue the make directory command to create the folder by typing: mkdir CS1Files
  - (d) Issue the *list files* command by typing 1s (that is "elle" "esss") followed by the enter key. Observe that your folder was created!
- 6. Change directories so that you are working within your CS1Files directory.
  - (a) Type: cd CS1Files
  - (b) Just to be sure that you are where you want to be, issue the command to print your working directory, pwd, and observe.
- 7. Create a folder called data within your CS1Files directory. Check to see that you were successful.
- 8. Create a folder called midi within your CS1Files directory. Check to see that you were successful.

### Task 3: Download two files that will serve as the basis of your Web work

- 1. Get into the browser and find your way to the following location: www.cs.oswego.edu/~blue/software/
- 2. From within the CS1 Web Site Resources area, download the style file for your Web work to the CS1WorkSite directory by carefully proceeding in the following manner:
  - (a) **Important**: Please perform the download in the following way:
    - i. Right click on the <u>CS1.css</u> link.
    - ii. Select the Save Link As ... option.
    - iii. Save the file to your CS1WorkSite directory.
  - (b) Check, in the *Terminal* window, to be sure that you were successful by listing the files in the CS1WorkSite directory and observing.
- 3. From within the CS1 Web Site Resources area, download the main content file for your Web work to the CS1WorkSite directory by carefully proceeding in the following manner:
  - (a) **Important**: Please perform the download in the following way:
    - i. Right click on the <u>index.html</u> link.

- ii. Select the Save Link As ... option.
- iii. Save the file to your CS1WorkSite directory.
- (b) Check, in the *Terminal* window, to be sure that you were successful by listing the files in the CS1WorkSite directory and observing.

### Task 4: Load the index.html file of the CS1WorkSite directory into your Web browser

The index.html file of the CS1WorkSite directory will serve as the main page for your work site. To view it, do one of the following two things:

- Find the index.html file icon in your CS1WorkSite directory by looking through your file icons, and then double click on the icon.
- Enter the *local address* of the index.html file of your CS1WorkSite directory into the browser. (Something like: file:///home/HOME/public\_html/CS1WorkSite/index.html should do.)

# Task 5: Edit the content file, index.html of the CS1WorkSite directory, so that your name replaces the backwards "noname" token

- 1. Activate the Emacs text editor to perform this task.
- 2. Within the text editor, load the index.html file of the CS1WorkSite directory. There are various ways to do this. My favorite is to use my hands and type CONTROL-x) then CONTROL-f) and then interact in the minibuffer.
- 3. Change the text, replacing "Emanon" with your name.
- 4. Save the file.
- 5. Reload the file in the browser and take a look.

### Task 6: Edit the style file by changing the background color of the site

- 1. Load the CS1.css file into Emacs.
- 2. Change the background color value in the CS1.css file.
- 3. Save the file.
- 4. Reload the index.html file of the CS1WorkSite directory into the browser, and observe.
- 5. If you are not pleased with the new background color that you have chosen, change it again.

### Task 7: Change the style of your site in two other respects

- 1. Edit the style file by changing the color used for main headers, level H1 and H2, to some color other than blue.
- 2. Reload the index.html file of the CS1WorkSite directory into the browser, and observe.
- 3. Edit the style file by changing the color used for links (hyperlinks) to the same color that you chose to use for your main headers.
- 4. Reload the index.html file of the CS1WorkSite directory into the browser, and observe.
- 5. If you don't like the color you chose for these entities, choose again!

### Task 8: Make a couple of simple observations

- 1. Observe (do some clicking) that the links in the "Lab 1 block" of links and in the "Lab 2 block" of links and in the "Assignment 1 block" of links do *not* work.
- 2. Observe that the links to the various external sites do work.

### Task 9: Operationalize your work site links for the Lab 1 source files

- FYI, this task involves saving the source files from Lab 1 that reside in the IntelliJ greetings package of your CS1 project to the greetings directory of your CS1WorkSite directory. IntelliJ will do some of the work for us

   remember that we haven't created the greetings directory (though we could have!). As you work through this, it is important that you get the names and places correct!
- 2. From within IntelliJ, activate your HelloWorld.java program.
- 3. Select Export to HTML... from the IntelliJ *File* menu, and then interact with the dialog box that pops up to save HelloWorld.java to your CS1WorkSite directory. IntelliJ will create a file, called HelloWorld.java.html, inside the appropriate package directory (CS1WorkSite/greetings, in this case).
- 4. From within IntelliJ, activate your HelloYou.java program.
- 5. Select Export to HTML... from the IntelliJ *File* menu, and then interact with the dialog box that pops up to save HelloYou.java as HelloYou.html to your CS1WorkSite directory. IntelliJ will create a file, called HelloWorld.java.html, inside the CS1WorkSite/greetings directory.
- 6. Back in your terminal, visit your CS1WorkSite folder and issue the *list files* command by typing ls (that is "elle" "esss") followed by the enter key. Observe that the greetings folder has been created! Navigate inside it to see that two files are there.
- 7. Reload the web page and check to make sure that both of the links do, indeed, work.

### Task 10: Operationalize your work site links for the Lab 2 source files

- 1. FYI, this task involves saving the source files from Lab 2 that reside in the IntelliJ npw and mmw packages of your CS1 project to the npw and mmw directories that are embedded within your CS1WorkSite directory. Remember that IntelliJ will do some of the work for us regarding the creation of, and placement of files in, the npw and mmw directories. It is important that you get the names and places correct!
- 2. By analogy with work that you did in the previous task, perform the following save operaton as you did previously for the following files:
  - BlueDot.java
  - Dorothy.java
  - BasicsListener.java
  - Melody.java
  - Target.java
- 3. Reload the web page and check to make sure that all of the links do, indeed, work.

### Task 11: Operationalize your work site links for the Lab 2 image files

- 1. Save the picture of the blue dot as an image file.
  - (a) Expand IntelliJ so that you can work with it again.
  - (b) Run the blue dot painting program.
  - (c) Make a screen shot of just the window containing the image of the blue dot (the painter's canvas) by doing the following:
    - i. Run the Screenshot program, perhaps by way of the Applications icon.
    - ii. In the window that appears, select the Grab the current window option.
    - iii. Set the delay to something like 5 seconds.
    - iv. Click to take the screen shot.
    - v. Within a couple of seconds click on the window of interest.
    - vi. After the snapshot has been taken, appropriately interact with the system so that the resulting image file (of type .png) is saved as BlueDot.png into the npw directory that is embedded within your CS1WorkSite directory (The other option of the dialog box is useful for navigating to the desired directory.)
- 2. Check to see that you were successful.
- 3. Save the picture of the target as an image file. Specifically, save the picture as Target.png within the npw directory that is embedded within your CS1WorkSite directory. Work by analogy with what you were asked to do for the blue dot.
- 4. Check to see that you were successful.

### Task 12: Operationalize your work site links for the Lab 1 standard demo files

- 1. Focus on getting a demo page up for the "hello world" program. To do this:
  - (a) Find your way the course page at:www.cs.oswego.edu/~blue/software/
  - (b) Simply left click on CS1StandardDemoTemplate.html link from within the CS1 Web Site Resources area.
  - (c) Find a way to view the page source. (In the Firefox browser, for example, you might go to the menu bar, open the *Tools* menu, select the Web Developer option, and select Page Source option from within it.)
  - (d) Select the text on the page (all of it), and copy it. You will paste it somewhere else soon enough.
  - (e) Find your way to the Emacs editor. Create a new file called HelloWorldDemo.html within the greetings folder of your CS1WorkSite directory. I like to do this sort of thing with my hands, typing CONTROL-x then CONTROL-f and then interacting with the minibuffer.
  - (f) Paste the text that you previously copied into the empty buffer.
  - (g) Change the two place holder items, the question mark and the pound sign, appropriately. The former with HelloWorld.java and the latter with code copied from the standard output stream in IntelliJ after you run the HelloWorld.java program.
  - (h) Save the file.
  - (i) Reload the index.html file from your CS1WorkSite directory and see if the link to this demo is working properly. If not, carry on trying to get it to work properly!
- 2. Working by analogy with what you just did for the "hello world" program, establish a demo page called HelloYouDemo.html within the greetings folder of your CS1WorkSite directory for the "hello you" program.

### Task 13: Operationalize your work site links for the Lab 2 standard demo files

- 1. Focus on getting a demo page up for the Dorothy program. To do this:
  - (a) Find your way the course page at: www.cs.oswego.edu/~blue/software/
  - (b) Simply left click on CS1StandardDemoTemplate.html link from within the CS1 Web Site Resources area.
  - (c) Find a way to view the page source.
  - (d) Select the text on the page (all of it), and copy it. You will paste it somewhere else soon enough.
  - (e) Find your way to the Emacs editor. Create a new file called DorothyDemo.html within the mmw folder of your CS1WorkSite directory. I like to do this sort of thing with my hands, typing CONTROL-x) then CONTROL-f) and then interacting with the minibuffer.
  - (f) Paste the text that you previously copied into the empty buffer.
  - (g) Change the two place holder items, the question mark and the pound sign, appropriately. The former with Dorothy.java and the latter with code copied from the standard output stream in IntelliJ after you run the Dorothy.java program.
  - (h) Save the file.
  - (i) Reload index.html file of your CS1WorkSite directory and see that the link to this demo is working properly. If not, carry on trying to get it to work properly!
- 2. Working by analogy with what you just did for the Dorothy program, establish a demo page within the mmw folder of your CS1WorkSite directory called BasicsListenerDemo.html for the BasicsListener program.
- 3. Again working by analogy with what you just did for the Dorothy program, establish within the mmw folder of your CS1WorkSite directory a demo page called MelodyDemo.html for the Melody program.

### Task 14: Incorporate Programming Assignment 1 artifacts into your site

Please incorporate all of the relevant Programming Assignment 1 artifacts into your site. (This task will be a good test of what you should have learned in the previous parts of this lab!) You will need to incorporate source programs, images (which are demos for graphics producing programs), and Standard IO demos into your site as you work through this task, all things that you were taught to do thus far in this lab.

### Task 15: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?
7 Presentation: Command Line OS, Text Editing, The Web

This material is presented in real time during the course lectures.

## 8 Lab 4: Expressions and Shapes World Problem Solving

## E. W. Dijkstra on ABSTRACTION

Being abstract is something profoundly different from being vague. The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.

## Overview

For this lab you will be asked to write two programs, each a little bit at a time. The first program involves *arithmetic* expressions. The second program involves *shapes* from the NPW. You will also be asked to post these programs, as well as output generated by these programs, to your work site.

## **Blue Diamond**

The blue diamond depicted here was the inspiration for the ShapesThing program that is featured in the second half of this lab. Keeping it in mind may make that part of the lab a bit more meaningful to you. Moreover, the loose geometric connotations that it conjures in the mind should prime you to better think thoughts with respect to elements of the ExpressionsThing program that is featured in the first part of the lab.



## Why do it?

As you work through this lab you will:

- 1. Introduce variables and bind them to values.
- 2. Perform computations involving fully parenthesized arithmetic expressions.
- 3. Create shapes from the NPW, both *directly* with constructors and *indirectly* via *inscribing/circumscribing* functionality.
- 4. Perform computations involving shapes in the NPW.

## Some things you will need to know ...

For this lab you need to know three specific things: (1) what it means for an expression to be *fully parenthesized*, (2) how to play a game called *Crypto*, and (3) a couple of specific concepts pertaining to shapes and the corresponding bits of shape generating *functionality* found in the NPW.

- 1. A fully parenthesized expression is an expression for which there is exactly one set of parentheses corresponding to each operator. For example: (5 + 3), ((9 4) \* 3), 5, and (((3 + 3) + 3) + 3) are all fully parenthesized arithmetic expressions. These are not: ((5 + 5)), (3 \* 3 \* 3), and (111).
- 2. The **Crypto Problem** is just this: Given N source numbers and one goal number, all integers within a prescribed range, construct a *fully parenthesized* arithmetic expression that evaluates to the goal number which uses all of the source numbers and zero or more occurrences of each of the four basic arithmetic operators. For example: Make 5 (the goal) from 8 4 5 2 (the sources). Here is one possible solution: (5 + (8 (4 \* 2)))).
- 3. The **inscribing circle** of a given square is the circle that intersects the square at the midpoint of each side of the square. The **inscribing square** of a given circle is the square that intersects the circle at each of its four corners. The **circumscribing circle** of a given square is the circle that intersects the square at each of its four corners. The **circumscribing square** of a given circle is the square that intersects the circle at the midpoint of each of its four sides. You may find it helpful to look over the specifications for the *simple shapes* functionality of the NPW (see Appendix 1) that pertain to these definitions.

## Task 1: Get ready to do some work

- 1. Log on to a sanctioned machine.
- 2. Open IntelliJ.
- 3. Open the CS1 project, if need be.

## Task 2: Start creating an ExpressionsThing program

- 1. Create a package...
  - (a) Right click on the src folder and create a new Package.
  - (b) In the window that appears, call your package expressions and click OK.
- 2. Create a source program...
  - (a) Right click on the expressions package and create a new Java Class.
  - (b) On the Create New Class form that appears ...

- i. Type ExpressionsThing into the Name field.
- ii. Select Class from the Kind field.
- iii. Click OK.
- 3. Write a multiline comment at the start of the file containing something reasonable; something to suggest the fact that this program affords opportunities to explore the construction of arithmetic expressions in the context of some very simple problem solving.

#### Task 3: Three related expressions

- 1. Create an empty main method. Look to previous lab source files for what the main method declaration should look like.
- 2. To the main method, add the following code which features three expressions "intended" to compute the *perimeter* of a circle of radius 5 (using 3.14 as an approximation to PI).

```
double one = 3.14 * 5 + 5;
System.out.println("one = " + one);
double two = 3.14 * ( 5 + 5 );
System.out.println("two = " + two);
double three = ( 3.14 * ( 5 + 5 ) );
System.out.println("three = " + three);
```

- 3. Run your program. Which expression is incorrect, in that it doesn't produce the expected result? (Write it down.)
- 4. Which expression is fully parenthesized? (Write it down)

#### Task 4: Translating fully parenthesized arithmetic expressions from English

- 1. FYI, items in this task will ask you to add pairs of statements to the ExpressionsThing program. The first statement of each pair is supposed to introduce a variable and bind it to a value expressed as a straightforward translation of an English phrase representing a numeric computation. The second statement of the pair is merely supposed to display the value of the expression, reasonably labelled.
- 2. On one line, introduce a variable called **four** of type **int**, and bind it to a fully parenthesized expression that computes the value of "two times three".
- 3. On the next line, print the value, labeled. That is, type: System.out.println("four = " + four);
- 4. Run your program.
- 5. On one line, introduce a variable called **five** of type **double**, and bind it to a fully parenthesized expression that computes the value of "one-half of fifty-five".
- 6. On the next line, print the value, labeled. That is, type: System.out.println("five = " + five);
- 7. Run your program. Please be sure to check your answer! Precisely, what is one-half of fifty-five?
- 8. On one line, introduce a variable called **six** of type **double**, and bind it to a fully parenthesized expression that computes the value of "one-third of sixty-five".
- 9. On the next line, print the value, labeled. That is, type: System.out.println("six = " + six);
- 10. Run your program.
- 11. On one line, making good use of previously bound variables, introduce a variable called **seven** of type **double**, and bind it to a fully parenthesized expression that computes the value of "one-half of fifty-five plus one-third of sixty-five".

- 12. On the next line, print the value, labeled. That is, type: System.out.println("seven = " + seven);
- 13. Run your program.

## Task 5: Computations based on simple geometric/algebraic conceptions

- 1. FYI, items in this task will ask you to add more pairs of statements to the ExpressionsThing program. The first statement of each pair is supposed to introduce a variable and bind it to a solution to simple geometric or algebraic problem. The second statement of the pair is merely supposed to display the value of the expression, reasonably labelled.
- On one line, introduce a variable called eight of type double, and bind it to a fully parenthesized expression that computes the value of the area of a circle of radius 11.3, using 3.14 for PI. (For this item, please use ( PI \* ( R \* R ) ) as the model for computing the area of a circle of radius R).
- 3. On the next line, print the value, labeled. That is, type: System.out.println("eight = " + eight);
- 4. Run your program.
- 5. On one line, introduce a variable called **nine** of type double, and bind it to a fully parenthesized expression that computes the value of the area of a square of side 27.7. (For this item, please use (S \* S) as the model for computing the area of a square of radius S).
- 6. On the next line, print the value, labeled. That is, type: System.out.println("nine = " + nine);
- 7. Run your program.
- 8. On one line, making good use of previously bound variables, introduce a variable called ten of type double, and bind it to a fully parenthesized expression that computes the value of the average area of a circle of radius 11.3 and the area of a square of side 27.7.
- 9. On the next line, print the value, labeled. That is, type: System.out.println("ten = " + ten);
- 10. Run your program.
- 11. On one line, introduce a variable called **eleven** of type **double**, and bind it to a fully parenthesized expression that computes 17 percent of 243.5.
- 12. On the next line, print the value, labeled. That is, type: System.out.println("eleven = " + eleven);
- 13. Run your program.

### Task 6: Simple computations to solve Crypto problems

- 1. FYI, items in this task will ask you to add yet more pairs of statements to the ExpressionsThing program. The first statement of each pair is supposed to introduce a variable and bind it to a solution to a *Crypto* problem. The second statement of the pair is merely supposed to display the value of the expression, reasonably labelled. Please don't forget that a solution to a Crypto problem is a *fully parenthesized* arithmetic expression!
- 2. On one line, introduce a variable called twelve of type int, and bind it to a fully parenthesized expression that uses the numbers 3 and 3 as sources and that evaluates to the number 1 as goal.
- 3. On the next line, print the value, labeled. That is, type: System.out.println("twelve = " + twelve);
- 4. Run your program.
- 5. On one line, introduce a variable called thirteen of type int, and bind it to a fully parenthesized expression that uses the numbers 4 and 7 and 2 as sources and that evaluates to the number 1 as goal.
- 6. On the next line, print the value, labeled. That is, type: System.out.println("thirteen = " + thirteen);
- 7. Run your program.
- 8. On one line, introduce a variable called **fourteen** of type **int**, and bind it to a fully parenthesized expression that uses the numbers 1 and 3 and 7 and 9 as *sources* and that evaluates to the number 4 as *goal*.
- 9. On the next line, print the value, labeled. That is, type: System.out.println("fourteen = " + fourteen);
- 10. Run your program.
- 11. On one line, introduce a variable called fifteen of type int, and bind it to a fully parenthesized expression that uses the numbers 2 and 2 and 4 and 6 and 8 as sources and that evaluates to the number 5 as goal.

12. On the next line, print the value, labeled. That is, type: System.out.println("fifteen = " + fifteen);

13. Run your program.

#### Task 7: Start creating a ShapesThing program

- 1. Create a package...
  - (a) Right click on the src folder and create a new Package.
  - (b) In the window that appears, call your package shapes and click OK.
- 2. Create a source program ...
  - (a) Right click on the shapes package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type ShapesThing into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.
- 3. Write a multiline comment at the start of the file containing something reasonable, something to suggest the fact that this program affords opportunities to explore the computational solution to simple geometrical problems by means of the construction and use of basic shapes.
- 4. Create an empty main method. Look to previous lab source files for what the main method declaration should look like.

## Task 8: Computations on a square

- 1. Add a line of code that introduces a variable called square of type SSquare, and bind it to a new square of side 400.
- 2. Add the following line of code to your program, and then run your program: System.out.println("square = " + square.toString());
- 3. Add the following line of code to your program, and then run your program: System.out.println("area of square = " + square.area());
- 4. Using the previous item as a model, add a line of code to compute and print the *perimeter* of the square. Then run your program. (No use of the four basic arithmetic operators allowed! Where would you look in this lab manual which might help?)
- 5. Using the previous two items as models, add a line of code to compute and print the *diagonal* of the square. Then run your program. (No use of the four basic arithmetic operators allowed! Where would you look in this lab manual which might help?)

#### Task 9: Computations on a circle

- 1. Add the following code to establish a circle called **disk** of type **SCircle** and bind it to the *inscribing circle* of the variable to which **square** is bound.
- SCircle disk = square.inscribingCircle();
- 2. Add the following line of code to your program, and then run your program: System.out.println("disk = " + disk.toString());
- 3. Working by analogy with the code to compute and display the area of the square, compute and display the area of the disk. Run the program.
- 4. Working by analogy with the code to compute and display the perimeter of the square, compute and display the perimeter of the disk. Run the program.

## Task 10: Computations on a another square

- 1. Add a line of code to establish a square called diamond of type SSquare and bind it to the *inscribing square* of the disk
- 2. Add the following line of code to your program, and then run your program: System.out.println("diamond = " + diamond.toString());
- 3. Write code to compute and display the area of the diamond. Run the program.

### Task 11: Post your code and associated demos to your work site

Create a space on your work site to represent the work that you have done for this lab. To do this, please open the index.html file of your CS1WorkSite directory in the Emacs text editor, and copy the clump of code associated with Lab 1 to a point just after the clump of code associated with Lab 3. Then appropriately edit this code. (Working in this fashion, you are more likely to faithfully adhere to the proper format for your CS1 work site than if you just type from scratch.) Broken down a bit, here is what you will want to do:

- 1. Edit the index.html file (in the manner suggested above) so that it properly represents Lab 4. You will want to be sure to arrange for this clump of code (the clump associated with Lab 4) to reference the four relevant files for this lab: appropriate versions of the source and demo for ExpressionsThing and for ShapesThing.
- 2. Use IntelliJ to generate the .html version of the ExpressionsThing.java. If you've done it correctly, it will place the file inside a folder called expressions inside your CS1WorkSite folder. Check to make sure the link works on your work site!
- 3. Use IntelliJ to generate the .html version of the ShapesThing.java. If you've done it correctly, it will place the file inside a folder called shapes inside your CS1WorkSite folder. Check to make sure the link works on your work site!
- 4. Carefully, working in the manner prescribed in your previous lab, create a Standard IO demo file for the ExpressionsThing program, and place it in the expressions folder of your work site work area. Check to make sure the link works on your work site!
- 5. Carefully, working in the manner prescribed in your previous lab, create a Standard IO demo file for the ShapesThing program, and place it in the shapes folder of your work site work area. Check to make sure the link works on your work site!

## Task 12: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it? 9 Presentation: Basic Data Types, Problem Solving Strategies

This material is presented in real time during the course lectures.

## 10 Lab 5: An Interpreter Featuring Loop Forever and Selection

## Albert Einstein on TEACHING

Example isn't another way to teach. It is the only way to teach.

## Overview

For this lab you will be asked to write a sequence of three *interpreters*. An **interpreter** is a program that *recognizes* and *responds to* commands. The interpreters will display dots of various colors in a window. Input will be obtained through one kind of *dialog box*. Output associated with a HELP command, as well as *error messages*, will be displayed through another kind of *dialog box*.

### Why do it?

In this lab you will gain experience in doing the following things:

- 1. Writing an interpreter, complete with a HELP command and an error reporting mechanism.
- 2. Iterating by means of a loop forever construct.
- 3. Making use of a break statement to escape from a loop forever construct.
- 4. Coding a multiway conditional construct.
- 5. Working from a previous program to create a new program.
- 6. Creating random colors!

## Task 1: Get ready to do some work

- 1. Log on to a sanctioned machine.
- 2. Open IntelliJ.
- 3. Open the CS1 project, if need be.

#### Task 2: Create an Interpreter1 program

- 1. Create a package...
  - (a) Right click on the src folder and create a new Package.
  - (b) In the window that appears, call your package interpreters and click OK.
- 2. Create a source program...
  - (a) Right click on the interpreters package and create a new Java Class.
  - (b) On the Create New Class form that appears ...

- i. Type Interpreter1 into the Name field.
- ii. Select Class from the Kind field.
- iii. Click OK.
- 3. Edit the template so that it looks like the following:

Interpreter1 Program

```
/*
 * This interpreter is intended to paint different colored dots in a window.
* The commands that the interpreter can recognize and respond to are:
 * - BLUE: paint a blue dot
 * - RED: paint a red dot
 * - HELP: show a list of the commands in a dialog message box
 * - EXIT: terminate the program
 */
package interpreters;
import java.awt.Color;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import painter.SPainter;
import shapes.SCircle;
public class Interpreter1 {
   private void interpreter() {
        // CREATE OBJECTS TO THINK WITH
        SPainter miro = new SPainter("Dot Thing",400,400);
        miro.setScreenLocation(0,0);
        SCircle dot = new SCircle(180);
        // REPEATEDLY TAKE A COMMAND FROM AN INPUT DIALOG BOX AND INTERPRET IT
        while ( true ) {
            String command = JOptionPane.showInputDialog(null,"Command?");
            if ( command == null ) { command = "exit"; } // user clicked on Cancel
            if ( command.equalsIgnoreCase("blue") ) {
                miro.setColor(Color.BLUE);
                miro.paint(dot);
            } else if ( command.equalsIgnoreCase("red") ) {
                miro.setColor(Color.RED);
                miro.paint(dot);
            } else if ( command.equalsIgnoreCase("help") ) {
                JOptionPane.showMessageDialog(null, "Valid commands are: "
                        + "RED | BLUE | HELP | EXIT ");
            } else if ( command.equalsIgnoreCase("exit") ) {
                miro.end();
                System.out.println("Thank you for viewing the dots ...");
                break;
            } else {
                JOptionPane.showMessageDialog(null, "Unrecognizable command: "
```

```
+ command.toUpperCase());
    }
}
// INFRASTRUCTURE FOR SOME SIMPLE PAINTING
public Interpreter1() {
    interpreter();
}
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new Interpreter1();
        }
    });
}
```

- 4. Read the program, doing your best to understand what it does and how it does what it does.
- 5. Run the program, providing it with the following commands (one at a time): RED BLUE GREEN RED BLUE HELP RED BLUE EXIT

## Task 3: Create an Interpreter2 program

1. Create a source program ...

}

- (a) Right click on the interpreters package and create a new Java Class.
- (b) On the Create New Class form that appears ...
  - i. Type Interpreter2 into the Name field.
  - ii. Select Class from the Kind field.
  - iii. Click OK.
- 2. Completely replace the text of this Interpreter2 program with the text of the Interpreter1 program.
- 3. Appropriately change the token Interpreter1 to the token Interpreter2 throughout the program. (Note that there are three instances to be changed!)
- 4. Run the Interpreter2 program, just to make sure that it is working like the Interpreter1 program.
- 5. Now, modify the program so that it can interpret two more commands: GREEN and YELLOW. In doing so, be sure to:
  - (a) Edit the opening comment of the program appropriately
  - (b) Add two cases to the multiway conditional statement
  - (c) Add the two commands to the HELP mechanism.
- 6. Run the program, being sure to try out the new commands. Give the revised HELP command a look, as well.

#### Task 4: Create an Interpreter3 program

- 1. Create a source program ...
  - (a) Right click on the interpreters package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type Interpreter3 into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.
- 2. Completely replace the text of this Interpreter3 program with the text of the Interpreter2 program. by doing the following:
- 3. Appropriately change the token Interpreter2 to the token Interpreter3 throughout the program. (Note that there are three instances to be changed!)
- 4. Run the Interpreter3 program, just to make sure that it is working like the Interpreter2 program.
- 5. Now, modify the program so that it can interpret one more command: RANDOM. In doing so:
  - (a) Edit the opening comment of the program appropriately
  - (b) Implement the new command by doing the following:
    - i. Add the following case to the multiway conditional statement:

```
} else if ( command.equalsIgnoreCase("random") ) {
    miro.setColor(randomColor());
    miro.paint(dot);
```

- ii. Notice that randomColor() is red. IntelliJ likes to use red text and red underlines to indicate problems. Click on the red text.
- iii. Use the light bulb which appears (after about a second) to create a method stub for the randomColor() method. A stub is simply a method with essentially no body.
- iv. Edit the randomColor() method so that it appears as follows:

```
private static Color randomColor() {
    int rv = (int)(Math.random()*256);
    int gv = (int)(Math.random()*256);
    int bv = (int)(Math.random()*256);
    return new Color(rv,gv,bv);
}
```

(c) Add the new command to the HELP mechanism.

6. Run the program, being sure to try out the new command. Give the revised HELP command a look, as well.

## Task 5: Post your code and selected demos to your work site

Create a space on your work site to represent the work that you have done for this lab. To do this, please open the index.html file of your CS1WorkSite directory in the Emacs text editor, and copy the clump of code associated with Lab 4 to a point just after the Lab 4 clump of code. Then appropriately edit this code. (Working in this fashion, you are more likely to faithfully adhere to the proper format for your CS1 work site than if you just type from scratch.) Broken down a bit, here is what you will want to do:

- 1. Edit the index.html file (in the manner suggested above) so that it properly represents Lab 5. You will want to be sure to arrange for this clump of code (the clump associated with Lab 5) to reference at least seven files: the .html version of the Interpreter1.java program, the .html version of the Interpreter2.java program, and the .html version of the Interpreter3.java program, and at least four separate snapshots which, minimally, capture a colored dot in a canvas, an input dialog box, an error message dialog box, and a help menu dialog box. Perhaps it would be most interesting to generate these with the third version of the program. Some context (regular text) for the links (hypertext) might be a very good idea!
- 2. Place the .html version of the Interpreter1.java program in the interpreters folder. Check to make sure the link works on your work site!
- 3. Place the .html version of the Interpreter2.java program in the interpreters folder. Check to make sure the link works on your work site!
- 4. Place the .html version of the Interpreter3.java program in the interpreters folder. Check to make sure the link works on your work site!
- 5. Carefully, working by analogy with the posting of the blue dot (refer back to Lab 3 if you need to), add to your work site a randomly colored dot canvas image that was generated by your Interpreter3 program.
- 6. Add to your site a reference to an image of an input dialog box.
- 7. Add to your site a reference to an image of a message dialog box that displays a HELP message.
- 8. Add to your site a reference to an image of a message dialog box that displays an error message.
- 9. As always, check to make sure the links are working as desired.

#### Task 6: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

11 Presentation: Selection (if) and Repetition (while)

This material is presented in real time during the course lectures.

## 12 Lab 6: Functions and Commands

## Abelson and Sussman on PROCEDURAL PROGRAMMING

The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence of what might best be called procedural epistemology - the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects.

## Simon Peyton Jones on FUNCTIONAL PROGRAMMING

When the limestone of imperative programming has worn away, the granite of functional programming will be revealed underneath!

## Pictorial preview of things/thinks to come



## Overview

For this lab you will be asked to write one program that features *functions* and one program that features *commands*. A **function** is a method that is characterized by the return of a value. Any actions performed during execution of the method are performed in the service of computing the value to be returned. A **command** is a method that is characterized by performing some action. Any values that are computed during execution of the method are computed in the service of performing the action. You will be asked to mindfully engage in the process of *stepwise refinement* as you prepare the first two programs for execution. Finally, you will be asked to perform a simple program alteration.

## Why do it?

In this lab you will gain experience in doing the following things:

- 1. Defining and using *functions*.
- 2. Defining and using commands.
- 3. Program construction by means of stepwise refinement.
- 4. Program modification.
- 5. Iteration using the while statement.
- 6. Conditional execution using the if statement.

## Task 1: Get ready to do some work

- 1. Log on to a sanctioned machine.
- 2. Open IntelliJ.
- 3. Open the CS1 project, if need be.

## Task 2: Create a SurfaceAreaOfCube program

- 1. Create a package...
  - (a) Right click on the src folder and create a new Package.
  - (b) In the window that appears, call your package mathematics and click OK.
- 2. Create a source program...
  - (a) Right click on the mathematics package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type SurfaceAreaOfCube into the Name field.
    - ii. Select Class from the Kind field.
    - iii. Click OK.
- 3. Edit the template so that it looks like the accompanying program. Do this in the following manner:
  - (a) Write the multiline comment at the start of the file appropriately.
  - (b) Create an empty main method, then edit it appropriately.
  - (c) Making good use of the *red text* and *light bulbs*, let IntelliJ help you to create both *stubs*: the one for the edgeLength method and the one for the surfaceArea method.
  - (d) Edit the edgeLength method. Notice that Scanner is colored red. Click on it and do as IntelliJ suggests to import the Scanner class.
  - (e) Edit the surfaceArea method. Let IntelliJ import the SSquare class as you do.

#### Program: SurfaceAreaOfCube

```
/*
 * Program that features two functions to compute the surface area of a cube.
 * - The edge length will be read from the standard input stream.
 * - The surface area will be printed to the standard output stream.
 * - A face of the cube will be modeled as a simple square.
 */
package mathematics;
import java.util.Scanner;
import shapes.SSquare;
public class SurfaceAreaOfCube {
    public static void main(String[] args) {
        double edgeLength = edgeLength();
        double surfaceArea = surfaceArea(edgeLength);
        System.out.println("surface area = " + surfaceArea);
    }
    private static double edgeLength() {
        System.out.print("Please enter the edge length of the cube: ");
        Scanner scanner = new Scanner(System.in);
        double edgeLength = scanner.nextDouble();
        return edgeLength;
    }
    private static double surfaceArea(double edgeLength) {
        SSquare face = new SSquare(edgeLength);
        int nrOfFaces = 6;
        double surfaceArea = face.area() * nrOfFaces;
        return surfaceArea;
    }
}
```

- 4. Run the program to compute and print the surface area of a cube of edge length 7.5 units.
- 5. Run the program to compute and print the surface area of a cube of edge length 12.95 units.

## Task 3: Thinking on stepwise refinement

To be productive in what you do, you generally want to become one with your tools. That is, you want to learn to make use of your tools in a natural, effective, efficient manner. One way that you can make good use of *IntelliJ* is to, more often than not, write your programs with the *principle of stepwise refinement* in mind. Just think on these things before continuing on with the next task in this lab:

- 1. According to the **principle of stepwise refinement**, you craft a solution to a problem at a level of abstraction which affords *naturalness of expression* with respect to the problem domain and the application of *powerful cognitive operators*. If the solution incorporates *abstractions*, which is normally the case, you then *refine* the abstractions by rendering them real in some sense. The term *stepwise* refers to the fact that refinement of an abstraction at one level may introduce further abstractions at the next level.
- 2. In the SurfaceAreaOfCube program, a solution to the problem was coded in the main method which introduced two abstractions, the edgeLength method and the surfaceArea method. Each of these methods was, in turn, refined by defining the methods.
- 3. Note that the process of stepwise refinement was facilitated by the mechanism in IntelliJ for automatically generating stubs for the methods by means of handy light bulbs which appear from time to time with suggestions for how you might like to proceed. Making appropriate, judicious use of these suggestions is one way that you work in a natural, efficient manner with IntelliJ.

## Task 4: Create a Balloons program

Establish a source program template ...

- 1. Right click on the npw package and create a new Java Class.
- 2. On the Create New Class form that appears ...
  - (a) Type Balloons into the Name field.
  - (b) Select Class from the Kind field.
  - (c) Click OK.
- 3. Edit the template so that it looks like the accompanying program. Be sure to work with IntelliJ in order to accomplish this task in a manner that is consistent with the principle of stepwise refinement!

#### Program: Balloons

```
/*
 * Program that paints 100 red, yellow and orange balloons in a blue sky.
 * It will feature commands.
 */
package npw;
import java.awt.Color;
import java.util.Random;
import javax.swing.SwingUtilities;
import painter.SPainter;
import shapes.SCircle;
import shapes.SSquare;
public class Balloons {
    // REQUIRED INFRASTRUCTURE
    public Balloons() {
        paintTheImage();
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
```

```
new Balloons();
        }
    });
}
// THE PAINTER DOING ITS THING
private void paintTheImage() {
    SPainter painter = new SPainter("Balloons", 600, 600);
    paintSky(painter); // ask IntelliJ to generate the stub
    int nrOfBalloons = 100;
    paintBalloons(painter,nrOfBalloons); // ask IntelliJ to generate the stub
}
private void paintSky(SPainter painter) {
    painter.setColor(Color.BLUE);
    SSquare sky = new SSquare(2000);
    painter.paint(sky);
}
private void paintBalloons(SPainter painter, int nrOfBalloons) {
    int i = 1;
    while ( i <= nrOfBalloons ) {</pre>
        paintOneBalloon(painter); // ask IntelliJ to generate the stub
        i = i + 1;
    }
}
private void paintOneBalloon(SPainter painter) {
    Random rgen = new Random();
    int rn = rgen.nextInt(3);
    if ( rn == 0 ) {
        painter.setColor(Color.RED);
    } else if ( rn == 1 ) {
        painter.setColor(Color.ORANGE);
    } else {
        painter.setColor(Color.YELLOW);
    }
    painter.move();
    int balloonRadius = 20;
    SCircle balloon = new SCircle(balloonRadius);
    painter.paint(balloon);
    painter.setColor(Color.BLACK);
    painter.draw(balloon);
}
```

```
4. Run the program.
```

}

#### Task 5: Create a AlternateBalloons program

Establish a source program template ...

- 1. Right click on the npw package and create a new Java Class.
- 2. On the Create New Class form that appears ...
  - (a) Type AlternateBalloons into the Name field.
  - (b) Select Class from the Kind field.
  - (c) Click OK.
- 3. Replace all of the code in the AlternateBalloons program with all of the code in the Balloons program.
- 4. Then, edit the AlternateBalloons program in such a way that 300 balloons, each of radius 30, of 6 different "nameless" colors, will randomly populate the sky. By nameless I don't meant that you can't imagine a name for the color, but merely that you must use a color constructor to obtain the color since there is not predefined name for it in Java. (Please don't forget to modify the leading comment.)
- 5. Run the program.

#### Task 6: Post your code and selected demos to your work site

Work by analogy with the way that you have posted artifacts to your site for previous labs. In brief, you will need to:

- 1. Edit the index.html file of your CS1WorkSite directory.
- 2. You will need to create two files for the SurfaceAreaOfCube program, one for the source program, and one for the Standard IO demo.
- 3. You will need to create two files for the Balloons program, one for the source program, and one for the image.
- 4. You will need to create two files for the AlternateBalloons program, one for the source program, and one for the image.

## Task 7: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

This material is presented in real time during the course lectures.

## 14 Lab 7: String Thing

## Kernighan and Pike on "RUBBER DUCKING"

Another effective [debugging] technique is to explain your code to someone else. This will often cause you to explain the bug to yourself. Sometimes it takes no more than a few sentences, followed by an embarrassed "Never mind, I see what's wrong. Sorry to bother you." This works remarkably well; you can even use non-programmers as listeners. One university computer center kept a teddy bear near the help desk. Students with mysterious bugs were required to explain them to the bear before they could speak to a human counselor.

## Overview

This lab features two programs. The first program, StringOps, which you will merely type in and run, provides you with an opportunity to get acquainted with some basic character string processing functionality. The second program, StringThing, provides you with an opportunity to write some character string processing code.

## Why do it?

As you work through this lab you will:

- 1. Get acquainted with some basic character string processing functionality.
- 2. Solve some simple problems in the context of character string programming.
- 3. Perform abstraction by writing methods, by introducing names (parameters) to stand for instances of data items.

## Task 1: Prepare to do the Java programming for this lab in IntelliJ

- 1. Log on to a sanctioned machine.
- 2. Get into IntelliJ.
- 3. Open the CS1 project, if need be.

### Task 2: Study (read/create/run/reflect upon) the accompanying StringOps program

- 1. Carefully read through the accompanying StringOps program.
- 2. Within a package called stringthing, establish a Java Class called StringOps. Then, enter the accompanying program, just as it is presented here.

The StringOps Program

```
/*
* Program to illustrate some basic character string processing functionality.
 */
package stringthing;
public class StringOps {
   public static void main(String[] args) {
        // ESTABLISH SOME STRINGS
        String date = "Wednesday, October 18, 1995";
        String time = "8 AM";
        String lab = "String Thing";
        // COMPUTE THE LENGTHS OF THE STRINGS
        int dateLength = date.length();
        int timeLength = time.length();
        int labLength = lab.length();
        System.out.println("\ndateLength = " + dateLength);
        System.out.println("timeLength = " + timeLength);
        System.out.println("labLength = " + labLength);
        // COMPUTE SOME POSITIONS
        int p1 = date.indexOf(",");
        int p2 = time.indexOf(" ");
        int p3 = lab.indexOf("ing");
        System.out.println("\np1 = " + p1);
        System.out.println("p2 = " + p2);
        System.out.println("p3 = " + p3);
        // COMPUTE SOME 2 ARGUMENT SUBSTRING VALUES
        System.out.println("\ndate.substring(0,9) = " + date.substring(0,9));
        System.out.println("time.substring(2,4) = " + time.substring(2,4));
        System.out.println("lab.substring(7,8) = " + lab.substring(7,8));
        // COMPUTE SOME 1 ARGUMENT SUBSTRING VALUES
        System.out.println("\ndate.substring(11) = " + date.substring(11));
        System.out.println("time.substring(2) = " + time.substring(2));
        System.out.println("lab.substring(7) = " + lab.substring(7));
        // CREATE A STRING
        String line = date + " | " + time + " | " + lab;
        System.out.println("\nline = " + line);
   }
}
```

- 3. Run the program.
- 4. Change the date in the first executable statement to today's date, and the time in the second executable statement to the current time.
- 5. Again, run the program.
- 6. Reflect upon the program and its output. As you do, take a few minutes to infer some of the string processing functionality and write down answers to the following questions.
  - (a) What does the length function do? Simply write a brief description of the length function.

(b) What does the indexOf function do? Simply write a brief description of the indexOf function.

(c) What does the substring function which takes *two* parameters do? Simply write a brief description of the substring function which takes *two* parameters.

(d) What does the substring function which takes *one* parameter do? Simply write a brief description of the substring function which takes *one* parameters.

(e) What does the "plus operator", aka the *concatenation* operator do? Simply write a brief description of this workhorse string processing operator.

## Task 3: Prepare to refine (complete) the accompanying StringThing program

- 1. Carefully read through the accompanying StringThing program.
- 2. Within your package called stringthing, establish a Java Class called StringThing. Then, enter the program just as it is presented here.

The StringThing Program

```
/*
 * This program will do a bit of character string processing.
 */
package stringthing;
public class StringThing {
    public static void main(String[] args) {
        // POINT A: CREATE A PRINT SOME STRINGS THAT REPRESENT NAMES
        // String singer = "Holiday, Billie";
        // String sculptor = "Claudel, Camille";
        // POINT B: COMPUTE AND PRINT THE LENGTHS OF THE STRINGS, WITHOUT LABELS
        // POINT C: COMPUTE AND PRINT THE LOCATION OF THE COMMA WITHIN EACH STRING, NO LABELS
        // POINT D: COMPUTE AND PRINT THE FIVE FIRST NAMES, WITH NO LABELS
        // POINT E: COMPUTE AND PRINT THE FIVE LAST NAMES, WITH NO LABELS
        // POINT F: COMPUTE AND PRINT THE FIRST NAMES, AGAIN
11
          System.out.println("\nFirst names, once again ...");
          System.out.println(firstName(singer));
11
11
          System.out.println(firstName(sculptor));
11
          System.out.println(firstName(painter));
11
          System.out.println(firstName(dancer));
11
          System.out.println(firstName(self));
        // POINT G: COMPUTE AND PRINT THE LAST NAMES, AGAIN
11
          System.out.println("\nLast names, once again ...");
          System.out.println(lastName(singer));
11
          System.out.println(lastName(sculptor));
11
11
          System.out.println(lastName(painter));
11
          System.out.println(lastName(dancer));
          System.out.println(lastName(self));
11
        // POINT H: COMPUTE AND PRINT THE FULL NAMES, NATURAL STYLE
11
          System.out.println("\nFull names, natural style ...");
          System.out.println(fullName(singer));
11
          System.out.println(fullName(sculptor));
11
          System.out.println(fullName(painter));
11
11
          System.out.println(fullName(dancer));
11
          System.out.println(fullName(self));
```

## Task 4: Point A programming

- 1. Toggle the comments on the two lines following the Point A comment.
- 2. Add a line of code which introduces a variable called painter and binds it to "Picasso, Pablo" the directory style string representation of Pablo Picasso.
- 3. Add a line of code which introduces a variable called dancer and binds it to "Zotto, Osvaldo" the directory style string representation of Osvaldo Zotto.
- 4. Add a line of code which introduces a variable called **self** and binds it to your *directory style* name.
- 5. Add the following line of code: System.out.println("\nNames ...");
- 6. Add five lines of code, one to print each of the five name strings (unlabelled), being sure to do so by referencing the name strings through the *variables* to which they are bound.
- 7. Run the program.

}

}

#### Task 5: Point B programming

For this task you will want to use the length method of the String class. If you should need a hint for how to do this, simply look at the part of the StringOps program that computes string lengths.

- 1. Introduce a variable called singerLength and bind it to the length of the name string of the singer. Be sure to arrange for the computer to compute the length. (Be sure not to do it yourself by counting!)
- 2. Introduce a variable called sculptorLength and bind it to the length of the name string of the sculptor. Be sure to arrange for the computer to compute the length. (Be sure not to do it yourself by counting!)
- 3. Introduce a variable called **painterLength** and bind it to the length of the name string of the painter. Be sure to arrange for the computer to compute the length. (Be sure not to do it yourself by counting!)
- 4. Introduce a variable called dancerLength and bind it to the length of the name string of the dancer. Be sure to arrange for the computer to compute the length. (Be sure not to do it yourself by counting!)
- 5. Introduce a variable called **selfLength** and bind it to the length of your name string. Be sure to arrange for the computer to compute the length. (Be sure not to do it yourself by counting!)
- 6. Add the following line of code: System.out.println("\nName lengths ...");
- 7. Add five lines of code, one to print the length of each of the five name string lengths (unlabelled), being sure to do so by referencing the lengths through the *variables* to which they are bound.
- 8. Run the program.

#### Task 6: Point C programming

For this task you will want to use the indexOf method of the String class. If you should need a hint for how to do this, simply look at the part of the StringOps program that computes string lengths.

- 1. Introduce a variable called singerCommaPosition and bind it to the position of the comma in the singer's name string. Be sure to arrange for the computer to compute the position of the comma.
- 2. Introduce a variable called sculptorCommaPosition and bind it to the position of the comma in the sculptor's name string. Be sure to arrange for the computer to compute the position of the comma.
- 3. Introduce a variable called painterCommaPosition and bind it to the position of the comma in the painter's name string. Be sure to arrange for the computer to compute the position of the comma.
- 4. Introduce a variable called dancerCommaPosition and bind it to the position of the comma in the dancer's name string. Be sure to arrange for the computer to compute the position of the comma.
- 5. Introduce a variable called **selfCommaPosition** and bind it to the position of the comma in your name string. Be sure to arrange for the computer to compute the position of the comma.
- 6. Add the following line of code: System.out.println("\nComma positions ...");
- 7. Add five lines of code, one to print each of the five comma positions (unlabelled), being sure to do so by referencing the comma positions through the *variables* to which they are bound.
- 8. Run the program.

## Task 7: Point D programming

For this task you will want to use the 1 argument substring method of the String class. If you should need a hint for how to do this, simply look at the part of the StringOps program that make use of the substring method with 1 argument. Also, please be sure to use the relevant variable that was bound in Task C of this program.

- 1. Introduce a variable called **singerFirst** and bind it to the first name of the singer. Be sure to arrange for the computer to compute the first name.
- 2. Introduce a variable called sculptorFirst and bind it to the first name of the sculptor. Be sure to arrange for the computer to compute the first name.
- 3. Introduce a variable called **painterFirst** and bind it to the first name of the painter. Be sure to arrange for the computer to compute the first name.
- 4. Introduce a variable called dancerFirst and bind it to the first name of the dancer. Be sure to arrange for the computer to compute the first name.
- 5. Introduce a variable called **selfFirst** and bind it to your first name. Be sure to arrange for the computer to compute the first name.
- 6. Add the following line of code: System.out.println("\nFirst names ...");
- 7. Add five lines of code, one to print each of the five last names (unlabelled), being sure to do so by referencing the comma positions through the *variables* to which they are bound.
- 8. Run the program.

#### Task 8: Point E programming

For this task you will want to use the 2 argument substring method of the String class. If you should need a hint for how to do this, simply look at the part of the StringOps program that make use of the substring method with 2 arguments. Also, please be sure to use the relevant variable that was bound in Task C of this program.

- 1. Introduce a variable called **singerLast** and bind it to the last name of the singer. Be sure to arrange for the computer to compute the last name.
- 2. Introduce a variable called sculptorLast and bind it to the last name of the sculptor. Be sure to arrange for the computer to compute the last name.

- 3. Introduce a variable called **painterLast** and bind it to the last name of the paitner. Be sure to arrange for the computer to compute the last name.
- 4. Introduce a variable called dancerLast and bind it to the last name of the dancer. Be sure to arrange for the computer to compute the last name.
- 5. Introduce a variable called **selfLast** and bind it to your last name. Be sure to arrange for the computer to compute the last name.
- 6. Add the following line of code: System.out.println("\nLast names ...");
- 7. Add five lines of code, one to print each of the five last names (unlabelled), being sure to do so by referencing the comma positions through the *variables* to which they are bound.
- 8. Run the program.

#### Task 9: Point F programming

- 1. Toggle the comments on the six lines **following** the Point F comment.
- 2. Use the red text and light bulb to create a stub for the firstName method.
- 3. Note that IntelliJ guessed at the type of the firstName method we wish to define, but got it wrong with respect to our intentions. IntelliJ also did its best to come up with a reasonable name for the parameter, but came up short on this count too. You should take this opportunity to adjust its guesses by doing the following:
  - (a) Change the type of the method from boolean to String.
  - (b) Change the name of the parameter to directoryStyleName.
- 4. What will be the value of the parameter, directoryStyleName, when the firstName method is called with the name of the singer? Please write it down.
- 5. What will be the value of the parameter, directoryStyleName, when the firstName method is called with the name of the dancer? Please write it down.
- 6. Fill in the body of the firstName method so that it computes and returns the first name of the individual whose name is given by the parameter. Please note that you will find the first name by looking to the parameter!
- 7. Run the program.

## Task 10: Point G programming

- 1. Toggle the comments on the six lines following the Point G comment.
- 2. Use the red text and light bulb to create a stub for the lastName method.
- 3. Edit the *header* of the lastName method in just the same manner that you previously edited the *header* of the firstName method.
- 4. What will be the value of the parameter, directoryStyleName, when the lastName method is called with the name of the singer? Please write it down.
- 5. What will be the value of the parameter, directoryStyleName, when the lastName method is called with the name of the dancer? Please write it down.
- 6. Fill in the body of the lastName method so that it computes and returns the last name of the individual whose name is given by the parameter. Please note that you will find the last name by looking to the parameter!

7. Run the program.

#### Task 11: Point H programming

- 1. Toggle the comments on the six lines following the Point H comment.
- 2. Use the red text and light bulb to create a stub for the fullName method.
- 3. Change the type of the fullName method to String and change the name of the parameter of the fullName method to dsn.
- 4. What will be the value of the parameter, dsn, when the fullName method is called with the name of the singer? Please write it down.
- 5. What will be the value of the parameter, dsn, when the fullName method is called with the name of the dancer? Please write it down.
- 6. Fill in the body of the fullName method so that it computes and returns the full name first name followed by a space followed by the last name of the individual whose name is given by the parameter. CONSTRAINT: It is required that you use both the firstName method and the lastName method to define the fullName method.
- 7. Run the program.

#### Task 12: Post your work

Please post your work for this lab on you Web site. Simply, post the source code and the demo for each of the two featured programs.

## Task 13: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

# 15 Presentation: Strings

This material is presented in real time during the course lectures.
# 16 Lab 8: Array Play

# F. Herbert (First Law of Mentat in DUNE) on UNDERSTANDING PROCESSES

A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join and flow with it.

#### Overview

This lab features three programs. The first two programs, **Primes** and **Streets**, are merely intended to help you to wrap your minds around the basics of array creation and array element referencing. Basically, you will read the **Primes** program and write, by analogy, the **Streets** program. The third program, **ReverseCopy**, features array processing and illustrates file IO by solving a relatively realistic problem, that of reading words from one file and writing them, in reverse order, to another file. This program can serve as a model for subsequent programs that you write involving file IO. The program also contrasts use of the **while** statement and the **for** statement, introduces you to the role that *exceptions* play in computer programming, and makes limited use of *properties* associated with the **System** class.

# Why do it?

As you work through this lab you will:

- 1. Get acquainted with basic array processing functionality.
- 2. Learn to appreciate basic concepts associated with doing simple file IO.
- 3. Perform some file processing reading words from a file and writing words to a file.
- 4. Make use of system properties in your programming.
- 5. See how exceptions play a role in computer programming.

# Task 1: Prepare to do the Java programming for this lab in IntelliJ

- 1. Log on to a sanctioned machine.
- 2. Get into IntelliJ.
- 3. Open the CS1 project, if need be.

# Task 2: Study (read/create/run/reflect upon) the Primes program

The accompanying **Primes** program features an array of single digit primes. By design, the program doesn't do much of anything, in order that attention can be clearly focussed on the bare essentials of array processing.

1. Carefully read through the **Primes** program and the demo that follows it. (The line numbers are included for ease of subsequent referencing.)

```
Program: Primes (just the main method)
```

```
public static void main(String[] args) {
1
2
3
         int[] primes = new int[4];
4
5
         primes[0] = 2;
6
         primes[1] = 3;
7
         primes[2] = 5;
8
         primes[3] = 7;
9
10
         System.out.println("length of primes array = " + primes.length);
         System.out.println("first prime = " + primes[0]);
11
         System.out.println("last prime = " + primes[3]);
12
13
         System.out.println("last prime = " + primes[primes.length-1]);
14
15
         System.out.println("\nThe initial array ...");
16
         int i = 0;
17
         while ( i < primes.length ) {</pre>
18
             System.out.println(primes[i]);
19
             i = i + 1;
20
         }
21
22
         int temp = primes[0];
23
         primes[0] = primes[primes.length-1];
24
         primes[primes.length-1] = temp;
25
26
         System.out.println("\nThe final array ...");
27
         for ( int x = 0; x < primes.length; x = x + 1 ) {
             System.out.println(primes[x]);
28
29
         }
30
     }
31
```

# Demo: Primes

```
length of primes array = 4
first prime = 2
last prime = 7
last prime = 7
The initial array ...
2
3
5
7
The final array ...
7
3
5
2
```

- 2. Within a package called **arrayplay**, establish a Java Class called **Primes** with a reasonable *lead comment* (craft a simple sentence that is appropriate) and the **main** method that has been provided.
- 3. Run the program.
- 4. Take a few minutes to answer the following questions:
  - (a) How does one *declare* an array variable in Java?
  - (b) How does one create an array object containing a certain number of elements in Java?
  - (c) How does one reference an element of an array in Java?
  - (d) How do you reference the *length* of an array in Java?
  - (e) What is the index of the *first* element of an array in Java?
  - (f) What is the index of the *last* element of an array in Java?

# Task 3: Write a Streets program, working by analogy with the Primes program

Write a program called **Streets**, bit by bit, according to the instructions which follow, working by analogy with the **Primes** program. The **Streets** program will feature names of streets that you can walk in the French Quarter of New Orleans.

- 1. Within the arrayplay package, establish a Java Class called Streets.
- 2. Change the *lead comment* to something reasonable.
- 3. Add a line to the main method of your Streets program to declare a String array called streets and bind it to an array capable of storing 12 String objects. (Work by analogy with line 3 of the main method of the Primes program, which creates an int array called primes capable of storing 4 int values.)
- 4. Place the following names of French Quarter streets into the streets array, in the order provided: "Iberville" "Decatur" "Toulouse" "Bourbon" "Dauphine" "Royal" "St Ann" "St Peter" "Conti" "Exchange" "Bienville" "Dumaine". (Work by analogy with lines 5-8 of the main method of the Primes program.)
- 5. Run the program.
- 6. Mimicking the code that appears in lines 10-13 of the main method of the Primes program, add statements to the main method of the Streets program to display, labelled, the length of the featured array, the first element of the featured array, and the last element of the featured array, twice.
- 7. Run the program.
- 8. Mimicking the code that appears in lines 15-20 of the main method of the Primes program, add statements to the main method of the Streets program to display, labelled, the elements of the streets array.
- 9. Run the program.
- 10. Mimicking the code that appears in lines 22-24 of the main method of the Primes program, add statements to the main method of the Streets program to swap the first element and the last element of the streets array.
- 11. Mimicking the code that appears in lines 26-29 of the main method of the Primes program, add statements to the main method of the Streets program to display, labelled, the elements of the streets array.
- 12. Run the program.

# Task 4: Study, implement, and run a program to reverse copy a disk file

Consider the following program. ReverseCopy reads words from one file and writes them in reverse order to a second file. (In this instance, the line numbers are included merely so that you will be better able to refer to lines of the program should you wish to discuss it with someone.)

# Program: ReverseCopy

```
/*
1
      * Program featuring straight up arrays and file IO to read and reverse copy a lyric.
2
      */
3
4
5
     package arrayplay;
6
7
     import java.io.File;
     import java.io.FileNotFoundException;
8
9
     import java.io.IOException;
10
     import java.io.PrintWriter;
11
     import java.util.Scanner;
12
     public class ReverseCopy {
13
```

```
14
15
         public static void main(String[] args) throws FileNotFoundException, IOException {
             String inputFileName = "ForeverYoung.text";
16
             String outputFileName = "ForeverYoungReversed.text";
17
             String[] words = readWordsFromFile(inputFileName);
18
             writeWordsToFile(words,outputFileName);
19
         }
20
21
22
         private static final int LIMIT = 1000;
23
         private static String[] readWordsFromFile(String inputFileName) throws FileNotFoundException {
24
25
             // Equate a scanner with the input file
26
             Scanner scanner = establishScanner(inputFileName);
27
             // Read the words from the file into an oversized array
28
             String[] temp = new String[LIMIT];
29
             int index = 0;
             while ( scanner.hasNext() ) {
30
                 String word = scanner.next();
31
32
                 temp[index] = word;
33
                 index = index + 1;
34
             }
35
             int wordCount = index;
36
             // Transfer the words to a perfectly sized array
37
             String[] words = new String[wordCount];
             for ( int x = 0; x < wordCount; x = x + 1 ) {
38
39
                 words[x] = temp[x];
40
             }
             // Return the words
41
42
             return words;
         }
43
44
         private static void writeWordsToFile(String[] words, String outputFileName) throws IOException {
45
46
             // Equate a printer with an output file
47
             PrintWriter printer = getPrintWriter(outputFileName);
48
             // Print the words to the file
             for ( int x = words.length-1; x \ge 0; x = x - 1 ) {
49
50
                 printer.println(words[x]);
51
             }
52
             printer.close();
53
         }
54
         private static Scanner establishScanner(String inputFileName) throws FileNotFoundException {
55
             String fullFileName = createFullFileName(inputFileName);
56
57
             return new Scanner(new File(fullFileName));
         }
58
59
60
         private static PrintWriter getPrintWriter(String outputFileName) throws FileNotFoundException {
61
             String fullFileName = createFullFileName(outputFileName);
             PrintWriter printer = new PrintWriter(fullFileName);
62
63
             return printer;
64
         }
65
66
         // Create the full file name for a simple file name, assuming that it will be
67
         // found in the CS1Files/data subdirectory of the user's home directory.
```

```
private static String createFullFileName(String fileName) {
68
69
             String separator = System.getProperty("file.separator");
70
             String home = System.getProperty("user.home");
             String path = home + separator + "CS1Files" + separator + "data" + separator;
71
             String fullFileName = path + fileName;
72
73
             return fullFileName;
74
         }
75
76
     }
```

# Illustration



#### Noteworthy aspects of this program

- 1. The partial file names are hard coded in the main method. This is done quite simply in order to focus without distraction on certain essential aspects of file IO. It would be a simple matter to program a more flexible mechanism for establishing file names.
- 2. Full file names are created very carefully, under the assumption that the input file will be found in the data subdirectory of the CS1Files subdirectory of the user's *home* directory, and that the output file will be placed in that same directory.
- 3. Use of the principle of stepwise refinement is clearly reflected in the structure of the program, especially with respect to the file IO.
- 4. A basic array is used to store the words, largely to emphasize the static nature of arrays, and by doing so set the stage for developing a deep appreciation for the List objects that will be featured in the next lab. Words are read into an oversized array and then transferred to a perfectly sized array.
- 5. A Scanner object is equated with the input file in order to facilitate the reading of words from the file.
- 6. A PrintWriter object is equated with the output file in order to facilitate the writing of words to the file.
- 7. System properties are incorporated into the program in the service of creating full file names.
- 8. The concept, and computational manifestation of, the exception is found to be lurking within this program.

#### Subtasks

- 1. Establish a data file.
  - (a) Find yourself a lyric for some song that resonates with you, something other than "Forever Young".
  - (b) Get into Emacs.
  - (c) Establish a buffer with a reasonable name, with the intention of entering the lyric into the file.
  - (d) Enter the lyric by hand into the file, stripping it of all punctuation as you do.
  - (e) Check your file carefully to be sure that it contains no punctuation.
  - (f) Save the file to the data subdirectory of the CS1Files subdirectory of your home directory.

#### 2. Enter the ReverseCopy program.

- (a) Get into IntelliJ.
- (b) Carefully, mindfully, enter the ReverseCopy program as a Java Class program within the arrayplay package, changing the *file names* in the given program to *file names* appropriate to your chosen song lyric. If you are using IntelliJ in a reasonable way, your will proceed in something like the following manner:
  - i. Type in the body of the main method.
  - ii. Use the red text and light bulbs to *create the stubs* for the readWordsFromFile method and the writeWordsToFile method.
  - iii. Add the line for the LIMIT constant.
  - iv. Type in the body of the readWordsFromFile method, and as you do: click on the red text and follow the pop-up instructions to *import* the Scanner class, and use the red text and light bulb that appears to *create the stub* for the establishScanner method.
  - v. Type in the body of the writeWordsToFile method, and as you do: click on the red text and follow the pop-up instructions to *import* the PrintWriter class, and use the red text and light bulb that appears to *create the stub* for the getPrintWriter method.
  - vi. Type in the body of the establishScanner method, and as you do: use the light bulb that appears to *create the stub* for the createFullFileName method, click on the red text and follow the instructions to *import* the File class, and use the light bulb that appears when clicking on red-underlined text to add the *throws* clause for the FileNotFoundException exception to the method signature.
  - vii. Type in the body of the getPrintWriter method, and as you do: use the light bulb that appears when clicking on red-underlined text to add the *throws* clause for the FileNotFoundException exception to the method signature.
  - viii. Type in the body of the createFullFileName method, and as you do: adjust IntelliJ's guess at the parameter name, changing it to fileName, and add the prefacing comment as well.
  - ix. Use the light bulbs appropriately to add the *throws* clauses for the FileNotFoundException exception to the writeWordsToFile method, the readWordsFromFile method, and the main method.
- 3. Run the program.
- 4. Check to see that the program did its job by looking for the output file in the data subdirectory of the CS1Files subdirectory of your *home* directory, and by checking its contents.

# Task 5: Post your work

Please post your work for this lab on you Web site. Post the source code and the demo for each of the first two programs. For the third program, post the source code, the song lyric file, and the file containing the words of the lyric in reverse order.

# Task 6: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

This material is presented in real time during the course lectures.

# 18 Lab 9: Simple List Processing

# **D.** Gelernter on MACHINE BEAUTY

Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defense against complexity. The geniuses of the computer field are the people with the keenest aesthetic senses, the ones who are capable of creating beauty. Beauty is decisive at every level: the most important interfaces, the most important programming languages, the winning algorithms are the beautiful ones.

#### Overview

This lab, like the Array Play lab, features three programs. In fact, the three programs featured in this lab are clear riffs on those featured in the Array Play lab. Whereas the programs in the Array Play lab were dedicated to acquainting you with basic concepts and mechanisms associated with the *array*, the programs in this lab are dedicated to acto acquainting you with basic concepts and mechanisms associated with the *ArrayList*.

# Why do it?

As you work through this lab you will:

- 1. Get acquainted with basic ArrayList processing functionality.
- 2. Have an opportunity to compare and contrast *array* processing (previous lab) with ArrayList processing (this lab).
- 3. Revisit many of the things that you were introduced to in the previous lab.

#### Task 1: Prepare to do the Java programming for this lab in IntelliJ

- 1. Log on to a sanctioned machine.
- 2. Get into IntelliJ.
- 3. Open the CS1 project, if need be.

# Task 2: Study (read/create/run/reflect upon) the accompanying Primes program

The accompanying **Primes** program features an **ArrayList** of single digit primes. Like the **primes** program of the previous lab, this **primes** program doesn't do much of anything, in order that attention can be clearly focused on some bare essentials. This time, the focus in on **ArrayList** processing.

1. Carefully read through the **Primes** program and the demo that follows it. (The line numbers are included for subsequent reference purposes.)

Program: Primes (just the main method)

```
public static void main(String[] args) {
1
2
3
          ArrayList<Integer> primes = new ArrayList<>();
4
5
          primes.add(2);
6
          primes.add(3);
7
          primes.add(5);
8
          primes.add(7);
9
10
          System.out.println("size of primes list = " + primes.size());
          System.out.println("first prime = " + primes.get(0));
11
          System.out.println("last prime = " + primes.get(3));
12
13
          System.out.println("last prime = " + primes.get(primes.size()-1));
14
15
          System.out.println("\nThe initial list ...");
16
          for ( Integer prime : primes ) {
              System.out.println(prime);
17
18
          }
19
20
          int temp = primes.get(0);
21
          primes.set(0, primes.get(primes.size()-1));
22
          primes.set(primes.size()-1,temp);
23
24
          System.out.println("\nThe final list ...");
25
          for ( Integer prime : primes ) {
26
              System.out.println(prime);
          }
27
28
29
      }
```

Demo: Primes

```
size of primes list = 4
first prime = 2
last prime = 7
last prime = 7
The initial list ...
2
3
5
7
The final list ...
7
3
5
2
```

- 2. Within a package called arraylistplay, establish a Java Class called Primes with a reasonable *lead comment* (craft a simple sentence that is appropriate) and the main method that has been provided.
- 3. Run the program.
- 4. Take a few minutes to answer the following questions.
  - (a) How does one declare an ArrayList variable capable of storing objects of a certain type in Java.
  - (b) How does one create an empty ArrayList object of a certain type in Java.
  - (c) How does one reference an element of an ArrayList object in Java.
  - (d) How does one change an element of an ArrayList object in Java.
  - (e) How do you reference the size of an ArrayList object in Java?
  - (f) What is the index of the first element of an ArrayList object in Java?
  - (g) What is the index of the *last* element of an ArrayList object in Java?

# Task 3: Write a Streets program, working by analogy with the Primes program

Write a program called **Streets**, bit by bit, according to the instructions which follow, working by analogy with the **Primes** program. The **Streets** program will feature names of streets that you can walk in the French Quarter of New Orleans.

- 1. Within the arraylistplay package, establish a Java Class called Streets.
- 2. Write a reasonable *lead comment*.
- 3. Create an empty main method.
- 4. Add a line to the main method of your Streets program to declare a variable called streets capable of storing an ArrayList of String values, and bind it to an empty array list. (Work by analogy with line 3 of the main method of the Primes program, which declares a variable called primes capable of storing an ArrayList of Integer values, and binds it to an empty array list.)
- 5. Place the following names of French Quarter streets into the ArrayList object to which the streets variable is bound, in the order provided: "Iberville" "Decatur" "Toulouse" "Bourbon" "Dauphine" "Royal" "St Ann" "St Peter" "Conti" "Exchange" "Bienville" "Dumaine". (Work by analogy with lines 5-8 of the main method of the Primes program.)
- 6. Run the program.
- 7. Mimicking the code that appears in lines 10-13 of the main method of the Primes program, add statements to the main method of the Streets program to display, labelled, the length of the featured array list, the first element of the featured array list, and the last element of the featured array list, twice.
- 8. Run the program.
- 9. Mimicking the code that appears in lines 15-18 of the main method of the Primes program, add statements to the main method of the Streets program to display, labelled, the elements of the array list to which streets is bound.
- 10. Run the program.
- 11. Mimicking the code that appears in lines 20-22 of the main method of the Primes program, add statements to the main method of the Streets program to swap the first element and the last element of the array list to which streets is bound.
- 12. Mimicking the code that appears in lines 24-27 of the main method of the Primes program, add statements to the main method of the Streets program to display, labelled, the elements of the array list to which streets streets is bound.
- 13. Run the program.

#### Task 4: Study, implement, and run a program to reverse copy a disk file

Consider the following program. Like the **ReverseCopy** program of the previous lab, this program reads words from one file and writes them in reverse order to a second file. (The line numbers are included merely so that you will be better able to refer to lines of the program should you wish to discuss it with someone.)

### Program: ReverseCopy

```
1 /*
2 * This program features an ArrayList to do its reverse copy thing
3 * from one file to another.
4 */
5
6 package arraylistplay;
7
```

```
8
     import java.io.File;
9
     import java.io.FileNotFoundException;
     import java.io.IOException;
10
     import java.io.PrintWriter;
11
     import java.util.ArrayList;
12
13
     import java.util.Scanner;
14
     public class ReverseCopy {
15
16
17
         public static void main(String[] args) throws FileNotFoundException, IOException {
             String inputFileName = "DesolationRow.text";
18
             String outputFileName = "DesolationRowReversed.text";
19
20
             ArrayList<String> words = readWordsFromFile(inputFileName);
21
             writeWordsToFile(words, outputFileName);
         }
22
23
         private static ArrayList<String> readWordsFromFile(String inputFileName)
24
25
                 throws FileNotFoundException {
26
             // Equate a scanner with the input file
             Scanner scanner = establishScanner(inputFileName);
27
28
             // Read the words from the file into a dynamically growing ArrayList
             ArrayList<String> words = new ArrayList<>();
29
30
             while (scanner.hasNext()) {
                 String word = scanner.next();
31
                 words.add(word);
32
             }
33
34
             // Return the words
35
             return words;
         }
36
37
         private static void writeWordsToFile(ArrayList<String> words, String outputFileName)
38
                 throws IOException {
39
40
             // Equate a printer with an output file
             PrintWriter printer = getPrintWriter(outputFileName);
41
42
             // Print the words to the file
             for (int x = words.size() - 1; x \ge 0; x = x - 1) {
43
44
                 printer.println(words.get(x));
45
             }
46
             printer.close();
47
         }
48
         private static Scanner establishScanner(String inputFileName)
49
50
                 throws FileNotFoundException {
51
             String fullFileName = createFullFileName(inputFileName);
52
             return new Scanner(new File(fullFileName));
53
         }
54
55
         private static PrintWriter getPrintWriter(String outputFileName)
                 throws FileNotFoundException {
56
57
             String fullFileName = createFullFileName(outputFileName);
             PrintWriter printer = new PrintWriter(fullFileName);
58
59
             return printer;
         }
60
61
```

```
62
         // Create the full file name for a simple file name, assuming that it will be
63
         // found in the CS1Files/data subdirectory of the user's home directory.
         private static String createFullFileName(String fileName) {
64
             String separator = System.getProperty("file.separator");
65
             String home = System.getProperty("user.home");
66
             String path = home + separator + "CS1Files" + separator + "data" + separator;
67
             String fullFileName = path + fileName;
68
69
             return fullFileName;
         }
70
71
     }
72
```

# Illustration



#### Noteworthy aspects of this program

1. An ArrayList is used to store the words. By comparison with the ReverseCopy program of the previous lab, you should be able to appreciate the *dynamic nature of lists*.

Now

- 2. The remaining bits of the program are essentially the same as in the previous version:
  - (a) As with the ReverseCopy program of the previous lab, the file names are hard coded, in order to focus without distraction on certain basic aspects of file IO. But note that a *different lyric* is featured in this ReverseCopy program than was featured in the previous ReverseCopy program.
  - (b) Full file names are created very carefully, under the assumption that the input file will be found in the data subdirectory of the CS1Files subdirectory of the user's *home* directory, and that the output file will be placed in that same directory.
  - (c) Use of the principle of stepwise refinement is clearly reflected in the structure of the program.

- (d) A Scanner object is equated with the input file in order to facilitate the reading of words from the file.
- (e) A PrintWriter object is equated with the output file in order to facilitate the writing of words to the file.
- (f) System properties are incorporated into the program in the service of creating full file names.
- (g) The concept, and computational manifestation of, the exception is found to be lurking within this program.

# The Subtasks

- 1. Establish a data file.
  - (a) Find yourself a lyric for some song that resonates with you, a song that is different from the one that you used in the previous lab.
  - (b) Get into Emacs.
  - (c) Establish a buffer with a reasonable name into which you can enter the lyric.
  - (d) Enter the lyric by hand into the file, stripping it of all punctuation as you do.
  - (e) Check your file carefully to be sure it contains no punctuation.
  - (f) Save the file to the data subdirectory of the CS1Files subdirectory of your home directory.
- 2. Enter the ReverseCopy program.
  - (a) Get into IntelliJ.
  - (b) Carefully, mindfully, enter the ReverseCopy program, changing the *file names* in the given program to *file names* appropriate to your chosen song lyric. Please do this in a way that makes reasonable use of IntelliJ!
- 3. Run the program.
- 4. Check to see that the program did its job by looking for the output file in the data subdirectory of the CS1Files subdirectory of your *home* directory, and by checking its contents.

### Task 5: Post your work

Please post your work for this lab on you Web site. Post the source code and the demo for each of the first two programs. For the third program, post the source code, the song lyric file, and the file containing the words of the lyric in reverse order.

# Task 6: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

This material is presented in real time during the course lectures.

# 20 Lab 10: Establishing and Using Classes

# D. Knuth on THE UTILITY OF TOY PROBLEMS

The educational value of a problem given to a student depends mostly on how often the thought processes that are invoked to solve it will be helpful in later situations. It has little to do with how useful the answer to the problem may be. On the other hand, a good problem must also motivate the students; they should be interested in seeing the answer. Since students differ so greatly, I cannot expect everyone to like the problems that please me.

#### Overview

This lab features two programs. The first program, Die, is a *class* which models a die – an object of chance that you tend to roll. The second program, Roller, creates and uses Die objects – instances of the Die class.

# Why do it?

As you work through this lab you will:

- 1. Learn something about modeling classes of objects.
- 2. Gain additional practice in creating and using computational objects.
- 3. "Mechanically" translate from a for statement to a while statement.

#### Conceivable demo for the eventual Roller program

```
run:
Roll a standard die 5 times ...
26565
Roll a twenty sided die 5 times ...
18 9 10 4 7
Roll a standard die 20 times ...
6 1 5 2 3 1 4 6 6 1 4 1 1 4 1 4 4 2 2 4
Roll a standard die 30 times ...
4 2 1 6 5 1 6 6 5 2 6 5 1 2 5 4 5 4 5 5 6 5 3 2 5 4 6 4 6 1
Roll a nine sided die 20 times ...
94432544651584517398
Roll a nine sided die 30 times ...
8 4 6 7 4 5 4 7 8 7 1 9 4 2 7 4 1 9 4 2 2 1 9 1 9 9 5 6 5 4
Ten times, roll a stanard die for a 1.
3 2 3 1
4 4 2 2 3 6 5 5 6 6 3 6 1
4 5 5 4 4 2 1
```

```
56654361
4 3 2 6 6 5 2 5 1
2 4 5 5 2 3 1
4 5 2 6 2 6 5 2 1
6 1
1
4 4 1
Ten times, roll a twelve sided die for a 1.
3 8 8 4 12 9 9 10 10 8 6 6 2 9 11 11 1
931
3 10 10 5 5 11 10 3 1
9 1
3 11 9 7 5 1
7 12 7 2 6 9 11 2 10 9 10 1
9 9 5 10 7 4 9 7 11 5 12 2 6 5 5 8 6 3 4 8 8 8 2 8 9 6 12 4 8 2 12 1
4 2 12 1
12 9 10 3 11 5 4 1
10 2 6 7 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Task 1: Prepare to do the Java programming for this lab in IntelliJ

- 1. Log on to a sanctioned machine.
- 2. Get into IntelliJ.
- 3. Open the CS1 project, if need be.

### Task 2: Create the Die class

Within a package called chance, mindfully establish a Java Class called Die that looks like the following program:

```
/*
 * Model a die in terms of two properties:
 * - order, the number of faces
 * - top, the value of the top face
 */
package chance;
public class Die {
    // THE INSTANCE VARIABLES (STATE)
    private int order;
    private int top;
    // THE CONSTRUCTORS
    public Die() {
        order = 6;
        top = (int) ( ( Math.random() * 6 ) + 1);
    }
```

```
public Die(int nrOfSides) {
    order = nrOfSides;
    top = (int) ( (Math.random() * nrOfSides ) + 1);
}
// THE METHODS (BEHAVIOR)
public int top() {
    return top;
}
public void roll() {
    top = (int) ( (Math.random() * order ) + 1);
}
```

Task 3: Create the Roller program

}

Within a package called chanceapps, establish a Java Class called Roller. One might call Roller a Java Main Class – it is one which can be executed because it has a main method (note that Die does not). Mindfully write the Roller class so that it looks like the following program:

```
/*
 * Program to make use of the Die class.
 */
package chanceapps;
import chance.Die;
public class Roller {
    public static void main(String[] args) {
        // CREATE A STANDARD DIE AND ROLL IT 5 TIMES
        createAndRollStandardDieFiveTimes();
        // CREATE A TWENTY SIDED DIE AND ROLL IT 5 TIMES
        createAndRollTwentySidedDieFiveTimes();
        // CREATE A STANDARD DIE AND ROLL IT 20 TIMES
        createAndRollStandardDie(20);
        // CREATE A STANDARD DIE AND ROLL IT 30 TIMES
        createAndRollStandardDie(30);
        // CREATE A TEN SIDED DIE AND ROLL IT 20 TIMES
        createAndRollNineSidedDie(20);
        // CREATE A NINE SIDED DIE AND ROLL IT 30 TIMES
        createAndRollNineSidedDie(30);
        // TEN TIMES, CREATE A STANDARD DIE AND ROLL IT UNTIL YOU GET A 1
        // System.out.println("Ten times, roll a standard die for a 1.");
        for (int i = 1; i <= 10; i++) {
            createAndRollStandardDieFor1();
        }
```

```
// TEN TIMES, CREATE A TWELVE SIDED DIE AND ROLL IT UNTIL YOU GET A 1
    // System.out.println("Ten times, roll a twelve sided die for a 1.");
    for (int i = 1; i <= 10; i++) {
        createAndRollTwelveSidedDieFor1();
    }
}
private static void createAndRollStandardDieFiveTimes() {
    System.out.println("Roll a standard die 5 times ...");
    Die die = new Die();
    die.roll(); System.out.print(die.top() + " ");
    System.out.println();
}
private static void createAndRollTwentySidedDieFiveTimes() {
    throw new UnsupportedOperationException("Not supported yet.");
}
private static void createAndRollStandardDie(int nrOfTimes) {
    throw new UnsupportedOperationException("Not supported yet.");
}
private static void createAndRollNineSidedDie(int nrOfTimes) {
    throw new UnsupportedOperationException("Not supported yet.");
}
private static void createAndRollStandardDieFor1() {
    throw new UnsupportedOperationException("Not supported yet.");
}
private static void createAndRollTwelveSidedDieFor1() {
    throw new UnsupportedOperationException("Not supported yet.");
}
```

### Task 4: Run / study the Roller program

Run the **Roller** program. Take a look at the output, including the output associated with the exception that was thrown. Can you anticipate the tasks that are awaiting you?

#### Task 5: Refine the createAndRollTwentySidedDieFiveTimes method

- 1. Replace the throw statement in the createAndRollTwentySidedDieFiveTimes method so that the method does what its name suggests. Work by *direct analogy* with the createAndRollStandardDieFiveTimes method. Just be sure to create a twenty sided die with the "nonstandard" constructor rather than a standard die with the "standard" constructor.
- 2. Run the Roller program.

}

#### Task 6: Reflection / rewriting

1. Consider the createAndRollStandardDieFiveTimes method. Does the way it is written invite you to think of an alternative way of writing it? In the space provided below (not in IntelliJ), rewrite the method replacing the five identical lines with a for statement that accomplishes the exact same task.

2. Consider the createAndRollTwentySidedDieFiveTimes method. Does the way it is written invite you to think of an alternative way of writing it? In the space provided below (not in IntelliJ), rewrite the method replacing the five identical lines with a for statement that accomplishes the exact same task.

#### Task 7: Refine the createAndRollStandardDie method

1. Replace the throw statement in the createAndRollStandardDie method so that the method creates and rolls a standard die the number of times specified by the value of the parameter. In doing so, simply enter the following code:

#### Code for the createAndRollStandardDie method

```
System.out.println("Roll a standard die " + nrOfTimes + " times ...");
Die lucky = new Die();
for (int i = 1; i <= nrOfTimes; i = i + 1) {
    lucky.roll();
    System.out.print(lucky.top() + " ");
}
System.out.println();
```

#### 2. Run the Roller program.

### Task 8: Translate the for statement to a while statement

1. Study the code in the createAndRollStandardDie method. Note the existence of a for statement. Change the for statement to a while statement in such a way that the behavior of the method is exactly the same. Rather than simply discarding the for statement and writing a while statement from scratch, perform a "mechanical translation" based on the following mapping of a for abstraction to a while abstraction:

#### Mechanical procedure for translating for to while

```
The for statement ...
for (INITIALIZATION; TEST; CHANGE) {
    STATEMENT-SEQUENCE
}
can be written in terms of the while statement ...
INITIALIZATION
while (TEST) {
    STATEMENT-SEQUENCE
    CHANGE
}
```

2. In order to assure yourself that you actually performed the mechanical translation correctly, run the Roller program.

#### Task 9: Refine the createAndRollNineSidedDie method

- 1. Replace the throw statement in the createAndRollNineSidedDie method so that the method does what its name suggests. Work by direct analogy with the createAndRollStandardDie method, as modified in the previous task.
- 2. Run the Roller program.

#### Task 10: Refine the createAndRollStandardDieFor1 method

- 1. Toggle the comment in the main method that reports the impending roll of a standard die for a 1 ten times.
- 2. Replace the throw statement in the createAndRollStandardDieFor1 method so that the method does what its name suggests. In doing so, base your Java code on the following pseudocode:

# Pseudocode to roll a standard die for a 1

create the die
roll the die
print the top face of the die followed by a space -- using print rather than println
while ( the top face is not a 1 ) do the following
 roll the die
 print the top face of the die followed by a space -- using print rather than println
end of the while
issue a println command (just to terminate printing on the line)

3. Run the Roller program.

#### Task 11: Refine the createAndRollTwelveSidedDieFor1 method

- 1. Toggle the comment in the main method that reports the impending roll of a twelve sided die for a 1 ten times.
- 2. Replace the throw statement in the createAndRollTwelveSidedDieFor1 method so that the method does what its name suggests. Work by analogy with the createAndRollStandardDieFor1 method.
- 3. Run the Roller program.

#### Task 12: Post your work

Please post your work for this lab on you Web site. Post the source code for Die and Roller. Post the final run of the Roller program as a demo.

# Task 13: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

21 Presentation: Class Definition, Interface Implementation

This material is presented in real time during the course lectures.

# 22 Lab 11: Modeling Objects with Classes

# D. Gelernter on OBJECT-ORIENTED PROGRAMMING

Object-oriented programming as it emerged in Simula 67 allows software structure to be based on real-world structures, and gives programmers a powerful way to simplify the design and construction of complex programs.

#### Overview

In this lab you are guided through the process of writing a class to model a person. The Person class will have 5 instance variables and one constructor. Initially, it will have just one method, the toString method. You will be given instruction on how to write a program, PersonDemo1, to test this class in its initial form.

You will then be introduced to the idea of defining and implementing a *Java interface*. You will be guided through the process of enhancing the initial Person class by *implementing* the interface for the class. Along with this modification to the Person class, you will be instructed on how to modify the PersonDemo1 test program in order to assure that the additional functionality is working.

Finally, you will be asked to write an alternate version of the test program, PersonDemo2, one which features an array of Person objects.

# Why do it?

As you work through this lab you will:

- 1. Craft a class more or less from scratch (model an object).
- 2. Establish a Java interface.
- 3. Implement a Java interface.
- 4. Write test programs.
- 5. Engage in the process of incremental program development.
- 6. Practice using arrays of objects.

# Task 1: Prepare to do the Java programming for this lab in IntelliJ

- 1. Log on to a sanctioned machine.
- 2. Get into IntelliJ.
- 3. Open the CS1 project, if need be.
- 4. Note that the tasks in this lab start out being very short. Please don't be unnerved by this!

#### Task 2: Establish a Person Java Class file

Within a package called **people**, establish a new Java Class file called **Person** in which to model a person (i.e., in which to develop a class which can be used to represent **Person** objects).

#### Task 3: Write the lead comment

Write a lead comment to reflect the fact that this program will model a person in terms of five properties, first name and last name (String values), month, day, and year of birth (int values).

# Task 4: Recall the basic approach to modeling a class

When writing a class you must: (1) establish instance variables, (2) define any nontrivial constructors, and (3) define some number of methods. Just be mindful of this as you proceed.

#### Task 5: Establish the instance variables

Within the class, establish the five instance variables by means of five distinct variable declarations, using the qualifier private for each. Call the instance variables firstName, lastName, month, day, and year. Here is one of the five lines:

private String firstName;

#### Task 6: Define a constructor

Define a constructor with **four** parameters. The first will be a **String** called **name**, and will represent a name as a first name followed by a space followed by a last name. For example, **name** might be bound to "William Smith", or **name** might be bound to "Maggie Jones". The second will be an int called **month**, the third will be an **int** called **day**, and the fourth will be an **int** called **year**. Your job within this constructor will be to bind the **five** instance variables to appropriate values. The most interesting aspect of writing this constructor is that you will have to extract the first name from the **name** parameter and also the last name from the **name** parameter. (You should have plenty of experience doing this sort of thing from your engagement in the *String Thing* lab.) Also of interest is the fact that you will have to disambiguate like named instance variables and parameters using: **this**.

#### Task 7: Define a parameterless toString method

This public method will simply return a String value of the form "FIRST LAST, born MONTH/DAY/YEAR", where the SLANTEDCAPS words are intended to be replaced by the values of the appropriate instance variables for the object.

#### Task 8: Establish a demo program for the Person class

Create a PersonDemo1 Java Main Class within your people package which is a completion of the following *partial* program in that it will create and textually display six Person objects, one for Bob Dylan, one for Noomi Rapace, one for Pharrell Williams, one for Frank Sinatra, one for Diana Krall, and one for you.

# PersonDemo1 Program

```
/*
 * PersonDemo1 is a simple program to create and textually display Person
 * objects.
 */
package people;
public class PersonDemo1 {
    public static void main(String[] args) {
        // CREATE THE SIX PERSON OBJECTS
        Person bd = new Person("Bob Dylan",5,24,1941);
        Person nr = new Person("Noomi Rapace",12,28,1974);
        . . .
        . . .
         . . .
        . . .
        // DISPLAY THE SIX PERSON OBJECTS TO THE STANDARD OUTPUT STREAM
        System.out.println(bd);
        System.out.println(nr);
        . . .
        . . .
        . . .
         . . .
    }
}
```

You will know that your Person class is correct if your output is consistent with the following sketch:

# Sketch execution of PersonDemo1

```
Bob Dylan, born 5/24/1941
Noomi Rapace, born 12/28/1974
...
...
```

#### Task 9: Create the PersonSpecification Java interface

An **interface** with respect to the Java programming language is essentially a store of method headers. A class can **implement** an interface by defining all of the methods specified in the interface. In this task, you are to establish a Java interface that you will be asked to implement in the next task.

- 1. Create a Java interface ...
  - (a) Right click on the people package and create a new Java Class.
  - (b) On the Create New Class form that appears ...
    - i. Type PersonSpecification into the Name field.
    - ii. Select Interface from the Kind field.
    - iii. Click OK.
- 2. Modify the template so that it matches the following:

#### PersonSpecification interface

```
/*
 * Person functionality
 */
package people;
public interface PersonSpecification {
    public String firstName();
    public String lastName();
    public int month();
    public int day();
    public int year();
    public String initials();
    public boolean isBoomer();
}
```

#### Task 10: Implement the PersonSpecification interface in the Person class

Perform the implementation of the PersonSpecification interface in the Person class according to the following three step process.

- 1. Change the opening line of Person class so that you are obligated to define all of the methods represented in the PersonSpecification:
  - FROM: public class Person {
  - TO: public class Person implements PersonSpecification {
- 2. Notice that the opening line of the **Person** class is now underlined in red. Click on it, and ask the light bulb to *Implement methods*. When you select that option, a window will appear asking you which methods from the interface you would like to add stubs for. Simply click OK and IntelliJ will generate a stub for each of the methods.
- 3. Refine each stub in a manner consistent with the following semantics:
  - Person.firstName() → String returns the value to which the firstName instance variable is bound
  - Person.lastName()  $\longrightarrow$  String returns the value to which the lastName instance variable is bound

- Person.month()  $\longrightarrow$  int returns the value to which the month instance variable is bound
- Person.day()  $\longrightarrow$  int returns the value to which the day instance variable is bound
- Person.year()  $\longrightarrow$  int returns the value to which the year instance variable is bound
- Person.initials()  $\longrightarrow$  String returns the two character string consisting of the first letter of the first name followed by the first letter of the last name, both in upper case
- Person.isBoomer() → boolean
   returns the value true is the person is a baby boomer, false if not

# Task 11: Modify the PersonDemo1 program

Modify the PersonDemo1 Java Main Class file within your people package so that it is the reasonable completion of the following *partial* program

#### Revised PersonDemo1 program

```
/*
 * PersonDemo1 is a simple program to create and textually display Person
 * objects, together with initials and an indication of whether or not the
 * person is a baby boomer.
 */
package people;
public class PersonDemo1 {
    public static void main(String[] args) {
        // CREATE THE SIX PERSON OBJECTS
        Person bd = new Person("Bob Dylan",5,24,1941);
        Person nr = new Person("Noomi Rapace",12,28,1974);
        . . .
        . . .
        . . .
        . . .
        // DISPLAY THE SIX PERSON OBJECTS TO THE STANDARD OUTPUT STREAM
        System.out.println(bd + " " + bd.initials() + " " + bd.isBoomer());
        System.out.println(nr + " " + nr.initials() + " " + nr.isBoomer());
        . . .
        . . .
        . . .
        . . .
    }
}
```

You will know that your Person class is correct if your output is consistent with the following sketch:

# Sketch execution of the revised PersonDemo1 program

```
Bob Dylan, born 5/24/1941 BD false
Noomi Rapace, born 12/28/1974 NR false
...
...
```

#### Task 12: Create the PersonDemo2 Java Main Class File

Create a PersonDemo2 Java Main Class file within your people package. This program will behave quite like the PersonDemo1 program. It will differ in that this program's Main method will feature an array of Person objects. The form of this method will be:

// CREATE AN ARRAY OF PERSON OBJECTS OF SIZE 6 AND FILL IT WITH THE DATA
...
// USE A FOR LOOP TO DISPLAY THE SIX PERSON OBJECTS IN THEIR TEXTUAL FORM
...

# Task 13: Post your work

Please post your work for this lab on you Web site. Post the source code for PersonSpecification, Person, PersonDemo1 and PersonDemo2. Post the final run of the PersonDemo1 program and the run of the PersonDemo2 program as demos.

# Task 14: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?
This material is presented in real time during the course lectures.

# 24 Lab 12: Grapheme to Color Synesthesia

# Seymour Papert on THE SIGNIFICANCE OF LEARNING TO PROGRAM

My basic idea is that programming is the most powerful medium of developing the sophisticated and rigorous thinking needed for mathematics, for grammar, for physics, for statistics, for all the "hard" subjects. In short, I believe more than ever that programming should be a key part of the intellectual development of people growing up.



### Overview

This lab features a program that simulates grapheme to color synesthesia. It takes the form of an interpreter. Two parallel arrays are featured in the grapheme to color mapping. Strings and arrays cooperatively perform the essential elements of the computation. The *simple painter* class easily supports the graphics processing.

# Why do it?

As you work through this lab you will:

- 1. Experience programming with parallel arrays.
- 2. Get acquainted with sequential search.
- 3. Focus yet again on the process of interpretation.
- 4. See some interesting interactions between strings and arrays.
- 5. Appreciate how the painter can render text synesthetically.

### Task 1: Prepare to do the Java programming for this lab in IntelliJ

1. Log on to a sanctioned machine.

- 2. Get into IntelliJ.
- 3. Open the CS1 project, if need be.

### Task 2: Create the GraphemeToColorSynesthesia program

Within a package called synesthesia, establish the following Java Class program, which simulates grapheme to color synesthesia. Please do so in a *mindful* manner!

#### GraphemeToColorSynesthesia program

```
/*
 * Program to simulate the phenomenon known as grapheme to color synesthesia.
 * This program is written as an interpreter that recognizes and responds to:
 * - exit | terminate the program
 * - remap | redefine the mapping from letters to colors
 * - WORD OR PHRASE | simply show the word or phrase in synesthetic color
 */
package synesthesia;
import java.awt.Color;
import java.awt.Point;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import painter.SPainter;
public class GraphemeToColorSynesthesia {
    private static final int fontsize = 30;
   private static final String theLetters = "AEIOU";
   private static String[] letters;
   private static Color[] colors;
   private void paintingCode() {
        // INITIALIZATION
        SPainter miro = new SPainter(1200,220);
        miro.setScreenLocation(30, 30);
        miro.setFontSize(fontsize);
        initializeColorMap(theLetters);
        // INTERPRETATION
        while ( true ) {
            String input = JOptionPane.showInputDialog(null,
                    "Please enter a word, or a few words ...");
            if ( input == null ) { input = "EXIT"; }
            input = input.toUpperCase();
            if ( input.equals("EXIT") ) {
                break:
            } else if ( input.equals("REMAP") ) {
```

```
initializeColorMap(theLetters);
            showLetters(miro,theLetters);
        } else {
            showLetters(miro,input.toUpperCase());
        }
    }
    miro.end();
}
private static void showLetters(SPainter miro, String input) {
    // READY
    eraseWhiteBoard(miro);
    // SET
    miro.moveTo(new Point.Double(100,100));
    // GO
    for ( int i = 0; i < input.length(); i = i + 1 ) {</pre>
        String letter = input.substring(i, i+1);
        Color color = getLetterColor(letter);
        miro.setColor(color);
        miro.draw(letter);
        miro.mrt(fontsize);
    }
}
private static void initializeColorMap(String specialLetters) {
    letters = new String[specialLetters.length()];
    colors = new Color[specialLetters.length()];
    for ( int i = 0; i < specialLetters.length(); i = i + 1) {</pre>
        letters[i] = specialLetters.substring(i,i+1);
        colors[i] = randomColor();
    }
}
private static Color getLetterColor(String letter) {
    for ( int i = 0; i < letters.length; i = i + 1 ) {
        if ( letter.equalsIgnoreCase(letters[i]) ) {
            return colors[i];
        }
    }
    return Color.BLACK;
}
private static Color randomColor() {
    int rv = (int)(Math.random()*256);
    int gv = (int)(Math.random()*256);
    int bv = (int)(Math.random()*256);
    return new Color(rv,gv,bv);
}
private static void eraseWhiteBoard(SPainter miro) {
    miro.setColor(Color.WHITE);
    miro.wash();
    miro.paintFrame(Color.black, 5);
}
```

```
// INFRASTRUCTURE FOR SOME SIMPLE PAINTING
public GraphemeToColorSynesthesia() {
    paintingCode();
}
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new GraphemeToColorSynesthesia();
        }
    });
}
```

Task 3: Study the program

The program makes use of *parallel arrays* to store the grapheme to color synesthetic knowledge. Although this is a very basic form of knowledge representation, parallel arrays can be *practically* useful on occasion, and *theoretically* interesting as a basis for comparison with more sophisticated knowledge representations.

1. Write down, in English, at the level of the array and the array element, a description of the initializeColorMap method.

2. Write down, in English, at the level of the array and the array element, a description of the getLetterColor method.

3. Write down, in English, at the level of the array and the array element, a description of the showLetters method.

# Task 4: Run the program

Enter different words and short phrases. Occasionally issue the **remap** command. Eventually enter the **exit** command.

### Task 5: Extend the program

Change the program so that it will map all of the letters of the alphabet to colors, not just the vowels. (Once you determine what to do, it should take you about 4 seconds to do this.)

# Task 6: Run the program

Enter different words and short phrases. Occasionally issue the remap command. Eventually enter the exit command.

### Task 7: Post your work

Please post your work for this lab on you Web site. Post the source code for the GraphemeToColorSynesthesia program, and at least one snapshot of text with all of its letters colored.

# Task 8: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it? This material is presented in real time during the course lectures.

# 26 Lab 13: Chromesthesia

# James Lovelock on COMPUTER PROGRAMMING AND LEARNING TO PROGRAM

Composing computer programs to solve scientific problems is like writing poetry. You must choose every word with care and link it with the other words in perfect syntax. There is no place for verbosity or carelessness. To become fluent in a computer language demands almost the antithesis of modern loose thinking. It requires many interactive sessions, the hands-on use of the device. You do not learn a foreign language from a book, rather you have to live in the country for years to let the language become an automatic part of you, and the same is true for computer languages.

### Overview

In this lab you get to make good use of both the NPW and and the MMW! You will simulate the experience of a chromesthete, someone who automatically, inflexibly, maps pitch classes to colors.

The intent of the first task is largely to afford you an opportunity to study a program, by which I mean study certain computational constructs used in context. The program can play notes chromestetically, but only for a subset of the pitch classes that make up the most standard of scales, and only for a single note duration. The second task asks that you extend the program so that all pitch classes of the scale can be rendered. The third task invites you extend the program so that the notes can be played with a variety of durations. Taken as a whole, this lab illustrates the process of *incremental programming*.

# Why do it?

As you work through this lab you will:

- 1. Gain experience with the technique of incremental programming.
- 2. Play with arrays of objects.
- 3. Get acquainted with a music knowledge representation.
- 4. Contribute the development of a multi-sensory program a chromesthetic program, in particular.

### Task 1: Create version 0 of the Chromesthesia program

- 1. FYI, the code featured in this lab is substantial enough that I have placed it right at the end of this lab. Look for it, both the *Chromesthesia* Java Main Class and the *Pitch* Java Class, at the end of this lab when you are asked to refer to the accompanying code.
- 2. Within a package called chromesthesiaO of your CS1 project, establish the accompanying Chromesthesia program as a Java Main Class.
- 3. Within the chromesthesia0 package, establish the accompanying Pitch program as a Java Class.
- 4. Run the Chromesthesia program, and enter one of the following lines, in turn, each time the text input box appears:

(a) C, D, E, C D E c d e
(b) C, C c D, D d E, E e
(c) C D E F G
(d) C C D D E E F F G F E D C C C C
(e) EXIT

#### Task 2: Create version 1 of the Chromesthesia program

- 1. Within a package called chromesthesia1 of your CS1 project, establish a program called Chromesthesia as a Java Main Class. Replace all of the text within this Chromesthesia program of the chromesthesia1 package with all of the text within the Chromesthesia program of the chromestisia0 package.
- 2. Within the chromesthesial package of your CS1 project, establish a program called Pitch as a Java Class. Replace all of the text within the Pitch program of the chromesthesial package with all of the text within the Pitch program of the chromestisia0 package.
- 3. Edit in the few obvious places. Run the Chromesthesia program of the chromesthesia1 package, and check it out to make sure that it works just like the Chromesthesia program of the chromesthesia0 package. If it does, good. If not, fix things so that it does. Once everything is in order, you are in a position to carry on with the development of the program within the chromesthesia1 package.
- 4. Extend the Pitch class so that it processes the three notes F, and F and f. Choose a nice color for this pitch class. Also, extend the establishPitches method of the Chromesthesia class. Test the program.
- 5. Extend the Pitch class so that it processes the three notes G, and G and g. Choose a nice color for this pitch class. Also, extend the establishPitches method of the Chromesthesia class. Test the program.
- 6. Extend the Pitch class so that it processes the three notes A, and A and a. Choose a nice color for this pitch class. Also, extend the establishPitches method of the Chromesthesia class. Test the program.
- 7. Extend the Pitch class so that it processes the three notes B, and B and b. Choose a nice color for this pitch class. Also, extend the establishPitches method of the Chromesthesia class. Test the program.
- 8. Create a file in a convenient location (perhaps you will want to make just such a location for it) and enter the following lines of text just so that you will be able to copy and paste them at will ...
  - (a) C D E F G A B c c B A G F E D C
  - (b) C D E C C D E C E F G E F G G A G F E C G A G F E C C G, C C G, C
  - (c) C C G G A A G F F E E D D C G G F F E E D G G F F E E D C C G G A A G F F E E D D C
- 9. Run the Chromesthesia program, entering each of the three lines of ABC notation text that you stored in your file.

### Task 3: Create version 2 of the Chromesthesia program

- 1. Within a package called chromesthesia2 of your CS1 project, establish a program called Chromesthesia as a Java Main Class. Replace all of the text within this Chromesthesia program of the chromesthesia2 package with all of the text within the Chromesthesia program of the chromestisia1 package.
- 2. Within the chromesthesia2 package of your CS1 project, establish a program called Pitch as a Java Class. Replace all of the text within the Pitch program of the chromesthesia2 package with all of the text within the Pitch program of the chromestisia1 package.
- 3. Within a package called chromesthesia2 of your CS1 project, establish as a *Java Main Class* called Chromesthesia. Replace all of the text within this Chromesthesia program of this chromesthesia2 package with all of the text within the Chromesthesia program of the chromestisia1 package.
- 4. Within the chromesthesia2 package of your CS1 project, establish as a Java Class called Pitch. Replace all of the text within the Pitch program of the chromesthesia2 package with all of the text within the Pitch program of the chromestisia1 package.

- 5. Edit in the few obvious places. Run the Chromesthesia program of the chromesthesia2 package, and check it out to make sure that it works just like the Chromesthesia program of the chromesthesia1 package. If it does, good. If not, fix things so that it does. Once everything is in order, you are in a position to carry on with the development of the program within the chromesthesia2 package.
- 6. Change the program so that the pitch class to color mapping is as follows:
  - A  $\longrightarrow$  new Color(0,0,255)
  - B  $\longrightarrow$  new Color(0,255,0)
  - C  $\longrightarrow$  new Color(127,0,127)
  - D  $\longrightarrow$  new Color(255,255,0)
  - $E \longrightarrow \text{new Color}(255,0,0)$
  - F  $\longrightarrow$  new Color(255,127,0)
  - G  $\longrightarrow$  new Color(0,255,255)
- 7. Run the program, and give it a thorough testing.
- 8. Arrange for the play method of the Pitch class to function with any of the following three instances of the parameter: "1" or "2" or "1/2". Do this by refining the following suggestive code:

```
public void play(String d) {
    painter.setColor(color);
    painter.paint(box);
    painter.setColor(randomColor());
    painter.draw(box);
    if ( the duration string equals "1" ) {
        simply play the note
    } else if ( the duration string equals "2" ) {
        double the duration of the note; play it; halve the duration
    } else if ( the duration string equals "1/2" ) {
        halve the duration of the note; play it; double the duration
    }
}
```

9. In the Chromesthesia program, change the playMelody method to the following code:

```
private static void playMelody(String input, Pitch[] pitches) throws Exception {
    Scanner scanner = new Scanner(input);
    while ( scanner.hasNext() ) {
        String token = scanner.next();
        String pitchName;
        String duration = "";
        if ( token.indexOf(",") < 0 ) {
            pitchName = token.substring(0,1);
            duration = token.substring(1);
        } else {
           pitchName = token.substring(0,2);
            duration = token.substring(2);
        }
        if (duration.length() == 0) { duration = "1"; }
        Pitch pitch = find(pitchName,pitches);
        pitch.play(duration);
    }
}
```

- 10. Stash the following lines of ABC code in your ABC code stash, and make use of the to give your program a relatively thorough testing.
  - $\rm (a)$  C2 C1 C C1/2 C1/2 E2 E1 E E1/2 E1/2 G2 G1 G G1/2 G1/2

- (b) D,2 D,1 D, D,1/2 D,1/2 F,2 F,1 F, F,1/2 F,1/2 A,2 A,1 A, A,1/2 A,1/2
- (c) b2 b1 b b1/2 b1/2 b1/2 b1/2 b b1 b2
- 11. Extend the duration functionality so that it properly works for: "3" and "1/3" AND "2/3".
- 12. Stash the following lines of ABC code in your ABC code stash, and run your program on each one.
  - (a) C,1/3 C,1/3 C,1/3 C, C,3 C1/3 C1/3 C1/3 C C3 c1/3 c1/3 c c3
  - (b) C,2/3 C,1/3 D,2/3 D,1/3 C,2/3 C,1/3 D,2/3 D,1/3 C,2/3 C,1/3 D,1 D,1 D,1 C,3
- 13. Add one more command to the set of commands that the interpreter can process. This will be the AGAIN command. When you issue it, the most recently entered melodic sequence will be played again. This one is for you to design and implement!
- 14. Run your program, to make sure it is working properly with AGAIN.
- 15. Stash a couple more ABC encoded sequences in your ABC code stash, doing your best to make them interesting, and also to make good use of the note rendering functionality. Play each of them a couple of times and observe.

### Task 4: Post your work

Please post your work for this lab on you Web site. Post the source code for the main Chromesthesia program and the Pitch class, post an image of the *input box*, and post an image of the *canvas* when a note is being played.

### Task 5: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

# The Code

This program is comprised of two files, a Java Main Class file called Chromesthesia and a Java Class file called Pitch. Both are placed in a packaged called chromesthesia0, in anticipation of subsequent versions of the program being placed in other packages.

### The Java Main Class Chromesthesia program

# /\* \* This program interprets melodic lines given in ABC notation as a

- \* chromesthete might.
- \*
- \* A Pitch class will be defined, and will take center stage in the
- \* processing.

\*

- \* Interpreting a melody in ABC notation will amount to flashing
- \* colored rectangles for prescribed durations, while sounding
- \* the pitch! The color of the rectangle will correspond to pitch
- \* class. The duration will correspond to the duration of the note.

```
* For this first version of the program, the duration will be held
 * constant at 1 beat.
 * Three sorts of images will appear on the screen, the chromesthetic
 * output box, a text input box, and an error message box. Simplicity
 * of design is rendered by permitting only one box to be on the screen
 * at a time.
 * ABC represents notes in a manner consistent with these examples:
 * C, D, E, C D E c d e
 * Google ABC music representation if you would like to know more about it.
 */
package chromesthesia0;
import java.util.Scanner;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import painter.SPainter;
public class Chromesthesia {
    // INFRASTRUCTURE FOR THE PROGRAM -- LAUNCHING A "GRAPHICS" THREAD
   public static void main(String[] args) {
        SwingUtilities.invokeLater(new ThreadForGUI());
    }
   private static class ThreadForGUI implements Runnable {
       @Override
       public void run() {
           new Chromesthesia();
       }
    }
   public Chromesthesia() {
        interpreter();
    }
    // FEATURED VARIABLES
    private static SPainter miro;
   private static Pitch[] pitches;
    // THE INTERPRETER
   public static void interpreter() {
        initialization(); // miro and pitches
        while ( true ) {
            String input = getInput();
```

```
if ( input.equalsIgnoreCase("EXIT") ) {
            break;
        } else {
            try {
                playMelody(input,pitches);
            } catch (Exception ex) {
                showErrorMessage(ex.toString());
            }
        }
    }
    cleanup(); // miro has to go
}
// METHODS PERTAINING TO THE CHROMESTHETIC PITCHES
private static Pitch[] establishPitches(SPainter painter) {
    Pitch[] pitches = new Pitch[9];
    Pitch pitchMiddleC = new Pitch("C",painter);
    pitches[0] = pitchMiddleC;
    Pitch pitchLowC = new Pitch("C,",painter);
    pitches[1] = pitchLowC;
    Pitch pitchHighC = new Pitch("c",painter);
    pitches[2] = pitchHighC;
    Pitch pitchMiddleD = new Pitch("D",painter);
    pitches[3] = pitchMiddleD;
    Pitch pitchLowD = new Pitch("D,",painter);
    pitches[4] = pitchLowD;
    Pitch pitchHighD = new Pitch("d",painter);
    pitches[5] = pitchHighD;
    Pitch pitchMiddleE = new Pitch("E",painter);
    pitches[6] = pitchMiddleE;
    Pitch pitchLowE = new Pitch("E,",painter);
    pitches[7] = pitchLowE;
    Pitch pitchHighE = new Pitch("e",painter);
    pitches[8] = pitchHighE;
    return pitches;
}
private static Pitch find(String token, Pitch[] pitches) throws Exception {
    for ( int i = 0; i < pitches.length; i = i + 1 ) {</pre>
        Pitch pitch = pitches[i];
        if ( pitch.abcName().equals(token) ) {
            return pitch;
        }
    }
    throw new Exception("### PITCH " + token + " NOT FOUND");
}
private static void display(Pitch[] pitches) {
    for ( int i = 0; i < pitches.length; i = i + 1 ) {
        System.out.println(pitches[i].toString());
    }
```

```
private static void playMelody(String input, Pitch[] pitches) throws Exception {
    Scanner scanner = new Scanner(input);
    while ( scanner.hasNext() ) {
        String token = scanner.next();
        Pitch pitch = find(token,pitches);
        pitch.play("1");
    }
}
// INITIALIZATION, CLEANUP, GETTING INPUT, ERROR MESSAGING
static private void showErrorMessage(String message) {
    miro.setVisible(false);
    JOptionPane.showMessageDialog(null, message);
}
private static void initialization() {
    // ESTABLISH THE PAINTER AND GIVE IT A SUBSTANTIAL BRUSH WIDTH
    miro = new SPainter("Chromesthesia",500,500);
    miro.setVisible(false);
    miro.setBrushWidth(7);
    // ESTABLISH THE CHROMESTITIC PITCH CLASS OBJECTS
    pitches = establishPitches(miro);
    display(pitches);
}
private static String getInput() {
    miro.setVisible(false);
    String label = "Please enter a melody in ABC notation, or EXIT ...
                                                                            ";
    String input = JOptionPane.showInputDialog(null,label);
    miro.setVisible(true);
    if ( input == null ) { input = ""; }
    return input;
}
static private void cleanup() {
    System.exit(0);
}
```

The Java Class Pitch program

/\*

}

- $\ast$  The Pitch class models the pitch of a note in a manner that will facilitate
- \* the chromesthetic processing of the pitch. A Pitch object will have five
- \* properties:
- \* String name | ABC notation pitch name
- \* SPainter painter | the painting agent
- \* Note note | a note that will be set to the pitch corresponding to the

```
ABC notation pitch name
 *
 * - SRectangle box | an SRectangle object that will chromesthetically
 *
    represent the pitch
 * - Color color | the color associated with the pitch for the presumed
    chromesthete
 *
 */
package chromesthesia0;
import java.awt.Color;
import note.SNote;
import painter.SPainter;
import shapes.SRectangle;
public class Pitch {
    // INSTANCE VARIABLES
    private String abcName;
    private SPainter painter;
   private SRectangle box;
    private SNote note;
   private Color color;
   public Pitch(String abcName, SPainter painter) {
        this.abcName = abcName;
        this.painter = painter;
        this.box = new SRectangle(painter.painterHeight-50,painter.painterWidth-50);
        this.note = createNoteForThisPitch(abcName);
        this.color = getPitchClassColor(abcName.substring(0,1).toUpperCase());
    }
    public String toString() {
        return "[ " + abcName + " | " + note.degree() + " | " + color + " ]";
    }
    public String abcName() {
        return abcName;
    }
    private SNote createNoteForThisPitch(String abcPitchClassName) {
        SNote note = new SNote();
        if ( abcPitchClassName.equals("C") ) {
          // nothing to do
        } else if ( abcPitchClassName.equals("C,") ) {
            note.lp(7);
        } else if ( abcPitchClassName.equals("c") ) {
            note.rp(7);
        } else if ( abcPitchClassName.equals("D") ) {
            note.rp(1);
        } else if ( abcPitchClassName.equals("D,") ) {
            note.lp(6);
        } else if ( abcPitchClassName.equals("d") ) {
            note.rp(8);
        } else if ( abcPitchClassName.equals("E") ) {
```

```
note.rp(2);
    } else if ( abcPitchClassName.equals("E,") ) {
        note.lp(5);
    } else if ( abcPitchClassName.equals("e") ) {
        note.rp(9);
    }
    return note;
}
private Color getPitchClassColor(String letter) {
    if ( letter.equals("C") ) {
        return Color.BLUE;
    } else if ( letter.equals("D") ) {
        return Color.GREEN;
    } else if ( letter.equals("E") ) {
        return new Color(127,0,127);
    } else {
        return Color.BLACK;
    }
}
public void play(String d) {
    painter.setColor(color);
    painter.paint(box);
    painter.setColor(randomColor());
    painter.draw(box);
    if ( d.equals("1") ) {
        note.play();
    }
}
private static Color randomColor() {
    int rv = (int)(Math.random()*256);
    int gv = (int)(Math.random()*256);
    int bv = (int)(Math.random()*256);
    return new Color(rv,gv,bv);
}
```

This material is presented in real time during the course lectures.

# 28 Lab 14: Fun with Fractals

### A. Hertzfeld on PROGRAMMING

It's [programming] the only job I can think of where I get to be both an engineer and an artist. There's an incredible, rigorous, technical element to it, which I like because you have to do very precise thinking. On the other hand, it has a wildly creative side where the boundaries of imagination are the only real limitation.

# H. Morowitz on COMPUTER SCIENCE AND BIOLOGY

Computer science is to biology what calculus is to physics. It's the natural mathematical technique that best maps the character of the subject.

### Overview

A brief introduction to L-Systems will be presented. Then, in the spirit of implementing something of algorithmic consequence with an interdisciplinary flavor (linguistics and biology), a class which represents *L-Systems* will be detailed. This class will serve as the basis of programs to perform some algorithmic composition, and to draw some now classic images, which are rendered according to intriguing mathematical sets. The algorithmic composition makes good use of abstract classes. The set rendering relies on *Turtle Geometry*.

## Why do it?

As you work through this lab you will:

- 1. Incorporate abstract classes into your programming.
- 2. Program a generative algorithm.
- 3. Play with L-Systems and fractals.
- 4. Render L-System strings graphically by means of Turtle Geometry.
- 5. Render L-System strings sonically to perform some algorithmic composition
- 6. Focus on the Javadoc mechanism for describing programs.

### Task 1: Prepare to do the Java programming for this lab in IntelliJ

Continue to work in the CS1 project. But anticipate this lab to take some time! It is not designed to be completed in the lab, or even within the scope of the semester. This one is for all those who would like to do something of some significance in anticipation of subsequent study within the realm of computer programming. Part of your preparation for this lab should be adopting a mindset in which you plan to work on gaining independence with respect to computer programming, and autonomy with respect to crafting goals within a well-defined domain, in this case the application of L-systems to the arts.

### Task 2: Get acquainted with L-Systems

The Wiki page for L-systems is really quite good. You might like to spend some time with it prior to continuing with this lab. That said, all you really need to know about L-Systems is what they are, and that they have application beyond the modeling of algae which inspired biologist Atistid Lindenmayer to invent them.

### What is an L-System?

An L-System, or Lindenmayer system, is a parallel rewriting system involving the following three components:

- 1. an alphabet (set of symbols)
- 2. an axiom (string of symbols)
- 3. a production for each alphabet symbol that maps the symbol into a list of symbols

An L-System can also be viewed as a *formal grammar*, a formalism that defines a set of strings of symbols. If you happen to be familiar with the much more well-known *context free grammar*, you can take delight in the fact that L-Systems are very different in a number of significant respects!

## Algae

For example, here is the Algae L-System, together with the first few generations of the system:

The Algae System:

- 1. Alphabet:  $\{A,B\}$
- 2. Axiom: A
- 3. Productions:
  - (a)  $A \to A B$
  - (b)  $B \to A$

The first several generations, each of which, other than the first, which is just the axiom, are derived from its predecessor simply by replacing *each* symbol by the string it produces according to its production.

- A
- A B
- A B A
- A B A A B
- A B A A B A B A
- A B A A B A B A A B A A B

# Cantor Dust

As a second example, here is the Cantor Dust L-System, together with the first few generations of the system:

The Cantor Dust System:

- 1. Alphabet:  $\{A,B\}$
- 2. Axiom: A
- 3. Productions:
  - (a)  $A \to A B A$
  - (b)  $B \to B B B$

The first several generations:

- A
- A B A
- A B A B B B A B A

# Sierpinski Triangle

And as a third example, here is the *Sierpinski Triangle* L-System, together with the first few generations of the system:

The Sierpinski Triangle System:

- 1. Alphabet:  $\{F,G,-,+\}$
- 2. Axiom: F G G
- 3. Productions:
  - (a)  $F \rightarrow F G + F + G F$
  - (b)  $\mathbf{G} \to \mathbf{G} \mathbf{G}$
  - (c)  $\rightarrow -$
  - (d)  $+ \rightarrow +$

The first several generations:

- F G G
- F G + F + G F G G G G
- F G + F + G F G G + F G + F + G F + G G F G + F + G F G -

### Task 3: Create the LSystem Java Class file and the Production Java Class file

Within a package called lsystem, establish two classes, both Java Class files, the first called LSystem to represent L-Systems, and the second called Production to represent the productions of the L-Systems. The code for both

classes is presented here. For now, simply study the code, and type it in to the appropriately named Java Class files. You will have an opportunity to test the code shortly.

### The LSystem Java Class File

```
/*
 * General LSystem class, which will be the super class to particular LSystem
 * classes. It represents an LSystem in terms of its name, its axiom, and its
 * production set.
 */
package lsystem;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
public class LSystem {
    // Instance variables. The axiom and productions are protected so that
    // they can be directly referenced from subclasses. It will be up to
    // the subclasses to fully instantiate the L-Systems by instantiating
    // their axiom and productions instance variables.
    private String name;
    protected String axiom;
   protected List<Production> productions;
    /**
     * Create an LSystem by giving it just its name. In the constructor of
     * the subclass the axiom and the productions will be provided.
     * Cparam name is the name of the L-System
     */
   public LSystem(String name) {
        this.name = name;
    }
    /**
     * Compute a textual representation of the L-System.
     * Oreturn the textual representation of the L-System
     */
   public String toString() {
        return "Name = " + name + "n" +
               "Axiom = " + axiom + "n" +
               "Productions ... \n" + textRepresentation(productions);
    }
    private String textRepresentation(List<Production> productions) {
        String text = "";
        for ( Production p : productions ) {
            text = text + p.toString() + "\n";
```

```
}
    return text;
}
/**
 * Compute the generation of the L-System indicated by the given value.
 * Oparam generationNumber indicates the generation to be produced
 * Oreturn the generation of the L-System indicated by the parameter
 */
protected String generation(int generationNumber) {
    LinkedList<String> generations = new LinkedList<String>();
    String generation = axiom;
    generations.add(generation);
    for ( int i = 1; i <= generationNumber; i++ ) {</pre>
        generation = next(generation);
        generations.add(generation);
    }
    return generations.getLast();
}
/**
 * Produce/display some desired number of generations of the L-System.
 * The user is asked for the number.
 */
protected void generate() {
    System.out.print("How many generations? ");
    Scanner scanner = new Scanner(System.in);
    int nrOfGenerations = scanner.nextInt();
    LinkedList<String> generations = new LinkedList<String>();
    String generation = axiom;
    System.out.println("generation 0 = " + generation);
    generations.add(generation);
    for ( int i = 1; i <= nrOfGenerations; i++ ) {</pre>
        generation = next(generation);
        generations.add(generation);
        System.out.println("generation " + i + " = " + generation);
    }
}
private String next(String generation) {
    String result = "";
    Scanner scanner = new Scanner(generation);
    while ( scanner.hasNext() ) {
        String symbol = scanner.next();
        Production production = find(symbol, productions);
        result = result + production.sequence() + " ";
    }
    return result.trim();
}
private Production find(String symbol, List<Production> productions) {
    for ( Production production : productions ) {
        if ( production.symbol().equalsIgnoreCase(symbol) ) {
            return production;
```

```
}
}
return null;
}
```

The Production Java Class File

/\* \* This class is used by the LSystem class in order to help represent the \* productions of the system. It stores the left hand side of the production \* as a symbol and the right hand side as a sequence of symbols, representing \* everything in terms of character strings. \*/ package lsystem; public class Production { // Instance varibales: the left hand side and the right hand side of the // production. private String symbol; private String sequence; /\*\* \* Create an LSystem production by providing its left hand side and its \* right hand side. \* Oparam symbol is the left hand side of the production \* Oparam sequence is the right hand side of the production \*/ public Production(String symbol, String sequence) { this.symbol = symbol; this.sequence = sequence; } /\*\* \* Referencer for the left hand side of the production. \* Creturn the left hand side of the production \*/ public String symbol() { return symbol; } /\*\* \* Referencer for the right hand side of the production. \* Creturn the right hand side of the production \*/ public String sequence() { return sequence;

```
}
/**
 * Compute a simple textual representation of the L-System production.
 * @return the textual representation of the production
 */
public String toString() {
   return symbol + " --> " + sequence;
}
```

# Task 4: Demo of the AlgaePainter Java Main Class file

The images displayed below are renderings of Algae strings produced by the AlgaePainter program. A demo of the run of the program that produced the images is presented so that you will have an opportunity to think on the behavior of the program, which takes the form of an interpreter, prior to establishing it, running it, varying its specialized painters, and running it some more. Please just enjoy a quick look at the images and study the demo in preparation for the next several tasks.

# Images Generated by AlgaePainter



### Standard IO demo of AlgaePainter

```
run:
Algae>>> help
HELP - display this help menu to the standard output stream
DISPLAY - display the Algae L-System, vocabulary and axiom and productions
GENERATE - generate some number of generations, as specified by the user
PAINT - paint a rendering of some generation of the Algae system
DISPOSE - get rid of the canvas on which the rendering was painted
EXIT - terminate execution of this program
Algae>>> display
Name = Algae
Axiom = A
Productions ...
A --> A B
B --> A
Algae>>> generate
How many generations? 6
generation 0 = A
generation 1 = A B
generation 2 = A B A
generation 3 = A B A A B
generation 4 = A B A A B A B A
generation 5 = A B A A B A B A A B A A B
generation 6 = A B A A B A B A A B A A B A A B A A B A A A A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B
Algae>>> paint
Which player (ALines or ACircles or ASquares)? ALines
Generation number? 8
Algae>>> dispose
Algae>>> paint
Which player (ALines or ACircles or ASquares)? ACircles
Generation number? 10
Algae>>> dispose
Algae>>> paint
Which player (ALines or ACircles or ASquares)? ASquares
Generation number? 8
 Algae>>> dispose
 Algae>>> exit
BUILD SUCCESSFUL (total time: 3 minutes 54 seconds)
```

## Task 5: Create some painters to render some Algae System images

Establish four Java Class files in a package called **painters**. First establish an *abstract class* which contains the basic logic for rendering strings of As and Bs. All this class lacks is any notion of what it means to render A or B. Then establish three different classes which *inherit* the functionality of the abstract class, and which complete the abstract class by defining just what it means to render an A and to render a B. For now, simply study the code, and then enter the code for each of the four classes (the abstract class and the three refinements). Soon enough you will have an opportunity to test the code.

```
/*
```

```
* This abstract class serves to render images based on strings of As and Bs.
 * The renderer is coded in such a way that the rendering of each symbol is
 * left left unspecified. To complete the renderer, the methods thingA and
 * thingB must be specified. That is the job of the classes which extend this
 * abstract class.
 */
package painters;
import java.awt.Color;
import painter.SPainter;
import java.util.Scanner;
public abstract class ABPainter {
    // The simple painter, with its canvas and everything else, is the sole
    // instance variable for this class.
   protected SPainter painter;
    /**
     * Create an ABPainter, a painter which bases its work on strings of As and
     * Bs. It is basically a simple painter (SPainter) which processes the As
     * and the Bs in the string, one at a time, by somehow graphically rendering
     * them.
     * Oparam p is the work horse painter
     */
    public ABPainter(SPainter p) {
        painter = p;
        painter.setScreenLocation(25,25);
        painter.toFront();
        painter.setVisible(true);
        painter.setColor(Color.BLACK);
    }
    /**
     * Paint an image by processing the given string of A and B symbols.
     * Oparam line is a string of As and Bs, presumably generated by some
     * L-System
     */
    public void paint(String line) {
        Scanner symbolString = new Scanner(line);
        while ( symbolString.hasNext() ) {
            String symbol = symbolString.next();
            if ( symbol.equals("A") ) {
                thingA();
            } else if ( symbol.equals("B") ) {
                thingB();
            }
```

```
}
}
/***
* Reference to an encoding of what it means to render the A symbol
* graphically.
*/
public abstract void thingA();
/**
* Reference to an encoding of what it means to render the A symbol
* graphically.
*/
public abstract void thingB();
```

The ABPainterALines Java Class file

```
/*
 * Subclass of the abstract ABPainter class which renders line images that
 * look something like ferns, something like trees, in terms of the symbols
 * A and B.
 */
package painters;
import java.awt.Color;
import painter.SPainter;
public class ABPainterALines extends ABPainter {
    /**
     * Create a specialization of an ABPainter which renders odd looking
     * fern like tree structures with red and blue limbs.
     * Cparam painter is the work horse painter
    */
   public ABPainterALines(SPainter painter) {
        super(painter);
        painter.mbk(165);
    }
    static private double distance = 90;
    static private int delta = 45;
    /**
     * Draw a red branch, to the right, off of a bit of black trunk.
     */
    public void thingA() {
        painter.dfd(distance);
        painter.tr(delta);
```

```
painter.setColor(Color.RED);
    painter.dfd(distance);
    painter.setColor(Color.BLACK);
    painter.mbk(distance);
    painter.tl(delta);
    painter.mbk(distance);
    painter.mfd(distance/10);
}
/**
 * Draw a blue branch, to the left, off of a bit of black trunk.
 */
public void thingB() {
    painter.dfd(distance);
    painter.tl(delta);
    painter.setColor(Color.BLUE);
    painter.dfd(distance);
    painter.setColor(Color.BLACK);
    painter.mbk(distance);
    painter.tr(delta);
    painter.mbk(distance);
    painter.mfd(distance/10);
}
```

```
}
```

# The ABPainterACircles Java Class file

```
/*
 * Subclass of the abstract ABPainter class which renders "dots images" that
 * look something like cones in terms of the symbols A and B.
 */
package painters;
import java.awt.Color;
import java.util.Random;
import painter.SPainter;
import shapes.SCircle;
public class ABPainterACircles extends ABPainter {
    /**
     * Create a specialization of an ABPainter which renders odd looking
     * cone-like structures in red and blue.
     * Cparam painter is the work horse painter
     */
    public ABPainterACircles(SPainter painter) {
        super(painter);
        painter.mbk(60);
    }
```

```
static private double distance = 4;
static private int delta = 1;
static private Random random = new Random();
static private SCircle dot = new SCircle(160);
/**
 * Draw a red dot, adjust the position of the painter, and shrink the dot.
 */
public void thingA() {
    painter.setColor(Color.RED);
    painter.paint(dot);
    painter.mfd(distance);
    dot.shrink(delta);
}
/**
 * Draw a blue dot, adjust the position of the painter, and shrink the dot.
 */
public void thingB() {
    painter.setColor(Color.BLUE);
    painter.paint(dot);
    painter.mfd(distance);
    dot.shrink(delta);
}
```

The ABPainterASquares Java Class file

```
/*
 * Subclass of the abstract ABPainter class which renders images based on
 * squares that look something like melting towers in terms of the symbols
 * A and B.
 */
package painters;
import java.awt.Color;
import java.util.Random;
import painter.SPainter;
import shapes.SSquare;
public class ABPainterASquares extends ABPainter {
    /**
     * Create a specialization of an ABPainter which renders odd looking
     * tower-like structures in red and blue.
     * Oparam painter is the work horse painter
     */
    public ABPainterASquares(SPainter painter) {
```

```
super(painter);
    painter.mbk(20);
    painter.tl(3);
}
static private double distance = 6;
static private int delta = 4;
static private Random random = new Random();
SSquare square = new SSquare(280);
/**
 * Draw a red square, adjust the position of the painter, and shrink the
 * the square, and alter the heading of the painter just a bit.
 */
public void thingA() {
    painter.setColor(Color.RED);
    painter.paint(square);
    painter.mfd(distance);
    square.shrink(delta);
    painter.tr(4);
}
/**
 * Draw a blue square, adjust the position of the painter, and shrink the
 * the square, and alter the heading of the painter just a bit.
 */
public void thingB() {
    painter.setColor(Color.BLUE);
    painter.paint(square);
    painter.mfd(distance);
    square.shrink(delta);
    painter.tl(6);
}
```

# Task 6: Establishment of the AlgaePainter Java Main Class file

The following program makes use of the refinements of the ABPainter class in order to draw images based on the Algae strings. It conditionally determines which Algae string to render, and which of the three renderers thus far established to employ. Study it. Then enter it.

### The AlgaePainter Java Class file

### /\*

- $\ast$  This program can generate and process generations of the Algae L-System,
- \* where processing amounts to performing graphical renderings of the strings
- $\ast$  of symbols A and B of the system. The program takes the form of an interpreter.
- \* See the comment prefacing the constructor for additional details.

\*/

```
package lsystem;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import javax.swing.SwingUtilities;
import painter.SPainter;
import painters.ABPainter;
import painters.ABPainterACircles;
import painters.ABPainterALines;
import painters.ABPainterASquares;
public class AlgaePainter extends LSystem {
    /**
     * Create an AlgaePainter object, which is so tightly coupled to its
    * behavior as an interpreter that its private interpreter method is
     * called from this constructor. The program can represent the Algae
     * L-System, and can derive successive generations of strings within
     * the system. Specifically, the interpreter can process the following
     * commands: <br>
     * *  HELP - display this help menu to the standard output stream<br>
     * *  DISPLAY - display the Algae L-System, vocabulary and axiom and productions<br>
     * *  GENERATE - generate some number of generations, as specified by the user<br/>br>
     * *  PAINT - paint a rendering of some generation of the Algae system<br>
     * *  DISPOSE - get rid of the canvas on which the rendering was painted<br>
     * *  EXIT - terminate execution of this program<br>
     * With respect to the painting, a conditional statement determines which
     * of a number of specialized renderers will be used to paint the image.
     * The painter will then do its thing to the particular generation of the
     * Algae system that it has to work with.
     */
    public AlgaePainter() {
        super("Algae");
        axiom = "A";
        productions = productions();
        interpreter();
    }
   private List<Production> productions() {
        Production p1 = new Production("A","A B");
        Production p2 = new Production("B","A");
        ArrayList<Production> productions = new ArrayList<>();
        productions.add(p1);
        productions.add(p2);
        return productions;
    }
    private void interpreter() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Algae>>> ");
```

```
String line = scanner.next();
```
```
if ( line.equalsIgnoreCase("exit") ) {
        System.exit(0);
    } else if ( line.equalsIgnoreCase("generate") ) {
        generate();
    } else if ( line.equalsIgnoreCase("paint") ) {
        paint();
    } else if ( line.equalsIgnoreCase("dispose") ) {
        miro.end();
    } else if ( line.equalsIgnoreCase("help") ) {
        help();
    } else if ( line.equalsIgnoreCase("display") ) {
        System.out.print(toString());
    } else {
        System.out.println("Sorry, I don't recognize: " + line);
    3
    interpreter();
}
private static void help() {
    System.out.println("HELP - display this help menu to " +
            "the standard output stream");
    System.out.println("DISPLAY - display the Algae L-System, " +
            "vocabulary and axiom and productions");
    System.out.println("GENERATE - generate some number of generations, " +
            "as specified by the user");
    System.out.println("PAINT - paint a rendering of some generation " +
            "of the Algae system");
    System.out.println("DISPOSE - get rid of the canvas " +
            "on which the rendering was painted");
    System.out.println("EXIT - terminate execution of this program");
}
private SPainter miro;
private void paint() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Which player (ALines or ACircles or ASquares)? ");
    String thePainter = scanner.next();
    System.out.print("Generation number? ");
    int generationNumber = scanner.nextInt();
    miro = new SPainter(500,800);
    ABPainter painter = new ABPainterALines(miro); // arbitrary
    if ( thePainter.equalsIgnoreCase("ALines") ) {
        painter = new ABPainterALines(miro);
    } else if ( thePainter.equalsIgnoreCase("ACircles") ) {
        painter = new ABPainterACircles(miro);
    } else if ( thePainter.equalsIgnoreCase("ASquares") ) {
        painter = new ABPainterASquares(miro);
    }
    painter.paint(generation(generationNumber));
}
```

/\*\*

\* Simply sets up the infrastructure for the program, and gets things started.

```
*/
public static void main(String[] args) {
   SwingUtilities.invokeLater(new Runnable() {
      public void run() {
         new AlgaePainter();
      }
   });
}
```

#### Task 7: Replicate the AlgaePainter demo

}

By now, you should have established, in your world, all of the code needed to replicate the demo of the AlgaePainter that was previously presented. Run the AlgaePainter program and replicate the demo!

#### Task 8: Change the painters and generate another AlgaePainter demo

By analogy with ABPainterALines, ABPainterACircles, and ABPainterASquares, write corresponding Java classes ABPainterALines2, ABPainterACircles2, and ABPainterASquares2 which will generate three images when the programs are run which are interestingly different from those generated by the given programs. Simply study the three given programs, and then write three alternates. Run the AlgaePainter program and interact with it to generate a demo quite like that which was previously presented – but which produces different images!

#### Task 9: Algae System music - discussion and demos

You can render a strings of As and Bs sonically as well as visually. For example, you could bind A and B to a couple of the simple composer's basic sequences, or to a couple of the simple composer's locomotive sequences, or to a couple sequences in of the simple composer's collection of Bach minuet fragments. Rather than rendering A and B visually with the simple painter, this program renders them sonically with the simple composer.

Three sound files were generated when the program AlgaePlayer, to be presented, was run in the manner shown in the following demo. If you would like to hear them before proceeding to implement the program, you will find them on the Web site associated with this text under the names BasicAlgaeSequence1.mp3, LocomotiveAlgaeSequence1.mp3, and BachAlgaeSequence1.mp3.

Please don't be expecting anything particularly good from a musical point of view. Rather, expect something suggestive of the sort of thing that is involved in algorithmic composition. To hear something good from a tonal music perspective, requires very different algorithms grounded in theories of human perception and traditions of western music. Still, this program does illustrate the basic idea behind algorithmic composition. Find an algorithm, render a sonic stream in a manner consistent with the algorithm, quite likely by means of other algorithms. In this case, the basic algorithm is an L-System generator, and the sole additional algorithmic infusion merely arranges for stepwise motion of the sequences to which the vocabulary symbols (A and B) produce.

## Standard IO demo of AlgaePlayer

```
run:
Algae>>> help
HELP - display this help menu to the standard output stream
DISPLAY - display the Algae L-System, vocabulary and axiom and productions
GENERATE - generate some number of generations, as specified by the user
PLAY - paint a sonic rendering of some generation of the Algae system
DISPOSE - get rid of the canvas on which the rendering was painted
EXIT - terminate execution of this program
Algae>>> display
Name = Algae
Axiom = A
Productions ...
A --> A B
B --> A
Algae>>> generate
How many generations? 5
generation 0 = A
generation 1 = A B
generation 2 = A B A
generation 3 = A B A A B
generation 4 = A B A A B A B A
generation 5 = A B A A B A B A A B A A B
Algae>>> play
Which player (Bach1 or Locomotion1 or Basics1)? Basics1
Generation number? 5
line = A B A A B A B A A B A A B
Stepwise motion = LRLRLRRRLLRL
(C, 1/2) \setminus (B, 1/2) / (C, 1)
(B,1/2) (B,1/2) (B,1/2) (B,1/2)
/ (C,1/2) \ (B,1/2) / (C,1)
(B,1/2) (A,1/2) / (B,1)
/ (C,1/2) (C,1/2) (C,1/2) (C,1/2)
(B,1/2) (A,1/2) / (B,1)
/ (C,1/2) (C,1/2) (C,1/2) (C,1/2)
/ (D,1/2) \ (C,1/2) / (D,1)
/ (E,1/2) \ (D,1/2) / (E,1)
\ (D,1/2) (D,1/2) (D,1/2) (D,1/2)
(C, 1/2) \setminus (B, 1/2) / (C, 1)
/ (D, 1/2) \setminus (C, 1/2) / (D, 1)
(C,1/2) (C,1/2) (C,1/2) (C,1/2)
(C, 3)
Score file name, without extension, from /Users/blue/ directory? CS1Files/midi/Basics1
Score saved as file /Users/blue/CS1Files/midi/Basics1.midi
Algae>>> play
Which player (Bach1 or Locomotion1 or Basics1)? Locomotion1
Generation number? 5
line = A B A A B A B A A B A A B
Stepwise motion = RLRLLLRLRRLR
(C,3/4) / (D,1/4) / (E,3/4) / (F,1/4) \setminus (E,3/4) \setminus (D,1/4) \setminus (C,1)
/ (A, 1/2) \setminus (G, 1/2) \setminus (F, 1/2) \setminus (E, 1/2) \setminus (D, 2)
```

```
(C,3/4) / (D,1/4) / (E,3/4) / (F,1/4) (E,3/4) (D,1/4) (C,1)
/ (D,3/4) / (E,1/4) / (F,3/4) / (G,1/4) \setminus (F,3/4) \setminus (E,1/4) \setminus (D,1)
/ (G, 1/2) \setminus (F, 1/2) \setminus (E, 1/2) \setminus (D, 1/2) \setminus (C, 2)
(B,3/4) / (C,1/4) / (D,3/4) / (E,1/4) (D,3/4) (C,1/4) (B,1)
/ (E,1/2) \ (D,1/2) \ (C,1/2) \ (B,1/2) \ (A,2)
/ (B,3/4) / (C,1/4) / (D,3/4) / (E,1/4) \ (D,3/4) \ (C,1/4) \ (B,1)
(A,3/4) / (B,1/4) / (C,3/4) / (D,1/4) (C,3/4) (B,1/4) (A,1)
/ (F,1/2) \ (E,1/2) \ (D,1/2) \ (C,1/2) \ (B,2)
/ (C,3/4) / (D,1/4) / (E,3/4) / (F,1/4) \ (E,3/4) \ (D,1/4) \ (C,1)
(B,3/4) / (C,1/4) / (D,3/4) / (E,1/4) (D,3/4) (C,1/4) (B,1)
/ (G,1/2) \ (F,1/2) \ (E,1/2) \ (D,1/2) \ (C,2)
(C,3)
Score file name, without extension, from /Users/blue/ directory? CS1Files/midi/Locomotion1
Score saved as file /Users/blue/CS1Files/midi/Locomotion1.midi
Algae>>> play
Which player (Bach1 or Locomotion1 or Basics1)? Bach1
Generation number? 5
line = A B A A B A B A A B A A B
Stepwise motion = LRRLRLLRRRLL
(C,1) \setminus (F,1/2) / (G,1/2) / (A,1/2) / (B,1/2)
(B,1) / (F,1/2) \setminus (E,1/2) / (F,1)
(C,1) \setminus (F,1/2) / (G,1/2) / (A,1/2) / (B,1/2)
/ (D,1) \ (G,1/2) / (A,1/2) / (B,1/2) / (C,1/2)
(C,1) / (G,1/2) \setminus (F,1/2) / (G,1)
(D,1) \setminus (G,1/2) / (A,1/2) / (B,1/2) / (C,1/2)
(C,1) / (G,1/2) \setminus (F,1/2) / (G,1)
(B,1) (E,1/2) / (F,1/2) / (G,1/2) / (A,1/2)
/ (C,1) \ (F,1/2) / (G,1/2) / (A,1/2) / (B,1/2)
/ (D,1) / (A,1/2) \setminus (G,1/2) / (A,1)
(E,1) (A,1/2) / (B,1/2) / (C,1/2) / (D,1/2)
(D,1) \setminus (G,1/2) / (A,1/2) / (B,1/2) / (C,1/2)
(C,1) / (G,1/2) \setminus (F,1/2) / (G,1)
\ (C,3)
Score file name, without extention, from /Users/blue/ directory? CS1Files/midi/Bach1
Score saved as file /Users/blue/CS1Files/midi/Bach1.midi
Algae>>> exit
BUILD SUCCESSFUL (total time: 2 minutes 42 seconds)
```

#### Task 10: Create some players to sonically render some Algae L-System strings

An abstract class is presented which contains the basic logic for rendering strings of As and Bs. All it lacks is any notion of what it means to render and A or a B. Three different classes are then presented which inherit the functionality of the abstract class, but which complete it by saying just what it means to render an A and to render a B. Simply enter the four classes, in appropriately named Java Class files, in a package called **players**.

```
/*
 * This abstract class serves to render melodies based on strings of As and Bs.
 * The renderer is coded in such a way that the rendering of each symbol is
 * left left unspecified. To complete the renderer, the methods thingA and
 * thingB must be specified. That is the job of the classes which extend this
 * abstract class.
 */
package players;
import composer.SComposer;
import java.util.Random;
import java.util.Scanner;
public abstract class ABPlayer {
    // The simple composer is the sole instance variable for this class.
   protected SComposer composer;
    /**
     * Create an ABPlayer, a performer which bases its work on strings of As and
     * Bs. It is basically a simple composer (SComposer) which processes the As
     * and the Bs in the string, one at a time, by somehow sonically rendering
     * them.
     * Oparam c is the work horse painter
     */
    public ABPlayer(SComposer c) {
        composer = c;
    }
    /**
     * Play a melody by processing the given string of A and B symbols.
     * Oparam line is a string of As and Bs, presumably generated by some
     * L-System. But there is a "twist". An "add on" imposes stepwise
     * motion, for the most part, on the melodic fragments to which the
     * A and B are bound.
     */
    public void play(String line) {
        composer.beginScore();
        composer.text();
        String motionLine = motionLine(line) + "S";
        int x = 0;
        Scanner symbolString = new Scanner(line);
        while ( symbolString.hasNext() ) {
            String symbol = symbolString.next();
            if ( symbol.equals("A") ) {
                thingA();
            } else if ( symbol.equals("B") ) {
                thingB();
```

```
}
        if ( symbolString.hasNext() ) {
            String direction = motionLine.substring(x,x+1);
            changePitch(composer,direction);
            x = x + 1;
        }
    }
    composer.mms_31_JSB_M1();
    composer.untext();
    composer.saveScore();
}
private String motionLine(String line) {
    System.out.println("line = " + line);
    int lineLength = lineLength(line);
    int motionLineLength = lineLength-1;
    String orderedMotionLine = orderedMotionLine(motionLineLength);
    String unorderedMotionLine = unorderedMotionLine(orderedMotionLine);
    System.out.println("Stepwise motion = " + unorderedMotionLine);
    return unorderedMotionLine;
}
private int lineLength(String line) {
    if ( line.equals("") ) {
        return 0;
    } else if ( line.substring(0,1).equals(" ") ) {
        return lineLength(line.substring(1));
    } else {
        return 1 + lineLength(line.substring(1));
    }
}
private String orderedMotionLine(int motionLineLength) {
    if ( motionLineLength == 0 ) {
        return "";
    } else if ( motionLineLength == 1 ) {
        return "S";
    } else {
        return "RL" + orderedMotionLine(motionLineLength-2);
    }
}
private String unorderedMotionLine(String orderedMotionLine) {
    if ( orderedMotionLine.length() < 2 ) {</pre>
        return orderedMotionLine;
    } else {
        String element = pick(orderedMotionLine);
        String remainder = remove(element, orderedMotionLine);
        return element + unorderedMotionLine(remainder);
    }
}
private Random generator = new Random();
```

```
146
```

```
private String pick(String orderedMotionLine) {
    int rn = generator.nextInt(orderedMotionLine.length());
    return orderedMotionLine.substring(rn,rn+1);
}
private String remove(String element, String bag) {
    int position = bag.indexOf(element);
    return bag.substring(0,position) + bag.substring(position+1);
}
private void changePitch(SComposer composer, String direction) {
    if ( direction.equalsIgnoreCase("R") ) {
        composer.rp();
    } else if ( direction.equalsIgnoreCase("L") ) {
        composer.lp();
    } else if ( direction.equalsIgnoreCase("S") ) {
    }
}
/**
 * Reference to an encoding of what it means to render the A symbol
 * sonically.
 */
public abstract void thingA();
/**
 * Reference to an encoding of what it means to render the B symbol
 * sonically.
 */
public abstract void thingB();
```

```
}
```

The ABPlayerBasics Java Class file

/\*
 \* Subclass of the abstract ABPlayer class which renders simple melodies by
 \* binding A and B each to a simple 4 beat sequence.
 \*/
package players;
import composer.SComposer;
public class ABPlayerBasics extends ABPlayer {
 /\*\*
 \* Create a specialization of an ABPlayer which renders simple melodic
 \* melodies by gluing together simple 4 beat fragments.

\* Cparam composer is the work horse composer

```
*/
public ABPlayerBasics(SComposer composer) {
    super(composer);
}
/**
 * Play a simple 4 beat sequence.
 */
public void thingA() {
    composer.s2(); composer.mms6(); composer.x2();
}
/**
 * Play a simple 4 beat sequence.
 */
public void thingB() {
    composer.s2(); composer.mms3(); composer.x2();
}
```

The ABPlayerLocomotion Java Class file

}

```
/*
 * Subclass of the abstract ABPlayer class which renders simple melodies by
 * binding A and B each to a simple locomotive sequences.
 */
package players;
import composer.SComposer;
public class ABPlayerLocomotion extends ABPlayer {
    /**
     * Create a specialization of an ABPlayer which renders simple melodic
     * melodies by gluing together simple locomotive fragments.
     * Cparam composer is the work horse composer
     */
   public ABPlayerLocomotion(SComposer composer) {
        super(composer);
    }
    /**
     * Play a simple 4 locomotive sequence.
     */
   public void thingA() {
        composer.s2(); composer.mms_87_StaggerUpDown(); composer.x2();
    }
    /**
     * Play a simple 4 locomotive sequence.
```

```
*/
public void thingB() {
    composer.s2(); composer.mms_85_StrollDown(); composer.x2();
}
```

The ABPlayerBach Java Class file

```
/*
 * Subclass of the abstract ABPlayer class which renders minuet like melodies
 * by binding A and B each to a 3 beat sequence lifted from a Bach minuet.
 */
package players;
import composer.SComposer;
public class ABPlayerBach extends ABPlayer {
    /**
     * Create a specialization of an ABPlayer which renders a minuet like
     * melody by sequencing fragments stolen from Bach minuets.
     * Cparam composer is the work horse composer
     */
   public ABPlayerBach(SComposer composer) {
        super(composer);
    }
    /**
     * Play a 5 note 3 beat sequence lifted from a Bach minuet.
     */
   public void thingA() {
        composer.mms_35_JSB_M9();
    }
    /**
     * Play a 4 note 3 beat sequence lifted from a Bach minuet.
     */
   public void thingB() {
        composer.mms_34_JSB_M7();
    }
}
```

### Task 11: Establishment of the AlgaePlayer Java Main Class file

The following program makes use of the refinements of the ABPlayer class in order to play melodic sequences based on the Algae strings. It conditionally determines which Algea string to render, and which of the three renderers thus far established to employ. Study it. Then enter it.

```
/*
```

```
* This abstract class serves to render melodies based on strings of As and Bs.
 * The renderer is coded in such a way that the rendering of each symbol is
 * left left unspecified. To complete the renderer, the methods thingA and
 * thingB must be specified. That is the job of the classes which extend this
 * abstract class.
 */
package players;
import composer.SComposer;
import java.util.Random;
import java.util.Scanner;
public abstract class ABPlayer {
    // The simple composer is the sole instance variable for this class.
   protected SComposer composer;
    /**
     * Create an ABPlayer, a performer which bases its work on strings of As and
     * Bs. It is basically a simple composer (SComposer) which processes the As
     * and the Bs in the string, one at a time, by somehow sonically rendering
     * them.
     * Oparam c is the work horse painter
     */
    public ABPlayer(SComposer c) {
        composer = c;
    }
    /**
     * Play a melody by processing the given string of A and B symbols.
     * Oparam line is a string of As and Bs, presumably generated by some
     * L-System. But there is a "twist". An "add on" imposes stepwise
     * motion, for the most part, on the melodic fragments to which the
     * A and B are bound.
     */
    public void play(String line) {
        composer.beginScore();
        composer.text();
        String motionLine = motionLine(line) + "S";
        int x = 0;
        Scanner symbolString = new Scanner(line);
        while ( symbolString.hasNext() ) {
            String symbol = symbolString.next();
            if ( symbol.equals("A") ) {
                thingA();
            } else if ( symbol.equals("B") ) {
                thingB();
```

```
}
        if ( symbolString.hasNext() ) {
            String direction = motionLine.substring(x,x+1);
            changePitch(composer,direction);
            x = x + 1;
        }
    }
    composer.mms_31_JSB_M1();
    composer.untext();
    composer.saveScore();
}
private String motionLine(String line) {
    System.out.println("line = " + line);
    int lineLength = lineLength(line);
    int motionLineLength = lineLength-1;
    String orderedMotionLine = orderedMotionLine(motionLineLength);
    String unorderedMotionLine = unorderedMotionLine(orderedMotionLine);
    System.out.println("Stepwise motion = " + unorderedMotionLine);
    return unorderedMotionLine;
}
private int lineLength(String line) {
    if ( line.equals("") ) {
        return 0;
    } else if ( line.substring(0,1).equals(" ") ) {
        return lineLength(line.substring(1));
    } else {
        return 1 + lineLength(line.substring(1));
    }
}
private String orderedMotionLine(int motionLineLength) {
    if ( motionLineLength == 0 ) {
        return "";
    } else if ( motionLineLength == 1 ) {
        return "S";
    } else {
        return "RL" + orderedMotionLine(motionLineLength-2);
    }
}
private String unorderedMotionLine(String orderedMotionLine) {
    if ( orderedMotionLine.length() < 2 ) {</pre>
        return orderedMotionLine;
    } else {
        String element = pick(orderedMotionLine);
        String remainder = remove(element, orderedMotionLine);
        return element + unorderedMotionLine(remainder);
    }
}
private Random generator = new Random();
```

```
151
```

```
private String pick(String orderedMotionLine) {
    int rn = generator.nextInt(orderedMotionLine.length());
    return orderedMotionLine.substring(rn,rn+1);
}
private String remove(String element, String bag) {
    int position = bag.indexOf(element);
    return bag.substring(0,position) + bag.substring(position+1);
}
private void changePitch(SComposer composer, String direction) {
    if ( direction.equalsIgnoreCase("R") ) {
        composer.rp();
    } else if ( direction.equalsIgnoreCase("L") ) {
        composer.lp();
    } else if ( direction.equalsIgnoreCase("S") ) {
    }
}
/**
 * Reference to an encoding of what it means to render the A symbol
 * sonically.
 */
public abstract void thingA();
/**
 * Reference to an encoding of what it means to render the B symbol
 * sonically.
 */
public abstract void thingB();
```

#### Task 12: Replicate the AlgaePlayer demo

}

By now, you should have established, in your world, all of the code needed to replicate the demo of the AlgaePlayer that was previously presented. Run the AlgaePlayer program and replicate the demo!

## Task 13: Change the players and generate a AlgaePlayer demo

By analogy with ABPlayerBasics, ABPlayerLocomotions, and ABPlayerBach, write corresponding Java classes ABPlayerBasics2, ABPlayerLocomotions2, and ABPlayerBach2 which will generate three melodic lines when the programs are run which are interestingly different from those generated by the given programs. Simply study the three given programs, and then write three alternates. Run the AlgaePlayer program and interact with it to generate a demo quite like that which was previously presented – but which produces different melodic lines!

## Task 14: Generating Sierpinski Triangle images

The Sierpinski Triangle is an example of a self similar object, or **fractal**. Fractals can be rendered at any level of complexity. Six renderings of the Sierpinski Triangle are presented in the following diagram:



These images have been generated by means of the previously presented L-System and a Turtle Geometry rendering scheme. At this point, you might like to Google "Turtle Geometry" and get acquainted with the basics of the system. The simple painter of the NPW was inspired by Turtle Geometry. For many purposes, you can forget that there is any distinction between the two. However you wish to conceive of the screen creature, this is the set of action bindings that is used to produce Sierpinski Triangles:

- 1. F: draw a line in the forward direction for some distance  $\delta$
- 2. G: draw a line in the forward direction for some distance  $\delta$
- 3. +: turn 120 degrees to the left
- 4. -: turn 120 degrees to the right

The six images shown were produced by means of running the SierpinskiTrianglePainter program as shown in the subsequent demo. (Each image was clipped from the canvas and subsequently merged into a composite image by hand.

After studying the demo in relation to the images, please study the accompanying code that forms an interpreter for deriving generations of the Sierpinski L-System and rendering corresponding Sierpinski Triangles. Then, type it in, and run it, mimicking the demo.

(You may have guessed that a variant of this program was used to create the connect-the-dots opportunity which appears on the front cover of this manual.)

## Standard IO demo of SierpinskiTrianglePainter

```
run:
SierpinskiTriangle>>> help
HELP - display this help menu to the standard output stream
DISPLAY - display the L-System, vocabulary and axiom and productions
GENERATE - generate some number of generations, as specified by the user
PAINT - paint a rendering of some generation of the system
SLOW - slow down the painting, if it is fast
FAST - slow down the painting, if it is slow
DISPOSE - get rid of the canvas on which the rendering was painted
EXIT - terminate execution of this program
SierpinskiTriangle>>> display
Name = SierpinskiTriangle
Axiom = F - G - G
Productions ...
F \longrightarrow F - G + F + G - F
G --> G G
- --> -
+ --> +
SierpinskiTriangle>>> generate
How many generations? 4
generation 0 = F - G - G
generation 1 = F - G + F + G - F - G G - G G
generation 2 = F - G + F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F - G G G G
  - G G G G
generation 3 = F - G + F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F - G G G G
  + F - G + F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F + G G G G - F - G +
  F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F - G G G G G G G G G - G G G G G
 GGG
generation 4 = F - G + F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F - G G G G
  + F - G + F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F + G G G G - F - G +
 F + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F - G G G G G G G G + F - G + F
  + G - F - G G + F - G + F + G - F + G G - F - G + F + G - F - G G G G + F - G + F + G - F -
 G G + F - G + F + G - F + G G - F - G + F + G - F + G G G G - F - G + F + G - F - G G + F -
  G + F + G - F + G G - F - G + F + G - F + G G G G G G G G - F - G + F + G - F - G G + F - G
  + F + G - F + G G - F - G + F + G - F - G G G G + F - G + F + G - F - G G + F - G + F + G -
 F + G G - F - G + F + G - F + G G G G - F - G + F + G - F - G G + F - G + F + G - F + G G -
 SierpinskiTriangle>>> paint
Generation number? 0
SierpinskiTriangle>>> dispose
SierpinskiTriangle>>> paint
Generation number? 1
SierpinskiTriangle>>> dispose
SierpinskiTriangle>>> paint
Generation number? 2
SierpinskiTriangle>>> dispose
SierpinskiTriangle>>> paint
Generation number? 3
SierpinskiTriangle>>> dispose
SierpinskiTriangle>>> paint
```

Generation number? 4 SierpinskiTriangle>>> dispose SierpinskiTriangle>>> paint Generation number? 5 SierpinskiTriangle>>> dispose SierpinskiTriangle>>> exit BUILD SUCCESSFUL (total time: 1 minute 28 seconds)

#### SierpinskiTrianglePainter program

/\*

\* This program can generate and process generations of the Sierpinski Triangle \* L-System, where processing amounts to performing graphical renderings of the \* strings of symbols A and B of the system. The program takes the form of an \* interpreter. See the comment prefacing the constructor for additional details. \*/ package lsystem; import java.util.ArrayList; import java.util.List;

import java.util.Scanner; import javax.swing.SwingUtilities; import painter.SPainter; import painters.STPainter; import painters.STPainterStar; public class SierpinskiTrianglePainter extends LSystem { /\*\* \* Create an SierpinskiTrianglePainter object, which is so tightly coupled to \* its behavior as an interpreter that its private interpreter method is \* called from this constructor. The program can represent the Algae \* L-System, and can derive successive generations of strings within \* the system. Specifically, the interpreter can process the following \* commands: <br> \* \* HELP - display this help menu to the standard output stream<br> \* \* DISPLAY - display the L-System, vocabulary and axiom and productions<br> \* \* GENERATE - generate some number of generations, as specified by the user<br/> \* \* PAINT - paint a rendering of some generation of the system<br> \* \* SLOW - slow down the painting, if it is fast<br> \* \* FAST - speed up the painting, if it is slow<br> \* \* DISPOSE - get rid of the canvas on which the rendering was painted<br> \* \* EXIT - terminate execution of this program<br> \*/ public SierpinskiTrianglePainter() { super("SierpinskiTriangle"); axiom = "F - G - G"; productions = productions(); interpreter(); }

```
private List<Production> productions() {
    Production p1 = new Production("F","F - G + F + G - F");
    Production p2 = new Production("G","G G");
    Production p3 = new Production("-","-");
    Production p4 = new Production("+","+");
    ArrayList<Production> productions = new ArrayList<>();
    productions.add(p1);
    productions.add(p2);
    productions.add(p3);
    productions.add(p4);
    return productions;
}
private void interpreter() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("SierpinskiTriangle>>> ");
    String line = scanner.next();
    if ( line.equalsIgnoreCase("exit") ) {
        System.exit(0);
    } else if ( line.equalsIgnoreCase("generate") ) {
        generate();
    } else if ( line.equalsIgnoreCase("paint") ) {
        paint();
    } else if ( line.equalsIgnoreCase("slow") ) {
        waitTime =500;
    } else if ( line.equalsIgnoreCase("fast") ) {
        waitTime = 0;
    } else if ( line.equalsIgnoreCase("dispose") ) {
        miro.end();
    } else if ( line.equalsIgnoreCase("help") ) {
        help();
    } else if ( line.equalsIgnoreCase("display") ) {
        System.out.print(toString());
    } else {
        System.out.println("Sorry, I don't recognize: " + line);
    3
    interpreter();
}
private static void help() {
    System.out.println("HELP - display this help menu to " +
            "the standard output stream");
    System.out.println("DISPLAY - display the L-System, " +
            "vocabulary and axiom and productions");
    System.out.println("GENERATE - generate some number of generations, " +
            "as specified by the user");
    System.out.println("PAINT - paint a rendering of some generation of the system");
    System.out.println("SLOW - slow down the painting, if it is fast");
    System.out.println("FAST - slow down the painting, if it is slow");
    System.out.println("DISPOSE - get rid of the canvas " +
            "on which the rendering was painted");
    System.out.println("EXIT - terminate execution of this program");
```

```
private SPainter miro;
private int waitTime = 0;
private void paint() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Generation number? ");
    int generationNumber = scanner.nextInt();
    miro = new SPainter(900,900);
    STPainterStar painter = new STPainterStar(miro);
    painter.paint(generationNumber,generation(generationNumber),waitTime);
}
/**
 * Simply sets up the infrastructure for the program, and gets things started.
 * Oparam args is not used
 */
public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
     public void run() {
        new SierpinskiTrianglePainter();
     }
  });
}
```

#### Task 15: Cantor Dust painting

}

}

By analogy with the AlgaePainter program, write a CantorDustPainter program. Notice that the Cantor L-System has a vocabulary of just two symbols, A and B, so that you can make use of the ABPainter program as is. By analogy with the ABPainterALines program write an ABPainterCDLines program. By analogy with the ABPainterACircles program write an ABPainterCDCircles program. By analogy with the ABPainterASquares program write an ABPainterCDSquares program. Then, run the CantorDustPainter program to generate three images, one based on lines, one based on circles, and one based on squares.

#### Task 16: Cantor Dust playing

By analogy with the AlgaePlayer program, write a CantorDustPlayer program. Again, notice that the Cantor L-System has a vocabulary of just two symbols, A and B, so that you can make use of the ABPainter program as is. By analogy with the ABPlayerBasics program write an ABPainterCDBasics program. By analogy with the ABPainterALocomotives program write an ABPainterCDLocomotives program. By analogy with the ABPainterABach program write an ABPainterCDBach program. Then, run the CantorDustPainter program to generate three images, one based on the simple composer's basics, one based on the simple composer's locomotives, and one based on the simple composer's stash of Bach sequences.

## Task 17: Generate the *Javadocs* for your CS1 project

Generate the *Javadocs* for your CS1 project. Google how to do it, if you should need to. (It is very easy!) Take a good look at the Web documents that are generated for your L-System programs, in particular.

#### Task 18: Incorporate artifacts into your site

Incorporate code, images, and sound files into your work site as you see fit.

## Task 19: Reflection

Think for a little while about this lab and your engagement with it. What did you learn that is conceptually significant? What did you learn that is technologically useful? What is your most salient thought about the lab and your engagement with it?

## C. A. R. Hoare on COMPUTER SCIENCE

What is the central core of the subject [computer science]? What is it that distinguishes it from the separate subjects with which it is related? What is the linking thread which gathers these disparate branches into a single discipline. My answer to these questions is simple - it is the art of programming a computer. It is the art of designing efficient and elegant methods of getting a computer to solve problems, theoretical or practical, small or large, simple or complex. It is the art of translating this design into an effective and accurate computer program.

#### Perspective

This text, and the course to which it contributes, are centered on *programming in the small*. If you methodically worked through all of the labs, and if you dedicated yourself to completing the programming assignments, you probably have a pretty good idea of how one person goes about writing a relatively small, relatively simple computer program. But you should know that this is just a part of the field of computer science. There are many other parts, as well. One of these other parts can be characterized as *programming in the large*. This pertains to writing large software systems. Teams of programmers are involved, and systems of *modules* are designed to fit together in elegant ways, ways that ideally afford ease of maintenance, modification, and extension. The CS2 course is designed to refine and expand your knowledge of programming in the small. Other courses in the curriculum explore ideas and issues surrounding programming in the large. Programming in the small vs programming in the large is just one of many dichotomous ways to think about aspects of computer science. The field is huge, packed with powerful ideas, and constantly pregnant with respect to practical applications.

Although computer science knowledge can certainly lead to attractive careers in terms of monetary reward and professional satisfaction, it is perhaps the fact that computer science is associated with a way of thinking that recommends it more than any other for a position in the college curriculum.

 $\implies$  Why study computer science? To learn a powerful way of thinking!

#### Summary of the Most Salient Elements of the Laboratories

- 1. Lab 1: Hello World! Hello You! ▷ Java ◇ Integrated Development Environment (IntelliJ) ◇ templates ◇ program execution ◇ program IO ◇ widgets ◇ appreciating the importance of tending to detail at the level of the token
- 2. Lab 2: Hello Painter! Hello Composer! ▷ object creation ◇ object use ◇ library files ◇ microworlds ◇ Nonrepresentational Painting World (NPW) ◇ Modular Melody World (MMW) ◇ graphics programming ◇ sonic programming
- 3. Lab 3: Establishing a CS1 Work Site > Emacs <> HTML <> CSS <> Web development <> file structure <> source files <> graphics files
- 4. Lab 4: Expressions and Shapes World Problem Solving ▷ fully parenthesized expression ◊ English/Java representations of arithmetic expressions ◊ simple geometric/algebraic problem solving ◊ Crypto problem solving ◊ problem solving with simple shape objects ◊ inscribing circles ◊ circumscribing circles

- 5. Lab 5: An Interpreter Featuring Loop Forever and Selection  $\triangleright$  interpreters  $\diamond$  loop forever  $\diamond$  break statement  $\diamond$  multiway conditional statement  $\diamond$  random number generation  $\diamond$  dialog box  $\diamond$  string comparison
- 6. Lab 6: Functions and Commands  $\triangleright$  define functions  $\diamond$  define commands  $\diamond$  apply the principle of stepwise refinement  $\diamond$  work with IntelliJ to effectively engage in stepwise refinement  $\diamond$  program by modifying an extant program  $\diamond$  while statement  $\diamond$  conditional execution  $\diamond$  generate random colors
- 7. Lab 7: String Thing ▷ get acquainted with length, indexOf, two versions of the substring function, and the equalsIgnoreCase function indfully perform abstraction by writing methods to generalize on specific computations ◊ attend to detail with respect to positions of items in a sequence
- 8. Lab 8: Array Play  $\triangleright$  array declaration  $\diamond$  array creation  $\diamond$  array indexing  $\diamond$  file processing  $\diamond$  exceptions  $\diamond$  program by analogy  $\diamond$  data file creation
- 9. Lab 9: Simple List Processing ▷ ArrayList ◊ generics ◊ list methods: add, set get, size ◊ comparing/contrasting arrays and lists
- 10. Lab 10: Establishing and Using Classes ▷ refining pseudocode ◇ "mechanical" translation of code ◇ data control loop vs counter control loop ◇ generating and refining stubs
- 11. Lab 11: Modeling Objects with Classes  $\triangleright$  class definition  $\diamond$  establish instance variables  $\diamond$  define constructors  $\diamond$  define methods  $\diamond$  establish an interface  $\diamond$  implement an interface
- 12. Lab 12: Grapheme to Color Synesthesia ▷ parallel arrays ◊ sequential search ◊ drawing text ◊ mapping ◊ synesthetic simulation
- 13. Lab 13: Chromesthesia ▷ incremental program development ◊ mapping pitch classes to sounds ◊ integrate graphic processing with sonic processing ◊ simulate the experience of a chromesthete ◊ process arrays of objects ◊ symbolic processing ◊ scanning/interpretation
- 14. Lab 14: Fun with Fractals  $\triangleright$  self-similarity  $\diamond$  fractals  $\diamond$  L-Systems  $\diamond$  programming generative algorithms  $\diamond$  abstract classes  $\diamond$  algorithmic composition  $\diamond$  Turtle Geometry

# 30 Appendix 1: Nonrepresentational Painting World (NPW)

This appendix presents a partial specification of the Nonrepresentational Painting World, or NPW, which features (1) two-dimensional shape objects, and (2) simple painter objects that can render the shapes in various ways on a 2D area called the *canvas*. This environment affords opportunities to explore ideas associated with the world of nonrepresentational art and the work of graphic design. Only the most basic elements of functionality are presented here.

## Interpreting Entries in the Appendices

Each entry in this appendix is made up of terminal symbols and *non-terminal* symbols. Terminal symbols are those which require no modification by you – type them exactly as shown. Non-terminal symbols stand for something of the specified type, and should be replaced with something appropriate to the type specified. For example, in order to use the version of paint which takes an *SCircle* argument:

#### SPainter.paint(SCircle)

you must replace *SPainter* and *SCircle* with instances of the appropriate types.

Methods and constructors which return values have the types of those values shown after the  $\rightarrow$  symbol. For example, you can see below that the constructors for SPainter return instances of that type.

#### **SPainter Functionality**

A simple painter is a screen creature that is bound to a 2D graphics area, called its *canvas*. The painter can move about the canvas and *paint* (fill in) or *draw* (outline) shapes of various sorts on the canvas, provided they are within reach. A *painter* is modeled in terms of several properties, most notably its *location*, and its *heading*. To do its rendering tasks, the painter possesses a *brush*, which has a *width*, and which can render shapes in virtually any *color*. There are a number of constructors associated with the SPainter class, including:

- new SPainter(String, int, int) → SPainter
   returns a simple painter in a canvas, labelled by the given string, the width and height of which are equal to the two integer values
- new SPainter(int, int)  $\longrightarrow$  SPainter

returns a simple painter in an undecorated (no close or shrink boxes), unlabelled black framed canvas, the width and height of which are equal to the two integer values

There is functionality to move the painter forward, backward, right or left, with respect to the direction in which it is facing, to turn it in various ways, to paint and draw certain shapes, to draw text, and to do a number of other things.:

- SPainter.canvasHeight()  $\longrightarrow$  int returns the height of the canvas (including 22 for the top bar, if there is one)
- SPainter.canvasWidth() → int returns the width of the canvas
- SPainter.center()  $\longrightarrow$  Point2D.Double returns the center point on the canvas

- SPainter.cvtDegToRad(double) → double converts the given number of degrees to the corresponding number of radians
- SPainter.dbk(double) the painter moves backward the given distance with respect to its present heading, leaving a trace
- SPainter.dfd(double) the painter moves forward the given distance with respect to its present heading, leaving a trace
- SPainter.draw(SCircle) draws (the border of) the given circle around to painter, using the current paint color
- SPainter.draw(SPolygon) draws (the border of) the given polygon around to painter, using the current paint color
- SPainter.draw(SRectangle) draws (the border of) the given rectangle around to painter, using the current paint color
- SPainter.draw(SSquare) draws (the border of) the given square around to painter, using the current paint color
- SPainter.draw(String) draws the string centered around the painter, horizontally
- SPainter.drawLineTo(Point2D.Double) the painter draws a line from its current position to the position given by the point, in the current color, and remains at the given point
- SPainter.drawLineToI(Point2D.Double) the painter draws a line from its current position to the position given by the point, in the current color, and then returns to the starting point
- SPainter.faceNorth() directs the painter to set its heading to 0 degrees (face north)
- SPainter.frame()  $\longrightarrow$  SRectangle returns the bordering rectangle of the canvas
- SPainter.getBoundingRectangle() → Rectangle2D.Double get the Java 2D rectangle which bounds the canvas
- SPainter.getBoundingSuperRectangle() → Rectangle2D.Double get a Java 2D rectangle which is a bit larger thang the one which bounds the canvas
- SPainter.heading() → double returns the painter's current heading, [0,360)
- SPainter.mbk(double) the painter moves backward the given distance with respect to its present heading, without leaving a trace
- SPainter.mfd(double) the painter moves forward the given distance with respect to its present heading, without leaving a trace
- SPainter.mlt(double) the painter moves to its left the given distance with respect to its present heading, without leaving a trace
  SPainter.move()
  - moves the painter to a random position (location) on the canvas
- SPainter.moveTo(Point2D.Double) sets the position (location) of the painter to the (x,y) coordinates embedded within the given point, with respect to the top-left corner of the canvas
- SPainter.moveToCenter() move the painter to the center of the canvas
- SPainter.moveWithinNeighborhood(int) move the painter to a position within the circle of radius equal to the given number centered at the location of the painter, without changing the heading of the painter
- SPainter.mrt(double) the painter moves to its right the given distance with respect to its present heading, without leaving a trace
- SPainter.paint(SCircle) paint the given circle around to painter, using the current paint color

• SPainter.paint(SPolygon) paint the given polygon around to painter, using the current paint color

- SPainter.paint(SRectangle) paint the given rectangle around to painter, using the current paint color
- SPainter.paint(SSquare) paint the given square around to painter, using the current paint color
- SPainter.paintBrushColor()  $\longrightarrow$  Color returns the color currently on the painter's brush
- SPainter.paintFrame(Color,int) paint a border around the painter's canvas of width equal to the given integer and of color equal to the given color
- SPainter.pause() ask the painter to pause for 1 second
- SPainter.pause(int) ask the painter to pause for the given number of milliseconds second
- SPainter.position() → Point2D.Double return the painter's position (location) on the canvas
- SPainter.random()  $\longrightarrow$  Point2D.Double returns a random point on the canvas
- SPainter.restoreColor() restore the most recently saved paint color
- SPainter.saveColor() save the current paint color
- SPainter.setBrushWidth(int) set the painter's brush width to the given value
- SPainter.setColor(Color) set the color that the painter will draw or paint with to the given color
- SPainter.setFontSize(int) set the font size to the given value
- SPainter.setHeading(int) the painter sets its heading to the given value
- SPainter.setPosition(Point2D.Double) sets the position (location) of the painter to the (x,y) coordinates embedded in the given point, with respect to the top-left corner of the canvas
- SPainter.setRandomColor() set the color that the painter will draw or paint with to a random color
- SPainter.setRandomBlueColor() set the color that the painter will draw or paint with to a random blue color
- SPainter.setRandomGreenColor() set the color that the painter will draw or paint with to a random green color
- SPainter.setRandomRedColor() set the color that the painter will draw or paint with to a random red color
- SPainter.setScreenLocation(int, int) place the painter's frame, the upper left corner, at the screen location given
- SPainter.setVisible(boolean) makes the frame within which the painter is housed appear or disappear on the screen
- SPainter.ta() the painter does an about face
- SPainter.tl() the painter turns 90 degrees to its left
- SPainter.tr() the painter turns 90 degrees to its right

• SPainter.wash() simply white wash the canvas

### **SShapes Functionality**

### **SCircle Functionality**

A *simple circle* is modeled in terms of just one property, its *radius*. There is just one constructor associated with the **SCircle** class, which takes the radius of the new circle as its sole parameter.

new SCircle(double) → SCircle
 return a simple circle, the radius of which is given by the real number

There is functionality for solving simple problems involving circles and for creating images based on circles, including:

- SCircle.area() → double return the area of the circle
- SCircle.circumscribingPolygon(int) → SPolygon return the circumscribing polygon of the given degree of the circle
- SCircle.circumscribingSquare() → SSquare return the circumscribing square of the circle
- SCircle.diameter() → double return the diameter of the circle
- *SCircle*.expand(*double*) increase the radius of the circle by the given number
- SCircle.inscribingPolygon(int) → SPolygon return the inscribing polygon of the given degree of the circle
- SCircle.inscribingSquare() → SSquare return the inscribing square of the circle
- SCircle.perimeter()  $\longrightarrow$  double return the perimeter of the circle
- SCircle.radius() → double return the radius of the circle
- *SCircle*.s2() halve the radius by the circle
- *SCircle.s3()* shrink the radius by the circle by a factor of 3
- SCircle.s5() shrink the radius by the circle by a factor of 5
- SCircle.s7() shrink the radius by the circle by a factor of 7
- SCircle.setRadius(double) set the radius by the circle to the given number
- *SCircle*.shrink(*double*) decrease the radius of the circle by the given number
- SCircle.toString()  $\longrightarrow$  String return a string representation of the circle
- SCircle.x2() double the radius by the circle

- SCircle.x3() expand the radius by the circle by a factor of 3
- SCircle.x5() expand the radius by the circle by a factor of 5
- *SCircle*.x7() expand the radius by the circle by a factor of 7

## **SPolygon Functionality**

A simple polygon is modeled in terms of two properties, its degree (number of sides) and its side (side length). There is just one constructor associated with the SPolygon class, which takes two parameters, the degree and the side of the polygon.

 new SPolygon(int, double) → SPolygon return a simple polygon, the degree of which is given by the integer, and the side of which is given by the real number

There is functionality for solving simple problems involving polygons and for creating images based on polygons, including:

- SPolygon.area() → double return the area of the polygon
- SPolygon.circumscribingCircle() → SCircle return the circumscribing circle of the polygon
- SSquare.dec() decrement the degree of the polygon by 1
- SSquare.decSide() decrement the side of the polygon by 1
- SSquare.degree() → double return the degree of the polygon
- SSquare.inc() increment the degree of the polygon by 1
- SSquare.incSide() increment the side of the polygon by 1
- SPolygon.inscribingCircle() → SCircle return the inscribing circle of the polygon
- SPolygon.perimeter() → double return the perimeter of the polygon
- SSquare.setSide(double) set the side of the polygon to the given number
- SSquare.side() → double return the side (length) of the polygon
- SSquare.toString() → String return a string representation of the polygon

## **SRectangle Functionality**

A simple rectangle is modeled in terms of just two properties, its *height* and its *width*. There is just one constructor associated with the SRectangle class, which takes two parameter, the height and the width of the rectangle.

• new SRectangle(double, double)  $\longrightarrow$  SRectangle return a new simple rectangle, the height of which is given by the first number, and the width of which is given by the second number

There is functionality for solving simple problems involving rectangles and creating images based on rectangles, including:

- SRectangle.area()  $\longrightarrow$  double return the area of the rectangle
- SRectangle.diagonal() → double return the diagonal of the rectangle
- SRectangle.expand(double,double) expand the height of the rectangle by adding the first value to it, and expand the width of the rectangle by adding the second value to it
- SRectangle.height() → double return the height of the rectangle
- SRectangle.perimeter()  $\longrightarrow$  double return the perimeter of the rectangle
- SRectangle.shrink(double,double) shrink the height of the rectangle by subtracting the first value from it, and shrink the width of the rectangle by subtracting the second value from it
- SRectangle.setHeight(double) set the height of the rectangle to the given value
- SRectangle.setWidth(double) set the width of the rectangle to the given value
- SRectangle.toString() → String return a string representation of the rectangle
- SRectangle.width()  $\longrightarrow$  double return the width of the rectangle

## **SS**quare Functionality

A simple square is modeled in terms of just one property, its side (side length). There is just one constructor associated with the SSquare class, which takes the side of the new square as its sole parameter.

• new SSquare(double) → SSquare return a simple square, the side of which is given by the real number

There is functionality for solving simple problems involving squares and creating images based on squares, including:

- SSquare.area() → double return the area of the square
- SSquare.circumscribingCircle() → SCircle return the circumscribing circle of the square
- SSquare.diagonal() → double return the diagonal of the circle
- SSquare.expand(double) increase the radius of the circle by the given number
- SSquare.inscribingCircle() → SCircle return the inscribing circle of the square

- SSquare.perimeter()  $\longrightarrow$  double return the perimeter of the square
- *SSquare*.s2() halve the radius by the square
- SSquare.s3() shrink the side by the square by a factor of 3
- SSquare.s5() shrink the side by the square by a factor of 5
- SSquare.s7() shrink the side by the square by a factor of 7
- SSquare.setSide(double) set the sides of the square to the given number
- SSquare.shrink(double) decrease the side of the square by the given number
- SSquare.side() → double return the side (length) of the square
- $SSquare.toString() \longrightarrow String$ return a string representation of the square
- SSquare.x2() double the side by the square
- SSquare.x3() expand the side by the square by a factor of 3
- SSquare.x5() expand the side by the square by a factor of 5
- SSquare.x7() expand the side by the square by a factor of 7

# 31 Appendix 2: Modular Melody World (MMW)

This appendix presents a partial specification for the Modular Melody World, or MMW, which features musical *note* objects and music *composer* objects, both of which can be used to render melodic sequences of notes. Some, not all, elements of this microworld's functionality is presented here.

## Interpreting Entries in the Appendices

Each entry in this appendix is made up of terminal symbols and non-terminal symbols. Terminal symbols are those which require no modification by you – type them exactly as shown. Non-terminal symbols stand for something of the specified type, and should be replaced with something appropriate to the type specified. For example, in order to use the changeVolume method, which takes a *String* argument:

SNote.changeVolume(String)

you must replace SNote and String with instances of the appropriate types.

Methods and constructors which return values have the types of those values shown after the  $\rightarrow$  symbol. For example, you can see below that the constructor for SComposer returns an instance of that type.

## **SComposer Functionality**

A simple composer may be thought of as an agent that can help you to craft melodic lines. A composer possesses a *note*, and knows how do a variety of things with it, the most notable of which is to play *modular melodic sequences* with it.

• new SComposer()  $\longrightarrow$  SComposer return a simple composer whose note is instantiated to the default values associated with a simple note, namely: degree=C-MAJOR, degree=1, duration=1, timbre=PIANO, and volume=MEDIUM

There is simple composer's functionality includes:

- SComposer.beginScore() begin a midi score
- SNote.changeVolume(String) sets the volume according to the given number, scale "0" .. "12000"
- SComposer.cp() either raise the pitch or lower the pitch one scale degree
- SComposer.doubleTime() change the tempo, increasing it by a factor of two (functionally the same as s2, but intended for a conceptually different use)
- SComposer.halfTime() change the tempo, decreasing it by a factor of two (functionally the same as x2, but intended for a conceptually different use)
- SComposer.loud() set the volume of the note to 12500
- SComposer.lp() lower the composer's note one scale degree

- SComposer.lp(int) lower the composer's note the given number of scale degrees
- SComposer.maxx() set the volume of the note to 15000
- SComposer.medd() set the volume of the note to 10000
- SComposer.minn() set the volume of the note to 2500
- SComposer.mms1() a simple modular melodic sequence
- SComposer.mms2() a simple modular melodic sequence
- SComposer.mms3() a simple modular melodic sequence
- SComposer.mms4() a simple modular melodic sequence
- SComposer.mms5() a simple modular melodic sequence
- SComposer.mms6() a simple modular melodic sequence
- SComposer.mms7() a simple modular melodic sequence
- SComposer.mms8() a simple modular melodic sequence
- SComposer.mms\_31\_JSB\_M1() a modular melodic sequence consisting of 1 note in 3 beats, the shape of which was lifted from a Bach minuet
- SComposer.mms\_33\_JSB\_M2() a modular melodic sequence consisting of 3 notes in 3 beats, the shape of which was lifted from a Bach minuet
  SComposer.mms\_33\_JSB\_M3()
- a modular melodic sequence consisting of 3 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_33\_JSB\_M4()
- a modular melodic sequence consisting of 3 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_33\_JSB\_M5()
- a modular melodic sequence consisting of 3 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_34\_JSB\_M6()
- a modular melodic sequence consisting of 4 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_34\_JSB\_M7()
- a modular melodic sequence consisting of 4 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_34\_JSB\_M8()
- a modular melodic sequence consisting of 4 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_35\_JSB\_M9()
- a modular melodic sequence consisting of 5 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_35\_JSB\_M10()
- a modular melodic sequence consisting of 5 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_35\_JSB\_M11()
- a modular melodic sequence consisting of 5 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_35\_JSB\_M12()
- a modular melodic sequence consisting of 5 notes in 3 beats, the shape of which was lifted from a Bach minuet • SComposer.mms\_35\_JSB\_M13()
- a modular melodic sequence consisting of 5 notes in 3 beats, the shape of which was lifted from a Bach minuet
  SComposer.mms\_36\_JSB\_M14()
- a modular melodic sequence consisting of 6 notes in 3 beats, the shape of which was lifted from a Bach minuet

- SComposer.mms\_36\_JSB\_M15() a modular melodic sequence consisting of 6 notes in 3 beats, the shape of which was lifted from a Bach minuet
- SComposer.mms\_85\_HillFlat() a modular melodic sequence consisting of 5 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_86\_HillStones() a modular melodic sequence consisting of 6 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_87\_Hill() a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_88\_Hills() a modular melodic sequence consisting of 8 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_86\_PrepJump() a modular melodic sequence consisting of 6 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_87\_Stagger() a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_87\_StaggerUpDown() a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_87\_Stroll() a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_87\_StrollUpDown() a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
  SComposer.mms\_85\_StrollDown()
- a modular melodic sequence consisting of 5 notes in 8 beats whose shape iconically resembles its name • SComposer.mms\_87\_ZagZig()
- a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
- SComposer.mms\_87\_ZigZag() a modular melodic sequence consisting of 7 notes in 8 beats whose shape iconically resembles its name
- SComposer.play() play the composer's note
- SComposer.rest() rest the composer's note
- SComposer.rp() raise the composer's note one scale degree
- SComposer.rp(int) raise the composer's note the given number of scale degrees
- SComposer.saveScore()

assuming that you have begun a score, the midi representation of the score that has been defined by playing the composer's note to the point of this method call will be written to the file that you are asked to specify, provided you specify a valid file name in the proper way (be sure the directory that you are eyeing already exists)

- SComposer.s2() shrink the composer's note by a factor of 2
- SComposer.s3() shrink the composer's note by a factor of 3
- SComposer.s5() shrink the composer's note by a factor of 5
- SComposer.soft() set the volume of the note to 7500
- SComposer.text() play and rest of the note textually (as well as sonically)
- SComposer.untext() stop the playing and resting the note textually (as well as sonically)
- SComposer.x2()

expand the composer's note by a factor of 2

- SComposer.x3() expand the composer's note by a factor of 3
- SComposer.x5() expand the composer's note by a factor of 5

## **SNote Functionality**

A simple note is modeled in terms of more than a dozen properties, the most salient of which are scale and degree, which collectively define the *pitch* of the note, duration which is measured in some number of beats, the timbre of a note, which pertains to the "instrument" through which the note may be rendered, and the volume of the note.

 new SNote() → SNote return a simple note with default values of degree=C-MAJOR, degree=1, duration=1, timbre=PIANO, and volume=MEDIUM

There is functionality for playing the note, resting the note, changing the degree, duration, volume, timbre, and other dimensions of the note. This functionality includes:

- SNote.beginScore() begin a midi score
- SNote.changeVolume(String) sets the volume according to the given number, scale "0" .. "12000"
- SNote.cp() either raise the pitch or lower the pitch one scale degree
- SNote.loud() set the volume of the note to 12500
- SNote.lp() lower the pitch of the note one scale degree
- SNote.lp(int) lower the pitch of the note the given number of scale degrees (within reason)
- SNote.maxx() set the volume of the note to 15000
- SNote.medd() set the volume of the note to 10000
- SNote.minn() set the volume of the note to 2500
- SNote.play() play the note
- SNote.rest() rest the note
- SNote.rp() raise the pitch of the note one scale degree
- SNote.rp(int)

raise the pitch of the note the given number of scale degrees (within reason)

- SNote.saveScore() assuming that you have begun a score, the midi representation of the score that has been assembled to the point of this method call will be written to the file that you are asked to specify, provided you specify a valid file name in the proper way (be sure the directory that you are eyeing already exists)
- SNote.shhh() set the volume of the note to 5000

- SNote.s2() shrink the duration of the note by a factor or 2
- SNote.s3() shrink the duration of the note by a factor or 3
  SNote.s5()

shrink the duration of the note by a factor or 5

- SNote.soft() set the volume of the note to 7500
- SNote.text() play and rest of the note textually (as well as sonically)
- SNote.untext()
  - stop the playing and resting the note textually (as well as sonically)
- SNote.x2() expand the duration of the note by a factor or 2
- SNote.x3() expand the duration of the note by a factor or 3
- SNote.x5()

expand the duration of the note by a factor or  $5\,$ 

## **32** Resources and References

For the most part, the items listed below served to inform my thinking with respect to framing the course and crafting the text. Some of the items were actually referenced in the text.

- 1. Claxton, Guy. Wise-Up: The Challenge of Lifelong Learning. Bloomsbury USA, 2000.
- 2. Dewey, John. Experience and Education. Free Press, 1997.
- 3. Dreyfus, Hubert. What Computers Can't Do. HarperCollins, 1978.
- 4. Dweck, Carol. Mindset: The New Psychology of Success. Ballantine Books, 2007.
- Graci, C. "A Brief Tour of the Learning Sciences Via a Cognitive Toule for Investigating Melodic Phenomena" in Journal of Educational Technology Systems, 2009-2010.
- 6. Hutchins, Edwin. Cognition in the Wild. A Bradford Book, 1995.
- Kafai, Y. "Constructionism" in The Cambridge handbook of the learning sciences, edited by R. Sawyer. Cambridge University Press, 2006.
- 8. Minsky, Marvin. Society of Mind. Simon & Schuster, 1988.
- 9. Langer, Ellen. The Power of Mindful Learning. Da Capo Press, 1998.
- 10. Martinez, Michael. Learning and Cognition: The Design of the Mind. Pearson, 2009.
- 11. Norman, Donald. "Cognitive artifacts" in Designing interaction: Psychology at the human-computer interface, edited by J. Carroll, 1991.
- 12. Papert, Seymour. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, 1993.
- 13. Pea, Roy. "Practices of distributed intelligence and designs for education" in *Distributed Cognitions*, edited by Garviel Salomon, 1997.
- 14. Perkins, D. N. "Person-solo: a distributed view of thinking and learning" in *Distributed Cognitions*, edited by Garviel Salomon, 1997.
- Quintana, C., N. Shin, C. Norris, & E. Soloway. "Learner-centered design" in The Cambridge handbook of the learning sciences, edited by R. Sawyer. Cambridge University Press, 2006
- 16. Salomon, Gavriel. "No distribution without individuals' cognition: a dynamic interactional view" in *Distributed Cognitions*, edited by Garviel Salomon, 1997.
- 17. Schwartz, D., & J. Heiser. "Spatial representations and imagery in learning" in *The Cambridge handbook of the learning sciences*, edited by R. Sawyer. Cambridge University Press, 2006.
- 18. Searle, John. "Brains, Minds, and Programs" in Behavioral and Brain Sciences. 1980.
- Winograd, Terry, and Fernando Flores. Understanding Computers and Cognition: A New Foundation for Design. Addison-Wesley, 1987.
