Annotated Solution Guide for

Thinking in Java

Fourth Edition

TIJ4 published February, 2006

Copyright & Disclaimer

About this Document

This is the annotated solution guide for *Thinking in Java, Fourth Edition*. *Thinking in Java, Fourth Edition* is available in print from Prentice Hall and for sale electronically from www.mindview.net. This solution guide is only available online as a PDF document along with the associated source code (if printed in the same format as *Thinking in Java*, this guide would be almost 900 pages).

You can download a free sample of this solution guide, up to and including the solutions for the chapters *Everything is an Object* and *Operators*, from http://www.mindviewinc.com/Books/TIJ4/Solutions. You should do this before buying the guide to make sure that you can properly install, compile and run the solutions on your system.

The complete version of this book including all exercises is only available electronically, for US $25 (credit cards or PayPal), from http://www.mindviewinc.com/Books/TIJ4/Solutions/.

Unpacking the Distribution

The *Annotated Solutions Guide* is distributed as a zipped file containing the book in Adobe Acrobat PDF format, along with a source-code tree in the **code.zip** file. **Please make sure your computer can unzip files before purchasing the guide**. There are free unzip utilities for virtually every platform, available by searching on the Internet.

Linux/Unix (including Mac OSX) Users: Unzip the file using Info-Zip (pre-installed on many Linux distributions, or available at http://www.info-zip.org/). Unzip the package for Linux/Unix using the **–a** option to correct for the difference between DOS and Unix newlines, like this:

```
unzip –a TIJ4-solutions.zip
```

All Users: This guide uses the *Ant* build system, and includes Ant **build.xml** files in each subdirectory of the code tree, which compile and run the code examples using the **javac** compiler in the Sun JDK (available at http://java.sun.com/j2se/). Ant, the standard build tool for Java projects, is an open-source tool. The full download, installation and configuration instructions, along with the Ant executable and documentation are available at http://ant.apache.org/.

Once you install and configure Ant on your computer, you can type **ant** at the command prompt of the guide's source-code root directory to build and test all the code. You can also choose to build and test the code for a particular chapter. For example, to build and test the code from the *Polymorphism* chapter, enter the **polymorphism** sub-directory and type **ant** from there.

Full detailed instructions for installation can be found after the table of contents.

No Exercises in Chapter 1

The chapter *Introduction to Objects* has no exercises.

Additional exercises

This guide features additional exercises not included in *Thinking in Java*, for which solutions are not provided, as a challenge to the reader.

Contents

Installing the Code

Detailed instructions for installing, configuring and testing the source code.

These instructions describe a Windows installation, but they will also act as a guide for OSX and Linux installations.

These instructions also work with the free demo version of the guide.

1. Create a directory called **C:\TIJ4-Solutions\code**.

2. Using WinZip or some other zip utility (if one is not preinstalled, search the web for a free utility), extract the zip file containing the code that you received when you purchased the guide. Unzip it into the **C:\TIJ4-Solutions\code** directory. When you're done, you should see numerous subdirectories in the **C:\TIJ4-Solutions\code** directory, including subdirectories corresponding to the chapters in the solution guide.

3. Install the Java SE Development Kit (JDK), version 5 or newer, from the download site at Sun (http://java.sun.com/javase/downloads/index.jsp). You'll also eventually want the documentation, which is available from the same site.

4. Set the CLASSPATH in your computer's environment. For Windows machines, right-click on the "My Computer" icon and select "Properties." Then select the "Advanced" tab and click the "Environment Variables" button at the bottom. Under "System Variables," look to see if there's already a "CLASSPATH" variable. If there is, double click it and add
   ;.;..;C:\TIJ4-Solutions\code;
   to the end of the current entry.

   If there is no "CLASSPATH" variable, click the "New" button and enter
   CLASSPATH
   In the "Variable name" box, and
   .;..;C:\TIJ4-Solutions\code;
   In the "Variable value" box, then click "OK". To verify that your classpath has been set, start a command prompt (see below), then enter **set** and look for the **CLASSPATH** information in the output.

5. Using the same technique as in Step 4, but for PATH instead of CLASSPATH, add the **bin** directory from your Java installation into your

system's PATH environment variable. On Windows, the default JDK installation path is under "**C:\Program Files**" and because this has spaces, you must quote that directory when adding it to the PATH: **C:\"Program Files"\Java\bin;**

6. Create a directory called **C:\jars**. Place the following files into this directory:

   - **javassist.jar** (download here: http://sourceforge.net/project/showfiles.php?group_id=22866; you may need to search for it).
   - **swt.jar** from the Eclipse SWT library (http://download.eclipse.org/eclipse/downloads/). Click on the most recent build number, then scroll down to "SWT Binary and Source" and select the file corresponding to your platform. Further details about finding the jar file are in *Thinking in Java, 4th Edition*, under the heading "Installing SWT."
   - **tools.jar**, which is actually part of the JDK, but you must explicitly add it to your classpath. You'll find it in the **lib** directory wherever you installed the JDK on your machine. (The default is **C:\"Program Files"\Java\lib**).
   - **javaws.jar**, also part of the JDK, in the **/jre/lib/** directory.
   - **xom.jar**, available from http://www.cafeconleche.org/XOM/.

7. You must explicitly add each of the Jar files to your CLASSPATH, following the directions in Step 4. However, *you must also include the name of the Jar file in the CLASSPATH entry*. For example, after you put the **javassist.jar** file into the **C:\jars** directory, the associated CLASSPATH entry is **C:\jars\javassist.jar;**.

8. Install the **Ant** 1.7 (or newer) build tool by following the instructions you will find in the Ant download at http://ant.apache.org/.
   Note: **Ant** is required in order to compile the examples in the book. Once you successfully run '**ant build**' in the root directory, you can also compile each example individually (once you have the CLASSPATH set, as described in Step 4) using the **javac** command-line compiler that was installed when you completed the steps 3 and 5. To compile a file called **MyProgram.java**, you type **javac MyProgram.java**.

9. Start a command prompt in the **C:\TIJ4- Solutions\code** directory. To do this in Windows, press the "Start" button, then select "Run" and type "**cmd**" and press "OK." then type

cd C:\TIJ4-Solutions\code
into the resulting command window.

10. At the prompt, type
ant build
The build should successfully compile all the chapters in the solution guide.

11. Once you've run **ant build** in the root directory, you can also move into individual chapters and type **ant** (to compile and execute the code in that chapter) or **ant build** (to compile the code only).

12. This code is designed to work without an IDE, but it has also been tested with Eclipse (free at http://www.eclipse.org/); see the following section for instructions on how to use the code with Eclipse.

    If you want to use this code inside other IDEs you might need to make appropriate adjustments. Different IDEs have different requirements and you might find it's more trouble than it's worth right now; instead, you may want to begin with a more basic editor like JEdit (free at http://www.jedit.org/).

13. Note: The output for the programs has been verified for Java 6. Certain programs (primarily those that use hashing) can produce different output from one version to the next.

Using Eclipse

Once you've followed the above instructions, you can use the code inside the Eclipse development environment as follows:

1. Install Eclipse from http://www.eclipse.org/downloads/; choose a version for Java developers and follow the installation instructions.

2. Start Eclipse, then choose File | New | Java Project from the main menu.

3. In the ensuing dialog box, under "Contents," select "Create Project from Existing Source." Press the "Browse" button and navigate to **C:\TIJ4-Solutions\code**. Enter "TIJ4-Solutions" as the project name and press the "Finish" button.

   Note: If you are installing the demo version of the solution guide, you do not need to perform any of the following steps.

4. Eclipse will work for awhile and then present you with a "problems" pane containing a lot of errors and warnings. We'll remove the errors in the following steps.

5. In the "Package Explorer" pane, right click on "TIJ4-Solutions" and select "Properties" (at the bottom). In the left column of the ensuing dialog box, select "Java Build Path."

6. Select the "Source" tab. The default Eclipse configuration may have chosen to exclude and include some files. Click on "Included" and press the "Remove" button, then click on "Excluded" and press "Remove."

7. Click on "Excluded" and press "Edit." Under "Exclusion Patterns," add the following files, which are not intended to compile. After you add the files, press the "Finish" button.

   - access/E04_ForeignClass.java
   - arrays/E11_AutoboxingWithArrays.java
   - interfaces/E02_Abstract.java
   - reusing/E06_ChessWithoutDefCtor.java
   - reusing/E20_OverrideAnnotation.java
   - reusing/E21_FinalMethod.java
   - reusing/E22_FinalClass.java

8. Click on the "Libraries" tab, then the "Add External Jars" button. Add **tools.jar**, **swt.jar**, **javassist.jar** and **xom.jar** that are described in step 6 of the previous section.

9. When you press OK to close the dialog box, the project should rebuild without any errors. The warnings that you see refer to code that is intentional for those solutions, in order to demonstrate features and issues of the language.

Packages & IDEs

When Java first appeared there was no *Integrated Development Environment* (IDE) support, so you typically used a text editor and the command-line compiler. Over the years, IDE support has gotten so good (and many prevalent IDEs are free) that it's less and less likely that you'll develop in Java – or even learn the language – without an IDE.

There's a conflict, however, between IDEs and the way that *Thinking in Java* attempts to teach the language: One step at a time, using language features only after they've been introduced.

An IDE like *Eclipse* (from *www.Eclipse.org)* likes to have all its code in packages (later versions have become more tolerant of unpackaged code, but it still prefers packages). Packages, however, are not introduced until the *Access Control* chapter.

Because of the prevalence of IDEs, we have chosen to include **package** statements for all the code in this book, even for chapters before *Access Control*. If you have solved the problems in those chapters without using **package** statements, your solutions are still correct.

Left to the Reader

We have left only a few exercises to the reader. For these, the solution typically requires some configuration on your own computer.

The exercises left to the reader include:

- Exercises 12 & 13 from the chapter *Everything Is an Object*
- Exercise 13 from the chapter *Initialization & Cleanup*
- Exercise 2 from the chapter *Access Control*
- Exercise 15 from the chapter *Generics*
- Exercise 8 from the chapter *Arrays*
- Exercise 21 from the chapter *Containers in Depth*
- Exercise 35, 36, 38, 39, and 43 from the chapter *Graphical User Interfaces*

Everything is an Object

To satisfy IDEs like *Eclipse*, we have included **package** statements for chapters before *Access Control*. If you have solved the problems in this chapter without using **package** statements, your solutions are still correct.

Exercise 1

```
//: object/E01_DefaultInitialization.java
/****************** Exercise 1 *****************
 * Create a class containing an int and a char
 * that are not initialized. Print their values
 * to verify that Java performs default
 * initialization.
 *********************************************/
package object;

public class E01_DefaultInitialization {
  int i;
  char c;
  public E01_DefaultInitialization() {
    System.out.println("i = " + i);
    System.out.println("c = [" + c + ']');
  }
  public static void main(String[] args) {
    new E01_DefaultInitialization();
  }
} /* Output:
i = 0
c = [ ]
*///:~
```

When you run the program you'll see that both variables are given default values: `0` for the **int**, and a "space" for the **char**.

Exercise 2

```
//: object/E02_HelloWorld.java
```

```
/****************** Exercise 2 ******************
 * Follow the HelloDate.java example in this
 * chapter to create a "hello, world" program that
 * simply displays that statement. You need only a
 * single method in your class (the "main" one that
 * executes when the program starts). Remember
 * to make it static and to include the argument
 * list (even though you don't use it).
 * Compile the program with javac and run it using
 * java. If you are using a different development
 * environment than the JDK, learn how to compile
 * and run programs in that environment.
 *************************************************/
package object;

public class E02_HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
} /* Output:
Hello, world!
*///:~
```

Exercise 3

```
//: object/E03_ATypeName.java
/****************** Exercise 3 ******************
 * Turn the code fragments involving ATypeName
 * into a program that compiles and
 * runs.
 *************************************************/
package object;

public class E03_ATypeName {
  public static void main(String[] args) {
    E03_ATypeName a = new E03_ATypeName();
  }
} ///:~
```

Exercise 4

```
//: object/E04_DataOnly.java
/****************** Exercise 4 ******************
 * Turn the DataOnly code fragments into a
 * program that compiles and runs.
```

```
 *************************************************/
package object;

public class E04_DataOnly {
  int i;
  double d;
  boolean b;
  public static void main(String[] args) {
    E04_DataOnly d = new E04_DataOnly();
    d.i = 47;
    d.d = 1.1;
    d.b = false;
  }
} ///:~
```

Exercise 5

```
//: object/E05_DataOnly2.java
/****************** Exercise 5 ******************
 * Modify Exercise 4 so the values
 * of the data in DataOnly are assigned to and
 * printed in main().
 *************************************************/
package object;

public class E05_DataOnly2 {
  public static void main(String[] args) {
    E04_DataOnly d = new E04_DataOnly();
    d.i = 47;
    System.out.println("d.i = " + d.i);
    d.d = 1.1;
    System.out.println("d.d = " + d.d);
    d.b = false;
    System.out.println("d.b = " + d.b);
  }
} /* Output:
d.i = 47
d.d = 1.1
d.b = false
*///:~
```

Exercise 6

```
//: object/E06_Storage.java
/****************** Exercise 6 ******************
```

```
 * Write a program that includes and calls the
 * storage() method defined as a code fragment in
 * this chapter.
 ***********************************************/
package object;

public class E06_Storage {
  String s = "Hello, World!";
  int storage(String s) {
    return s.length() * 2;
  }
  void print() {
    System.out.println("storage(s) = " + storage(s));
  }
  public static void main(String[] args) {
    E06_Storage st = new E06_Storage();
    st.print();
  }
} /* Output:
storage(s) = 26
*///:~
```

Exercise 7

```
//: object/E07_Incrementable.java
/****************** Exercise 7 *****************
 * Turn the Incrementable code fragments into a
 * working program.
 ***********************************************/
package object;

class StaticTest {
  static int i = 47;
}

public class E07_Incrementable {
  static void increment() { StaticTest.i++; }
  public static void main(String[] args) {
    E07_Incrementable sf = new E07_Incrementable();
    sf.increment();
    E07_Incrementable.increment();
    increment();
  }
} ///:~
```

You can call **increment( )** by itself, because a **static** method (**main( )**, in this case) can call another **static** method without qualification.

Exercise 8

```
//: object/E08_StaticTest.java
/****************** Exercise 8 ******************
 * Write a program to demonstrate that no
 * matter how many objects you create of a
 * particular class, there is only one instance
 * of a particular static field in that class.
 ************************************************/
package object;

public class E08_StaticTest {
  static int i = 47;
  public static void main(String[] args) {
    E08_StaticTest st1 = new E08_StaticTest();
    E08_StaticTest st2 = new E08_StaticTest();
    System.out.println(st1.i + " == " + st2.i);
    st1.i++;
    System.out.println(st1.i + " == " + st2.i);
  }
} /* Output:
47 == 47
48 == 48
*///:~
```

The output shows that both instances of **E08_StaticTest** share the same **static** field. We incremented the shared field through the first instance and the effect was visible in the second instance.

Exercise 9

```
//: object/E09_AutoboxingTest.java
/****************** Exercise 9 ******************
 * Write a program to demonstrate that
 * autoboxing works for all the primitive types
 * and their wrappers.
 ************************************************/
package object;

public class E09_AutoboxingTest {
  public static void main(String[] args) {
```

```
    Byte by = 1;
    byte bt = by;
    System.out.println("byte = " + bt);
    Short sh = 1;
    short s = sh;
    System.out.println("short = " + s);
    Integer in = 1;
    int i = in;
    System.out.println("int = " + i);
    Long lo = 1L;
    long l = lo;
    System.out.println("long = " + l);
    Boolean bo = true;
    boolean b = bo;
    System.out.println("boolean = " + b);
    Character ch = 'x';
    char c = ch;
    System.out.println("char = " + c);
    Float fl = 1.0f;
    float f = fl;
    System.out.println("float = " + f);
    Double db = 1.0d;
    double d = db;
    System.out.println("double = " + d);
  }
} /* Output:
byte = 1
short = 1
int = 1
long = 1
boolean = true
char = x
float = 1.0
double = 1.0
*///:~
```

The terms *Autoboxing* and *Autounboxing* appear often in the literature. The only difference is the direction of the conversion: autoboxing converts from the primitive type to the wrapper object, and autounboxing converts from the wrapped type to the primitive type.

Exercise 10

```
//: object/E10_ShowArgs.java
// {Args: A B C}
/****************** Exercise 10 ****************
```

```
 * Write a program that prints three arguments
 * taken from the command line.
 * You'll need to index into the command-line
 * array of Strings.
 *********************************************/
package object;

public class E10_ShowArgs {
  public static void main(String[] args) {
    System.out.println(args[0]);
    System.out.println(args[1]);
    System.out.println(args[2]);
  }
} /* Output:
A
B
C
*///:~
```

Remember, when you want to get an argument from the command line:

- Arguments are provided in a **String** array.
- **args[0]** is the first command-line argument and *not* the name of the program (as it is in C).
- You'll cause a runtime exception if you run the program without enough arguments.

You can test for the length of the command-line argument array like this:

```
//: object/E10_ShowArgs2.java
// {Args: A B C}
package object;

public class E10_ShowArgs2 {
  public static void main(String[] args) {
    if(args.length < 3) {
      System.err.println("Need 3 arguments");
      System.exit(1);
    }
    System.out.println(args[0]);
    System.out.println(args[1]);
    System.out.println(args[2]);
  }
} /* Output:
A
B
```

```
C
*///:~
```

System.exit( ) terminates the program and passes its argument back to the operating system as a status code. (With most operating systems, a non-zero status code indicates that the program execution failed.) Typically, you send error messages to **System.err**, as shown above.

Exercise 11

```
//: object/E11_AllTheColorsOfTheRainbow.java
/****************** Exercise 11 *****************
 * Turn the AllTheColorsOfTheRainbow example into
 * a program that compiles and runs.
 ************************************************/
package object;

public class E11_AllTheColorsOfTheRainbow {
  int anIntegerRepresentingColors;
  void changeTheHueOfTheColor(int newHue) {
    anIntegerRepresentingColors = newHue;
  }
  public static void main(String[] args) {
    E11_AllTheColorsOfTheRainbow all =
      new E11_AllTheColorsOfTheRainbow();
    all.changeTheHueOfTheColor(27);
  }
} ///:~
```

Exercise 12

```
//: object/E12_LeftToReader.java
/****************** Exercise 12 *****************
 * Find the code for the second version of
 * HelloDate.java, the simple comment-
 * documentation example. Execute Javadoc on the
 * file and view the results with your Web browser.
 ************************************************/
package object;

public class E12_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
} ///:~
```

Note that **Javadoc** doesn't automatically create the destination directory. Consult the **Javadoc** reference in the JDK documentation to learn the many uses of **Javadoc**.

Exercise 13

```
//: object/E13_LeftToReader.java
/****************** Exercise 13 *****************
 * Run Documentation1.java, Documentation2.java,
 * and Documentation3.java through Javadoc. Verify
 * the resulting documentation with your Web
 * browser.
 ************************************************/
package object;

public class E13_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
} ///:~
```

Exercise 14

```
//: object/E14_DocTest.java
/****************** Exercise 14 *****************
 * Add an HTML list of items to the documentation
 * in Exercise 13.
 ************************************************/
package object;

/** A class comment
* <pre>
* System.out.println(new Date());
* </pre>
*/
public class E14_DocTest {
  /** A variable comment */
  public int i;
  /** A method comment
  * You can <em>even</em> insert a list:
  * <ol>
  * <li> Item one
  * <li> Item two
  * <li> Item three
```

```
   * </ol>
   */
   public void f() {}
} ///:~
```

We simply added the HTML code fragments from the chapter examples.

Exercise 15

```
//: object/E15_HelloWorldDoc.java
/****************** Exercise 15 *****************
 * Add comment documentation to the program in Exercise 2.
 * Extract it into an HTML file using Javadoc
 * and view it with your Web browser.
 ************************************************/
package object;

/** A first example from <i>sTIJ4</i>.
* Demonstrates the basic class
* structure and the creation of a
* <code>main()</code> method.
*/
public class E15_HelloWorldDoc {
  /** The <code>main()</code> method which is
  called when the program is executed by saying
  <code>java E15_HelloWorldDoc</code>.
  @param args array passed from the command-line
  */
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
} /* Output:
Hello, world!
*///:~
```

Exercise 16

```
//: object/E16_OverloadingDoc.java
/****************** Exercise 16 *****************
 * In the Initialization and Cleanup chapter, add
 * Javadoc documentation to the Overloading.java example.
 * Extract it into an HTML file using Javadoc
 * and view it with your Web browser.
 ************************************************/
package object;
```

```
/** Model of a single arboreal unit. */
class Tree {
  /** Current vertical aspect to the tip. */
  int height;  // 0 by default
  /** Plant a seedling. Assume height can
      be considered as zero. */
  Tree() {
    System.out.println("Planting a seedling");
  }
  /** Transplant an existing tree with a given height. */
  Tree(int i) {
    System.out.println("Creating new Tree that is "
      + i + " feet tall");
    height = i;
  }
  /** Produce information about this unit. */
  void info() {
    System.out.println("Tree is " + height + " feet tall");
  }
  /** Produce information with optional message. */
  void info(String s) {
    System.out.println(s + ": Tree is "
      + height + " feet tall");
  }
}

/** Simple test code for Tree class */
public class E16_OverloadingDoc {
  /** Creates <b>Tree</b> objects and exercises the two
      different <code>info()</code> methods. */
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++) {
      Tree t = new Tree(i);
      t.info();
      t.info("overloaded method");
    }
    // Overloaded constructor:
    new Tree();
  }
} /* Output:
Creating new Tree that is 0 feet tall
Tree is 0 feet tall
overloaded method: Tree is 0 feet tall
Creating new Tree that is 1 feet tall
Tree is 1 feet tall
overloaded method: Tree is 1 feet tall
```

```
Creating new Tree that is 2 feet tall
Tree is 2 feet tall
overloaded method: Tree is 2 feet tall
Creating new Tree that is 3 feet tall
Tree is 3 feet tall
overloaded method: Tree is 3 feet tall
Creating new Tree that is 4 feet tall
Tree is 4 feet tall
overloaded method: Tree is 4 feet tall
Planting a seedling
*///:~
```

The one-argument constructor does not check the input argument, which should be greater than zero. Statements that control the execution flow of the program appear in a later chapter of *TIJ4*.

Operators

To satisfy IDEs like *Eclipse*, we have included **package** statements for chapters before *Access Control*. If you have solved the problems in this chapter without using **package** statements, your solutions are still correct.

Exercise 1

```
//: operators/E01_PrintStatements.java
/****************** Exercise 1 *****************
 * Write a program that uses the "short" and
 * normal form of print statement.
 ***********************************************/
package operators;
import java.util.Date;
import static net.mindview.util.Print.*;

public class E01_PrintStatements {
  public static void main(String[] args) {
    Date currDate = new Date();
    System.out.println("Hello, it's: " + currDate);
    print("Hello, it's: " + currDate);
  }
} /* Output: (Sample)
Hello, it's: Wed Mar 30 17:39:26 CEST 2005
Hello, it's: Wed Mar 30 17:39:26 CEST 2005
*///:~
```

Exercise 2

```
//: operators/E02_Aliasing.java
/****************** Exercise 2 *****************
 * Create a class containing a float and use it to
 * demonstrate aliasing.
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;

class Integral {
  float f;
}
```

```
public class E02_Aliasing {
  public static void main(String[] args) {
    Integral n1 = new Integral();
    Integral n2 = new Integral();
    n1.f = 9f;
    n2.f = 47f;
    print("1: n1.f: " + n1.f + ", n2.f: " + n2.f);
    n1 = n2;
    print("2: n1.f: " + n1.f + ", n2.f: " + n2.f);
    n1.f = 27f;
    print("3: n1.f: " + n1.f + ", n2.f: " + n2.f);
  }
} /* Output:
1: n1.f: 9.0, n2.f: 47.0
2: n1.f: 47.0, n2.f: 47.0
3: n1.f: 27.0, n2.f: 27.0
*///:~
```

You can see the effect of aliasing after **n2** is assigned to **n1**: they both point to the same object.

Exercise 3

```
//: operators/E03_Aliasing2.java
/****************** Exercise 3 *****************
 * Create a class containing a float and use it
 * to demonstrate aliasing during method calls.
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;

public class E03_Aliasing2 {
  static void f(Integral y) { y.f = 1.0f; }
  public static void main(String[] args) {
    Integral x = new Integral();
    x.f = 2.0f;
    print("1: x.f: " + x.f);
    f(x);
    print("2: x.f: " + x.f);
  }
} /* Output:
1: x.f: 2.0
2: x.f: 1.0
*///:~
```

This exercise emphasizes that you're always passing references around, thus you're always aliasing. Even when you don't actually see changes being made to the code you're writing or the method you're calling, that code or method could be calling other methods that modify the object.

Exercise 4

```
//: operators/E04_Velocity.java
// {Args: 30.5 3.2}
/****************** Exercise 4 *****************
 * Write a program that calculates velocity
 * using a constant distance and a constant time.
 ***********************************************/
package operators;

public class E04_Velocity {
  public static void main(String[] args) {
    if(args.length < 2) {
      System.err.println(
        "Usage: java E04_Velocity distance time");
      System.exit(1);
    }
    float distance = Float.parseFloat(args[0]);
    float time = Float.parseFloat(args[1]);
    System.out.print("Velocity = ");
    System.out.print(distance / time);
    // Change the next line if you want to use a different
    // unit for 'distance'
    System.out.println(" m/s");
  }
} /* Output:
Velocity = 9.53125 m/s
*///:~
```

Here we take the **distance** and **time** values from the command line. Arguments come in as a **String** array; if you need a **float** instead, use the **static parseFloat( )** method of class **Float**. This can be difficult to locate using the JDK HTML documentation; you must remember either "parse" or that it's part of class **Float**.

Note the difference between **System.out.print( )** and **System.out.println( )**; the latter terminates the current line by writing the line separator string.

Exercise 5

```
//: operators/E05_Dogs.java
/****************** Exercise 5 *****************
 * Create a class called Dog with two Strings:
 * name and says. In main(), create two dogs,
 * "spot" who says, "Ruff!", and "scruffy" who
 * says, "Wurf!". Then display their names and
 * what they say.
 ***********************************************/
package operators;

class Dog {
  String name;
  String says;
}

public class E05_Dogs {
  public static void main(String[] args) {
    Dog dog1 = new Dog();
    Dog dog2 = new Dog();
    dog1.name = "spot";      dog1.says = "ruff!";
    dog2.name = "scruffy";   dog2.says = "wurf!";
    System.out.println(dog1.name + " says " + dog1.says);
    System.out.println(dog2.name + " says " + dog2.says);
  }
} /* Output:
spot says ruff!
scruffy says wurf!
*///:~
```

This walks you through the basics of objects, and demonstrates that each object has its own distinct storage space.

Exercise 6

```
//: operators/E06_DogsComparison.java
/****************** Exercise 6 *****************
 * Following Exercise 5 assign, a new Dog
 * reference to spot's object. Test for comparison
 * using == and equals() for all references.
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;
```

```
public class E06_DogsComparison {
  static void compare(Dog dog1, Dog dog2) {
    print("== on top references: " + (dog1 == dog2));
    print(
      "equals() on top references: " + dog1.equals(dog2)
    );
    print("== on names: " + (dog1.name == dog2.name));
    print(
      "equals() on names: " + dog1.name.equals(dog2.name)
    );
    print("== on says: " + (dog1.says == dog2.says));
    print(
      "equals() on says: " + dog1.says.equals(dog2.says)
    );
  }
  public static void main(String[] args) {
    Dog dog1 = new Dog();
    Dog dog2 = new Dog();
    Dog dog3 = dog1;  // "Aliased" reference
    dog1.name = "spot";    dog1.says = "ruff!";
    dog2.name = "scruffy"; dog2.says = "wurf!";
    print("Comparing dog1 and dog2 objects...");
    compare(dog1, dog2);
    print("\nComparing dog1 and dog3 objects...");
    compare(dog1, dog3);
    print("\nComparing dog2 and dog3 objects...");
    compare(dog2, dog3);
  }
} /* Output:
Comparing dog1 and dog2 objects...
== on top references: false
equals() on top references: false
== on names: false
equals() on names: false
== on says: false
equals() on says: false

Comparing dog1 and dog3 objects...
== on top references: true
equals() on top references: true
== on names: true
equals() on names: true
== on says: true
equals() on says: true

Comparing dog2 and dog3 objects...
== on top references: false
```

```
equals() on top references: false
== on names: false
equals() on names: false
== on says: false
equals() on says: false
*///:~
```

Guess whether the following line compiles:

```
print("== on top references: " + dog1 == dog2);
```

Why or why not? (Hint: Read the *Precedence* and *String operator + and +=* sections in *TIJ4*.) Apply the same reasoning to the next case and explain why the comparison always results in **false**:

```
print("== on says: " + dog1.name == dog2.name);
```

Exercise 7

```
//: operators/E07_CoinFlipping.java
/****************** Exercise 7 *****************
 * Write a program that simulates coin-flipping.
 ***********************************************/
package operators;
import java.util.Random;

public class E07_CoinFlipping {
  public static void main(String[] args) {
    Random rand = new Random(47);
    boolean flip = rand.nextBoolean();
    System.out.print("OUTCOME: ");
    System.out.println(flip ? "HEAD" : "TAIL");
  }
} /* Output:
OUTCOME: HEAD
*///:~
```

This is partly an exercise in Standard Java Library usage. After familiarizing yourself with the HTML documentation for the JDK (downloadable from *java.sun.com*), select "R" at the JDK index to see various ways to generate random numbers.

The program uses a ternary if-else operator to produce output. (See the *Ternary if-else Operator* section in *TIJ4* for more information.)

NOTE: You will normally create a **Random** object with no arguments to produce different output for each execution of the program. Otherwise it can

hardly be called a *simulator*. In this exercise and throughout the book, we use the seed value of 47 to make the output identical, thus verifiable, for each run.

Exercise 8

```
//: operators/E08_LongLiterals.java
/****************** Exercise 8 *****************
 * Show that hex and octal notations work with long
 * values. Use Long.toBinaryString() to display
 * the results.
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;

public class E08_LongLiterals {
  public static void main(String[] args) {
    long l1 = 0x2f; // Hexadecimal (lowercase)
    print("l1: " + Long.toBinaryString(l1));
    long l2 = 0X2F; // Hexadecimal (uppercase)
    print("l2: " + Long.toBinaryString(l2));
    long l3 = 0177; // Octal (leading zero)
    print("l3: " + Long.toBinaryString(l3));
  }
} /* Output:
l1: 101111
l2: 101111
l3: 1111111
*///:~
```

Note that **Long.toBinaryString( )** does not print leading zeroes.

Exercise 9

```
//: operators/E09_MinMaxExponents.java
/****************** Exercise 9 *****************
 * Display the largest and smallest numbers for
 * both float and double exponential notation.
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;

public class E09_MinMaxExponents {
  public static void main(String[] args) {
    print("Float MIN: " + Float.MIN_VALUE);
    print("Float MAX: " + Float.MAX_VALUE);
```

```
    print("Double MIN: " + Double.MIN_VALUE);
    print("Double MAX: " + Double.MAX_VALUE);
  }
} /* Output:
Float MIN: 1.4E-45
Float MAX: 3.4028235E38
Double MIN: 4.9E-324
Double MAX: 1.7976931348623157E308
*///:~
```

Exercise 10

```
//: operators/E10_BitwiseOperators.java
/****************** Exercise 10 *****************
 * Write a program with two constant values, one
 * with alternating binary ones and zeroes, with
 * a zero in the least-significant digit, and the
 * second, also alternating, with a one in the
 * least-significant digit. (Hint: It's easiest to
 * use hexadecimal constants for this.) Combine
 * these two values every way possible using the
 * bitwise operators.  Display the results using
 * Integer.toBinaryString().
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;

public class E10_BitwiseOperators {
  public static void main(String[] args) {
    int i1 = 0xaaaaaaaa;
    int i2 = 0x55555555;
    print("i1 = " + Integer.toBinaryString(i1));
    print("i2 = " + Integer.toBinaryString(i2));
    print("~i1 = " + Integer.toBinaryString(~i1));
    print("~i2 = " + Integer.toBinaryString(~i2));
    print("i1 & i1 = " + Integer.toBinaryString(i1 & i1));
    print("i1 | i1 = " + Integer.toBinaryString(i1 | i1));
    print("i1 ^ i1 = " + Integer.toBinaryString(i1 ^ i1));
    print("i1 & i2 = " + Integer.toBinaryString(i1 & i2));
    print("i1 | i2 = " + Integer.toBinaryString(i1 | i2));
    print("i1 ^ i2 = " + Integer.toBinaryString(i1 ^ i2));
  }
} /* Output:
i1 = 10101010101010101010101010101010
i2 = 01010101010101010101010101010101
~i1 = 01010101010101010101010101010101
```

```
~i2 = 10101010101010101010101010101010
i1 & i1 = 10101010101010101010101010101010
i1 | i1 = 10101010101010101010101010101010
i1 ^ i1 = 0
i1 & i2 = 0
i1 | i2 = 11111111111111111111111111111111
i1 ^ i2 = 11111111111111111111111111111111
*///:~
```

Note that **Integer.toBinaryString( )** does not print leading zeroes.

Exercise 11

```
//: operators/E11_SignedRightShift.java
/****************** Exercise 11 *****************
 * Start with a number that has a binary one in
 * the most significant position. (Hint: Use a
 * hexadecimal constant.) Use the signed
 * right-shift operator to right shift your
 * number through all its binary positions.
 * Display each result using Integer.toBinaryString().
 ************************************************/
package operators;
import static net.mindview.util.Print.*;

public class E11_SignedRightShift {
  public static void main(String[] args) {
    int i = 0x80000000;
    print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
```

```
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
    i >>= 1;  print(Integer.toBinaryString(i));
  }
} /* Output:
10000000000000000000000000000000
11000000000000000000000000000000
11100000000000000000000000000000
11110000000000000000000000000000
11111000000000000000000000000000
11111100000000000000000000000000
11111110000000000000000000000000
11111111000000000000000000000000
11111111100000000000000000000000
11111111110000000000000000000000
11111111111000000000000000000000
11111111111100000000000000000000
11111111111110000000000000000000
11111111111111000000000000000000
11111111111111100000000000000000
11111111111111110000000000000000
11111111111111111000000000000000
11111111111111111100000000000000
11111111111111111110000000000000
11111111111111111111000000000000
11111111111111111111100000000000
11111111111111111111110000000000
11111111111111111111111000000000
11111111111111111111111100000000
11111111111111111111111110000000
11111111111111111111111111000000
11111111111111111111111111100000
11111111111111111111111111110000
11111111111111111111111111111000
11111111111111111111111111111100
11111111111111111111111111111110
```

```
11111111111111111111111111111111
*///:~
```

This exercise required duplicating the line of code (the “brute force” approach). In the next chapter you learn to use looping structures to greatly reduce the amount of code you see above.

Exercise 12

```
//: operators/E12_UnsignedRightShift.java
/****************** Exercise 12 ******************
 * Start with a number that is all binary ones.
 * Left shift it, then use the unsigned right-shift
 * operator to right shift through all of its binary
 * positions. Display each result using
 * Integer.toBinaryString().
 ***********************************************/
package operators;
import static net.mindview.util.Print.*;

public class E12_UnsignedRightShift {
  public static void main(String[] args) {
    int i = -1 << 1;
    print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
```

```
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
    i >>>= 1;  print(Integer.toBinaryString(i));
  }
} /* Output:
11111111111111111111111111111110
1111111111111111111111111111111
111111111111111111111111111111
11111111111111111111111111111
1111111111111111111111111111
111111111111111111111111111
11111111111111111111111111
1111111111111111111111111
111111111111111111111111
11111111111111111111111
1111111111111111111111
111111111111111111111
11111111111111111111
1111111111111111111
111111111111111111
11111111111111111
1111111111111111
111111111111111
11111111111111
1111111111111
111111111111
11111111111
1111111111
111111111
11111111
1111111
111111
11111
1111
111
11
1
*///:~
```

See the note for Exercise 11.

Exercise 13

```
//: operators/E13_BinaryChar.java
/****************** Exercise 13 *****************
 * Write a method to display char values in
 * binary form. Demonstrate it using several
 * different characters.
 ************************************************/
package operators;
import static net.mindview.util.Print.*;

public class E13_BinaryChar {
  public static void main(String[] args) {
    print("A: " + Integer.toBinaryString('A'));
    print("!: " + Integer.toBinaryString('!'));
    print("x: " + Integer.toBinaryString('x'));
    print("7: " + Integer.toBinaryString('7'));
  }
} /* Output:
A: 1000001
!: 100001
x: 1111000
7: 110111
*///:~
```

Exercise 14

```
//: operators/E14_CompareStrings.java
/****************** Exercise 14 *****************
 * Write a method that compares two String arguments
 * using all the Boolean comparisons and print the
 * results. Perform the equals() test for the == and
 * !=. In main(), call your method with different
 * String objects.
 ************************************************/
package operators;

public class E14_CompareStrings {
  public static void p(String s, boolean b) {
    System.out.println(s + ": " + b);
  }
  public static void compare(String lval, String rval) {
    System.out.println("lval: " + lval + " rval: " + rval);
    //! p("lval < rval: " + lval < rval);
    //! p("lval > rval: " + lval > rval);
```

```
    //! p("lval <= rval: " + lval <= rval);
    //! p("lval >= rval: " + lval >= rval);
    p("lval == rval", lval == rval);
    p("lval != rval", lval != rval);
    p("lval.equals(rval)", lval.equals(rval));
  }
  public static void main(String[] args) {
    compare("Hello", "Hello");
    // Force creation of separate object:
    String s = new String("Hello");
    compare("Hello", s);
    compare("Hello", "Goodbye");
  }
} /* Output:
lval: Hello rval: Hello
lval == rval: true
lval != rval: false
lval.equals(rval): true
lval: Hello rval: Hello
lval == rval: false
lval != rval: true
lval.equals(rval): true
lval: Hello rval: Goodbye
lval == rval: false
lval != rval: true
lval.equals(rval): false
*///:~
```

The only comparisons that actually compile are == and **!**=. This (slightly tricky) exercise highlights the critical difference between the == and **!**= operators, which compare *references*, and **equals( )**, which actually compares *content*.

Remember that quoted character arrays also produce references to **String** objects. In the first case, the compiler recognizes that the two strings actually contain the same values. Because **String** objects are immutable (you cannot change their contents), the compiler can merge the two **String** objects into one, so == returns **true** in that case. However, when you create a separate **String s** you also create a distinct object with the same contents, therefore the == returns **false**. *The only reliable way to compare objects for equality is with* **equals( )**. Be wary of any comparison that uses ==, which always and only compares two references to see if they are identical (that is, they point to the same object).

Controlling Execution

To satisfy IDEs like *Eclipse*, we have included **package** statements for chapters before *Access Control*. If you have solved the problems in this chapter without using **package** statements, your solutions are still correct.

Exercise 1

```
//: control/E01_To100.java
/****************** Exercise 1 *****************
 * Write a program to print values from one to
 * 100.
 ***********************************************/
package control;

public class E01_To100 {
  public static void main(String[] args) {
    for(int i = 1; i <= 100; i++)
      System.out.print(i + " ");
  }
} /* Output:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
*///:~
```

This is the most trivial use of a **for** loop.

Try to simplify Exercises 11 and 12 in the previous chapter using a **for** loop.

Exercise 2

```
//: control/E02_RandomInts.java
/****************** Exercise 2 *****************
 * Write a program to generate 25 random int
 * values. Use an if-else statement for each value
 * to classify it as greater than, less than, or
 * equal to a second randomly generated value.
***********************************************/
package control;
```

```
import java.util.*;

public class E02_RandomInts {
  static Random r = new Random(47);
  public static void compareRand() {
    int a = r.nextInt();
    int b = r.nextInt();
    System.out.println("a = " + a + ", b = " + b);
    if(a < b)
      System.out.println("a < b");
    else if(a > b)
      System.out.println("a > b");
    else
      System.out.println("a = b");
  }
  public static void main(String[] args) {
    for(int i = 0; i < 25; i++)
      compareRand();
  }
} /* Output:
a = -1172028779, b = 1717241110
a < b
a = -2014573909, b = 229403722
a < b
a = 688081923, b = -1812486437
a > b
a = 809509736, b = 1791060401
a < b
a = -2076178252, b = -1128074882
a < b
a = 1150476577, b = -210207040
a > b
a = 1122537102, b = 491149179
a > b
a = 218473618, b = -1946952740
a > b
a = -843035300, b = 865149722
a < b
a = -1021916256, b = -1916708780
a > b
a = -2016789463, b = 674708281
a < b
a = -2020372274, b = 1703464645
a < b
a = 2092435409, b = 1072754767
a > b
a = -846991883, b = 488201151
```

```
a < b
a = 100996820, b = -855894611
a > b
a = -1612351948, b = 1891197608
a < b
a = -56789395, b = 849275653
a < b
a = 2078628644, b = -1099465504
a > b
a = 39716067, b = 875665968
a < b
a = 1738084688, b = -914835675
a > b
a = 1169976606, b = 1947946283
a < b
a = 691554276, b = -1004355271
a > b
a = -541407364, b = 1920737378
a < b
a = -1278072925, b = 281473985
a < b
a = -1439435803, b = -955419343
a < b
*///:~
```

In the solution above, we create a method that generates and compares the random numbers, then call that method 25 times. In your solution, you may have created all the code inline, inside **main( )**.

Exercise 3

```
//: control/E03_RandomInts2.java
// {RunByHand}
/****************** Exercise 3 ******************
 * Modify Exercise 2 so your code is
 * surrounded by an "infinite" while loop. It
 * will then run until you interrupt it,
 * typically with Control-C.
 ***********************************************/
package control;

public class E03_RandomInts2 {
  public static void main(String[] args) {
    while(true)
      E02_RandomInts.compareRand();
  }
```

```
} ///:~
```

A method outside of **main( )** did most of the work in the previous exercise, so this solution requires only a minor change to **main( )**. Structure a program properly and it requires fewer code changes during its lifetime. The benefit may lie in reducing the maintenance costs of the software rather than the cost of the initial release, but a well-designed program is usually easier to get running in the first place.

Exercise 4

```
//: control/E04_FindPrimes.java
/****************** Exercise 4 *****************
 * Write a program to detect and print prime numbers
 * (integers evenly divisible only by themselves
 * and 1), using two nested for loops and the
 * modulus operator (%).
 ***********************************************/
package control;

public class E04_FindPrimes {
  public static void main(String[] args) {
    int max = 100;
    // Get the max value from the command line,
    // if the argument has been provided:
    if(args.length != 0)
      max = Integer.parseInt(args[0]);
    for(int i = 1; i < max; i++) {
      boolean prime = true;
      for(int j = 2; j < i; j++)
        if(i % j == 0)
          prime = false;
      if(prime)
        System.out.print(i + " ");
    }
  }
} /* Output:
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
79 83 89 97
*///:~
```

Note that the program includes 1 as a prime number even though 2 is ordinarily considered the smallest prime.

One of the fastest ways of finding prime numbers is called the Sieve of Eratosthenes. The following program uses a **boolean** array to mark prime numbers.

```
//: control/E04_FindPrimes2.java
package control;

import static java.lang.Math.*;
import static net.mindview.util.Print.*;

public class E04_FindPrimes2 {
  public static void main(String[] args) {
    int max = 100;
    // Get the max value from the command line,
    // if the argument has been provided:
    if(args.length != 0)
      max = Integer.parseInt(args[0]);
    boolean[] sieve = new boolean[max + 1];
    int limit = (int)floor(sqrt(max));
    printnb(1 + " ");
    if(max > 1)
      printnb(2 + " ");
    // Detect prime numbers
    for(int i = 3; i <= limit; i += 2)
      if(!sieve[i])
        for(int j = 2 * i; j <= max; j += i)
          sieve[j] = true;
    // Print prime numbers
    for(int i = 3; i <= max; i += 2)
      if(!sieve[i])
        printnb(i + " ");
  }
} /* Output:
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
79 83 89 97
*///:~
```

You need only test odd numbers, since 2 is the only even prime. The program uses a **max + 1** sized array to make indexing easier. The search continues until you have crossed out all numbers divisible by $\lfloor\sqrt{\max}\rfloor$, where $\lfloor x \rfloor$ is the **floor** function. Additional optimizations are left as exercises.

Exercise 5

```
//: control/E05_BitwiseOperators2.java
```

```
/****************** Exercise 5 ******************
 * Repeat Exercise 10 from the previous chapter,
 * but use the ternary operator and a bitwise test
 * instead of Integer.toBinaryString() to display
 * the ones and zeroes.
 ************************************************/
package control;
import static net.mindview.util.Print.*;

public class E05_BitwiseOperators2 {
  private static void toBinaryString(int i) {
    char[] buffer = new char[32];
    int bufferPosition = 32;
    do {
      buffer[--bufferPosition] =
        ((i & 0x01) != 0) ? '1' : '0';
      i >>>= 1;
    } while (i != 0);
    for(int j = bufferPosition; j < 32; j++)
      printnb(buffer[j]);
    print();
  }
  public static void main(String[] args) {
    int i1 = 0xaaaaaaaa;
    int i2 = 0x55555555;
    printnb("i1 = ");  toBinaryString(i1);
    printnb("i2 = ");  toBinaryString(i2);
    printnb("~i1 = ");  toBinaryString(~i1);
    printnb("~i2 = ");  toBinaryString(~i2);
    printnb("i1 & i1 = ");  toBinaryString(i1 & i1);
    printnb("i1 | i1 = ");  toBinaryString(i1 | i1);
    printnb("i1 ^ i1 = ");  toBinaryString(i1 ^ i1);
    printnb("i1 & i2 = ");  toBinaryString(i1 & i2);
    printnb("i1 | i2 = ");  toBinaryString(i1 | i2);
    printnb("i1 ^ i2 = ");  toBinaryString(i1 ^ i2);
  }
} /* Output:
i1 = 10101010101010101010101010101010
i2 = 01010101010101010101010101010101
~i1 = 01010101010101010101010101010101
~i2 = 10101010101010101010101010101010
i1 & i1 = 10101010101010101010101010101010
i1 | i1 = 10101010101010101010101010101010
i1 ^ i1 = 0
i1 & i2 = 0
i1 | i2 = 11111111111111111111111111111111
i1 ^ i2 = 11111111111111111111111111111111
```

```
*///:~
```

The private **static** method **toBinaryString( )** behaves like **Integer.toBinaryString( )**, using **buffer** to hold the binary digits because printing out the digits as encountered would produce an inverted output.

Exercise 6

```
//: control/E06_RangeTest.java
/****************** Exercise 6 *****************
 * Modify the two test() methods in the previous
 * two programs so they take two extra
 * arguments, begin and end, and so testval is
 * tested to see if it is within the range between
 * (and including) begin and end.
 ***********************************************/
package control;

public class E06_RangeTest {
  static boolean test(int testval, int begin, int end) {
    boolean result = false;
    if(testval >= begin && testval <= end)
      result = true;
    return result;
  }
  public static void main(String[] args) {
    System.out.println(test(10, 5, 15));
    System.out.println(test(5, 10, 15));
    System.out.println(test(5, 5, 5));
  }
} /* Output:
true
false
true
*///:~
```

The **test( )** methods are now only testing for two conditions, so we changed the return value to **Boolean**.

Note that by using **return** in the following program, no intermediate **result** variable is necessary:

```
//: control/E06_RangeTest2.java
// No intermediate 'result' value necessary:
package control;
```

```
public class E06_RangeTest2 {
  static boolean test(int testval, int begin, int end) {
    if(testval >= begin && testval <= end)
      return true;
    return false;
  }
  public static void main(String[] args) {
    System.out.println(test(10, 5, 15));
    System.out.println(test(5, 10, 15));
    System.out.println(test(5, 5, 5));
  }
} /* Output:
true
false
true
*///:~
```

Exercise 7

```
//: control/E07_To98.java
/****************** Exercise 7 ******************
 * Modify Exercise 1 so the program exits by
 * using the break keyword at value 99. Try using
 * return instead.
 ***********************************************/
package control;

public class E07_To98 {
  public static void main(String[] args) {
    for(int i = 1; i <= 100; i++) {
      if(i == 99)
        break;
      System.out.print(i + " ");
    }
  }
} /* Output:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98
*///:~
```

There is no observable difference whether you use **break** or **return** in this program.

Exercise 8

This straightforward exercise demonstrates all the behaviors of **switch**. We wrote two programs, one with **break**s and one without. Here's the version with **break**s:

```
//: control/E08_SwitchDemo.java
/****************** Exercise 8 *****************
 * Create a switch statement inside a for loop
 * that tries each case and prints a message. Put
 * a break after each case and test it, then see
 * what happens when you remove the breaks.
 ***********************************************/
package control;

public class E08_SwitchDemo {
  public static void main(String[] args) {
    for(int i = 0; i < 7; i++)
      switch(i) {
        case 1: System.out.println("case 1");
                break;
        case 2: System.out.println("case 2");
                break;
        case 3: System.out.println("case 3");
                break;
        case 4: System.out.println("case 4");
                break;
        case 5: System.out.println("case 5");
                break;
        default: System.out.println("default");
      }
  }
} /* Output:
default
case 1
case 2
case 3
case 4
case 5
default
*///:~
```

As a demonstration, we allowed the value of **i** to go out of bounds. You can see that anything that doesn't match one of the cases goes to the **default** statement.

Here's the same program with the **break**s removed:

```
//: control/E08_SwitchDemo2.java
// E08_SwitchDemo.java with the breaks removed.
package control;

public class E08_SwitchDemo2 {
  public static void main(String[] args) {
    for(int i = 0; i < 7; i++)
      switch(i) {
        case 1: System.out.println("case 1");
        case 2: System.out.println("case 2");
        case 3: System.out.println("case 3");
        case 4: System.out.println("case 4");
        case 5: System.out.println("case 5");
        default: System.out.println("default");
      }
  }
} /* Output:
default
case 1
case 2
case 3
case 4
case 5
default
case 2
case 3
case 4
case 5
default
case 3
case 4
case 5
default
case 4
case 5
default
case 5
default
default
*///:~
```

Without the **break**, each case falls through to the next one. When you select **case 1** you get all the other cases as well, so you'll almost always want a **break** at the end of each **case**.

Exercise 9

```
//: control/E09_Fibonacci.java
// {Args: 20}
/****************** Exercise 9 *********************
 * A Fibonacci sequence is the sequence of numbers 1,
 * 1, 2, 3, 5, 8, 13, 21, 34, etc., where each
 * number (from the third on) is the sum of the previous
 * two. Create a method that takes an integer as an
 * argument and displays that many Fibonacci numbers
 * starting from the beginning. If, e.g., you run java
 * Fibonacci 5 (where Fibonacci is the name of the
 * class) the output will be: 1, 1, 2, 3, 5.
 ************************************************/
package control;

public class E09_Fibonacci {
  static int fib(int n) {
    if (n <= 2)
      return 1;
    return fib(n-1) + fib(n-2);
  }
  public static void main(String[] args) {
    // Get the max value from the command line:
    int n = Integer.parseInt(args[0]);
    if(n < 0) {
      System.out.println("Cannot use negative numbers");
      return;
    }
    for(int i = 1; i <= n; i++)
      System.out.print(fib(i) + ", ");
  }
} /* Output:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610,
987, 1597, 2584, 4181, 6765,
*///:~
```

This problem, commonly presented in introductory programming classes, uses *recursion*, meaning that a function calls itself until it reaches a bottoming-out condition and returns.

As an additional exercise, rewrite the solution without relying on recursion. (Hint: Use Binet's formula for the **n**th Fibonacci number.)

Exercise 10

```
//: control/E10_Vampire.java
/****************** Exercise 10 *********************
 * A vampire number has an even number of digits and
 * is formed by multiplying a pair of numbers containing
 * half the number of digits of the result. The digits
 * are taken from the original number in any order.
 * Pairs of trailing zeroes are not allowed. Examples
 * include:
 * 1260 = 21 * 60
 * 1827 = 21 * 87
 * 2187 = 27 * 81
 * Write a program that finds all the 4-digit vampire
 * numbers. (Suggested by Dan Forhan.)
 ****************************************************/
package control;

public class E10_Vampire {
  public static void main(String[] args) {
    int[] startDigit = new int[4];
    int[] productDigit = new int[4];
    for(int num1 = 10; num1 <= 99; num1++)
      for(int num2 = num1; num2 <= 99; num2++) {
        // Pete Hartley's theoretical result:
        // If x·y is a vampire number then
        // x·y == x+y (mod 9)
        if((num1 * num2) % 9 != (num1 + num2) % 9)
          continue;
        int product = num1 * num2;
        startDigit[0] = num1 / 10;
        startDigit[1] = num1 % 10;
        startDigit[2] = num2 / 10;
        startDigit[3] = num2 % 10;
        productDigit[0] = product / 1000;
        productDigit[1] = (product % 1000) / 100;
        productDigit[2] = product % 1000 % 100 / 10;
        productDigit[3] = product % 1000 % 100 % 10;
        int count = 0;
        for(int x = 0; x < 4; x++)
          for(int y = 0; y < 4; y++) {
            if(productDigit[x] == startDigit[y]) {
              count++;
              productDigit[x] = -1;
              startDigit[y] = -2;
              if(count == 4)
```

```
                System.out.println(num1 + " * " + num2
                  + " : " + product);
            }
          }
        }
    }
} /* Output:
15 * 93 : 1395
21 * 60 : 1260
21 * 87 : 1827
27 * 81 : 2187
30 * 51 : 1530
35 * 41 : 1435
80 * 86 : 6880
*///:~
```

The program does not produce duplicates, and is optimized by using Pete Hartley's theoretical result (see the comment inside **main( )**).

Initialization & Cleanup

To satisfy IDEs like *Eclipse*, we have included **package** statements for chapters before *Access Control*. If you have solved the problems in this chapter without using **package** statements, your solutions are still correct.

Exercise 1

```
//: initialization/E01_StringRefInitialization.java
/****************** Exercise 01 ****************
 * Create a class with an uninitialized
 * String reference. Demonstrate that this
 * reference is initialized by Java to null.
 ***********************************************/
package initialization;

public class E01_StringRefInitialization {
  String s;
  public static void main(String args[]) {
    E01_StringRefInitialization sri =
      new E01_StringRefInitialization();
    System.out.println("sri.s = " + sri.s);
  }
} /* Output:
sri.s = null
*///:~
```

Exercise 2

```
//: initialization/E02_StringInitialization.java
/****************** Exercise 2 *****************
 * Create a class with a String field initialized
 * at the point of definition, and another one
 * initialized by the constructor. What is the
 * difference between the two approaches?
 ***********************************************/
package initialization;
```

```
public class E02_StringInitialization {
  String s1 = "Initialized at definition";
  String s2;
  public E02_StringInitialization(String s2i) {
    s2 = s2i;
  }
  public static void main(String args[]) {
    E02_StringInitialization si =
      new E02_StringInitialization(
        "Initialized at construction");
    System.out.println("si.s1 = " + si.s1);
    System.out.println("si.s2 = " + si.s2);
  }
} /* Output:
si.s1 = Initialized at definition
si.s2 = Initialized at construction
*///:~
```

The **s1** field is initialized before the constructor is entered; technically, so is the **s2** field, which is set to **null** as the object is created. The more flexible **s2** field lets you choose what value to give it when you call the constructor, whereas **s1** always has the same value.

Exercise 3

```
//: initialization/E03_DefaultConstructor.java
/****************** Exercise 3 ******************
 * Create a class with a default constructor (one
 * that takes no arguments) that prints a
 * message. Create an object of this class.
 ************************************************/
package initialization;

public class E03_DefaultConstructor {
  E03_DefaultConstructor() {
    System.out.println("Default constructor");
  }
  public static void main(String args[]) {
    new E03_DefaultConstructor();
  }
} /* Output:
Default constructor
*///:~
```

Here we create the **E03_DefaultConstructor** object for the side effects of the constructor call, so there is no need to create and hold a reference to the object.

In practice, when an operation doesn't actually require an object, then a **static** utility method is more appropriate. (See *The meaning of **static*** section in *TIJ4* for more information.)

Exercise 4

```
//: initialization/E04_OverloadedConstructor.java
/****************** Exercise 4 *****************
 * Add an overloaded constructor to Exercise 3 that
 * takes a String argument and prints it along with
 * your message.
 *********************************************/
package initialization;

public class E04_OverloadedConstructor {
  E04_OverloadedConstructor() {
    System.out.println("Default constructor");
  }
  E04_OverloadedConstructor(String s) {
    System.out.println("String arg constructor");
    System.out.println(s);
  }
  public static void main(String args[]) {
    // Call default version:
    new E04_OverloadedConstructor();
    // Call overloaded version:
    new E04_OverloadedConstructor("Overloaded");
  }
} /* Output:
Default constructor
String arg constructor
Overloaded
*///:~
```

Exercise 5

```
//: initialization/E05_OverloadedDog.java
/****************** Exercise 5 *****************
 * Create a class called Dog with an overloaded
 * bark() method. Your method should be
 * overloaded based on various primitive data
 * types, and should print different types of barking,
 * howling, etc., depending on which overloaded
 * version is called. Write a main() that calls
```

```
 * all the different versions.
 *********************************************/
package initialization;

class Dog {
  public void bark() {
    System.out.println("Default bark!");
  }
  public void bark(int i) {
    System.out.println("int bark = howl");
  }
  public void bark(double d) {
    System.out.println("double bark = yip");
  }
  // Etc. ...
}

public class E05_OverloadedDog {
  public static void main(String args[]) {
    Dog dog = new Dog();
    dog.bark();
    dog.bark(1);
    dog.bark(1.1);
  }
} /* Output:
Default bark!
int bark = howl
double bark = yip
*///:~
```

As an additional challenge, write a class with a method **boolean print(int)** that prints a value and returns a **boolean**. Now overload the method to return a **long**. (This is similar to some questions on the Sun Java Certification Exam.)

Exercise 6

```
//: initialization/E06_SwappedArguments.java
/****************** Exercise 6 *****************
 * Modify Exercise 5 so two of the overloaded
 * methods have two arguments of two different
 * types, but in reversed order relative to each
 * other. Verify that this works.
 ***********************************************/
package initialization;

class Dog2 {
```

```
  public void bark(int i, double d) {
    System.out.println("int, double bark");
  }
  public void bark(double d, int i) {
    System.out.println("double, int bark");
  }
}

public class E06_SwappedArguments {
  public static void main(String args[]) {
    Dog2 dog = new Dog2();
    dog.bark(1, 2.2);
    dog.bark(2.2, 1);
  }
} /* Output:
int, double bark
double, int bark
*///:~
```

Note that not only the *type* of arguments *but also their order* distinguish an overloaded method. In the example, the two versions of **bark( )** are unique.

Exercise 7

```
//: initialization/E07_SynthesizedConstructor.java
/****************** Exercise 7 ******************
 * Create a class without a constructor, then
 * create an object of that class in main() to
 * verify that the default constructor is
 * automatically synthesized.
 ***********************************************/
package initialization;

public class E07_SynthesizedConstructor {
  public static void main(String args[]) {
    // Call the synthesized default constructor
    // for this class:
    new E07_SynthesizedConstructor();
  }
} ///:~
```

Because it's possible to call the constructor, you know it was created, even if you can't see it.

Exercise 8

```
//: initialization/E08_ThisMethodCall.java
/****************** Exercise 8 *****************
 * Create a class with two methods. Within the
 * first method call the second method twice to
 * see it work, the first time without using this,
 * and the second time using this.
 * (You should not use this form in practice.)
 *********************************************/
package initialization;

public class E08_ThisMethodCall {
  public void a() {
    b();
    this.b();
  }
  public void b() {
    System.out.println("b() called");
  }
  public static void main(String args[]) {
    new E08_ThisMethodCall().a();
  }
} /* Output:
b() called
b() called
*///:~
```

This exercise shows that **this** refers to the current object. Use the **this.b( )** style of method call only when necessary; otherwise you risk confusing the reader/maintainer of your code.

Exercise 9

```
//: initialization/E09_ThisConstructorCall.java
/****************** Exercise 9 *****************
 * Create a class with two (overloaded)
 * constructors. Using this, call the second
 * constructor inside the first one.
 *********************************************/
package initialization;

public class E09_ThisConstructorCall {
  public E09_ThisConstructorCall(String s) {
    System.out.println("s = " + s);
```

```
  }
  public E09_ThisConstructorCall(int i) {
    this("i = " + i);
  }
  public static void main(String args[]) {
    new E09_ThisConstructorCall("String call");
    new E09_ThisConstructorCall(47);
  }
} /* Output:
s = String call
s = i = 47
*///:~
```

Here's a situation where you are forced to use the **this** keyword.

Exercise 10

```
//: initialization/E10_FinalizeCall.java
/****************** Exercise 10 *****************
 * Create a class with a finalize() method that
 * prints a message. In main(), create an object
 * of your class. Explain the behavior of your
 * program.
 ************************************************/
package initialization;

public class E10_FinalizeCall {
  protected void finalize() {
    System.out.println("finalize() called");
  }
  public static void main(String args[]) {
    new E10_FinalizeCall();
  }
} ///:~
```

You probably won't see the finalizer called because the program doesn't usually generate enough garbage for the collector to run.

Exercise 11

```
//: initialization/E11_FinalizeAlwaysCalled.java
/****************** Exercise 11 ****************
 * Modify Exercise 10 so your
 * finalize() will always be called.
 ************************************************/
```

```
package initialization;

public class E11_FinalizeAlwaysCalled {
  protected void finalize() {
    System.out.println("finalize() called");
  }
  public static void main(String args[]) {
    new E11_FinalizeAlwaysCalled();
    System.gc();
    System.runFinalization();
  }
} /* Output:
finalize() called
*///:~
```

Calling **System.gc( )** and **System.runFinalization( )** in sequence will probably *but not necessarily* call your finalizer (The behavior of finalize has been uncertain from one version of JDK to another.) The call to these methods is just a request; it doesn't ensure the finalizer will actually run. Ultimately, *nothing* guarantees that **finalize( )** will be called.

Exercise 12

```
//: initialization/E12_TankWithTerminationCondition.java
/********************* Exercise 12 *******************
 * Create a class called Tank that can be filled
 * and emptied, with a termination condition that it
 * must be empty when the object is cleaned up.
 * Write a finalize() that verifies this termination
 * condition. In main(), test the possible
 * scenarios that can occur when you use Tank.
 ****************************************************/
package initialization;

class Tank {
  static int counter;
  int id = counter++;
  boolean full;
  public Tank() {
    System.out.println("Tank " + id + " created");
    full = true;
  }
  public void empty() { full = false; }
  protected void finalize() {
    if(full)
      System.out.println(
```

```
        "Error: tank " + id + " must be empty at cleanup");
    else
      System.out.println("Tank " + id + " cleaned up OK");
  }
  public String toString() { return "Tank " + id; }
}

public class E12_TankWithTerminationCondition {
  public static void main(String args[]) {
    new Tank().empty();
    new Tank();
    // Don't empty the second one
    System.gc(); // Force finalization?
    System.runFinalization();
  }
} /* Output:
Tank 0 created
Tank 1 created
Error: tank 1 must be empty at cleanup
Tank 0 cleaned up OK
*///:~
```

We created no references to the two instances of type **Tank**, because those references would be in scope when **System.gc( )** was called so they wouldn't be cleaned up, thus they wouldn't be finalized. Another option is to set references to zero when you want them to be garbage-collected. Modify the above example to try this method.

You can never be sure finalizers will be called, so their utility is limited. A finalizer can, for example, check the state of objects when it does run, to ensure they've been cleaned up properly.

Exercise 13

```
//: initialization/E13_LeftToReader.java
/****************** Exercise 13 *****************
 * Comment the line marked (1) in
 * ExplicitStatic.java and verify that the static
 * initialization clause is not called. Now
 * uncomment one of the lines marked (2) and
 * verify that the static initialization clause
 * is called. Finally, uncomment the other line marked
 * (2) and verify that static initialization
 * occurs only once.
 ***********************************************/
package initialization;
```

```
public class E13_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
} ///:~
```

Exercise 14

```
//: initialization/E14_StaticStringInitialization.java
/****************** Exercise 14 *****************
 * Create a class with a static String field that
 * is initialized at the point of definition, and
 * another one initialized by the static
 * block. Add a static method that prints both
 * fields and demonstrates that they are both
 * initialized before they are used.
 *********************************************/
package initialization;

public class E14_StaticStringInitialization {
  static String s1 = "Initialized at definition";
  static String s2;
  static { s2 = "Initialized in static block"; }
  public static void main(String args[]) {
    System.out.println("s1 = " + s1);
    System.out.println("s2 = " + s2);
  }
} /* Output:
s1 = Initialized at definition
s2 = Initialized in static block
*///:~
```

main( ) is a **static** method, so we used it to print the values.

Exercise 15

```
//: initialization/E15_StringInstanceInitialization.java
/****************** Exercise 15 ****************
 * Create a class with a String that is
 * initialized using "instance initialization."
 *********************************************/
package initialization;

public class E15_StringInstanceInitialization {
```

```
  String s;
  { s = "'instance initialization'"; }
  public E15_StringInstanceInitialization() {
    System.out.println("Default constructor; s = " + s);
  }
  public E15_StringInstanceInitialization(int i) {
    System.out.println("int constructor; s = " + s);
  }
  public static void main(String args[]) {
    new E15_StringInstanceInitialization();
    new E15_StringInstanceInitialization(1);
  }
} /* Output:
Default constructor; s = 'instance initialization'
int constructor; s = 'instance initialization'
*///:~
```

When you run this program, you'll see that instance initialization occurs before either of the two constructors.

Exercise 16

```
//: initialization/E16_StringArray.java
/****************** Exercise 16 *****************
 * Assign a string to each element of an array of
 * String objects. Print the array using a for loop.
 ************************************************/
package initialization;

public class E16_StringArray {
  public static void main(String args[]) {
    // Doing it the hard way:
    String sa1[] = new String[4];
    sa1[0] = "These";
    sa1[1] = "are";
    sa1[2] = "some";
    sa1[3] = "strings";
    for(int i = 0; i < sa1.length; i++)
      System.out.println(sa1[i]);
    // Using aggregate initialization to
    // make it easier:
    String sa2[] = {
      "These", "are", "some", "strings"
    };
    for(int i = 0; i < sa2.length; i++)
      System.out.println(sa2[i]);
```

```
  }
} /* Output:
These
are
some
strings
These
are
some
strings
*///:~
```

The solution shows both ways to solve the exercise: you can either create the array object and assign a string into each slot by hand, or use *aggregate initialization*, which creates and initializes the array object in a single step.

Exercise 17

```
//: initialization/E17_ObjectReferences.java
/****************** Exercise 17 *****************
 * Create a class with a constructor that takes
 * a String argument. During construction, print
 * the argument. Create an array of object
 * references to this class, but don't
 * create objects to assign into the
 * array. When you run the program, notice
 * whether the initialization messages from the
 * constructor calls are printed.
 ***********************************************/
package initialization;

class Test {
  Test(String s) {
    System.out.println("String constructor; s = " + s);
  }
}
public class E17_ObjectReferences {
  // You can define the array as a field in the class:
  Test[] array1 = new Test[5];
  public static void main(String args[]) {
    // Or as a temporary inside main:
    Test[] array2 = new Test[5];
  }
} ///:~
```

This code creates only the array, not the objects that go into it. You don't see initialization messages in **Test**'s constructors because no instances of that class exist.

Exercise 18

```
//: initialization/E18_ObjectArray.java
/****************** Exercise 18 *****************
 * Create objects to attach to the array of
 * references for Exercise 17.
 ************************************************/
package initialization;

public class E18_ObjectArray {
  public static void main(String args[]) {
    Test[] array = new Test[5];
    for(int i = 0; i < array.length; i++)
      array[i] = new Test(Integer.toString(i));
  }
} /* Output:
String constructor; s = 0
String constructor; s = 1
String constructor; s = 2
String constructor; s = 3
String constructor; s = 4
*///:~
```

Integer.toString( ) returns a **String** object representing the specified integer. Can you find a way to convert an integer into a **String** without using this utility method? (*Hint:* Recall previous discussions of the overloaded **String** operator +.)

Exercise 19

```
//: initialization/E19_VarargStringArray.java
/****************** Exercise 19 *****************
 * Write a method that takes a vararg String
 * array. Verify that you can pass either a
 * comma-separated list of Strings or a
 * String[] into this method.
 ************************************************/
package initialization;

public class E19_VarargStringArray {
```

```
  static void printStrings(String... strs) {
    for(String s : strs)
        System.out.println(s);
  }
  public static void main(String args[]) {
    printStrings("These", "are", "some", "strings");
    printStrings(
      new String[] { "These", "are", "some", "strings" }
    );
  }
} /* Output:
These
are
some
strings
These
are
some
strings
*///:~
```

Exercise 20

```
//: initialization/E20_VarargMain.java
// {Args: These, are, some, strings}
/****************** Exercise 20 *****************
 * Create a main() that uses varargs instead
 * of the ordinary main() syntax. Print all the
 * elements in the resulting args array. Test it
 * with various numbers of command-line
 * arguments.
 ***********************************************/
package initialization;

public class E20_VarargMain {
  public static void main(String... args) {
    E19_VarargStringArray.printStrings(args);
  }
} /* Output:
These,
are,
some,
strings
*///:~
```

Exercise 21

```
//: initialization/E21_PaperCurrencyTypesEnum.java
/****************** Exercise 21 ****************
 * Create an enum of the six lowest denominations
 * of paper currency. Loop through the values()
 * and print each value and its ordinal().
 ***********************************************/
package initialization;

enum PaperCurrencyTypes {
  ONE, TWO, FIVE, TEN, TWENTY, FIFTY
}

public class E21_PaperCurrencyTypesEnum {
  public static void main(String args[]) {
    for(PaperCurrencyTypes s : PaperCurrencyTypes.values())
      System.out.println(s + ", ordinal " + s.ordinal());
  }
} /* Output:
ONE, ordinal 0
TWO, ordinal 1
FIVE, ordinal 2
TEN, ordinal 3
TWENTY, ordinal 4
FIFTY, ordinal 5
*///:~
```

Exercise 22

```
//: initialization/E22_PaperCurrencyTypesEnum2.java
/****************** Exercise 22 ****************
 * Write a switch statement for the enum in
 * Exercise 21. For each case, output a
 * description of that particular currency.
 ***********************************************/
package initialization;
import static net.mindview.util.Print.*;

public class E22_PaperCurrencyTypesEnum2 {
  static void describe(PaperCurrencyTypes pct) {
    printnb(pct + " has a portrait of ");
    switch(pct) {
      case ONE:    print("George Washington");
                   break;
```

```
      case TWO:    print("Thomas Jefferson");
                   break;
      case FIVE:   print("Abraham Lincoln");
                   break;
      case TEN:    print("Alexander Hamilton");
                   break;
      case TWENTY: print("Andrew Jackson");
                   break;
      case FIFTY:  print("U.S. Grant");
                   break;
    }
  }
  public static void main(String args[]) {
    for(PaperCurrencyTypes s : PaperCurrencyTypes.values())
      describe(s);
  }
} /* Output:
ONE has a portrait of George Washington
TWO has a portrait of Thomas Jefferson
FIVE has a portrait of Abraham Lincoln
TEN has a portrait of Alexander Hamilton
TWENTY has a portrait of Andrew Jackson
FIFTY has a portrait of U.S. Grant
*///:~
```

Access Control

Exercise 1

```
//: access/local/E01_PackagedClass.java
/****************** Exercise 1 ****************
 * Create a class in a package. Create an
 * instance of your class outside of that package.
 *********************************************/
package access.local;

public class E01_PackagedClass {
} ///:~
```

Create the above file in the **access/local** subdirectory. (Be sure the file is in a directory that starts at a **CLASSPATH** location, then continues into **access/local**.) Then create the following file in the **access** directory, which is above the **local** directory and thus outside of the **access.local** package:

```
//: access/E01_ForeignClass.java
package access;

public class E01_ForeignClass {
  public static void main(String[] args) {
    new access.local.E01_PackagedClass();
  }
} ///:~
```

Exercise 2

```
//: access/E02_LeftToReader.java
/****************** Exercise 2 ******************
 * Turn the code fragments in the "Collisions"
 * section into a program, and  verify that
 * collisions do in fact occur.
 *********************************************/
package access;

public class E02_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
```

```
} ///:~
```

Exercise 3

```
//: access/debug/E03_Debug.java
/****************** Exercise 3 ****************
 * Create two packages: debug and debugoff,
 * containing an identical class with a debug()
 * method. The first version displays its String
 * argument to the console, the second does nothing.
 * Import the class into a test program
 * using a static import line, and demonstrate
 * the conditional compilation effect.
 *********************************************/
package access.debug;

public class E03_Debug {
  public static void debug(String msg) {
    System.out.println("Message: " + msg);
  }
} ///:~
```

```
//: access/debugoff/E03_Debug.java
package access.debugoff;

public class E03_Debug {
  public static void debug(String msg) {}
} ///:~
```

```
//: access/E03_DebugApp.java
package access;
import static access.debug.E03_Debug.*;

public class E03_DebugApp {
   public static void main(String[] args) {
      debug("DEBUG VERSION");
   }
} /* Output:
Message: DEBUG VERSION
*///:~
```

```
//: access/E03_ReleaseApp.java
package access;
import static access.debugoff.E03_Debug.*;

public class E03_ReleaseApp {
   public static void main(String[] args) {
```

```
      debug("RELEASE VERSION");
    }
} ///:~
```

The release version **E03_ReleaseApp** prints no message.

Exercise 4

```
//: access/local/E04_PackagedClass.java
/****************** Exercise 4 ****************
 * Show that protected methods have package
 * access but are not public.
 *********************************************/
package access.local;

public class E04_PackagedClass {
  protected static void greeting() {
    System.out.println("Hello client programmer!");
  }
} ///:~
```

```
//: access/local/E04_ConsumerInSamePackage.java
package access.local;

public class E04_ConsumerInSamePackage {
  public static void main(String[] args) {
    E04_PackagedClass.greeting();
  }
} /* Output:
Hello client programmer!
*///:~
```

```
//: access/E04_ForeignClass.java
// {CompileTimeError} to see results
package access;

public class E04_ForeignClass {
  public static void main(String[] args) {
    access.local.E04_PackagedClass.greeting();
  }
} ///:~
```

Explain why the compiler generates an error for **E04_ForeignClass.java**. Would making the **E04_ForeignClass** class part of the **access.local** package make a difference?

Solution: E04_PackagedClass is in its own package, and **greeting( )** is not a **public** method so is generally unavailable outside of the package **access.local**. If **E04_ForeignClass** were included in **access.local**, it would share the same package as **E04_PackagedClass.greeting( )**, and so could access it. (See Exercises 6 & 9.)

Exercise 5

```
//: access/local/E05_AccessControl.java
/****************** Exercise 5 ******************
 * Create a class with public, private,
 * protected, and package-access fields and
 * method members. Create an object of this class
 * and see what kind of compiler messages you get
 * when you try to access all the class members.
 * Remember that classes in the same directory
 * are part of the "default" package.
 *********************************************/
package access.local;

public class E05_AccessControl {
  public int a;
  private int b;
  protected int c;
  int d; // Package access
  public void f1() {}
  private void f2() {}
  protected void f3() {}
  void f4() {} // Package access
  public static void main(String args[]) {
    E05_AccessControl test = new E05_AccessControl();
    // No problem accessing everything inside
    // of main() for this class, since main()
    // is a member and therefore has access:
    test.a = 1;
    test.b = 2;
    test.c = 3;
    test.d = 4;
    test.f1();
    test.f2();
    test.f3();
    test.f4();
  }
} ///:~
```

You can see that **main( )** is a member and so has access to everything. If you create a separate class within the same package, that class cannot access the private members:

```
//: access/local/E05_Other.java
// A separate class in the same package cannot
// access private elements:
package access.local;

public class E05_Other {
  public E05_Other() {
    E05_AccessControl test = new E05_AccessControl();
    test.a = 1;
    //! test.b = 2; // Can't access:  private
    test.c = 3;
    test.d = 4;
    test.f1();
    //! test.f2(); // Can't access:  private
    test.f3();
    test.f4();
  }
} ///:~
```

When you create a class in a separate package (by either using a package statement or putting it in a different directory) then it can access *only* **public** members:

```
//: access/E05_Other.java
// A separate class in the other package cannot
// access private, protected and package elements:
package access;
import access.local.E05_AccessControl;

public class E05_Other {
  public E05_Other() {
    E05_AccessControl test = new E05_AccessControl();
    test.a = 1;
    //! test.b = 2; // Can't access: private
    //! test.c = 3; // Can't access: protected
    //! test.d = 4; // Can't access: package
    test.f1();
    //! test.f2(); // Can't access:  private
    //! test.f3(); // Can't access: protected
    //! test.f4(); // Can't access: package
  }
} ///:~
```

Exercise 6

```
//: access/E06_ProtectedManipulation.java
/****************** Exercise 6 *****************
 * Create one class with protected data, and a
 * second class in the same file with a method
 * that manipulates that protected data.
 ***********************************************/
package access;

class WithProtected {
  protected int i;
}

public class E06_ProtectedManipulation {
  public static void main(String args[]) {
    WithProtected wp  = new WithProtected();
    wp.i = 47;
    System.out.println("wp.i = " + wp.i);
  }
} /* Output:
wp.i = 47
*///:~
```

This exercise shows that **protected** also means "package access" (a.k.a. "friendly"). You can always access **protected** fields within the same package. As a further exercise, add a **protected** method to **WithProtected** and access it from within **E06_ProtectedManipulation**.

Exercise 7

```
//: access/e07/E07_Widget.java
/****************** Exercise 7 *****************
 * Create the library according to the code
 * fragments describing access and Widget. Create
 * a Widget in a class that is not part of the
 * access package.
 ***********************************************/
package access.e07;
import access.Widget;

public class E07_Widget {
  public static void main(String args[]) {
    new Widget();
  }
```

```
} /* Output:
Making a Widget
*///:~
```

```
//: access/Widget.java
package access;

public class Widget {
  public Widget() {
    System.out.println("Making a Widget");
  }
} ///:~
```

Exercise 8

```
//: access/E08_ConnectionManager.java
/****************** Exercise 8 *****************
 * Following the form of the example Lunch.java,
 * create a class called ConnectionManager that
 * manages a fixed array of Connection objects.
 * The client programmer must not be able to
 * create Connection objects, but only get them
 * via a static method in ConnectionManager.
 * ConnectionManager returns a null reference when
 * it runs out of objects. Test the classes in main().
 *********************************************/
package access;
import access.connection.*;

public class E08_ConnectionManager {
  public static void main(String args[]) {
    Connection c = ConnectionManager.getConnection();
    while(c != null) {
      System.out.println(c);
      c.doSomething();
      c = ConnectionManager.getConnection();
    }
  }
} /* Output:
Connection 0
Connection 1
Connection 2
Connection 3
Connection 4
Connection 5
Connection 6
```

```
Connection 7
Connection 8
Connection 9
*///:~
```

```
//: access/connection/Connection.java
package access.connection;

public class Connection {
  private static int counter = 0;
  private int id = counter++;
  Connection() {}
  public String toString() {
    return "Connection " + id;
  }
  public void doSomething() {}
} ///:~
```

```
//: access/connection/ConnectionManager.java
package access.connection;

public class ConnectionManager {
  private static Connection[] pool = new Connection[10];
  private static int counter = 0;
  static {
    for(int i = 0; i < pool.length; i++)
      pool[i] = new Connection();
  }
  // Very simple -- just hands out each one once:
  public static Connection getConnection() {
    if(counter < pool.length)
      return pool[counter++];
    return null;
  }
} ///:~
```

The **Connection** class identifies each **Connection** object with a **static int** called **counter**, which produces the identifier as part of its **toString( )** representation. **Connection** also has a **doSomething( )** method to indicate the task for which you created the **Connection** object.

Note that the constructor for **Connection** has package access; it is unavailable outside of this package, so the client programmer cannot access it to make instances of **Connection** directly. The only way to get **Connection** objects is through the **ConnectionManager**.

ConnectionManager initializes a **static** array of objects inside the *static clause* that is called only once when the class loads. (*TIJ4* covers static clauses in detail.)

Here is a more sophisticated connection manager that allows the client programmer to "check in" a connection when finished with it:

```
//: access/E08_ConnectionManager2.java
// Connections that can be checked in.
package access;
import access.connection2.*;

public class E08_ConnectionManager2 {
  public static void main(String args[]) {
    Connection[] ca = new Connection[10];
    // Use up all the connections
    for(int i = 0; i < 10; i++)
      ca[i] = ConnectionManager.getConnection();
    // Should produce "null" since there are no
    // more connections:
    System.out.println(ConnectionManager.getConnection());
    // Return connections, then get them out:
    for(int i = 0; i < 5; i++) {
      ca[i].checkIn();
      Connection c = ConnectionManager.getConnection();
      System.out.println(c);
      c.doSomething();
      c.checkIn();
    }
  }
} /* Output:
null
Connection 0
Connection 0
Connection 0
Connection 0
Connection 0
*///:~
```

```
//: access/connection2/Connection.java
package access.connection2;

public class Connection {
  private static int counter = 0;
  private int id = counter++;
  Connection() {}
  public String toString() {
```

```
    return "Connection " + id;
  }
  public void doSomething() {}
  public void checkIn() {
    ConnectionManager.checkIn(this);
  }
} ///:~
```

```
//: access/connection2/ConnectionManager.java
package access.connection2;

public class ConnectionManager {
  private static Connection[] pool = new Connection[10];
  static {
    for(int i = 0; i < pool.length; i++)
      pool[i] = new Connection();
  }
  // Produce the first available connection:
  public static Connection getConnection() {
    for(int i = 0; i < pool.length; i++)
      if(pool[i] != null) {
        Connection c = pool[i];
        pool[i] = null; // Indicates "in use"
        return c;
      }
    return null; // None left
  }
  public static void checkIn(Connection c) {
    for(int i = 0; i < pool.length; i++)
      if(pool[i] == null) {
        pool[i] = c; // Check it back in
        return;
      }
  }
} ///:~
```

When a **Connection** is checked out, its slot in **pool** is set to **null**. When the client programmer is done with the connection, **checkIn( )** returns it to the connection pool by assigning it to a **null** slot in **pool**.

However, there are all kinds of potential problems with this approach. What if a client checks in a **Connection** and then continues to use it, or checks in more than once? We address the problem of the connection pool more thoroughly in *Thinking in Patterns with Java* (available from www.MindView.net).

Exercise 9

(See *TIJ4* for problem description.)

Solution: PackagedClass is in its own package and is not a **public** class, so is unavailable outside of **package access.local**. If **Foreign** were also part of **access.local**, then it would be in the same package as **PackagedClass** and would have access to it.

Reusing Classes

Exercise 1

```
//: reusing/E01_Composition.java
/****************** Exercise 1 *****************
 * Create a simple class. Inside a second class,
 * define a reference to an object of the first
 * class. Use lazy initialization to instantiate
 * this object.
 ***********************************************/
package reusing;
import static net.mindview.util.Print.*;

class Simple {
  String s;
  public Simple(String si) { s = si; }
  public String toString() { return s; }
  public void setString(String sNew) { s = sNew; }
}

class Second {
  Simple simple;
  String s;
  public Second(String si) {
    s = si; // 'simple' not initialized
  }
  public void check() {
    if(simple == null)
      print("not initialized");
    else
      print("initialized");
  }
  private Simple lazy() {
    if(simple == null) {
      print("Creating Simple");
      simple = new Simple(s);
    }
    return simple;
  }
  public Simple getSimple() { return lazy(); }
  public String toString() {
```

```
    return lazy().toString();
  }
  public void setSimple(String sNew) {
    lazy().setString(sNew);
  }
}

public class E01_Composition {
  public static void main(String args[]) {
    Second second = new Second("Init String");
    second.check();
    print(second.getSimple());
    second.check();
    print(second); // toString() call
    second.setSimple("New String");
    print(second);
  }
} /* Output:
not initialized
Creating Simple
Init String
initialized
Init String
New String
*///:~
```

The **Simple** class has some data and methods. The **Second** class performs lazy initialization through the **lazy( )** method, which creates (if it hasn't been) the **Simple** object and then returns it. The **lazy( )** method is called by all the other methods to access the **Simple** object.

We added print statements to show when initialization occurs, and that it happens only once.

Exercise 2

```
//: reusing/E02_NewDetergent.java
/****************** Exercise 2 *****************
 * Inherit a new class from class Detergent.
 * Override scrub() and add a new method called
 * sterilize().
 ***********************************************/
package reusing;

class NewDetergent extends Detergent {
  public void scrub() {
```

```
    append(" NewDetergent.scrub()");
    super.scrub(); // Doesn't have to be first
  }
  public void sterilize() { append(" sterilize()"); }
}

public class E02_NewDetergent {
  public static void main(String args[]) {
    NewDetergent nd = new NewDetergent();
    nd.dilute();
    nd.scrub();
    nd.sterilize();
    System.out.println(nd);
  }
} /* Output:
Cleanser dilute() NewDetergent.scrub() Detergent.scrub()
scrub() sterilize()
*///:~
```

Exercise 3

```
//: reusing/E03_CartoonWithDefCtor.java
/****************** Exercise 3 ******************
 * Even if you don't create a constructor for
 * Cartoon(), the compiler will synthesize a
 * default constructor that calls the base-class
 * constructor. Prove that assertion.
 ***********************************************/
package reusing;

class CartoonWithDefCtor extends Drawing {
//!  CartoonWithDefCtor() {
//!    System.out.println("CartoonWithDefCtor constructor");
//!  }
}

public class E03_CartoonWithDefCtor {
  public static void main(String args[]) {
    new CartoonWithDefCtor ();
  }
} /* Output:
Art constructor
Drawing constructor
*///:~
```

We commented out the **CartoonWithDefCtor** constructor above. *TIJ4* shows this output by printing in the constructor.

The compiler synthesizes the default **CartoonWithDefCtor** constructor, in which it calls the base class **Drawing** default constructor, which in turn calls the base class **Art** default constructor.

The compiler ensures that a constructor is called. If you don't call a constructor, it calls the default constructor if available; however, a default constructor is *not* synthesized if you define any constructors but *not* the default.

Exercise 4

```
//: reusing/E04_ConstructorOrder.java
/****************** Exercise 4 ******************
 * Prove that base-class constructors are (a)
 * always called and (b) called before
 * derived-class constructors.
 ************************************************/
package reusing;

class Base1 {
  public Base1() { System.out.println("Base1"); }
}

class Derived1 extends Base1 {
  public Derived1() { System.out.println("Derived1"); }
}

class Derived2 extends Derived1 {
  public Derived2() { System.out.println("Derived2"); }
}

public class E04_ConstructorOrder {
  public static void main(String args[]) {
    new Derived2();
  }
} /* Output:
Base1
Derived1
Derived2
*///:~
```

Exercise 5

```
//: reusing/E05_SimpleInheritance.java
/****************** Exercise 5 *****************
 * Create classes A and B with default
 * constructors (empty argument lists) that
 * announce themselves. Inherit a new class
 * called C from A, and create a member of class
 * B inside C. Do not create a constructor for C.
 * Create an object of class C and observe the
 * results.
 ***********************************************/
package reusing;

class A {
  public A() { System.out.println("A()"); }
}

class B {
  public B() { System.out.println("B()"); }
}

class C extends A {
  B b = new B();
}

public class E05_SimpleInheritance {
  public static void main(String args[]) {
    new C();
  }
} /* Output:
A()
B()
*///:~
```

Here the compiler synthesizes a constructor for **C**, first calling the base-class constructor, then the member object constructors.

Exercise 6

```
//: reusing/E06_ChessWithoutDefCtor.java
// {CompileTimeError}
/****************** Exercise 6 *****************
 * If you don't call the base-class constructor
 * in BoardGame(), the compiler will respond
```

```
 * that it can't find a constructor of the form
 * Game(). The call to the base-class
 * constructor must be the first thing you do
 * in the derived-class constructor. (The compiler
 * will remind you if you get it wrong.)
 * Use Chess.java to prove those assertions.
 *********************************************/
package reusing;

class ChessWithoutDefCtor extends BoardGame {
  //ChessWithoutDefCtor () {
  //  System.out.println("ChessWithoutDefCtor constructor");
  //  super(11);
  //}
}

public class E06_ChessWithoutDefCtor {
  public static void main(String args[]) {
    new ChessWithoutDefCtor();
  }
} ///:~
```

BoardGame has no default constructor for **ChessWithoutDefCtor**, nor its own default constructor that the compiler can use to synthesize one, so the program won't compile. **BoardGame** defines a constructor that takes an argument, so the compiler cannot generate a default. Moreover, if you uncomment the default constructor definition for **ChessWithoutDefCtor**, the compiler will demand that you first call the base-class constructor in the derived-class constructor.

The **{CompileTimeError}** directive takes the compilation out of the build process because this compilation fails.

Exercise 7

```
//: reusing/E07_SimpleInheritance2.java
/****************** Exercise 7 ******************
 * Modify Exercise 5 so A and B have
 * constructors with arguments instead of default
 * constructors. Write a constructor for C and
 * perform all initialization within it.
 *********************************************/
package reusing;

class A2 {
  public A2(String s) { System.out.println("A2(): " + s); }
```

```
}

class B2 {
  public B2(String s) { System.out.println("2B(): " + s); }
}

class C2 extends A2 {
  B2 b;
  public C2(String s) {
    super(s);
    b = new B2(s);
  }
}

public class E07_SimpleInheritance2 {
  public static void main(String args[]) {
    new C2("Init string");
  }
} /* Output:
A2(): Init string
2B(): Init string
*///:~
```

We added the 2's to keep class names in the same directory from clashing.

Remember that **super** calls the base-class constructor and must be the first call in a derived-class constructor.

Exercise 8

```
//: reusing/E08_CallBaseConstructor.java
/****************** Exercise 8 *****************
 * Create a base class with only a non-default
 * constructor, and a derived class with both a
 * default (no-arg) and non-default constructor.
 * Call the base-class constructor in the
 * derived-class constructors.
 ***********************************************/
package reusing;

class BaseNonDefault {
  public BaseNonDefault(int i) {}
}

class DerivedTwoConstructors extends BaseNonDefault {
  public DerivedTwoConstructors() { super(47); }
```

```
  public DerivedTwoConstructors(int i) { super(i); }
}

public class E08_CallBaseConstructor {
  public static void main(String args[]) {
    new DerivedTwoConstructors();
    new DerivedTwoConstructors(74);
  }
} ///:~
```

Exercise 9

```
//: reusing/E09_ConstructorOrder2.java
/****************** Exercise 9 *****************
 * Create a class called Root and an instance of
 * each of three classes, Component1, Component2, and
 * Component3. Derive a class Stem from Root that
 * also contains an instance of each "component."
 * Default constructors for each class should
 * print a message about that class.
 ***********************************************/
package reusing;

class Component1 {
  public Component1() { System.out.println("Component1"); }
}

class Component2 {
  public Component2() { System.out.println("Component2"); }
}

class Component3 {
  public Component3() { System.out.println("Component3"); }
}

class Root {
  Component1 c1 = new Component1();
  Component2 c2 = new Component2();
  Component3 c3 = new Component3();
  public Root() { System.out.println("Root"); }
}

class Stem extends Root {
  Component1 c1 = new Component1();
  Component2 c2 = new Component2();
  Component3 c3 = new Component3();
```

```
  public Stem() { System.out.println("Stem"); }
}

public class E09_ConstructorOrder2 {
  public static void main(String args[]) {
    new Stem();
  }
} /* Output:
Component1
Component2
Component3
Root
Component1
Component2
Component3
Stem
*///:~
```

Exercise 10

```
//: reusing/E10_ConstructorOrder3.java
/****************** Exercise 10 *****************
 * Modify Exercise 9 so each class only has
 * non-default constructors.
 ************************************************/
package reusing;

class Component1b {
  public Component1b(int i) {
    System.out.println("Component1b " + i);
  }
}

class Component2b {
  public Component2b(int i) {
    System.out.println("Component2b " + i);
  }
}

class Component3b {
  public Component3b(int i) {
    System.out.println("Component3b " + i);
  }
}

class Rootb {
```

```
  Component1b c1 = new Component1b(1);
  Component2b c2 = new Component2b(2);
  Component3b c3 = new Component3b(3);
  public Rootb(int i) { System.out.println("Rootb"); }
}

class Stemb extends Rootb {
  Component1b c1 = new Component1b(4);
  Component2b c2 = new Component2b(5);
  Component3b c3 = new Component3b(6);
  public Stemb(int i) {
    super(i);
    System.out.println("Stemb");
  }
}

public class E10_ConstructorOrder3 {
  public static void main(String args[]) {
    new Stemb(47);
  }
} /* Output:
Component1b 1
Component2b 2
Component3b 3
Rootb
Component1b 4
Component2b 5
Component3b 6
Stemb
*///:~
```

We display the "Component" argument to clarify the order of constructor calls.

Exercise 11

```
//: reusing/E11_Delegation.java
/****************** Exercise 11 *****************
 * Modify Detergent.java so it uses delegation.
 ************************************************/
package reusing;
import static net.mindview.util.Print.*;

class DetergentDelegation {
  private Cleanser cleanser = new Cleanser();
  // Delegated methods:
  public void append(String a) { cleanser.append(a); }
```

```
  public void dilute() { cleanser.dilute(); }
  public void apply() { cleanser.apply(); }
  public String toString() { return cleanser.toString(); }
  public void scrub() {
    append(" DetergentDelegation.scrub()");
    cleanser.scrub();
  }
  public void foam() { append(" foam()"); }
  public static void main(String[] args) {
    DetergentDelegation x = new DetergentDelegation();
    x.dilute();
    x.apply();
    x.scrub();
    x.foam();
    print(x);
    print("Testing base class:");
    Cleanser.main(args);
  }
}

public class E11_Delegation {
  public static void main(String[] args) {
    DetergentDelegation.main(args);
  }
} /* Output:
Cleanser dilute() apply() DetergentDelegation.scrub()
scrub() foam()
Testing base class:
Cleanser dilute() apply() scrub()
*///:~
```

Exercise 12

```
//: reusing/E12_Dispose.java
/****************** Exercise 12 *****************
 * Add a proper hierarchy of dispose() methods to
 * all the classes in Exercise 9.
 ***********************************************/
package reusing;

class Component1c {
  public Component1c(int i) {
    System.out.println("Component1c");
  }
  public void dispose() {
    System.out.println("Component1c dispose");
```

```
  }
}

class Component2c {
  public Component2c(int i) {
    System.out.println("Component2c");
  }
  public void dispose() {
    System.out.println("Component2c dispose");
  }
}

class Component3c {
  public Component3c(int i) {
    System.out.println("Component3c");
  }
  public void dispose() {
    System.out.println("Component3c dispose");
  }
}

class Rootc {
  Component1c c1 = new Component1c(1);
  Component2c c2 = new Component2c(2);
  Component3c c3 = new Component3c(3);
  public Rootc(int i) { System.out.println("Rootc"); }
  public void dispose() {
    System.out.println("Rootc dispose");
    c3.dispose();
    c2.dispose();
    c1.dispose();
  }
}

class Stemc extends Rootc {
  Component1c c1 = new Component1c(4);
  Component2c c2 = new Component2c(5);
  Component3c c3 = new Component3c(6);
  public Stemc(int i) {
    super(i);
    System.out.println("Stemc");
  }
  public void dispose() {
    System.out.println("Stemc dispose");
    c3.dispose();
    c2.dispose();
    c1.dispose();
```

```
    super.dispose();
  }
}

public class E12_Dispose {
  public static void main(String args[]) {
    new Stemc(47).dispose();
  }
} /* Output:
Component1c
Component2c
Component3c
Rootc
Component1c
Component2c
Component3c
Stemc
Stemc dispose
Component3c dispose
Component2c dispose
Component1c dispose
Rootc dispose
Component3c dispose
Component2c dispose
Component1c dispose
*///:~
```

Remember, it's important to call the **dispose( )** methods in the reverse order of initialization.

Exercise 13

```
//: reusing/E13_InheritedOverloading.java
/****************** Exercise 13 *****************
 * Create a class with a method that is
 * overloaded three times. Inherit a new class,
 * add a new overloading of the method, and show
 * that all four methods are available in the
 * derived class.
 ************************************************/
package reusing;

class ThreeOverloads {
  public void f(int i) {
    System.out.println("f(int i)");
  }
```

```
  public void f(char c) {
    System.out.println("f(char c)");
  }
  public void f(double d) {
    System.out.println("f(double d)");
  }
}

class MoreOverloads extends ThreeOverloads {
  public void f(String s) {
    System.out.println("f(String s)");
  }
}

public class E13_InheritedOverloading {
  public static void main(String args[]) {
    MoreOverloads mo = new MoreOverloads();
    mo.f(1);
    mo.f('c');
    mo.f(1.1);
    mo.f("Hello");
  }
} /* Output:
f(int i)
f(char c)
f(double d)
f(String s)
*///:~
```

Exercise 14

```
//: reusing/E14_ServicableEngine.java
/****************** Exercise 14 *****************
 * In Car.java add a service() method to Engine
 * and call this method in main().
 ***********************************************/
package reusing;

class ServicableEngine extends Engine {
  public void service() {}
}

class ServicableCar {
  public ServicableEngine engine = new ServicableEngine();
  public Wheel[] wheel = new Wheel[4];
  public Door
```

```
    left = new Door(),
    right = new Door(); // 2-door
  public ServicableCar() {
    for(int i = 0; i < 4; i++)
      wheel[i] = new Wheel();
  }
}

public class E14_ServicableEngine {
  public static void main(String[] args) {
    ServicableCar car = new ServicableCar();
    car.left.window.rollup();
    car.wheel[0].inflate(72);
    car.engine.service();
  }
} ///:~
```

Exercise 15

```
//: reusing/protect/E15_Protected.java
/****************** Exercise 15 *****************
 * Create a class with a protected method inside
 * a package. Try to call the protected method
 * outside the package, and explain the results.
 * Now inherit from your class and call the
 * protected method from inside a method of your
 * derived class.
 ***********************************************/
package reusing.protect;

public class E15_Protected {
  protected void f() {}
} ///:~
```

```
//: reusing/E15_ProtectedTest.java
package reusing;
import reusing.protect.*;

class Derived extends E15_Protected {
  public void g() {
    f(); // Accessible in derived class
  }
}

public class E15_ProtectedTest {
  public static void main(String args[]) {
```

```
    //! new E15_Protected().f(); // Cannot access
    new Derived().g();
  }
} ///:~
```

Outside the package, you can only access a **protected** member inside an inherited class.

Exercise 16

```
//: reusing/E16_Frog.java
/****************** Exercise 16 *****************
 * Create a class called Amphibian. From it,
 * inherit a class from it called Frog. Put
 * appropriate methods in the base class. In
 * main(), create a Frog, upcast it to Amphibian,
 * and demonstrate that all the methods still work.
 ************************************************/
package reusing;

class Amphibian {
  public void moveInWater() {
    System.out.println("Moving in Water");
  }
  public void moveOnLand() {
    System.out.println("Moving on Land");
  }
}

class Frog extends Amphibian {}

public class E16_Frog {
  public static void main(String args[]) {
    Amphibian a = new Frog();
    a.moveInWater();
    a.moveOnLand();
  }
} /* Output:
Moving in Water
Moving on Land
*///:~
```

Exercise 17

```
//: reusing/E17_Frog2.java
```

```
/****************** Exercise 17 *****************
 * Modify Exercise 16 so Frog overrides the
 * method definitions from the base class
 * (provides new definitions using the same
 * method signatures). Note what happens in
 * main().
 ************************************************/
package reusing;

class Frog2 extends Amphibian {
  public void moveInWater() {
    System.out.println("Frog swimming");
  }
  public void moveOnLand() {
    System.out.println("Frog jumping");
  }
}

public class E17_Frog2 {
  public static void main(String args[]) {
    Amphibian a = new Frog2();
    a.moveInWater();
    a.moveOnLand();
  }
} /* Output:
Frog swimming
Frog jumping
*///:~
```

Since the compiler has a reference to an **Amphibian**, you might guess it will call the **Amphibian** methods. Instead, it calls the **Frog2** methods. Since **a** is indeed a reference to a **Frog2**, this is the appropriate result. That's polymorphism: The right behavior happens even if you are talking to a base-class reference.

Exercise 18

```
//: reusing/E18_FinalFields.java
/****************** Exercise 18 *****************
 * Create a class with a static final field and a
 * final field and demonstrate the difference
 * between the two.
 ************************************************/
package reusing;

class SelfCounter {
  private static int count;
```

```
  private int id = count++;
  public String toString() { return "SelfCounter " + id; }
}

class WithFinalFields {
  final SelfCounter scf = new SelfCounter();
  static final SelfCounter scsf = new SelfCounter();
  public String toString() {
    return "scf = " + scf + "\nscsf = " + scsf;
  }
}

public class E18_FinalFields {
  public static void main(String args[]) {
    System.out.println("First object:");
    System.out.println(new WithFinalFields());
    System.out.println("Second object:");
    System.out.println(new WithFinalFields());
  }
} /* Output:
First object:
scf = SelfCounter 1
scsf = SelfCounter 0
Second object:
scf = SelfCounter 2
scsf = SelfCounter 0
*///:~
```

Because class loading initializes the **static final,** it has the same value in both instances of **WithFinalFields**, whereas the regular **final's** values are different for each instance.

Exercise 19

```
//: reusing/E19_BlankFinalField.java
/****************** Exercise 19 *****************
 * Create a class with a blank final reference to
 * an object. Perform initialization of the
 * blank final inside all constructors.
 * Demonstrate that the final must
 * be initialized before use, and cannot
 * be changed once initialized.
 ***********************************************/
package reusing;

class WithBlankFinalField {
```

```
  private final Integer i;
  // Without this constructor, you'll get a compiler error:
  // "variable i might not have been initialized"
  public WithBlankFinalField(int ii) {
    i = new Integer(ii);
  }
  public Integer geti() {
    // This won't compile. The error is:
    // "cannot assign a value to final variable i"
    // if(i == null)
    //   i = new Integer(47);
    return i;
  }
}

public class E19_BlankFinalField {
  public static void main(String args[]) {
    WithBlankFinalField wbff = new WithBlankFinalField(10);
    System.out.println(wbff.geti());
  }
} /* Output:
10
*///:~
```

Exercise 20

```
//: reusing/E20_OverrideAnnotation.java
// {CompileTimeError}
/****************** Exercise 20 ****************
 * Show that the @Override annotation solves the
 * problem from the "final and private" section.
 ***********************************************/
package reusing;
import static net.mindview.util.Print.*;

class OverridingPrivateE20 extends WithFinals {
  @Override private final void f() {
    print("OverridingPrivateE20.f()");
  }
  @Override private void g() {
    print("OverridingPrivateE20.g()");
  }
}

class OverridingPrivate2E20 extends OverridingPrivateE20 {
  @Override public final void f() {
```

```
    print("OverridingPrivate2E20.f()");
  }
  @Override public void g() {
    print("OverridingPrivate2E20.g()");
  }
}

public class E20_OverrideAnnotation {
  public static void main(String[] args) {
    OverridingPrivate2E20 op2 = new OverridingPrivate2E20();
    op2.f();
    op2.g();
  }
} ///:~
```

Exercise 21

```
//: reusing/E21_FinalMethod.java
// {CompileTimeError}
/****************** Exercise 21 ****************
 * Create a class with a final method. Inherit
 * from that class and attempt to override that
 * method.
 ***********************************************/
package reusing;

class WithFinalMethod {
  final void f() {}
}

public class E21_FinalMethod extends WithFinalMethod {
  void f() {}
  public static void main(String args[]) {}
} ///:~
```

Exercise 22

```
//: reusing/E22_FinalClass.java
// {CompileTimeError}
/****************** Exercise 22 ****************
 * Create a final class and attempt to inherit
 * from it.
 ***********************************************/
package reusing;
```

```
final class FinalClass {}

public class E22_FinalClass extends FinalClass {
  public static void main(String args[]) {}
} ///:~
```

Exercise 23

```
//: reusing/E23_ClassLoading.java
/****************** Exercise 23 ****************
 * Prove that class loading takes place only
 * once and may be caused by either the creation
 * of the first instance of that class or by
 * accessing a static member.
 ***********************************************/
package reusing;

class LoadTest {
  // The static clause is executed
  // upon class loading:
  static {
    System.out.println("Loading LoadTest");
  }
  static void staticMember() {}
}

public class E23_ClassLoading {
  public static void main(String args[]) {
    System.out.println("Calling static member");
    LoadTest.staticMember();
    System.out.println("Creating an object");
    new LoadTest();
  }
} /* Output:
Calling static member
Loading LoadTest
Creating an object
*///:~
```

Now modify the code so object creation occurs before the **static** member call to see that object creation loads the object. Remember, a constructor is a **static** method, even though you don't define it using the **static** keyword.

Exercise 24

```
//: reusing/E24_JapaneseBeetle.java
/****************** Exercise 24 ****************
 * In Beetle.java, inherit a specific type of
 * beetle from class Beetle, following the same
 * format as the existing classes. Trace and
 * explain the output.
 **********************************************/
package reusing;
import static net.mindview.util.Print.*;

class JapaneseBeetle extends Beetle {
  int m = printInit("JapaneseBeetle.m initialized");
  JapaneseBeetle() {
    print("m = " + m);
    print("j = " + j);
  }
  static int x3 =
    printInit("static JapaneseBeetle.x3 initialized");
}

public class E24_JapaneseBeetle {
  public static void main(String args[]) {
    new JapaneseBeetle();
  }
} /* Output:
static Insect.x1 initialized
static Beetle.x2 initialized
static JapaneseBeetle.x3 initialized
i = 9, j = 0
Beetle.k initialized
k = 47
j = 39
JapaneseBeetle.m initialized
m = 47
j = 39
*///:~
```

Loading the class initializes the **static** variables. The base class loads first, then the next-derived class, and finally the most-derived class. This creates the object and initializes the non-**static** members, also starting at the root class.

Polymorphism

Exercise 1

```
//: polymorphism/E01_Upcasting.java
/****************** Exercise 1 ****************
 * Create a Cycle class, with subclasses
 * Unicycle, Bicycle, and Tricycle. Demonstrate
 * that an instance of each type can be upcast
 * to Cycle via a ride() method.
 *********************************************/
package polymorphism;
import polymorphism.cycle.*;

public class E01_Upcasting {
  public static void ride(Cycle c) {}
  public static void main(String[] args) {
    ride(new Cycle());    // No upcasting
    ride(new Unicycle()); // Upcast
    ride(new Bicycle());  // Upcast
    ride(new Tricycle()); // Upcast
  }
} ///:~
```

```
//: polymorphism/cycle/Cycle.java
package polymorphism.cycle;

public class Cycle {
} ///:~
```

```
//: polymorphism/cycle/Unicycle.java
package polymorphism.cycle;

public class Unicycle extends Cycle {
} ///:~
```

```
//: polymorphism/cycle/Bicycle.java
package polymorphism.cycle;

public class Bicycle extends Cycle {
} ///:~
```

```
//: polymorphism/cycle/Tricycle.java
package polymorphism.cycle;
```

```
public class Tricycle extends Cycle {
} ///:~
```

We created the classes in individual files because they are used in a later example.

Exercise 2

```
//: polymorphism/E02_Shapes.java
/****************** Exercise 2 ****************
 * Add the @Override annotation to the shapes
 * example.
 *********************************************/
package polymorphism;
import polymorphism.shape.Shape;
import polymorphism.oshape.*;

public class E02_Shapes {
  private static RandomShapeGenerator gen =
    new RandomShapeGenerator();
  public static void main(String[] args) {
    Shape[] shapes = new Shape[9];
    // Fill up the array with shapes:
    for(int i = 0; i < shapes.length; i++)
      shapes[i] = gen.next();
    // Make polymorphic method calls:
    for(Shape shape : shapes)
      shape.draw();
  }
} /* Output:
Triangle.draw()
Triangle.draw()
Square.draw()
Triangle.draw()
Square.draw()
Triangle.draw()
Square.draw()
Triangle.draw()
Circle.draw()
*///:~
```

```
//: polymorphism/oshape/RandomShapeGenerator.java
// A "factory" that randomly creates shapes.
package polymorphism.oshape;
import polymorphism.shape.Shape;
```

```
import java.util.*;

public class RandomShapeGenerator {
  private Random rand = new Random(47);
  public Shape next() {
    switch(rand.nextInt(3)) {
      default:
      case 0: return new Circle();
      case 1: return new Square();
      case 2: return new Triangle();
    }
  }
} ///:~
```

```
//: polymorphism/oshape/Square.java
package polymorphism.oshape;
import polymorphism.shape.Shape;
import static net.mindview.util.Print.*;

public class Square extends Shape {
  @Override public void draw() { print("Square.draw()"); }
  @Override public void erase() { print("Square.erase()"); }
} ///:~
```

```
//: polymorphism/oshape/Triangle.java
package polymorphism.oshape;
import polymorphism.shape.Shape;
import static net.mindview.util.Print.*;

public class Triangle extends Shape {
  @Override public void draw() { print("Triangle.draw()"); }
  @Override public void erase() {
    print("Triangle.erase()");
  }
} ///:~
```

```
//: polymorphism/oshape/Circle.java
package polymorphism.oshape;
import polymorphism.shape.Shape;
import static net.mindview.util.Print.*;

public class Circle extends Shape {
  @Override public void draw() { print("Circle.draw()"); }
  @Override public void erase() { print("Circle.erase()"); }
} ///:~
```

Exercise 3

```
//: polymorphism/E03_NewShapeMethod.java
/****************** Exercise 3 *****************
 * Add a new method in the base class of
 * Shapes.java that prints a message, but don't
 * override it in the derived classes. Explain
 * what happens. Now override it in only one of the
 * derived classes and see what happens. Finally,
 * override it in all the derived classes.
 ***********************************************/
package polymorphism;
import polymorphism.newshape.*;

public class E03_NewShapeMethod {
  public static void main(String args[]) {
    Shape[] shapes = {
      new Circle(), new Square(), new Triangle(),
    };
    // Make polymorphic method calls:
    for(Shape shape : shapes) {
      shape.draw();
      shape.erase();
      shape.msg();
    }
  }
} /* Output:
Circle.draw()
Circle.erase()
Circle.msg()
Square.draw()
Square.erase()
Square.msg()
Triangle.draw()
Triangle.erase()
Triangle.msg()
*///:~
```

```
//: polymorphism/newshape/Shape.java
package polymorphism.newshape;
import static net.mindview.util.Print.*;

public class Shape {
  public void draw() {}
  public void erase() {}
  public void msg() { print("Base class msg()"); }
```

```
} ///:~
```

```
//: polymorphism/newshape/Circle.java
package polymorphism.newshape;
import static net.mindview.util.Print.*;

public class Circle extends Shape {
  public void draw() { print("Circle.draw()"); }
  public void erase() { print("Circle.erase()"); }
  public void msg() { print("Circle.msg()"); }
} ///:~
```

```
//: polymorphism/newshape/Square.java
package polymorphism.newshape;
import static net.mindview.util.Print.*;

public class Square extends Shape {
  public void draw() { print("Square.draw()"); }
  public void erase() { print("Square.erase()"); }
  public void msg() { print("Square.msg()"); }
} ///:~
```

```
//: polymorphism/newshape/Triangle.java
package polymorphism.newshape;
import static net.mindview.util.Print.*;

public class Triangle extends Shape {
  public void draw() { print("Triangle.draw()"); }
  public void erase() { print("Triangle.erase()"); }
  public void msg() { print("Triangle.msg()"); }
} ///:~
```

This final version overrides **print( )** in all classes.

Defining **print( )** in only the base class results in:

```
Circle.draw()
Circle.erase()
Base class msg()
Square.draw()
Square.erase()
Base class msg()
Triangle.draw()
Triangle.erase()
Base class msg()
```

Nothing overrides the base-class definition, so it is used everywhere. Overriding **print( )** in **Circle** produces:

```
Circle.draw()
Circle.erase()
Circle.msg()
Square.draw()
Square.erase()
Base class msg()
Triangle.draw()
Triangle.erase()
Base class msg()
```

Anywhere we override, we use that version, as in **Circle**; otherwise we use the default base-class version.

Exercise 4

```
//: polymorphism/E04_NewShapeType.java
/****************** Exercise 4 *****************
 * Add a new type of Shape to Shapes.java and
 * verify in main() that polymorphism works for
 * your new type as it does in the old types.
 ***********************************************/
package polymorphism;
import polymorphism.newshape.*;

public class E04_NewShapeType {
  public static void main(String args[]) {
    Shape[] shapes = {
      new Circle(), new Square(), new Triangle(),
      new Tetrahedron()
    };
    // Make polymorphic method calls:
    for(Shape shape : shapes) {
      shape.draw();
      shape.erase();
      shape.msg();
    }
  }
} /* Output:
Circle.draw()
Circle.erase()
Circle.msg()
Square.draw()
Square.erase()
Square.msg()
Triangle.draw()
Triangle.erase()
```

```
Triangle.msg()
Tetrahedron.draw()
Tetrahedron.erase()
Tetrahedron.msg()
*///:~
```

```
//: polymorphism/newshape/Tetrahedron.java
package polymorphism.newshape;
import static net.mindview.util.Print.*;

public class Tetrahedron extends Shape {
  public void draw() { print("Tetrahedron.draw()"); }
  public void erase() { print("Tetrahedron.erase()"); }
  public void msg() { print("Tetrahedron.msg()"); }
} ///:~
```

The other shape definitions are in the same package so we just add the new shape and override the methods. The code in the **for** loop is unchanged from the previous example.

Exercise 5

```
//: polymorphism/E05_Wheels.java
/****************** Exercise 5 ****************
 * Starting from Exercise 1, add a wheels()
 * method in Cycle, which returns the number of
 * wheels. Modify ride() to call wheels() and
 * verify that polymorphism works.
 *********************************************/
package polymorphism;
import polymorphism.cycle2.*;

public class E05_Wheels {
  public static void ride(Cycle c) {
     System.out.println("Num. of wheels: " + c.wheels());
  }
  public static void main(String[] args) {
    ride(new Unicycle());
    ride(new Bicycle());
    ride(new Tricycle());
  }
} /* Output:
Num. of wheels: 1
Num. of wheels: 2
Num. of wheels: 3
*///:~
```

```
//: polymorphism/cycle2/Cycle.java
package polymorphism.cycle2;

public class Cycle {
  public int wheels() { return 0; }
} ///:~
```

```
//: polymorphism/cycle2/Unicycle.java
package polymorphism.cycle2;

public class Unicycle extends Cycle {
  public int wheels() { return 1; }
} ///:~
```

```
//: polymorphism/cycle2/Bicycle.java
package polymorphism.cycle2;

public class Bicycle extends Cycle {
  public int wheels() { return 2; }
} ///:~
```

```
//: polymorphism/cycle2/Tricycle.java
package polymorphism.cycle2;

public class Tricycle extends Cycle {
  public int wheels() { return 3; }
} ///:~
```

Cycle defines **wheels( )** to return `0`. In later chapters of *TIJ4*, you'll learn better ways to specify **wheels( )** than using a dummy implementation.

Exercise 6

```
//: polymorphism/E06_MusicToString.java
/****************** Exercise 6 *****************
 * Change Music3.java so what() becomes the root
 * Object method toString(). Print the Instrument
 * objects using System.out.println() (without
 * any casting).
 ***********************************************/
package polymorphism;
import polymorphism.music.Note;
import static net.mindview.util.Print.*;

class Instrument {
  void play(Note n) { print("Instrument.play() " + n); }
  public String toString() { return "Instrument"; }
```

```
  void adjust() {}
}

class Wind extends Instrument {
  void play(Note n) { print("Wind.play() " + n); }
  public String toString () { return "Wind"; }
}

class Percussion extends Instrument {
  void play(Note n) { print("Percussion.play() " + n); }
  public String toString () { return "Percussion"; }
}

class Stringed extends Instrument {
  void play(Note n) { print("Stringed.play() " + n); }
  public String toString () { return "Stringed"; }
}

class Brass extends Wind {
  void play(Note n) { print("Brass.play() " + n); }
  void adjust() { print("Brass.adjust()"); }
}

class Woodwind extends Wind {
  void play(Note n) { print("Woodwind.play() " + n); }
  public String toString () { return "Woodwind"; }
}

public class E06_MusicToString {
  static Instrument[] orchestra = {
    new Wind(),
    new Percussion(),
    new Stringed(),
    new Brass(),
    new Woodwind()
  };
  public static void printAll(Instrument[] orch) {
    for(Instrument i : orch)
      System.out.println(i);
  }
  public static void main(String args[]) {
    printAll(orchestra);
  }
} /* Output:
Wind
Percussion
Stringed
```

```
Wind
Woodwind
*///:~
```

Exercise 7

```
//: polymorphism/E07_NewInstrument.java
/****************** Exercise 7 *****************
 * Add a new type of Instrument to Music3.java
 * and verify that polymorphism works for your
 * new type.
 ***********************************************/
package polymorphism;
import polymorphism.music.Note;
import static net.mindview.util.Print.*;

class Electronic extends Instrument {
  void play(Note n) { print("Electronic.play() " + n); }
  public String toString() { return "Electronic"; }
}

public class E07_NewInstrument {
  static Instrument[] orchestra = {
    new Wind(),
    new Percussion(),
    new Stringed(),
    new Brass(),
    new Woodwind(),
    new Electronic()
  };
  public static void main(String args[]) {
    for(Instrument i : orchestra) {
      i.play(Note.MIDDLE_C);
      i.adjust();
      print(i);
    }
  }
} /* Output:
Wind.play() MIDDLE_C
Wind
Percussion.play() MIDDLE_C
Percussion
Stringed.play() MIDDLE_C
Stringed
Brass.play() MIDDLE_C
Brass.adjust()
```

```
Wind
Woodwind.play() MIDDLE_C
Woodwind
Electronic.play() MIDDLE_C
Electronic
*///:~
```

Exercise 8

```
//: polymorphism/E08_RandomInstruments.java
/****************** Exercise 8 *****************
 * Modify Music3.java so it randomly creates
 * Instrument objects the way Shapes.java does.
 ***********************************************/
package polymorphism;

class InstrumentGenerator {
  java.util.Random gen = new java.util.Random(47);
  public Instrument next() {
    switch(gen.nextInt(6)) {
      default:
      case 0: return new Wind();
      case 1: return new Percussion();
      case 2: return new Stringed();
      case 3: return new Brass();
      case 4: return new Woodwind();
      case 5: return new Electronic();
    }
  }
}

public class E08_RandomInstruments {
  public static void main(String args[]) {
    InstrumentGenerator gen = new InstrumentGenerator();
    for(int i = 0; i < 20; i++)
      System.out.println(gen.next());
  }
} /* Output:
Stringed
Electronic
Percussion
Electronic
Percussion
Electronic
Woodwind
Stringed
```

```
Wind
Percussion
Wind
Wind
Wind
Percussion
Electronic
Woodwind
Woodwind
Percussion
Stringed
Woodwind
*///:~
```

A *generator* produces a new value each time you call it, though you don't give it any parameters. The generator **nextInt( )** produces a random value within the array between zero (inclusive) and the value of the argument (exclusive).

Managing the **case** statement with a random generator can be error-prone. However, features of the **Class** object enable us to index an array that itself generates various instruments, as in this more elegant solution:

```
//: polymorphism/E08_RandomInstruments2.java
// A more sophisticated solution using features
// you'll learn about in later chapters.
package polymorphism;

class InstrumentGenerator2 {
  java.util.Random gen = new java.util.Random(47);
  Class<?> instruments[] = {
    Wind.class,
    Percussion.class,
    Stringed.class,
    Brass.class,
    Woodwind.class,
    Electronic.class,
  };
  public Instrument next() {
    try {
      int idx = Math.abs(gen.nextInt(instruments.length));
      return (Instrument) instruments[idx].newInstance();
    } catch(Exception e) {
      throw new RuntimeException("Cannot Create", e);
    }
  }
}
```

```
public class E08_RandomInstruments2 {
  public static void main(String args[]) {
    InstrumentGenerator2 gen = new InstrumentGenerator2();
    for(int i = 0; i < 20; i++)
      System.out.println(gen.next());
  }
} /* Output:
Stringed
Electronic
Percussion
Electronic
Percussion
Electronic
Woodwind
Stringed
Wind
Percussion
Wind
Wind
Wind
Percussion
Electronic
Woodwind
Woodwind
Percussion
Stringed
Woodwind
*///:~
```

The **.class** syntax in the array definition produces references to **Class** objects for each type of instrument. **Class.newInstance( )** creates an object of the class it is called for, but it can throw exceptions. Here, we create and throw a **RuntimeException** for this programming error, so your code doesn't have to catch such exceptions. You can embed the cause of an error inside a thrown exception to pass detailed information about the condition to a client programmer. (We cover exceptions in detail in the chapter *Error Handling with Exceptions*.) The benefit of this design is that you can add a new type to the system by only adding it to the **Class** array; the rest of the code takes care of itself.

Exercise 9

```
//: polymorphism/E09_Rodents.java
/****************** Exercise 9 *****************
 * Create an inheritance hierarchy of Rodent:
```

```
 * Mouse, Gerbil, Hamster, etc. In the base
 * class, provide methods that are common to all
 * Rodents, and override these in the derived
 * classes to perform different behaviors
 * depending on the specific type of Rodent.
 * Create an array of Rodent, fill it with
 * different specific types of Rodents, and call
 * your base-class methods to see what happens.
 *********************************************/
package polymorphism;
import static net.mindview.util.Print.*;

class Rodent {
  public void hop() { print("Rodent hopping"); }
  public void scurry() { print("Rodent scurrying"); }
  public void reproduce() { print("Making more Rodents"); }
  public String toString() { return "Rodent"; }
}

class Mouse extends Rodent {
  public void hop() { print("Mouse hopping"); }
  public void scurry() { print("Mouse scurrying"); }
  public void reproduce() { print("Making more Mice"); }
  public String toString() { return "Mouse"; }
}

class Gerbil extends Rodent {
  public void hop() { print("Gerbil hopping"); }
  public void scurry() { print("Gerbil scurrying"); }
  public void reproduce() { print("Making more Gerbils"); }
  public String toString() { return "Gerbil"; }
}

class Hamster extends Rodent {
  public void hop() { print("Hamster hopping"); }
  public void scurry() { print("Hamster scurrying"); }
  public void reproduce() { print("Making more Hamsters"); }
  public String toString() { return "Hamster"; }
}

public class E09_Rodents {
  public static void main(String args[]) {
    Rodent[] rodents = {
      new Mouse(),
      new Gerbil(),
      new Hamster(),
    };
```

```
    for(Rodent r : rodents) {
      r.hop();
      r.scurry();
      r.reproduce();
      print(r);
    }
  }
} /* Output:
Mouse hopping
Mouse scurrying
Making more Mice
Mouse
Gerbil hopping
Gerbil scurrying
Making more Gerbils
Gerbil
Hamster hopping
Hamster scurrying
Making more Hamsters
Hamster
*///:~
```

Exercise 10

```
//: polymorphism/E10_MethodCalls.java
/****************** Exercise 10 *****************
 * Create a base class with two methods. In the
 * first method, call the second method. Inherit
 * a class and override the second method. Create
 * an object of the derived class, upcast it to
 * the base type, and call the first method.
 * Explain what happens.
 ***********************************************/
package polymorphism;

class TwoMethods {
  public void m1() {
    System.out.println("Inside m1, calling m2");
    m2();
  }
  public void m2() {
    System.out.println("Inside m2");
  }
}

class Inherited extends TwoMethods {
```

```
  public void m2() {
    System.out.println("Inside Inherited.m2");
  }
}

public class E10_MethodCalls {
  public static void main(String args[]) {
    TwoMethods x = new Inherited();
    x.m1();
  }
} /* Output:
Inside m1, calling m2
Inside Inherited.m2
*///:~
```

The first method isn't overridden, but it calls the second method, which is. Java always uses the most-derived method for the object type; this is very powerful (and may surprise the unaware). The *Template Method* design pattern makes heavy use of polymorphism. (See *Thinking in Patterns with Java* at www.MindView.net.)

Exercise 11

```
//: polymorphism/E11_Pickle.java
/****************** Exercise 11 *****************
 * Add class Pickle to Sandwich.java.
 ************************************************/
package polymorphism;
import static net.mindview.util.Print.*;

class Pickle {
  Pickle() { print("Pickle()"); }
}

class Sandwich2 extends PortableLunch {
  Bread b = new Bread();
  Cheese c = new Cheese();
  Lettuce l = new Lettuce();
  Pickle p = new Pickle();
  Sandwich2() { print("Sandwich()"); }
}

public class E11_Pickle {
  public static void main(String args[]) {
    new Sandwich2();
  }
```

```
} /* Output:
Meal()
Lunch()
PortableLunch()
Bread()
Cheese()
Lettuce()
Pickle()
Sandwich()
*///:~
```

Notice that **Pickle** appears in the correct order, after the other member objects.

Exercise 12

```
//: polymorphism/E12_RodentInitialization.java
/****************** Exercise 12 *****************
 * Modify Exercise 9 so it demonstrates the
 * order of initialization of the base classes
 * and derived classes. Now add member objects to
 * both the base and derived classes, and show
 * the order in which their initialization occurs
 * during construction.
 ***********************************************/
package polymorphism;
import static net.mindview.util.Print.*;

class Member {
  public Member(String id) {
    print("Member constructor " + id);
  }
}

class Rodent2 {
  Member m1 = new Member("r1"), m2 = new Member("r2");
  public Rodent2() { print("Rodent constructor"); }
  public void hop() { print("Rodent hopping"); }
  public void scurry() { print("Rodent scurrying"); }
  public void reproduce() { print("Making more Rodents"); }
  public String toString() { return "Rodent"; }
}

class Mouse2 extends Rodent2 {
  Member m1 = new Member("m1"), m2 = new Member("m2");
  public Mouse2() { print("Mouse constructor"); }
  public void hop() { print("Mouse hopping"); }
```

```
  public void scurry() { print("Mouse scurrying"); }
  public void reproduce() { print("Making more Mice"); }
  public String toString() { return "Mouse"; }
}

class Gerbil2 extends Rodent2 {
  Member m1 = new Member("g1"), m2 = new Member("g2");
  public Gerbil2() { print("Gerbil constructor"); }
  public void hop() { print("Gerbil hopping"); }
  public void scurry() { print("Gerbil scurrying"); }
  public void reproduce() { print("Making more Gerbils"); }
  public String toString() { return "Gerbil"; }
}

class Hamster2 extends Rodent2 {
  Member m1 = new Member("h1"), m2 = new Member("h2");
  public Hamster2() { print("Hamster constructor"); }
  public void hop() { print("Hamster hopping"); }
  public void scurry() { print("Hamster scurrying"); }
  public void reproduce() { print("Making more Hamsters"); }
  public String toString() { return "Hamster"; }
}

public class E12_RodentInitialization {
  public static void main(String args[]) {
    new Hamster2();
  }
} /* Output:
Member constructor r1
Member constructor r2
Rodent constructor
Member constructor h1
Member constructor h2
Hamster constructor
*///:~
```

We initialize the base class first, starting with the member objects in order of definition, then the derived class, starting with its member objects.

Exercise 13

```
//: polymorphism/E13_VerifiedRefCounting.java
/****************** Exercise 13 *****************
 * Add a finalize() method to ReferenceCounting.java
 * to verify the termination condition. (See
 * the Initialization & Cleanup chapter.)
```

```
 *********************************************/
package polymorphism;
import static net.mindview.util.Print.*;

class Shared {
  private int refcount = 0;
  private static int counter = 0;
  private int id = counter++;
  public Shared() {
    print("Creating " + this);
  }
  public void addRef() { refcount++; }
  protected void dispose() {
    if(--refcount == 0)
      print("Disposing " + this);
  }
  protected void finalize() {
    if(refcount != 0)
      print("Error: object is not properly cleaned-up!");
  }
  public String toString() { return "Shared " + id; }
}

class Composing {
  private Shared shared;
  private static int counter = 0;
  private int id = counter++;
  public Composing(Shared shared) {
    print("Creating " + this);
    this.shared = shared;
    this.shared.addRef();
  }
  protected void dispose() {
    print("disposing " + this);
    shared.dispose();
  }
  public String toString() { return "Composing " + id; }
}

public class E13_VerifiedRefCounting {
  public static void main(String[] args) {
    Shared shared = new Shared();
    Composing[] composing = { new Composing(shared),
      new Composing(shared), new Composing(shared),
      new Composing(shared), new Composing(shared) };
    for(Composing c : composing)
      c.dispose();
```

```
    System.gc();
    // Verify failure:
    new Composing(new Shared());
    System.gc();
  }
} /* Output:
Creating Shared 0
Creating Composing 0
Creating Composing 1
Creating Composing 2
Creating Composing 3
Creating Composing 4
disposing Composing 0
disposing Composing 1
disposing Composing 2
disposing Composing 3
disposing Composing 4
Disposing Shared 0
Creating Shared 1
Creating Composing 5
Error: object is not properly cleaned-up!
*///:~
```

We kept our last **Composing( )** object alive, so you can see the termination condition report the mistake.

Exercise 14

```
//: polymorphism/E14_SharedRodentInitialization.java
/****************** Exercise 14 *****************
 * Modify Exercise 12 so one of the member
 * objects is a shared object with reference
 * counting, and demonstrate that it works
 * properly.
 ************************************************/
package polymorphism;
import static net.mindview.util.Print.*;

class NonSharedMember {
  public NonSharedMember(String id) {
    print("Non shared member constructor " + id);
  }
}

class SharedMember {
  private int refcount;
```

```
  public void addRef() {
    print("Number of references " + ++refcount);
  }
  protected void dispose() {
    if(--refcount == 0)
      print("Disposing " + this);
  }
  public SharedMember() {
    print("Shared member constructor");
  }
  public String toString() { return "Shared member"; }
}

class Rodent3 {
  private SharedMember m;
  NonSharedMember m1 = new NonSharedMember("r1"),
                  m2 = new NonSharedMember("r2");
  public Rodent3(SharedMember sm) {
    print("Rodent constructor");
    m = sm;
    m.addRef();
  }
  public void hop() { print("Rodent hopping"); }
  public void scurry() { print("Rodent scurrying"); }
  public void reproduce() { print("Making more Rodents"); }
  protected void dispose() {
    print("Disposing " + this);
    m.dispose();
  }
  public String toString() { return "Rodent"; }
}

class Mouse3 extends Rodent3 {
  NonSharedMember m1 = new NonSharedMember("m1"),
                  m2 = new NonSharedMember("m2");
  public Mouse3(SharedMember sm) {
    super(sm);
    print("Mouse constructor");
  }
  public void hop() { print("Mouse hopping"); }
  public void scurry() { print("Mouse scurrying"); }
  public void reproduce() { print("Making more Mice"); }
  public String toString() { return "Mouse"; }
}

class Gerbil3 extends Rodent3 {
  private SharedMember m;
```

```
  NonSharedMember m1 = new NonSharedMember("g1"),
                  m2 = new NonSharedMember("g2");
  public Gerbil3(SharedMember sm) {
    super(sm);
    print("Gerbil constructor");
  }
  public void hop() { print("Gerbil hopping"); }
  public void scurry() { print("Gerbil scurrying"); }
  public void reproduce() {
    print("Making more Gerbils");
  }
  public String toString() { return "Gerbil"; }
}

class Hamster3 extends Rodent3 {
  private SharedMember m;
  NonSharedMember m1 = new NonSharedMember("h1"),
                  m2 = new NonSharedMember("h2");
  public Hamster3(SharedMember sm) {
    super(sm);
    print("Hamster constructor");
  }
  public void hop() { print("Hamster hopping"); }
  public void scurry() { print("Hamster scurrying"); }
  public void reproduce() {
    print("Making more Hamsters");
  }
  public String toString() { return "Hamster"; }
}

public class E14_SharedRodentInitialization {
  public static void main(String args[]) {
    SharedMember sm = new SharedMember();
    Rodent3[] rodents = {
      new Hamster3(sm),
      new Gerbil3(sm),
      new Mouse3(sm),
    };
    for(Rodent3 r : rodents)
      r.dispose();
  }
} /* Output:
Shared member constructor
Non shared member constructor r1
Non shared member constructor r2
Rodent constructor
Number of references 1
```

```
Non shared member constructor h1
Non shared member constructor h2
Hamster constructor
Non shared member constructor r1
Non shared member constructor r2
Rodent constructor
Number of references 2
Non shared member constructor g1
Non shared member constructor g2
Gerbil constructor
Non shared member constructor r1
Non shared member constructor r2
Rodent constructor
Number of references 3
Non shared member constructor m1
Non shared member constructor m2
Mouse constructor
Disposing Hamster
Disposing Gerbil
Disposing Mouse
Disposing Shared member
*///:~
```

All types of rodents share one member object. When we instantiate a new concrete rodent, the reference counter of the shared member is incremented. When a rodent is destroyed, the reference counter is decremented. The shared member is automatically disposed after release of the last reference.

Exercise 15

```
//: polymorphism/E15_PolyConstructors2.java
/****************** Exercise 15 *****************
 * Add a RectangularGlyph to PolyConstructors.java
 * and demonstrate the problem described in this
 * section.
 ***********************************************/
package polymorphism;
import static net.mindview.util.Print.*;

class RectangularGlyph extends Glyph {
  private int width = 4;
  private int height = 5;
  RectangularGlyph(int width, int height) {
    this.width = width;
    this.height = height;
    print("RectangularGlyph.RectangularGlyph(), width = " +
```

```
      width + ", height = " + height);
  }
  void draw() {
    print("RectangularGlyph.draw(), area = " + width *
      height);
  }
}

public class E15_PolyConstructors2 {
  public static void main(String[] args) {
    new RoundGlyph(5);
    new RectangularGlyph(2,2);
  }
} /* Output:
Glyph() before draw()
RoundGlyph.draw(), radius = 0
Glyph() after draw()
RoundGlyph.RoundGlyph(), radius = 5
Glyph() before draw()
RectangularGlyph.draw(), area = 0
Glyph() after draw()
RectangularGlyph.RectangularGlyph(), width = 2, height = 2
*///:~
```

This section shows how polymorphism calls the most-derived method, even before completely initializing the object. During construction, you see output of **area = 0** because the base-class constructor calls **draw( )**, though this isn't obvious in the code.

Exercise 16

```
//: polymorphism/E16_Starship.java
/****************** Exercise 16 *****************
 * Following the example in Transmogrify.java,
 * create a Starship class containing an
 * AlertStatus reference that can indicate three
 * different states. Include methods to change
 * the states.
 ***********************************************/
package polymorphism;

class AlertStatus {
  public String getStatus() { return "None"; }
}

class RedAlertStatus extends AlertStatus {
```

```
  public String getStatus() { return "Red"; };
}

class YellowAlertStatus extends AlertStatus {
  public String getStatus() { return "Yellow"; };
}

class GreenAlertStatus extends AlertStatus {
  public String getStatus() { return "Green"; };
}

class Starship {
  private AlertStatus status = new GreenAlertStatus();
  public void setStatus(AlertStatus istatus) {
    status = istatus;
  }
  public String toString() { return status.getStatus(); }
}

public class E16_Starship {
  public static void main(String args[]) {
    Starship eprise = new Starship();
    System.out.println(eprise);
    eprise.setStatus(new YellowAlertStatus());
    System.out.println(eprise);
    eprise.setStatus(new RedAlertStatus());
    System.out.println(eprise);
  }
} /* Output:
Green
Yellow
Red
*///:~
```

This is an example of the *State* design pattern, wherein the object's behavior is state-dependent. The three states of an **AlertStatus** object, to which the **Starship** class holds a reference, represent behaviors of the **Starship**. You switch the **setStatus( )** reference from one state to another to change the behavior of the **Starship**.

Exercise 17

```
//: polymorphism/E17_RTTI.java
// {ThrowsException}
/****************** Exercise 17 ****************
```

```
 * Add a balance() method to Unicycle and Bicycle
 * but not to Tricycle, using the Cycle hierarchy
 * from Exercise 1. Upcast instances of all
 * three types to an array of Cycle. Try to call
 * balance() on each element of the  array and
 * observe the results. Downcast and call
 * balance() and observe what happens.
 *********************************************/
package polymorphism;
import polymorphism.cycle.Cycle;
import polymorphism.cycle.Tricycle;
import polymorphism.cycle3.*;

public class E17_RTTI {
  public static void main(String[] args) {
    Cycle[] cycles = new Cycle[]{ new Unicycle(),
      new Bicycle(), new Tricycle() };
    // Compile time: method not found in Cycle:
    // cycles[0].balance();
    // cycles[1].balance();
    // cycles[2].balance();
    ((Unicycle)cycles[0]).balance();  // Downcast/RTTI
    ((Bicycle)cycles[1]).balance();   // Downcast/RTTI
    ((Unicycle)cycles[2]).balance();  // Exception thrown
  }
} ///:~
```

```
//: polymorphism/cycle3/Unicycle.java
package polymorphism.cycle3;
import polymorphism.cycle.Cycle;

public class Unicycle extends Cycle {
  public void balance() {}
} ///:~
```

```
//: polymorphism/cycle3/Bicycle.java
package polymorphism.cycle3;
import polymorphism.cycle.Cycle;

public class Bicycle extends Cycle {
  public void balance() {}
} ///:~
```

Interfaces

Exercise 1

```
//: interfaces/E01_AbstractRodent.java
/****************** Exercise 1 *****************
 * Modify Exercise 9 in the previous chapter so
 * Rodent is an abstract class. Make the
 * methods of Rodent abstract whenever possible.
 ***********************************************/
package interfaces;
import static net.mindview.util.Print.*;

abstract class Rodent {
  public abstract void hop();
  public abstract void scurry();
  public abstract void reproduce();
}

class Mouse extends Rodent {
  public void hop() { print("Mouse hopping"); }
  public void scurry() { print("Mouse scurrying"); }
  public void reproduce() { print("Making more Mice"); }
  public String toString() { return "Mouse"; }
}

class Gerbil extends Rodent {
  public void hop() { print("Gerbil hopping"); }
  public void scurry() { print("Gerbil scurrying"); }
  public void reproduce() { print("Making more Gerbils"); }
  public String toString() { return "Gerbil"; }
}

class Hamster extends Rodent {
  public void hop() { print("Hamster hopping"); }
  public void scurry() { print("Hamster scurrying"); }
  public void reproduce() { print("Making more Hamsters"); }
  public String toString() { return "Hamster"; }
}

public class E01_AbstractRodent {
  public static void main(String args[]) {
```

```
    Rodent[] rodents = {
      new Mouse(),
      new Gerbil(),
      new Hamster(),
    };
    for(Rodent r : rodents) {
      r.hop();
      r.scurry();
      r.reproduce();
      print(r);
    }
  }
} /* Output:
Mouse hopping
Mouse scurrying
Making more Mice
Mouse
Gerbil hopping
Gerbil scurrying
Making more Gerbils
Gerbil
Hamster hopping
Hamster scurrying
Making more Hamsters
Hamster
*///:~
```

Note that the root class method **Object.toString( )** can be left out of the **abstract** base class.

Exercise 2

```
//: interfaces/E02_Abstract.java
// {CompileTimeError}
/****************** Exercise 2 *****************
 * Create a class as abstract without including
 * any abstract methods, and verify that you
 * cannot create any instances of that class.
 ***********************************************/
package interfaces;

abstract class NoAbstractMethods {
  void f() { System.out.println("f()"); }
}

public class E02_Abstract {
```

```
  public static void main(String args[]) {
    new NoAbstractMethods();
  }
} ///:~
```

Exercise 3

```
//: interfaces/E03_Initialization.java
/****************** Exercise 3 *****************
 * Create a base class with an abstract print()
 * method that is overridden in a derived class.
 * The overridden version of the method prints
 * the value of an int variable defined in the
 * derived class. Define this variable with a
 * nonzero value. Call print() in the
 * base-class constructor. Create an object of
 * the derived type in main(), then call its
 * print() method. Explain the results.
 ***********************************************/
package interfaces;

abstract class BaseWithPrint {
  public BaseWithPrint() { print(); }
  public abstract void print();
}

class DerivedWithPrint extends BaseWithPrint {
  int i = 47;
  public void print() {
    System.out.println("i = " + i);
  }
}

public class E03_Initialization {
  public static void main(String args[]) {
    DerivedWithPrint dp = new DerivedWithPrint();
    dp.print();
  }
} /* Output:
i = 0
i = 47
*///:~
```

The java virtual machine zeroes the bits of the object after it allocates storage, producing a default value for **i** before any other initialization occurs. The code

calls the base-class constructor before running the derived-class initialization, so we see the zeroed value of **i** as the initial output.

The danger of calling a method inside a constructor is when that method depends on a derived initialization. Before the derived-class constructor is called, the object may be in an unexpected state (in Java, at least that state is *defined*; this is not true with all languages – C++, for example). The safest approach is to set the object into a known good state as simply as possible, and then perform any other operations outside the constructor.

Exercise 4

```
//: interfaces/E04_AbstractBase.java
/****************** Exercise 4 *****************
 * Create an abstract class with no methods.
 * Derive a class and add a method. Create a
 * static method that downcasts a reference from
 * the base class to the derived class and calls
 * the method. Demonstrate that it works in main().
 * Eliminate the need for the downcast by moving
 * the abstract declaration to the base class.
 ***********************************************/
package interfaces;

abstract class NoMethods {}

class Extended1 extends NoMethods {
  public void f() {
    System.out.println("Extended1.f");
  }
}

abstract class WithMethods {
  abstract public void f();
}

class Extended2 extends WithMethods {
  public void f() {
    System.out.println("Extended2.f");
  }
}

public class E04_AbstractBase {
  public static void test1(NoMethods nm) {
    // Must downcast to access f():
```

```
    ((Extended1)nm).f();
  }
  public static void test2(WithMethods wm) {
    // No downcast necessary:
    wm.f();
  }
  public static void main(String args[]) {
    NoMethods nm = new Extended1();
    test1(nm);
    WithMethods wm = new Extended2();
    test2(wm);
  }
} /* Output:
Extended1.f
Extended2.f
*///:~
```

test1( ) needs the downcast to call **f( )**, while **test2( )** doesn't need a downcast because **f( )** is defined in the base class.

Exercise 5

```
//: interfaces/E05_ImplementInterface.java
/****************** Exercise 5 ******************
 * Create an interface with three methods in its
 * own package. Implement the interface in a
 * different package.
 ***********************************************/
package interfaces;
import interfaces.ownpackage.*;
import static net.mindview.util.Print.*;

class ImplementInterface implements AnInterface {
  public void f() { print("ImplementInterface.f"); }
  public void g() { print("ImplementInterface.g"); }
  public void h() { print("ImplementInterface.h"); }
}

public class E05_ImplementInterface {
  public static void main(String args[]) {
    ImplementInterface imp = new ImplementInterface();
    imp.f();
    imp.g();
    imp.h();
  }
} /* Output:
```

```
ImplementInterface.f
ImplementInterface.g
ImplementInterface.h
*///:~
```

The elements of an interface are **public** but the interface itself is not, until we declare it **public** for use outside its package:

```
//: interfaces/ownpackage/AnInterface.java
package interfaces.ownpackage;

public interface AnInterface {
  void f();
  void g();
  void h();
} ///:~
```

Exercise 6

```
//: interfaces/E06_InterfacePublicMethods.java
/****************** Exercise 6 ******************
 * Prove that all the methods in an interface are
 * automatically public.
 ***********************************************/
package interfaces;
import interfaces.ownpackage.*;

public class E06_InterfacePublicMethods
  implements AnInterface {
  // Each of these produces a compile-time error message,
  // stating that you cannot reduce the access of the
  // base class public method in a derived class.
//!  void f() {}
//!  void g() {}
//!  void h() {}
  // Compiles OK:
  public void f() {}
  public void g() {}
  public void h() {}
  public static void main(String args[]) {}
} ///:~
```

Exercise 7

```
//: interfaces/E07_RodentInterface.java
```

```
/****************** Exercise 7 ******************
 * Change Rodent to an interface in Exercise 9 of
 * the Polymorphism chapter.
 ***********************************************/
package interfaces;
import static net.mindview.util.Print.*;

interface Rodent2 {
  void hop();
  void scurry();
  void reproduce();
}

class Mouse2 implements Rodent2 {
  public void hop() { print("Mouse hopping"); }
  public void scurry() { print("Mouse scurrying"); }
  public void reproduce() { print("Making more Mice"); }
  public String toString() { return "Mouse"; }
}

class Gerbil2 implements Rodent2 {
  public void hop() { print("Gerbil hopping"); }
  public void scurry() { print("Gerbil scurrying"); }
  public void reproduce() { print("Making more Gerbils"); }
  public String toString() { return "Gerbil"; }
}

class Hamster2 implements Rodent2 {
  public void hop() { print("Hamster hopping"); }
  public void scurry() { print("Hamster scurrying"); }
  public void reproduce() { print("Making more Hamsters"); }
  public String toString() { return "Hamster"; }
}

public class E07_RodentInterface {
  public static void main(String args[]) {
    Rodent2[] rodents = {
      new Mouse2(),
      new Gerbil2(),
      new Hamster2(),
    };
    for(Rodent2 r : rodents) {
      r.hop();
      r.scurry();
      r.reproduce();
      print(r);
    }
```

```
  }
} /* Output:
Mouse hopping
Mouse scurrying
Making more Mice
Mouse
Gerbil hopping
Gerbil scurrying
Making more Gerbils
Gerbil
Hamster hopping
Hamster scurrying
Making more Hamsters
Hamster
*///:~
```

Exercise 8

```
//: interfaces/E08_FastFood.java
/****************** Exercise 8 ******************
 * Create an interface called FastFood (with
 * appropriate methods) in
 * polymorphism.Sandwich.java, and change Sandwich
 * so it also implements FastFood.
 ***********************************************/
package interfaces;
import polymorphism.Sandwich;
import static net.mindview.util.Print.*;

interface FastFood {
  void rushOrder();
  void gobble();
}

class FastSandwich extends Sandwich implements FastFood {
  public void rushOrder() {
    print("Rushing your sandwich order");
  }
  public void gobble() { print("Chomp! Snort! Gobble!"); }
}

public class E08_FastFood {
  public static void main(String args[]) {
    FastSandwich burger = new FastSandwich();
    print("Fries with that?");
    print("Super Size?");
```

```
    burger.rushOrder();
    burger.gobble();
  }
} /* Output:
Meal()
Lunch()
PortableLunch()
Bread()
Cheese()
Lettuce()
Sandwich()
Fries with that?
Super Size?
Rushing your sandwich order
Chomp! Snort! Gobble!
*///:~
```

Exercise 9

```
//: interfaces/E09_AbstractMusic5.java
/****************** Exercise 9 ******************
 * Refactor Music5.java by moving the common
 * methods in Wind, Percussion and Stringed into
 * an abstract class.
 ***********************************************/
package interfaces;
import polymorphism.music.Note;
import static net.mindview.util.Print.*;

abstract class Instrument {
  public void play(Note n) {
    print(this + ".play() " + n);
  }
  public void adjust() { print(this + ".adjust()"); }
  // Forces implementation in derived class:
  public abstract String toString();
}

class Wind extends Instrument {
  public String toString() { return "Wind"; }
}

class Percussion extends Instrument {
  public String toString() { return "Percussion"; }
}
```

```
class Stringed extends Instrument {
  public String toString() { return "Stringed"; }
}

class Brass extends Wind {
  public String toString() { return "Brass"; }
}

class Woodwind extends Wind {
  public String toString() { return "Woodwind"; }
}

public class E09_AbstractMusic5 {
  static void tune(Instrument i) {
    i.adjust();
    i.play(Note.MIDDLE_C);
  }
  static void tuneAll(Instrument[] e) {
    for(Instrument i : e)
      tune(i);
  }
  public static void main(String[] args) {
    Instrument[] orchestra = {
      new Wind(),
      new Percussion(),
      new Stringed(),
      new Brass(),
      new Woodwind()
    };
    tuneAll(orchestra);
  }
} /* Output:
Wind.adjust()
Wind.play() MIDDLE_C
Percussion.adjust()
Percussion.play() MIDDLE_C
Stringed.adjust()
Stringed.play() MIDDLE_C
Brass.adjust()
Brass.play() MIDDLE_C
Woodwind.adjust()
Woodwind.play() MIDDLE_C
*///:~
```

We eliminate code duplication, moving common functionality into the **abstract** base class. **toString( )** is now an **abstract** method, so all classes that implement **Instrument** provide a definition for it. Without the redefinition of

toString( ), all **Instrument**s would otherwise use the original, non-**abstract** **toString( )** from the root class **Object**.

Exercise 10

```
//: interfaces/E10_PlayableMusic5.java
/****************** Exercise 10 ******************
 * Add a Playable to Modify Music5.java, and move
 * the play() declaration from Instrument to
 * Playable. Include Playable in the implements
 * list to add it to the derived classes.
 * Change tune() so it takes a Playable instead
 * of an Instrument.
 *********************************************/
package interfaces;
import polymorphism.music.Note;
import static net.mindview.util.Print.*;

interface Instrument2 {
  void adjust();
}

interface Playable {
  void play(Note n);
}

class Wind2 implements Instrument2, Playable {
  public void play(Note n) {
    print(this + ".play() " + n);
  }
  public String toString() { return "Wind"; }
  public void adjust() { print(this + ".adjust()"); }
}

class Percussion2 implements Instrument2, Playable {
  public void play(Note n) {
    print(this + ".play() " + n);
  }
  public String toString() { return "Percussion"; }
  public void adjust() { print(this + ".adjust()"); }
}

class Stringed2 implements Instrument2, Playable {
  public void play(Note n) {
    print(this + ".play() " + n);
```

```
  }
  public String toString() { return "Stringed"; }
  public void adjust() { print(this + ".adjust()"); }
}

class Brass2 extends Wind2 {
  public String toString() { return "Brass"; }
}

class Woodwind2 extends Wind2 {
  public String toString() { return "Woodwind"; }
}

public class E10_PlayableMusic5 {
  static void tune(Playable p) { p.play(Note.MIDDLE_C); }
  static void tuneAll(Playable[] e) {
    for(Playable p : e)
      tune(p);
  }
  public static void main(String[] args) {
    Playable[] orchestra = {
      new Wind2(),
      new Percussion2(),
      new Stringed2(),
      new Brass2(),
      new Woodwind2()
    };
    tuneAll(orchestra);
  }
} /* Output:
Wind.play() MIDDLE_C
Percussion.play() MIDDLE_C
Stringed.play() MIDDLE_C
Brass.play() MIDDLE_C
Woodwind.play() MIDDLE_C
*///:~
```

We make **Playable** a concrete class to eliminate code duplication, as **Wind2**, **Percussion2**, and **Stringed2** only use interfaces, and don't inherit from any concrete classes.

Exercise 11

```
//: interfaces/E11_Swapper.java
/****************** Exercise 11 ******************
 * Create a class with a method that takes a String
```

```
 * argument and produces a result that swaps each
 * pair of characters in that argument. Adapt the
 * class to work with
 * interfaceprocessor.Apply.process().
 *********************************************/
package interfaces;
import interfaces.interfaceprocessor.*;

class CharacterPairSwapper {
  static String swap(String s) {
    StringBuilder sb = new StringBuilder(s);
    for(int i = 0; i < sb.length() - 1; i += 2) {
      char c1 = sb.charAt(i);
      char c2 = sb.charAt(i + 1);
      sb.setCharAt(i, c2);
      sb.setCharAt(i + 1, c1);
    }
    return sb.toString();
  }
}

class SwapperAdapter implements Processor {
  public String name() {
    return CharacterPairSwapper.class.getSimpleName();
  }
  public String process(Object input) {
    return CharacterPairSwapper.swap((String)input);
  }
}

public class E11_Swapper {
  public static void main(String[] args) {
    Apply.process(new SwapperAdapter(), "1234");
    Apply.process(new SwapperAdapter(), "abcde");
  }
} /* Output:
Using Processor CharacterPairSwapper
2143
Using Processor CharacterPairSwapper
badce
*///:~
```

CharacterPairSwapper uses the methods of the **StringBuilder** class to access and modify individual characters inside the character sequence. (See the J2SE5 API documentation for details.) **CharacterPairSwapper** has a completely different interface and cannot be directly integrated into the rest of

the input processing system. The *Adapter* pattern application solves this problem.

Because **swap( )** is a **static** method, you don't need a new **SwapperAdapter** instance for each **SwapperAdapter**. Refactor the **SwapperAdapter** class as a *Singleton* as an additional exercise.

Exercise 12

```
//: interfaces/E12_CanClimb.java
/****************** Exercise 12 ******************
 * Follow the form of the other
 * interfaces to add an interface called
 * CanClimb in Adventure.java.
 *********************************************/
package interfaces;

interface CanClimb {
  void climb();
}

class Hero2 extends ActionCharacter
  implements CanFight, CanSwim, CanFly, CanClimb {
  public void swim() {}
  public void fly() {}
  public void climb() {}
}

public class E12_CanClimb {
  static void t(CanFight x) { x.fight(); }
  static void u(CanSwim x) { x.swim(); }
  static void v(CanFly x) { x.fly(); }
  static void z(CanClimb x) { x.climb(); }
  static void w(ActionCharacter x) { x.fight(); }
  public static void main(String[] args) {
    Hero2 h = new Hero2();
    t(h); // Treat it as a CanFight
    u(h); // Treat it as a CanSwim
    v(h); // Treat it as a CanFly
    z(h); // Treat it as a CanClimb
    w(h); // Treat it as an ActionCharacter
  }
} ///:~
```

Exercise 13

```
//: interfaces/E13_Diamond.java
/****************** Exercise 13 ******************
 * Create an interface, inherit two new
 * interfaces from it, then multiply-inherit
 * a third interface from the second two.
 *********************************************/
package interfaces;

interface BaseInterface {
  void f();
}

interface IntermediateInterface1 extends BaseInterface {
  void f();
}

interface IntermediateInterface2 extends BaseInterface {
  void f();
}

interface CombinedInterface
extends IntermediateInterface1, IntermediateInterface2 {
  void f();
}

class CombinedImpl implements CombinedInterface {
  public void f() {
    System.out.println("CombinedImpl.f()");
  }
}

public class E13_Diamond {
  public static void main(String[] args) {
    new CombinedImpl().f();
  }
} /* Output:
CombinedImpl.f()
*///:~
```

Java allows multiple interface inheritance but not multiple implementation inheritance, which eliminates ambiguity about which of two identical members we use when combining implementations of the same base class. We replicate **f( )** in the interfaces above to demonstrate that Java avoids the "diamond problem" (so called because of the diamond-shaped class diagram produced by

multiple inheritance. C++ requires extra base-class syntax resolve the ambiguity created by concrete multiple inheritance).

Exercise 14

```
//: interfaces/E14_InterfaceInheritance.java
/****************** Exercise 14 ******************
 * Create three interfaces, each with two methods.
 * Inherit a new interface from each, adding
 * a new method. Use the new interface to create
 * a class, and inherit from a concrete class.
 * Now write four methods, each of which takes one
 * of the four interfaces as an argument. Create
 * an object of your class in main(), and pass it
 * to each of the methods.
 *********************************************/
package interfaces;
import static net.mindview.util.Print.*;

interface Interface1 {
  void f1();
  void g1();
}

interface Interface2 {
  void f2();
  void g2();
}

interface Interface3 {
  void f3();
  void g3();
}

interface Multiple
  extends Interface1, Interface2, Interface3 {
  void h();
}

class Concrete {
  String s;
  public Concrete(String s) { this.s = s; }
}

class All extends Concrete implements Multiple {
```

```
  All() { super("All"); }
  public void h() { print("All.h"); }
  public void f1() { print("All.f1"); }
  public void g1() { print("All.g1"); }
  public void f2() { print("All.f2"); }
  public void g2() { print("All.g2"); }
  public void f3() { print("All.f3"); }
  public void g3() { print("All.g3"); }
}

public class E14_InterfaceInheritance {
  static void takes1(Interface1 i) {
    i.f1();
    i.g1();
  }
  static void takes2(Interface2 i) {
    i.f2();
    i.g2();
  }
  static void takes3(Interface3 i) {
    i.f3();
    i.g3();
  }
  static void takesAll(All a) {
    a.f1();
    a.g1();
    a.f2();
    a.g2();
    a.f3();
    a.g3();
    a.h();
  }
  public static void main(String args[]) {
    All a = new All();
    takes1(a);
    takes2(a);
    takes3(a);
    takesAll(a);
  }
} /* Output:
All.f1
All.g1
All.f2
All.g2
All.f3
All.g3
All.f1
```

```
All.g1
All.f2
All.g2
All.f3
All.g3
All.h
*///:~
```

Exercise 15

```
//: interfaces/E15_AbstractsAndInterfaces.java
/****************** Exercise 15 *****************
 * Modify Exercise 14 by creating an abstract class
 * and inheriting it into the derived class.
 ***********************************************/
package interfaces;
import static net.mindview.util.Print.*;

abstract class Abstract {
  String s;
  public Abstract(String s) { this.s = s; }
  abstract void af();
}

class All2 extends Abstract implements Multiple {
  All2() { super("All2"); }
  void af() { print("All.af"); }
  public void f1() { print("All.f1"); }
  public void g1() { print("All.g1"); }
  public void f2() { print("All.f2"); }
  public void g2() { print("All.g2"); }
  public void f3() { print("All.f3"); }
  public void g3() { print("All.g3"); }
  public void h() { print("All2.h"); }
}

public class E15_AbstractsAndInterfaces {
  static void takes1(Interface1 i) {
    i.f1();
    i.g1();
  }
  static void takes2(Interface2 i) {
    i.f2();
    i.g2();
  }
  static void takes3(Interface3 i) {
```

```
    i.f3();
    i.g3();
  }
  static void takesAll(All2 a) {
    a.f1();
    a.g1();
    a.f2();
    a.g2();
    a.f3();
    a.g3();
    a.h();
  }
  static void takesAbstract(Abstract a) {
    a.af();
  }
  public static void main(String args[]) {
    All2 a = new All2();
    takes1(a);
    takes2(a);
    takes3(a);
    takesAll(a);
    takesAbstract(a);
  }
} /* Output:
All.f1
All.g1
All.f2
All.g2
All.f3
All.g3
All.f1
All.g1
All.f2
All.g2
All.f3
All.g3
All2.h
All.af
*///:~
```

Exercise 16

```
//: interfaces/E16_AdaptedCharSequence.java
/****************** Exercise 16 ******************
 * Create a class that produces a sequence of chars.
 * Adapt this class so that it can be an input to a
```

```
 * Scanner object.
 *********************************************/
package interfaces;
import java.nio.*;
import java.util.*;

class CharSequence {
  private static Random rand = new Random(47);
  private static final char[] capitals =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();
  private static final char[] lowers =
    "abcdefghijklmnopqrstuvwxyz".toCharArray();
  private static final char[] vowels =
    "aeiou".toCharArray();
  char[] generate() {
    char[] buffer = new char[10];
    int idx = 0;
    buffer[idx++] = capitals[rand.nextInt(capitals.length)];
    for(int i = 0; i < 4; i++) {
      buffer[idx++] = vowels[rand.nextInt(vowels.length)];
      buffer[idx++] = lowers[rand.nextInt(lowers.length)];
    }
    buffer[idx] = ' ';
    return buffer;
  }
}

class E16_AdaptedCharSequence extends CharSequence
implements Readable {
  private int count;
  public E16_AdaptedCharSequence(int count) {
    this.count = count;
  }
  public int read(CharBuffer cb) {
    if(count-- == 0)
      return -1; // Indicates end of input
    char[] buffer = generate();
    cb.put(buffer);
    return buffer.length;
  }
  public static void main(String[] args) {
    Scanner s =
      new Scanner(new E16_AdaptedCharSequence(10));
    while(s.hasNext())
      System.out.println(s.next());
  }
} /* Output:
```

```
Yazeruyac
Fowenucor
Goeazimom
Raeuuacio
Nuoadesiw
Hageaikux
Ruqicibui
Numasetih
Kuuuuozog
Waqizeyoy
*///:~
```

Though structurally different, this program behaves like **RandomWords.java** from the *Interfaces* chapter. Because **CharSequence** alone does not work as an input to a **Scanner** instance, an *Adapter* is created to provide a **read( )** method.

Exercise 17

```
//: interfaces/E17_ImplicitStaticFinal.java
/****************** Exercise 17 ******************
 * Prove that the fields in an interface are
 * implicitly static and final.
 *********************************************/
package interfaces;

interface StaticFinalTest {
  String RED = "Red";
}

class Test implements StaticFinalTest {
  public Test() {
    // Compile-time error: cannot assign a value
    // to final variable RED:
    //!  RED = "Blue";
  }
}

public class E17_ImplicitStaticFinal {
  public static void main(String args[]) {
    // Accessing as a static field:
    System.out.println("StaticFinalTest.RED = "
                       + StaticFinalTest.RED);
  }
} /* Output:
StaticFinalTest.RED = Red
```

```
*///:~
```

The compiler tells you **RED** is a final variable when you try to assign a value to it. **RED** is clearly **static** because you can access it using **static** syntax.

Exercise 18

```
//: interfaces/E18_CycleFactories.java
/****************** Exercise 18 ******************
 * Create a Cycle interface, with implementations
 * Unicycle, Bicycle, and Tricycle. Create factories
 * for each type of Cycle, and code that uses
 * these factories.
 *********************************************/
package interfaces;

interface Cycle {
  int wheels();
}

interface CycleFactory {
  Cycle getCycle();
}

class Unicycle implements Cycle {
  public int wheels() { return 1; }
}

class UnicycleFactory implements CycleFactory {
  public Unicycle getCycle() { return new Unicycle(); }
}

class Bicycle implements Cycle {
  public int wheels() { return 2; }
}

class BicycleFactory implements CycleFactory {
  public Bicycle getCycle() { return new Bicycle(); }
}

class Tricycle implements Cycle {
  public int wheels() { return 3; }
}

class TricycleFactory implements CycleFactory {
  public Tricycle getCycle() { return new Tricycle(); }
```

```
}

public class E18_CycleFactories {
  public static void ride(CycleFactory fact) {
     Cycle c = fact.getCycle();
     System.out.println("Num. of wheels: " + c.wheels());
  }
  public static void main(String[] args) {
    ride(new UnicycleFactory());
    ride(new BicycleFactory ());
    ride(new TricycleFactory ());
  }
} /* Output:
Num. of wheels: 1
Num. of wheels: 2
Num. of wheels: 3
*///:~
```

This solution has several more classes than Exercise 5 in the *Polymorphism* chapter. To anticipate every possibility, programmers often use this interface + factory form to create classes, because it allows them to add new classes anywhere. However, complexity and necessary maintenance make this a wise choice only when you know you'll be adding new classes.

Exercise 19

```
//: interfaces/E19_TossingFramework.java
/****************** Exercise 19 ******************
 * Create a framework using Factory Methods that
 * performs both coin tossing and dice tossing.
 *********************************************/
package interfaces;

interface Tossing { boolean event(); }

interface TossingFactory { Tossing getTossing(); }

class CoinTossing implements Tossing {
  private int events;
  private static final int EVENTS = 2;
  public boolean event() {
    System.out.println("Coin tossing event " + events);
    return ++events != EVENTS;
  }
}
```

```
class CoinTossingFactory implements TossingFactory {
  public CoinTossing getTossing() {
    return new CoinTossing();
  }
}

class DiceTossing implements Tossing {
  private int events;
  private static final int EVENTS = 6;
  public boolean event() {
    System.out.println("Dice tossing event " + events);
    return ++events != EVENTS;
  }
}

class DiceTossingFactory implements TossingFactory {
  public DiceTossing getTossing() {
    return new DiceTossing();
  }
}

public class E19_TossingFramework {
  public static void simulate(TossingFactory fact) {
    Tossing t = fact.getTossing();
    while(t.event())
      ;
  }
  public static void main(String[] args) {
    simulate(new CoinTossingFactory());
    simulate(new DiceTossingFactory());
  }
} /* Output:
Coin tossing event 0
Coin tossing event 1
Dice tossing event 0
Dice tossing event 1
Dice tossing event 2
Dice tossing event 3
Dice tossing event 4
Dice tossing event 5
*///:~
```

Inner Classes

Exercise 1

```
//: innerclasses/E01_ReferenceToInnerClass.java
/****************** Exercise 1 *****************
 * Write a class named Outer containing an
 * inner class named Inner. Add a method to Outer
 * that returns an object of type Inner. In
 * main(), create and initialize a reference to
 * an Inner.
 ***********************************************/
package innerclasses;

class Outer {
  class Inner {
    { System.out.println("Inner created"); }
  }
  Inner getInner() { return new Inner(); }
}

public class E01_ReferenceToInnerClass {
  public static void main(String[] args) {
    Outer o = new Outer();
    Outer.Inner i = o.getInner();
  }
} /* Output:
Inner created
*///:~
```

Exercise 2

```
//: innerclasses/E02_SequenceOfStringHolders.java
/****************** Exercise 2 *****************
 * Create a class that holds a String, with a
 * toString() method that displays this String.
 * Add several instances of your new class to a
 * Sequence object, then display them.
 ***********************************************/
package innerclasses;
```

```
class StringHolder {
  private String data;
  StringHolder(String data) { this.data = data; }
  public String toString() { return data; }
}

public class E02_SequenceOfStringHolders {
  public static void main(String[] args) {
    Sequence sequence = new Sequence(10);
    for(int i = 0; i < 10; i++)
      sequence.add(new StringHolder(Integer.toString(i)));
    Selector selector = sequence.selector();
    while(!selector.end()) {
      System.out.print(selector.current() + " ");
      selector.next();
    }
  }
} /* Output:
0 1 2 3 4 5 6 7 8 9
*///:~
```

Exercise 3

```
//: innerclasses/E03_InnerAccessingOuter.java
/****************** Exercise 3 *****************
 * Modify Exercise 1 so Outer has a private
 * String field (initialized by the constructor),
 * and Inner has a toString() that displays this
 * field. Create an object of type Inner and
 * display it.
 ***********************************************/
package innerclasses;

class Outer2 {
  private final String data;
  class Inner {
    public String toString() { return data; }
  }
  Outer2(String data) { this.data = data; }
  Inner getInner() { return new Inner(); }
}

public class E03_InnerAccessingOuter {
  public static void main(String[] args) {
    Outer2 o = new Outer2("Inner accessing outer!");
    Outer2.Inner i = o.getInner();
```

```
    System.out.println(i.toString());
  }
} /* Output:
Inner accessing outer!
*///:~
```

Exercise 4

```
//: innerclasses/E04_SequenceSelectorToSequence.java
/****************** Exercise 4 *****************
 * Add a method to the class Sequence.SequenceSelector
 * that produces the reference to the outer class
 * Sequence.
 ***********************************************/
package innerclasses;

class Sequence2 {
  private Object[] items;
  private int next;
  public Sequence2(int size) { items = new Object[size]; }
  public void add(Object x) {
    if(next < items.length)
      items[next++] = x;
  }
  private class SequenceSelector implements Selector {
    private int i;
    public boolean end() { return i == items.length; }
    public Object current() { return items[i]; }
    public void next() { if(i < items.length) i++; }
    public Sequence2 sequence() { return Sequence2.this; }
  }
  public Selector selector() {
    return new SequenceSelector();
  }
  public boolean check() {
    return
      this == ((SequenceSelector)selector()).sequence();
  }
}

public class E04_SequenceSelectorToSequence {
  public static void main(String[] args) {
    Sequence2 s = new Sequence2(10);
    System.out.println(s.check());
  }
} /* Output:
```

```
true
*///:~
```

The private inner class **SequenceSelector** is inaccessible outside of **Sequence2**, so **check( )** performs the validation.

For more information about this topic, see the *Inner classes and upcasting* section of *TIJ4*.

Exercise 5

```
//: innerclasses/E05_InstanceOfInner.java
/****************** Exercise 5 *****************
 * Create a class with an inner class. In a
 * separate class, make an instance of the inner
 * class.
 ***********************************************/
package innerclasses;

class Outer3 {
  class Inner {
    { System.out.println("Inner created"); }
  }
}

public class E05_InstanceOfInner {
  public static void main(String args[]) {
    Outer3 o = new Outer3();
    Outer3.Inner i = o.new Inner();
  }
} /* Output:
Inner created
*///:~
```

To create a reference in the separate class **E05_InstanceOfInner**, you must fully resolve the name of the inner class **Outer3.Inner**. The inner class object has a connection to the outer-class object, so the **new** expression must specify the object that creates the inner class.

Exercise 6

```
//: innerclasses/E06_ProtectedInnerClass.java
/****************** Exercise 6 *****************
 * Create an interface with at least one method,
 * in its own package. Create a class in a
```

```
 * separate package. Add a protected inner class
 * that implements the interface. In a third
 * package, inherit from your class and, inside a
 * method, return an object of the protected
 * inner class, upcasting to the interface during
 * the return.
 ***********************************************/
package innerclasses;
import innerclasses.exercise6b.*;
import innerclasses.exercise6.*;

public class E06_ProtectedInnerClass
extends SimpleClass {
  public SimpleInterface get() {
    return new Inner();
  }
  public static void main(String args[]) {
    new E06_ProtectedInnerClass().get().f();
  }
} ///:~
```

```
//: innerclasses/exercise6/SimpleInterface.java
package innerclasses.exercise6;

public interface SimpleInterface {
  void f();
} ///:~
```

```
//: innerclasses/exercise6b/SimpleClass.java
package innerclasses.exercise6b;
import innerclasses.exercise6.*;

public class SimpleClass {
  protected class Inner implements SimpleInterface {
    // Force constructor to be public:
    public Inner() {}
    public void f() {}
  }
} ///:~
```

You cannot access the synthesized default constructor from **E06 ProtectedInnerClass** because it has the same **protected** access as the defining class.

Exercise 7

```
//: innerclasses/E07_InnerClassAccess.java
```

```
/****************** Exercise 7 *****************
 * Create a class with a private field and a
 * private method. Create an inner class with a
 * method that modifies the outer-class field and
 * calls the outer-class method. In a second
 * outer-class method, create an object of the
 * inner class and call its method, then show
 * the effect on the outer-class object.
 ***********************************************/
package innerclasses;

public class E07_InnerClassAccess {
  private int i = 10;
  private void f() {
    System.out.println("E07_InnerClassAccess.f");
  }
  class Inner {
    void g() {
      i++;
      f();
    }
  }
  public void h() {
    Inner in = new Inner();
    in.g();
    System.out.println("i = " + i);
  }
  public static void main(String args[]) {
    E07_InnerClassAccess ica = new E07_InnerClassAccess();
    ica.h();
  }
} /* Output:
E07_InnerClassAccess.f
i = 11
*///:~
```

This exercise shows that inner classes have transparent access to their outer-class objects, even **private** fields and methods.

Exercise 8

```
//: innerclasses/E08_OuterAccessingInner.java
/****************** Exercise 8 *****************
 * Determine whether an outer class has access to
 * the private elements of its inner class.
 ***********************************************/
```

```
package innerclasses;

class Outer4 {
  class Inner {
    private int j;
    private void h() {
      System.out.println("Inner.h called");
      System.out.println("Inner.j = " + j);
    }
  }
  public void testInnerAccess() {
    Inner i = new Inner();
    i.j = 47;
    i.h();
  }
}

public class E08_OuterAccessingInner {
  public static void main(String args[]) {
    Outer4 o = new Outer4();
    o.testInnerAccess();
  }
} /* Output:
Inner.h called
Inner.j = 47
*///:~
```

As you can see from the output, the accessibility goes both ways.

Exercise 9

```
//: innerclasses/E09_InnerClassInMethod.java
/****************** Exercise 9 *****************
 * Create an interface with at least one method,
 * and implement it by defining an
 * inner class within a method that returns a
 * reference to your interface.
 **********************************************/
package innerclasses;
import innerclasses.exercise6.*;

public class E09_InnerClassInMethod {
  public SimpleInterface get() {
    class SI implements SimpleInterface{
      public void f() { System.out.println("SI.f"); }
    }
```

```
    return new SI();
  }
  public static void main(String args[]) {
    SimpleInterface si =
      new E09_InnerClassInMethod().get();
    si.f();
  }
} /* Output:
SI.f
*///:~
```

Exercise 10

```
//: innerclasses/E10_InnerClassInScope.java
/****************** Exercise 10 *****************
 * Repeat Exercise 9 but define the inner
 * class within a scope within a method.
 ***********************************************/
package innerclasses;
import innerclasses.exercise6.*;

public class E10_InnerClassInScope {
  public SimpleInterface get() {
    {
      class SI implements SimpleInterface{
        public void f() {
          System.out.println("SI.f");
        }
      }
      return new SI();
    }
  }
  public static void main(String args[]) {
    SimpleInterface si =
      new E10_InnerClassInScope().get();
    si.f();
  }
} /* Output:
SI.f
*///:~
```

The inner class remains visible only if the **return** statement is in its scope; if not, the inner class definition goes out of scope.

Exercise 11

```
//: innerclasses/E11_HiddenInnerClass.java
/****************** Exercise 11 *****************
 * Create a private inner class that implements a
 * public interface. Write a method that returns
 * a reference to an instance of the private
 * inner class, upcast to the interface. Show
 * that the inner class is completely hidden by
 * trying to downcast to it.
 ***********************************************/
package innerclasses;
import innerclasses.exercise6.*;

class Outer5 {
  private class Inner implements SimpleInterface {
    public void f() {
      System.out.println("Outer5.Inner.f");
    }
  }
  public SimpleInterface get() { return new Inner(); }
  public Inner get2() { return new Inner(); }
}

public class E11_HiddenInnerClass {
  public static void main(String args[]) {
    Outer5 out = new Outer5();
    SimpleInterface si = out.get();
    si = out.get2();
    // Won't compile -- 'Inner' not visible:
    //! Inner i1 = out.get2();
    //! Inner i2 = (Inner)si;
  }
} ///:~
```

The **public get( )** method returns the private class **Inner** instance, upcast to **SimpleInterface**.

Notice that **get2( )** returns an object of the private class **Inner**. However, when you call **get2( )** from outside of **Outer**, you can't use the return value's actual type because it's **private** and visible only inside the class. You can only upcast the return value to a visible base interface. Thus, **Outer** methods can use the actual type, while methods of other classes must use the upcast result.

Exercise 12

```
//: innerclasses/E12_AnonymousInnerClassAccess.java
/****************** Exercise 12 *****************
 * Repeat Exercise 7 using an anonymous inner
 * class.
 ************************************************/
package innerclasses;

public class E12_AnonymousInnerClassAccess {
  private int i = 10;
  private void f() {
    System.out.println("E12_AnonymousInnerClassAccess.f");
  }
  public void h() {
    new Object() {
      void g() {
        i++;
        f();
      }
    }.g();
    System.out.println("i = " + i);
  }
  public static void main(String args[]) {
    E12_AnonymousInnerClassAccess ica =
      new E12_AnonymousInnerClassAccess();
    ica.h();
  }
} /* Output:
E12_AnonymousInnerClassAccess.f
i = 11
*///:~
```

Exercise 13

```
//: innerclasses/E13_AnonymousInnerClassInMethod.java
/****************** Exercise 13 *****************
 * Repeat Exercise 9 using an anonymous inner
 * class.
 ************************************************/
package innerclasses;
import innerclasses.exercise6.*;

public class E13_AnonymousInnerClassInMethod {
  public SimpleInterface get() {
```

```
    return new SimpleInterface() {
      public void f() {
        System.out.println("SimpleInterface.f");
      }
    };
  }
  public static void main(String args[]) {
    SimpleInterface si =
      new E13_AnonymousInnerClassInMethod().get();
    si.f();
  }
} /* Output:
SimpleInterface.f
*///:~
```

Exercise 14

```
//: innerclasses/E14_HorrorShow2.java
/****************** Exercise 14 ******************
 * Modify interfaces/HorrorShow.java to implement
 * DangerousMonster and Vampire using anonymous
 * classes.
 ************************************************/
package innerclasses;

public class E14_HorrorShow2 {
  public static void main(String[] args) {
    DangerousMonster barney = new DangerousMonster() {
      public void menace() {}
      public void destroy() {}
    };
    HorrorShow.u(barney);
    HorrorShow.v(barney);
    Vampire vlad = new Vampire() {
      public void menace() {}
      public void destroy() {}
      public void kill() {}
      public void drinkBlood() {}
    };
    HorrorShow.u(vlad);
    HorrorShow.v(vlad);
    HorrorShow.w(vlad);
  }
} ///:~
```

Exercise 15

```
//: innerclasses/E15_ReturningAnonymousIC.java
/****************** Exercise 15 *****************
 * Create a class with a non-default constructor
 * (one with arguments) and no default constructor
 * (no "no-arg" constructor). Create a second class
 * with a method that returns a reference to
 * an object of the first class. Create the object
 * you return by making an anonymous inner
 * class inherit from the first class.
 ***********************************************/
package innerclasses;

class NoDefault {
  private int i;
  public NoDefault(int i) { this.i = i; }
  public void f() { System.out.println("NoDefault.f"); }
}

class Second {
  public NoDefault get1(int i) {
    // Doesn't override any methods:
    return new NoDefault(i) {};
  }
  public NoDefault get2(int i) {
    // Overrides f():
    return new NoDefault(i) {
      public void f() {
        System.out.println("Second.get2.f");
      }
    };
  }
}

public class E15_ReturningAnonymousIC {
  public static void main(String args[]) {
    Second sec = new Second();
    NoDefault nd = sec.get1(47);
    nd.f();
    nd = sec.get2(99);
    nd.f();
  }
} /* Output:
NoDefault.f
Second.get2.f
```

```
*///:~
```

In **get1( )**, you inherit **NoDefault** in the anonymous inner class without overriding any methods; usually you'll override a method when you inherit, as in **get2( )**.

Exercise 16

```
//: innerclasses/E16_AnonymousCycleFactories.java
/****************** Exercise 16 ******************
 * Use anonymous inner classes to modify the
 * solution to Exercise 18 from the Interfaces chapter.
 ***********************************************/
package innerclasses;

interface Cycle {
  int wheels();
}

interface CycleFactory {
  Cycle getCycle();
}

class Unicycle implements Cycle {
  public int wheels() { return 1; }
  public static CycleFactory factory =
    new CycleFactory() {
      public Unicycle getCycle() { return new Unicycle(); }
    };
}

class Bicycle implements Cycle {
  public int wheels() { return 2; }
  public static CycleFactory factory =
    new CycleFactory() {
      public Bicycle getCycle() { return new Bicycle(); }
    };
}

class Tricycle implements Cycle {
  public int wheels() { return 3; }
  public static CycleFactory factory =
    new CycleFactory() {
      public Tricycle getCycle() { return new Tricycle(); }
    };
}
```

```
public class E16_AnonymousCycleFactories {
  public static void ride(CycleFactory fact) {
     Cycle c = fact.getCycle();
     System.out.println("Num. of wheels: " + c.wheels());
  }
  public static void main(String[] args) {
    ride(Unicycle.factory);
    ride(Bicycle.factory);
    ride(Tricycle.factory);
  }
} /* Output:
Num. of wheels: 1
Num. of wheels: 2
Num. of wheels: 3
*///:~
```

Exercise 17

```
//: innerclasses/E17_AnonymousTossingFramework.java
/****************** Exercise 17 ******************
 * Use anonymous inner classes to modify the solution
 * to Exercise 19 from the Interfaces chapter.
 ***********************************************/
package innerclasses;

interface Tossing { boolean event(); }

interface TossingFactory { Tossing getTossing(); }

class CoinTossing implements Tossing {
  private int events;
  private static final int EVENTS = 2;
  public boolean event() {
    System.out.println("Coin tossing event " + events);
    return ++events != EVENTS;
  }
  public static TossingFactory factory =
    new TossingFactory() {
      public CoinTossing getTossing() {
        return new CoinTossing();
      }
    };
}

class DiceTossing implements Tossing {
```

```
  private int events;
  private static final int EVENTS = 6;
  public boolean event() {
    System.out.println("Dice tossing event " + events);
    return ++events != EVENTS;
  }
  public static TossingFactory factory =
    new TossingFactory() {
      public DiceTossing getTossing() {
        return new DiceTossing();
      }
    };
}

public class E17_AnonymousTossingFramework {
  public static void simulate(TossingFactory fact) {
    Tossing t = fact.getTossing();
    while(t.event())
      ;
  }
  public static void main(String[] args) {
    simulate(CoinTossing.factory);
    simulate(DiceTossing.factory);
  }
} /* Output:
Coin tossing event 0
Coin tossing event 1
Dice tossing event 0
Dice tossing event 1
Dice tossing event 2
Dice tossing event 3
Dice tossing event 4
Dice tossing event 5
*///:~
```

Exercise 18

```
//: innerclasses/E18_NestedClass.java
/****************** Exercise 18 *****************
 * Create a class containing a nested class.
 * In main(), create an instance of the nested
 * class.
 ************************************************/
package innerclasses;

public class E18_NestedClass {
```

```
  static class Nested {
    void f() { System.out.println("Nested.f"); }
  }
  public static void main(String args[]) {
    Nested ne = new Nested();
    ne.f();
  }
}

class Other {
  // Specifying the nested type outside
  // the scope of the class:
  void f() {
    E18_NestedClass.Nested ne =
      new E18_NestedClass.Nested();
  }
} /* Output:
Nested.f
*///:~
```

You can refer to just the class name when inside the method of a class with a defined nested (**static** inner) class, but outside the class, you must specify the outer class and nested class, as shown in **Other**, above.

Exercise 19

```
//: innerclasses/E19_InnerInsideInner.java
/****************** Exercise 19 *****************
 * Create a class containing an inner class that
 * itself contains an inner class. Repeat this
 * using static inner classes. Note the names of
 * the .class files produced by the compiler.
 ***********************************************/
package innerclasses;

public class E19_InnerInsideInner {
  class Inner1 {
    class Inner2 {
      void f() {}
    }
    Inner2 makeInner2() { return new Inner2(); }
  }
  Inner1 makeInner1() { return new Inner1(); }
  static class Nested1 {
    static class Nested2 {
      void f() {}
```

```
    }
    void f() {}
  }
  public static void main(String args[]) {
    new E19_InnerInsideInner.Nested1().f();
    new E19_InnerInsideInner.Nested1.Nested2().f();
    E19_InnerInsideInner x1 = new E19_InnerInsideInner();
    E19_InnerInsideInner.Inner1 x2 = x1.makeInner1();
    E19_InnerInsideInner.Inner1.Inner2 x3 =
      x2.makeInner2();
    x3.f();
  }
} ///:~
```

The class names produced are:

```
E19_InnerInsideInner.class
E19_InnerInsideInner$Inner1.class
E19_InnerInsideInner$Inner1$Inner2.class
E19_InnerInsideInner$Nested1.class
E19_InnerInsideInner$Nested1$Nested2.class
```

Exercise 20

```
//: innerclasses/E20_InterfaceWithNested.java
/****************** Exercise 20 *****************
 * Create an interface containing a nested class.
 * Implement this interface and create an
 * instance of the nested class.
 ************************************************/
package innerclasses;

interface WithNested {
  class Nested {
    int i;
    public Nested(int i) { this.i = i; }
    void f() { System.out.println("Nested.f"); }
  }
}

class B2 implements WithNested {}

public class E20_InterfaceWithNested {
  public static void main(String args[]) {
    B2 b = new B2();
    WithNested.Nested ne = new WithNested.Nested(5);
```

```
    ne.f();
  }
} /* Output:
Nested.f
*///:~
```

Even if an **interface** itself has no use, a nested class defined within it can still be useful. If we define **Nested** within **WithNested**, that just means we locate its name there, since all elements of an **interface** are **public**. **Nested** has no added access to the elements of **WithNested**.

Exercise 21

```
//: innerclasses/E21_InterfaceWithNested2.java
/****************** Exercise 21 ******************
 * Create an interface with a nested class
 * and a static method that calls the methods
 * of your interface and displays the results.
 * Implement your interface and pass an instance of
 * your implementation to the method.
 ***********************************************/
package innerclasses;

interface I {
  void f();
  void g();
  class Nested {
    static void call(I impl) {
      System.out.println("Calling I.f()");
      impl.f();
      System.out.println("Calling I.g()");
      impl.g();
    }
  }
}

public class E21_InterfaceWithNested2 {
  public static void main(String[] args) {
    I impl = new I() {
      public void f() {}
      public void g() {}
    };
    I.Nested.call(impl);
  }
} /* Output:
Calling I.f()
```

```
Calling I.g()
*///:~
```

Notice that we use an anonymous inner class to implement interface **I**. It's generally clearer to list all methods of an interface, then define nested classes.

Exercise 22

```
//: innerclasses/E22_GetRSelector.java
/****************** Exercise 22 *****************
 * Implement reverseSelector() in Sequence.java.
 ***********************************************/
package innerclasses;

class Sequence3 {
  private Object[] objects;
  private int next;
  public Sequence3(int size) {
    objects = new Object[size];
  }
  public void add(Object x) {
    if(next < objects.length)
      objects[next++] = x;
  }
  private class ReverseSelector implements Selector {
    int i = objects.length - 1;
    public boolean end() { return i < 0; }
    public Object current() { return objects[i]; }
    public void next() { if(i >= 0) i--; }
  }
  private class SequenceSelector implements Selector {
    private int i;
    public boolean end() { return i == objects.length; }
    public Object current() { return objects[i]; }
    public void next() { if(i < objects.length) i++; }
  }
  public Selector selector() {
    return new SequenceSelector();
  }
  public Selector reverseSelector() {
    return new ReverseSelector();
  }
}
public class E22_GetRSelector {
  public static void main(String[] args) {
    Sequence3 sequence = new Sequence3(10);
```

```
    for(int i = 0; i < 10; i++)
      sequence.add(Integer.toString(i));
    Selector selector = sequence.reverseSelector();
    while(!selector.end()) {
      System.out.print(selector.current() + " ");
      selector.next();
    }
  }
} /* Output:
9 8 7 6 5 4 3 2 1 0
*///:~
```

This is a copy-and-paste solution with minor logic changes in the **ReverseSelector** class.

Exercise 23

```
//: innerclasses/E23_UAB.java
/****************** Exercise 23 *****************
 * Create an interface U with three methods.
 * Create a class A with a method that produces a
 * reference to a U by building an anonymous
 * inner class. Create a second class B that
 * contains an array of U. B should have one
 * method that accepts and stores a reference to
 * a U in the array, a second method that sets a
 * reference in the array (specified by the
 * method argument) to null, and a third method
 * that moves through the array and calls the
 * methods in U. In main(), create a group of A
 * objects and a single B. Fill the B with U
 * references produced by the A objects. Use the
 * B to call back into all the A objects. Remove
 * some of the U references from the B.
 *********************************************/
package innerclasses;

interface U {
  void f();
  void g();
  void h();
}

class A {
  public U getU() {
    return new U() {
```

```
      public void f() { System.out.println("A.f"); }
      public void g() { System.out.println("A.g"); }
      public void h() { System.out.println("A.h"); }
    };
  }
}

class B {
  U[] ua;
  public B(int size) {
    ua = new U[size];
  }
  public boolean add(U elem) {
    for(int i = 0; i < ua.length; i++) {
      if(ua[i] == null) {
        ua[i] = elem;
        return true;
      }
    }
    return false; // Couldn't find any space
  }
  public boolean setNull(int i) {
    if(i < 0 || i >= ua.length)
      return false; // Value out of bounds
    // (Normally throw an exception)
    ua[i] = null;
    return true;
  }
  public void callMethods() {
    for(int i = 0; i < ua.length; i++)
      if(ua[i] != null) {
        ua[i].f();
        ua[i].g();
        ua[i].h();
      }
  }
}

public class E23_UAB {
  public static void main(String args[]) {
    A[] aa = { new A(), new A(), new A() };
    B b = new B(3);
    for(int i = 0; i < aa.length; i++)
      b.add(aa[i].getU());
    b.callMethods();
    System.out.println("****");
    b.setNull(0);
```

```
    b.callMethods();
  }
} /* Output:
A.f
A.g
A.h
A.f
A.g
A.h
A.f
A.g
A.h
****
A.f
A.g
A.h
A.f
A.g
A.h
*///:~
```

Notice that we remove the zero element.

Exercise 24

```
//: innerclasses/E24_GreenhouseInnerEvent.java
// {Args: 5000}
/****************** Exercise 24 *****************
 * Add Event inner classes that turn fans on and
 * off in GreenhouseControls.java. Configure
 * GreenhouseController.java to use these new
 * Event objects.
 ***********************************************/
package innerclasses;
import innerclasses.controller.*;

class GreenhouseControlsWithFan extends Controller {
  private boolean light = false;
  public class LightOn extends Event {
    public LightOn(long delayTime) { super(delayTime); }
    public void action() {
      // Put hardware control code here to
      // physically turn on the light.
      light = true;
    }
    public String toString() { return "Light is on"; }
```

```
  }
  public class LightOff extends Event {
    public LightOff(long delayTime) { super(delayTime); }
    public void action() {
      // Put hardware control code here to
      // physically turn off the light.
      light = false;
    }
    public String toString() { return "Light is off"; }
  }
  private boolean fan = false;
  public class FanOn extends Event {
    public FanOn(long delayTime) { super(delayTime); }
    public void action() {
      // Put hardware control code here to
      // physically turn on the Fan.
      fan = true;
    }
    public String toString() { return "Fan is on"; }
  }
  public class FanOff extends Event {
    public FanOff(long delayTime) { super(delayTime); }
    public void action() {
      // Put hardware control code here to
      // physically turn off the Fan.
      fan = false;
    }
    public String toString() { return "Fan is off"; }
  }
  private boolean water = false;
  public class WaterOn extends Event {
    public WaterOn(long delayTime) { super(delayTime); }
    public void action() {
      // Put hardware control code here.
      water = true;
    }
    public String toString() {
      return "Greenhouse water is on";
    }
  }
  public class WaterOff extends Event {
    public WaterOff(long delayTime) { super(delayTime); }
    public void action() {
      // Put hardware control code here.
      water = false;
    }
    public String toString() {
```

```
      return "Greenhouse water is off";
    }
  }
  private String thermostat = "Day";
  public class ThermostatNight extends Event {
    public ThermostatNight(long delayTime) {
      super(delayTime);
    }
    public void action() {
      // Put hardware control code here.
      thermostat = "Night";
    }
    public String toString() {
      return "Thermostat on night setting";
    }
  }
  public class ThermostatDay extends Event {
    public ThermostatDay(long delayTime) {
      super(delayTime);
    }
    public void action() {
      // Put hardware control code here.
      thermostat = "Day";
    }
    public String toString() {
      return "Thermostat on day setting";
    }
  }
  // An example of an action() that inserts a
  // new one of itself into the event list:
  public class Bell extends Event {
    public Bell(long delayTime) { super(delayTime); }
    public void action() {
      addEvent(new Bell(delayTime));
    }
    public String toString() { return "Bing!"; }
  }
  public class Restart extends Event {
    private Event[] eventList;
    public Restart(long delayTime, Event[] eventList) {
      super(delayTime);
      this.eventList = eventList;
      for(Event e : eventList)
        addEvent(e);
    }
    public void action() {
      for(Event e : eventList) {
```

```
        e.start(); // Rerun each event
        addEvent(e);
      }
      start(); // Rerun this Event
      addEvent(this);
    }
    public String toString() {
      return "Restarting system";
    }
  }
  public static class Terminate extends Event {
    public Terminate(long delayTime) { super(delayTime); }
    public void action() { System.exit(0); }
    public String toString() { return "Terminating";  }
  }
}

public class E24_GreenhouseInnerEvent {
  public static void main(String[] args) {
    GreenhouseControlsWithFan gc =
      new GreenhouseControlsWithFan();
    // Instead of hard-wiring, you could parse
    // configuration information from a text file here:
    gc.addEvent(gc.new Bell(900));
    Event[] eventList = {
      gc.new ThermostatNight(0),
      gc.new LightOn(200),
      gc.new FanOn(300),
      gc.new LightOff(400),
      gc.new FanOff(500),
      gc.new WaterOn(600),
      gc.new WaterOff(800),
      gc.new ThermostatDay(1400)
    };
    gc.addEvent(gc.new Restart(2000, eventList));
    if(args.length == 1)
      gc.addEvent(new GreenhouseControlsWithFan
        .Terminate(new Integer(args[0])));
    gc.run();
  }
} /* Output:
Bing!
Thermostat on night setting
Light is on
Fan is on
Light is off
Fan is off
```

```
Greenhouse water is on
Greenhouse water is off
Thermostat on day setting
Restarting system
Terminating
*///:~
```

This is basically a copy and paste exercise, but it helps ensure that you understand the structure of the program.

Exercise 25

```
//: innerclasses/E25_GreenhouseController.java
// {Args: 5000}
/****************** Exercise 25 ******************
 * Inherit from GreenhouseControls in
 * GreenhouseControls.java to add Event inner
 * classes that turn water mist generators on
 * and off. Write a new version of
 * GreenhouseController.java to use these new
 * Event objects.
 ************************************************/
package innerclasses;
import innerclasses.controller.*;

class GreenhouseControlsWithWMG extends GreenhouseControls {
  private boolean generator = false;
  public class WatermistGeneratorOn extends Event {
    public WatermistGeneratorOn(long delayTime) {
      super(delayTime);
    }
    public void action() {
      generator = true;
    }
    public String toString() {
      return "Water mist generator is on";
    }
  }
  public class WatermistGeneratorOff extends Event {
    public WatermistGeneratorOff(long delayTime) {
      super(delayTime);
    }
    public void action() {
      generator = false;
    }
    public String toString() {
```

```
      return "Water mist generator is off";
    }
  }
}

public class E25_GreenhouseController {
  public static void main(String[] args) {
    GreenhouseControlsWithWMG gc =
      new GreenhouseControlsWithWMG();
    // Instead of hard-wiring, you could parse
    // configuration information from a text file here:
    gc.addEvent(gc.new Bell(900));
    Event[] eventList = {
      gc.new ThermostatNight(0),
      gc.new LightOn(200),
      gc.new LightOff(400),
      gc.new WaterOn(600),
      gc.new WaterOff(800),
      gc.new ThermostatDay(1400),
      gc.new WatermistGeneratorOn(1600),
      gc.new WatermistGeneratorOff(1800)
    };
    gc.addEvent(gc.new Restart(2000, eventList));
    if(args.length == 1)
      gc.addEvent(new GreenhouseControlsWithWMG
        .Terminate(new Integer(args[0])));
    gc.run();
  }
} /* Output:
Bing!
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Thermostat on day setting
Water mist generator is on
Water mist generator is off
Restarting system
Terminating
*///:~
```

Exercise 26

```
//: innerclasses/E26_InnerClassInheritance.java
/****************** Exercise 26 *****************
```

```
 * Create a class with an inner class that has a
 * non-default constructor (one that takes
 * arguments). Create a second class with an inner
 * class that inherits from the first inner class.
 *********************************************/
package innerclasses;

class WithNonDefault {
  class Inner {
    int i;
    public Inner(int i) { this.i = i; }
    public Inner() { i = 47; }
    public void f() { System.out.println("Inner.f"); }
  }
}

public class E26_InnerClassInheritance {
  class Inner2 extends WithNonDefault.Inner {
    // Won't compile -- WithNonDefault not available:
    //! public Inner2(int i) {
    //!   super(i);
    //! }
    public Inner2(WithNonDefault wnd, int i) {
      wnd.super(i);
    }
    public void f() {
      System.out.println("Inner2.f");
      super.f();
    }
  }
  public static void main(String args[]) {
    WithNonDefault wnd = new WithNonDefault();
    E26_InnerClassInheritance ici =
      new E26_InnerClassInheritance();
    Inner2 i2 = ici.new Inner2(wnd, 47);
    i2.f();
  }
} /* Output:
Inner2.f
Inner.f
*///:~
```

We use the **new** expression from the outer class object to create an instance of an inner class. To create an instance of one inner class inheriting from another, we provide the constructor with an instance of the outer base class, so creating a new **Inner2** object is doubly complex. However, the **Inner2** object now has an intimate connection with the objects **WithNonDefault** and

E26_InnerClassInheritance; this creates a private mediation between the two. (See *Mediator* in the design patterns literature.)

Holding Your Objects

Exercise 1

```
//: holding/E01_Gerbil.java
/****************** Exercise 1 ******************
 * Create a new class called Gerbil with an int
 * gerbilNumber initialized in the constructor.
 * Give it a method called hop() that prints out
 * which gerbil number this is, and that it's hopping.
 * Create an ArrayList and add Gerbil objects to
 * the List. Now use the get() method to move
 * through the List and call hop() for each Gerbil.
 ***********************************************/
package holding;
import java.util.*;

class Gerbil {
  private final int gerbilNumber;
  Gerbil(int gerbilNumber) {
    this.gerbilNumber = gerbilNumber;
  }
  public String toString() {
    return "gerbil " + gerbilNumber;
  }
  public void hop() {
    System.out.println(this + " is hopping");
  }
}

public class E01_Gerbil {
  public static void main(String args[]) {
    ArrayList<Gerbil> gerbils = new ArrayList<Gerbil>();
    for(int i = 0; i < 10; i++)
      gerbils.add(new Gerbil(i));
    for(int i = 0; i < gerbils.size(); i++)
      gerbils.get(i).hop();
  }
} /* Output:
gerbil 0 is hopping
gerbil 1 is hopping
gerbil 2 is hopping
```

```
gerbil 3 is hopping
gerbil 4 is hopping
gerbil 5 is hopping
gerbil 6 is hopping
gerbil 7 is hopping
gerbil 8 is hopping
gerbil 9 is hopping
*///:~
```

Exercise 2

```
//: holding/E02_SimpleCollection2.java
/****************** Exercise 2 *****************
 * Modify SimpleCollection.java to use a Set for c.
 ***********************************************/
package holding;
import java.util.*;

public class E02_SimpleCollection2 {
  public static void main(String[] args) {
    Collection<Integer> c = new HashSet<Integer>();
    for(int i = 0; i < 10; i++)
      c.add(i); // Autoboxing
    for(Integer i : c)
      System.out.print(i + ", ");
  }
} /* Output:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
*///:~
```

Generally, a **Set** does not hold elements in insertion order.

Exercise 3

```
//: holding/E03_UnlimitedSequence.java
/****************** Exercise 3 *****************
 * Modify innerclasses/Sequence.java so you
 * can add any number of elements to it.
 ***********************************************/
package holding;
import java.util.*;

class UnlimitedSequence {
  private final List<Object> items =
    new ArrayList<Object>();
```

```
  public void add(Object x) { items.add(x); }
  private class SequenceSelector implements Selector {
    private int i;
    public boolean end() { return i == items.size(); }
    public Object current() { return items.get(i); }
    public void next() { if(i < items.size()) i++; }
  }
  public Selector selector() {
    return new SequenceSelector();
  }
}

public class E03_UnlimitedSequence {
  public static void main(String[] args) {
    UnlimitedSequence sequence = new UnlimitedSequence();
    for(int i = 0; i < 10; i++)
      sequence.add(Integer.toString(i));
    Selector selector = sequence.selector();
    while(!selector.end()) {
      System.out.print(selector.current() + " ");
      selector.next();
    }
  }
} /* Output:
0 1 2 3 4 5 6 7 8 9
*///:~
```

Exercise 4

```
//: holding/E04_MovieNameGenerator.java
/****************** Exercise 4 ******************
 * Create a generator class that produces String objects
 * with the names of characters from your favorite
 * movie each time you call next(), and then loops
 * around to the beginning of the character list
 * when it runs out of names. Use this generator to
 * fill an array, an ArrayList, a LinkedList, a
 * HashSet, a LinkedHashSet, and a TreeSet, then
 * print each container.
 *********************************************/
package holding;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class MovieNameGenerator implements Generator<String> {
```

```
  String[] characters = {
    "Grumpy", "Happy", "Sleepy", "Dopey", "Doc", "Sneezy",
    "Bashful", "Snow White", "Witch Queen", "Prince"
  };
  int next;
  public String next() {
    String r = characters[next];
    next = (next + 1) % characters.length;
    return r;
  }
}

public class E04_MovieNameGenerator {
  private static final MovieNameGenerator mng =
    new MovieNameGenerator();
  static String[] fill(String[] array) {
    for(int i = 0; i < array.length; i++)
      array[i] = mng.next();
    return array;
  }
  static Collection<String>
  fill(Collection<String> collection) {
    for(int i = 0; i < 5; i++)
      collection.add(mng.next());
    return collection;
  }
  public static void main(String[] args) {
    print(Arrays.toString(fill(new String[5])));
    print(fill(new ArrayList<String>()));
    print(fill(new LinkedList<String>()));
    print(fill(new HashSet<String>()));
    print(fill(new LinkedHashSet<String>()));
    print(fill(new TreeSet<String>()));
  }
} /* Output:
[Grumpy, Happy, Sleepy, Dopey, Doc]
[Sneezy, Bashful, Snow White, Witch Queen, Prince]
[Grumpy, Happy, Sleepy, Dopey, Doc]
[Snow White, Witch Queen, Sneezy, Bashful, Prince]
[Grumpy, Happy, Sleepy, Dopey, Doc]
[Bashful, Prince, Sneezy, Snow White, Witch Queen]
*///:~
```

All data generators use the simple **Generator<T>** parameterized interface, introduced later in *TIJ4*.

Exercise 5

```
//: holding/E05_IntegerListFeatures.java
/****************** Exercise 5 ******************
 * Use Integers instead of Pets to modify
 * ListFeatures.java  (remember autoboxing).
 * Explain any difference in results.
 *********************************************/
package holding;
import java.util.*;
import static net.mindview.util.Print.*;

public class E05_IntegerListFeatures {
  static Random rand = new Random(47);
  public static void main(String[] args) {
    List<Integer> ints = new ArrayList<Integer>(
      Arrays.asList(1, 2, 3, 4, 5, 6, 7));
    print("1: " + ints);
    ints.add(8);
    print("2: " + ints);
    print("3: " + ints.contains(8));
    ints.remove(Integer.valueOf(8));
    Integer i = ints.get(2);
    print("4: " +  i + " " + ints.indexOf(i));
    Integer j = Integer.valueOf(1);
    print("5: " + ints.indexOf(j));
    print("6: " + ints.remove(j));
    print("7: " + ints.remove(i));
    print("8: " + ints);
    ints.add(3, 0);
    print("9: " + ints);
    List<Integer> sub = ints.subList(1, 4);
    print("subList: " + sub);
    print("10: " + ints.containsAll(sub));
    Collections.sort(sub);
    print("sorted subList: " + sub);
    print("11: " + ints.containsAll(sub));
    Collections.shuffle(sub, rand);
    print("shuffled subList: " + sub);
    print("12: " + ints.containsAll(sub));
    List<Integer> copy = new ArrayList<Integer>(ints);
    sub = Arrays.asList(ints.get(1), ints.get(4));
    print("sub: " + sub);
    copy.retainAll(sub);
    print("13: " + copy);
    copy = new ArrayList<Integer>(ints);
```

```
    copy.remove(2);
    print("14: " + copy);
    copy.removeAll(sub);
    print("15: " + copy);
    copy.set(1, 9);
    print("16: " + copy);
    copy.addAll(2, sub);
    print("17: " + copy);
    print("18: " + ints.isEmpty());
    ints.clear();
    print("19: " + ints);
    print("20: " + ints.isEmpty());
    ints.addAll(Arrays.asList(1, 2, 3, 4));
    print("21: " + ints);
    Object[] o = ints.toArray();
    print("22: " + o[3]);
    Integer[] ia = ints.toArray(new Integer[0]);
    print("22: " + ia[3]);
  }
} /* Output:
1: [1, 2, 3, 4, 5, 6, 7]
2: [1, 2, 3, 4, 5, 6, 7, 8]
3: true
4: 3 2
5: 0
6: true
7: true
8: [2, 4, 5, 6, 7]
9: [2, 4, 5, 0, 6, 7]
subList: [4, 5, 0]
10: true
sorted subList: [0, 4, 5]
11: true
shuffled subList: [4, 0, 5]
12: true
sub: [4, 6]
13: [4, 6]
14: [2, 4, 5, 6, 7]
15: [2, 5, 7]
16: [2, 9, 7]
17: [2, 9, 4, 6, 7]
18: false
19: []
20: true
21: [1, 2, 3, 4]
22: 4
22: 4
```

```
*///:~
```

List behavior varies depending on **equals( )**, as *TIJ4* explains. Two **Integer**s are equal if their contents are identical in the output.

Be vigilant with overloaded methods like **remove( )**; it's easy to make mistakes due to autoboxing. If, for example, you type **remove(2)** instead of **remove(Integer.valueOf(2))** you remove the third *element* from the list (as the first element's index is `0`), instead of an element whose *value is 2*.

Exercise 6

```
//: holding/E06_StringListFeatures.java
/****************** Exercise 6 *****************
 * Using Strings instead of Pets, modify
 * ListFeatures.java . Explain any difference in
 * results.
 *********************************************/
package holding;
import java.util.*;
import static net.mindview.util.Print.*;

public class E06_StringListFeatures {
  static Random rand = new Random(47);
  public static void main(String[] args) {
    List<String> strs = new ArrayList<String>(
      Arrays.asList("A", "B", "C", "D", "E", "F", "G"));
    print("1: " + strs);
    strs.add("H");
    print("2: " + strs);
    print("3: " + strs.contains("H"));
    strs.remove("H");
    String s1 = strs.get(2);
    print("4: " +  s1 + " " + strs.indexOf(s1));
    String s2 = "A";
    print("5: " + strs.indexOf(s2));
    print("6: " + strs.remove(s2));
    print("7: " + strs.remove(s1));
    print("8: " + strs);
    strs.add(3, "0");
    print("9: " + strs);
    List<String> sub = strs.subList(1, 4);
    print("subList: " + sub);
    print("10: " + strs.containsAll(sub));
    Collections.sort(sub);
    print("sorted subList: " + sub);
```

```
    print("11: " + strs.containsAll(sub));
    Collections.shuffle(sub, rand);
    print("shuffled subList: " + sub);
    print("12: " + strs.containsAll(sub));
    List<String> copy = new ArrayList<String>(strs);
    sub = Arrays.asList(strs.get(1), strs.get(4));
    print("sub: " + sub);
    copy.retainAll(sub);
    print("13: " + copy);
    copy = new ArrayList<String>(strs);
    copy.remove(2);
    print("14: " + copy);
    copy.removeAll(sub);
    print("15: " + copy);
    copy.set(1, "I");
    print("16: " + copy);
    copy.addAll(2, sub);
    print("17: " + copy);
    print("18: " + strs.isEmpty());
    strs.clear();
    print("19: " + strs);
    print("20: " + strs.isEmpty());
    strs.addAll(Arrays.asList("A", "B", "C", "D"));
    print("21: " + strs);
    Object[] o = strs.toArray();
    print("22: " + o[3]);
    String[] sa = strs.toArray(new String[0]);
    print("22: " + sa[3]);
  }
} /* Output:
1: [A, B, C, D, E, F, G]
2: [A, B, C, D, E, F, G, H]
3: true
4: C 2
5: 0
6: true
7: true
8: [B, D, E, F, G]
9: [B, D, E, 0, F, G]
subList: [D, E, 0]
10: true
sorted subList: [0, D, E]
11: true
shuffled subList: [D, 0, E]
12: true
sub: [D, F]
13: [D, F]
```

```
14: [B, D, E, F, G]
15: [B, E, G]
16: [B, I, G]
17: [B, I, D, F, G]
18: false
19: []
20: true
21: [A, B, C, D]
22: D
22: D
*///:~
```

Again, be careful when using overloaded methods.

Exercise 7

```
//: holding/E07_TestList.java
/****************** Exercise 7 *****************
 * Create a class and make an initialized array
 * of your class objects. Fill a List from
 * your array. Create a subset of your List using
 * subList(), then remove this subset from
 * your List.
 *********************************************/
package holding;
import java.util.*;

class IDClass {
  private static int counter;
  private int count = counter++;
  public String toString() {
    return "IDClass " + count;
  }
}

public class E07_TestList {
  public static void main(String args[]) {
    IDClass[] idc = new IDClass[10];
    for(int i = 0; i < idc.length; i++)
      idc[i] = new IDClass();
    List<IDClass> lst = new ArrayList<IDClass>(
      Arrays.asList(idc));
    System.out.println("lst = " + lst);
    List<IDClass> subSet =
      lst.subList(lst.size()/4, lst.size()/2);
    System.out.println("subSet = " + subSet);
```

```
    // The semantics of the sub list become undefined if the
    // backing list is structurally modified!
    //! lst.removeAll(subSet);
    subSet.clear();
    System.out.println("lst = " + lst);
  }
} /* Output:
lst = [IDClass 0, IDClass 1, IDClass 2, IDClass 3, IDClass
4, IDClass 5, IDClass 6, IDClass 7, IDClass 8, IDClass 9]
subSet = [IDClass 2, IDClass 3, IDClass 4]
lst = [IDClass 0, IDClass 1, IDClass 5, IDClass 6, IDClass
7, IDClass 8, IDClass 9]
*///:~
```

The methods **asList( )** and **subList( )** return immutable **Lists** because they are "backed" by the underlying array and list, respectively. If you structurally modify the backing list as we did in the commented-out section, you get a concurrent modification exception. Therefore, the program operates on the sublist instead of the backing list. Alternatively, avoid errors by creating a separate copy of the returned sublist and use that as an argument to **removeAll( )**.

Exercise 8

```
//: holding/E08_GerbilIterator.java
/****************** Exercise 8 ******************
 * Modify Exercise 1 so it uses an Iterator to
 * move through the List while calling hop().
 ************************************************/
package holding;
import java.util.*;

public class E08_GerbilIterator {
  public static void main(String args[]) {
    ArrayList<Gerbil> gerbils = new ArrayList<Gerbil>();
    for(int i = 0; i < 10; i++)
      gerbils.add(new Gerbil(i));
    for(Iterator<Gerbil> it = gerbils.iterator();
        it.hasNext();
        it.next().hop());
  }
} /* Output:
gerbil 0 is hopping
gerbil 1 is hopping
gerbil 2 is hopping
gerbil 3 is hopping
```

```
gerbil 4 is hopping
gerbil 5 is hopping
gerbil 6 is hopping
gerbil 7 is hopping
gerbil 8 is hopping
gerbil 9 is hopping
*///:~
```

Exercise 9

```
//: holding/E09_SequenceIterator.java
/****************** Exercise 9 ******************
 * Modify innerclasses/Sequence.java so that
 * Sequence works with an Iterator instead of a
 * Selector.
 ************************************************/
package holding;
import java.util.*;

class Sequence2 {
  private Object[] items;
  private int next;
  public Sequence2(int size) { items = new Object[size]; }
  public void add(Object x) {
    if(next < items.length)
      items[next++] = x;
  }
  private class SequenceIterator
  implements Iterator<Object> {
    private int i;
    public boolean hasNext() { return i < items.length; }
    public Object next() {
      if(hasNext())
        return items[i++];
      throw new NoSuchElementException();
    }
    public void remove() {
      throw new UnsupportedOperationException();
    }
  }
  public Iterator<Object> iterator() {
    return new SequenceIterator();
  }
}

public class E09_SequenceIterator {
```

```
  public static void main(String[] args) {
    Sequence2 sequence = new Sequence2(10);
    for(int i = 0; i < 10; i++)
      sequence.add(Integer.toString(i));
    for(Iterator<Object> it = sequence.iterator();
        it.hasNext();)
      System.out.print(it.next() + " ");
  }
} /* Output:
0 1 2 3 4 5 6 7 8 9
*///:~
```

Exercise 10

```
//: holding/E10_RodentIterator.java
/****************** Exercise 10 ******************
 * change Exercise 9 in the Polymorphism chapter
 * to use an ArrayList to hold the Rodents and an
 * Iterator to move through their sequence.
 ***********************************************/
package holding;
import java.util.*;
import static net.mindview.util.Print.*;

class Rodent {
  public void hop() { print("Rodent hopping"); }
  public void scurry() { print("Rodent scurrying"); }
  public void reproduce() { print("Making more Rodents"); }
  public String toString() { return "Rodent"; }
}

class Mouse extends Rodent {
  public void hop() { print("Mouse hopping"); }
  public void scurry() { print("Mouse scurrying"); }
  public void reproduce() { print("Making more Mice"); }
  public String toString() { return "Mouse"; }
}

class Hamster extends Rodent {
  public void hop() { print("Hamster hopping"); }
  public void scurry() { print("Hamster scurrying"); }
  public void reproduce() { print("Making more Hamsters"); }
  public String toString() { return "Hamster"; }
}

public class E10_RodentIterator {
```

```
  public static void main(String args[]) {
    ArrayList<Rodent> rodents = new ArrayList<Rodent>(
      Arrays.asList(
        new Rodent(), new Mouse(), new Hamster()));
    Rodent r;
    for(Iterator<Rodent> it = rodents.iterator();
        it.hasNext();) {
      r = it.next();
      r.hop();
      r.scurry();
      r.reproduce();
      print(r);
    }
  }
} /* Output:
Rodent hopping
Rodent scurrying
Making more Rodents
Rodent
Mouse hopping
Mouse scurrying
Making more Mice
Mouse
Hamster hopping
Hamster scurrying
Making more Hamsters
Hamster
*///:~
```

Exercise 11

```
//: holding/E11_IterToString.java
/****************** Exercise 11 *****************
 * Write a method that uses an Iterator to step
 * through a Collection and print the toString()
 * of each object in the container. Fill all the
 * different types of Collections with objects and
 * apply your method to each container.
 ***********************************************/
package holding;
import java.util.*;

public class E11_IterToString {
  public static void printToStrings(Iterator<?> it) {
    while(it.hasNext())
      System.out.println(it.next().toString());
```

```
  }
  @SuppressWarnings("unchecked")
  public static void main(String args[]) {
    List<Collection<String>> ca =
      Arrays.<Collection<String>>asList(
        new ArrayList<String>(),
        new LinkedList<String>(),
        new HashSet<String>(),
        new TreeSet<String>());
    for(Collection<String> c : ca)
      E04_MovieNameGenerator.fill(c);
    for(Collection<String> c : ca)
      printToStrings(c.iterator());
  }
} /* Output:
Grumpy
Happy
Sleepy
Dopey
Doc
Sneezy
Bashful
Snow White
Witch Queen
Prince
Happy
Doc
Sleepy
Grumpy
Dopey
Bashful
Prince
Sneezy
Snow White
Witch Queen
*///:~
```

Exercise 12

```
//: holding/E12_ListIterators.java
/****************** Exercise 12 *****************
 * Create two List<Integer>s of the same size,
 * and populate one of them. Use ListIterators
 * to insert elements from the first List into
 * the second in reverse order. (You may want to
 * explore a number of different ways to solve
```

```
 * this problem.)
 *********************************************/
package holding;
import java.util.*;

public class E12_ListIterators {
  static void reverse(List<Integer> list) {
    ListIterator<Integer> fwd = list.listIterator();
    ListIterator<Integer> rev =
      list.listIterator(list.size());
    int mid = list.size() >> 1;
    for(int i = 0; i < mid; i++) {
      Integer tmp = fwd.next();
      fwd.set(rev.previous());
      rev.set(tmp);
    }
  }
  public static void main(String[] args) {
    List<Integer> src =
      Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    List<Integer> dest = new LinkedList<Integer>(src);
    System.out.println("source: " + src);
    System.out.println("destination: " + dest);
    reverse(dest);
    System.out.println("source: " + src);
    System.out.println("destination: " + dest);
  }
} /* Output:
source: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
destination: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
source: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
destination: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
*///:~
```

We fill the destination list with elements from the source list, then reverse their order using **reverse( )**. Practice by finding other solutions.

Exercise 13

```
//: holding/E13_GreenhouseLinkedList.java
// {Args: 5000}
/****************** Exercise 13 *****************
 * In the innerclasses/GreenhouseController.java
 * example, the class Controller uses an ArrayList.
 * Change the code to use a LinkedList instead, and
 * use an Iterator to cycle through the set of events.
```

```
 *********************************************/
package holding;
import java.util.*;
import innerclasses.controller.Event;

class Controller {
  // List changed to a LinkedList:
  private List<Event> eventList =
    new LinkedList<Event>();
  public void addEvent(Event c) { eventList.add(c); }
  public void run() {
    while(eventList.size() > 0) {
      Iterator<Event> it =
        new LinkedList<Event>(eventList).iterator();
      while(it.hasNext()) {
        Event e = it.next();
        if(e.ready()) {
          System.out.println(e);
          e.action();
          it.remove();
        }
      }
    }
  }
}

class GreenhouseControls extends Controller {
  private boolean light = false;
  public class LightOn extends Event {
    public LightOn(long delayTime) { super(delayTime); }
    public void action() { light = true; }
    public String toString() { return "Light is on"; }
  }
  public class LightOff extends Event {
    public LightOff(long delayTime) { super(delayTime); }
    public void action() { light = false; }
    public String toString() { return "Light is off"; }
  }
  private boolean water = false;
  public class WaterOn extends Event {
    public WaterOn(long delayTime) { super(delayTime); }
    public void action() { water = true; }
    public String toString() {
      return "Greenhouse water is on";
    }
  }
  public class WaterOff extends Event {
```

```
    public WaterOff(long delayTime) { super(delayTime); }
    public void action() { water = false; }
    public String toString() {
      return "Greenhouse water is off";
    }
  }
  private String thermostat = "Day";
  public class ThermostatNight extends Event {
    public ThermostatNight(long delayTime) {
      super(delayTime);
    }
    public void action() { thermostat = "Night"; }
    public String toString() {
      return "Thermostat on night setting";
    }
  }
  public class ThermostatDay extends Event {
    public ThermostatDay(long delayTime) {
      super(delayTime);
    }
    public void action() { thermostat = "Day"; }
    public String toString() {
      return "Thermostat on day setting";
    }
  }
  public class Bell extends Event {
    public Bell(long delayTime) { super(delayTime); }
    public void action() {
      addEvent(new Bell(delayTime));
    }
    public String toString() { return "Bing!"; }
  }
  public class Restart extends Event {
    private Event[] eventList;
    public Restart(long delayTime, Event[] eventList) {
      super(delayTime);
      this.eventList = eventList;
      for(Event e : eventList)
        addEvent(e);
    }
    public void action() {
      for(Event e : eventList) {
        e.start();
        addEvent(e);
      }
      start();
      addEvent(this);
```

```
    }
    public String toString() {
      return "Restarting system";
    }
  }
  public static class Terminate extends Event {
    public Terminate(long delayTime) { super(delayTime); }
    public void action() { System.exit(0); }
    public String toString() { return "Terminating"; }
  }
}

public class E13_GreenhouseLinkedList {
  public static void main(String[] args) {
    GreenhouseControls gc = new GreenhouseControls();
    gc.addEvent(gc.new Bell(900));
    Event[] eventList = {
      gc.new ThermostatNight(0),
      gc.new LightOn(200),
      gc.new LightOff(400),
      gc.new WaterOn(600),
      gc.new WaterOff(800),
      gc.new ThermostatDay(1400)
    };
    gc.addEvent(gc.new Restart(2000, eventList));
    if(args.length == 1)
      gc.addEvent(
        new GreenhouseControls.Terminate(
          new Integer(args[0])));
    gc.run();
  }
} /* Output:
Bing!
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Thermostat on day setting
Restarting system
Terminating
*///:~
```

Here we simply copy all the classes from *TIJ4* into a single file, only modifying **Controller**.

Exercise 14

```
//: holding/E14_MiddleInsertion.java
/****************** Exercise 14 *****************
 * Create an empty LinkedList<Integer>. Using a
 * ListIterator, add Integers to the List by always
 * inserting them in the middle of the List.
 ************************************************/
package holding;
import java.util.*;

public class E14_MiddleInsertion {
  public static void main(String[] args) {
    LinkedList<Integer> list = new LinkedList<Integer>();
    ListIterator<Integer> it = list.listIterator();
    for(int i = 1; i <= 10; i++) {
      it.add(i);
      if(i % 2 == 0)
        it.previous();
    }
    System.out.println(list);
  }
} /* Output:
[1, 3, 5, 7, 9, 10, 8, 6, 4, 2]
*///:~
```

You must calibrate the implicit cursor position every second iteration to keep the insertion point at the middle of the list.

Exercise 15

```
//: holding/E15_Evaluator.java
/****************** Exercise 15 *****************
 * Stacks are often used to evaluate expressions
 * in programming languages. Using
 * net.mindview.util.Stack, evaluate the following
 * expression, where '+' means "push the following
 * letter onto the stack," and '-' means "pop the
 * top of the stack and print it":
 * "+U+n+c---+e+r+t---+a-+i-+n+t+y---+ -+r+u--+l+e+s---"
 ************************************************/
package holding;
import net.mindview.util.Stack;

public class E15_Evaluator {
```

```
  private final static Stack<Character> stack =
    new Stack<Character>();
  private static void evaluate(String expr) {
    char data[] = expr.toCharArray();
    for(int i = 0; i < data.length;)
      switch(data[i++]) {
        case '+' : stack.push(data[i++]);
                   break;
        case '-' : System.out.print(stack.pop());
      }
  }
  public static void main(String[] args) {
    evaluate(
     "+U+n+c---+e+r+t---+a-+i-+n+t+y---+ -+r+u--+l+e+s---");
  }
} /* Output:
cnUtreaiytn ursel
*///:~
```

Of course, a real-world program also needs some error handling logic (e.g., to avert deadlock in case of improper input).

Exercise 16

```
//: holding/E16_Vowels.java
/****************** Exercise 16 *****************
 * Create a Set of the vowels. Working from
 * UniqueWords.java, count and display the number of
 * vowels in each input word, and also display the
 * total number of vowels in the input file.
 *********************************************/
package holding;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E16_Vowels {
  private final static Set<Character> vowels =
    new HashSet<Character>(Arrays.asList('a', 'e', 'o', 'u',
      'i', 'A', 'E', 'O', 'U', 'I'));
  public static void main(String[] args) {
    HashSet<String> processedWords = new HashSet<String>();
    int fileVowels = 0;
    int wordVowels;
    for(String word :
        new TextFile("E16_Vowels.java", "\\W+")) {
```

```
      wordVowels = 0;
      for(char letter : word.toCharArray())
        if(vowels.contains(letter))
          wordVowels++;
      if(!processedWords.contains(word)) {
        processedWords.add(word);
        print(word + " has " + wordVowels + " vowel(s)");
      }
      fileVowels += wordVowels;
    }
    print("Total number of vowels in file: " + fileVowels);
  }
} /* Output: (Sample)
holding has 2 vowel(s)
E16_Vowels has 3 vowel(s)
java has 2 vowel(s)
Exercise has 4 vowel(s)
16 has 0 vowel(s)
Create has 3 vowel(s)
...
contains has 3 vowel(s)
add has 1 vowel(s)
print has 1 vowel(s)
has has 1 vowel(s)
vowel has 2 vowel(s)
s has 0 vowel(s)
Total has 2 vowel(s)
Total number of vowels in file: 240
*///:~
```

Exercise 17

```
//: holding/E17_GerbilMap.java
/****************** Exercise 17 ******************
 * Move the Gerbil class from Exercise 1
 * into a Map, and associate each Gerbil (the value)
 * with it's name as a String (the key).
 * Use an Iterator for the keySet() to move
 * through the Map, look up the Gerbil for each key,
 * print out the key, and tell the Gerbil to hop().
 *********************************************/
package holding;
import java.util.*;
import java.util.Map.Entry;

public class E17_GerbilMap {
```

```
  public static void main(String args[]) {
    Map<String,Gerbil> map = new HashMap<String,Gerbil>();
    map.put("Fuzzy", new Gerbil(1));
    map.put("Spot", new Gerbil(2));
    map.put("Joe", new Gerbil(3));
    map.put("Ted", new Gerbil(4));
    map.put("Heather", new Gerbil(5));
    Iterator<Entry<String, Gerbil>> it =
      map.entrySet().iterator();
    while(it.hasNext()) {
      Entry<String, Gerbil> entry = it.next();
      System.out.print(entry.getKey() + ": ");
      entry.getValue().hop();
    }
  }
} /* Output:
Ted: gerbil 4 is hopping
Heather: gerbil 5 is hopping
Spot: gerbil 2 is hopping
Joe: gerbil 3 is hopping
Fuzzy: gerbil 1 is hopping
*///:~
```

The hashing function (described in *TIJ4)* does not store objects in entry order, so use a **LinkedHashMap** if you want to maintain that order.

Exercise 18

```
//: holding/E18_MapOrder.java
/******************* Exercise 18 ************************
 * Fill a HashMap with key-value pairs. Print the results
 * to show ordering by hash code. Extract the pairs, sort
 * by key, and place the result into a LinkedHashMap.
 * Show that insertion order is maintained.
 ******************************************************/
package holding;
import java.util.*;
import net.mindview.util.*;

public class E18_MapOrder {
  public static void main(String[] args) {
    Map<String,String> m1 =
      new HashMap<String,String>(Countries.capitals(25));
    System.out.println(m1);
    String[] keys = m1.keySet().toArray(new String[0]);
    Arrays.sort(keys);
```

```
    Map<String,String> m2 =
      new LinkedHashMap<String,String>();
    for(String key : keys)
      m2.put(key, m1.get(key));
    System.out.println(m2);
  }
} /* Output:
{KENYA=Nairobi, THE GAMBIA=Banjul, CHAD=N'djamena, CAPE
VERDE=Praia, COTE D'IVOIR (IVORY COAST)=Yamoussoukro,
GUINEA=Conakry, COMOROS=Moroni, CAMEROON=Yaounde,
ANGOLA=Luanda, EGYPT=Cairo, BURKINA FASO=Ouagadougou,
BENIN=Porto-Novo, BURUNDI=Bujumbura, ETHIOPIA=Addis Ababa,
BOTSWANA=Gaberone, DJIBOUTI=Dijibouti, CONGO=Brazzaville,
CENTRAL AFRICAN REPUBLIC=Bangui, ERITREA=Asmara,
ALGERIA=Algiers, EQUATORIAL GUINEA=Malabo, GHANA=Accra,
GABON=Libreville, BISSAU=Bissau, LESOTHO=Maseru}
{ALGERIA=Algiers, ANGOLA=Luanda, BENIN=Porto-Novo,
BISSAU=Bissau, BOTSWANA=Gaberone, BURKINA FASO=Ouagadougou,
BURUNDI=Bujumbura, CAMEROON=Yaounde, CAPE VERDE=Praia,
CENTRAL AFRICAN REPUBLIC=Bangui, CHAD=N'djamena,
COMOROS=Moroni, CONGO=Brazzaville, COTE D'IVOIR (IVORY
COAST)=Yamoussoukro, DJIBOUTI=Dijibouti, EGYPT=Cairo,
EQUATORIAL GUINEA=Malabo, ERITREA=Asmara, ETHIOPIA=Addis
Ababa, GABON=Libreville, GHANA=Accra, GUINEA=Conakry,
KENYA=Nairobi, LESOTHO=Maseru, THE GAMBIA=Banjul}
*///:~
```

Exercise 19

```
//: holding/E19_SetOrder.java
/****************** Exercise 19 *****************
 * Repeat Exercise 18 with a HashSet and
 * LinkedHashSet.
 ************************************************/
package holding;
import java.util.*;
import net.mindview.util.*;

public class E19_SetOrder {
  public static void main(String[] args) {
    Set<String> s1 =
      new HashSet<String>(Countries.names(25));
    System.out.println(s1);
    String[] elements = s1.toArray(new String[0]);
    Arrays.sort(elements);
    Set<String> s2 = new LinkedHashSet<String>();
```

```
    for(String element : elements)
      s2.add(element);
    System.out.println(s2);
  }
} /* Output:
[KENYA, THE GAMBIA, CHAD, CAPE VERDE, COTE D'IVOIR (IVORY
COAST), GUINEA, COMOROS, CAMEROON, ANGOLA, EGYPT, BURKINA
FASO, BENIN, BURUNDI, ETHIOPIA, BOTSWANA, DJIBOUTI, CONGO,
CENTRAL AFRICAN REPUBLIC, ERITREA, ALGERIA, EQUATORIAL
GUINEA, GHANA, GABON, BISSAU, LESOTHO]
[ALGERIA, ANGOLA, BENIN, BISSAU, BOTSWANA, BURKINA FASO,
BURUNDI, CAMEROON, CAPE VERDE, CENTRAL AFRICAN REPUBLIC,
CHAD, COMOROS, CONGO, COTE D'IVOIR (IVORY COAST), DJIBOUTI,
EGYPT, EQUATORIAL GUINEA, ERITREA, ETHIOPIA, GABON, GHANA,
GUINEA, KENYA, LESOTHO, THE GAMBIA]
*///:~
```

Exercise 20

```
//: holding/E20_VowelsInfo.java
/****************** Exercise 20 *****************
 * Modify Exercise 16 to count the occurrence of
 * each vowel.
 ************************************************/
package holding;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E20_VowelsInfo {
  private final static Set<Character> vowels =
    new HashSet<Character>(Arrays.asList('a', 'e', 'o', 'u',
      'i', 'A', 'E', 'O', 'U', 'I'));
  static void
  updateStat(Map<Character,Integer> stat, char letter) {
    Character ch = Character.toLowerCase(letter);
    Integer freq = stat.get(ch);
    stat.put(ch, freq == null ? 1 : freq + 1);
  }
  public static void main(String[] args) {
    HashMap<Character,Integer> fileStat =
      new HashMap<Character,Integer>();
    HashSet<String> processedWords = new HashSet<String>();
    HashMap<Character,Integer> wordStat =
      new HashMap<Character,Integer>();
    for(String word :
```

```
        new TextFile("E20_VowelsInfo.java", "\\W+")) {
      wordStat.clear();
      for(char letter : word.toCharArray())
        if(vowels.contains(letter)) {
          updateStat(wordStat, letter);
          updateStat(fileStat, letter);
        }
      if(!processedWords.contains(word)) {
        processedWords.add(word);
        print("Vowels in " + word + ": " + wordStat);
      }
    }
    print("*************************");
    print("Vowels in the whole file: " + fileStat);
  }
} /* Output: (Sample)
Vowels in holding: {o=1, i=1}
Vowels in E20_VowelsInfo: {o=2, i=1, e=2}
Vowels in java: {a=2}
Vowels in Exercise: {i=1, e=3}
Vowels in 20: {}
Vowels in Modify: {o=1, i=1}
Vowels in 16: {}
Vowels in so: {o=1}
...
Vowels in Vowels: {o=1, e=1}
Vowels in in: {i=1}
Vowels in whole: {o=1, e=1}
Vowels in file: {i=1, e=1}
*************************
Vowels in the whole file: {o=49, i=56, a=79, u=15, e=96}
*///:~
```

Try to accelerate the code, processing only the first occurrence of each word.

Exercise 21

```
//: holding/E21_WordsInfo.java
/****************** Exercise 21 *****************
 * Using a Map<String,Integer>, follow the form of
 * UniqueWords.java to create a program that counts
 * the occurrence of words in a file. Sort the
 * results using Collections.sort() with a second
 * argument of String.CASE_INSENSITIVE_ORDER (to
 * produce an alphabetic sort), and display the result.
 ***********************************************/
```

```
package holding;
import java.util.*;
import net.mindview.util.*;

public class E21_WordsInfo {
  public static void main(String[] args) {
    Map<String,Integer> wordsStat =
      new HashMap<String,Integer>();
    for(String word :
        new TextFile("E21_WordsInfo.java", "\\W+")) {
      Integer freq = wordsStat.get(word);
      wordsStat.put(word, freq == null ? 1 : freq + 1);
    }
    List<String> keys =
      new ArrayList<String>(wordsStat.keySet());
    Collections.sort(keys, String.CASE_INSENSITIVE_ORDER);
    for(String key : keys)
      System.out.println(key + " => " + wordsStat.get(key));
  }
} /* Output: (Sample)
0 => 1
1 => 2
21 => 1
a => 4
alphabetic => 1
an => 1
and => 1
args => 1
...
W => 1
with => 1
word => 3
words => 1
wordsStat => 5
*///:~
```

Exercise 22

```
//: holding/E22_WordsInfo2.java
/****************** Exercise 22 *****************
 * Modify the previous exercise so that it uses a
 * class containing a String and a count field to
 * store each different word, and a Set of these
 * objects to maintain the list of words.
 ************************************************/
package holding;
```

```
import java.util.*;
import net.mindview.util.*;

class WordCounter {
  public static final
  Comparator<WordCounter> CASE_INSENSITIVE_ORDER =
    new Comparator<WordCounter>() {
      public int compare(WordCounter o1, WordCounter o2) {
        return o1.word.compareToIgnoreCase(o2.word);
      }
    };
  private final String word;
  private int frequency;
  WordCounter(String word) {
    this.word = word;
    frequency = 1;
  }
  void incFrequency() { ++frequency; }
  String getWord() { return word; }
  int getFrequency() { return frequency; }
  public boolean equals(Object o) {
    return o instanceof WordCounter &&
           word.equals(((WordCounter)o).word);
  }
  public int hashCode() { return word.hashCode(); }
}

public class E22_WordsInfo2 {
  static void
  updateStat(Iterator<WordCounter> it, WordCounter wc) {
    while(it.hasNext()) {
      WordCounter currentWC = it.next();
      if(currentWC.equals(wc))
        currentWC.incFrequency();
    }
  }
  public static void main(String[] args) {
    Set<WordCounter> stat = new HashSet<WordCounter>();
    for(String word :
        new TextFile("E22_WordsInfo2.java", "\\W+")) {
      WordCounter wc = new WordCounter(word);
      if(stat.contains(wc))
        updateStat(stat.iterator(), wc);
      else
        stat.add(wc);
    }
    List<WordCounter> l = new ArrayList<WordCounter>(stat);
```

```
    Collections.sort(
      l, WordCounter.CASE_INSENSITIVE_ORDER);
    for(WordCounter wc : l)
      System.out.println(wc.getWord() + " => "
        + wc.getFrequency());
  }
} /* Output: (Sample)
0 => 1
1 => 1
22 => 1
a => 4
add => 1
and => 2
args => 1
...
util => 2
void => 3
W => 1
wc => 9
while => 1
word => 13
WordCounter => 19
words => 1
*///:~
```

The **WordCounter** class contains a **String** and a **frequency** field to store each different word. It uses **equals( )** and **hashCode( )** methods, respectively, to store class instances inside a **HashSet**.

We created a custom **Comparator** to use the **Collections.sort( )** method in the last exercise. Here it acts like **String.CASE_INSENSITIVE_ORDER**.

Without a **HashMap**, the situation gets more complicated, which shows how useful it is to keep your code base as small as possible (and boost productivity) by learning the classes offered by the JDK.

Additionally, try a **TreeSet** instead of a bare **HashSet** to maintain the list of words, and compare what (if any) improvement this makes to the program.

Exercise 23

```
//: holding/E23_MoreProbable.java
/****************** Exercise 23 *****************
 * Starting with Statistics.java, create a
 * program that runs the test repeatedly and
 * looks to see if any one number tends to appear
```

```
 * more than the others in the results.
 *********************************************/
package holding;
import java.util.*;
import java.util.Map.Entry;

class Counter {
  int i = 1;
  public String toString() { return Integer.toString(i); }
}

class HistoUnit implements Comparable<HistoUnit> {
  Counter counter;
  Integer val;
  public HistoUnit(Counter counter, Integer val) {
    this.counter = counter;
    this.val = val;
  }
  public int compareTo(HistoUnit o) {
    int lv = o.counter.i;
    int rv = counter.i;
    return (lv < rv ? -1 : (lv == rv ? 0 : 1));
  }
  public String toString() {
    return "Value = " + val
      + ", Occurrences = " + counter.i + "\n";
  }
}

public class E23_MoreProbable {
  private static Random rand = new Random(47);
  public static void main(String[] args) {
    Map<Integer, Counter> m =
      new HashMap<Integer, Counter>();
    for(int i = 0; i < 10000000; i++) {
      // Produce a number between 0 and 100:
      int r = rand.nextInt(100);
      if(m.containsKey(r))
        m.get(r).i++;
      else
        m.put(r, new Counter());
    }
    // Make a histogram:
    List<HistoUnit> lst = new ArrayList<HistoUnit>();
    Iterator<Entry<Integer, Counter>> it =
      m.entrySet().iterator();
    while(it.hasNext()) {
```

```
      Entry<Integer, Counter> entry = it.next();
      lst.add(new HistoUnit(
        entry.getValue(), entry.getKey()));
    }
    Collections.sort(lst);
    System.out.println("lst = " + lst);
  }
} /* Output: (Sample)
lst = [Value = 34, Occurrences = 100764
, Value = 92, Occurrences = 100660
, Value = 55, Occurrences = 100539
, Value = 81, Occurrences = 100503
...
, Value = 6, Occurrences = 99466
, Value = 16, Occurrences = 99427
, Value = 53, Occurrences = 99310
, Value = 86, Occurrences = 99095
]
*///:~
```

To optimize built-in functionality, we make a list of objects called **HistoUnit** and sort based on the value of **Counter** with **Collections.sort( )**. Printing the list shows the results in order of occurrence.

There is no preferred value. To determine how well the randomization function works, you could calculate the distribution and standard deviation.

Exercise 24

```
//: holding/E24_MapOrder2.java
/******************* Exercise 24 *************************
 * Fill a LinkedHashMap with String keys and objects.
 * Extract the pairs, sort them based on the keys, and
 * re-insert them into the Map.
 *****************************************************/
package holding;
import java.util.*;
import net.mindview.util.*;

public class E24_MapOrder2 {
  public static void main(String[] args) {
    Map<String,String> m1 =
      new LinkedHashMap<String,String>(
        Countries.capitals(25));
    System.out.println(m1);
    String[] keys = m1.keySet().toArray(new String[0]);
```

```
    Arrays.sort(keys);
    Map<String,String> m2 =
      new LinkedHashMap<String,String>();
    for(String key : keys)
      m2.put(key, m1.get(key));
    System.out.println(m2);
  }
} /* Output:
{ALGERIA=Algiers, ANGOLA=Luanda, BENIN=Porto-Novo,
BOTSWANA=Gaberone, BURKINA FASO=Ouagadougou,
BURUNDI=Bujumbura, CAMEROON=Yaounde, CAPE VERDE=Praia,
CENTRAL AFRICAN REPUBLIC=Bangui, CHAD=N'djamena,
COMOROS=Moroni, CONGO=Brazzaville, DJIBOUTI=Dijibouti,
EGYPT=Cairo, EQUATORIAL GUINEA=Malabo, ERITREA=Asmara,
ETHIOPIA=Addis Ababa, GABON=Libreville, THE GAMBIA=Banjul,
GHANA=Accra, GUINEA=Conakry, BISSAU=Bissau, COTE D'IVOIR
(IVORY COAST)=Yamoussoukro, KENYA=Nairobi, LESOTHO=Maseru}
{ALGERIA=Algiers, ANGOLA=Luanda, BENIN=Porto-Novo,
BISSAU=Bissau, BOTSWANA=Gaberone, BURKINA FASO=Ouagadougou,
BURUNDI=Bujumbura, CAMEROON=Yaounde, CAPE VERDE=Praia,
CENTRAL AFRICAN REPUBLIC=Bangui, CHAD=N'djamena,
COMOROS=Moroni, CONGO=Brazzaville, COTE D'IVOIR (IVORY
COAST)=Yamoussoukro, DJIBOUTI=Dijibouti, EGYPT=Cairo,
EQUATORIAL GUINEA=Malabo, ERITREA=Asmara, ETHIOPIA=Addis
Ababa, GABON=Libreville, GHANA=Accra, GUINEA=Conakry,
KENYA=Nairobi, LESOTHO=Maseru, THE GAMBIA=Banjul}
*///:~
```

Exercise 25

```
//: holding/E25_WordsInfo3.java
/****************** Exercise 25 *****************
 * Create a Map<String,ArrayList<Integer>>. Use
 * net.mindview.TextFile to open a text file and
 * read it in a word at a time (use "\\W+" as the
 * second argument to the TextFile constructor).
 * Count the words as you read them in, and for each
 * word in the file, record in the ArrayList<Integer>
 * the word count associated with that word - this is,
 * in effect, the location in the file where that
 * word was found.
 ***********************************************/
package holding;
import java.util.*;
import net.mindview.util.*;
```

```
public class E25_WordsInfo3 {
  public static void main(String[] args) {
    Map<String, ArrayList<Integer>> stat =
      new HashMap<String, ArrayList<Integer>>();
    int wordCount = 0;
    for(String word :
        new TextFile("E25_WordsInfo3.java", "\\W+")) {
      ArrayList<Integer> loc = stat.get(word);
      if(loc == null) {
        loc = new ArrayList<Integer>();
        stat.put(word, loc);
      }
      loc.add(++wordCount);
    }
    System.out.println(stat);
  }
} /* Output: (Sample)
{holding=[1], and=[21, 48], argument=[35], time=[29],
where=[76], loc=[120, 125, 127, 134, 135], file=[20, 54,
75], ArrayList=[10, 58, 99, 105, 118, 129], use=[30],
0=[109], Count=[40], associated=[63], out=[139], Map=[8,
97], that=[65, 77], println=[140], null=[126], main=[94],
open=[17], each=[50], as=[32, 43], constructor=[39], a=[7,
18, 25, 28], location=[72], in=[24, 47, 52, 56, 69, 73],
Use=[12], read=[22, 45], for=[49, 110], the=[33, 37, 41, 53,
57, 60, 71, 74], word=[26, 51, 61, 66, 78, 112, 123, 133],
class=[89], put=[132], new=[102, 113, 128], with=[64],
them=[46], 25=[5], second=[34], you=[44], Integer=[11, 59,
100, 106, 119, 130], text=[19], it=[23], util=[83, 87],
this=[67], static=[92], record=[55], get=[122], W=[31, 117],
import=[81, 84], net=[13, 85], public=[88, 91],
wordCount=[108, 137], stat=[101, 121, 131, 141], void=[93],
effect=[70], mindview=[14, 86], E25_WordsInfo3=[2, 90, 115],
String=[9, 95, 98, 104, 111], add=[136], Create=[6],
args=[96], words=[42], int=[107], System=[138], found=[80],
at=[27], was=[79], to=[16, 36], java=[3, 82, 116],
TextFile=[15, 38, 114], if=[124], HashMap=[103],
Exercise=[4], is=[68], count=[62]}
*///:~
```

Exercise 26

```
//: holding/E26_WordsInfo4.java
/****************** Exercise 26 *****************
 * Take the resulting Map from the previous
 * exercise and recreate the order of the words as
```

```
 * they appeared in the original file.
 *********************************************/
package holding;
import java.util.*;
import net.mindview.util.*;

public class E26_WordsInfo4 {
  public static void main(String[] args) {
    Map<String,ArrayList<Integer>> stat =
      new HashMap<String,ArrayList<Integer>>();
    int wordCount = 0;
    List<String> origWords =
      new TextFile("E26_WordsInfo4.java", "\\W+");
    for(String word : origWords) {
      ArrayList<Integer> loc = stat.get(word);
      if(loc == null) {
        loc = new ArrayList<Integer>();
        stat.put(word, loc);
      }
      loc.add(++wordCount);
    }
    // Now recreate the original order of the words.
    // We will use an inverted structure, where the key
    // is the position of the word in the file. Also,
    // we will sort words based on their positions.
    TreeMap<Integer,String> words =
      new TreeMap<Integer,String>();
    for(Map.Entry<String,ArrayList<Integer>> entry :
        stat.entrySet())
      for(Integer pos : entry.getValue())
        words.put(pos, entry.getKey());
    // Test the correctness.
    System.out.println(origWords);
    System.out.println(words.values());
  }
} /* Output: (Sample)
[holding, E26_WordsInfo4, java, Exercise, 26, Take, the,
resulting, Map, from, the, previous, exercise, and,
recreate, the, order, of, the, words, as, they, appeared,
in, the, original, file, import, java, util, import, net,
mindview, util, public, class, E26_WordsInfo4, public,
static, void, main, String, args, Map, String, ArrayList,
Integer, stat, new, HashMap, String, ArrayList, Integer,
int, wordCount, 0, List, String, origWords, new, TextFile,
E26_WordsInfo4, java, W, for, String, word, origWords,
ArrayList, Integer, loc, stat, get, word, if, loc, null,
loc, new, ArrayList, Integer, stat, put, word, loc, loc,
```

```
add, wordCount, Now, recreate, the, original, order, of,
the, words, We, will, use, an, inverted, structure, where,
the, key, is, the, position, of, the, word, in, the, file,
Also, we, will, sort, words, based, on, their, positions,
TreeMap, Integer, String, words, new, TreeMap, Integer,
String, for, Map, Entry, String, ArrayList, Integer, entry,
stat, entrySet, for, Integer, pos, entry, getValue, words,
put, pos, entry, getKey, Test, the, correctness, System,
out, println, origWords, System, out, println, words,
values]
[holding, E26_WordsInfo4, java, Exercise, 26, Take, the,
resulting, Map, from, the, previous, exercise, and,
recreate, the, order, of, the, words, as, they, appeared,
in, the, original, file, import, java, util, import, net,
mindview, util, public, class, E26_WordsInfo4, public,
static, void, main, String, args, Map, String, ArrayList,
Integer, stat, new, HashMap, String, ArrayList, Integer,
int, wordCount, 0, List, String, origWords, new, TextFile,
E26_WordsInfo4, java, W, for, String, word, origWords,
ArrayList, Integer, loc, stat, get, word, if, loc, null,
loc, new, ArrayList, Integer, stat, put, word, loc, loc,
add, wordCount, Now, recreate, the, original, order, of,
the, words, We, will, use, an, inverted, structure, where,
the, key, is, the, position, of, the, word, in, the, file,
Also, we, will, sort, words, based, on, their, positions,
TreeMap, Integer, String, words, new, TreeMap, Integer,
String, for, Map, Entry, String, ArrayList, Integer, entry,
stat, entrySet, for, Integer, pos, entry, getValue, words,
put, pos, entry, getKey, Test, the, correctness, System,
out, println, origWords, System, out, println, words,
values]
*///:~
```

Map.Entry holds one key-value pair of a map. You can retrieve all entries of a map by calling the map's **entrySet( )** method.

Exercise 27

```
//: holding/E27_CommandQueue.java
/****************** Exercise 27 *****************
 * Write a class called Command that contains a
 * String and has a method operation() that
 * displays the String. Write a second class with
 * a method that fills a Queue with Command objects
 * and returns it. Pass the filled Queue to a method
 * in a third class that consumes the objects in the
```

```
 * Queue and calls their operation() methods.
 *********************************************/
package holding;
import java.util.*;

class Command {
  private final String cmd;
  Command(String cmd) { this.cmd = cmd; }
  public void operation() { System.out.println(cmd); }
}

class Producer {
  public static void produce(Queue<Command> q) {
    q.offer(new Command("load"));
    q.offer(new Command("delete"));
    q.offer(new Command("save"));
    q.offer(new Command("exit"));
  }
}

class Consumer {
  public static void consume(Queue<Command> q) {
    while(q.peek() != null)
      q.remove().operation();
  }
}

public class E27_CommandQueue {
  public static void main(String[] args) {
    Queue<Command> cmds = new LinkedList<Command>();
    Producer.produce(cmds);
    Consumer.consume(cmds);
  }
} /* Output:
load
delete
save
exit
*///:~
```

This classic *producer-consumer* scenario uses the queue to transfer objects from one area of a program (*producer*) to another (*consumer*), with the **main( )** method acting as a *controller*.

Exercise 28

```
//: holding/E28_PriorityQueue.java
/****************** Exercise 28 *****************
 * Fill a PriorityQueue (using offer()) with
 * Double values created using java.util.Random,
 * then remove the elements using poll() and
 * display them.
 ************************************************/
package holding;
import java.util.*;

public class E28_PriorityQueue {
  static Random rand = new Random(47);
  public static void printQ(Queue<?> queue) {
    for(Object data = queue.poll(); data != null;
        data = queue.poll())
      System.out.print(data + " ");
    System.out.println();
  }
  public static void main(String[] args) {
    PriorityQueue<Double> priorityQueue =
      new PriorityQueue<Double>();
    for(int i = 0; i < 10; i++)
      priorityQueue.offer(rand.nextDouble());
    printQ(priorityQueue);
  }
} /* Output:
0.0508673570556899 0.16020656493302599 0.18847866977771732
0.2613610344283964 0.2678662084200585 0.5166020801268457
0.5309454508634242 0.7271157860730044 0.7620665811558285
0.8037155449603999
*///:~
```

Exercise 29

```
//: holding/E29_PriorityQueueSubtlety.java
// {ThrowsException}
/****************** Exercise 29 *****************
 * Create a simple class that inherits from Object
 * and contains no members, and show that you cannot
 * successfully add multiple elements of that class
 * to a PriorityQueue. This issue will be fully
 * explained in the Containers in Depth chapter.
 ************************************************/
```

```
package holding;
import java.util.*;

class Dummy {}

public class E29_PriorityQueueSubtlety {
  public static void main(String[] args) {
    PriorityQueue<Dummy> priorityQueue =
      new PriorityQueue<Dummy>();
    System.out.println("Adding 1st instance...");
    priorityQueue.offer(new Dummy());
    System.out.println("Adding 2nd instance...");
    priorityQueue.offer(new Dummy());
  }
} ///:~
```

The JDK documentation states: *"A priority queue relying on natural ordering also does not permit insertion of non-comparable objects (doing so may result in ClassCastException)."* The second insertion will trigger an exception.

Exercise 30

```
//: holding/E30_CollectionSequence2.java
/****************** Exercise 30 *****************
 * Modify CollectionSequence.java so that it does
 * not inherit from AbstractCollection, but instead
 * implements Collection.
 ************************************************/
package holding;
import java.util.*;
import typeinfo.pets.*;

class CollectionSequence2 extends PetSequence
implements Collection<Pet> {
  static void display(Iterator<Pet> it) {
    while(it.hasNext()) {
      Pet p = it.next();
      System.out.print(p.id() + ":" + p + " ");
    }
    System.out.println();
  }
  static void display(Collection<Pet> pets) {
    for(Pet p : pets)
      System.out.print(p.id() + ":" + p + " ");
    System.out.println();
  }
```

```
  public int size() { return pets.length; }
  public Iterator<Pet> iterator() {
    return new Iterator<Pet>() {
      private int index;
      public boolean hasNext() {
        return index < pets.length;
      }
      public Pet next() { return pets[index++]; }
      public void remove() { // Not implemented
        throw new UnsupportedOperationException();
      }
    };
  }
  // Other methods, which are also need to be provided.
  public boolean add(Pet o) {
    throw new UnsupportedOperationException();
  }
  public boolean addAll(Collection<? extends Pet> c) {
    throw new UnsupportedOperationException();
  }
  public void clear() {
    throw new UnsupportedOperationException();
  }
  public boolean contains(Object o) {
    if(o == null) return false;
    for(int i = 0; i < pets.length; i++)
      if(o.equals(pets[i])) return true;
    return false;
  }
  public boolean containsAll(Collection<?> c) {
    Iterator<?> it = c.iterator();
    while (it.hasNext())
      if(!contains(it.next())) return false;
    return true;
  }
  public boolean isEmpty() { return pets.length == 0; }
  public boolean remove(Object o) {
    throw new UnsupportedOperationException();
  }
  public boolean removeAll(Collection<?> c) {
    throw new UnsupportedOperationException();
  }
  public boolean retainAll(Collection<?> c) {
    throw new UnsupportedOperationException();
  }
  public Object[] toArray() {
    Object[] result = new Object[pets.length];
```

```
    System.arraycopy(pets, 0, result, 0, pets.length);
    return result;
  }
  @SuppressWarnings("unchecked")
  public <T> T[] toArray(T[] a) {
    if (a.length < pets.length)
      a = (T[])java.lang.reflect.Array.newInstance(
        a.getClass().getComponentType(), pets.length);
    T[] result = a;
    System.arraycopy(pets, 0, result, 0, pets.length);
    if (result.length > pets.length)
      result[pets.length] = null;
    return result;
  }
}

public class E30_CollectionSequence2 {
  public static void main(String[] args) {
    CollectionSequence2 c = new CollectionSequence2();
    CollectionSequence2.display(c);
    CollectionSequence2.display(c.iterator());
    System.out.println(
      Arrays.toString(c.toArray(new Pet[0])));
  }
} /* Output:
0:Rat 1:Manx 2:Cymric 3:Mutt 4:Pug 5:Cymric 6:Pug 7:Manx
0:Rat 1:Manx 2:Cymric 3:Mutt 4:Pug 5:Cymric 6:Pug 7:Manx
[Rat, Manx, Cymric, Mutt, Pug, Cymric, Pug, Manx]
*///:~
```

In the minimalist approach used here, optional operations (as marked in the interface description of **Collection**) are not supported, nor is **equals( )**. We also override **hashCode( )** and **toString( )**. Even with these restrictions, the code base is much bigger than that of **AbstractCollection** in the original version.

Note the use of **System.arraycopy( )** to copy the arrays quickly inside **toArray( )**.

Exercise 31

```
//: holding/E31_IterableRandomShapeGenerator.java
/****************** Exercise 31 *****************
 * Modify polymorphism/shape/RandomShapeGenerator.java
 * to make it Iterable. You'll need to add a
 * constructor that takes the number of elements
```

```
 * that you want the iterator to produce before
 * stopping. Verify that it works.
 *********************************************/
package holding;
import java.util.*;
import polymorphism.shape.*;

class RandomShapeGenerator implements Iterable<Shape> {
  private Random rand = new Random(47);
  private final int quantity;
  RandomShapeGenerator(int quantity) {
    this.quantity = quantity;
  }
  public Iterator<Shape> iterator() {
    return new Iterator<Shape>() {
      private int count;
      public boolean hasNext() {
        return count < quantity;
      }
      public Shape next() {
        ++count;
        return nextShape();
      }
      public void remove() { // Not implemented
        throw new UnsupportedOperationException();
      }
    };
  }
  private Shape nextShape() {
    switch(rand.nextInt(3)) {
      default:
      case 0: return new Circle();
      case 1: return new Square();
      case 2: return new Triangle();
    }
  }
}

public class E31_IterableRandomShapeGenerator {
  public static void main(String[] args) {
    RandomShapeGenerator rsg = new RandomShapeGenerator(10);
    for(Shape shape : rsg)
      System.out.println(shape.getClass().getSimpleName());
  }
} /* Output:
Triangle
Triangle
```

```
Square
Triangle
Square
Triangle
Square
Triangle
Circle
Square
*///:~
```

Exercise 32

```
//: holding/E32_MultiIterableNonCollectionSeq.java
/****************** Exercise 32 *****************
 * Following the example of MultiIterableClass,
 * add reversed() and randomized() methods to
 * NonCollectionSequence.java, as well as making
 * NonCollectionSequence implement Iterable, and
 * show that all the approaches work in foreach
 * statements.
 ************************************************/
package holding;
import java.util.*;
import typeinfo.pets.*;

class PetSequence {
  protected Pet[] pets = Pets.createArray(8);
}

class NonCollectionSequence extends PetSequence
implements Iterable<Pet> {
  public Iterable<Pet> reversed() {
    return new Iterable<Pet>() {
      public Iterator<Pet> iterator() {
        return new Iterator<Pet>() {
          int current = pets.length - 1;
          public boolean hasNext() { return current > -1; }
          public Pet next() { return pets[current--]; }
          public void remove() { // Not implemented
            throw new UnsupportedOperationException();
          }
        };
      }
    };
  }
  public Iterable<Pet> randomized() {
```

```
    return new Iterable<Pet>() {
      public Iterator<Pet> iterator() {
        List<Pet> shuffled =
          new ArrayList<Pet>(Arrays.asList(pets));
        Collections.shuffle(shuffled, new Random(47));
        return shuffled.iterator();
      }
    };
  }
  public Iterator<Pet> iterator() {
    return new Iterator<Pet>() {
      private int index = 0;
      public boolean hasNext() {
        return index < pets.length;
      }
      public Pet next() { return pets[index++]; }
      public void remove() { // Not implemented
        throw new UnsupportedOperationException();
      }
    };
  }
}

public class E32_MultiIterableNonCollectionSeq {
  public static void main(String[] args) {
    NonCollectionSequence nc = new NonCollectionSequence ();
    for(Pet pet : nc.reversed())
      System.out.print(pet + " ");
    System.out.println();
    for(Pet pet : nc.randomized())
      System.out.print(pet + " ");
    System.out.println();
    for(Pet pet : nc)
      System.out.print(pet + " ");
  }
} /* Output:
Manx Pug Cymric Pug Mutt Cymric Manx Rat
Pug Mutt Pug Rat Manx Manx Cymric Cymric
Rat Manx Cymric Mutt Pug Cymric Pug Manx
*///:~
```

Error Handling with Exceptions

Exercise 1

```
//: exceptions/E01_SimpleException.java
/****************** Exercise 1 ******************
 * Create a class with a main() that throws an
 * object of class Exception inside a try block.
 * Give the constructor for Exception a String
 * argument. Catch the exception inside a catch
 * clause and print the String argument. Add a
 * finally clause and print a message to prove
 * you were there.
 ************************************************/
package exceptions;

public class E01_SimpleException {
  public static void main(String args[]) {
    try {
      throw new Exception("An exception in main");
    } catch(Exception e) {
      System.out.println(
        "e.getMessage() = " + e.getMessage());
    } finally {
      System.out.println("In finally clause");
    }
  }
} /* Output:
e.getMessage() = An exception in main
In finally clause
*///:~
```

Exercise 2

```
//: exceptions/E02_NullReference.java
/****************** Exercise 2 ******************
 * Define an object reference and initialize it
 * to null. Try to call a method through this
```

```
 * reference. Now wrap the code in a try-catch
 * clause to catch the exception.
 *********************************************/
package exceptions;

public class E02_NullReference {
  public static void main(String args[]) {
    String s = null;
    // Causes a NullPointerException:
    //! s.toString();
    try {
      s.toString();
    } catch(Exception e) {
      System.out.println("Caught exception " + e);
    }
  }
} /* Output:
Caught exception java.lang.NullPointerException
*///:~
```

When you catch the exception, the program runs to completion.

Exercise 3

```
//: exceptions/E03_ArrayIndexBounds.java
/****************** Exercise 3 *****************
 * Write code to generate and catch an
 * ArrayIndexOutOfBoundsException.
 *********************************************/
package exceptions;

public class E03_ArrayIndexBounds {
  public static void main(String args[]) {
    char[] array = new char[10];
    try {
      array[10] = 'x';
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("e = " + e);
    }
  }
} /* Output:
e = java.lang.ArrayIndexOutOfBoundsException: 10
*///:~
```

Exercise 4

```
//: exceptions/E04_ExceptionClass.java
/****************** Exercise 4 ******************
 * Create your own exception class using the
 * extends keyword. Write a constructor for this
 * class that takes a String argument and stores
 * it inside the object with a String reference.
 * Write a method that prints out the stored
 * String. Create a try-catch clause to exercise
 * your new exception.
 ***********************************************/
package exceptions;

// Following the instructions to the letter:
class MyException extends Exception {
  String msg;
  public MyException(String msg) {
    this.msg = msg;
  }
  public void printMsg() {
    System.out.println("msg = " + msg);
  }
}

// Or take a more clever approach,
// noting that string storage and printing are
// built into Exception:
class MyException2 extends Exception {
  public MyException2(String s) { super(s); }
}

public class E04_ExceptionClass {
  public static void main(String args[]) {
    try {
      throw new MyException("MyException message");
    } catch(MyException e) {
      e.printMsg();
    }
    try {
      throw new MyException2("MyException2 message");
    } catch(MyException2 e) {
      System.out.println(
        "e.getMessage() = " + e.getMessage());
    }
  }
```

```
} /* Output:
msg = MyException message
e.getMessage() = MyException2 message
*///:~
```

Exercise 5

```
//: exceptions/E05_Resumption.java
/****************** Exercise 5 ******************
 * Create your own resumption-like behavior using
 * a while loop that repeats until an exception
 * is no longer thrown.
 ***********************************************/
package exceptions;

class ResumerException extends Exception {}

class Resumer {
  static int count = 3;
  static void f() throws ResumerException {
    if(--count > 0)
      throw new ResumerException();
  }
}

public class E05_Resumption {
  public static void main(String args[]) {
    while(true) {
      try {
        Resumer.f();
      } catch(ResumerException e) {
        System.out.println("Caught " + e);
        continue;
      }
      System.out.println("Got through...");
      break;
    }
    System.out.println("Successful execution");
  }
} /* Output:
Caught exceptions.ResumerException
Caught exceptions.ResumerException
Got through...
Successful execution
*///:~
```

Exercise 6

```
//: exceptions/E06_LoggingExceptions.java
/****************** Exercise 6 ******************
 * Create two exception classes, each of which
 * performs its own logging automatically.
 * Demonstrate that these work.
 ************************************************/
package exceptions;
import java.util.logging.*;
import java.io.*;

class LoggingException1 extends Exception {
  private static Logger logger =
    Logger.getLogger("LoggingException1");
  public LoggingException1() {
    StringWriter trace = new StringWriter();
    printStackTrace(new PrintWriter(trace));
    logger.severe(trace.toString());
  }
}

class LoggingException2 extends Exception {
  private static Logger logger =
    Logger.getLogger("LoggingException2");
  public LoggingException2() {
    StringWriter trace = new StringWriter();
    printStackTrace(new PrintWriter(trace));
    logger.severe(trace.toString());
  }
}

public class E06_LoggingExceptions {
  public static void main(String[] args) {
    try {
      throw new LoggingException1();
    } catch(LoggingException1 e) {
      System.err.println("Caught " + e);
    }
    try {
      throw new LoggingException2();
    } catch(LoggingException2 e) {
      System.err.println("Caught " + e);
    }
  }
} /* Output: (45% match)
```

```
Sep 12, 2007 3:10:36 PM exceptions.LoggingException1 <init>
SEVERE: exceptions.LoggingException1
        at
exceptions.E06_LoggingExceptions.main(E06_LoggingExceptions.
java:34)

Caught exceptions.LoggingException1
Sep 12, 2007 3:10:36 PM exceptions.LoggingException2 <init>
SEVERE: exceptions.LoggingException2
        at
exceptions.E06_LoggingExceptions.main(E06_LoggingExceptions.
java:39)

Caught exceptions.LoggingException2
*///:~
```

Each exception employs its own logger instance, **LoggingException1** and **LoggingException2**, respectively.

Exercise 7

```
//: exceptions/E07_LoggedArrayIndexBounds.java
/****************** Exercise 7 ******************
 * Modify Exercise 3 so that the catch clause logs
 * the results.
 ***********************************************/
package exceptions;
import java.util.logging.*;
import java.io.*;

public class E07_LoggedArrayIndexBounds {
  private static Logger logger =
    Logger.getLogger("E07_LoggedArrayIndexBounds");
  static void logException(Exception e) {
    StringWriter trace = new StringWriter();
    e.printStackTrace(new PrintWriter(trace));
    logger.severe(trace.toString());
  }
  public static void main(String args[]) {
    char[] array = new char[10];
    try {
      array[10] = 'x';
    } catch(ArrayIndexOutOfBoundsException e) {
      logException(e);
    }
  }
```

```
} /* Output: (80% match)
2005.09.16. 10:51:22 E07_LoggedArrayIndexBounds logException
SEVERE: java.lang.ArrayIndexOutOfBoundsException: 10
        at
E07_LoggedArrayIndexBounds.main(E07_LoggedArrayIndexBounds.j
ava:20)
*///:~
```

Exercise 8

```
//: exceptions/E08_ExceptionSpecification.java
/****************** Exercise 8 ******************
 * Write a class with a method that throws an
 * exception of the type created in Exercise 4.
 * Try compiling it without an exception
 * specification to see what the compiler says.
 * Add the appropriate exception specification.
 * Try out your class and its exception inside a
 * try-catch clause.
 ************************************************/
package exceptions;

class Thrower {
  public void f() {
    // Compiler gives an error: "unreported
    // exception MyException; must be caught
    // or declared to be thrown"
    //! throw new MyException("Inside f()");
  }
  public void g() throws MyException {
    throw new MyException("Inside g()");
  }
}

public class E08_ExceptionSpecification {
  public static void main(String args[]) {
    Thrower t = new Thrower();
    try {
      t.g();
    } catch(MyException e) {
      e.printMsg();
    }
  }
} /* Output:
msg = Inside g()
*///:~
```

Exercise 9

```
//: exceptions/E09_CatchAll.java
/****************** Exercise 9 ******************
 * Create three new types of exceptions. Write a
 * class with a method that throws all three. In
 * main(), call the method but only use a single
 * catch clause that will catch all three types
 * of exceptions.
 ***********************************************/
package exceptions;

class ExBase extends Exception {}
class Ex1 extends ExBase {}
class Ex2 extends ExBase {}
class Ex3 extends ExBase {}

class Thrower2 {
  void f() throws Ex1, Ex2, Ex3 {
    throw new Ex1();
    // You aren't forced to throw all the
    // exceptions in the specification.
  }
}

public class E09_CatchAll {
  public static void main(String args[]) {
    Thrower2 t = new Thrower2();
    try {
      t.f();
    } catch(ExBase e) {
      System.out.println("caught " + e);
    } catch(Exception e) {
      System.out.println("caught " + e);
    }
  }
} /* Output:
caught exceptions.Ex1
*///:~
```

We create a common base class for all three exceptions, then catch the common base exception. Alternatively, you can just catch **Exception**, from which all exceptions inherit.

Exercise 10

```
//: exceptions/E10_ChangeException.java
/****************** Exercise 10 ******************
 * Create a class with two methods, f() and g().
 * In g(), throw an exception of a new type that
 * you define. In f(), call g(), catch its
 * exception and, in the catch clause, throw a
 * different exception (of a second type that you
 * define). Test your code in main().
 *********************************************/
package exceptions;

class AnException extends Exception {}
class AnotherException extends Exception {}

public class E10_ChangeException {
  public void g() throws AnException {
    throw new AnException();
  }
  public void f() throws AnotherException {
    try {
      g();
    } catch(AnException e) {
      throw new AnotherException();
    }
  }
  public static void main(String args[]) {
    E10_ChangeException ce = new E10_ChangeException();
    try {
      ce.f();
    } catch(AnotherException e) {
      System.out.println("Caught " + e);
    }
  }
} /* Output:
Caught exceptions.AnotherException
*///:~
```

Once the **catch** clause grabs the exception, it's handled, and you can do anything with it, including throw another exception.

Exercise 11

```
//: exceptions/E11_ChangeToRuntimeException.java
```

```
// {ThrowsException}
/****************** Exercise 11 *****************
 * Repeat the previous exercise, but inside the
 * catch clause, wrap g()'s exception in a
 * RuntimeException.
 ************************************************/
package exceptions;

class AnException2 extends Exception {}

public class E11_ChangeToRuntimeException {
  public void g() throws AnException2 {
    throw new AnException2();
  }
  public void f() {
    try {
      g();
    } catch(AnException2 e) {
      throw new RuntimeException(e);
    }
  }
  public static void main(String args[]) {
    E11_ChangeToRuntimeException ce =
      new E11_ChangeToRuntimeException();
    ce.f();
  }
} ///:~
```

RuntimeException now envelopes **Exception**, so we do not need a try block.

Exercise 12

```
//: exceptions/E12_SequenceExceptions.java
// {ThrowsException}
/************************ Exercise 12 *******************
 * Modify innerclasses/Sequence.java so that it throws an
 * appropriate exception if you try to put in too many
 * elements.
 ********************************************************/
package exceptions;

class SequenceFullException extends RuntimeException {}

class Sequence2 {
  private Object[] objects;
  private int next;
```

```
  public Sequence2(int size) { objects = new Object[size]; }
  public void add(Object x) {
    if(next < objects.length)
      objects[next++] = x;
    else
      throw new SequenceFullException();
  }
  private class SequenceSelector implements Selector {
    private int i;
    public boolean end() { return i == objects.length; }
    public Object current() { return objects[i]; }
    public void next() { if(i < objects.length) i++; }
  }
  public Selector selector() {
    return new SequenceSelector();
  }
}

public class E12_SequenceExceptions {
  public static void main(String[] args) {
    Sequence2 sequence = new Sequence2(10);
    for(int i = 0; i < 11; i++)
      sequence.add(Integer.toString(i));
  }
} ///:~
```

SequenceFullException is more appropriate than **RuntimeException** because it indicates a programmer error.

Exercise 13

```
//: exceptions/E13_Finally.java
// {ThrowsException}
/****************** Exercise 13 *****************
 * Modify Exercise 9 by adding a finally clause.
 * Verify that your finally clause is executed, even
 * if a NullPointerException is thrown.
 ***********************************************/
package exceptions;

public class E13_Finally {
  public static void throwNull() {
    throw new NullPointerException();
  }
  public static void main(String args[]) {
    Thrower2 t = new Thrower2();
```

```
    try {
      t.f();
    } catch(ExBase e) {
      System.err.println("caught " + e);
    } finally {
      System.out.println("In finally clause A");
    }
    try {
      throwNull();
      t.f();
    } catch(ExBase e) {
      System.err.println("caught " + e);
    } finally {
      System.out.println("In finally clause B");
    }
  }
} ///:~
```

The output is:

```
caught Ex1
In finally clause A
In finally clause B
Exception in thread "main" java.lang.NullPointerException
        at E13_Finally.throwNull(E13_Finally.java:10)
        at E13_Finally.main(E13_Finally.java:22)
```

Although the code doesn't catch the **NullPointerException**, it still executes the **finally** clause.

Exercise 14

```
//: exceptions/E14_OnOffSwitch.java
/****************** Exercise 14 *****************
 * Show that OnOffSwitch.java can fail by
 * throwing a RuntimeException inside the try
 * block.
 ***********************************************/
package exceptions;

public class E14_OnOffSwitch {
  static Switch sw = new Switch();
  static void f() throws OnOffException1, OnOffException2 {
    throw new RuntimeException("Inside try");
  }
  public static void main(String[] args) {
```

```
    try {
      try {
        sw.on();
        // Code that can throw exceptions...
        f();
        sw.off();
      } catch(OnOffException1 e) {
        System.out.println("OnOffException1");
        sw.off();
      } catch(OnOffException2 e) {
        System.out.println("OnOffException2");
        sw.off();
      }
    } catch(RuntimeException e) {
      System.out.println(sw);
      System.out.println("Oops! the exception '"
        + e + "' slipped through without "
        + "turning the switch off!");
    }
  }
} /* Output:
on
on
Oops! the exception 'java.lang.RuntimeException: Inside try'
slipped through without turning the switch off!
*///:~
```

Exercise 15

```
//: exceptions/E15_WithFinally.java
/****************** Exercise 15 *****************
 * Show that WithFinally.java doesn't fail by
 * throwing a RuntimeException inside the try
 * block.
 ************************************************/
package exceptions;

public class E15_WithFinally {
  static Switch sw = new Switch();
  static void f() throws OnOffException1, OnOffException2 {
    throw new RuntimeException("Inside try");
  }
  public static void main(String[] args) {
    try {
      try {
        sw.on();
```

```
        // Code that can throw exceptions...
        f();
      } catch(OnOffException1 e) {
        System.out.println("OnOffException1");
      } catch(OnOffException2 e) {
        System.out.println("OnOffException2");
      } finally {
        sw.off();
      }
    } catch(RuntimeException e) {
      System.out.println("Exception '" + e +
        "'. Did the switch get turned off?");
      System.out.println(sw);
    }
  }
} /* Output:
on
off
Exception 'java.lang.RuntimeException: Inside try'. Did the
switch get turned off?
off
*///:~
```

The **finally** clause guarantees the necessary cleanup, even with an unexpected exception.

Exercise 16

```
//: exceptions/E16_CADSystem.java
/****************** Exercise 16 *****************
 * Modify reusing/CADSystem.java to demonstrate
 * that returning from the middle of a try-finally
 * will still perform proper cleanup.
 ************************************************/
package exceptions;

public class E16_CADSystem {
  public static void main(String[] args) {
    reusing.CADSystem x = new reusing.CADSystem(47);
    try {
      return;
    } finally {
      x.dispose();
    }
  }
} /* Output:
```

```
Shape constructor
Shape constructor
Drawing Line: 0, 0
Shape constructor
Drawing Line: 1, 1
Shape constructor
Drawing Line: 2, 4
Shape constructor
Drawing Circle
Shape constructor
Drawing Triangle
Combined constructor
CADSystem.dispose()
Erasing Triangle
Shape dispose
Erasing Circle
Shape dispose
Erasing Line: 2, 4
Shape dispose
Erasing Line: 1, 1
Shape dispose
Erasing Line: 0, 0
Shape dispose
Shape dispose
*///:~
```

Exercise 17

```
//: exceptions/E17_Frog.java
/****************** Exercise 17 *****************
 * Modify polymorphism/Frog.java so that it uses
 * try-finally to guarantee proper cleanup, and
 * show that this works even if you return from the
 * middle of the try-finally.
 ***********************************************/
package exceptions;

// Frog.dispose() is protected, cannot be called directly
class Frog2 extends polymorphism.Frog {
  protected void dispose() { super.dispose(); }
}

public class E17_Frog {
  public static void main(String[] args) {
    Frog2 frog = new Frog2();
    try {
```

```
      // No return in the middle...
    } finally {
      frog.dispose();
    }
    frog = new Frog2();
    try {
      // With return in the middle...
      return;
    } finally {
      frog.dispose();
    }
  }
} /* Output:
Creating Characteristic is alive
Creating Description Basic Living Creature
LivingCreature()
Creating Characteristic has heart
Creating Description Animal not Vegetable
Animal()
Creating Characteristic can live in water
Creating Description Both water and land
Amphibian()
Creating Characteristic Croaks
Creating Description Eats Bugs
Frog()
Frog dispose
disposing Description Eats Bugs
disposing Characteristic Croaks
Amphibian dispose
disposing Description Both water and land
disposing Characteristic can live in water
Animal dispose
disposing Description Animal not Vegetable
disposing Characteristic has heart
LivingCreature dispose
disposing Description Basic Living Creature
disposing Characteristic is alive
Creating Characteristic is alive
Creating Description Basic Living Creature
LivingCreature()
Creating Characteristic has heart
Creating Description Animal not Vegetable
Animal()
Creating Characteristic can live in water
Creating Description Both water and land
Amphibian()
Creating Characteristic Croaks
```

```
Creating Description Eats Bugs
Frog()
Frog dispose
disposing Description Eats Bugs
disposing Characteristic Croaks
Amphibian dispose
disposing Description Both water and land
disposing Characteristic can live in water
Animal dispose
disposing Description Animal not Vegetable
disposing Characteristic has heart
LivingCreature dispose
disposing Description Basic Living Creature
disposing Characteristic is alive
*///:~
```

As you see, both cases elicit the proper cleanup.

Exercise 18

```
//: exceptions/E18_LostMessage.java
/****************** Exercise 18 *****************
 * Add a second level of exception loss to
 * LostMessage.java so that the HoHumException is
 * itself replaced by a third exception.
 ************************************************/
package exceptions;

class YetAnotherException extends Exception {
  public String toString() {
    return "Yet another exception";
  }
}

class LostMessage2 {
  void f() throws VeryImportantException {
    throw new VeryImportantException();
  }
  void dispose() throws HoHumException {
    throw new HoHumException();
  }
  void cleanup() throws YetAnotherException {
    throw new YetAnotherException();
  }
}
```

```
public class E18_LostMessage {
  public static void main(String[] args) {
    try {
      LostMessage2 lm = new LostMessage2();
      try {
        try {
          lm.f();
        } finally {
          lm.dispose();
        }
      } finally {
        lm.cleanup();
      }
    } catch(Exception e) {
      System.out.println(e);
    }
  }
} /* Output:
Yet another exception
*///:~
```

Again, the outer **try** block enables the program to run to completion. Note, however, that **YetAnotherException** appears in the output, not **VeryImportantException** or **HoHumException**.

Exercise 19

```
//: exceptions/E19_GuardedLostMessage.java
/****************** Exercise 19 *****************
 * Repair the problem in LostMessage.java by
 * guarding the call in the finally clause.
 ***********************************************/
package exceptions;

public class E19_GuardedLostMessage {
  void f() throws VeryImportantException {
    throw new VeryImportantException();
  }
  void dispose() throws HoHumException {
    throw new HoHumException();
  }
  public static void main(String[] args) {
    try {
      LostMessage lm = new LostMessage();
      try {
        lm.f();
```

```
      } finally {
        try {
          lm.dispose();
        } catch(HoHumException e) {
          System.out.println(e);
        }
      }
    } catch(Exception e) {
      System.out.println(e);
    }
  }
} /* Output:
A trivial exception
A very important exception!
*///:~
```

This time, the outer **try** block properly reports **VeryImportantException**.

Exercise 20

```
//: exceptions/E20_UmpireArgument.java
/****************** Exercise 20 *****************
 * Modify StormyInning.java by adding an
 * UmpireArgument exception type and methods
 * that throw this exception. Test the modified
 * hierarchy.
 ************************************************/
package exceptions;

class BaseballException extends Exception {}
class Foul extends BaseballException {}
class Strike extends BaseballException {}
class UmpireArgument extends BaseballException {}

abstract class Inning {
  Inning() throws BaseballException {}
  void event () throws BaseballException {}
  abstract void atBat()
    throws Strike, Foul, UmpireArgument;
  abstract void decision() throws UmpireArgument;
  void walk() {} // Throws nothing
}

class StormException extends Exception {}
class RainedOut extends StormException {}
class PopFoul extends Foul {}
```

```
interface Storm {
  public void event() throws RainedOut;
  public void rainHard() throws RainedOut;
}

class StormyInning extends Inning
  implements Storm {
  StormyInning() throws RainedOut,
    BaseballException {}
  StormyInning(String s) throws Foul,
    BaseballException {}
  public void rainHard() throws RainedOut {}
  public void event() {}
  void atBat() throws PopFoul, UmpireArgument {
    throw new UmpireArgument();
  }
  void decision() throws UmpireArgument {
    throw new UmpireArgument();
  }
}

public class E20_UmpireArgument {
  public static void main(String[] args) {
    // Same code as before, still catches
    // the new exception:
    try {
      StormyInning si = new StormyInning();
      si.atBat();
    } catch(PopFoul e) {
      System.out.println("Pop foul");
    } catch(RainedOut e) {
      System.out.println("Rained out");
    } catch(BaseballException e) {
      System.out.println("Generic error");
    }
    // Strike not thrown in derived version.
    try {
      Inning i = new StormyInning();
      i.atBat();
    } catch(Strike e) {
      System.out.println("Strike");
    } catch(Foul e) {
      System.out.println("Foul");
    } catch(RainedOut e) {
      System.out.println("Rained out");
    } catch(BaseballException e) {
```

```
      System.out.println("Generic baseball exception");
    }
    // Or you can add code to catch the
    // specific type of exception:
    try {
      StormyInning si = new StormyInning();
      si.atBat();
      si.decision();
    } catch(PopFoul e) {
      System.out.println("Pop foul");
    } catch(RainedOut e) {
      System.out.println("Rained out");
    } catch(UmpireArgument e) {
      System.out.println(
        "Argument with the umpire");
    } catch(BaseballException e) {
      System.out.println("Generic error");
    }
  }
} /* Output:
Generic error
Generic baseball exception
Argument with the umpire
*///:~
```

The exception hierarchy in the first two lines allows us to add new exceptions without forcing changes to existing code.

Exercise 21

```
//: exceptions/E21_ConstructorExceptions.java
/****************** Exercise 21 *****************
 * Demonstrate that a derived-class constructor
 * cannot catch exceptions thrown by its
 * base-class constructor.
 ************************************************/
package exceptions;

class Except1 extends Exception {
  public Except1(String s) { super(s); }
}

class BaseWithException {
  public BaseWithException() throws Except1 {
    throw new Except1("thrown by BaseWithException");
  }
```

```
}

class DerivedWE extends BaseWithException {
  // Produces compile-time error:
  //    unreported exception Except1
  // ! public DerivedWE() {}
  // Gives compile error: call to super must be
  // first statement in constructor:
  //! public DerivedWE() {
  //!   try {
  //!     super();
  //!   } catch(Except1 ex1) {
  //!   }
  //! }
  public DerivedWE() throws Except1 {}
}

public class E21_ConstructorExceptions {
  public static void main(String args[]) {
    try {
      new DerivedWE();
    } catch(Except1 ex1) {
      System.out.println("Caught " + ex1);
    }
  }
} /* Output:
Caught exceptions.Except1: thrown by BaseWithException
*///:~
```

Remember, when you throw an exception, the compiler *creates **no** object*. This is why a derived-class constructor cannot catch a base-class constructor exception: it can't "recover" from the exception failure, since there's no base-class sub-object.

Exercise 22

```
//: exceptions/E22_FailingConstructor.java
/****************** Exercise 22 *****************
 * Create a class called FailingConstructor with a
 * constructor that might fail partway through the
 * construction process and throw an exception. In
 * main(), write code that properly guards against
 * this failure.
 ***********************************************/
package exceptions;
```

```
class ConstructionException extends Exception {}

class FailingConstructor {
  FailingConstructor(boolean fail)
    throws ConstructionException {
    if(fail) throw new ConstructionException();
  }
}

public class E22_FailingConstructor {
  public static void main(String args[]) {
    for(boolean b = false; ; b = !b)
      try {
        System.out.println("Constructing...");
        FailingConstructor fc = new FailingConstructor(b);
        try {
          System.out.println("Processing...");
        } finally {
          System.out.println("Cleanup...");
        }
      } catch(ConstructionException e) {
        System.out.println("Construction has failed: " + e);
        break;
      }
  }
} /* Output:
Constructing...
Processing...
Cleanup...
Constructing...
Construction has failed: exceptions.ConstructionException
*///:~
```

This program shows both the success and failure of the construction process. The code follows the idiom presented in *TIJ4*.

Exercise 23

```
//: exceptions/E23_FailingConstructor2.java
/****************** Exercise 23 *****************
 * Add a class with a dispose() method to the
 * previous exercise. Modify FailingConstructor so
 * that the constructor creates one of these
 * disposable objects as a member object, after which
 * the constructor might throw an exception, after
 * which it creates a second disposable member object.
```

```
 * Write code to properly guard against failure, and
 * in main() verify that all possible failure
 * situations are covered.
 *********************************************/
package exceptions;

class WithDispose {
  private final int id;
  WithDispose(int id) { this.id = id; }
  void dispose() {
    System.out.println("WithDispose.dispose(): " + id);
  }
}

class FailingConstructor2 {
  private final WithDispose wd1, wd2;
  FailingConstructor2(boolean fail)
    throws ConstructionException {
    wd1 = new WithDispose(1);
    try {
      // "Simulate" other code that might throw exceptions
      if(fail) throw new ConstructionException();
    } catch(ConstructionException e) {
      wd1.dispose();
      throw e;
    }
    wd2 = new WithDispose(2);
  }
}

public class E23_FailingConstructor2 {
  public static void main(String args[]) {
    for(boolean b = false; ; b = !b)
      try {
        System.out.println("Constructing...");
        FailingConstructor2 fc = new FailingConstructor2(b);
        try {
          System.out.println("Processing...");
        } finally {
          // We do not have a decent method to call for
          // cleanup!
          System.out.println("Cleanup...");
        }
      } catch(ConstructionException e) {
        System.out.println("Construction has failed: " + e);
        break;
      }
```

```
  }
} /* Output:
Constructing...
Processing...
Cleanup...
Constructing...
WithDispose.dispose(): 1
Construction has failed: exceptions.ConstructionException
*///:~
```

Exercise 24

```
//: exceptions/E24_FailingConstructor3.java
/****************** Exercise 24 *****************
 * Add a dispose() method to the FailingConstructor
 * class and write code to properly use this class.
 ************************************************/
package exceptions;

class FailingConstructor3 {
  private final WithDispose wd1, wd2;
  FailingConstructor3(boolean fail)
    throws ConstructionException {
    wd1 = new WithDispose(1);
    try {
      // "Simulate" other code that might throw exceptions
      if(fail) throw new ConstructionException();
    } catch(ConstructionException e) {
      wd1.dispose();
      throw e;
    }
    wd2 = new WithDispose(2);
  }
  void dispose() {
    wd2.dispose();
    wd1.dispose();
  }
}

public class E24_FailingConstructor3 {
  public static void main(String args[]) {
    for(boolean b = false; ; b = !b)
      try {
        System.out.println("Constructing...");
        FailingConstructor3 fc = new FailingConstructor3(b);
        try {
```

```
          System.out.println("Processing...");
        } finally {
          System.out.println("Cleanup...");
          fc.dispose();
        }
      } catch(ConstructionException e) {
        System.out.println("Construction has failed: " + e);
        break;
      }
  }
} /* Output:
Constructing...
Processing...
Cleanup...
WithDispose.dispose(): 2
WithDispose.dispose(): 1
Constructing...
WithDispose.dispose(): 1
Construction has failed: exceptions.ConstructionException
*///:~
```

Exercise 25

```
//: exceptions/E25_ThreeLevelExceptions.java
/****************** Exercise 25 ******************
 * Create a three-level hierarchy of exceptions.
 * Now create a base class A with a method that
 * throws an exception at the base of your
 * hierarchy. Inherit B from A and override the
 * method so it throws an exception at level two
 * of your hierarchy. Repeat by inheriting class
 * C from B. In main(), create a C and upcast it
 * to A, then call the method.
 *********************************************/
package exceptions;

class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}

class A {
  public void f() throws Level1Exception {
    throw new Level1Exception();
  }
}
```

```
class B extends A {
  public void f() throws Level2Exception {
    throw new Level2Exception();
  }
}

class C extends B {
  public void f() throws Level3Exception {
    throw new Level3Exception();
  }
}

public class E25_ThreeLevelExceptions {
  public static void main(String args[]) {
    A a = new C();
    try {
      a.f();
    } catch(Level1Exception e) {
      System.out.println("Caught " + e);
    }
  }
} /* Output:
Caught exceptions.Level3Exception
*///:~
```

The compiler forces you to catch a **Level1Exception** because that's what **A.f( )** throws.

Exercise 26

```
//: exceptions/E26_MainException.java
// {ThrowsException}
/****************** Exercise 26 *****************
 * Change the file name string in MainException.java to
 * name a file that doesn't exist. Run the program and
 * note the result
 ************************************************/
package exceptions;
import java.io.*;

public class E26_MainException {
  // Pass all exceptions to the console:
  public static void main(String[] args) throws Exception {
    // Open the file:
    FileInputStream file =
      new FileInputStream("DoesnotExist.file");
```

```
    // Use the file ...
    // Close the file:
    file.close();
  }
} ///:~
```

The output is:

```
Exception in thread "main" java.io.FileNotFoundException:
DoesnotExist.file (The system cannot find the file
specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.io.FileInputStream.<init>(Unknown Source)
        at E27_MainException.main(E27_MainException.java:14)
```

Exercise 27

```
//: exceptions/E27_RuntimeArrayIndexBounds.java
// {ThrowsException}
/****************** Exercise 27 *****************
 * Modify Exercise 3 to convert the exception to a
 * RuntimeException.
 ***********************************************/
package exceptions;

public class E27_RuntimeArrayIndexBounds {
  public static void main(String args[]) {
    char[] array = new char[10];
    try {
      array[10] = 'x';
    } catch(ArrayIndexOutOfBoundsException e) {
      throw new RuntimeException(e);
    }
  }
} ///:~
```

Exercise 28

```
//: exceptions/E28_RuntimeExceptionClass.java
// {ThrowsException}
/****************** Exercise 28 *****************
 * Modify Exercise 4 so that the custom exception
 * class inherits from RuntimeException, and show
```

```
 * that the compiler allows you to leave out the
 * try block.
 *********************************************/
package exceptions;

class MyRuntimeException extends RuntimeException {
  public MyRuntimeException(String s) { super(s); }
}

public class E28_RuntimeExceptionClass {
  public static void main(String args[]) {
    throw new MyRuntimeException("MyRuntimeException msg");
  }
} ///:~
```

MyRuntimeException is now inherited from **RuntimeException**, so we do not need a try block.

Exercise 29

```
//: exceptions/E29_StormyInning2.java
// {ThrowsException}
/****************** Exercise 29 *****************
 * Modify all the exception types in StormyInning.java
 * so that they extend RuntimeException, and show
 * that no exception specifications or try blocks are
 * necessary. Remove the '//!' comments and show how
 * the methods can be compiled without specifications.
 *********************************************/
package exceptions;

class BaseballException2 extends RuntimeException {}
class Foul2 extends BaseballException2 {}
class Strike2 extends BaseballException2 {}

abstract class Inning2 {
  Inning2() {}
  public void event() {}
  abstract void atBat();
  void walk() {}
}

class StormException2 extends RuntimeException {}
class RainedOut2 extends StormException2 {}
class PopFoul2 extends Foul2 {}
```

```
interface Storm2 {
  void event();
  void rainHard();
}

class StormyInning2 extends Inning2 implements Storm2 {
  StormyInning2() {}
  StormyInning2(String s) {}
  public void rainHard() {}
  void atBat() { throw new PopFoul2(); }
}

public class E29_StormyInning2 {
  public static void main(String[] args) {
    StormyInning2 si = new StormyInning2();
    si.atBat();
  }
} ///:~
```

Exercise 30

```
//: exceptions/E30_Human.java
/****************** Exercise 30 ******************
 * Modify main() in Human.java so that the
 * technique in TurnOffChecking.java is used to
 * handle the different types of exceptions.
 ***********************************************/
package exceptions;

public class E30_Human {
  static void throwRuntimeException(int type) {
    try {
      switch(type) {
        case 0: throw new Annoyance();
        case 1: throw new Sneeze();
        default: return;
      }
    } catch(Exception e) { // Adapt to unchecked:
      throw new RuntimeException(e);
    }
  }
  public static void main(String[] args) {
    // Let RuntimeExceptions go out of the method:
    throwRuntimeException(2);
    // Or let catch exceptions:
    for(int i = 0; i < 2; i++)
```

```
      try {
        throwRuntimeException(i);
      } catch(RuntimeException re) {
        try {
          throw re.getCause();
        } catch(Sneeze e) {
          System.out.println("Caught Sneeze");
        } catch(Annoyance e) {
          System.out.println("Caught Annoyance");
        } catch(Throwable t) {
          System.out.println(t);
        }
      }
  }
} /* Output:
Caught Annoyance
Caught Sneeze
*///:~
```

Strings

Exercise 1

Invoke the **javap** tool (with **javap –c SprinklerSystem**) to generate this simplified output for the **toString( )** method of the **SprinklerSystem** class:

```
0:   new             #5;  //class StringBuilder
3:   dup
4:   invokespecial   #6;  //StringBuilder."<init>":()
7:   ldc     #7;
9:   invokevirtual   #8;  //StringBuilder.append:()
12:  aload_0
13:  getfield        #9;
16:  invokevirtual   #8;  //StringBuilder.append:()
19:  ldc     #10;
21:  invokevirtual   #8;  //StringBuilder.append:()
24:  ldc     #11;
26:  invokevirtual   #8;  //StringBuilder.append:()
29:  aload_0
30:  getfield        #12;
33:  invokevirtual   #8;  //StringBuilder.append:()
36:  ldc     #10;
38:  invokevirtual   #8;  //StringBuilder.append:()
41:  ldc     #13;
43:  invokevirtual   #8;  //StringBuilder.append:()
46:  aload_0
47:  getfield        #14;
50:  invokevirtual   #8;  //StringBuilder.append:()
53:  ldc     #10;
55:  invokevirtual   #8;  //StringBuilder.append:()
58:  ldc     #15;
60:  invokevirtual   #8;  //StringBuilder.append:()
63:  aload_0
64:  getfield        #16;
67:  invokevirtual   #8;  //StringBuilder.append:()
70:  ldc     #17;
72:  invokevirtual   #8;  //StringBuilder.append:()
75:  ldc     #18;
77:  invokevirtual   #8;  //StringBuilder.append:()
80:  aload_0
81:  getfield        #19;
```

```
   84:  invokevirtual   #20; //StringBuilder.append:()
   87:  ldc     #10;
   89:  invokevirtual   #8;  //StringBuilder.append:()
   92:  ldc     #21;
   94:  invokevirtual   #8;  //StringBuilder.append:()
   97:  aload_0
   98:  getfield        #22;
   101: invokevirtual   #23; //StringBuilder.append:()
   104: ldc     #10;
   106: invokevirtual   #8;  //StringBuilder.append:()
   109: ldc     #24;
   111: invokevirtual   #8;  //StringBuilder.append:()
   114: aload_0
   115: getfield        #4;
   118: invokevirtual   #25; //StringBuilder.append:()
   121: invokevirtual   #26; //StringBuilder.toString:()
   124: areturn
```

The operations are simple (e.g., no looping is involved), so the compiler generates reasonably optimized code. Note that you create only one instance of **StringBuilder**.

Though different indexes refer to **StringBuilder.append( )**, each overloaded version has a unique index.

Exercise 2

```
//: strings/E02_RepairInfinite.java
/****************** Exercise 2 *****************
 * Repair InfiniteRecursion.java.
 ***********************************************/
package strings;
import java.util.*;

public class E02_RepairInfinite {
  public String toString() {
    return " E02_RepairInfinite address: "
      + super.toString() + "\n";
  }
  public static void main(String[] args) {
    List<E02_RepairInfinite> v =
      new ArrayList<E02_RepairInfinite>();
    for(int i = 0; i < 10; i++)
      v.add(new E02_RepairInfinite());
    System.out.println(v);
  }
```

```
} /* Output: (70% match)
[ E02_RepairInfinite address: E02_RepairInfinite@3e25a5
,  E02_RepairInfinite address: E02_RepairInfinite@19821f
,  E02_RepairInfinite address: E02_RepairInfinite@addbf1
,  E02_RepairInfinite address: E02_RepairInfinite@42e816
,  E02_RepairInfinite address: E02_RepairInfinite@9304b1
,  E02_RepairInfinite address: E02_RepairInfinite@190d11
,  E02_RepairInfinite address: E02_RepairInfinite@a90653
,  E02_RepairInfinite address: E02_RepairInfinite@de6ced
,  E02_RepairInfinite address: E02_RepairInfinite@c17164
,  E02_RepairInfinite address: E02_RepairInfinite@1fb8ee3
]
*///:~
```

Exercise 3

```
//: strings/E03_TurtleRedirected.java
/****************** Exercise 3 ******************
 * Modify Turtle.java so that it sends all output
 * to System.err.
 ************************************************/
package strings;
import java.util.*;

class Turtle {
  private final String name;
  private final Formatter f;
  public Turtle(String name, Formatter f) {
    this.name = name;
    this.f = f;
  }
  public void move(int x, int y) {
    f.format("%s The Turtle is at (%d,%d)\n", name, x, y);
  }
}

public class E03_TurtleRedirected {
  public static void main(String[] args) {
    Formatter f = new Formatter(System.err);
    Turtle tommy = new Turtle("Tommy",f),
           terry = new Turtle("Terry",f);
    tommy.move(0,0);
    terry.move(4,8);
    tommy.move(3,4);
    terry.move(2,5);
    tommy.move(3,3);
```

```
    terry.move(3,3);
  }
} /* Output:
Tommy The Turtle is at (0,0)
Terry The Turtle is at (4,8)
Tommy The Turtle is at (3,4)
Terry The Turtle is at (2,5)
Tommy The Turtle is at (3,3)
Terry The Turtle is at (3,3)
*///:~
```

Exercise 4

```
//: strings/E04_CustomizableReceipt.java
/****************** Exercise 4 ******************
 * Modify Receipt.java so that the widths are all
 * controlled by a single set of constant values.
 * The goal is to allow you to easily change a
 * width by changing a single value in one place.
 ************************************************/
package strings;
import java.util.*;

class Receipt {
  public static final int ITEM_WIDTH = 15;
  public static final int QTY_WIDTH = 5;
  public static final int PRICE_WIDTH = 10;
  private static final String TITLE_FRMT =
    "%-" + ITEM_WIDTH + "s %" + QTY_WIDTH + "s %" +
    PRICE_WIDTH + "s\n";
  private static final String ITEM_FRMT =
    "%-" + ITEM_WIDTH + "." + ITEM_WIDTH + "s %" +
    QTY_WIDTH + "d %" + PRICE_WIDTH + ".2f\n";
  private static final String TOTAL_FRMT =
    "%-" + ITEM_WIDTH + "s %" + QTY_WIDTH + "s %" +
    PRICE_WIDTH + ".2f\n";
  private double total = 0;
  Formatter f = new Formatter(System.out, Locale.US);
  public void printTitle() {
    f.format(TITLE_FRMT, "Item", "Qty", "Price");
    f.format(TITLE_FRMT, "----", "---", "-----");
  }
  public void print(String name, int qty, double price) {
    f.format(ITEM_FRMT, name, qty, price);
    total += price;
  }
```

```
  public void printTotal() {
    f.format(TOTAL_FRMT, "Tax", "", total*0.06);
    f.format(TITLE_FRMT, "", "", "-----");
    f.format(TOTAL_FRMT, "Total", "",
      total * 1.06);
  }
}

public class E04_CustomizableReceipt {
  public static void main(String[] args) {
    Receipt receipt = new Receipt();
    receipt.printTitle();
    receipt.print("Jack's Magic Beans", 4, 4.25);
    receipt.print("Princess Peas", 3, 5.1);
    receipt.print("Three Bears Porridge", 1, 14.29);
    receipt.printTotal();
  }
} /* Output:
Item              Qty      Price
----              ---      -----
Jack's Magic Be     4       4.25
Princess Peas       3       5.10
Three Bears Por     1      14.29
Tax                         1.42
                           -----
Total                      25.06
*///:~
```

Exercise 5

```
//: strings/E05_ComplexConversion.java
/****************** Exercise 5 ******************
 * For each of the basic conversion types in the
 * above table, write the most complex formatting
 * expression possible. That is, use all the
 * possible format specifiers available for that
 * conversion type.
 ************************************************/
package strings;
import java.math.*;
import java.util.*;
import static net.mindview.util.Print.*;

public class E05_ComplexConversion {
  public static void main(String[] args) {
    Formatter f = new Formatter(System.out, Locale.US);
```

```
    char u = 'a';
    print("u = 'a'");
    f.format("s: %1$-#10s\n", u);
    f.format("c: %1$-10c\n", u);
    f.format("b: %1$-10.10b\n", u);
    f.format("h: %1$-10.10h\n", u);
    int v = 1000;
    print("v = 1000");
    f.format("d 1: %1$(,0+10d\n", v);
    f.format("d 2: %1$-(, 10d\n", v);
    f.format("d 3, v = -v: %1$-(, 10d\n", -v);
    f.format("c, v = 121: %1$-10c\n", 121);
    f.format("b: %1$-10.10b\n", v);
    f.format("s: %1$#-10.10s\n", v);
    f.format("x: %1$-#10x\n", v);
    f.format("h: %1$-10.10h\n", v);
    BigInteger w = new BigInteger("50000000000000");
    print("w = new BigInteger(\"50000000000000\")");
    f.format("d 1: %1$(,0+10d\n", w);
    f.format("d 2: %1$-(, 10d\n", w);
    f.format("d 3, w = -w: %1$-(, 10d\n", w.negate());
    f.format("b: %1$-10.10b\n", w);
    f.format("s: %1$#-10.10s\n", w);
    f.format("x 1: %1$(0+10x\n", w);
    f.format("x 2: %1$-( 10x\n", w);
    f.format("x 3, w = -w: %1$-( 10x\n", w.negate());
    f.format("h: %1$-10.10h\n", w);
    double x = 179.543;
    print("x = 179.543");
    f.format("b: %1$-10.10b\n", x);
    f.format("s: %1$#-10.10s\n", x);
    f.format("f 1: %1$#(,0+10.2f\n", x);
    f.format("f 2: %1$#(,- 10.2f\n", x);
    f.format("f 3, x = -x: %1$#(,0+10.2f\n", -x);
    f.format("e 1: %1$#(0+10.2e\n", x);
    f.format("e 2: %1$#(- 10.2e\n", x);
    f.format("e 3, x = -x: %1$#(0+10.2e\n", -x);
    f.format("h: %1$-10.10h\n", x);
    Object y = new Object();
    print("y = new Object()");
    f.format("b: %1$-10.10b\n", y);
    f.format("s: %1$#-10.10s\n", y);
    f.format("h: %1$-10.10h\n", y);
    boolean z = false;
    print("z = false");
    f.format("b: %1$-10.10b\n", z);
    f.format("s: %1$#-10.10s\n", z);
```

```
    f.format("h: %1$-10.10h\n", z);
    // A special no argument conversion type
    f.format("%%: %-10%");
  }
} /* Output: (95% match)
u = 'a'
s: a
c: a
b: true
h: 61
v = 1000
d 1: +00001,000
d 2:  1,000
d 3, v = -v: (1,000)
c, v = 121: y
b: true
s: 1000
x: 0x3e8
h: 3e8
w = new BigInteger("50000000000000")
d 1: +50,000,000,000,000
d 2:  50,000,000,000,000
d 3, w = -w: (50,000,000,000,000)
b: true
s: 5000000000
x 1: +2d79883d2000
x 2:  2d79883d2000
x 3, w = -w: (2d79883d2000)
h: 8842a1a7
x = 179.543
b: true
s: 179.543
f 1: +000179.54
f 2:  179.54
f 3, x = -x: (00179.54)
e 1: +01.80e+02
e 2:  1.80e+02
e 3, x = -x: (1.80e+02)
h: 1ef462c
y = new Object()
b: true
s: java.lang.
h: 3e25a5
z = false
b: false
s: false
h: 4d5
```

```
%: %
*///:~
```

The above program includes format specifiers not mentioned in *TIJ4* (the JDK documentation elaborates on all of them). The format specifiers applicable to a particular conversion type depends on the variable type, as well. We omitted incompatible cases where a conversion type is not appropriate for a particular variable type. Note that not all combinations of format specifiers are possible. For example, giving both the '-' and '0' flags throws an **IllegalFormatFlagsException**.

We omitted the ',' flag for the 'e' because it's inapplicable as a conversion type (though the J2SE5 API documentation mistakenly says otherwise).

Exercise 6

```
//: strings/E06_ClassDump.java
/****************** Exercise 6 ******************
 * Create a class that contains int, long, float
 * and double fields. Create a toString() method
 * for this class that uses String.format(), and
 * demonstrate that your class works correctly.
 ************************************************/
package strings;
import java.util.*;
import static java.lang.String.*;

class DataHolder {
  int intField = 1;
  long longField = 2L;
  float floatField = 3.0f;
  double doubleField = 4.0;
  public String toString() {
    StringBuilder result =
      new StringBuilder("DataHolder: \n");
    result.append(
      format(Locale.US, "intField: %d\n", intField));
    result.append(
      format(Locale.US, "longField: %d\n", longField));
    result.append(
      format(Locale.US, "floatField: %f\n", floatField));
    result.append(
      format(Locale.US, "doubleField: %e\n", doubleField));
    return result.toString();
  }
}
```

```
public class E06_ClassDump {
  public static void main(String[] args) {
    System.out.println(new DataHolder());
  }
} /* Output:
DataHolder:
intField: 1
longField: 2
floatField: 3.000000
doubleField: 4.000000e+00
*///:~
```

Exercise 7

```
//: strings/E07_SentenceChecker.java
/****************** Exercise 7 *****************
 * Using the documentation for java.util.regex.Pattern
 * as a resource, write and test a regular expression
 * that checks a sentence to see that it begins with a
 * capital letter and ends with a period.
 ***********************************************/
package strings;

public class E07_SentenceChecker {
  public static boolean matches(String text) {
    return text.matches("\\p{javaUpperCase}.*\\.");
  }
  public static void main(String[] args) {
    System.out.println(matches("This is correct."));
    System.out.println(matches("bad sentence 1."));
    System.out.println(matches("Bad sentence 2"));
    System.out.println(matches("This is also correct..."));
  }
} /* Output:
true
false
false
true
*///:~
```

It's easy to test for a capital letter in a regular expression, but not in all languages. This exercise highlights the difficulty of writing multilingual applications. In the rest of the text, we use only U.S. English.

Exercise 8

```
//: strings/E08_Splitting2.java
/****************** Exercise 8 ******************
 * Split the string Splitting.knights on the words
 * "the" or "you."
 ***********************************************/
package strings;
import java.util.*;

public class E08_Splitting2 {
  public static void split(String regex) {
    System.out.println(
      Arrays.toString(Splitting.knights.split(regex)));
  }
  public static void main(String[] args) {
    split("the|you");
  }
} /* Output:
[Then, when ,  have found ,  shrubbery, ,  must cut down ,
mightiest tree in ,  forest... with... a herring!]
*///:~
```

Exercise 9

```
//: strings/E09_Replacing2.java
/****************** Exercise 9 ******************
 * Using the documentation for java.util.regex.Pattern
 * as a resource, replace all the vowels in
 * Splitting.knights with underscores.
 ***********************************************/
package strings;

public class E09_Replacing2 {
  public static void main(String[] args) {
     System.out.println(Splitting.knights.replaceAll(
       "(?i)[aeiou]","_"));
  }
} /* Output:
Th_n, wh_n y__ h_v_ f__nd th_ shr_bb_ry, y__ m_st c_t d_wn
th_ m_ght__st tr__ _n th_ f_r_st... w_th... _ h_rr_ng!
*///:~
```

The embedded flag expression (?i) enables case-insensitive matching. The regular expression demonstrates how to handle mixed uppercase/lowercase vowels, (though none occur above).

Exercise 10

```
//: strings/E10_CheckForMatch.java
/********************* Exercise 10 *********************
 * For the phrase "Java now has regular expressions"
 * evaluate whether the following expressions will find a
 * match:
 *
 *   ^Java
 *   \Breg.*
 *   n.w\s+h(a|i)s
 *   s?
 *   s*
 *   s+
 *   s{4}
 *   s{1}.
 *   s{0,3}
 *******************************************************/
package strings;
import java.util.regex.*;

public class E10_CheckForMatch {
  public static void main(String[] args) {
    String source = "Java now has regular expressions";
    String[] regEx = {"^Java", "\\Breg.*",
      "n.w\\s+h(a|i)s", "s?", "s*", "s+", "s{4}", "s{1}.",
      "s{0,3}"};
    System.out.println("Source string: " + source);
    for(String pattern : regEx) {
      System.out.println(
        "Regular expression: \"" + pattern + "\"");
      Pattern p = Pattern.compile(pattern);
      Matcher m = p.matcher(source);
      while(m.find()) {
        System.out.println("Match \"" + m.group() +
        "\" at positions " + m.start() + "-" +
        (m.end() - 1));
      }
    }
  }
} /* Output:
```

```
Source string: Java now has regular expressions
Regular expression: "^Java"
Match "Java" at positions 0-3
Regular expression: "\Breg.*"
Regular expression: "n.w\s+h(a|i)s"
Match "now has" at positions 5-11
Regular expression: "s?"
Match "" at positions 0--1
Match "" at positions 1-0
Match "" at positions 2-1
Match "" at positions 3-2
Match "" at positions 4-3
Match "" at positions 5-4
Match "" at positions 6-5
Match "" at positions 7-6
Match "" at positions 8-7
Match "" at positions 9-8
Match "" at positions 10-9
Match "s" at positions 11-11
Match "" at positions 12-11
Match "" at positions 13-12
Match "" at positions 14-13
Match "" at positions 15-14
Match "" at positions 16-15
Match "" at positions 17-16
Match "" at positions 18-17
Match "" at positions 19-18
Match "" at positions 20-19
Match "" at positions 21-20
Match "" at positions 22-21
Match "" at positions 23-22
Match "" at positions 24-23
Match "" at positions 25-24
Match "s" at positions 26-26
Match "s" at positions 27-27
Match "" at positions 28-27
Match "" at positions 29-28
Match "" at positions 30-29
Match "s" at positions 31-31
Match "" at positions 32-31
Regular expression: "s*"
Match "" at positions 0--1
Match "" at positions 1-0
Match "" at positions 2-1
Match "" at positions 3-2
Match "" at positions 4-3
Match "" at positions 5-4
```

```
Match "" at positions 6-5
Match "" at positions 7-6
Match "" at positions 8-7
Match "" at positions 9-8
Match "" at positions 10-9
Match "s" at positions 11-11
Match "" at positions 12-11
Match "" at positions 13-12
Match "" at positions 14-13
Match "" at positions 15-14
Match "" at positions 16-15
Match "" at positions 17-16
Match "" at positions 18-17
Match "" at positions 19-18
Match "" at positions 20-19
Match "" at positions 21-20
Match "" at positions 22-21
Match "" at positions 23-22
Match "" at positions 24-23
Match "" at positions 25-24
Match "ss" at positions 26-27
Match "" at positions 28-27
Match "" at positions 29-28
Match "" at positions 30-29
Match "s" at positions 31-31
Match "" at positions 32-31
Regular expression: "s+"
Match "s" at positions 11-11
Match "ss" at positions 26-27
Match "s" at positions 31-31
Regular expression: "s{4}"
Regular expression: "s{1}."
Match "s " at positions 11-12
Match "ss" at positions 26-27
Regular expression: "s{0,3}"
Match "" at positions 0--1
Match "" at positions 1-0
Match "" at positions 2-1
Match "" at positions 3-2
Match "" at positions 4-3
Match "" at positions 5-4
Match "" at positions 6-5
Match "" at positions 7-6
Match "" at positions 8-7
Match "" at positions 9-8
Match "" at positions 10-9
Match "s" at positions 11-11
```

```
Match "" at positions 12-11
Match "" at positions 13-12
Match "" at positions 14-13
Match "" at positions 15-14
Match "" at positions 16-15
Match "" at positions 17-16
Match "" at positions 18-17
Match "" at positions 19-18
Match "" at positions 20-19
Match "" at positions 21-20
Match "" at positions 22-21
Match "" at positions 23-22
Match "" at positions 24-23
Match "" at positions 25-24
Match "ss" at positions 26-27
Match "" at positions 28-27
Match "" at positions 29-28
Match "" at positions 30-29
Match "s" at positions 31-31
Match "" at positions 32-31
*///:~
```

Exercise 11

```
//: strings/E11_CheckForMatch2.java
/************************* Exercise 11 *****************
 *Apply the regular expression
 *     (?i)((^[aeiou])|(\s+[aeiou]))\w+?[aeiou]\b
 *    to
 *   "Arline ate eight apples and one orange while Anita
 *    hadn't any"
 ******************************************************/
package strings;
import java.util.regex.*;

public class E11_CheckForMatch2 {
  public static void main(String[] args) {
    Pattern p = Pattern.compile(
      "(?i)((^[aeiou])|(\\s+[aeiou]))\\w+?[aeiou]\\b");
    Matcher m = p.matcher("Arline ate eight apples and " +
      "one orange while Anita hadn't any");
    while(m.find()) {
      System.out.println("Match \"" + m.group() +
        "\" at positions " + m.start() + "-" +
        (m.end() - 1));
    }
```

```
  }
} /* Output:
Match "Arline" at positions 0-5
Match " ate" at positions 6-9
Match " one" at positions 27-30
Match " orange" at positions 31-37
Match " Anita" at positions 44-49
*///:~
```

Exercise 12

```
//: strings/E12_Groups2.java
/************************ Exercise 12 *****************
 * Modify Groups.java to count all unique words
 * with no initial capital letter.
 ******************************************************/
package strings;
import java.util.*;
import java.util.regex.*;
import static net.mindview.util.Print.*;

public class E12_Groups2 {
  public static void main(String[] args) {
    Set<String> words = new HashSet<String>();
    Matcher m =
      Pattern.compile("\\b((?![A-Z])\\w+)\\b")
        .matcher(Groups.POEM);
    while(m.find())
      words.add(m.group(1));
    print("Number of unique words = " + words.size());
    print(words.toString());
  }
} /* Output:
Number of unique words = 25
[catch, jaws, shun, raths, were, brillig, borogoves, mome,
frumious, son, the, claws, in, my, wabe, gimble, and, bird,
that, bite, slithy, mimsy, outgrabe, gyre, toves]
*///:~
```

The regular expression represented by **\b\w+\b** allows you to perform a "whole words" search (in Java it becomes **\\b\\w+\\b**). The program tests for initial capitals with a *negative lookahead*. A *lookaround* (the common name for *lookahead* and *lookbehind*) is sometimes indispensable, and often makes the whole regular expression seem more intuitive. As an additional exercise, rewrite this expression with no *lookaround*.

Exercise 13

```
//: strings/E13_StartEnd2.java
/************************* Exercise 13 *****************
 * Modify StartEnd.java so that it uses Groups.POEM as
 * input, but still produces positive outputs for find(),
 * lookingAt() and matches().
 ******************************************************/
package strings;
import java.util.regex.*;
import static net.mindview.util.Print.*;

public class E13_StartEnd2 {
  private static class Display {
    private boolean regexPrinted = false;
    private String regex;
    Display(String regex) { this.regex = regex; }
    void display(String message) {
      if(!regexPrinted) {
        print(regex);
        regexPrinted = true;
      }
      print(message);
    }
  }
  static void examine(String s, String regex) {
    Display d = new Display(regex);
    Pattern p = Pattern.compile(regex);
    Matcher m = p.matcher(s);
    while(m.find())
      d.display("find() '" + m.group() +
        "' start = "+ m.start() + " end = " + m.end());
    if(m.lookingAt()) // No reset() necessary
      d.display("lookingAt() start = "
        + m.start() + " end = " + m.end());
    if(m.matches()) // No reset() necessary
      d.display("matches() start = "
        + m.start() + " end = " + m.end());
  }
  public static void main(String[] args) {
    for(String in : Groups.POEM.split("\n")) {
      print("input : " + in);
      for(String regex : new String[] {"\\w*ere\\w*",
        "\\w*at", "t\\w+", "T.*?."})
        examine(in, regex);
    }
```

```
  }
} /* Output:
input : Twas brillig, and the slithy toves
t\w+
find() 'the' start = 18 end = 21
find() 'thy' start = 25 end = 28
find() 'toves' start = 29 end = 34
T.*?.
find() 'Tw' start = 0 end = 2
lookingAt() start = 0 end = 2
matches() start = 0 end = 34
input : Did gyre and gimble in the wabe.
t\w+
find() 'the' start = 23 end = 26
input : All mimsy were the borogoves,
\w*ere\w*
find() 'were' start = 10 end = 14
t\w+
find() 'the' start = 15 end = 18
input : And the mome raths outgrabe.
\w*at
find() 'rat' start = 13 end = 16
t\w+
find() 'the' start = 4 end = 7
find() 'ths' start = 15 end = 18
find() 'tgrabe' start = 21 end = 27
input :
input : Beware the Jabberwock, my son,
t\w+
find() 'the' start = 7 end = 10
input : The jaws that bite, the claws that catch.
\w*at
find() 'that' start = 9 end = 13
find() 'that' start = 30 end = 34
find() 'cat' start = 35 end = 38
t\w+
find() 'that' start = 9 end = 13
find() 'te' start = 16 end = 18
find() 'the' start = 20 end = 23
find() 'that' start = 30 end = 34
find() 'tch' start = 37 end = 40
T.*?.
find() 'Th' start = 0 end = 2
lookingAt() start = 0 end = 2
matches() start = 0 end = 41
input : Beware the Jubjub bird, and shun
t\w+
```

```
find() 'the' start = 7 end = 10
input : The frumious Bandersnatch.
\w*at
find() 'Bandersnat' start = 13 end = 23
t\w+
find() 'tch' start = 22 end = 25
T.*?.
find() 'Th' start = 0 end = 2
lookingAt() start = 0 end = 2
matches() start = 0 end = 26
*///:~
```

Exercise 14

```
//: strings/E14_SplitDemo2.java
/************************ Exercise 14 *****************
 * Rewrite SplitDemo using String.split().
 ******************************************************/
package strings;
import java.util.*;
import static net.mindview.util.Print.*;

public class E14_SplitDemo2 {
  public static void main(String[] args) {
    String input =
      "This!!unusual use!!of exclamation!!points";
    print(Arrays.toString(input.split("!!")));
    // Only do the first three:
    print(Arrays.toString(input.split("!!", 3)));
  }
} /* Output:
[This, unusual use, of exclamation, points]
[This, unusual use, of exclamation!!points]
*///:~
```

Exercise 15

```
//: strings/E15_JGrep2.java
// {Args: E15_JGrep2.java "\b[Ssct]\w+" CASE_INSENSITIVE}
/********************** Exercise 15 *********************
 * Modify JGrep.java to accept flags as arguments (e.g.,
 * Pattern.CASE_INSENSITIVE, Pattern.MULTILINE).
 ******************************************************/
package strings;
import java.util.regex.*;
```

```
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E15_JGrep2 {
  public static void main(String[] args) throws Exception {
    if(args.length < 3) {
      print("Usage: java E15_JGrep2 file regex pattern");
      print("pattern can take one of the following values");
      print("CANON_EQ, CASE_INSENSITIVE, COMMENTS, " +
        "DOTALL, MULTILINE, UNICODE_CASE, UNIX_LINES");
      System.exit(0);
    }
    int flag = 0;
    if(args[2].equalsIgnoreCase("CANON_EQ")) {
      flag = Pattern.CANON_EQ;
    } else
      if(args[2].equalsIgnoreCase("CASE_INSENSITIVE")) {
      flag = Pattern.CASE_INSENSITIVE;
    } else if(args[2].equalsIgnoreCase("COMMENTS")) {
      flag = Pattern.COMMENTS;
    } else if(args[2].equalsIgnoreCase("DOTALL")) {
      flag = Pattern.DOTALL;
    } else if(args[2].equalsIgnoreCase("MULTILINE")) {
      flag = Pattern.MULTILINE;
    } else if(args[2].equalsIgnoreCase("UNICODE_CASE")) {
      flag = Pattern.UNICODE_CASE;
    } else if(args[2].equalsIgnoreCase("UNIX_LINES")) {
      flag = Pattern.UNIX_LINES;
    }
    Pattern p = Pattern.compile(args[1], flag);
    // Iterate through the lines of the input file:
    int index = 0;
    Matcher m = p.matcher("");
    for(String line : new TextFile(args[0])) {
      m.reset(line);
      while(m.find())
        print(index++ + ": " + m.group() + ": " +
          m.start());
    }
  }
} /* Output: (Sample)
0: strings: 4
1: Ssct: 30
2: CASE_INSENSITIVE: 40
3: to: 21
4: CASE_INSENSITIVE: 11
5: static: 7
```

```
6: class: 7
7: static: 9
8: String: 26
9: throws: 41
10: can: 21
11: take: 25
12: the: 37
13: CANON_EQ: 13
14: CASE_INSENSITIVE: 23
15: COMMENTS: 41
16: System: 6
17: CANON_EQ: 33
18: CANON_EQ: 21
19: CASE_INSENSITIVE: 35
20: CASE_INSENSITIVE: 21
21: COMMENTS: 40
22: COMMENTS: 21
23: compile: 24
24: through: 15
25: the: 23
26: the: 36
27: String: 8
28: TextFile: 26
29: start: 12
*///:~
```

Exercise 16

```
//: strings/E16_JGrep3.java
// {Args: E16_JGrep3.java java}
/********************** Exercise 16 *********************
 * Modify JGrep.java to accept a directory name or a file
 * name as argument (if a directory is provided, search
 * should include all files in the directory).
 * Hint: you can generate a list of filenames with
 * File[] files = new File(".").listFiles();
 *******************************************************/
package strings;
import java.io.*;
import java.util.regex.*;
import net.mindview.util.*;

public class E16_JGrep3 {
  public static void main(String[] args) throws Exception {
    if(args.length < 2) {
      System.out.println(
```

```
          "Usage: java E16_JGrep3 file regex");
        System.exit(0);
      }
      File file = new File(args[0]);
      File[] files = null;
      if(file.isDirectory()) files = file.listFiles();
      else files = new File[]{new File(args[0])};
      Pattern p = Pattern.compile(args[1]);
      Matcher m = p.matcher("");
      for(File f : files) {
        System.out.println("-- File:" + f.getName() + "--");
        // Iterate through the lines of the input file:
        int index = 0;
        for(String line : new TextFile(f.getAbsolutePath())) {
          m.reset(line);
          while(m.find())
            System.out.println(index++ + ": " +
              m.group() + ": " + m.start());
        }
      }
    }
} /* Output: (Sample)
-- File:E16_JGrep3.java--
0: java: 23
1: java: 21
2: java: 26
3: java: 16
4: java: 7
5: java: 7
6: java: 16
*///:~
```

Exercise 17

This exercise looks like the previous ones; you just follow the program template, applying the right regular expression to extract comments from a Java source file. However, it's not that simple. We need state information to distinguish between a real comment and text that *looks* like a comment (for example, `"//this is part of a string and not a comment"`). The program needs more than a search facility based in regular expression to locate itself inside the source code: it needs a miniature Java source code parser. We can either write a parser or enhance an existing general parser (e.g., to display comments). We can reuse a stable base code to great advantage, as most of the work is already done.

Compare the alternative methods in the next three solutions. (The second one amounts to a miniature tutorial for **JavaCC.**) Just provide the same input for both and check the differences in the outputs.

Alternative A

```
//: strings/E17_JCommentExtractor.java
// {Args: E17_JCommentExtractor.java}
/********************** Exercise 17 *********************
 * Write a program that reads a Java source-code file (you
 * provide the file name on the command line) and displays
 * all the comments.
 ********************************************************/
package strings;
import java.util.regex.*;
import net.mindview.util.*;

public class E17_JCommentExtractor {
  static final String CMNT_EXT_REGEX =
    "(?x)(?m)(?s) # Comments, Multiline & Dotall: ON\n" +
    "/\\*         # Match START OF THE COMMENT\n" +
    "(.*?)        # Match all chars\n" +
    "\\*/         # Match END OF THE COMMENT\n" +
    "| //(.*?)$   # OR Match C++ style comments\n";
  public static void main(String[] args) throws Exception {
    if(args.length < 1) {
      System.out.println(
        "Usage: java E17_JCommentExtractor file");
      System.exit(0);
    }
    String src = TextFile.read(args[0]);
    Pattern p = Pattern.compile(CMNT_EXT_REGEX);
    Matcher m = p.matcher(src);
    while(m.find())
       System.out.println(m.group(1) != null ?
         m.group(1) : m.group(2));
  }
} /* Output: (Sample)
: strings/E17_JCommentExtractor.java
 {Args: E17_JCommentExtractor.java}
********************** Exercise 17 **********************
 * Write a program that reads a Java source-code file (you
 * provide the file name on the command line) and displays
 * all the comments.
 ********************************************************
(.*?)$   # OR Match C++ style comments\n";
```

```
/:~
*///:~
```

We must remove the comment markers to compile the new source code (created by the automated test system). The **(?x)** flag enables the so-called COMMENTS mode, white space is ignored, and embedded comments starting with # are ignored until the end of a line.

However, the program is flawed; it erroneously reports as a comment part of the regular expression pattern clearly situated inside a Java string (like **(.*?)$ # OR Match C++ style comments\n";**).

Alternative B

JavaCC is a parser generator for Java applications (free at https://javacc.dev.java.net/). A parser generator converts a grammar specification to a Java program that recognizes the grammar.

We can cover only part of the topic of parser generators here, so we look at only those changes we must make to an existing J2SE5 grammar file (**Java1.5.jj**). Assume **JavaCC** is installed and that we can build the example in the **examples\JavaGrammars\1.5** directory (see its **README** file for instructions).

Java1.5.jj contains everything to produce a fully functional J2SE5 parser. At this level the parser signals only whether the input is a legal Java program, but as that solves 99% of exercises 17, 18 and 19, we present the solution for all three at once.

The program, *Java Code Sniffer*, accepts two command line arguments, the input file and the option to indicate a processing choice ("C" for Display Comments, "S" for Display String Literals, and "N" for Display Class Names). An invalid invocation causes usage instructions to appear onscreen.

We build the program with the **CodeSniffer.jj** grammar file and the **Token.java** file (from the **examples\JavaGrammars\1.5** directory).

Follow these steps to create **CodeSniffer.jj**:

1. Copy **Java1.5.jj** into a local folder and rename it **CodeSniffer.jj**;

2. Copy **Token.java** into the **CodeSniffer.jj** folder (this may not be necessary for later releases of **JavaCC**);

3. Open **CodeSniffer.jj** in a text editor;

4. Insert the following code fragment right after the line **import java.io.***:

```
import static net.mindview.util.Print.*;

enum OpMode {
  DisplayComments, DisplayStrings, DisplayClassNames
}
```

5. Replace the **main( )** method with the following code:

```
static OpMode mode;
public static void main(String args[]) {
  JavaParser parser;
  if (args.length == 2) {
    print(
      "Java Code Sniffer Version 1.0:  Reading from file " +
      args[0] + " . . .");
    try {
      parser = new JavaParser(
        new java.io.FileInputStream(args[0]));
    } catch (java.io.FileNotFoundException e) {
      print("Java Code Sniffer Version 1.0:  File " +
        args[0] + " not found.");
      return;
    }
    if (args[1].equalsIgnoreCase("c")) {
      mode = OpMode.DisplayComments;
      print("DISPLAYING COMMENTS...\n");
    } else if (args[1].equalsIgnoreCase("s")) {
      mode = OpMode.DisplayStrings;
      print("DISPLAYING STRING LITERALS...\n");
    } else if (args[1].equalsIgnoreCase("n")) {
      mode = OpMode.DisplayClassNames;
      print("DISPLAYING CLASS NAMES...\n");
    } else {
      usage();
      return;
    }
  } else {
    usage();
    return;
  }
  try {
    parser.CompilationUnit();
    print("\nJava Code Sniffer Version 1.0:  Java program" +
      " parsed successfully.");
  } catch (ParseException e) {
    print(e.getMessage());
    print("\nJava Code Sniffer Version 1.0:  Encountered " +
```

```
        "errors during parse.");
    }
  }
  private static void usage() {
    print("Java Code Sniffer Version 1.0:  Usage is:");
    print("         java JavaParser file <option>");
    print("where option is one of:");
    print("         C - Display Comments");
    print("         S - Display String Literals");
    print("         N - Display Class Names");
  }
```

Save your changes and everything should compile. The parser now recognizes command line arguments and prints usage information. Boilerplate code implements the corresponding actions.

Now build the parser using the steps below. (Assume the grammar file is in the current working directory and that **CLASSPATH** is properly set):

- `javacc CodeSniffer.jj`
- `javac *.java`
- `java JavaParser JavaParser.java C`

We add only three lines of code to display comments. It's surprisingly easy to build a parser with **JavaCC** because you're reusing code, and it tends to be less of a struggle than finding the right regular expression to extract comments from the source code (However, you must still know regular expressions in order to understand the grammar, because the parser uses them to process input text).

To solve the exercise:

1. Insert the following code fragment immediately after **<SINGLE_LINE_COMMENT: "\n" | "\r" | "\r\n" >**:

```
{ if (JavaParser.mode == OpMode.DisplayComments)
    printnb(image.toString()); }
```

The modified grammar should look like this:

```
<IN_SINGLE_LINE_COMMENT>
SPECIAL_TOKEN :
{
  <SINGLE_LINE_COMMENT: "\n" | "\r" | "\r\n" >
  { if (JavaParser.mode == OpMode.DisplayComments)
      printnb(image.toString()); } : DEFAULT
}
```

2. Insert the next code fragment right after both the **<FORMAL_COMMENT: "*/" >** and **<MULTI_LINE_COMMENT: "*/" >** constructions:

```
{ if (JavaParser.mode == OpMode.DisplayComments)
    print(image.toString()); }
```

Passing **image.toString( )** to a custom routine allows us to pre-process extracted comments before sending them to standard output. Try removing the comment markers as an additional exercise.

Exercise 18

Alternative A

```
//: strings/E18_JStringExtractor.java
// {Args: E18_JStringExtractor.java}
/********************** Exercise 18 *********************
 * Write a program that reads a Java source-code file
 * and displays all the string literals in the code
 * (provide the file name on the command line).
 *******************************************************/
package strings;
import java.util.regex.*;
import net.mindview.util.*;

public class E18_JStringExtractor {
  static final String STR_EXT_REGEX =
    "\"(?:[^\"\\\\\\n\\r]|(?:\\\\(?:[untbrf\\\\'\"]"
    + "|[0-9A-Fa-f]{4}|[0-7]{1,2}|[0-3][0-7]{2})))*\"";
  public static void main(String[] args) throws Exception {
    if(args.length < 1) {
      System.out.println(
        "Usage: java E18_JStringExtractor file");
      System.exit(0);
    }
    String src = TextFile.read(args[0]);
    Pattern p =
      Pattern.compile(STR_EXT_REGEX);
    Matcher m = p.matcher(src);
    while(m.find())
       System.out.println(m.group().
         substring(1, m.group().length() - 1));

    // "This is NOT a string but a comment!"
    String dummy = "\u003F\u003f\n\060\607";
```

```
  }
} /* Output:
\"(?:[^\"\\\\\\n\\r]|(?:\\\\(?:[untbrf\\\\'\"]
|[0-9A-Fa-f]{4}|[0-7]{1,2}|[0-3][0-7]{2})))*\"
Usage: java E18_JStringExtractor file
This is NOT a string but a comment!
\u003F\u003f\n\060\607
*///:~
```

The program incorrectly treats “string literals” inside comments as real strings.

We pruned away string literal markers to avoid confusing *TIJ4*’s automated test system. Try commenting the regular expression as we did in the previous solution.

Alternative B

This solution uses **JavaCC** following Exercise 17, Alternative B. Display string literals by inserting the following code fragment:

```
{ if (JavaParser.mode == OpMode.DisplayStrings)
    print(image.toString()); }
```

immediately after the construction:

```
< STRING_LITERAL:
      "\""
      (   (~["\"","\\","\n","\r"])
        | ("\\"
            ( ["n","t","b","r","f","\\","'","\""]
            | ["0"-"7"] ( ["0"-"7"] )?
            | ["0"-"3"] ["0"-"7"] ["0"-"7"]
            )
          )
      )*
      "\""
  >
```

JavaCC runs java-like Unicode escape processing automatically, so you see processed characters rather than raw \uXXXX forms in the output.

Passing **image.toString( )** to a custom routine allows us to further process extracted string literals before they go to standard output. Try to remove the string literal markers as an additional exercise.

Exercise 19

This exercise extends Exercise 18.

First, we must unambiguously define "class names used in a particular program." The program will dump type names that occur in the following contexts:

- Class declaration (including class names on the **extends** list)
- Object creation

The program won't report used class names because we can't tell whether a particular reference type is an interface or a class without further examination (probably involving other external files).

Alternative A

```
//: strings/E19_JClassUsageReporter.java
// {Args: E19_JClassUsageReporter.java}
/********************** Exercise 19 *********************
 * Build on Exercises 17 and 18 to write a program
 * that examines Java source code and produces all
 * class names used in a particular program.
 *******************************************************/
package strings;
import java.util.regex.*;
import net.mindview.util.*;

public class E19_JClassUsageReporter {
  private static final String Identifier =
    "[$A-Za-z_][$A-Za-z_0-9]*";
  private static final String ClassOrInterfaceType =
    Identifier + "(?:\\." + Identifier + ")*";
  static final String CU_REP_REGEX =
    "class\\s+" + Identifier +
    "|extends\\s+" + ClassOrInterfaceType +
    "|new\\s+" + ClassOrInterfaceType;
  public static void main(String[] args) throws Exception {
    if(args.length < 1) {
      System.out.println(
        "Usage: java E19_JClassUsageReporter file");
      System.exit(0);
    }
    String src = TextFile.read(args[0]);
    // Prune away comments ...
```

```
    src = src.replaceAll(
      E17_JCommentExtractor.CMNT_EXT_REGEX, "");
    // Prune away string literals ...
    src = src.replaceAll(
      E18_JStringExtractor.STR_EXT_REGEX, "");
    Pattern p = Pattern.compile(CU_REP_REGEX);
    Matcher m = p.matcher(src);
    while(m.find())
       System.out.println(m.group());
  }
} /* Output:
class E19_JClassUsageReporter
*///:~
```

Here we make a complex regular expression more comprehensible by splitting it into constituent parts and assembling the pieces.

Alternative B

Incorporate the desired functionality using these steps:

1. Locate the method definition for **ClassOrInterfaceDeclaration(int modifiers)** then insert the following variable definition immediately after **boolean isInterface = false**:

```
Token t;
```

2. Replace the **<IDENTIFIER>** statement (inside the same method) with this code fragment:

```
t = <IDENTIFIER>
{if (!isInterface &&
     JavaParser.mode == OpMode.DisplayClassNames)
  print("class: " + t.image.toString());}
```

3. Locate the method definition **ExtendsList(boolean isInterface)** and replace the piece of code that starts with "**extends**" and ends before **{**:

```
"extends"
{if (JavaParser.mode == OpMode.DisplayClassNames)
  print("extends:");}
ClassOrInterfaceType(!isInterface)
("," ClassOrInterfaceType(!isInterface)
```

4. Replace the method **ClassOrInterfaceType( )** with the following:

```
void ClassOrInterfaceType(Boolean... args):
{
  Token t;
```

```
  boolean shouldPrint = args.length == 1 &&
    args[0].booleanValue() &&
    JavaParser.mode == OpMode.DisplayClassNames;
}
{
  t = <IDENTIFIER>
  {if (shouldPrint) printnb(t.image.toString());}
  [ LOOKAHEAD(2) TypeArguments() ]
  ( LOOKAHEAD(2) "." t = <IDENTIFIER>
    {if (shouldPrint) printnb("." + t.image.toString());}
    [ LOOKAHEAD(2) TypeArguments() ]
  )*
  {if (shouldPrint) print();}
}
```

5. Locate the method **AllocationExpression( )** and replace the **ClassOrInterfaceType( )** statement with this code fragment:

```
{if (JavaParser.mode == OpMode.DisplayClassNames)
  printnb("new: ");}
ClassOrInterfaceType(true)
```

If you instantiate a class name from more than one place, both programs will display it multiple times, and will list even those interface names used to create anonymous inner classes. As an exercise, write a custom component that filters out duplicate class names, or arrange the output in some other way.

Exercise 20

```
//: strings/E20_Scanner.java
/********************* Exercise 20 *******************
 * Create a class that contains int, long, float and double
 * and String fields. Create a constructor for this class
 * that takes a single String argument, and scans that
 * string into the various fields. Add a toString() method
 * and demonstrate that your class works correctly.
 ****************************************************/
package strings;
import java.util.*;

class DataHolder2 {
  private int i;
  private long l;
  private float f;
  private double d;
  private String s;
```

```
  DataHolder2(String data) {
    Scanner stdin = new Scanner(data);
    stdin.useLocale(Locale.US);
    i = stdin.nextInt();
    l = stdin.nextLong();
    f = stdin.nextFloat();
    d = stdin.nextDouble();
    s = stdin.next("\\w+");
  }
  public String toString() {
    return i + " " + l + " " + f + " " + d + " " + s;
  }
}

public class E20_Scanner {
  public static void main(String[] args) {
    DataHolder2 dh =
      new DataHolder2("1 10000000000000 1.1 1e55 Howdy Hi");
    System.out.println(dh.toString());
  }
} /* Output:
1 10000000000000 1.1 1.0E55 Howdy
*///:~
```

Type Information

Exercise 1

```
//: typeinfo/E01_ToyTest.java
/********************* Exercise 1 *********************
 * In ToyTest.java, comment out Toy's default constructor
 * and explain what happens.
 ******************************************************/
package typeinfo;
import static net.mindview.util.Print.*;

// Copy-pasted because these interfaces are not public
interface HasBatteries {}
interface Waterproof {}
interface Shoots {}

class Toy {
  //Toy() {}
  Toy(int i) {}
}

class FancyToy extends Toy
implements HasBatteries, Waterproof, Shoots {
  FancyToy() { super(1); }
}

public class E01_ToyTest {
  static void printInfo(Class<?> cc) {
    print("Class name: " + cc.getName() +
      " is interface? [" + cc.isInterface() + "]");
    print("Simple name: " + cc.getSimpleName());
    print("Canonical name : " + cc.getCanonicalName());
  }
  public static void main(String[] args) {
    Class<?> c = null;
    try {
      c = Class.forName("typeinfo.FancyToy");
    } catch(ClassNotFoundException e) {
      print("Can't find FancyToy");
      return;
    }
```

```
    printInfo(c);
    for(Class<?> face : c.getInterfaces())
      printInfo(face);
    Class<?> up = c.getSuperclass();
    Object obj = null;
    try {
      // Requires default constructor:
      obj = up.newInstance();
    } catch(InstantiationException e) {
      print("Cannot instantiate");
      return;
    } catch(IllegalAccessException e) {
      print("Cannot access");
      return;
    }
    printInfo(obj.getClass());
  }
} /* Output:
Class name: typeinfo.FancyToy is interface? [false]
Simple name: FancyToy
Canonical name : typeinfo.FancyToy
Class name: typeinfo.HasBatteries is interface? [true]
Simple name: HasBatteries
Canonical name : typeinfo.HasBatteries
Class name: typeinfo.Waterproof is interface? [true]
Simple name: Waterproof
Canonical name : typeinfo.Waterproof
Class name: typeinfo.Shoots is interface? [true]
Simple name: Shoots
Canonical name : typeinfo.Shoots
Cannot instantiate
*///:~
```

You must define the required default constructor for **up.newInstance( )**; the compiler can't create it because a non-default constructor already exists.

Exercise 2

```
//: typeinfo/E02_ToyTest2.java
/********************* Exercise 2 *********************
 * Incorporate a new kind of interface into ToyTest.java
 * and verify that it is detected and displayed properly.
 ******************************************************/
package typeinfo;
import static net.mindview.util.Print.*;
```

```
interface HasCPU {}

class FancierToy extends FancyToy implements HasCPU {}

public class E02_ToyTest2 {
  static void printInfo(Class<?> cc) {
    print("Class name: " + cc.getName() +
      " is interface? [" + cc.isInterface() + "]");
    print("Simple name: " + cc.getSimpleName());
    print("Canonical name : " + cc.getCanonicalName());
  }
  public static void main(String[] args) {
    Class<?> c = null;
    try {
      c = Class.forName("typeinfo.FancierToy");
    } catch(ClassNotFoundException e) {
      print("Can't find FancierToy");
      System.exit(1);
    }
    printInfo(c);
    for(Class<?> face : c.getInterfaces())
      printInfo(face);
    Class<?> up = c.getSuperclass();
    Object obj = null;
    try {
      obj = up.newInstance();
    } catch(InstantiationException e) {
      print("Cannot instantiate");
      System.exit(1);
    } catch(IllegalAccessException e) {
      print("Cannot access");
      System.exit(1);
    }
    printInfo(obj.getClass());
  }
} /* Output:
Class name: typeinfo.FancierToy is interface? [false]
Simple name: FancierToy
Canonical name : typeinfo.FancierToy
Class name: typeinfo.HasCPU is interface? [true]
Simple name: HasCPU
Canonical name : typeinfo.HasCPU
Class name: typeinfo.FancyToy is interface? [false]
Simple name: FancyToy
Canonical name : typeinfo.FancyToy
*///:~
```

Exercise 3

```
//: typeinfo/E03_Rhomboid.java
/****************** Exercise 3 ******************
 * Add Rhomboid to Shapes.java. Create a
 * Rhomboid, upcast it to a Shape, then downcast
 * it back to a Rhomboid. Try downcasting to a
 * Circle and see what happens.
 ***********************************************/
package typeinfo;
import java.util.*;

class Rhomboid extends Shape {
  public String toString() { return "Rhomboid"; }
}

public class E03_Rhomboid {
  public static void main(String[] args) {
    List<Shape> shapes = Arrays.asList(
      new Circle(), new Square(), new Triangle(),
      new Rhomboid()
    );
    for(Shape shape : shapes)
      shape.draw();
    // Upcast to shape:
    Shape shape = new Rhomboid();
    // Downcast to Rhomboid:
    Rhomboid r = (Rhomboid)shape;
    // Downcast to Circle. Succeeds at compile time,
    // but fails at runtime with a ClassCastException:
    //! Circle c = (Circle)shape;
  }
} /* Output:
Circle.draw()
Square.draw()
Triangle.draw()
Rhomboid.draw()
*///:~
```

Exercise 4

```
//: typeinfo/E04_Instanceof.java
/****************** Exercise 4 ******************
 * Modify the previous exercise so that it uses
 * instanceof to check the type before performing
```

```
 * the downcast.
 *********************************************/
package typeinfo;
public class E04_Instanceof {
  public static void main(String[] args) {
    // Upcast to shape:
    Shape shape = new Rhomboid();
    // Downcast to Rhomboid:
    Rhomboid r = (Rhomboid)shape;
    // Test before Downcasting:
    Circle c = null;
    if(shape instanceof Circle)
      c = (Circle)shape;
    else
      System.out.println("shape not a Circle");
  }
} /* Output:
shape not a Circle
*///:~
```

Exercise 5

```
//: typeinfo/E05_RotateShapes.java
/****************** Exercise 5 ******************
 * Implement a rotate(Shape) method in Shapes.java,
 * such that it checks to see if it is rotating a
 * Circle (and, if so, doesn't perform the
 * operation).
 *********************************************/
package typeinfo;
import java.util.*;

abstract class RShape {
  void draw() { System.out.println(this + ".draw()"); }
  abstract public String toString();
  void rotate(int degrees) {
    System.out.println("Rotating " + this +
      " by " + degrees + " degrees");
  }
}

class RCircle extends RShape {
  public String toString() { return "Circle"; }
  void rotate(int degrees) {
    throw new UnsupportedOperationException();
  }
```

```
}

class RSquare extends RShape {
  public String toString() { return "Square"; }
}

class RTriangle extends RShape {
  public String toString() { return "Triangle"; }
}

public class E05_RotateShapes {
  static void rotateAll(List<RShape> shapes) {
    for(RShape shape : shapes)
      if(!(shape instanceof RCircle))
        shape.rotate(45);
  }

  public static void main(String[] args) {
    List<RShape> shapes = Arrays.asList(
      new RCircle(), new RSquare(), new RTriangle()
    );
    rotateAll(shapes);
  }
} /* Output:
Rotating Square by 45 degrees
Rotating Triangle by 45 degrees
*///:~
```

This program performs the required check using **rotateAll( )** instead of **rotate(Shape)**.

Exercise 6

```
//: typeinfo/E06_Highlight.java
/****************** Exercise 6 ******************
 * Modify Shapes.java so that it can "highlight"
 * (set a flag) in all shapes of a particular
 * type. The toString() method for each derived
 * Shape should indicate whether that Shape is
 * "highlighted."
 ***********************************************/
package typeinfo;
import java.util.*;
import java.lang.reflect.*;

class HShape {
```

```
  boolean highlighted = false;
  public void highlight() { highlighted = true; }
  public void clearHighlight() { highlighted = false; }
  void draw() { System.out.println(this + " draw()"); }
  public String toString() {
    return getClass().getName() +
      (highlighted ? " highlighted" : " normal");
  }
  static List<HShape> shapes = new ArrayList<HShape>();
  HShape() { shapes.add(this); }
  // Basic approach (code duplication)
  static void highlight1(Class<?> type) {
    for(HShape shape : shapes)
      if(type.isInstance(shape))
        shape.highlight();
  }
  static void clearHighlight1(Class<?> type) {
    for(HShape shape : shapes)
      if(type.isInstance(shape))
        shape.clearHighlight();
  }
  // Create an applicator and reuse it. All exceptions
  // indicate programmer error, and are thus
  // RuntimeExceptions:
  static void forEach(Class<?> type, String method) {
    try {
      Method m =
        HShape.class.getMethod(method, (Class<?>[])null);
      for(HShape shape : shapes)
        if(type.isInstance(shape))
          m.invoke(shape, (Object[])null);
    } catch(Exception e) {
      throw new RuntimeException(e);
    }
  }
  static void highlight2(Class<?> type) {
    forEach(type, "highlight");
  }
  static void clearHighlight2(Class<?> type) {
    forEach(type, "clearHighlight");
  }
}

class HCircle extends HShape {}
class HSquare extends HShape {}
class HTriangle extends HShape {}
```

```
public class E06_Highlight {
  public static void main(String[] args) {
    List<HShape> shapes = Arrays.asList(
      new HCircle(), new HSquare(),
      new HTriangle(), new HSquare(),
      new HTriangle(), new HCircle(),
      new HCircle(), new HSquare());
    HShape.highlight1(HCircle.class);
    HShape.highlight2(HCircle.class);
    for(HShape shape : shapes)
      shape.draw();
    System.out.println("*******");
    // Highlight them all:
    HShape.highlight1(HShape.class);
    HShape.highlight2(HShape.class);
    for(HShape shape : shapes)
      shape.draw();
    System.out.println("*******");
    // Not in the hierarchy:
    HShape.clearHighlight1(ArrayList.class);
    HShape.clearHighlight2(ArrayList.class);
    for(HShape shape : shapes)
      shape.draw();
  }
} /* Output:
typeinfo.HCircle highlighted draw()
typeinfo.HSquare normal draw()
typeinfo.HTriangle normal draw()
typeinfo.HSquare normal draw()
typeinfo.HTriangle normal draw()
typeinfo.HCircle highlighted draw()
typeinfo.HCircle highlighted draw()
typeinfo.HSquare normal draw()
*******
typeinfo.HCircle highlighted draw()
typeinfo.HSquare highlighted draw()
typeinfo.HTriangle highlighted draw()
typeinfo.HSquare highlighted draw()
typeinfo.HTriangle highlighted draw()
typeinfo.HCircle highlighted draw()
typeinfo.HCircle highlighted draw()
typeinfo.HSquare highlighted draw()
*******
typeinfo.HCircle highlighted draw()
typeinfo.HSquare highlighted draw()
typeinfo.HTriangle highlighted draw()
typeinfo.HSquare highlighted draw()
```

```
typeinfo.HTriangle highlighted draw()
typeinfo.HCircle highlighted draw()
typeinfo.HCircle highlighted draw()
typeinfo.HSquare highlighted draw()
*///:~
```

We eliminate duplicate code by moving all the methods (starting with **toString**) into the base class, and use RTTI to determine the name of the class. **toString( )** also prints whether the object is highlighted.

You must keep track of all objects in a class to highlight or clear all objects of a particular type. We keep track of **shape** objects with the **static ArrayList shapes**; the **HShape** default constructor adds the current object to **shapes**. We set or clear a **boolean** flag, using methods in **HShape**, to change highlighting on a per-object basis.

The **static** method **highlight1( )** takes an argument of type **Class**, which is the type to highlight. It iterates through **shapes** and calls **highlight( )** for each matched object, testing for a match with **Class.isInstance( )**. **clearHighlight1( )** works the same way.

Alternatively, we could use reflection in **forEach( )** to combine redundant code that appears in **highlight1( )** and **clearHighlight1( )** during iteration through the list. **Highlight2( )** and **clearHighlight2( )** each call **forEach( )**.

To call **HShape.highlight( )**, pass the name of the specific **HShape** you want to highlight, with **.class** appended to produce the **Class** reference. If you pass **HShape.class** as the argument, it matches and highlights every **HShape** in the list. Use **clearHighlight( )** the same way to clear all highlighting.

If you pass a class that isn't in the **HShape** hierarchy, there's never a match so the **highlight** and **clearHighlight** methods do nothing.

Exercise 7

```
//: typeinfo/E07_CommandLoad.java
// {Args: typeinfo.Gum typeinfo.Cookie}
/****************** Exercise 7 ******************
 * Modify SweetShop.java so that each type of
 * object creation is controlled by a
 * command-line argument. That is, if your
 * command line is "java SweetShop Candy," then
 * only the Candy object is created. Notice how
 * you can control which Class objects are loaded
 * via the command-line argument.
 ***********************************************/
```

```
package typeinfo;
import static net.mindview.util.Print.*;

class Candy {
  static { print("Loading Candy"); }
}

class Gum {
  static { print("Loading Gum"); }
}

class Cookie {
  static { print("Loading Cookie"); }
}

public class E07_CommandLoad {
  public static void main(String[] args) throws Exception {
    for(String arg : args)
      Class.forName(arg);
  }
} /* Output:
Loading Gum
Loading Cookie
*///:~
```

Exercise 8

```
//: typeinfo/E08_RecursiveClassPrint.java
// {Args: typeinfo.Circle typeinfo.FancyToy}
/****************** Exercise 8 ******************
 * Write a method that takes an object and
 * recursively prints all the classes in that
 * object's hierarchy.
 *********************************************/
package typeinfo;
import static net.mindview.util.Print.*;

public class E08_RecursiveClassPrint {
  static void printClasses(Class<?> c) {
    // getSuperclass() returns null on Object:
    if(c == null) return;
    print(c.getName());
    // Produces the interfaces that this class
    // implements:
    for(Class<?> k : c.getInterfaces()) {
      print("Interface: " + k.getName());
```

```
      printClasses(k.getSuperclass());
    }
    printClasses(c.getSuperclass());
  }
  public static void main(String args[]) throws Exception {
    for(int i = 0; i < args.length; i++) {
      print("Displaying " + args[i]);
      printClasses(Class.forName(args[i]));
      if(i < args.length - 1)
        System.out.println("==================");
    }
  }
} /* Output:
Displaying typeinfo.Circle
typeinfo.Circle
typeinfo.Shape
java.lang.Object
==================
Displaying typeinfo.FancyToy
typeinfo.FancyToy
Interface: typeinfo.HasBatteries
Interface: typeinfo.Waterproof
Interface: typeinfo.Shoots
typeinfo.Toy
java.lang.Object
*///:~
```

We enhance the solution a bit, running it on **FancyToy** (from Exercise 1) to print all the interfaces.

Exercise 9

```
//: typeinfo/E09_GetDeclaredFields.java
// {Args: typeinfo.Derived}
/****************** Exercise 9 ******************
 * Modify the previous exercise so that it uses
 * Class.getDeclaredFields() to also display
 * information about the fields in a class.
 ************************************************/
package typeinfo;
import java.lang.reflect.*;
import java.util.*;
import static net.mindview.util.Print.*;

interface Iface {
  int i = 47;
```

```
  void f();
}

class Base implements Iface {
  String s;
  double d;
  public void f() { System.out.println("Base.f"); }
}

class Composed {
  Base b;
}

class Derived extends Base {
  Composed c;
  String s;
}

public class E09_GetDeclaredFields {
  static Set<Class<?>> alreadyDisplayed =
    new HashSet<Class<?>>();
  static void printClasses(Class<?> c) {
    // getSuperclass() returns null on Object:
    if(c == null) return;
    print(c.getName());
    Field[] fields = c.getDeclaredFields();
    if(fields.length != 0)
      print("Fields:");
    for(Field fld : fields) {
      print("   " + fld);
    }
    for(Field fld : fields) {
      Class<?> k = fld.getType();
      if(!alreadyDisplayed.contains(k)) {
        printClasses(k);
        alreadyDisplayed.add(k);
      }
    }
    // Produces the interfaces that this class
    // implements:
    for(Class<?> k : c.getInterfaces()) {
      print("Interface: " + k.getName());
      printClasses(k.getSuperclass());
    }
    printClasses(c.getSuperclass());
  }
  public static void main(String args[]) throws Exception {
```

```
    for(int i = 0; i < args.length; i++) {
      print("Displaying " + args[i]);
      printClasses(Class.forName(args[i]));
      if(i < args.length - 1)
        System.out.println("==================");
    }
  }
} /* Output:
Displaying typeinfo.Derived
typeinfo.Derived
Fields:
   typeinfo.Composed typeinfo.Derived.c
   java.lang.String typeinfo.Derived.s
typeinfo.Composed
Fields:
   typeinfo.Base typeinfo.Composed.b
typeinfo.Base
Fields:
   java.lang.String typeinfo.Base.s
   double typeinfo.Base.d
java.lang.String
Fields:
   private final char[] java.lang.String.value
   private final int java.lang.String.offset
   private final int java.lang.String.count
   private int java.lang.String.hash
   private static final long
java.lang.String.serialVersionUID
   private static final java.io.ObjectStreamField[]
java.lang.String.serialPersistentFields
   public static final java.util.Comparator
java.lang.String.CASE_INSENSITIVE_ORDER
[C
Interface: java.lang.Cloneable
Interface: java.io.Serializable
java.lang.Object
int
long
[Ljava.io.ObjectStreamField;
Interface: java.lang.Cloneable
Interface: java.io.Serializable
java.lang.Object
java.util.Comparator
Interface: java.io.Serializable
Interface: java.lang.Comparable
Interface: java.lang.CharSequence
java.lang.Object
```

```
double
Interface: typeinfo.Iface
java.lang.Object
java.lang.Object
typeinfo.Base
Fields:
   java.lang.String typeinfo.Base.s
   double typeinfo.Base.d
Interface: typeinfo.Iface
java.lang.Object
*///:~
```

The program uses an interesting class hierarchy of interfaces and composition of complex types.

Here, a **HashSet** keeps the output tidy, displaying fields only once. Exercise 10 may help you understand the output.

Exercise 10

```
//: typeinfo/E10_PrimitiveOrTrue.java
/****************** Exercise 10 *****************
 * Write a program to determine whether an array
 * of char is a primitive type or a true object.
 ************************************************/
package typeinfo;
import static net.mindview.util.Print.*;

public class E10_PrimitiveOrTrue {
  public static void main(String args[]) {
    char[] ac = "Hello, World!".toCharArray();
    print("ac.getClass() = " + ac.getClass());
    print("ac.getClass().getSuperclass() = "
      + ac.getClass().getSuperclass());
    char c = 'c';
    //! c.getClass(); // Can't do it, primitives
                      // are not true objects.
    int[] ia = new int[3];
    print("ia.getClass() = " + ia.getClass());
    long[] la = new long[3];
    print("la.getClass() = " + la.getClass());
    double[] da = new double[3];
    print("da.getClass() = " + da.getClass());
    String[] sa = new String[3];
    print("sa.getClass() = " + sa.getClass());
    E10_PrimitiveOrTrue[] pot =
```

```
      new E10_PrimitiveOrTrue[3];
    print("pot.getClass() = " + pot.getClass());
    // Multi-dimensional arrays:
    int[][][] threed = new int[3][][];
    print("threed.getClass() = " + threed.getClass());
  }
} /* Output:
ac.getClass() = class [C
ac.getClass().getSuperclass() = class java.lang.Object
ia.getClass() = class [I
la.getClass() = class [J
da.getClass() = class [D
sa.getClass() = class [Ljava.lang.String;
pot.getClass() = class [Ltypeinfo.E10_PrimitiveOrTrue;
threed.getClass() = class [[[I
*///:~
```

We create an array of **char** by defining a string and calling **toCharArray( )**. Note that you can call **getClass( )** and **getSuperclass( )** for **ac** because it is an object of a true class, but if you try to call **getClass( )** for a primitive like **char c**, the compiler complains. Not all elements of Java are objects (remember this example when you hear that Java is a "pure" OO language).

We continue looking at other arrays to examine the outputs when you print their class names.

All array class names begin with a '**[**' followed by a one-character code for primitive types, and an **L** followed by the type of element contained in the array for arrays of objects.

Multi-dimensional arrays add a '**[**' for each dimension. Learn more about such details in *Inside the Java Virtual Machine, 2nd Edition*, by Bill Venners (McGraw-Hill 2000).

Exercise 11

Here we show only changes to the existing **typeinfo.pets** library and a new version of **PetCount.java**. Other variants needed only the proper **import** directive to work with the newly-added **Gerbil** type.

```
//: typeinfo/pets1/Gerbil.java
/****************** Exercise 11 *****************
 * Add Gerbil to the typeinfo.pets library and
 * modify all the examples in this chapter to adapt
 * to this new class.
 ***********************************************/
```

```
package typeinfo.pets1;

public class Gerbil extends typeinfo.pets.Rodent {
  public Gerbil(String name) { super(name); }
  public Gerbil() {}
} ///:~
```

```
//: typeinfo/pets1/ForNameCreator.java
package typeinfo.pets1;
import java.util.*;
import typeinfo.pets.*;

@SuppressWarnings("unchecked")
public class ForNameCreator extends PetCreator {
  private static List<Class<? extends Pet>> types =
    new ArrayList<Class<? extends Pet>>();
  // Types that you want to be randomly created:
  private static String[] typeNames = {
    "typeinfo.pets.Mutt",
    "typeinfo.pets.Pug",
    "typeinfo.pets.EgyptianMau",
    "typeinfo.pets.Manx",
    "typeinfo.pets.Cymric",
    "typeinfo.pets.Rat",
    "typeinfo.pets.Mouse",
    "typeinfo.pets.Hamster",
    "typeinfo.pets1.Gerbil"
  };
  static {
    try {
      for(String name : typeNames)
        types.add(
          (Class<? extends Pet>)Class.forName(name));
    } catch(ClassNotFoundException e) {
      throw new RuntimeException(e);
    }
  }
  public List<Class<? extends Pet>> types() {return types;}
} ///:~
```

```
//: typeinfo/pets1/LiteralPetCreator1.java
// Using class literals.
package typeinfo.pets1;
import java.util.*;
import typeinfo.pets.*;

@SuppressWarnings("unchecked")
```

```
public class LiteralPetCreator1 extends PetCreator {
  // No try block needed.
  public static final List<Class<? extends Pet>> allTypes =
    Collections.unmodifiableList(Arrays.asList(
      Pet.class, Dog.class, Cat.class,  Rodent.class,
      Mutt.class, Pug.class, EgyptianMau.class, Manx.class,
      Cymric.class, Rat.class, Mouse.class,Hamster.class,
      Gerbil.class));
  // Types for random creation:
  private static final List<Class<? extends Pet>> types =
    allTypes.subList(allTypes.indexOf(Mutt.class),
      allTypes.size());
  public List<Class<? extends Pet>> types() {
    return types;
  }
  public static void main(String[] args) {
    System.out.println(types);
  }
} /* Output:
[class typeinfo.pets.Mutt, class typeinfo.pets.Pug, class
typeinfo.pets.EgyptianMau, class typeinfo.pets.Manx, class
typeinfo.pets.Cymric, class typeinfo.pets.Rat, class
typeinfo.pets.Mouse, class typeinfo.pets.Hamster, class
typeinfo.pets1.Gerbil]
*///:~
```

```
//: typeinfo/PetCount.java
// Using instanceof.
package typeinfo;
import java.util.*;
import static net.mindview.util.Print.*;
import typeinfo.pets.*;
import typeinfo.pets1.Gerbil;
import typeinfo.pets1.ForNameCreator;

public class PetCount {
  static class PetCounter extends HashMap<String, Integer>{
    public void count(String type) {
      Integer quantity = get(type);
      if(quantity == null)
        put(type, 1);
      else
        put(type, quantity + 1);
    }
  }
  public static void
  countPets(PetCreator creator) {
```

```
    PetCounter counter= new PetCounter();
    for(Pet pet : creator.createArray(20)) {
      // List each individual pet:
      printnb(pet.getClass().getSimpleName() + " ");
      if(pet instanceof Pet)
        counter.count("Pet");
      if(pet instanceof Dog)
        counter.count("Dog");
      if(pet instanceof Mutt)
        counter.count("Mutt");
      if(pet instanceof Pug)
        counter.count("Pug");
      if(pet instanceof Cat)
        counter.count("Cat");
      if(pet instanceof Manx)
        counter.count("EgyptianMau");
      if(pet instanceof Manx)
        counter.count("Manx");
      if(pet instanceof Manx)
        counter.count("Cymric");
      if(pet instanceof Rodent)
        counter.count("Rodent");
      if(pet instanceof Rat)
        counter.count("Rat");
      if(pet instanceof Mouse)
        counter.count("Mouse");
      if(pet instanceof Hamster)
        counter.count("Hamster");
      if(pet instanceof Gerbil)
        counter.count("Gerbil");
    }
    // Show the counts:
    print();
    print(counter);
  }
  public static void main(String[] args) {
    countPets(new ForNameCreator());
  }
} /* Output:
EgyptianMau Gerbil Cymric EgyptianMau Cymric EgyptianMau Pug
Rat Mutt Cymric Manx Manx Manx Cymric EgyptianMau Pug
Hamster Cymric Gerbil Pug
{Rat=1, Cymric=8, Gerbil=2, Cat=12, Pet=20, Dog=4, Manx=8,
EgyptianMau=8, Pug=3, Rodent=4, Hamster=1, Mutt=1}
*///:~
```

See the *Registered Factories* section of *TIJ4* for a discussion of the problems with this method of generating objects of the **Pets** hierarchy.

Exercise 12

```
//: typeinfo/E12_CoffeeCount.java
/****************** Exercise 12 *****************
 * Use TypeCounter with the CoffeeGenerator.java
 * class in the Generics chapter.
 ************************************************/
package typeinfo;
import java.util.*;
import generics.coffee.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E12_CoffeeCount {
  public static void main(String[] args) {
    TypeCounter counter = new TypeCounter(Coffee.class);
    for(Iterator<Coffee> it =
        new CoffeeGenerator(20).iterator(); it.hasNext();) {
      Coffee coffee = it.next();
      printnb(coffee.getClass().getSimpleName() + " ");
      counter.count(coffee);
    }
    print();
    print(counter);
  }
} /* Output:
Americano Latte Americano Mocha Mocha Breve Americano Latte
Cappuccino Cappuccino Americano Americano Mocha Breve Breve
Americano Americano Mocha Latte Americano
{Latte=3, Americano=8, Cappuccino=2, Coffee=20, Breve=3,
Mocha=4}
*///:~
```

Exercise 13

```
//: typeinfo/E13_PartCount.java
/****************** Exercise 13 *****************
 * Use TypeCounter with the RegisteredFactories.java
 * example in this chapter.
 ************************************************/
package typeinfo;
```

```
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E13_PartCount {
  public static void main(String[] args) {
    TypeCounter counter = new TypeCounter(Part.class);
    Part part;
    for(int i = 0; i < 20; i++) {
      part = Part.createRandom();
      printnb(part.getClass().getSimpleName() + " ");
      counter.count(part);
    }
    print();
    print(counter);
  }
} /* Output:
GeneratorBelt CabinAirFilter GeneratorBelt AirFilter
PowerSteeringBelt CabinAirFilter FuelFilter
PowerSteeringBelt PowerSteeringBelt FuelFilter
CabinAirFilter PowerSteeringBelt FanBelt AirFilter OilFilter
OilFilter AirFilter PowerSteeringBelt FuelFilter
CabinAirFilter
{OilFilter=2, GeneratorBelt=2, Part=20, Belt=8,
FuelFilter=3, PowerSteeringBelt=5, Filter=12,
CabinAirFilter=4, FanBelt=1, AirFilter=3}
*///:~
```

These two exercises show that writing reusable code is extremely useful; we see the versatility of the **TypeCounter** class in different scenarios.

Exercise 14

```
//: typeinfo/E14_RegisteredFactories2.java
/****************** Exercise 14 *****************
 * A constructor is a kind of factory method. Modify
 * RegisteredFactories.java so that instead of using
 * an explicit factory, the class object is stored
 * in the List, and newInstance() is used to create
 * each object.
 ***********************************************/
package typeinfo;
import java.util.*;

@SuppressWarnings("unchecked")
class Part2 {
  public String toString() {
```

```
    return getClass().getSimpleName();
  }
  static List<Class<? extends Part2>> partClasses =
    Arrays.asList(FuelFilter2.class, AirFilter2.class,
      CabinAirFilter2.class, OilFilter2.class,
      FanBelt2.class, PowerSteeringBelt2.class,
      GeneratorBelt2.class);
  private static Random rand = new Random(47);
  public static Part2 createRandom() {
    int n = rand.nextInt(partClasses.size());
    try {
      return partClasses.get(n).newInstance();
    } catch(InstantiationException e) {
      throw new RuntimeException(e);
    } catch(IllegalAccessException e) {
      throw new RuntimeException(e);
    }
  }
}

class Filter2 extends Part2 {}

class FuelFilter2 extends Filter2 {}

class AirFilter2 extends Filter2 {}

class CabinAirFilter2 extends Filter2 {}

class OilFilter2 extends Filter2 {}

class Belt2 extends Part2 {}

class FanBelt2 extends Belt2 {}

class GeneratorBelt2 extends Belt2 {}

class PowerSteeringBelt2 extends Belt2 {}

public class E14_RegisteredFactories2 {
  public static void main(String[] args) {
    for(int i = 0; i < 10; i++)
      System.out.println(Part2.createRandom());
  }
} /* Output:
GeneratorBelt2
CabinAirFilter2
GeneratorBelt2
```

```
AirFilter2
PowerSteeringBelt2
CabinAirFilter2
FuelFilter2
PowerSteeringBelt2
PowerSteeringBelt2
FuelFilter2
*///:~
```

This solution closely resembles the *Prototype* design pattern.

Exercise 15

We show only the noteworthy changes to **typeinfo.pets**. Clients who access the library exclusively through a *Façade* will transparently switch to the new **PetCreator** (**PetCount4.java** in our case). This shows again that systems built around mature patterns are more easily understood and maintained than ad hoc solutions. Moreover, a properly designed, dependable framework or library spares clients the annoyance of internal changes.

The new library is **typeinfo.pets2**.

In this solution, we'll use the *Façade* itself as a central place to incorporate the registry, since it represents an entry point for the library. Another candidate is the base class of the hierarchy of interest, which we used in *TIJ4*.

```
//: typeinfo/pets2/Cymric.java
/****************** Exercise 15 *****************
 * Implement a new PetCreator using Registered
 * Factories, and modify the Pets Façade so that
 * it uses this one instead of the other two.
 * Ensure that the rest of the examples that use
 * Pets.java still work correctly.
 ***********************************************/
package typeinfo.pets2;

public class Cymric extends Manx {
  public static class Factory
  implements typeinfo.factory.Factory<Cymric> {
    public Cymric create() { return new Cymric(); }
  }
} ///:~
```

```
//: typeinfo/pets2/EgyptianMau.java
package typeinfo.pets2;
import typeinfo.pets.Cat;
```

```
public class EgyptianMau extends Cat {
  public static class Factory
  implements typeinfo.factory.Factory<EgyptianMau> {
    public EgyptianMau create() {
      return new EgyptianMau();
    }
  }
} ///:~
```

```
//: typeinfo/pets2/Hamster.java
package typeinfo.pets2;
import typeinfo.pets.Rodent;

public class Hamster extends Rodent {
  public static class Factory
  implements typeinfo.factory.Factory<Hamster> {
    public Hamster create() { return new Hamster(); }
  }
} ///:~
```

```
//: typeinfo/pets2/Manx.java
package typeinfo.pets2;
import typeinfo.pets.Cat;

public class Manx extends Cat {
  public static class Factory
  implements typeinfo.factory.Factory<Manx> {
    public Manx create() { return new Manx(); }
  }
} ///:~
```

```
//: typeinfo/pets2/Mouse.java
package typeinfo.pets2;
import typeinfo.pets.Rodent;

public class Mouse extends Rodent {
  public static class Factory
  implements typeinfo.factory.Factory<Mouse> {
    public Mouse create() { return new Mouse(); }
  }
} ///:~
```

```
//: typeinfo/pets2/Mutt.java
package typeinfo.pets2;
import typeinfo.pets.Dog;

public class Mutt extends Dog {
```

```
  public static class Factory
  implements typeinfo.factory.Factory<Mutt> {
    public Mutt create() { return new Mutt(); }
  }
} ///:~
```

```
//: typeinfo/pets2/Pug.java
package typeinfo.pets2;
import typeinfo.pets.Dog;

public class Pug extends Dog {
  public static class Factory
  implements typeinfo.factory.Factory<Pug> {
    public Pug create() { return new Pug(); }
  }
} ///:~
```

```
//: typeinfo/pets2/Rat.java
package typeinfo.pets2;
import typeinfo.pets.Rodent;

public class Rat extends Rodent {
  public static class Factory
  implements typeinfo.factory.Factory<Rat> {
    public Rat create() { return new Rat(); }
  }
} ///:~
```

```
//: typeinfo/pets2/Pets.java
// Facade to produce a default PetCreator.
package typeinfo.pets2;
import java.util.*;
import typeinfo.factory.Factory;
import typeinfo.pets.PetCreator;
import typeinfo.pets.Pet;

@SuppressWarnings("unchecked")
public class Pets {
  private static class RFPetCreator extends PetCreator {
    static List<Factory<? extends Pet>> petFactories =
      Arrays.asList(new Mutt.Factory(), new Pug.Factory(),
        new EgyptianMau.Factory(), new Manx.Factory(),
         new Cymric.Factory(), new Rat.Factory(),
         new Mouse.Factory(), new Hamster.Factory());
    @Override public List<Class<? extends Pet>> types() {
      return null;  // Dummy value, this method is not used!
    }
    @Override public Pet randomPet() { // Make 1 random Pet
```

```
      int n = rand.nextInt(petFactories.size());
      return petFactories.get(n).create();
    }
  }
  private static Random rand = new Random(47);
  public static final PetCreator creator =
    new RFPetCreator();
  public static Pet randomPet() {
    return creator.randomPet();
  }
  public static Pet[] createArray(int size) {
    return creator.createArray(size);
  }
  public static ArrayList<Pet> arrayList(int size) {
    return creator.arrayList(size);
  }
} ///:~
```

In the rest of the examples, we change only the references to the library; the referred package is **typeinfo.pets2** instead of **typeinfo.pets**.

Exercise 16

```
//: typeinfo/E16_CoffeeGenerator.java
/****************** Exercise 16 *****************
 * Modify the Coffee hierarchy in the Generics
 * chapter to use Registered Factories.
 ***********************************************/
package typeinfo;
import java.util.*;
import net.mindview.util.*;
import typeinfo.coffee2.*;

public class E16_CoffeeGenerator
implements Generator<Coffee>, Iterable<Coffee> {
  public E16_CoffeeGenerator() {}
  private int size = 0;
  public E16_CoffeeGenerator(int sz) { size = sz; }
  public Coffee next() { return Coffee.createRandom(); }
  class CoffeeIterator implements Iterator<Coffee> {
    int count = size;
    public boolean hasNext() { return count > 0; }
    public Coffee next() {
      count--;
      return E16_CoffeeGenerator.this.next();
    }
```

```
    public void remove() {} // Not implemented
  };
  public Iterator<Coffee> iterator() {
    return new CoffeeIterator();
  }
  public static void main(String[] args) {
    for(Coffee c : new E16_CoffeeGenerator(10))
      System.out.println(c);
  }
} /* Output:
Mocha 0
Americano 1
Mocha 2
Breve 3
Breve 4
Cappucino 5
Mocha 6
Americano 7
Latte 8
Latte 9
*///:~
```

```
//: typeinfo/coffee2/Coffee.java
package typeinfo.coffee2;
import java.util.*;
import typeinfo.factory.Factory;

@SuppressWarnings("unchecked")
public class Coffee {
  private static int counter = 0;
  private int id = counter++;
  private static
    List<Factory<? extends Coffee>> coffeeFactories =
      Arrays.asList(new Americano.Factory(),
        new Breve.Factory(), new Latte.Factory(),
        new Mocha.Factory(), new Cappucino.Factory());
  private static Random rand = new Random(47);
  public static Coffee createRandom() {
    int n = rand.nextInt(coffeeFactories.size());
    return coffeeFactories.get(n).create();
  }
  public String toString() {
    return getClass().getSimpleName() + " " + id;
  }
} ///:~
```

```
//: typeinfo/coffee2/Americano.java
```

```
package typeinfo.coffee2;

public class Americano extends Coffee {
  public static class Factory
  implements typeinfo.factory.Factory<Americano> {
    public Americano create() { return new Americano(); }
  }
} ///:~
```

```
//: typeinfo/coffee2/Breve.java
package typeinfo.coffee2;

public class Breve extends Coffee {
  public static class Factory
  implements typeinfo.factory.Factory<Breve> {
    public Breve create() { return new Breve(); }
  }
} ///:~
```

```
//: typeinfo/coffee2/Cappucino.java
package typeinfo.coffee2;

public class Cappucino extends Coffee {
  public static class Factory
  implements typeinfo.factory.Factory<Cappucino> {
    public Cappucino create() { return new Cappucino(); }
  }
} ///:~
```

```
//: typeinfo/coffee2/Latte.java
package typeinfo.coffee2;

public class Latte extends Coffee {
  public static class Factory
  implements typeinfo.factory.Factory<Latte> {
    public Latte create() { return new Latte(); }
  }
} ///:~
```

```
//: typeinfo/coffee2/Mocha.java
package typeinfo.coffee2;

public class Mocha extends Coffee {
  public static class Factory
  implements typeinfo.factory.Factory<Mocha> {
    public Mocha create() { return new Mocha(); }
  }
} ///:~
```

Exercise 17

```
//: typeinfo/E17_ShowMethods2.java
// {Args: typeinfo.E17_ShowMethods2}
/********************* Exercise 17 *********************
 * Modify the regular expression in ShowMethods.java to
 * also strip off the keywords native and final.
 * (Hint: Use the OR operator '|'.)
 *******************************************************/
package typeinfo;
import java.lang.reflect.*;
import java.util.regex.*;
import static net.mindview.util.Print.*;

public class E17_ShowMethods2 {
  private static String usage =
    "usage:\n" +
    "E17_ShowMethods2 qualified.class.name\n" +
    "To show all methods in class or:\n" +
    "E17_ShowMethods2 qualified.class.name word\n" +
    "To search for methods involving 'word'";
  private static Pattern p =
    Pattern.compile("\\w+\\.|native\\s|final\\s");
  public static void main(String[] args) {
    if(args.length < 1) {
      print(usage);
      System.exit(0);
    }
    int lines = 0;
    try {
      Class<?> c = Class.forName(args[0]);
      Method[] methods = c.getMethods();
      Constructor<?>[] ctors = c.getConstructors();
      if(args.length == 1) {
        for(Method method : methods)
          print(
            p.matcher(method.toString()).replaceAll(""));
        for(Constructor<?> ctor : ctors)
          print(p.matcher(ctor.toString()).replaceAll(""));
        lines = methods.length + ctors.length;
      } else {
        for(Method method : methods)
          if(method.toString().indexOf(args[1]) != -1) {
            print(
              p.matcher(method.toString()).replaceAll(""));
            lines++;
```

```
          }
        for(Constructor<?> ctor : ctors)
          if(ctor.toString().indexOf(args[1]) != -1) {
            print(p.matcher(
              ctor.toString()).replaceAll(""));
            lines++;
          }
      }
    } catch(ClassNotFoundException e) {
      print("No such class: " + e);
    }
  }
} /* Output:
public static void main(String[])
public int hashCode()
public Class getClass()
public void wait(long,int) throws InterruptedException
public void wait() throws InterruptedException
public void wait(long) throws InterruptedException
public boolean equals(Object)
public String toString()
public void notify()
public void notifyAll()
public E17_ShowMethods2()
*///:~
```

Exercise 18

```
//: typeinfo/E18_ShowMethods3.java
// {Args: typeinfo.E18_ShowMethods3}
/********************** Exercise 18 **********************
 * Make ShowMethods a non-public class and verify that
 * the synthesized default constructor no longer appears in
 * the output.
 *********************************************************/
package typeinfo;
import java.lang.reflect.*;
import java.util.regex.*;
import static net.mindview.util.Print.*;

class E18_ShowMethods3 {
  private static String usage =
    "usage:\n" +
    "E18_ShowMethods3 qualified.class.name\n" +
    "To show all methods in class or:\n" +
    "E18_ShowMethods3 qualified.class.name word\n" +
```

```
    "To search for methods involving 'word'";
  private static Pattern p = Pattern.compile("\\w+\\.");
  public static void main(String[] args) {
    if(args.length < 1) {
      print(usage);
      System.exit(0);
    }
    int lines = 0;
    try {
      Class<?> c = Class.forName(args[0]);
      Method[] methods = c.getMethods();
      Constructor<?>[] ctors = c.getConstructors();
      if(args.length == 1) {
        for(Method method : methods)
          print(
            p.matcher(method.toString()).replaceAll(""));
        for(Constructor<?> ctor : ctors)
          print(p.matcher(ctor.toString()).replaceAll(""));
        lines = methods.length + ctors.length;
      } else {
        for(Method method : methods)
          if(method.toString().indexOf(args[1]) != -1) {
            print(
              p.matcher(method.toString()).replaceAll(""));
            lines++;
          }
        for(Constructor<?> ctor : ctors)
          if(ctor.toString().indexOf(args[1]) != -1) {
            print(p.matcher(
              ctor.toString()).replaceAll(""));
            lines++;
          }
      }
    } catch(ClassNotFoundException e) {
      print("No such class: " + e);
    }
  }
} /* Output:
public static void main(String[])
public native int hashCode()
public final native Class getClass()
public final void wait(long,int) throws InterruptedException
public final void wait() throws InterruptedException
public final native void wait(long) throws
InterruptedException
public boolean equals(Object)
public String toString()
```

```
public final native void notify()
public final native void notifyAll()
*///:~
```

Exercise 19

```
//: typeinfo/E19_ReflectionToyCreation.java
/****************** Exercise 19 *****************
 * In ToyTest.java, use reflection to create a
 * Toy object using the nondefault constructor.
 ************************************************/
package typeinfo;
import java.lang.reflect.*;

class SuperToy extends FancierToy {
  int IQ;
  public SuperToy(int intelligence) { IQ = intelligence; }
  public String toString() {
    return "I'm a SuperToy. I'm smarter than you.";
  }
}

public class E19_ReflectionToyCreation {
  public static Toy makeToy(String toyName, int IQ) {
    try {
      Class<?> tClass = Class.forName(toyName);
      for(Constructor<?> ctor : tClass.getConstructors()) {
        // Look for a constructor with a single parameter:
        Class<?>[] params = ctor.getParameterTypes();
        if(params.length == 1)
          if(params[0] == int.class)
            return (Toy)ctor.newInstance(
              new Object[]{ Integer.valueOf(IQ) } );
      }
    } catch(Exception e) {
      throw new RuntimeException(e);
    }
    return null;
  }
  public static void main(String args[]) {
    System.out.println(makeToy("typeinfo.SuperToy", 150));
  }
} /* Output:
I'm a SuperToy. I'm smarter than you.
*///:~
```

This approach is rather limited. If you want to get fancier, ask the user (via the console) what values to use once you get the argument types for the constructor.

Exercise 20

```
//: typeinfo/E20_ClassDump.java
// {Args: java.lang.String typeinfo.SuperToy}
/****************** Exercise 20 *****************
 * Look up the interface for java.lang.Class in
 * the JDK documentation from java.sun.com.
 * Write a program that takes the name of a class
 * as a command-line argument, then uses the
 * Class methods to dump all the information
 * available for that class. Test your program
 * with a standard library class and a class you
 * create.
 ***********************************************/
package typeinfo;
// The solution is a much-modified version of
// Showmethods.java.
import static net.mindview.util.Print.*;

public class E20_ClassDump {
  public static void display(String msg, Object[] vals) {
    print(msg);
    for(Object val : vals)
      print("   " + val);
  }
  public static void
  classInfo(Class<?> c) throws Exception {
    print("c.getName(): " + c.getName());
    print("c.getPackage(): " + c.getPackage());
    print("c.getSuperclass(): " + c.getSuperclass());
    // This produces all the classes declared as members:
    display("c.getDeclaredClasses()",
      c.getDeclaredClasses());
    display("c.getClasses()", c.getClasses());
    display("c.getInterfaces()", c.getInterfaces());
    // The "Declared" version includes all
    // versions, not just public:
    display("c.getDeclaredMethods()",
      c.getDeclaredMethods());
    display("c.getMethods()", c.getMethods());
    display("c.getDeclaredConstructors()",
      c.getDeclaredConstructors());
```

```
    display("c.getConstructors()", c.getConstructors());
    display("c.getDeclaredFields()",
      c.getDeclaredFields());
    display("c.getFields()", c.getFields());
    if(c.getSuperclass() != null) {
      print("Base class: --------");
      classInfo(c.getSuperclass());
    }
    print("End of " + c.getName());
  }
  public static void main(String[] args) throws Exception {
    for(String arg : args)
      classInfo(Class.forName(arg));
  }
} /* (Execute to see output) *///:~
```

This is a good sample of the methods in **Class** that give reflection information. Also, it recursively goes up the inheritance hierarchy.

Exercise 21

```
//: typeinfo/E21_SimpleProxyDemo.java
/****************** Exercise 21 *****************
 * Modify SimpleProxyDemo.java so it measures
 * method-call times.
 ************************************************/
package typeinfo;
import static net.mindview.util.Print.*;

interface Interface {
  void doSomething();
  void somethingElse(String arg);
}

class RealObject implements Interface {
  public void doSomething() { print("doSomething"); }
  public void somethingElse(String arg) {
    print("somethingElse " + arg);
  }
}

class SimpleProxy implements Interface {
  private Interface proxied;
  public SimpleProxy(Interface proxied) {
    this.proxied = proxied;
  }
```

```
  public void doSomething() {
    print("SimpleProxy doSomething");
    long start = System.nanoTime();
    proxied.doSomething();
    long duration = System.nanoTime() - start;
    print("METHOD-CALL TIME: " + duration);
  }
  public void somethingElse(String arg){
    print("SimpleProxy somethingElse " + arg);
    long start = System.nanoTime();
    proxied.somethingElse(arg);
    long duration = System.nanoTime() - start;
    print("METHOD-CALL TIME: " + duration);
  }
}

public class E21_SimpleProxyDemo {
  public static void consumer(Interface iface) {
    iface.doSomething();
    iface.somethingElse("bonobo");
  }
  public static void main(String[] args) {
    consumer(new RealObject());
    consumer(new SimpleProxy(new RealObject()));
  }
} /* Output: (90% match)
doSomething
somethingElse bonobo
SimpleProxy doSomething
doSomething
METHOD-CALL TIME: 52800
SimpleProxy somethingElse bonobo
somethingElse bonobo
METHOD-CALL TIME: 56711
*///:~
```

As stated in the JDK documentation, **System.nanoTime( )** "*provides nanosecond precision, but not necessarily nanosecond accuracy*" (results vary on different computers). This is the main reason the output cannot be matched with greater than 90% certainty.

Exercise 22

```
//: typeinfo/E22_SimpleDynamicProxyDemo.java
/****************** Exercise 22 *****************
 * Modify SimpleDynamicProxy.java so that it
```

```
 * measures method-call times.
 *********************************************/
package typeinfo;
import java.lang.reflect.*;

class DynamicProxyHandler implements InvocationHandler {
  private Object proxied;
  public DynamicProxyHandler(Object proxied) {
    this.proxied = proxied;
  }
  public Object
  invoke(Object proxy, Method method, Object[] args)
  throws Throwable {
    System.out.println("**** proxy: " + proxy.getClass() +
      ", method: " + method + ", args: " + args);
    if(args != null)
      for(Object arg : args)
        System.out.println("  " + arg);
    long start = System.nanoTime();
    Object ret = method.invoke(proxied, args);
    long duration = System.nanoTime() - start;
    System.out.println("METHOD-CALL TIME: " + duration);
    return ret;
  }
}

class E22_SimpleDynamicProxyDemo {
  public static void consumer(Interface iface) {
    iface.doSomething();
    iface.somethingElse("bonobo");
  }
  public static void main(String[] args) {
    RealObject real = new RealObject();
    consumer(real);
    Interface proxy = (Interface)Proxy.newProxyInstance(
      Interface.class.getClassLoader(),
      new Class<?>[]{ Interface.class },
      new DynamicProxyHandler(real));
    consumer(proxy);
  }
} /* Output: (85% match)
doSomething
somethingElse bonobo
**** proxy: class $Proxy0, method: public abstract void
Interface.doSomething(), args: null
doSomething
METHOD-CALL TIME: 301155
```

```
**** proxy: class $Proxy0, method: public abstract void
Interface.somethingElse(java.lang.String), args:
[Ljava.lang.Object;@42e816
  bonobo
somethingElse bonobo
METHOD-CALL TIME: 84648
*///:~
```

Here, we centralize the time measurement of method calls, previously dispersed among all methods of a proxy.

Exercise 23

```
//: typeinfo/E23_SimpleDynamicProxyDemo2.java
// {ThrowsException}
/****************** Exercise 23 *****************
 * Inside invoke() in SimpleDynamicProxy.java,
 * try to print the proxy argument and explain
 * what happens.
 ************************************************/
package typeinfo;
import java.lang.reflect.*;

class DynamicProxyHandler2 implements InvocationHandler {
  private Object proxied;
  public DynamicProxyHandler2(Object proxied) {
    this.proxied = proxied;
  }
  public Object
  invoke(Object proxy, Method method, Object[] args)
  throws Throwable {
    System.out.println("proxy: " + proxy);
    return method.invoke(proxied, args);
  }
}

class E23_SimpleDynamicProxyDemo2 {
  public static void consumer(Interface iface) {
    iface.doSomething();
    iface.somethingElse("bonobo");
  }
  public static void main(String[] args) {
    RealObject real = new RealObject();
    consumer(real);
    Interface proxy = (Interface)Proxy.newProxyInstance(
      Interface.class.getClassLoader(),
```

```
      new Class<?>[]{ Interface.class },
      new DynamicProxyHandler2(real));
    consumer(proxy);
  }
} ///:~
```

This program enters an infinite loop inside **invoke( )** and eventually crashes (with a **java.lang.StackOverflowError**). We call the **toString( )** method to print the **proxy** object; however, we redirect calls through the interface through the proxy, as well as calls of the methods inherited from **Object**.

Exercise 24

```
//: typeinfo/E24_RegisteredFactories3.java
/****************** Exercise 24 *****************
 * Add Null Objects to RegisteredFactories.java.
 ************************************************/
package typeinfo;
import java.lang.reflect.*;
import java.util.*;
import net.mindview.util.Null;
import typeinfo.factory.*;

class NullPartProxyHandler implements InvocationHandler {
  private String nullName;
  private IPart proxied = new NPart();
  NullPartProxyHandler(Class<? extends IPart> type) {
    nullName = type.getSimpleName() + ": [Null Part]";
  }
  private class NPart implements Null, IPart {
    public String toString() { return nullName; }
  }
  public Object
  invoke(Object proxy, Method method, Object[] args)
  throws Throwable {
    return method.invoke(proxied, args);
  }
}

interface IPart {}

class Part3 implements IPart {
  public String toString() {
    return getClass().getSimpleName();
  }
  public static IPart
```

```
  newNull(Class<? extends IPart> type) {
    return (IPart)Proxy.newProxyInstance(
      IPart.class.getClassLoader(),
      new Class<?>[]{ Null.class, IPart.class },
      new NullPartProxyHandler(type));
  }
  static List<Factory<IPart>> partFactories =
    new ArrayList<Factory<IPart>>();
  static {
    partFactories.add(new FuelFilter3.Factory());
    partFactories.add(new AirFilter3.Factory());
    partFactories.add(new CabinAirFilter3.Factory());
    partFactories.add(new OilFilter3.Factory());
    partFactories.add(new FanBelt3.Factory());
    partFactories.add(new PowerSteeringBelt3.Factory());
    partFactories.add(new GeneratorBelt3.Factory());
  }
  private static Random rand = new Random(47);
  public static IPart createRandom() {
    int n = rand.nextInt(partFactories.size());
    return partFactories.get(n).create();
  }
}

class Filter3 extends Part3 {}

class FuelFilter3 extends Filter3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() { return new FuelFilter3(); }
  }
}

class AirFilter3 extends Filter3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() { return new AirFilter3(); }
  }
}

class CabinAirFilter3 extends Filter3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() {
      return new CabinAirFilter3();
    }
  }
```

```
}

class OilFilter3 extends Filter3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() { return new OilFilter3(); }
  }
}

class Belt3 extends Part3 {}

class FanBelt3 extends Belt3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() { return new FanBelt3(); }
  }
}

class GeneratorBelt3 extends Belt3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() {
      return new GeneratorBelt3();
    }
  }
}

class PowerSteeringBelt3 extends Belt3 {
  public static class Factory
  implements typeinfo.factory.Factory<IPart> {
    public IPart create() {
      return new PowerSteeringBelt3();
    }
  }
}

public class E24_RegisteredFactories3 {
  public static void main(String[] args) {
    for(int i = 0; i < 10; i++) {
      IPart part = Part3.createRandom();
      // Real object
      System.out.println(part);
      // Null companion
      System.out.println(Part3.newNull(part.getClass()));
    }
  }
} /* Output:
```

```
GeneratorBelt3
GeneratorBelt3: [Null Part]
CabinAirFilter3
CabinAirFilter3: [Null Part]
GeneratorBelt3
GeneratorBelt3: [Null Part]
AirFilter3
AirFilter3: [Null Part]
PowerSteeringBelt3
PowerSteeringBelt3: [Null Part]
CabinAirFilter3
CabinAirFilter3: [Null Part]
FuelFilter3
FuelFilter3: [Null Part]
PowerSteeringBelt3
PowerSteeringBelt3: [Null Part]
PowerSteeringBelt3
PowerSteeringBelt3: [Null Part]
FuelFilter3
FuelFilter3: [Null Part]
*///:~
```

Following *TIJ4*'s technique, we use a *Dynamic Proxy* to create *Null Objects* for different parts. The **IPart** interface is the most important innovation to **RegisteredFactories.java**. Notice that the rest of the class hierarchy is unchanged.

Exercise 25

```
//: typeinfo/E25_HiddenMethodCalls.java
/****************** Exercise 25 *****************
 * Create a class containing private, protected and
 * package access methods. Write code to access these
 * methods from outside of the class's package.
 ***********************************************/
package typeinfo;
import java.lang.reflect.*;
import typeinfo.classa.*;

public class E25_HiddenMethodCalls {
  static void callHiddenMethod(Object a, String methodName)
  throws Exception {
    Method g = a.getClass().getDeclaredMethod(methodName);
    g.setAccessible(true);
    g.invoke(a);
  }
```

```
  public static void main(String[] args) throws Exception {
    class B extends A {
      protected void b() { super.b(); }
    }
    A objA = new A();
    //! objA.a();  Compile time error
    //! objA.b();  Compile time error
    //! objA.c();  Compile time error
    callHiddenMethod(objA, "a");
    callHiddenMethod(objA, "b");
    callHiddenMethod(objA, "c");
    B objB = new B();
    objB.b();
  }
} /* Output:
A.a()
A.b()
A.c()
A.b()
*///:~
```

```
//: typeinfo/classa/A.java
package typeinfo.classa;

public class A {
  private void a() { System.out.println("A.a()"); }
  protected void b() { System.out.println("A.b()"); }
  void c() { System.out.println("A.c()"); }
} ///:~
```

Exercise 26

```
//: typeinfo/E26_ClearSpitValve.java
/****************** Exercise 26 *****************
 * Implement clearSpitValve() as described in the
 * summary.
 ***********************************************/
package typeinfo;
// The summary reads:
/*
One option is to put a clearSpitValve() method in
the base class Instrument, but this is confusing
because it implies that Percussion, Stringed and
Electronic instruments also have spit valves. RTTI
provides a much more reasonable solution because you
can place the method in the specific class where it's
```

```
appropriate (Wind, in this case). At the same time,
you may discover that there's a more sensible
solution—here, a prepareInstrument() method in the
base class. However, you might not see such a solution
when you're first solving the problem and could
mistakenly assume that you must use RTTI.
*/
// We'll use the last-defined version of the
// instrument hierarchy:
import static net.mindview.util.Print.*;

interface Instrument {
  void play();
  String what();
  void adjust();
  void prepareInstrument();
}

class Wind implements Instrument {
  public void play() { print("Wind.play()"); }
  public String what() { return "Wind"; }
  public void adjust() {}
  public void clearSpitValve() {
    print("Wind.clearSpitValve");
  }
  public void prepareInstrument() {
    clearSpitValve();
  }
}

class Percussion implements Instrument {
  public void play() { print("Percussion.play()"); }
  public String what() { return "Percussion"; }
  public void adjust() {}
  public void prepareInstrument() {
    print("Percussion.prepareInstrument");
  }
}

class Stringed implements Instrument {
  public void play() { print("Stringed.play()"); }
  public String what() { return "Stringed"; }
  public void adjust() {}
  public void prepareInstrument() {
    print("Stringed.prepareInstrument");
  }
}
```

```
class Brass extends Wind {
  public void play() { print("Brass.play()"); }
  public void adjust() { print("Brass.adjust()"); }
  public void clearSpitValve() {
    print("Brass.clearSpitValve");
  }
}

class Woodwind extends Wind {
  public void play() { print("Woodwind.play()"); }
  public String what() { return "Woodwind"; }
  public void clearSpitValve() {
    print("Woodwind.clearSpitValve");
  }
}

class Music5 {
  static void tune(Instrument i) {
    // ...
    i.play();
  }
  static void tuneAll(Instrument[] e) {
    for(Instrument instrument : e)
      tune(instrument);
  }
  static void prepareAll(Instrument[] e) {
    for(Instrument instrument : e)
      instrument.prepareInstrument();
  }
}

public class E26_ClearSpitValve {
  public static void main(String[] args) {
    Instrument[] orchestra = {
      new Wind(), new Percussion(),
      new Stringed(), new Brass(),
      new Woodwind(),
    };
    Music5.prepareAll(orchestra);
    Music5.tuneAll(orchestra);
  }
} /* Output:
Wind.clearSpitValve
Percussion.prepareInstrument
Stringed.prepareInstrument
Brass.clearSpitValve
```

```
Woodwind.clearSpitValve
Wind.play()
Percussion.play()
Stringed.play()
Brass.play()
Woodwind.play()
*///:~
```

Generics

Exercise 1

```
//: generics/E01_PetsHolder.java
/****************** Exercise 1 *****************
 * Use Holder3 with the typeinfo.pets library to
 * show that a Holder3 that is specified to hold
 * a base type can also hold a derived type.
 ***********************************************/
package generics;
import typeinfo.pets.*;

public class E01_PetsHolder {
  public static void main(String[] args) {
    Holder3<Pet> h3 = new Holder3<Pet>(new Mouse("Mickey"));
    System.out.println(h3.get());
  }
} /* Output:
Mouse Mickey
*///:~
```

Mouse is a kind of **Pet**, so **Holder3<Pet>** can hold an instance of **Mouse**.

Exercise 2

```
//: generics/E02_Holder4.java
/****************** Exercise 2 *****************
 * Create a holder class that holds three objects
 * of the same type, along with the methods to
 * store and fetch those objects and a constructor
 * to initialize all three.
 ***********************************************/
package generics;
class Holder4<T> {
  private T a, b, c;
  public Holder4(T a, T b, T c) {
    this.a = a;
    this.b = b;
    this.c = c;
  }
  public void setA(T a) { this.a = a; }
```

```
  public void setB(T b) { this.b = b; }
  public void setC(T c) { this.c = c; }
  public T getA() { return a; }
  public T getB() { return b; }
  public T getC() { return c; }
}

public class E02_Holder4 {
  public static void main(String[] args) {
    Holder4<String> h4 =
      new Holder4<String>("A", "B", "C");
    System.out.println(h4.getA());
    System.out.println(h4.getB());
    System.out.println(h4.getC());
    h4.setC("D");
    System.out.println(h4.getC());
  }
} /* Output:
A
B
C
D
*///:~
```

Exercise 3

```
//: generics/E03_SixTuple.java
/****************** Exercise 3 *****************
 * Create and test a SixTuple generic.
 ***********************************************/
package generics;
import net.mindview.util.*;

class SixTuple<A,B,C,D,E,F> extends FiveTuple<A,B,C,D,E> {
  public final F sixth;
  public SixTuple(A a, B b, C c, D d, E e, F f) {
    super(a, b, c, d, e);
    sixth = f;
  }
  public String toString() {
    return "(" + first + ", " + second + ", " +
      third + ", " + fourth + ", " + fifth + ", " +
      sixth + ")";
  }
}
```

```
public class E03_SixTuple {
  static
  SixTuple<Vehicle,Amphibian,String,Float,Double,Byte> a() {
    return new
      SixTuple<Vehicle,Amphibian,String,Float,Double,Byte>(
        new Vehicle(), new Amphibian(), "hi", (float)4.7,
        1.1, (byte)1);
  }
  public static void main(String[] args) {
    System.out.println(a());
  }
} /* Output: (75% match)
(generics.Vehicle@de6ced, generics.Amphibian@c17164, hi,
4.7, 1.1, 1)
*///:~
```

Exercise 4

```
//: generics/E04_GenericSequence.java
/****************** Exercise 4 *****************
 * "Generify" innerclasses/Sequence.java.
 ***********************************************/
package generics;
interface Selector<T> {
  boolean end();
  T current();
  void next();
}

class Sequence<T> {
  private Object[] items;
  private int next;
  public Sequence(int size) { items = new Object[size]; }
  public void add(T x) {
    if(next < items.length)
      items[next++] = x;
  }
  private class SequenceSelector implements Selector<T> {
    private int i;
    public boolean end() { return i == items.length; }
    @SuppressWarnings("unchecked")
    public T current() { return (T)items[i]; }
    public void next() { if(i < items.length) i++; }
  }
  public Selector<T> selector() {
    return new SequenceSelector();
```

```
  }
}

public class E04_GenericSequence {
  public static void main(String[] args) {
    Sequence<String> sequence = new Sequence<String>(10);
    for(int i = 0; i < 10; i++)
      sequence.add(Integer.toString(i));
    Selector<String> selector = sequence.selector();
    while(!selector.end()) {
      System.out.print(selector.current() + " ");
      selector.next();
    }
  }
} /* Output:
0 1 2 3 4 5 6 7 8 9
*///:~
```

Exercise 5

```
//: generics/E05_LinkedStack2.java
/****************** Exercise 5 *****************
 * Remove the type parameter on the Node class and
 * modify the rest of the code in LinkedStack.java
 * to show that an inner class has access to the
 * generic type parameters of its outer class.
 ***********************************************/
package generics;
class LinkedStack<T> {
  private class Node {
    T item;
    Node next;
    Node() { item = null; next = null; }
    Node(T item, Node next) {
      this.item = item;
      this.next = next;
    }
    boolean end() { return item == null && next == null; }
  }
  private Node top = new Node(); // End sentinel
  public void push(T item) { top = new Node(item, top); }
  public T pop() {
    T result = top.item;
    if(!top.end())
      top = top.next;
    return result;
```

```
  }
}

public class E05_LinkedStack2 {
  public static void main(String[] args) {
    LinkedStack<String> lss = new LinkedStack<String>();
    for(String s : "Phasers on stun!".split(" "))
      lss.push(s);
    String s;
    while((s = lss.pop()) != null)
      System.out.println(s);
  }
} /* Output:
stun!
on
Phasers
*///:~
```

Exercise 6

```
//: generics/E06_RandomListTest.java
/****************** Exercise 6 *****************
 * Use RandomList with two more types in addition
 * to the one shown in main().
 ***********************************************/
package generics;
import net.mindview.util.*;

public class E06_RandomListTest {
  private static void dump(RandomList<?> rl) {
    for(int i = 0; i < 11; i++)
      System.out.print(rl.select() + " ");
    System.out.println();
  }
  public static void main(String[] args) {
    RandomList<String> rs = new RandomList<String>();
    for(String s: ("The quick brown fox jumped over " +
        "the lazy brown dog").split(" "))
      rs.add(s);
    dump(rs);
    RandomList<Integer> ri = new RandomList<Integer>();
    Generator<Integer> gi =
      new CountingGenerator.Integer();
    for(int i = 0; i < 11; i ++)
      ri.add(gi.next());
    dump(ri);
```

```
    RandomList<Character> rc = new RandomList<Character>();
    Generator<Character> gc =
      new CountingGenerator.Character();
    for(int i = 0; i < 11; i ++)
      rc.add(gc.next());
    dump(rc);
  }
} /* Output:
brown over fox quick quick dog brown The brown lazy brown
8 1 9 10 0 0 1 4 5 2 9
i b j k a a b e f c j
*///:~
```

We use the **Generator** interface and the **CountingGenerator** class to produce sequences of integers and characters, respectively. You'll learn more about these later in *TIJ4*. Notice the parameter type of the **dump( )** method: **RandomList<?>** accepts any **RandomList** parameterized with an unknown type.

Exercise 7

```
//: generics/E07_IterableFibonacci2.java
/****************** Exercise 7 *****************
 * Use composition instead of inheritance to adapt
 * Fibonacci to make it Iterable.
 ***********************************************/
package generics;
import java.util.*;

class IterableFibonacci implements Iterable<Integer> {
  private Fibonacci fib = new Fibonacci();
  private int n;
  public IterableFibonacci(int count) { n = count; }
  public Iterator<Integer> iterator() {
    return new Iterator<Integer>() {
      public boolean hasNext() { return n > 0; }
      public Integer next() {
        --n;
        return fib.next();
      }
      public void remove() { // Not implemented
        throw new UnsupportedOperationException();
      }
    };
  }
}
```

```
public class E07_IterableFibonacci2 {
  public static void main(String[] args) {
    for(int i : new IterableFibonacci(18))
      System.out.print(i + " ");
  }
} /* Output:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
*///:~
```

Exercise 8

```
//: generics/E08_CharacterGenerator.java
/****************** Exercise 8 *****************
 * Following the form of the Coffee example, create
 * a hierarchy of StoryCharacters from your favorite
 * movie, dividing them into GoodGuys and BadGuys.
 * Create a generator for StoryCharacters, following
 * the form of CoffeeGenerator.
 ***********************************************/
package generics;
import java.util.*;
import net.mindview.util.*;

class StoryCharacter {
  private static long counter;
  private final long id = counter++;
  public String toString() {
    return getClass().getSimpleName() + " " + id;
  }
}

class GoodGuy extends StoryCharacter {
  public String toString() {
    return super.toString() + " is a good guy";
  }
}

class BadGuy extends StoryCharacter {
  public String toString() {
    return super.toString() + " is a bad guy";
  }
}

class Morton extends BadGuy {}
```

```
class Frank extends BadGuy {}

class Harmonica extends GoodGuy {}

class Cheyenne extends GoodGuy {}

class CharacterGenerator implements
Generator<StoryCharacter>, Iterable<StoryCharacter> {
  private Class<?>[] types = {
    Morton.class, Frank.class,
    Harmonica.class, Cheyenne.class
  };
  private static Random rand = new Random(47);
  public CharacterGenerator() {}
  private int size = 0;
  public CharacterGenerator(int sz) { size = sz; }
  public StoryCharacter next() {
    try {
      return (StoryCharacter)
        types[rand.nextInt(types.length)].newInstance();
    } catch(Exception e) {
      throw new RuntimeException(e);
    }
  }
  class CharacterIterator implements
  Iterator<StoryCharacter> {
    int count = size;
    public boolean hasNext() { return count > 0; }
    public StoryCharacter next() {
      count--;
      return CharacterGenerator.this.next();
    }
    public void remove() { // Not implemented
      throw new UnsupportedOperationException();
    }
  };
  public Iterator<StoryCharacter> iterator() {
    return new CharacterIterator();
  }
}

public class E08_CharacterGenerator {
  public static void main(String[] args) {
    CharacterGenerator gen = new CharacterGenerator();
    for(int i = 0; i < 7; i++)
      System.out.println(gen.next());
    for(StoryCharacter p : new CharacterGenerator(7))
```

```
      System.out.println(p);
  }
} /* Output:
Harmonica 0 is a good guy
Frank 1 is a bad guy
Harmonica 2 is a good guy
Morton 3 is a bad guy
Morton 4 is a bad guy
Harmonica 5 is a good guy
Morton 6 is a bad guy
Frank 7 is a bad guy
Harmonica 8 is a good guy
Harmonica 9 is a good guy
Frank 10 is a bad guy
Cheyenne 11 is a good guy
Frank 12 is a bad guy
Morton 13 is a bad guy
*///:~
```

Exercise 9

```
//: generics/E09_GenericMethods2.java
/****************** Exercise 9 *****************
 * Modify GenericMethods.java so that f() accepts
 * three arguments, all of which are of a different
 * parameterized type.
 ***********************************************/
package generics;
public class E09_GenericMethods2 {
  public <A,B,C> void f(A a, B b, C c) {
    System.out.println(a.getClass().getName());
    System.out.println(b.getClass().getName());
    System.out.println(c.getClass().getName());
  }
  public static void main(String[] args) {
    E09_GenericMethods2 gm = new E09_GenericMethods2();
    gm.f("", 1, 1.0);
    gm.f(1.0F, 'c', gm);
  }
} /* Output:
java.lang.String
java.lang.Integer
java.lang.Double
java.lang.Float
java.lang.Character
generics.E09_GenericMethods2
```

```
*///:~
```

Exercise 10

```
//: generics/E10_GenericMethods3.java
/****************** Exercise 10 *****************
 * Modify the previous exercise so that one of
 * f()'s arguments is non-parameterized.
 ************************************************/
package generics;
public class E10_GenericMethods3 {
  public <A,B> void f(A a, B b, Boolean c) {
    System.out.println(a.getClass().getName());
    System.out.println(b.getClass().getName());
    System.out.println(c.getClass().getName());
  }
  public static void main(String[] args) {
    E10_GenericMethods3 gm = new E10_GenericMethods3();
    gm.f("", 1, true);
    gm.f(1.0, 1.0F, false);
    gm.f('c', gm, true);
  }
} /* Output:
java.lang.String
java.lang.Integer
java.lang.Boolean
java.lang.Double
java.lang.Float
java.lang.Boolean
java.lang.Character
generics.E10_GenericMethods3
java.lang.Boolean
*///:~
```

Exercise 11

```
//: generics/E11_NewTest.java
/****************** Exercise 11 *****************
 * Test New.java by creating your own classes and
 * ensuring that New will work properly with them.
 ************************************************/
package generics;
import java.util.*;
import net.mindview.util.*;
```

```
public class E11_NewTest {
  public static void main(String[] args) {
    List<SixTuple<Byte,Short,String,Float,Double,Integer>>
      list = New.list();
    list.add(
      new SixTuple<Byte,Short,String,Float,Double,Integer>(
        (byte)1, (short)1, "A", 1.0F, 1.0, 1));
    System.out.println(list);
    Set<Sequence<String>> set = New.set();
    set.add(new Sequence<String>(5));
    System.out.println(set);
  }
} /* Output: (Sample)
[(1, 1, A, 1.0, 1.0, 1)]
[Sequence@3e25a5]
*///:~
```

Here we used the **SixTuple** and **Sequence** classes introduced earlier in this chapter.

Exercise 12

```
//: generics/E12_NewTest2.java
/****************** Exercise 12 *****************
 * Repeat the previous exercise using explicit type
 * specification.
 ************************************************/
package generics;
import java.util.*;
import net.mindview.util.*;

public class E12_NewTest2 {
  static void f(List<
    SixTuple<Byte,Short,String,Float,Double,Integer>> l) {
    l.add(
      new SixTuple<Byte,Short,String,Float,Double,Integer>(
        (byte)1, (short)1, "A", 1.0F, 1.0, 1));
    System.out.println(l);
  }
  static void g(Set<Sequence<String>> s) {
    s.add(new Sequence<String>(5));
    System.out.println(s);
  }
  public static void main(String[] args) {
    f(New.<SixTuple<Byte,Short,String,Float,Double,Integer>>
      list());
```

```
    g(New.<Sequence<String>>set());
  }
} /* Output: (78% match)
[(1, 1, A, 1.0, 1.0, 1)]
[Sequence@3e25a5]
*///:~
```

Exercise 13

```
//: generics/E13_OverloadedFill.java
/****************** Exercise 13 *****************
 * Overload the fill() method so the arguments
 * and return types are the specific subtypes of
 * Collection: List, Queue and Set. This way, you
 * don't lose the type of container. Can you overload
 * to distinguish between List and LinkedList?
 ************************************************/
package generics;
import generics.coffee.*;
import java.util.*;
import net.mindview.util.*;

public class E13_OverloadedFill {
  public static <T> List<T>
  fill(List<T> list, Generator<T> gen, int n) {
    for(int i = 0; i < n; i++)
      list.add(gen.next());
    return list;
  }
  public static <T> Queue<T>
  fill(Queue<T> queue, Generator<T> gen, int n) {
    for(int i = 0; i < n; i++)
      queue.add(gen.next());
    return queue;
  }
  public static <T> LinkedList<T>
  fill(LinkedList<T> llist, Generator<T> gen, int n) {
    for(int i = 0; i < n; i++)
      llist.add(gen.next());
    return llist;
  }
  public static <T> Set<T>
  fill(Set<T> set, Generator<T> gen, int n) {
    for(int i = 0; i < n; i++)
      set.add(gen.next());
    return set;
```

```
  }
  public static void main(String[] args) {
    List<Coffee> coffeeList = fill(
      new ArrayList<Coffee>(), new CoffeeGenerator(), 4);
    for(Coffee c : coffeeList)
      System.out.println(c);
    Queue<Integer> iQueue = fill(
      (Queue<Integer>)new LinkedList<Integer>(),
      new Fibonacci(), 12);
    for(int i : iQueue)
      System.out.print(i + " ");
    System.out.println();
    LinkedList<Character> cLList = fill(
      new LinkedList<Character>(),
      new CountingGenerator.Character(), 12);
    for(char ch : cLList)
      System.out.print(ch);
    System.out.println();
    Set<Boolean> bSet = fill(
      new HashSet<Boolean>(),
      new CountingGenerator.Boolean(), 10);
    for(Boolean b : bSet)
      System.out.println(b);
  }
} /* Output:
Americano 0
Latte 1
Americano 2
Mocha 3
1 1 2 3 5 8 13 21 34 55 89 144
abcdefghijkl
false
true
*///:~
```

You need explicit casting from **LinkedList** to **Queue** because **LinkedList** implements both **List** and **Queue** interfaces; without an explicit cast, the compiler will complain of an ambiguous method call. You can even overload to distinguish between **List** and **LinkedList**.

Exercise 14

```
//: generics/E14_BasicGeneratorDemo2.java
/****************** Exercise 14 *****************
 * Modify BasicGeneratorDemo.java to use the
 * explicit form of creation for the Generator
```

```
 * (that is, use the explicit constructor instead
 * of the generic create() method).
 *********************************************/
package generics;
import net.mindview.util.*;

public class E14_BasicGeneratorDemo2 {
  public static void main(String[] args) {
    Generator<CountedObject> gen =
      new BasicGenerator<CountedObject>(
        CountedObject.class);
    for(int i = 0; i < 5; i++)
      System.out.println(gen.next());
  }
} /* Output:
CountedObject 0
CountedObject 1
CountedObject 2
CountedObject 3
CountedObject 4
*///:~
```

Exercise 15

```
//: generics/E15_LeftToReader.java
/****************** Exercise 15 *****************
 * "Notice that f() returns a parameterized
 * TwoTuple object, while f2() returns an
 * unparameterized TwoTuple object. The compiler
 * doesn't warn about f2() in this case because the
 * return value is not being used in a parameterized
 * fashion; in a sense, it is being "upcast" to an
 * unparameterized TwoTuple. However, if you were to
 * try to capture the result of f2() into a
 * parameterized TwoTuple, the compiler would issue a
 * warning."
 *
 * Verify the previous statement.
 *********************************************/
package generics;

public class E15_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
} ///:~
```

Exercise 16

```
//: generics/E16_TupleTest3.java
/****************** Exercise 16 *****************
 * Add a SixTuple to Tuple.java, and test it in
 * TupleTest2.java.
 ************************************************/
package generics;
import net.mindview.util.*;
import static net.mindview.util.Tuple.*;

class Tuple2 extends Tuple {
  public static <A,B,C,D,E,F> SixTuple<A,B,C,D,E,F>
  tuple(A a, B b, C c, D d, E e, F f) {
    return new SixTuple<A,B,C,D,E,F>(a, b, c, d, e, f);
  }
}

public class E16_TupleTest3 {
  static TwoTuple<String,Integer> f() {
    return tuple("hi", 47);
  }
  static ThreeTuple<Amphibian,String,Integer> g() {
    return tuple(new Amphibian(), "hi", 47);
  }
  static
  FourTuple<Vehicle,Amphibian,String,Integer> h() {
    return tuple(new Vehicle(), new Amphibian(), "hi", 47);
  }
  static
  FiveTuple<Vehicle,Amphibian,String,Integer,Double> k() {
    return tuple(new Vehicle(), new Amphibian(),
      "hi", 47, 11.1);
  }
  static
  SixTuple<Vehicle,Amphibian,String,Integer,Double,Float>
  l() {
    return Tuple2.tuple(new Vehicle(), new Amphibian(),
      "hi", 47, 11.1, 1.0F);
  }
  public static void main(String[] args) {
    System.out.println(f());
    System.out.println(g());
    System.out.println(h());
    System.out.println(k());
```

```
    System.out.println(l());
  }
} /* Output: (65% match)
(hi, 47)
(generics.Amphibian@1fb8ee3, hi, 47)
(generics.Vehicle@61de33, generics.Amphibian@14318bb, hi,
47)
(generics.Vehicle@ca0b6, generics.Amphibian@10b30a7, hi, 47,
11.1)
(generics.Vehicle@1a758cb, generics.Amphibian@1b67f74, hi,
47, 11.1, 1.0)
*///:~
```

Exercise 17

```
//: generics/E17_Sets2.java
/****************** Exercise 17 *****************
 * Study the JDK documentation for EnumSet. You'll
 * see there's a clone() method.
 * However, you cannot clone() from the reference
 * to the Set interface passed in Sets.java. Can you
 * modify Sets.java to handle both the general case
 * of a Set interface as shown, and the special case
 * of an EnumSet, using clone() instead of creating
 * a new HashSet?
 ***********************************************/
package generics;
import generics.watercolors.*;
import java.util.*;
import static net.mindview.util.Print.*;
import static generics.watercolors.Watercolors.*;

class Sets2 {
  @SuppressWarnings("unchecked")
  protected static <T> Set<T> copy(Set<T> s) {
    if(s instanceof EnumSet)
      return ((EnumSet)s).clone();
    return new HashSet<T>(s);
  }
  public static <T> Set<T> union(Set<T> a, Set<T> b) {
    Set<T> result = copy(a);
    result.addAll(b);
    return result;
  }
  public static <T>
  Set<T> intersection(Set<T> a, Set<T> b) {
```

```
    Set<T> result = copy(a);
    result.retainAll(b);
    return result;
  }
  public static <T> Set<T>
  difference(Set<T> superset, Set<T> subset) {
    Set<T> result = copy(superset);
    result.removeAll(subset);
    return result;
  }
  public static <T> Set<T> complement(Set<T> a, Set<T> b) {
    return difference(union(a, b), intersection(a, b));
  }
}

public class E17_Sets2 {
  public static void main(String[] args) {
    Set<Watercolors> set1 =
      EnumSet.range(BRILLIANT_RED, VIRIDIAN_HUE);
    Set<Watercolors> set2 =
      EnumSet.range(CERULEAN_BLUE_HUE, BURNT_UMBER);
    print("set1: " + set1);
    print("set2: " + set2);
    print("union(set1, set2): " + Sets2.union(set1, set2));
    print("union(set1, set2) type: " +
      Sets2.union(set1, set2).getClass().getSimpleName());
    Set<Integer> set3 = new HashSet<Integer>();
    set3.add(1);
    Set<Integer> set4 = new HashSet<Integer>();
    set4.add(2);
    print("set3: " + set3);
    print("set4: " + set4);
    print("union(set3, set4): " + Sets2.union(set3, set4));
    print("union(set3, set4) type: " +
      Sets2.union(set3, set4).getClass().getSimpleName());

  }
} /* Output:
set1: [BRILLIANT_RED, CRIMSON, MAGENTA, ROSE_MADDER, VIOLET,
CERULEAN_BLUE_HUE, PHTHALO_BLUE, ULTRAMARINE,
COBALT_BLUE_HUE, PERMANENT_GREEN, VIRIDIAN_HUE]
set2: [CERULEAN_BLUE_HUE, PHTHALO_BLUE, ULTRAMARINE,
COBALT_BLUE_HUE, PERMANENT_GREEN, VIRIDIAN_HUE, SAP_GREEN,
YELLOW_OCHRE, BURNT_SIENNA, RAW_UMBER, BURNT_UMBER]
union(set1, set2): [BRILLIANT_RED, CRIMSON, MAGENTA,
ROSE_MADDER, VIOLET, CERULEAN_BLUE_HUE, PHTHALO_BLUE,
ULTRAMARINE, COBALT_BLUE_HUE, PERMANENT_GREEN, VIRIDIAN_HUE,
```

```
SAP_GREEN, YELLOW_OCHRE, BURNT_SIENNA, RAW_UMBER,
BURNT_UMBER]
union(set1, set2) type: RegularEnumSet
set3: [1]
set4: [2]
union(set3, set4): [1, 2]
union(set3, set4) type: HashSet
*///:~
```

This straightforward **copy( )** method is based on the *Factory Method* design pattern.

Exercise 18

```
//: generics/E18_OceanLife.java
/****************** Exercise 18 *****************
 * Following the form of BankTeller.java, create an
 * example where BigFish eat LittleFish in the Ocean.
 ************************************************/
package generics;
import java.util.*;
import net.mindview.util.*;

class LittleFish {
  private static long counter = 1;
  private final long id = counter++;
  private LittleFish() {}
  public String toString() { return "LittleFish " + id; }
  public static Generator<LittleFish> generator() {
    return new Generator<LittleFish>() {
      public LittleFish next() { return new LittleFish(); }
    };
  }
}

class BigFish {
  private static long counter = 1;
  private final long id = counter++;
  private BigFish() {}
  public String toString() { return "BigFish " + id; }
  public static Generator<BigFish> generator =
    new Generator<BigFish>() {
      public BigFish next() { return new BigFish(); }
    };
}
```

```
public class E18_OceanLife {
  public static void eat(BigFish bf, LittleFish lf) {
    System.out.println(bf + " eat " + lf);
  }
  public static void main(String[] args) {
    Random rand = new Random(47);
    List<LittleFish> littleF = new LinkedList<LittleFish>();
    Generators.fill(littleF, LittleFish.generator(), 15);
    List<BigFish> bigF = new ArrayList<BigFish>();
    Generators.fill(bigF, BigFish.generator, 4);
    for(LittleFish lf : littleF)
      eat(bigF.get(rand.nextInt(bigF.size())), lf);
  }
} /* Output:
BigFish 3 eat LittleFish 1
BigFish 2 eat LittleFish 2
BigFish 3 eat LittleFish 3
BigFish 1 eat LittleFish 4
BigFish 1 eat LittleFish 5
BigFish 3 eat LittleFish 6
BigFish 1 eat LittleFish 7
BigFish 2 eat LittleFish 8
BigFish 3 eat LittleFish 9
BigFish 3 eat LittleFish 10
BigFish 2 eat LittleFish 11
BigFish 4 eat LittleFish 12
BigFish 2 eat LittleFish 13
BigFish 1 eat LittleFish 14
BigFish 1 eat LittleFish 15
*///:~
```

Exercise 19

```
//: generics/E19_CargoShip.java
/****************** Exercise 19 *****************
 * Following the form of Store.java, build a model
 * of a containerized cargo ship.
 ***********************************************/
package generics;
import java.util.*;

class Container extends ArrayList<Product> {
  public Container(int nProducts) {
    Generators.fill(this, Product.generator, nProducts);
  }
}
```

```
class CargoHold extends ArrayList<Container> {
  public CargoHold(int nContainers, int nProducts) {
    for(int i = 0; i < nContainers; i++)
      add(new Container(nProducts));
  }
}

class Crane {}
class CommandSection {}

class CargoShip extends ArrayList<CargoHold> {
  private ArrayList<Crane> cranes = new ArrayList<Crane>();
  private CommandSection cmdSection = new CommandSection();
  public CargoShip(int nCargoHolds, int nContainers,
    int nProducts) {
    for(int i = 0; i < nCargoHolds; i++)
      add(new CargoHold(nContainers, nProducts));
  }
  public String toString() {
    StringBuilder result = new StringBuilder();
    for(CargoHold ch : this)
      for(Container c : ch)
        for(Product p : c) {
          result.append(p);
          result.append("\n");
        }
    return result.toString();
  }
}

public class E19_CargoShip {
  public static void main(String[] args) {
    System.out.println(new CargoShip(14, 5, 10));
  }
} /* Output: (Sample)
258: Test, price: $400.99
861: Test, price: $160.99
868: Test, price: $417.99
207: Test, price: $268.99
551: Test, price: $114.99
278: Test, price: $804.99
520: Test, price: $554.99
140: Test, price: $530.99
704: Test, price: $250.99
575: Test, price: $24.99
674: Test, price: $440.99
```

```
826: Test, price: $484.99
...
*///:~
```

Exercise 20

```
//: generics/E20_Bounds.java
/****************** Exercise 20 *****************
 * Create an interface with two methods, and a class
 * that implements that interface and adds another
 * method. In another class, create a generic method
 * with an argument type that is bounded by the
 * interface, and show that the methods in the
 * interface are callable inside this generic method.
 * In main(), pass an instance of the implementing
 * class to the generic method.
 ***********************************************/
package generics;

interface Top {
  void a();
  void b();
}

class CombinedImpl implements Top {
  public void a() { System.out.println("Top::a()"); }
  public void b() { System.out.println("Top::b()"); }
  public void c() {
    System.out.println("CombinedImpl::c()");
  }
}

public class E20_Bounds {
  static <T extends Top> void f(T obj) {
    obj.a();
    obj.b();
    // c() is not part of an interface
    // obj.c();
  }
  public static void main(String[] args) {
    f(new CombinedImpl());
  }
} /* Output:
Top::a()
Top::b()
*///:~
```

Exercise 21

```
//: generics/E21_ClassTypeCapture2.java
/****************** Exercise 21 *****************
 * Modify ClassTypeCapture.java by adding a
 * Map<String,Class<?>>, a method
 * addType(String typename, Class<?> kind), and a
 * method createNew(String typename).createNew()
 * will either produce a new instance of the class
 * associated with its argument string, or produce
 * an error message.
 ***********************************************/
package generics;
import java.util.*;
import static net.mindview.util.Print.*;

class ClassTypeCapture2 {
  Map<String,Class<?>> types =
    new HashMap<String,Class<?>>();
  public Object createNew(String typename) {
    Class<?> cl = types.get(typename);
    try {
      return cl.newInstance();
    } catch(NullPointerException e) {
      print("Not a registered typename: " + typename);
    } catch(Exception e) {
      print(e.toString());
    }
    return null;
  }
  public void addType(String typename, Class<?> kind) {
    types.put(typename, kind);
  }
}

public class E21_ClassTypeCapture2 {
  public static void main(String[] args) {
    ClassTypeCapture2 ctt = new ClassTypeCapture2();
    ctt.addType("Building", Building.class);
    ctt.addType("House", House.class);
    ctt.addType("Product", Product.class);
    print(ctt.createNew("Building").getClass());
    print(ctt.createNew("House").getClass());
    ctt.createNew("Product");
    ctt.createNew("Car");
```

```
  }
} /* Output:
class generics.Building
class generics.House
java.lang.InstantiationException: generics.Product
Not a registered typename: Car
*///:~
```

The **InstantiationException** is thrown because **Product** does not have a no-arg constructor. **ClassTypeCapture2** allows you to register any kind of class.

Exercise 22

```
//: generics/E22_InstantiateGenericType2.java
/****************** Exercise 22 *****************
 * Use a type tag along with reflection to create
 * a method that uses the argument version of
 * newInstance() to create an object of a class
 * with a constructor that has arguments.
 ************************************************/
package generics;
import java.lang.reflect.*;
import static net.mindview.util.Print.*;

class ClassAsFactory<T> {
  Class<T> kind;
  public ClassAsFactory(Class<T> kind) { this.kind = kind; }
  public T create(int arg) {
    try {
      // Technique 1 (verbose)
      for(Constructor<?> ctor : kind.getConstructors()) {
        // Look for a constructor with a single parameter:
        Class<?>[] params = ctor.getParameterTypes();
        if(params.length == 1)
          if(params[0] == int.class)
            return kind.cast(ctor.newInstance(arg));
      }
      // Technique 2 (direct)
      // Constructor<T> ct = kind.getConstructor(int.class);
      // return ct.newInstance(arg);
    } catch(Exception e) {
      throw new RuntimeException(e);
    }
    return null;
  }
}
```

```
public class E22_InstantiateGenericType2 {
  public static void main(String[] args) {
    ClassAsFactory<Employee> fe =
      new ClassAsFactory<Employee>(Employee.class);
    Employee emp = fe.create(1);
    if(emp == null)
      print("Employee cannot be instantiated!");
    ClassAsFactory<Integer> fi =
      new ClassAsFactory<Integer>(Integer.class);
    Integer i = fi.create(0);
    if(i == null)
      print("Integer cannot be instantiated!");
  }
} /* Output:
Employee cannot be instantiated!
*///:~
```

The main difference here is that technique 1 throws no exception if it can't find a suitable constructor, whereas technique 2 does. Also note that we use the **cast( )** method to perform a dynamic type-cast.

If you want to get fancier, ask the user (via the console) what values to use.

Exercise 23

```
//: generics/E23_FactoryConstraint2.java
/****************** Exercise 23 *****************
 * Modify FactoryConstraint.java so that create()
 * takes an argument.
 ************************************************/
package generics;

interface FactoryI<T> {
  T create(int arg);
}

class Foo<T> {
  private T x;
  public Foo(FactoryI<T> factory) {
    x = factory.create(1);
  }
  // ...
}

class IntegerFactory implements FactoryI<Integer> {
```

```
  public Integer create(int arg) {
    return new Integer(arg);
  }
}

class Widget {
  private final int id;
  Widget(int ident) { id = ident; }
  public String toString() { return "Widget " + id; }
  public static class Factory implements FactoryI<Widget> {
    public Widget create(int arg) {
      return new Widget(arg);
    }
  }
}

public class E23_FactoryConstraint2 {
  public static void main(String[] args) {
    new Foo<Integer>(new IntegerFactory());
    new Foo<Widget>(new Widget.Factory());
  }
} ///:~
```

Obviously, the factory and constrained approach for producing objects with arguments is much easier, and the whole solution leaner.

Exercise 24

```
//: generics/E24_FactoryCapture.java
/****************** Exercise 23 *****************
 * Modify Exercise 21 so that factory objects are
 * held in the Map instead of Class<?>.
 ************************************************/
package generics;
import java.util.*;
import static net.mindview.util.Print.*;

class FactoryCapture {
  Map<String,FactoryI<?>> factories =
    new HashMap<String,FactoryI<?>>();
  public Object createNew(String factoryname, int arg) {
    FactoryI<?> f = factories.get(factoryname);
    try {
      return f.create(arg);
    } catch(NullPointerException e) {
      print("Not a registered factoryname: " + factoryname);
```

```
      return null;
    }
  }
  public void
  addFactory(String factoryname, FactoryI<?> factory) {
    factories.put(factoryname, factory);
  }
}

public class E24_FactoryCapture {
  public static void main(String[] args) {
    FactoryCapture fc = new FactoryCapture();
    fc.addFactory("Integer", new IntegerFactory());
    fc.addFactory("Widget", new Widget.Factory());
    print(fc.createNew("Integer", 1));
    print(fc.createNew("Widget", 2));
    fc.createNew("Product", 3);
  }
} /* Output:
1
Widget 2
Not a registered factoryname: Product
*///:~
```

Factories eliminate the problems of not finding a suitable constructor.

Exercise 25

```
//: generics/E25_Bounds.java
/****************** Exercise 25 *****************
 * Create two interfaces and a class that implements
 * both. Create two generic methods, one whose argument
 * parameter is bounded by the first interface and
 * one whose argument parameter is bounded by the
 * second interface. Create an instance of the class
 * that implements both interfaces, and show that
 * it can be used with both generic methods.
 ************************************************/
package generics;

interface Low {
  void c();
  void d();
}

class TopLowImpl implements Top, Low {
```

```
  public void a() { System.out.println("Top::a()"); }
  public void b() { System.out.println("Top::b()"); }
  public void c() { System.out.println("Low::c()"); }
  public void d() { System.out.println("Low::d()"); }
}

public class E25_Bounds {
  static <T extends Top> void top(T obj) {
    obj.a();
    obj.b();
  }
  static <T extends Low> void low(T obj) {
    obj.c();
    obj.d();
  }
  public static void main(String[] args) {
    TopLowImpl tli = new TopLowImpl();
    top(tli);
    low(tli);
  }
} /* Output:
Top::a()
Top::b()
Low::c()
Low::d()
*///:~
```

Exercise 26

```
//: generics/E26_CovariantArrays2.java
/****************** Exercise 26 *****************
 * Demonstrate array covariance using Numbers and
 * Integers.
 ************************************************/
package generics;

public class E26_CovariantArrays2 {
  public static void main(String[] args) {
    Number[] nums = new Integer[10];
    nums[0] = Integer.valueOf(1); // OK
    // Runtime type is Integer[], not Float[] or Byte[]:
    try {
      // Compiler allows you to add Float:
      nums[1] = new Float(1.0); // ArrayStoreException
    } catch(Exception e) { System.out.println(e); }
    try {
```

```
      // Compiler allows you to add Byte:
      nums[2] = Byte.valueOf((byte)1);
      // Above line produces an ArrayStoreException
    } catch(Exception e) { System.out.println(e); }
  }
} /* Output:
java.lang.ArrayStoreException: java.lang.Float
java.lang.ArrayStoreException: java.lang.Byte
*///:~
```

Exercise 27

```
//: generics/E27_GenericsAndCovariance2.java
/****************** Exercise 27 *****************
 * Show that covariance doesn't work with Lists,
 * using Numbers and Integers, then introduce
 * wildcards.
 ************************************************/
package generics;
import java.util.*;

public class E27_GenericsAndCovariance2 {
  public static void main(String[] args) {
    // Compile Error: incompatible types:
    // List<Number> nlist = new ArrayList<Integer>();
    // Wildcards allow covariance:
    List<? extends Number> nlist = new ArrayList<Integer>();
    // Compile Error: can't add any type of object:
    // nlist.add(new Integer(1));
    // nlist.add(new Float(1.0));
    // nlist.add(new Object());
    nlist.add(null); // Legal but uninteresting
    // We know that it returns at least Number:
    Number n = nlist.get(0);
  }
} ///:~
```

Exercise 28

```
//: generics/E28_GenericReadAndWrite.java
/****************** Exercise 28 *****************
 * Create a generic class Generic1<T> with a single
 * method that takes an argument of type T. Create
 * a second generic class Generic2<T> with a single
 * method that returns an argument of type T. Write
```

```
 * a generic method with a contravariant argument of
 * the first generic class that calls its method.
 * Write a second generic method with a covariant
 * argument of the second generic class that calls
 * its method. Test using the typeinfo.pets library.
 *********************************************/
package generics;
import typeinfo.pets.*;

class Generic1<T> {
  public void set(T arg) {}
}

class Generic2<T> {
  public T get() { return null; }
}

public class E28_GenericReadAndWrite {
  static <T> void f1(Generic1<? super T> obj, T item) {
    obj.set(item);
  }
  static <T> T f2(Generic2<? extends T> obj) {
    return obj.get();
  }
  public static void main(String[] args) {
    Generic1<Rodent> g1 = new Generic1<Rodent>();
    f1(g1, new Mouse());  // OK
    // Compile error: Cat is not a Rodent
    // f1(g1, new Cat());
    Generic2<Pet> g2 = new Generic2<Pet>();
    Pet pet = f2(g2);  // OK
    Generic2<Cat> g3 = new Generic2<Cat>();
    Cat cat = f2(g3);  // OK
    pet = f2(g3);  // OK
  }
} ///:~
```

Exercise 29

```
//: generics/E29_WildcardTest.java
/****************** Exercise 29 *****************
 * Create a generic method that takes as an
 * argument a Holder<List<?>>. Determine what
 * methods you can and can't call for the Holder
 * and for the List. Repeat for an argument of
 * List<Holder<?>>.
```

```
 *************************************************/
package generics;
import java.util.*;
import static net.mindview.util.Print.*;

public class E29_WildcardTest {
  static void f1(Holder<List<?>> holder) {
    List<?> list = holder.get();
    print(holder.equals(list));
    // Compile errors:
    // list.add(1);
    // list.add(new Object());
    Integer i = (Integer)list.get(0);
    print(i);
    print(list.contains(i));
    print(list.remove(i));
    holder.set(new ArrayList<Float>());
  }
  static void f2(List<Holder<?>> list) {
    Holder<?> holder = (Holder<?>)list.get(0);
    print(holder.equals(Integer.valueOf(1)));
    // Compile error:
    // holder.set(new Integer(2));
    print(holder.get());
    list.add(new Holder<Float>(1.0F));
    print(list.get(1).get());
    list.remove(0);
    print(list.size());
  }
  public static void main(String[] args) {
    List<Integer> list1 = new ArrayList<Integer>();
    list1.add(1);
    f1(new Holder<List<?>>(list1));
    List<Holder<?>> list2 = new ArrayList<Holder<?>>();
    list2.add(new Holder<Integer>(1));
    f2(list2);
  }
} /* Output:
true
1
true
true
true
1
1.0
1
*///:~
```

The compiler does not allow you to "write" into a generic type through an unbounded reference, but permits all other operations.

Exercise 30

```
//: generics/E30_AutoboxingUnboxingTest.java
/****************** Exercise 30 *****************
 * Create a Holder for each of the primitive
 * wrapper types, and show that autoboxing and
 * autounboxing works for the set() and get()
 * methods of each instance.
 ************************************************/
package generics;
import static net.mindview.util.Print.*;

public class E30_AutoboxingUnboxingTest {
  public static void main(String[] args) {
    Holder<Integer> hi = new Holder<Integer>();
    hi.set(1);
    int i = hi.get();
    print(i == 1);
    Holder<Byte> hb = new Holder<Byte>();
    hb.set((byte)1);
    byte b = hb.get();
    print(b == 1);
    Holder<Short> hs = new Holder<Short>();
    hs.set((short)1);
    short s = hs.get();
    print(s == 1);
    Holder<Long> hl = new Holder<Long>();
    hl.set(1L);
    long l = hl.get();
    print(l == 1);
    Holder<Float> hf = new Holder<Float>();
    hf.set(1.0F);
    float f = hf.get();
    print(f == 1.0F);
    Holder<Double> hd = new Holder<Double>();
    hd.set(1.0);
    double d = hd.get();
    print(d == 1.0);
    Holder<Character> hc = new Holder<Character>();
    hc.set('A');
    char c = hc.get();
    print(c == 'A');
```

```
    Holder<Boolean> hbool = new Holder<Boolean>();
    hbool.set(true);
    boolean bool = hbool.get();
    print(bool);
  }
} /* Output:
true
true
true
true
true
true
true
true
*///:~
```

Exercise 31

```
//: generics/E31_MultipleInterfaceVariants2.java
/****************** Exercise 31 *****************
 * Remove all the generics from
 * MultipleInterfaceVariants.java and modify the
 * code so that the example compiles.
 ************************************************/
package generics;
interface Payable {}

class Employee implements Payable {}

class Hourly extends Employee implements Payable {}

public class E31_MultipleInterfaceVariants2 {
  public static void main(String[] args) {
    new Employee();
    new Hourly();
  }
} ///:~
```

Exercise 32

```
//: generics/E32_FixedSizeStackTest.java
/****************** Exercise 32 *****************
 * Verify that FixedSizeStack in GenericCast.java
 * generates exceptions if you try to go out of
 * its bounds. Does this mean that bounds-checking
```

```
 * code is not required?
 *********************************************/
package generics;

public class E32_FixedSizeStackTest {
  public static void main(String[] args) {
    FixedSizeStack<Integer> stack =
      new FixedSizeStack<Integer>(1);
    stack.push(1);
    System.out.println(stack.pop());
    try {
      stack.pop();
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println(e.toString());
    }
    stack = new FixedSizeStack<Integer>(1);
    stack.push(2);
    try {
      stack.push(2);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println(e.toString());
    }
  }
} /* Output:
1
java.lang.ArrayIndexOutOfBoundsException: -1
java.lang.ArrayIndexOutOfBoundsException: 1
*///:~
```

Obviously, the **FixedSizeStack** does not check whether the index went out of bounds before accessing the internal array. This triggers an inappropriate exception; e.g., in case of underflow, the program signals **ArrayIndexOutOfBoundsException** instead of **java.util.EmptyStackException**, which may be confusing. The program also signals **ArrayIndexOutOfBoundsException** in case of overflow, instead of a custom exception (like **FullStackException**). Bounds checking both conceals the implementation details of a class and throws exceptions meaningful to the client.

Exercise 33

```
//: generics/E33_GenericCast2.java
/****************** Exercise 33 *****************
 * Repair GenericCast.java using an ArrayList.
 ************************************************/
```

```
package generics;
import java.util.*;
import static net.mindview.util.Print.*;

class FullStackException extends RuntimeException {}

class FixedSizeStack2<T> {
  private int index;
  private final int size;
  private final List<T> storage;
  public FixedSizeStack2(int size) {
    storage = new ArrayList<T>(size);
    this.size = size;
  }
  public void push(T item) {
    if(index < size) {
      index++;
      storage.add(item);
    } else
      throw new FullStackException();
  }
  public T pop() {
    if(index > 0)
      return storage.get(--index);
    throw new EmptyStackException();
  }
}

public class E33_GenericCast2 {
  public static final int SIZE = 10;
  public static void main(String[] args) {
    FixedSizeStack2<String> strings =
      new FixedSizeStack2<String>(SIZE);
    for(String s : "A B C D E F G H I J".split(" "))
      strings.push(s);
    for(int i = 0; i < SIZE; i++) {
      String s = strings.pop();
      printnb(s + " ");
    }
    print();
    try {
      strings.pop();
    } catch(EmptyStackException e) {
      print("Stack is empty");
    }
    strings = new FixedSizeStack2<String>(1);
    strings.push("A");
```

```
    try {
      strings.push("B");
    } catch(FullStackException e) {
      print("Stack is full");
    }
  }
} /* Output:
J I H G F E D C B A
Stack is empty
Stack is full
*///:~
```

Exercise 34

```
//: generics/E34_SelfBounded.java
/****************** Exercise 34 *****************
 * Create a self-bounded generic type that contains
 * an abstract method that takes an argument of the
 * generic type parameter and produces a return value
 * of the generic type parameter. In a non-abstract
 * method of the class, call the abstract method and
 * return its result. Inherit from the self-bounded
 * type and test the resulting class.
 ************************************************/
package generics;

abstract class
GenericProcessor<T extends GenericProcessor<T>> {
  abstract T process(T arg);
  T test() { return process(null); }
}

class ConcreteProcessor
extends GenericProcessor<ConcreteProcessor> {
  ConcreteProcessor process(ConcreteProcessor arg) {
    if(arg == null)
      return this;
    return arg;
  }
}

public class E34_SelfBounded {
  public static void main(String[] args) {
    ConcreteProcessor cp = new ConcreteProcessor();
    System.out.println(cp == cp.test());
  }
```

```
} /* Output:
true
*///:~
```

A self-bounded generic type enables covariant return and argument types.

Exercise 35

```
//: generics/E35_CheckedList2.java
/****************** Exercise 35 *****************
 * Modify CheckedList.java so that it uses the Coffee
 * classes defined in this chapter.
 ************************************************/
package generics;
import generics.coffee.*;
import java.util.*;

public class E35_CheckedList2 {
  @SuppressWarnings("unchecked")
  static void oldStyleMethod(List probablyAmericanos) {
    probablyAmericanos.add(new Breve());
  }
  public static void main(String[] args) {
    List<Americano> am1 = new ArrayList<Americano>();
    oldStyleMethod(am1); // Quietly accepts a Breve
    List<Americano> am2 = Collections.checkedList(
      new ArrayList<Americano>(), Americano.class);
    try {
      oldStyleMethod(am2); // Throws an exception
    } catch(Exception e) {
      System.out.println(e);
    }
    // Derived types work fine:
    List<Coffee> coffees = Collections.checkedList(
      new ArrayList<Coffee>(), Coffee.class);
    coffees.add(new Americano());
    coffees.add(new Breve());
  }
} /* Output:
java.lang.ClassCastException: Attempt to insert class
generics.coffee.Breve element into collection with element
type class generics.coffee.Americano
*///:~
```

Exercise 36

```
//: generics/E36_GenericExceptions.java
/****************** Exercise 36 *****************
 * Add a second parameterized exception to the
 * Processor class and demonstrate that the exceptions
 * can vary independently.
 ************************************************/
package generics;
import java.util.*;

interface
Processor<T,E extends Exception,F extends Exception> {
  void process(List<T> resultCollector) throws E, F;
}

class
ProcessRunner<T,E extends Exception,F extends Exception>
extends ArrayList<Processor<T,E,F>> {
  List<T> processAll() throws E, F {
    List<T> resultCollector = new ArrayList<T>();
    for(Processor<T,E,F> processor : this)
      processor.process(resultCollector);
    return resultCollector;
  }
}

class Failure1_1 extends Exception {}
class Failure1_2 extends Exception {}

class Processor1 implements
Processor<String,Failure1_1,Failure1_2> {
  static Random rnd = new Random(47);
  static int count = 3;
  public void process(List<String> resultCollector)
  throws Failure1_1, Failure1_2 {
    if(count-- > 1)
      resultCollector.add("Hep!");
    else
      resultCollector.add("Ho!");
    if(count < 0)
      if(rnd.nextBoolean())
        throw new Failure1_1();
      throw new Failure1_2();
  }
}
```

```
class Failure2_1 extends Exception {}
class Failure2_2 extends Exception {}

class Processor2 implements
Processor<Integer,Failure2_1,Failure2_2> {
  static Random rnd = new Random(47);
  static int count = 2;
  public void
  process(List<Integer> resultCollector)
  throws Failure2_1, Failure2_2 {
    if(count-- == 0)
      resultCollector.add(47);
    else {
      resultCollector.add(11);
    }
    if(count < 0)
      if(rnd.nextBoolean())
        throw new Failure2_1();
      throw new Failure2_2();
  }
}

public class E36_GenericExceptions {
  public static void main(String[] args) {
    ProcessRunner<String,Failure1_1,Failure1_2> runner =
      new ProcessRunner<String,Failure1_1,Failure1_2>();
    for(int i = 0; i < 3; i++)
      runner.add(new Processor1());
    try {
      System.out.println(runner.processAll());
    } catch(Failure1_1 e) {
      System.out.println(e);
    } catch(Failure1_2 e) {
      System.out.println(e);
    }
    ProcessRunner<Integer,Failure2_1,Failure2_2> runner2 =
      new ProcessRunner<Integer,Failure2_1,Failure2_2>();
    for(int i = 0; i < 3; i++)
      runner2.add(new Processor2());
    try {
      System.out.println(runner2.processAll());
    } catch(Failure2_1 e) {
      System.out.println(e);
    } catch(Failure2_2 e) {
      System.out.println(e);
    }
```

```
  }
} /* Output:
generics.Failure1_2
generics.Failure2_2
*///:~
```

Exercise 37

```
//: generics/E37_Mixins2.java
/****************** Exercise 37 *****************
 * Add a new mixin class Colored to Mixins.java,
 * mix it in to Mixin, and show that it works.
 ************************************************/
package generics;
import java.awt.Color;
import java.util.*;

interface Colored { Color getColor(); }

class ColoredImp implements Colored {
  private static Random rnd = new Random(47);
  private final Color clr =
    new Color(rnd.nextInt(16777216));  // 2^24
  public Color getColor() { return clr; }
}

class Mixin2 extends Mixin implements Colored {
  private Colored colored = new ColoredImp();
  public Color getColor() { return colored.getColor(); }
}

public class E37_Mixins2 {
  public static void main(String[] args) {
    Mixin2 mixin1 = new Mixin2(), mixin2 = new Mixin2();
    mixin1.set("test string 1");
    mixin2.set("test string 2");
    System.out.println(mixin1.get() + " " +
      mixin1.getStamp() +  " " +
      mixin1.getSerialNumber() + " " + mixin1.getColor());
    System.out.println(mixin2.get() + " " +
      mixin2.getStamp() +  " " +
      mixin2.getSerialNumber() + " " + mixin2.getColor());
  }
} /* Output: (78% match)
test string 1 1135953794432 1
java.awt.Color[r=186,g=36,b=66]
```

```
test string 2 1135953794479 2 java.awt.Color[r=102,g=91,b=5]
*///:~
```

Exercise 38

```
//: generics/E38_DecoratorSystem.java
/****************** Exercise 38 *****************
 * Create a simple Decorator system by starting
 * with basic coffee, then providing decorators
 * of steamed milk, foam, chocolate, caramel
 * and whipped cream.
 ************************************************/
package generics;
import java.awt.Color;

class BasicCoffee {
  private String type;
  public BasicCoffee() {}
  public BasicCoffee(String type) { setType(type); }
  public void setType(String type) { this.type = type; }
  public String getType() { return type; }
}

class CoffeeDecorator extends BasicCoffee {
  protected BasicCoffee basic;
  public CoffeeDecorator(BasicCoffee basic) {
    this.basic = basic;
  }
  public void setType(String type) { basic.setType(type); }
  public String getType() { return basic.getType(); }
}

class SteamedMilk extends CoffeeDecorator {
  public SteamedMilk(BasicCoffee basic) {
    super(basic);
    setType(getType() + " & steamed milk");
  }
}

class Foam extends CoffeeDecorator {
  public Foam(BasicCoffee basic) {
    super(basic);
    setType(getType() + " & foam");
  }
}
```

```
class Chocolate extends CoffeeDecorator {
  private final Color color;
  public Chocolate(BasicCoffee basic, Color color) {
    super(basic);
    this.color = color;
    setType(getType() + " & chocolate[color = " +
      getColor() + "]");
  }
  public Color getColor() { return color; }
}

class Caramel extends CoffeeDecorator {
  public Caramel(BasicCoffee basic) {
    super(basic);
    setType(getType() + " & caramel");
  }
}

class WhippedCream extends CoffeeDecorator {
  public WhippedCream(BasicCoffee basic) {
    super(basic);
    setType(getType() + " & whipped cream");
  }
}

public class E38_DecoratorSystem {
  public static void main(String[] args) {
    CoffeeDecorator cappuccino = new Foam(
      new SteamedMilk(new BasicCoffee("espresso")));
    System.out.println(
      "Capuccino is: " + cappuccino.getType());
    CoffeeDecorator whiteChocolateCoffee = new WhippedCream(
      new Chocolate(
        new BasicCoffee("hot coffee"), Color.WHITE));
    System.out.println("White Chocolate Coffee is: " +
      whiteChocolateCoffee.getType());
  }
} /* Output:
Capuccino is: espresso & steamed milk & foam
White Chocolate Coffee is: hot coffee & chocolate[color =
java.awt.Color[r=255,g=255,b=255]] & whipped cream
*///:~
```

Here, we "prepared" a cappuccino and a white-chocolate coffee. Make some more coffees as an extra exercise. Notice that only **Chocolate** adds a new method to the **CoffeeDecorator**.

Exercise 39

```
//: generics/E39_DynamicProxyMixin2.java
/****************** Exercise 39 *****************
 * Add a new mixin class Colored to
 * DynamicProxyMixin.java, mix it in to mixin, and
 * show that it works.
 ************************************************/
package generics;
import static net.mindview.util.Tuple.*;

public class E39_DynamicProxyMixin2 {
  public static void main(String[] args) {
    Object mixin = MixinProxy.newInstance(
      tuple(new BasicImp(), Basic.class),
      tuple(new TimeStampedImp(), TimeStamped.class),
      tuple(new SerialNumberedImp(),SerialNumbered.class),
      tuple(new ColoredImp(), Colored.class));
    Basic b = (Basic)mixin;
    TimeStamped t = (TimeStamped)mixin;
    SerialNumbered s = (SerialNumbered)mixin;
    Colored c = (Colored)mixin;
    b.set("Hello");
    System.out.println(b.get());
    System.out.println(t.getStamp());
    System.out.println(s.getSerialNumber());
    System.out.println(c.getColor());
  }
} /* Output: (75% match)
Hello
1135959445620
1
java.awt.Color[r=186,g=36,b=66]
*///:~
```

Exercise 40

```
//: generics/E40_SpeakingPets.java
/****************** Exercise 40 *****************
 * Add a speak() method to all the pets in
 * typeinfo.pets. Modify Apply.java to call the
 * speak() method for a heterogeneous collection
 * of Pet.
 ************************************************/
package generics;
```

```
import java.util.*;

class SPet extends typeinfo.pets.Individual {
  public SPet(String name) { super(name); }
  public SPet() { super(); }
  public void speak() {
    System.out.println(this + " speak");
  }
}

class SRodent extends SPet {
  public SRodent(String name) { super(name); }
  public SRodent() { super(); }
}

class SRat extends SRodent {
  public SRat(String name) { super(name); }
  public SRat() { super(); }
}

class SPug extends SDog {
  public SPug(String name) { super(name); }
  public SPug() { super(); }
}

class SMutt extends SDog {
  public SMutt(String name) { super(name); }
  public SMutt() { super(); }
}

class SMouse extends SRodent {
  public SMouse(String name) { super(name); }
  public SMouse() { super(); }
}

class SManx extends SCat {
  public SManx(String name) { super(name); }
  public SManx() { super(); }
}

class SHamster extends SRodent {
  public SHamster(String name) { super(name); }
  public SHamster() { super(); }
}

class SEgyptianMau extends SCat {
  public SEgyptianMau(String name) { super(name); }
```

```
  public SEgyptianMau() { super(); }
}

class SDog extends SPet {
  public SDog(String name) { super(name); }
  public SDog() { super(); }
}

class SCymric extends SManx {
  public SCymric(String name) { super(name); }
  public SCymric() { super(); }
}

class SCat extends SPet {
  public SCat(String name) { super(name); }
  public SCat() { super(); }
}

public class E40_SpeakingPets {
  public static void main(String[] args) throws Exception {
    List<SPet> pets = Arrays.asList(new SRat(), new SPug(),
      new SMutt(), new SMouse(), new SManx(),
      new SHamster(), new SEgyptianMau(), new SCymric());
    Apply.apply(pets, SPet.class.getMethod("speak"));
  }
} /* Output:
SRat speak
SPug speak
SMutt speak
SMouse speak
SManx speak
SHamster speak
SEgyptianMau speak
SCymric speak
*///:~
```

Exercise 41

```
//: generics/E41_Fill3.java
/****************** Exercise 41 *****************
 * Modify Fill2.java to use the classes in
 * typeinfo.pets instead of the Coffee classes.
 ************************************************/
package generics;
import java.util.*;
import static net.mindview.util.Print.*;
```

```
import typeinfo.pets.*;

public class E41_Fill3 {
  public static void main(String[] args) throws Exception {
    // Adapt a Collection:
    List<Pet> carrier = new ArrayList<Pet>();
    Fill2.fill(
      new AddableCollectionAdapter<Pet>(carrier),
      Pet.class, 3);
    // Helper method captures the type:
    Fill2.fill(Adapter.collectionAdapter(carrier),
      Mouse.class, 2);
    for(Pet p: carrier)
      print(p);
    print("----------------------");
    // Use an adapted class:
    AddableSimpleQueue<Pet> petQueue =
      new AddableSimpleQueue<Pet>();
    Fill2.fill(petQueue, Mutt.class, 4);
    Fill2.fill(petQueue, Cymric.class, 1);
    for(Pet p: petQueue)
      print(p);
  }
} /* Output:
Pet
Pet
Pet
Mouse
Mouse
----------------------
Mutt
Mutt
Mutt
Mutt
Cymric
*///:~
```

Exercise 42

```
//: generics/E42_Functional2.java
/****************** Exercise 42 *****************
 * Create two separate classes, with nothing in
 * common. Each class should hold a value, and at
 * least have methods that produce that value and
 * perform a modification upon that value. Modify
 * Functional.java so that it performs functional
```

```
 * operations on collections of your classes (these
 * operations do not have to be arithmetic as they
 * are in Functional.java).
 *********************************************/
package generics;
import java.util.*;
import static net.mindview.util.Print.*;

final class DataManipulator1
implements Comparable<DataManipulator1> {
  private int value;
  public DataManipulator1(int value) { this.value = value; }
  public void increment() { ++value; }
  public int getValue() { return value; }
  public int compareTo(DataManipulator1 other) {
    return Integer.valueOf(value).compareTo(other.value);
  }
  public String toString() {
    return Integer.toString(value);
  }
}

final class DataManipulator2 {
  private String value;
  public DataManipulator2(String value) { setValue(value); }
  public void setValue(String value) { this.value = value; }
  public String getValue() { return value; }
}

public class E42_Functional2 {
  // To use the above generic methods, we need to create
  // function objects to adapt to our particular needs:
  static class Incrementor implements
  UnaryFunction<DataManipulator1,DataManipulator1> {
    public DataManipulator1 function(DataManipulator1 x) {
      x.increment();
      return x;
    }
  }
  static class UpperCaseConverter
  implements UnaryFunction<String,DataManipulator2> {
    public String function(DataManipulator2 x) {
      return x.getValue().toUpperCase();
    }
  }
  static class Concatenator
  implements Combiner<DataManipulator2> {
```

```
    public DataManipulator2
    combine(DataManipulator2 x,DataManipulator2 y) {
      x.setValue(x.getValue() + y.getValue());
      return x;
    }
  }
  public static void main(String[] args) {
    DataManipulator1 rf = new DataManipulator1(4);
    List<DataManipulator1> ldm1 = Arrays.asList(
      new DataManipulator1(3), new DataManipulator1(10),
      new DataManipulator1(1),new DataManipulator1(7));
    print(
      Functional.transform(
        Functional.filter(ldm1,
          new Functional.GreaterThan<DataManipulator1>(rf)),
        new Incrementor()));

    List<DataManipulator2> ldm2 = Arrays.asList(
      new DataManipulator2("a"), new DataManipulator2("B"),
      new DataManipulator2("c"),new DataManipulator2("d"));
    print(
      Functional.transform(ldm2, new UpperCaseConverter()));
    print(Functional.reduce(ldm2,
      new Concatenator()).getValue());
  }
} /* Output:
[11, 8]
[A, B, C, D]
aBcd
*///:~
```

Arrays

Exercise 1

```
//: arrays/E01_ArrayInitialization.java
/****************** Exercise 1 *****************
 * Create a method that takes an array of
 * BerylliumSphere as an argument. Call the method,
 * creating the argument dynamically. Demonstrate
 * that ordinary aggregate array initialization
 * doesn't work in this case. Discover the only
 * situations where ordinary aggregate array
 * initialization works, and where dynamic aggregate
 * initialization is redundant.
 ***********************************************/
package arrays;

public class E01_ArrayInitialization {
  static void hide(BerylliumSphere[] s) {
    System.out.println("Hiding " + s.length + " sphere(s)");
  }
  public static void main(String[] args) {
    // Dynamic aggregate initialization:
    hide(new BerylliumSphere[]{ new BerylliumSphere(),
      new BerylliumSphere() });
    // The following line produces a compilation error.
    //! hide({ new BerylliumSphere() });
    // Aggregate initialization:
    BerylliumSphere[] d = { new BerylliumSphere(),
      new BerylliumSphere(), new BerylliumSphere() };
    hide(d);
    // Dynamic aggregate initialization is redundant
    // in the next case:
    BerylliumSphere[] a = new BerylliumSphere[]{
      new BerylliumSphere(), new BerylliumSphere() };
    hide(a);
  }
} /* Output:
Hiding 2 sphere(s)
Hiding 3 sphere(s)
Hiding 2 sphere(s)
*///:~
```

You must use aggregate initialization at the point of an array variable's definition. With dynamic initialization you can create and initialize an array object anywhere.

Exercise 2

```
//: arrays/E02_ReturningArray.java
/****************** Exercise 2 ******************
 * Write a method that takes an int argument and
 * returns an array of that size, filled with
 * BerylliumSphere objects.
 ***********************************************/
package arrays;
import java.util.Arrays;

public class E02_ReturningArray {
  static BerylliumSphere[] createArray(int size) {
    BerylliumSphere[] a = new BerylliumSphere[size];
    for(int i = 0; i < size; i++)
      a[i] = new BerylliumSphere();
    return a;
  }
  public static void main(String[] args) {
    System.out.println(Arrays.toString(createArray(10)));
  }
} /* Output:
[Sphere 0, Sphere 1, Sphere 2, Sphere 3, Sphere 4, Sphere 5,
Sphere 6, Sphere 7, Sphere 8, Sphere 9]
*///:~
```

Exercise 3

```
//: arrays/E03_TwoDDoubleArray.java
/****************** Exercise 3 *****************
 * Write a method that creates and initializes a
 * two-dimensional array of double. The size of
 * the array is determined by the arguments of
 * the method, and the initialization values are
 * a range determined by beginning and ending
 * values that are also arguments of the method.
 * Create a second method that will print the
 * array generated by the first method. In main()
 * test the methods by creating and printing
 * several different sizes of arrays.
```

```
 *************************************************/
package arrays;
import java.util.Locale;
import static net.mindview.util.Print.*;

public class E03_TwoDDoubleArray {
  public static double[][] twoDDoubleArray(
    int xLen, int yLen, double valStart, double valEnd) {
    double[][] array = new double[xLen][yLen];
    double increment = (valEnd - valStart)/(xLen * yLen);
    double val = valStart;
    for(int i = 0; i < array.length; i++)
      for(int j = 0; j < array[i].length; j++) {
        array[i][j] = val;
        val += increment;
      }
    return array;
  }
  public static void printArray(double[][] array) {
    for(int i = 0; i < array.length; i++) {
      for(int j = 0; j < array[i].length; j++)
        System.out.printf(Locale.US, "%.2f ", array[i][j]);
      print();
    }
  }
  public static void main(String args[]) {
    double[][] twoD = twoDDoubleArray(4, 6, 47.0, 99.0);
    printArray(twoD);
    print("**********************");
    double[][] twoD2 = twoDDoubleArray(2, 2, 47.0, 99.0);
    printArray(twoD2);
    print("**********************");
    double[][] twoD3 = twoDDoubleArray(9, 5, 47.0, 99.0);
    printArray(twoD3);
  }
} /* Output:
47.00 49.17 51.33 53.50 55.67 57.83
60.00 62.17 64.33 66.50 68.67 70.83
73.00 75.17 77.33 79.50 81.67 83.83
86.00 88.17 90.33 92.50 94.67 96.83
**********************
47.00 60.00
73.00 86.00
**********************
47.00 48.16 49.31 50.47 51.62
52.78 53.93 55.09 56.24 57.40
58.56 59.71 60.87 62.02 63.18
```

```
64.33 65.49 66.64 67.80 68.96
70.11 71.27 72.42 73.58 74.73
75.89 77.04 78.20 79.36 80.51
81.67 82.82 83.98 85.13 86.29
87.44 88.60 89.76 90.91 92.07
93.22 94.38 95.53 96.69 97.84
*///:~
```

Calculate the step value for the initialization range by dividing the range by the number of elements in the array. You need a separate print routine, since **Arrays.deepToString( )** produces output of low perspicuity (double values are printed with varying numbers of decimals).

Exercise 4

```
//: arrays/E04_ThreeDDoubleArray.java
/****************** Exercise 4 *****************
 * Repeat the previous exercise for a
 * three-dimensional array.
 ***********************************************/
package arrays;
import java.util.Locale;
import static net.mindview.util.Print.*;

public class E04_ThreeDDoubleArray {
  public static double[][][] threeDDoubleArray(
    int xLen, int yLen, int zLen,
    double valStart, double valEnd) {
    double[][][] array = new double[xLen][yLen][zLen];
    double increment =
      (valEnd - valStart)/(xLen * yLen * zLen);
    double val = valStart;
    for(int i = 0; i < array.length; i++)
      for(int j = 0; j < array[i].length; j++)
        for(int k = 0; k < array[i][j].length; k++) {
          array[i][j][k] = val;
          val += increment;
        }
    return array;
  }
  public static void printArray(double[][][] array) {
    for(int i = 0; i < array.length; i++) {
      for(int j = 0; j < array[i].length; j++) {
        for(int k = 0; k < array[i][j].length; k++)
          System.out.printf(
            Locale.US, "%.2f ", array[i][j][k]);
```

```
        print();
      }
      print();
    }
  }
  public static void main(String args[]) {
    double[][][] threeD =
      threeDDoubleArray(4, 6, 2, 47.0, 99.0);
    printArray(threeD);
    print("**********************");
    double[][][] threeD2 =
      threeDDoubleArray(2, 2, 5, 47.0, 99.0);
    printArray(threeD2);
    print("**********************");
    double[][][] threeD3 =
      threeDDoubleArray(9, 5, 7, 47.0, 99.0);
    printArray(threeD3);
  }
} /* Output:
47.00 48.08
49.17 50.25
51.33 52.42
53.50 54.58
55.67 56.75
57.83 58.92

60.00 61.08
62.17 63.25
64.33 65.42
66.50 67.58
68.67 69.75
70.83 71.92

73.00 74.08
75.17 76.25
77.33 78.42
79.50 80.58
81.67 82.75
83.83 84.92

86.00 87.08
88.17 89.25
90.33 91.42
92.50 93.58
94.67 95.75
96.83 97.92
```

```
**********************
47.00 49.60 52.20 54.80 57.40
60.00 62.60 65.20 67.80 70.40

73.00 75.60 78.20 80.80 83.40
86.00 88.60 91.20 93.80 96.40

**********************
47.00 47.17 47.33 47.50 47.66 47.83 47.99
48.16 48.32 48.49 48.65 48.82 48.98 49.15
49.31 49.48 49.64 49.81 49.97 50.14 50.30
50.47 50.63 50.80 50.96 51.13 51.29 51.46
51.62 51.79 51.95 52.12 52.28 52.45 52.61

52.78 52.94 53.11 53.27 53.44 53.60 53.77
53.93 54.10 54.26 54.43 54.59 54.76 54.92
55.09 55.25 55.42 55.58 55.75 55.91 56.08
56.24 56.41 56.57 56.74 56.90 57.07 57.23
57.40 57.57 57.73 57.90 58.06 58.23 58.39

58.56 58.72 58.89 59.05 59.22 59.38 59.55
59.71 59.88 60.04 60.21 60.37 60.54 60.70
60.87 61.03 61.20 61.36 61.53 61.69 61.86
62.02 62.19 62.35 62.52 62.68 62.85 63.01
63.18 63.34 63.51 63.67 63.84 64.00 64.17

64.33 64.50 64.66 64.83 64.99 65.16 65.32
65.49 65.65 65.82 65.98 66.15 66.31 66.48
66.64 66.81 66.97 67.14 67.30 67.47 67.63
67.80 67.97 68.13 68.30 68.46 68.63 68.79
68.96 69.12 69.29 69.45 69.62 69.78 69.95

70.11 70.28 70.44 70.61 70.77 70.94 71.10
71.27 71.43 71.60 71.76 71.93 72.09 72.26
72.42 72.59 72.75 72.92 73.08 73.25 73.41
73.58 73.74 73.91 74.07 74.24 74.40 74.57
74.73 74.90 75.06 75.23 75.39 75.56 75.72

75.89 76.05 76.22 76.38 76.55 76.71 76.88
77.04 77.21 77.37 77.54 77.70 77.87 78.03
78.20 78.37 78.53 78.70 78.86 79.03 79.19
79.36 79.52 79.69 79.85 80.02 80.18 80.35
80.51 80.68 80.84 81.01 81.17 81.34 81.50

81.67 81.83 82.00 82.16 82.33 82.49 82.66
82.82 82.99 83.15 83.32 83.48 83.65 83.81
83.98 84.14 84.31 84.47 84.64 84.80 84.97
```

```
85.13 85.30 85.46 85.63 85.79 85.96 86.12
86.29 86.45 86.62 86.78 86.95 87.11 87.28

87.44 87.61 87.77 87.94 88.10 88.27 88.43
88.60 88.77 88.93 89.10 89.26 89.43 89.59
89.76 89.92 90.09 90.25 90.42 90.58 90.75
90.91 91.08 91.24 91.41 91.57 91.74 91.90
92.07 92.23 92.40 92.56 92.73 92.89 93.06

93.22 93.39 93.55 93.72 93.88 94.05 94.21
94.38 94.54 94.71 94.87 95.04 95.20 95.37
95.53 95.70 95.86 96.03 96.19 96.36 96.52
96.69 96.85 97.02 97.18 97.35 97.51 97.68
97.84 98.01 98.17 98.34 98.50 98.67 98.83
*///:~
```

As complex as this seems, it's still simpler than doing it in C or C++, and you get built-in array bounds checking. In C/C++, if you run off the end of a multi-dimensional array, you probably won't catch the error.

Exercise 5

```
//: arrays/E05_NonPrimitiveMultiDArray.java
/****************** Exercise 5 *****************
 * Demonstrate that multidimensional arrays of
 * non-primitive types are automatically initialized
 * to null.
 ***********************************************/
package arrays;
import java.util.Arrays;

public class E05_NonPrimitiveMultiDArray {
  public static void main(String[] args) {
    System.out.println(
      Arrays.deepToString(new Object[3][3][3]));
  }
} /* Output:
[[[null, null, null], [null, null, null], [null, null,
null]], [[null, null, null], [null, null, null], [null,
null, null]], [[null, null, null], [null, null, null],
[null, null, null]]]
*///:~
```

Exercise 6

```
//: arrays/E06_Filling2DArray.java
/****************** Exercise 6 *****************
 * Write a method that takes two int arguments,
 * indicating the two sizes of a 2-D array. The
 * method should create and fill a 2-D array of
 * BerylliumSphere according to the size arguments.
 ***********************************************/
package arrays;
import java.util.Arrays;

public class E06_Filling2DArray {
  static BerylliumSphere[][] fill(int xLen, int yLen) {
    BerylliumSphere[][] a = new BerylliumSphere[xLen][yLen];
    for(int i = 0; i < xLen; i++)
      for(int j = 0; j < yLen; j++)
        a[i][j] = new BerylliumSphere();
    return a;
  }
  public static void main(String[] args) {
    System.out.println(Arrays.deepToString(fill(3, 3)));
  }
} /* Output:
[[Sphere 0, Sphere 1, Sphere 2], [Sphere 3, Sphere 4, Sphere
5], [Sphere 6, Sphere 7, Sphere 8]]
*///:~
```

Exercise 7

```
//: arrays/E07_Filling3DArray.java
/****************** Exercise 7 *****************
 * Repeat the previous exercise for a 3-D array.
 ***********************************************/
package arrays;
import java.util.Arrays;

public class E07_Filling3DArray {
  static BerylliumSphere[][][]
  fill(int xLen, int yLen, int zLen) {
    BerylliumSphere[][][] a =
      new BerylliumSphere[xLen][yLen][zLen];
    for(int i = 0; i < xLen; i++)
      for(int j = 0; j < yLen; j++)
        for(int k = 0; k < zLen; k++)
```

```
          a[i][j][k] = new BerylliumSphere();
    return a;
  }
  public static void main(String[] args) {
    System.out.println(Arrays.deepToString(fill(3, 3, 3)));
  }
} /* Output:
[[[Sphere 0, Sphere 1, Sphere 2], [Sphere 3, Sphere 4,
Sphere 5], [Sphere 6, Sphere 7, Sphere 8]], [[Sphere 9,
Sphere 10, Sphere 11], [Sphere 12, Sphere 13, Sphere 14],
[Sphere 15, Sphere 16, Sphere 17]], [[Sphere 18, Sphere 19,
Sphere 20], [Sphere 21, Sphere 22, Sphere 23], [Sphere 24,
Sphere 25, Sphere 26]]]
*///:~
```

Exercise 8

```
//: arrays/E08_LeftToReader.java
/****************** Exercise 8 *****************
 * "Erasure gets in the way again—this example
 * attempts to create arrays of types that have been
 * erased, and are thus unknown types. Notice that
 * you can create an array of Object, and cast it,
 * but you get an “unchecked” warning at compile
 * time because the array doesn’t really hold or
 * dynamically check for type T. That is, if I create
 * a String[], Java will enforce at both compile time
 * and run time that I can only place String objects
 * in that array. However, if I create an Object[],
 * I can put anything except primitive types in that
 * array."
 *
 * Demonstrate the assertions in the previous
 * paragraph.
 ***********************************************/
package arrays;

public class E08_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
} ///:~
```

Exercise 9

```
//: arrays/E09_PeelBanana.java
/****************** Exercise 9 *****************
 * Create the classes necessary for the Peel<Banana>
 * example and show that the compiler doesn't accept
 * it. Fix the problem using an ArrayList.
 *********************************************/
package arrays;
import java.util.*;

class Banana {
  private final int id;
  Banana(int id) { this.id = id; }
  public String toString() {
    return getClass().getSimpleName() + " " + id;
  }
}

class Peel<T> {
  T fruit;
  Peel(T fruit) { this.fruit = fruit; }
  void peel() { System.out.println("Peeling "+ fruit); }
}

public class E09_PeelBanana {
  public static void main(String[] args) {
    // Compile error:
    //! Peel<Banana>[] a = new Peel<Banana>[10];
    ArrayList<Peel<Banana>> a =
      new ArrayList<Peel<Banana>>();
    for(int i = 0; i < 10; i++)
      a.add(new Peel<Banana>(new Banana(i)));
    for(Peel<Banana> p : a)
      p.peel();
  }
} /* Output:
Peeling Banana 0
Peeling Banana 1
Peeling Banana 2
Peeling Banana 3
Peeling Banana 4
Peeling Banana 5
Peeling Banana 6
Peeling Banana 7
Peeling Banana 8
```

```
Peeling Banana 9
*///:~
```

Exercise 10

```
//: arrays/E10_ArrayOfGenerics2.java
/****************** Exercise 10 *****************
 * Modify ArrayOfGenerics.java to use containers
 * instead of arrays. Show that you can eliminate
 * the compile-time warnings.
 ************************************************/
package arrays;
import java.util.*;

public class E10_ArrayOfGenerics2 {
  public static void main(String[] args) {
    ArrayList<List<String>> ls =
      new ArrayList<List<String>>();
    ls.add(new ArrayList<String>());
    // Compile-time checking produces an error:
    //! ls.add(new ArrayList<Integer>());
    ls.get(0).add("Array of Generics");
    System.out.println(ls.toString());
  }
} /* Output:
[[Array of Generics]]
*///:~
```

Exercise 11

```
//: arrays/E11_AutoboxingWithArrays.java
// {CompileTimeError}
/****************** Exercise 11 *****************
 * Show that autoboxing doesn't work with arrays.
 ************************************************/
package arrays;

public class E11_AutoboxingWithArrays {
  public static void main(String[] args) {
    int[] pa = { 1, 2, 3, 4, 5 };
    Integer[] wa = pa;
    Integer[] wb = { 1, 2, 3, 4, 5 };
    int[] pb = wb;
  }
} ///:~
```

Exercise 12

```
//: arrays/E12_GeneratedDArray.java
/****************** Exercise 12 *****************
 * Create an initialized array of double using
 * CountingGenerator. Print the results.
 ************************************************/
package arrays;
import java.util.Arrays;
import net.mindview.util.*;

public class E12_GeneratedDArray {
  public static void main(String[] args) {
    double[] d = ConvertTo.primitive(Generated.array(
      Double.class, new CountingGenerator.Double(), 15));
    System.out.println(Arrays.toString(d));
  }
} /* Output:
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0,
11.0, 12.0, 13.0, 14.0]
*///:~
```

Exercise 13

```
//: arrays/E13_StringFill.java
/****************** Exercise 13 *****************
 * Fill a String using CountingGenerator.Character.
 ************************************************/
package arrays;
import net.mindview.util.*;

public class E13_StringFill {
  public static void main(String[] args) {
    String s = new CountingGenerator.String(15).next();
    System.out.println(s);
  }
} /* Output:
abcdefghijklmno
*///:~
```

Since **CountingGenerator.String** automatically uses **CountingGenerator.Character**, we just use the former class's **next( )** method to perform the task.

Exercise 14

```
//: arrays/E14_PrimitiveArraysFill.java
/****************** Exercise 14 *****************
 * Create an array of each primitive type, then
 * fill each array by using CountingGenerator.
 * Print each array.
 ************************************************/
package arrays;
import java.lang.reflect.Array;
import java.util.Arrays;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class Fill {
  public static void
  primitive(Object array, Generator<?> gen) {
    int size = Array.getLength(array);
    Class<?> type = array.getClass().getComponentType();
    for(int i = 0; i < size; i++)
      if(type == Boolean.TYPE)
        Array.setBoolean(array, i, (Boolean)gen.next());
      else if (type == Byte.TYPE)
        Array.setByte(array, i, (Byte)gen.next());
      else if (type == Short.TYPE)
        Array.setShort(array, i, (Short)gen.next());
      else if (type == Integer.TYPE)
        Array.setInt(array, i, (Integer)gen.next());
      else if (type == Character.TYPE)
        Array.setChar(array, i, (Character)gen.next());
      else if (type == Float.TYPE)
        Array.setFloat(array, i, (Float)gen.next());
      else if (type == Double.TYPE)
        Array.setDouble(array, i, (Double)gen.next());
      else if (type == Long.TYPE)
        Array.setLong(array, i, (Long)gen.next());
  }
}

public class E14_PrimitiveArraysFill {
  public static void main(String[] args) {
    int size = 6;
    // First create all primitive arrays
    boolean[] a1 = new boolean[size];
    byte[] a2 = new byte[size];
    char[] a3 = new char[size];
```

```
    short[] a4 = new short[size];
    int[] a5 = new int[size];
    long[] a6 = new long[size];
    float[] a7 = new float[size];
    double[] a8 = new double[size];
    // Now fill them using a matching generator
    Fill.primitive(a1, new CountingGenerator.Boolean());
    print("a1 = " + Arrays.toString(a1));
    Fill.primitive(a2, new CountingGenerator.Byte());
    print("a2 = " + Arrays.toString(a2));
    Fill.primitive(a3, new CountingGenerator.Character());
    print("a3 = " + Arrays.toString(a3));
    Fill.primitive(a4, new CountingGenerator.Short());
    print("a4 = " + Arrays.toString(a4));
    Fill.primitive(a5, new CountingGenerator.Integer());
    print("a5 = " + Arrays.toString(a5));
    Fill.primitive(a6, new CountingGenerator.Long());
    print("a6 = " + Arrays.toString(a6));
    Fill.primitive(a7, new CountingGenerator.Float());
    print("a7 = " + Arrays.toString(a7));
    Fill.primitive(a8, new CountingGenerator.Double());
    print("a8 = " + Arrays.toString(a8));
  }
} /* Output:
a1 = [true, false, true, false, true, false]
a2 = [0, 1, 2, 3, 4, 5]
a3 = [a, b, c, d, e, f]
a4 = [0, 1, 2, 3, 4, 5]
a5 = [0, 1, 2, 3, 4, 5]
a6 = [0, 1, 2, 3, 4, 5]
a7 = [0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
a8 = [0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
*///:~
```

We use **Fill.primitive( )** to fill a primitive array with data from the appropriate **Generator**. There are other ways to solve the exercise, but this one highlights Java's dynamicity. It also illustrates how powerful Java can be if you leverage its generics and reflection features.

Note that you cannot invoke the overloaded **Generated.array( )** method directly, because it does not support primitive types and autoboxing does not work with arrays.

Exercise 15

```
//: arrays/E15_BSContainerComparison.java
```

```
/****************** Exercise 15 *****************
 * Modify ContainerComparison.java by creating a
 * Generator for BerylliumSphere, and change main()
 * to use that Generator with Generated.array().
 ************************************************/
package arrays;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class BSGenerator implements Generator<BerylliumSphere> {
  public BerylliumSphere next() {
    return new BerylliumSphere();
  }
}

public class E15_BSContainerComparison {
  public static void main(String[] args) {
    BSGenerator gen = new BSGenerator();
    BerylliumSphere[] spheres = Generated.array(
      BerylliumSphere.class, gen, 5);
    print(Arrays.toString(spheres));
    print(spheres[4]);

    List<BerylliumSphere> sphereList = Arrays.asList(
      Generated.array(BerylliumSphere.class, gen, 5));
    print(sphereList);
    print(sphereList.get(4));

    int[] integers = { 0, 1, 2, 3, 4, 5 };
    print(Arrays.toString(integers));
    print(integers[4]);

    List<Integer> intList = new ArrayList<Integer>(
      Arrays.asList(0, 1, 2, 3, 4, 5));
    intList.add(97);
    print(intList);
    print(intList.get(4));
  }
} /* Output:
[Sphere 0, Sphere 1, Sphere 2, Sphere 3, Sphere 4]
Sphere 4
[Sphere 5, Sphere 6, Sphere 7, Sphere 8, Sphere 9]
Sphere 9
[0, 1, 2, 3, 4, 5]
4
[0, 1, 2, 3, 4, 5, 97]
```

```
4
*///:~
```

Exercise 16

```
//: arrays/E16_SkipGenerator.java
/****************** Exercise 16 *****************
 * Starting with CountingGenerator.java, create a
 * SkipGenerator class that produces new values by
 * incrementing according to a constructor argument.
 * Modify TestArrayGeneration.java to show that your
 * new class works correctly.
 ***********************************************/
package arrays;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class SkipGenerator {
  public static class
  Boolean implements Generator<java.lang.Boolean> {
    private boolean value;
    private boolean step;
    public Boolean(boolean step) { this.step = step; }
    public java.lang.Boolean next() {
      value = step ? !value : value;
      return value;
    }
  }
  public static class
  Byte implements Generator<java.lang.Byte> {
    private byte value;
    private byte step;
    public Byte(byte step) { this.step = step; }
    public java.lang.Byte next() {
      byte oldValue = value;
      value += step;
      return oldValue;
    }
  }
  static char[] chars = ("abcdefghijklmnopqrstuvwxyz" +
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ").toCharArray();
  public static class
  Character implements Generator<java.lang.Character> {
    int index;
    private int step;
```

```
    public Character(int step) { this.step = step; }
    public java.lang.Character next() {
      char oldValue = chars[index];
      index = (index + step) % chars.length;
      return oldValue;
    }
  }
  public static class
  String implements Generator<java.lang.String> {
    private int length;
    Generator<java.lang.Character> cg;
    public String(int step) { this(step, 7); }
    public String(int step, int length) {
      this.length = length;
      cg = new Character(step);
    }
    public java.lang.String next() {
      char[] buf = new char[length];
      for(int i = 0; i < length; i++)
        buf[i] = cg.next();
      return new java.lang.String(buf);
    }
  }
  public static class
  Short implements Generator<java.lang.Short> {
    private short value;
    private short step;
    public Short(short step) { this.step = step; }
    public java.lang.Short next() {
      short oldValue = value;
      value += step;
      return oldValue;
    }
  }
  public static class
  Integer implements Generator<java.lang.Integer> {
    private int value;
    private int step;
    public Integer(int step) { this.step = step; }
    public java.lang.Integer next() {
      int oldValue = value;
      value += step;
      return oldValue;
    }
  }
  public static class
  Long implements Generator<java.lang.Long> {
```

```
    private long value;
    private long step;
    public Long(long step) { this.step = step; }
    public java.lang.Long next() {
      long oldValue = value;
      value += step;
      return oldValue;
    }
  }
  public static class
  Float implements Generator<java.lang.Float> {
    private float value;
    private float step;
    public Float(float step) { this.step = step; }
    public java.lang.Float next() {
      float oldValue = value;
      value += step;
      return oldValue;
    }
  }
  public static class
  Double implements Generator<java.lang.Double> {
    private double value;
    private double step;
    public Double(double step) { this.step = step; }
    public java.lang.Double next() {
      double oldValue = value;
      value += step;
      return oldValue;
    }
  }
}

public class E16_SkipGenerator {
  public static void main(String[] args) {
    boolean[] a1 = ConvertTo.primitive(Generated.array(
      Boolean.class, new SkipGenerator.Boolean(true), 6));
    print("a1 = " + Arrays.toString(a1));
    byte[] a2 = ConvertTo.primitive(Generated.array(
      Byte.class, new SkipGenerator.Byte((byte)1), 6));
    print("a2 = " + Arrays.toString(a2));
    char[] a3 = ConvertTo.primitive(Generated.array(
      Character.class,
      new SkipGenerator.Character(2), 6));
    print("a3 = " + Arrays.toString(a3));
    short[] a4 = ConvertTo.primitive(Generated.array(
      Short.class, new SkipGenerator.Short((short)3), 6));
```

```
    print("a4 = " + Arrays.toString(a4));
    int[] a5 = ConvertTo.primitive(Generated.array(
      Integer.class, new SkipGenerator.Integer(4), 6));
    print("a5 = " + Arrays.toString(a5));
    long[] a6 = ConvertTo.primitive(Generated.array(
      Long.class, new SkipGenerator.Long(5l), 6));
    print("a6 = " + Arrays.toString(a6));
    float[] a7 = ConvertTo.primitive(Generated.array(
      Float.class, new SkipGenerator.Float(1.5f), 6));
    print("a7 = " + Arrays.toString(a7));
    double[] a8 = ConvertTo.primitive(Generated.array(
      Double.class, new SkipGenerator.Double(2.0), 6));
    print("a8 = " + Arrays.toString(a8));
  }
} /* Output:
a1 = [true, false, true, false, true, false]
a2 = [0, 1, 2, 3, 4, 5]
a3 = [a, c, e, g, i, k]
a4 = [0, 3, 6, 9, 12, 15]
a5 = [0, 4, 8, 12, 16, 20]
a6 = [0, 5, 10, 15, 20, 25]
a7 = [0.0, 1.5, 3.0, 4.5, 6.0, 7.5]
a8 = [0.0, 2.0, 4.0, 6.0, 8.0, 10.0]
*///:~
```

Exercise 17

```
//: arrays/E17_BigDecimalGenerator.java
/****************** Exercise 17 *****************
 * Create and test a Generator for BigDecimal, and
 * ensure that it works with the Generated methods.
 ************************************************/
package arrays;
import java.math.*;
import java.util.*;
import net.mindview.util.*;

class BigDecimalGenerator implements Generator<BigDecimal> {
  private BigDecimal value = new BigDecimal(0);
  private BigDecimal step;
  BigDecimalGenerator(BigDecimal step) { this.step = step; }
  public BigDecimal next() {
    BigDecimal oldValue = value;
    value = value.add(step);
    return oldValue;
  }
```

```
}

public class E17_BigDecimalGenerator {
  public static void main(String[] args) {
    BigDecimal[] a = { new BigDecimal(9), new BigDecimal(8),
      new BigDecimal(7), new BigDecimal(6) };
    System.out.println(Arrays.toString(a));
    a = Generated.array(a, new BigDecimalGenerator(
      new BigDecimal("0.1")));
    System.out.println(Arrays.toString(a));
    BigDecimal[] b = Generated.array(BigDecimal.class,
      new BigDecimalGenerator(new BigDecimal("0.2")), 15);
    System.out.println(Arrays.toString(b));
  }
} /* Output:
[9, 8, 7, 6]
[0, 0.1, 0.2, 0.3]
[0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2,
2.4, 2.6, 2.8]
*///:~
```

Exercise 18

```
//: arrays/E18_ArrayCopy.java
/****************** Exercise 18 *****************
 * Create and fill an array of BerylliumSphere.
 * Copy this array to a new array and show that it's
 * a shallow copy.
 ************************************************/
package arrays;
import java.util.*;
import java.lang.reflect.*;
import static net.mindview.util.Print.*;

public class E18_ArrayCopy {
  // BerylliumSphere.id is private, so we need to use
  // reflection to alter its value.
  static void setID(BerylliumSphere bs, long value) {
    try {
      Field fid =
        BerylliumSphere.class.getDeclaredField("id");
      fid.setAccessible(true);
      fid.setLong(bs, value);
    } catch(Exception e) {
      e.printStackTrace();
    }
```

```
  }
  public static void main(String[] args) {
    BerylliumSphere[] a = new BerylliumSphere[4];
    Arrays.fill(a, new BerylliumSphere());
    BerylliumSphere[] b = new BerylliumSphere[4];
    print("a = " + Arrays.toString(a));
    print("b = " + Arrays.toString(b));
    System.arraycopy(a, 0, b, 0, b.length);
    print("b = " + Arrays.toString(b));
    // Changing a reference in 'a' will not impact 'b'.
    a[1] = a[3] = new BerylliumSphere();
    print("a = " + Arrays.toString(a));
    print("b = " + Arrays.toString(b));
    // Changing an object's state will impact 'b', as well.
    setID(a[0], -1L);
    print("a = " + Arrays.toString(a));
    print("b = " + Arrays.toString(b));
  }
} /* Output:
a = [Sphere 0, Sphere 0, Sphere 0, Sphere 0]
b = [null, null, null, null]
b = [Sphere 0, Sphere 0, Sphere 0, Sphere 0]
a = [Sphere 0, Sphere 1, Sphere 0, Sphere 1]
b = [Sphere 0, Sphere 0, Sphere 0, Sphere 0]
a = [Sphere -1, Sphere 1, Sphere -1, Sphere 1]
b = [Sphere -1, Sphere -1, Sphere -1, Sphere -1]
*///:~
```

Exercise 19

```
//: arrays/E19_ArrayEquals.java
/****************** Exercise 19 *****************
 * Create a class with an int field that's initialized
 * from a constructor argument. Create two arrays
 * of these objects, using identical initialization
 * values for each array, and show that Arrays.equals()
 * says that they are unequal. Add an equals() method
 * to your class to fix the problem.
 ************************************************/
package arrays;
import java.util.*;

class DataHolder {
  protected int data;
  DataHolder(int data) { this.data = data; }
}
```

```
class DataHolderWithEquals extends DataHolder {
  DataHolderWithEquals(int data) { super(data); }
  public boolean equals(Object o) {
    return o instanceof DataHolderWithEquals &&
           data == ((DataHolder)o).data;
  }
}

public class E19_ArrayEquals {
  public static void main(String[] args) {
    DataHolder[] a1 = new DataHolder[5];
    DataHolder[] a2 = new DataHolder[5];
    Arrays.fill(a1, new DataHolder(1));
    Arrays.fill(a2, new DataHolder(1));
    System.out.println(Arrays.equals(a1, a2));
    Arrays.fill(a1, new DataHolderWithEquals(1));
    Arrays.fill(a2, new DataHolderWithEquals(1));
    System.out.println(Arrays.equals(a1, a2));
  }
} /* Output:
false
true
*///:~
```

Exercise 20

```
//: arrays/E20_ArrayDeepEquals.java
/****************** Exercise 20 *****************
 * Demonstrate deepEquals() for multidimensional
 * arrays.
 ************************************************/
package arrays;
import java.util.*;

public class E20_ArrayDeepEquals {
  public static void main(String[] args) {
    int[][] table1 =
      {{1, 2, 3},
       {4, 5},
       {7, 8, 9, 10}};
    int[][] table2 =
      {{1, 2, 3},
       {4, 5},
       {7, 8, 9, 10}};
    Integer[][] table3 =
```

```
      {{1, 2, 3},
       {4, 5},
       {7, 8, 9, 10}};
    int[][] table4 =
      {{1, 2, 3},
       {6, 5, 4},
       {7, 8}};
    System.out.println(Arrays.deepEquals(table1, table2));
    System.out.println(Arrays.deepEquals(table1, table3));
    System.out.println(Arrays.deepEquals(table1, table4));

    // Let us check manually for equality between table1 and
    // table3
    boolean res = true;
exit_loop:
    for(int i = 0; i < table1.length; i++)
      for(int j = 0; j < table1[i].length; j++)
        if(table1[i][j] != table3[i][j]) {
          res = false;
          break exit_loop;
        }
    System.out.println(res);
  }
} /* Output:
true
false
false
true
*///:~
```

Arrays.deepEquals( ) does not report **table1** and **table3** as equal, though they are semantically the same, because the compiler sees them as different. However, when we compare the two arrays "manually," transparent autoboxing produces the desired outcome.

Exercise 21

```
//: arrays/E21_ArraySort.java
/****************** Exercise 21 *****************
 * Try to sort an array of the objects in Exercise
 * 18. Implement Comparable to fix the problem. Now
 * create a Comparator to sort the objects into reverse
 * order.
 ************************************************/
package arrays;
import java.util.*;
```

```
import java.lang.reflect.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class ComparableBerylliumSphere extends BerylliumSphere
implements Comparable<BerylliumSphere> {
  // BerylliumSphere.id is private, so we need to use
  // reflection to get its value.
  static long getID(BerylliumSphere bs) {
    try {
      Field fid =
        BerylliumSphere.class.getDeclaredField("id");
      fid.setAccessible(true);
      return fid.getLong(bs);
    } catch(Exception e) {
      e.printStackTrace();
      return 0L;  // Bogus value
    }
  }
  public int compareTo(BerylliumSphere rv) {
    long id = getID(this);
    long rvid = getID(rv);
    return (id < rvid ? -1 : (id == rvid ? 0 : 1));
  }
}

public class E21_ArraySort {
  public static void main(String[] args) {
    Random r = new Random(47);
    BerylliumSphere[] a = Generated.array(
      BerylliumSphere.class, new BSGenerator(), 5);
    Collections.shuffle(Arrays.asList(a), r);
    print("Before sort 1: a = " + Arrays.toString(a));
    try {
      Arrays.sort(a);
      print("After sort 1: a = " + Arrays.toString(a));
    } catch(ClassCastException e) {
      System.out.println("Array cannot be sorted!");
    }
    for(int i = 0; i < a.length; i++)
      a[i] = new ComparableBerylliumSphere();
    Collections.shuffle(Arrays.asList(a), r);
    print("Before sort 2: a = " + Arrays.toString(a));
    Arrays.sort(a);
    print("After sort 2: a = " + Arrays.toString(a));
    Collections.shuffle(Arrays.asList(a), r);
    print("Before rev. sort 3: a = " + Arrays.toString(a));
```

```
    Arrays.sort(a, Collections.reverseOrder());
    print("After rev. sort 3: a = " + Arrays.toString(a));
  }
} /* Output:
Before sort 1: a = [Sphere 2, Sphere 0, Sphere 4, Sphere 1,
Sphere 3]
Array cannot be sorted!
Before sort 2: a = [Sphere 8, Sphere 5, Sphere 9, Sphere 7,
Sphere 6]
After sort 2: a = [Sphere 5, Sphere 6, Sphere 7, Sphere 8,
Sphere 9]
Before rev. sort 3: a = [Sphere 8, Sphere 6, Sphere 5,
Sphere 9, Sphere 7]
After rev. sort 3: a = [Sphere 9, Sphere 8, Sphere 7, Sphere
6, Sphere 5]
*///:~
```

Sorting an array of objects without the **Comparable** interface triggers **ClassCastException** (as the output shows).

Exercise 22

```
//: arrays/E22_ArrayBinarySearch.java
/****************** Exercise 22 *****************
 * Show that the results of performing a
 * binarySearch() on an unsorted array are
 * unpredictable.
 ************************************************/
package arrays;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E22_ArrayBinarySearch {
  public static void main(String[] args) {
    Generator<Integer> gen =
      new RandomGenerator.Integer(1000);
    int[] a = ConvertTo.primitive(
      Generated.array(new Integer[25], gen));
    print("Unsorted array: " + Arrays.toString(a));
    int location = Arrays.binarySearch(a, a[10]);
    printnb("Location of " + a[10] + " is " + location);
    if(location >= 0)
      print(", a[" + location + "] = " + a[location]);
    else
      print();
```

```
    location = Arrays.binarySearch(a, a[5]);
    printnb("Location of " + a[5] + " is " + location);
    if(location >= 0)
      print(", a[" + location + "] = " + a[location]);
  }
} /* Output:
Unsorted array: [258, 555, 693, 861, 961, 429, 868, 200,
522, 207, 288, 128, 551, 589, 809, 278, 998, 861, 520, 258,
916, 140, 511, 322, 704]
Location of 288 is -2
Location of 429 is 5, a[5] = 429
*///:~
```

Clearly, you cannot trust a binary search of an unsorted array. Our search finds the sixth element but not the eleventh.

Exercise 23

```
//: arrays/E23_ArraySort2.java
/****************** Exercise 23 *****************
 * Create an array of Integer, fill it with random
 * int values (using autoboxing), and sort it into
 * reverse order using a Comparator.
 ************************************************/
package arrays;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E23_ArraySort2 {
  public static void main(String[] args) {
    Generator<Integer> gen =
      new RandomGenerator.Integer(1000);
    Integer[] a = Generated.array(new Integer[25], gen);
    print("Unsorted array: " + Arrays.toString(a));
    Arrays.sort(a, Collections.reverseOrder());
    print("Sorted array: " + Arrays.toString(a));
  }
} /* Output:
Unsorted array: [258, 555, 693, 861, 961, 429, 868, 200,
522, 207, 288, 128, 551, 589, 809, 278, 998, 861, 520, 258,
916, 140, 511, 322, 704]
Sorted array: [998, 961, 916, 868, 861, 861, 809, 704, 693,
589, 555, 551, 522, 520, 511, 429, 322, 288, 278, 258, 258,
207, 200, 140, 128]
*///:~
```

Primitive arrays do not allow sorting with a **Comparator**, but arrays filled with wrapper objects do. Note that autoboxing from **int** to **Integer** happens behind the scenes in the **RandomGenerator.Integer** class.

Exercise 24

```
//: arrays/E24_ArraySearch.java
/****************** Exercise 24 *****************
 * Show that the class from Exercise 19 can be
 * searched.
 ************************************************/
package arrays;
import java.util.*;
import static net.mindview.util.Print.*;

public class E24_ArraySearch {
  public static void main(String[] args) {
    Comparator<DataHolder> comp =
      new Comparator<DataHolder>() {
        public int compare(DataHolder o1, DataHolder o2) {
          return (o1.data < o2.data ? -1 :
                  (o1.data == o2.data ? 0 : 1));
        }
      };
    DataHolder[] a = new DataHolderWithEquals[10];
    for(int i = 0; i < a.length; i++)
      a[i] = new DataHolderWithEquals(i);
    Arrays.sort(a, comp);
    int location = Arrays.binarySearch(a, a[7], comp);
    printnb("Location of " + a[7] + " is " + location);
    if(location >= 0)
      print(", a[" + location + "] = " + a[location]);
    else
      print();
    location = Arrays.binarySearch(a, a[5], comp);
    printnb("Location of " + a[5] + " is " + location);
    if(location >= 0)
      print(", a[" + location + "] = " + a[location]);
  }
} /* Output: (83% match)
Location of arrays.DataHolderWithEquals@a90653 is 7, a[7] =
arrays.DataHolderWithEquals@a90653
Location of arrays.DataHolderWithEquals@de6ced is 5, a[5] =
arrays.DataHolderWithEquals@de6ced
*///:~
```

You must supply the proper **Comparator** to perform the search in **Arrays.sort( )** and **Arrays.binarySearch( )**.

You must, at a minimum, use the **equals( )** method to make a class searchable. To search for a specific instance of that class you can use **indexOf( )** method (part of the **List** interface), though it isn't very efficient. It iterates over the elements in the collection, checking each element in turn for equality with the specified element—a cumbersome method for large collections.

Exercise 25

```
//: arrays/E25_PythonLists.java
/****************** Exercise 25 *****************
 * Rewrite PythonLists.py in Java.
 ************************************************/
package arrays;
import java.util.*;
import static net.mindview.util.Print.*;

public class E25_PythonLists {
  public static void main(String[] args) {
    List<Integer> aList =
      new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));
    print(aList.getClass().getSimpleName());
    print(aList);
    print(aList.get(4));
    aList.add(6);
    aList.addAll(Arrays.asList(7, 8));
    print(aList);
    List<Integer> aSlice = aList.subList(2, 4);
    print(aSlice);
    class MyArrayList<T> extends ArrayList<T> {
      MyArrayList(Collection<? extends T> c) { super(c); }
      MyArrayList(int initialCapacity) {
        super(initialCapacity);
      }
      List<T> getReversed() {
        List<T> reversed = new MyArrayList<T>(size());
        ListIterator<T> it = listIterator(size());
        while(it.hasPrevious())
          reversed.add(it.previous());
        return reversed;
      }
    }
    MyArrayList<Integer> list2 =
```

```
      new MyArrayList<Integer>(aList);
    print(list2.getClass().getSimpleName());
    print(list2.getReversed());
  }
} /* Output:
ArrayList
[1, 2, 3, 4, 5]
5
[1, 2, 3, 4, 5, 6, 7, 8]
[3, 4]
MyArrayList
[8, 7, 6, 5, 4, 3, 2, 1]
*///:~
```

The sophisticated collection classes in **java.util** make the conversion trivial. Python has a more concise way of expressing this; for example, you don't manually code the different constructors, but rather inherit them automatically (unlike in Java).

Containers in Depth

Exercise 1

```
//: containers/E01_CountryList.java
/****************** Exercise 1 ******************
 * Create a List (try both ArrayList and LinkedList)
 * and fill it using Countries. Sort the list and
 * print it, then apply Collections.shuffle() to the
 * list repeatedly, printing it each time so you
 * can see how the shuffle() method randomizes the
 * list differently each time.
 *********************************************/
package containers;
import java.util.*;
import static net.mindview.util.Print.*;
import static net.mindview.util.Countries.*;

public class E01_CountryList {
  private static Random rnd = new Random(47);
  public static void main(String[] args) {
    List<String> l = new ArrayList<String>(names(8));
    Collections.sort(l);
    print("sorted: " + l);
    for(int i = 1; i < 5; i++) {
      Collections.shuffle(l, rnd);
      print("shuffled (" + i + "): " + l);
    }
  }
} /* Output:
sorted: [ALGERIA, ANGOLA, BENIN, BOTSWANA, BURKINA FASO,
BURUNDI, CAMEROON, CAPE VERDE]
shuffled (1): [BURKINA FASO, BOTSWANA, CAMEROON, ALGERIA,
CAPE VERDE, ANGOLA, BENIN, BURUNDI]
shuffled (2): [BENIN, CAMEROON, BURKINA FASO, CAPE VERDE,
BURUNDI, BOTSWANA, ANGOLA, ALGERIA]
shuffled (3): [ALGERIA, BURKINA FASO, BOTSWANA, ANGOLA,
CAMEROON, BURUNDI, CAPE VERDE, BENIN]
shuffled (4): [BENIN, ANGOLA, BOTSWANA, BURUNDI, CAMEROON,
CAPE VERDE, ALGERIA, BURKINA FASO]
*///:~
```

We solve the problem for **ArrayList**, and you can change the solution in only one place to solve it using **LinkedList**.

Exercise 2

```
//: containers/E02_ACountries.java
/****************** Exercise 2 ******************
 * Produce a Map and a Set containing all the
 * countries that begin with 'A'.
 ************************************************/
package containers;
import java.util.*;
import static net.mindview.util.Countries.*;

public class E02_ACountries {
  public static void main(String[] args) {
    TreeMap<String,String> map =
      new TreeMap<String,String>(capitals());
    TreeSet<String> set = new TreeSet<String>(names());
    String b = null;
    for(String s : map.keySet())
      if(s.startsWith("B")) {
        b = s;
        break;
      }
    Map<String,String> aMap = map.headMap(b);
    Set<String> aSet = set.headSet(b);
    System.out.println("aMap = " + aMap);
    System.out.println("aSet = " + aSet);
  }
} /* Output:
aMap = {AFGHANISTAN=Kabul, ALBANIA=Tirana, ALGERIA=Algiers,
ANDORRA=Andorra la Vella, ANGOLA=Luanda, ANTIGUA AND
BARBUDA=Saint John's, ARGENTINA=Buenos Aires,
ARMENIA=Yerevan, AUSTRALIA=Canberra, AUSTRIA=Vienna,
AZERBAIJAN=Baku}
aSet = [AFGHANISTAN, ALBANIA, ALGERIA, ANDORRA, ANGOLA,
ANTIGUA AND BARBUDA, ARGENTINA, ARMENIA, AUSTRALIA, AUSTRIA,
AZERBAIJAN]
*///:~
```

You solve the problem easily by recognizing that the compiler sorts **Map** and **Set** trees automatically, filling both with the country information. Then it finds the first string that doesn't begin with 'A,' using an iterator from either container (since both sort themselves). The methods **headMap( )** and **headSet( )**

(belonging to **TreeMap** and **TreeSet**) create subsets using this pattern. You can find descriptions of both in the JDK HTML documentation.

Exercise 3

```
//: containers/E03_VerifySet.java
/****************** Exercise 3 ******************
 * Using Countries, fill a Set with the same data
 * multiple times, then verify that the Set ends up
 * with only one of each instance. Try this with
 * HashSet, LinkedHashSet, and TreeSet.
 ***********************************************/
package containers;
import java.util.*;
import static net.mindview.util.Countries.*;

public class E03_VerifySet {
  public static void main(String[] args) {
    Set<String> set = new LinkedHashSet<String>();
    for(int i = 0; i < 5; i++)
      set.addAll(names(10));
    System.out.println(set);
  }
} /* Output:
[ALGERIA, ANGOLA, BENIN, BOTSWANA, BURKINA FASO, BURUNDI,
CAMEROON, CAPE VERDE, CENTRAL AFRICAN REPUBLIC, CHAD]
*///:~
```

Note that the output lists each country only once. We solve for **LinkedHashSet**, but just one change in the code solves for **HashSet** and **TreeSet**. The output differs because each **Set** orders items differently.

Exercise 4

TIJ4 already provides a solution for this exercise (see **holding/UniqueWords.java** program from the *Holding Your Objects* chapter). There, the compiler hands **TextFile**'s data to the **TreeSet** constructor, which adds the contents of the **List** to itself.

Exercise 5

```
//: containers/E05_CountingMapData2.java
/****************** Exercise 5 ******************
```

```
 * Modify CountingMapData.java to fully implement
 * the flyweight by adding a custom EntrySet class
 * like the one in Countries.java.
 *********************************************/
package containers;
import java.util.*;

class CountingMapData extends AbstractMap<Integer,String> {
  private int size;
  private static String[] chars =
    "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z"
    .split(" ");
  public CountingMapData(int size) {
    if(size < 0) this.size = 0;
    this.size = size;
  }
  private class Entry implements Map.Entry<Integer,String> {
    int index;
    Entry(int index) { this.index = index; }
    public boolean equals(Object o) {
      return o instanceof Entry &&
             index == ((Entry)o).index;
    }
    public Integer getKey() { return index; }
    public String getValue() {
      return
        chars[index % chars.length] +
        Integer.toString(index / chars.length);
    }
    public String setValue(String value) {
      throw new UnsupportedOperationException();
    }
    public int hashCode() {
      return Integer.valueOf(index).hashCode();
    }
  }
  class EntrySet
  extends AbstractSet<Map.Entry<Integer,String>> {
    public int size() { return size; }
    private class Iter
    implements Iterator<Map.Entry<Integer,String>> {
      private Entry entry = new Entry(-1);
      public boolean hasNext() {
        return entry.index < size - 1;
      }
      public Map.Entry<Integer,String> next() {
        entry.index++;
```

```
        return entry;
      }
      public void remove() {
        throw new UnsupportedOperationException();
      }
    }
    public
    Iterator<Map.Entry<Integer,String>> iterator() {
      return new Iter();
    }
  }
  private Set<Map.Entry<Integer,String>> entries =
    new EntrySet();
  public Set<Map.Entry<Integer,String>> entrySet() {
    return entries;
  }
}

public class E05_CountingMapData2 {
  public static void main(String[] args) {
    System.out.println(new CountingMapData(60));
  }
} /* Output:
{0=A0, 1=B0, 2=C0, 3=D0, 4=E0, 5=F0, 6=G0, 7=H0, 8=I0, 9=J0,
10=K0, 11=L0, 12=M0, 13=N0, 14=O0, 15=P0, 16=Q0, 17=R0,
18=S0, 19=T0, 20=U0, 21=V0, 22=W0, 23=X0, 24=Y0, 25=Z0,
26=A1, 27=B1, 28=C1, 29=D1, 30=E1, 31=F1, 32=G1, 33=H1,
34=I1, 35=J1, 36=K1, 37=L1, 38=M1, 39=N1, 40=O1, 41=P1,
42=Q1, 43=R1, 44=S1, 45=T1, 46=U1, 47=V1, 48=W1, 49=X1,
50=Y1, 51=Z1, 52=A2, 53=B2, 54=C2, 55=D2, 56=E2, 57=F2,
58=G2, 59=H2}
*///:~
```

Entry and **EntrySet** are now inner classes and not nested ones (like in **Countries.java**). **EntrySet** refers to the **size** instance field defined in its parent class, so you don't need a constructor to specify the size of the returned chunk of data.

Exercise 6

```
//: containers/E06_UnsupportedList.java
/****************** Exercise 6 ******************
 * Note that List has additional "optional"
 * operations that are not included in Collection.
 * Write a version of Unsupported.java that tests
 * these additional optional operations.
```

```
 *************************************************/
package containers;
import java.util.*;

public class E06_UnsupportedList {
  static void test(String msg, List<String> l) {
    System.out.println("--- " + msg + " ---");
    List<String> subList =
      new ArrayList<String>(l.subList(1,8));
    try { l.add(0, "Test"); } catch(Exception e) {
      System.out.println("add(index, element): " + e);
    }
    try { l.addAll(0, subList); } catch(Exception e) {
      System.out.println("addAll(index, c): " + e);
    }
    try { l.remove(0); } catch(Exception e) {
      System.out.println("remove(index): " + e);
    }
  }
  public static void main(String[] args) {
    List<String> list =
      Arrays.asList("A B C D E F G H I J K L".split(" "));
    test("Modifiable Copy", new ArrayList<String>(list));
    test("Arrays.asList()", list);
    test("unmodifiableList()",
      Collections.unmodifiableList(
        new ArrayList<String>(list)));
  }
} /* Output:
--- Modifiable Copy ---
--- Arrays.asList() ---
add(index, element): java.lang.UnsupportedOperationException
addAll(index, c): java.lang.UnsupportedOperationException
remove(index): java.lang.UnsupportedOperationException
--- unmodifiableList() ---
add(index, element): java.lang.UnsupportedOperationException
addAll(index, c): java.lang.UnsupportedOperationException
remove(index): java.lang.UnsupportedOperationException
*///:~
```

Since we already presented **set( )** in **Unsupported.java**, this program just demonstrates those overloaded methods that cannot be found in the **Collections** interface.

Exercise 7

```
//: containers/E07_CrossInsertion.java
/****************** Exercise 7 *****************
 * Create both an ArrayList and a LinkedList, and
 * fill each using the Countries.names() generator.
 * Print each list using an ordinary Iterator, then
 * insert one list into the other by using a
 * ListIterator, inserting at every other location.
 * Now perform the insertion starting at the end of
 * the first list and moving backward.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E07_CrossInsertion {
  public static void main(String args[]) {
    ArrayList<String> al =
      new ArrayList<String>(Countries.names(10));
    LinkedList<String> ll =
      new LinkedList<String>(
        Countries.names(20).subList(10, 20));
    for(Iterator<String> it = al.iterator(); it.hasNext();)
      print(it.next());
    print("********");
    for(Iterator<String> it = ll.iterator(); it.hasNext();)
      print(it.next());
    print("********");
    int alindex = 0;
    for(ListIterator<String> lit2 = ll.listIterator();
      lit2.hasNext();) {
      al.add(alindex, lit2.next());
      alindex += 2;
    }
    print("al = " + al);
    print("********");
    alindex = 0;
    // Start at the end:
    for(ListIterator<String>
      lit2 = ll.listIterator(ll.size());
      lit2.hasPrevious();) {
      al.add(alindex, lit2.previous());
      alindex += 2;
    }
```

```
    print("al = " + al);
  }
} /* Output:
ALGERIA
ANGOLA
BENIN
BOTSWANA
BURKINA FASO
BURUNDI
CAMEROON
CAPE VERDE
CENTRAL AFRICAN REPUBLIC
CHAD
********
COMOROS
CONGO
DJIBOUTI
EGYPT
EQUATORIAL GUINEA
ERITREA
ETHIOPIA
GABON
THE GAMBIA
GHANA
********
al = [COMOROS, ALGERIA, CONGO, ANGOLA, DJIBOUTI, BENIN,
EGYPT, BOTSWANA, EQUATORIAL GUINEA, BURKINA FASO, ERITREA,
BURUNDI, ETHIOPIA, CAMEROON, GABON, CAPE VERDE, THE GAMBIA,
CENTRAL AFRICAN REPUBLIC, GHANA, CHAD]
********
al = [GHANA, COMOROS, THE GAMBIA, ALGERIA, GABON, CONGO,
ETHIOPIA, ANGOLA, ERITREA, DJIBOUTI, EQUATORIAL GUINEA,
BENIN, EGYPT, EGYPT, DJIBOUTI, BOTSWANA, CONGO, EQUATORIAL
GUINEA, COMOROS, BURKINA FASO, ERITREA, BURUNDI, ETHIOPIA,
CAMEROON, GABON, CAPE VERDE, THE GAMBIA, CENTRAL AFRICAN
REPUBLIC, GHANA, CHAD]
*///:~
```

Writing this solution for two lists of the same size is a little risky, but incorrect sizes cause an exception.

Exercise 8

```
//: containers/E08_SList.java
/****************** Exercise 8 *****************
 * Create a generic, singly-linked list class
```

```
 * called SList, which, to keep things simple,
 * does not implement the List interface. Each Link
 * object in the list should contain a reference to
 * the next element in the list, but not the previous
 * one (LinkedList, in contrast, is a doubly-linked
 * list, which means it maintains links in both
 * directions). Create your own SListIterator which,
 * again for simplicity, does not implement ListIterator.
 * The only method in SList other than toString()
 * should be iterator(), which produces an
 * SListIterator. The only way to insert and remove
 * elements from an SList is through SListIterator.
 * Write code to demonstrate SList.
 *********************************************/
package containers;
import java.util.*;
import static net.mindview.util.Print.*;

interface SListIterator<T> {
  boolean hasNext();
  T next();
  void remove();
  void add(T element);
}

class SList<T> {
  // Utilization of the 'Null Object' pattern
  private final Link<T> header = new Link<T>(null, null);
  SList() { header.next = header; }
  public String toString() {
    StringBuilder buf = new StringBuilder();
    buf.append("[");
    for(SListIterator<T> it = iterator(); it.hasNext();) {
      T element = it.next();
      buf.append(element == this ? "(this SList)" :
        String.valueOf(element));
      if(it.hasNext())
        buf.append(", ");
    }
    buf.append("]");
    return buf.toString();
  }
  public SListIterator<T> iterator() {
    return new SListIteratorImpl();
  }
  private static class Link<T> {
    T element;
```

```
    Link<T> next;
    Link(T element, Link<T> next) {
      this.element = element;
      this.next = next;
    }
  }
  private class SListIteratorImpl
  implements SListIterator<T> {
    private Link<T> lastReturned = header;
    private Link<T> next;
    SListIteratorImpl() { next = header.next; }
    public boolean hasNext() { return next != header; }
    public T next() {
      if(next == header)
        throw new NoSuchElementException();
      lastReturned = next;
      next = next.next;
      return lastReturned.element;
    }
    public void remove() {
      if(lastReturned == header)
        throw new IllegalStateException();
      // Find an element before the last returned one
      for(Link<T> curr = header; ; curr = curr.next)
        if(curr.next == lastReturned) {
          curr.next = lastReturned.next;
          break;
        }
      lastReturned = header;
    }
    public void add(T element) {
      lastReturned = header;
      Link<T> newLink = new Link<T>(element, next);
      if(header.next == header)   // Empty list
        header.next = newLink;
      else {
        // Find an element before the one pointed by 'next'
        for(Link<T> curr = header; ; curr = curr.next)
          if(curr.next == next) {
            curr.next = newLink;
            break;
          }
      }
    }
  }
}
```

```
public class E08_SList {
  public static void main(String[] args) {
    // First, show some use cases for SListIterator
    print("Demonstrating SListIterator...");
    SList<String> sl = new SList<String>();
    print(sl);
    SListIterator<String> slit = sl.iterator();
    slit.add("One");
    slit.add("Two");
    slit.add("Three");
    print(slit.hasNext());
    print(sl);
    slit = sl.iterator();
    slit.add("Four");
    for(; slit.hasNext();)
      print(slit.next());
    print(sl);
    slit = sl.iterator();
    print(slit.next());
    slit.remove();
    print(slit.next());
    print(sl);
    // Now, show the same use cases for ListIterator, too
    print("\nDemonstrating ListIterator...");
    List<String> l = new ArrayList<String>();
    print(l);
    ListIterator<String> lit = l.listIterator();
    lit.add("One");
    lit.add("Two");
    lit.add("Three");
    print(lit.hasNext());
    print(l);
    lit = l.listIterator();
    lit.add("Four");
    for(; lit.hasNext();)
      print(lit.next());
    print(l);
    lit = l.listIterator();
    print(lit.next());
    lit.remove();
    print(lit.next());
    print(l);
  }
} /* Output:
Demonstrating SListIterator...
[]
false
```

```
[One, Two, Three]
One
Two
Three
[Four, One, Two, Three]
Four
One
[One, Two, Three]

Demonstrating ListIterator...
[]
false
[One, Two, Three]
One
Two
Three
[Four, One, Two, Three]
Four
One
[One, Two, Three]
*///:~
```

SList Iterator's set of operations, though much smaller than **ListIterator**'s, is sufficient for this demonstration. The behavior of each operation in **SListIterator** partially meets the specifications in the JDK (which see). Also, **main( )** contains a test program to show you the similarity of these two interfaces.

Note that the code is not optimized; insertions and removals are especially time-consuming. As an additional exercise, try to improve the speed. (Hint: get rid of search loops.)

Exercise 9

```
//: containers/E09_RandTreeSet.java
/****************** Exercise 9 *****************
 * Use RandomGenerator.String to fill a TreeSet,
 * but use alphabetic ordering. Print the TreeSet
 * to verify the sort order.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

public class E09_RandTreeSet {
  public static void main(String args[]) {
```

```
    TreeSet<String> ts =
      new TreeSet<String>(String.CASE_INSENSITIVE_ORDER);
    ts.addAll(CollectionData.list(
      new RandomGenerator.String(), 10));
    System.out.println("ts = " + ts);
  }
} /* Output:
ts = [ahKcxrE, GcFOWZn, GZMmJMR, naMesbt, oEsuEcU, OneOEdL,
qUCBbkI, smwHLGE, TcQrGse, YNzbrny]
*///:~
```

Exercise 10

Before we delve into our solution, we simplify the use of **SortedSet** with the following stipulations:

- There is no need to support an additional **Comparator**, so our **SortedSet** will use the elements' natural ordering. Assume the set is going to hold only objects of a class whose natural ordering is consistent with equals.
- The set will not tolerate **null** elements.
- We won't optimize the code for speed (for which purpose **LinkedList** is unsuited anyway).
- Our sorted set will have only one public no-arg constructor.

```
//: containers/E10_CustomSortedSet.java
/****************** Exercise 10 *****************
 * Using a LinkedList as your underlying
 * implementation, define your own SortedSet.
 ************************************************/
package containers;
import java.util.*;
import static java.util.Collections.binarySearch;
import static net.mindview.util.Print.*;

class CustomSortedSet<T> implements SortedSet<T> {
  private final List<T> list;
  public CustomSortedSet() { list = new LinkedList<T>(); }
  // If this sorted set is backed by an another one
  private CustomSortedSet(List<T> list) {
    this.list = list;
  }
  public String toString() { return list.toString(); }
  /*** Methods defined in the Collection interface ***/
```

```
  public int size() { return list.size(); }
  public boolean isEmpty() { return list.isEmpty(); }
  @SuppressWarnings("unchecked")
  public boolean contains(Object o) {
    checkForNull(o);
    return
      binarySearch((List<Comparable<T>>)list, (T)o) >= 0;
  }
  public Iterator<T> iterator() { return list.iterator(); }
  public Object[] toArray() { return list.toArray(); }
  public <T> T[] toArray(T[] a) { return list.toArray(a); }
  @SuppressWarnings("unchecked")
  public boolean add(T o) {
    checkForNull(o);
    int ip = binarySearch((List<Comparable<T>>)list, o);
    if(ip < 0) {
      ip = -(ip + 1);
      if(ip == list.size())
        list.add(o);
      else
        list.add(ip, o);
      return true;
    }
    return false;
  }
  public boolean remove(Object o) {
    checkForNull(o);
    return list.remove(o);
  }
  public boolean containsAll(Collection<?> c) {
    checkForNull(c);
    return list.containsAll(c);
  }
  public boolean addAll(Collection<? extends T> c) {
    checkForNull(c);
    checkForNullElements(c);
    boolean res = false;
    for(T item : c)
      res |= add(item);
    return res;
  }
  public boolean removeAll(Collection<?> c) {
    checkForNull(c);
    return list.removeAll(c);
  }
  public boolean retainAll(Collection<?> c) {
    checkForNull(c);
```

```
    return list.retainAll(c);
  }
  public void clear() { list.clear(); }
  public boolean equals(Object o) {
    return o instanceof CustomSortedSet &&
           list.equals(((CustomSortedSet<?>)o).list);
  }
  public int hashCode() { return list.hashCode(); }
  /*** Methods defined in the SortedSet interface ***/
  public Comparator<? super T> comparator() { return null; }
  public SortedSet<T> subSet(T fromElement, T toElement) {
    checkForNull(fromElement);
    checkForNull(toElement);
    int fromIndex = list.indexOf(fromElement);
    int toIndex = list.indexOf(toElement);
    checkForValidIndex(fromIndex);
    checkForValidIndex(toIndex);
    try {
      return new CustomSortedSet<T>(
        list.subList(fromIndex, toIndex));
    } catch(IndexOutOfBoundsException e) {
      throw new IllegalArgumentException(e);
    }
  }
  public SortedSet<T> headSet(T toElement) {
    checkForNull(toElement);
    int toIndex = list.indexOf(toElement);
    checkForValidIndex(toIndex);
    try {
      return new CustomSortedSet<T>(
        list.subList(0, toIndex));
    } catch(IndexOutOfBoundsException e) {
      throw new IllegalArgumentException(e);
    }
  }
  public SortedSet<T> tailSet(T fromElement) {
    checkForNull(fromElement);
    int fromIndex = list.indexOf(fromElement);
    checkForValidIndex(fromIndex);
    try {
      return new CustomSortedSet<T>(
        list.subList(fromIndex, list.size()));
    } catch(IndexOutOfBoundsException e) {
      throw new IllegalArgumentException(e);
    }
  }
  public T first() {
```

```
    try {
      return list.get(0);
    } catch(IndexOutOfBoundsException e) {
      throw new NoSuchElementException();
    }
  }
  public T last() {
    try {
      return list.get(list.size() - 1);
    } catch(IndexOutOfBoundsException e) {
      throw new NoSuchElementException();
    }
  }
  /*** Utility methods ***/
  private void checkForNullElements(Collection<?> c) {
    for(Iterator<?> it = c.iterator(); it.hasNext();)
      if(it.next() == null)
        throw new NullPointerException();
  }
  private void checkForNull(Object o) {
    if(o == null)
      throw new NullPointerException();
  }
  private void checkForValidIndex(int idx) {
    if(idx == -1)
      throw new IllegalArgumentException();
  }
}

public class E10_CustomSortedSet {
  // The whole main() method is basically copy-pasted from
  // containers/SortedSetDemo.java!
  public static void main(String[] args) {
    SortedSet<String> sortedSet =
      new CustomSortedSet<String>();
    Collections.addAll(sortedSet,
      "one two three four five six seven eight"
        .split(" "));
    print(sortedSet);
    String low = sortedSet.first();
    String high = sortedSet.last();
    print(low);
    print(high);
    Iterator<String> it = sortedSet.iterator();
    for(int i = 0; i <= 6; i++) {
      if(i == 3) low = it.next();
      if(i == 6) high = it.next();
```

```
      else it.next();
    }
    print(low);
    print(high);
    print(sortedSet.subSet(low, high));
    print(sortedSet.headSet(high));
    print(sortedSet.tailSet(low));
    print(sortedSet.contains("three"));
    print(sortedSet.contains("eleven"));
    print(sortedSet.addAll(Arrays.asList(
      "three", "eleven")));
    print(sortedSet);
    print(sortedSet.retainAll(Arrays.asList(
      "three", "eleven")));
    print(sortedSet);
    // Demonstrate data integrity
    try {
      sortedSet.addAll(Arrays.asList("zero", null));
    } catch(NullPointerException e) {
      System.out.println("Null elements not supported!");
    }
    // The set will not contain "zero"!
    print(sortedSet);
  }
} /* Output:
[eight, five, four, one, seven, six, three, two]
eight
two
one
two
[one, seven, six, three]
[eight, five, four, one, seven, six, three]
[one, seven, six, three, two]
true
false
true
[eight, eleven, five, four, one, seven, six, three, two]
true
[eleven, three]
Null elements not supported!
[eleven, three]
*///:~
```

Most of the methods just forward the request to the underlying **LinkedList**. Only the **contains( )** and **add( )** methods are particularly important. Both use the fact that the list is already sorted, which the **add( )** method guarantees. The program lacks efficiency but highlights the basics.

Notice that the exception handling satisfies the JDK documentation requirements.

Also notice that we assure data integrity. The JDK says this about the **addAll( )** method: *"Throws **NullPointerException** - if the specified collection contains one or more **null** elements and this set does not support **null** elements, or if the specified collection is **null**."* Given this vague description, we choose to roll back all changes in case of an exception. If you comment out the **checkForNullElements(c)** call you get a second variation. This also happens for a **ClassCastException** when an element of the specified collection cannot be added to the set due to its class.

Exercise 11

```
//: containers/E11_PriorityQueue.java
/****************** Exercise 11 *****************
 * Create a class that contains an Integer that is
 * initialized to a value between 0 and 100 using
 * java.util.Random. Implement Comparable using this
 * Integer field. Fill a PriorityQueue with objects of
 * your class, and extract the values using poll() to
 * show that it produces the expected order.
 ************************************************/
package containers;
import java.util.*;

class Item implements Comparable<Item> {
  private static final Random rnd = new Random(47);
  private Integer priority = rnd.nextInt(101);
  public int compareTo(Item arg) {
    return priority < arg.priority ? -1 :
           priority == arg.priority ? 0 : 1;
  }
  public String toString() {
    return Integer.toString(priority);
  }
}

public class E11_PriorityQueue {
  public static void main(String[] args) {
    PriorityQueue<Item> queue = new PriorityQueue<Item>();
    for(int i = 0; i < 10; i++)
      queue.add(new Item());
    Item item;
    while((item = queue.poll()) != null)
```

```
      System.out.println(item);
  }
} /* Output:
15
17
18
20
22
62
65
67
95
100
*///:~
```

Exercise 12

```
//: containers/E12_MapsDemo.java
/****************** Exercise 12 *****************
 * Substitute a HashMap, a TreeMap, and a LinkedHashMap
 * in AssociativeArray.java's main().
 ***********************************************/
package containers;
import java.util.*;
import static net.mindview.util.Print.*;

public class E12_MapsDemo {
  private static void test(Map<String, String> map) {
    map.put("sky", "blue");
    map.put("grass", "green");
    map.put("ocean", "dancing");
    map.put("tree", "tall");
    map.put("earth", "brown");
    map.put("sun", "warm");
    try {
      map.put("extra", "object");
    } catch(ArrayIndexOutOfBoundsException e) {
      // Never happen!
      print("Too many objects!");
    }
    print(map);
    print(map.get("ocean"));
  }
  public static void main(String[] args) {
    test(new HashMap<String, String>());
    test(new TreeMap<String, String>());
```

```
    test(new LinkedHashMap<String, String>());
  }
} /* Output:
{sky=blue, tree=tall, grass=green, extra=object,
ocean=dancing, earth=brown, sun=warm}
dancing
{earth=brown, extra=object, grass=green, ocean=dancing,
sky=blue, sun=warm, tree=tall}
dancing
{sky=blue, grass=green, ocean=dancing, tree=tall,
earth=brown, sun=warm, extra=object}
dancing
*///:~
```

Notice that different **Map** implementations hold and present the pairs in different orders.

Exercise 13

```
//: containers/E13_WordCounter.java
/****************** Exercise 13 *****************
 * Use AssociativeArray.java to create a
 * word-occurrence counter, mapping String to Integer.
 * Using the net.mindview.util.TextFile utility in
 * this book, open a text file and break up the
 * words in that file using whitespace and
 * punctuation, and count the occurrence of the
 * words in that file.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

class AssociativeArray<K,V> {
  private Object[][] pairs;
  private int index;
  public AssociativeArray(int length) {
    pairs = new Object[length][2];
  }
  public void put(K key, V value) {
    for(int i = 0; i < index; i++)
      if(key.equals(pairs[i][0])) {
        pairs[i] = new Object[]{ key, value };
        return;
      }
    if(index >= pairs.length)
```

```
      throw new ArrayIndexOutOfBoundsException();
    pairs[index++] = new Object[]{ key, value };
  }
  @SuppressWarnings("unchecked")
  public V get(K key) {
    for(int i = 0; i < index; i++)
      if(key.equals(pairs[i][0]))
        return (V)pairs[i][1];
    return null; // Did not find key
  }
  public String toString() {
    StringBuilder result = new StringBuilder();
    for(int i = 0; i < index; i++) {
      result.append(pairs[i][0].toString());
      result.append(" : ");
      result.append(pairs[i][1].toString());
      if(i < index - 1)
        result.append("\n");
    }
    return result.toString();
  }
}

public class E13_WordCounter {
  public static void main(String[] args) {
    List<String> words =
      new TextFile("E12_MapsDemo.java", "\\W+");
    AssociativeArray<String, Integer> map =
      new AssociativeArray<String, Integer>(words.size());
    for(String word : words) {
      Integer freq = map.get(word);
      map.put(word, freq == null ? 1 : freq + 1);
    }
    System.out.println(map);
  }
} /* Output: (Sample)
containers : 1
E12_MapsDemo : 2
java : 3
Exercise : 1
12 : 1
Substitute : 1
a : 3
HashMap : 2
TreeMap : 2
...
objects : 1
```

```
get : 1
args : 1
new : 3
Output : 1
*///:~
```

The original **AssociativeArray** is inadequate here: if the array previously held a mapping for some key, the new value would not replace the old one. To remedy this, we alter the method **put( )**. Notice that we cannot simply create a new class by inheriting it from the original **AssociativeArray**, since all attributes are **private** and there are no corresponding **getXXX( )** methods.

Exercise 14

```
//: containers/E14_PropertiesDemo.java
/****************** Exercise 14 *****************
 * Show that java.util.Properties works in the above
 * program (containers/Maps.java from the book).
 ************************************************/
package containers;
import java.util.*;
import static net.mindview.util.Print.*;

public class E14_PropertiesDemo {
  static void printKeys(Map<Object, Object> map) {
    printnb("Size = " + map.size() + ", ");
    printnb("Keys: ");
    print(map.keySet()); // Produce a Set of the keys
  }
  static void test(Map<Object, Object> map) {
    print(map.getClass().getSimpleName());
    map.putAll(new CountingMapData(25));
    // Map has 'Set' behavior for keys:
    map.putAll(new CountingMapData(25));
    printKeys(map);
    // Producing a Collection of the values:
    printnb("Values: ");
    print(map.values());
    print(map);
    print("map.containsKey(11): " + map.containsKey(11));
    print("map.get(11): " + map.get(11));
    print("map.containsValue(\"F0\"): "
      + map.containsValue("F0"));
    Object key = map.keySet().iterator().next();
    print("First key in map: " + key);
    map.remove(key);
```

```
    printKeys(map);
    map.clear();
    print("map.isEmpty(): " + map.isEmpty());
    map.putAll(new CountingMapData(25));
    // Operations on the Set change the Map:
    map.keySet().removeAll(map.keySet());
    print("map.isEmpty(): " + map.isEmpty());
  }
  public static void main(String[] args) {
    test(new Properties());
  }
} /* Output:
Properties
Size = 25, Keys: [24, 23, 22, 21, 20, 19, 18, 17, 16, 15,
14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Values: [Y0, X0, W0, V0, U0, T0, S0, R0, Q0, P0, O0, N0, M0,
L0, K0, J0, I0, H0, G0, F0, E0, D0, C0, B0, A0]
{24=Y0, 23=X0, 22=W0, 21=V0, 20=U0, 19=T0, 18=S0, 17=R0,
16=Q0, 15=P0, 14=O0, 13=N0, 12=M0, 11=L0, 10=K0, 9=J0, 8=I0,
7=H0, 6=G0, 5=F0, 4=E0, 3=D0, 2=C0, 1=B0, 0=A0}
map.containsKey(11): true
map.get(11): L0
map.containsValue("F0"): true
First key in map: 24
Size = 24, Keys: [23, 22, 21, 20, 19, 18, 17, 16, 15, 14,
13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
map.isEmpty(): true
map.isEmpty(): true
*///:~
```

java.util.Properties extends **Hashtable<Object,Object>**, and requires only a few changes to integrate it into the original code.

Exercise 15

```
//: containers/E15_WordCounter2.java
/****************** Exercise 15 *****************
 * Repeat Exercise 13 using a SlowMap.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

public class E15_WordCounter2 {
  public static void main(String[] args) {
    List<String> words =
```

```
      new TextFile("E12_MapsDemo.java", "\\W+");
    SlowMap<String,Integer> map =
      new SlowMap<String,Integer>();
    for(String word : words) {
      Integer freq = map.get(word);
      map.put(word, freq == null ? 1 : freq + 1);
    }
    System.out.println(map);
  }
} /* Output:
{AssociativeArray=1, test=4, args=1, package=1, Output=1,
tree=4, sky=4, import=2, objects=1, happen=1, static=3,
Too=1, grass=4, extra=4, get=1, Map=1, class=1, and=1,
Print=1, many=1, util=2, String=9, try=1, Substitute=1,
put=7, dancing=7, warm=4, catch=1, sun=4, E12_MapsDemo=2,
net=1, private=1, 12=1, ocean=5, object=4, map=10,
LinkedHashMap=2, tall=4, mindview=1, green=4, e=1,
containers=2, Never=1, a=3, HashMap=2, TreeMap=2,
Exercise=1, void=2, in=1, print=3, main=2, new=3, blue=4,
s=1, java=3, public=2, brown=4, earth=4,
ArrayIndexOutOfBoundsException=1}
*///:~
```

Exercise 16

```
//: containers/E16_SlowMapTest.java
/****************** Exercise 16 *****************
 * Apply the tests in Maps.java to SlowMap to
 * verify that it works. Fix anything in SlowMap
 * that doesn't work correctly.
 ***********************************************/
package containers;
import java.util.*;
import static net.mindview.util.Print.*;

// Does not support null value as key!
class SlowMap2<K,V> extends AbstractMap<K,V> {
  private List<K> keys = new ArrayList<K>();
  private List<V> values = new ArrayList<V>();
  @Override public V put(K key, V value) {
    if(key == null)
      throw new NullPointerException();
    V oldValue = get(key); // The old value or null
    if(!keys.contains(key)) {
      keys.add(key);
      values.add(value);
```

```
    } else
      values.set(keys.indexOf(key), value);
    return oldValue;
  }
  @Override public V get(Object key) {
    if(key == null)
      throw new NullPointerException();
    if(!keys.contains(key))
      return null;
    return values.get(keys.indexOf(key));
  }
  private EntrySet entrySet = new EntrySet();
  @Override public Set<Map.Entry<K,V>> entrySet() {
    return entrySet;
  }
  // Uses the 'Flyweight' pattern
  private class EntrySet
  extends AbstractSet<Map.Entry<K,V>> {
    @Override public Iterator<Map.Entry<K,V>> iterator() {
      return new Iterator<Map.Entry<K,V>>() {
        private int index = -1;
        boolean canRemove;
        public boolean hasNext() {
          return index < keys.size() - 1;
        }
        public Map.Entry<K,V> next() {
          canRemove = true;
          ++index;
          return new MapEntry<K,V>(
            keys.get(index), values.get(index));
        }
        public void remove() {
          if(!canRemove)
            throw new IllegalStateException();
          canRemove = false;
          keys.remove(index);
          values.remove(index--);
        }
      };
    }
    @SuppressWarnings("unchecked")
    @Override public boolean contains(Object o) {
      if(o instanceof MapEntry) {
        MapEntry<K,V> e = (MapEntry<K,V>)o;
        K key = e.getKey();
        if(keys.contains(key))
          return e.equals(
```

```
            new MapEntry<K,V>(key, get(key)));
      }
      return false;
    }
    @SuppressWarnings("unchecked")
    @Override public boolean remove(Object o) {
      if(contains(o)) {
        MapEntry<K,V> e = (MapEntry<K,V>)o;
        K key = e.getKey();
        V val = e.getValue();
        keys.remove(key);
        values.remove(val);
        return true;
      }
      return false;
    }
    @Override public int size() { return keys.size(); }
    @Override public void clear() {
      keys.clear();
      values.clear();
    }
  }
}

public class E16_SlowMapTest {
  public static void printKeys(Map<Integer,String> map) {
    printnb("Size = " + map.size() + ", ");
    printnb("Keys: ");
    print(map.keySet()); // Produce a Set of the keys
  }
  public static void test(Map<Integer,String> map) {
    print(map.getClass().getSimpleName());
    map.putAll(new CountingMapData(25));
    // Map has 'Set' behavior for keys:
    map.putAll(new CountingMapData(25));
    printKeys(map);
    // Producing a Collection of the values:
    printnb("Values: ");
    print(map.values());
    print(map);
    print("map.containsKey(11): " + map.containsKey(11));
    print("map.get(11): " + map.get(11));
    print("map.containsValue(\"F0\"): "
      + map.containsValue("F0"));
    Integer key = map.keySet().iterator().next();
    print("First key in map: " + key);
    map.remove(key);
```

```
    printKeys(map);
    map.clear();
    print("map.isEmpty(): " + map.isEmpty());
    map.putAll(new CountingMapData(25));
    // Operations on the Set change the Map:
    map.keySet().removeAll(map.keySet());
    print("map.isEmpty(): " + map.isEmpty());
  }
  public static void main(String[] args) {
    System.out.println("Testing SlowMap");
    test(new SlowMap<Integer,String>());
    System.out.println("Testing SlowMap2");
    test(new SlowMap2<Integer,String>());
  }
} /* Output:
Testing SlowMap
SlowMap
Size = 25, Keys: [14, 12, 10, 8, 22, 6, 24, 4, 18, 2, 20, 0,
15, 16, 13, 11, 9, 7, 21, 5, 23, 3, 17, 1, 19]
Values: [O0, M0, K0, I0, W0, G0, Y0, E0, S0, C0, U0, A0, P0,
Q0, N0, L0, J0, H0, V0, F0, X0, D0, R0, B0, T0]
{14=O0, 12=M0, 10=K0, 8=I0, 22=W0, 6=G0, 24=Y0, 4=E0, 18=S0,
2=C0, 20=U0, 0=A0, 15=P0, 16=Q0, 13=N0, 11=L0, 9=J0, 7=H0,
21=V0, 5=F0, 23=X0, 3=D0, 17=R0, 1=B0, 19=T0}
map.containsKey(11): true
map.get(11): L0
map.containsValue("F0"): true
First key in map: 14
Size = 25, Keys: [14, 12, 10, 8, 22, 6, 24, 4, 18, 2, 20, 0,
15, 16, 13, 11, 9, 7, 21, 5, 23, 3, 17, 1, 19]
map.isEmpty(): false
map.isEmpty(): false
Testing SlowMap2
SlowMap2
Size = 25, Keys: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Values: [A0, B0, C0, D0, E0, F0, G0, H0, I0, J0, K0, L0, M0,
N0, O0, P0, Q0, R0, S0, T0, U0, V0, W0, X0, Y0]
{0=A0, 1=B0, 2=C0, 3=D0, 4=E0, 5=F0, 6=G0, 7=H0, 8=I0, 9=J0,
10=K0, 11=L0, 12=M0, 13=N0, 14=O0, 15=P0, 16=Q0, 17=R0,
18=S0, 19=T0, 20=U0, 21=V0, 22=W0, 23=X0, 24=Y0}
map.containsKey(11): true
map.get(11): L0
map.containsValue("F0"): true
First key in map: 0
Size = 24, Keys: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

```
map.isEmpty(): true
map.isEmpty(): true
*///:~
```

As stated in the JDK, **entrySet( )** should return a set view of the mappings contained in a map. **SlowMap** returns a fresh copy of the keys instead, so integrating it into **Maps.java** is troublesome. The returned entry set doesn't properly remove elements, i.e., operations on the set do not change the map. The **SlowMap2** class fixes this; see the difference in the output.

Exercise 17

The **SlowMap2** class already uses all of **Map** (which you can test by invoking each method listed in **Map** on **SlowMap2**). The two most frequently overridden methods of **AbstractMap** (besides those already overridden in **SlowMap2**) are **containsKey( )** and **remove( )**. Their default use requires linear time in the size of the map though, which is awkward. However, overriding them in **SlowMap2** gains nothing. You really can't speed up the code, so extra development is unnecessary.

Remember, sometimes the abstract base version of the container you are creating already uses the methods you need (not that you create your own containers all that often).

Exercise 18

```
//: containers/E18_SlowSet.java
/****************** Exercise 18 *****************
 * Using SlowMap.java for inspiration, create a
 * SlowSet.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

class SlowSet<K> extends AbstractSet<K> {
  private List<K> keys = new ArrayList<K>();
  public boolean add(K key) {
    if(!contains(key)) {
      keys.add(key);
      return true;
    }
    return false;
  }
```

```
  public Iterator<K> iterator() { return keys.iterator(); }
  public int size() { return keys.size(); }
}

public class E18_SlowSet {
  public static void main(String[] args) {
    SlowSet<String> slowSet = new SlowSet<String>();
    slowSet.addAll(Countries.names(10));
    slowSet.addAll(Countries.names(10));
    slowSet.addAll(Countries.names(10));
    System.out.println(slowSet);
  }
} /* Output:
[ALGERIA, ANGOLA, BENIN, BOTSWANA, BURKINA FASO, BURUNDI,
CAMEROON, CAPE VERDE, CENTRAL AFRICAN REPUBLIC, CHAD]
*///:~
```

We test by repeatedly adding the same elements to the set.

Exercise 19

```
//: containers/E19_WordCounter3.java
/****************** Exercise 19 *****************
 * Repeat Exercise 13 using a SimpleHashMap.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

public class E19_WordCounter3 {
  public static void main(String[] args) {
    List<String> words =
      new TextFile("E12_MapsDemo.java", "\\W+");
    SimpleHashMap<String,Integer> map =
      new SimpleHashMap<String,Integer>();
    for(String word : words) {
      Integer freq = map.get(word);
      map.put(word, freq == null ? 1 : freq + 1);
    }
    System.out.println(map);
  }
} /* Output:
{AssociativeArray=1, test=4, args=1, package=1, Output=1,
tree=4, sky=4, import=2, objects=1, happen=1, static=3,
grass=4, Too=1, extra=4, get=1, Map=1, class=1, and=1,
Print=1, many=1, util=2, String=9, try=1, Substitute=1,
```

```
put=7, dancing=7, warm=4, catch=1, sun=4, E12_MapsDemo=2,
net=1, private=1, 12=1, ocean=5, object=4, map=10,
LinkedHashMap=2, tall=4, mindview=1, green=4, e=1,
containers=2, Never=1, a=3, HashMap=2, TreeMap=2,
Exercise=1, void=2, in=1, main=2, print=3, new=3, blue=4,
s=1, java=3, public=2, brown=4,
ArrayIndexOutOfBoundsException=1, earth=4}
*///:~
```

Exercise 20

```
//: containers/E20_SimpleHashMapCollisons.java
/****************** Exercise 20 *****************
 * Modify SimpleHashMap so it reports
 * collisions, and test this by adding the same
 * data set twice so you see collisions.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

class SimpleHashMap2<K,V> extends SimpleHashMap<K,V> {
  @Override public V put(K key, V value) {
    V oldValue = null;
    int index = Math.abs(key.hashCode()) % SIZE;
    MapEntry<K,V> pair = new MapEntry<K,V>(key, value);
    if(buckets[index] == null)
      buckets[index] = new LinkedList<MapEntry<K,V>>();
    // Lines added here:
    else {
      System.out.println(
        "Collision while adding\n" + pair
        + "\nBucket already contains:");
      Iterator<MapEntry<K,V>> it =
        buckets[index].iterator();
      while(it.hasNext())
        System.out.println(it.next());
    }
    // End of lines added
    LinkedList<MapEntry<K,V>> bucket = buckets[index];
    boolean found = false;
    ListIterator<MapEntry<K,V>> it = bucket.listIterator();
    while(it.hasNext()) {
      MapEntry<K,V> iPair = it.next();
      if(iPair.getKey().equals(key)) {
        oldValue = iPair.getValue();
```

```
        it.set(pair); // Replace old with new
        found = true;
        break;
      }
    }
    if(!found)
      buckets[index].add(pair);
    return oldValue;
  }
}

public class E20_SimpleHashMapCollisons {
  public static void main(String[] args) {
    SimpleHashMap2<String,String> m =
      new SimpleHashMap2<String,String>();
    m.putAll(Countries.capitals(25));
    m.putAll(Countries.capitals(25));
    System.out.println(m);
  }
} /* Output:
Collision while adding
ALGERIA=Algiers
Bucket already contains:
ALGERIA=Algiers
Collision while adding
ANGOLA=Luanda
Bucket already contains:
ANGOLA=Luanda
Collision while adding
BENIN=Porto-Novo
Bucket already contains:
BENIN=Porto-Novo
Collision while adding
BOTSWANA=Gaberone
Bucket already contains:
BOTSWANA=Gaberone
Collision while adding
BURKINA FASO=Ouagadougou
Bucket already contains:
BURKINA FASO=Ouagadougou
Collision while adding
BURUNDI=Bujumbura
Bucket already contains:
BURUNDI=Bujumbura
Collision while adding
CAMEROON=Yaounde
Bucket already contains:
```

```
CAMEROON=Yaounde
Collision while adding
CAPE VERDE=Praia
Bucket already contains:
CAPE VERDE=Praia
Collision while adding
CENTRAL AFRICAN REPUBLIC=Bangui
Bucket already contains:
CENTRAL AFRICAN REPUBLIC=Bangui
Collision while adding
CHAD=N'djamena
Bucket already contains:
CHAD=N'djamena
Collision while adding
COMOROS=Moroni
Bucket already contains:
COMOROS=Moroni
Collision while adding
CONGO=Brazzaville
Bucket already contains:
CONGO=Brazzaville
Collision while adding
DJIBOUTI=Dijibouti
Bucket already contains:
DJIBOUTI=Dijibouti
Collision while adding
EGYPT=Cairo
Bucket already contains:
EGYPT=Cairo
Collision while adding
EQUATORIAL GUINEA=Malabo
Bucket already contains:
EQUATORIAL GUINEA=Malabo
Collision while adding
ERITREA=Asmara
Bucket already contains:
ERITREA=Asmara
Collision while adding
ETHIOPIA=Addis Ababa
Bucket already contains:
ETHIOPIA=Addis Ababa
Collision while adding
GABON=Libreville
Bucket already contains:
GABON=Libreville
Collision while adding
THE GAMBIA=Banjul
```

```
Bucket already contains:
THE GAMBIA=Banjul
Collision while adding
GHANA=Accra
Bucket already contains:
GHANA=Accra
Collision while adding
GUINEA=Conakry
Bucket already contains:
GUINEA=Conakry
Collision while adding
BISSAU=Bissau
Bucket already contains:
BISSAU=Bissau
Collision while adding
COTE D'IVOIR (IVORY COAST)=Yamoussoukro
Bucket already contains:
COTE D'IVOIR (IVORY COAST)=Yamoussoukro
Collision while adding
KENYA=Nairobi
Bucket already contains:
KENYA=Nairobi
Collision while adding
LESOTHO=Maseru
Bucket already contains:
LESOTHO=Maseru
{CHAD=N'djamena, BISSAU=Bissau, CONGO=Brazzaville,
BURUNDI=Bujumbura, DJIBOUTI=Dijibouti, EQUATORIAL
GUINEA=Malabo, GUINEA=Conakry, LESOTHO=Maseru, EGYPT=Cairo,
GHANA=Accra, CENTRAL AFRICAN REPUBLIC=Bangui, BENIN=Porto-
Novo, GABON=Libreville, COTE D'IVOIR (IVORY
COAST)=Yamoussoukro, KENYA=Nairobi, ETHIOPIA=Addis Ababa,
ALGERIA=Algiers, BOTSWANA=Gaberone, COMOROS=Moroni,
ANGOLA=Luanda, ERITREA=Asmara, CAPE VERDE=Praia, BURKINA
FASO=Ouagadougou, THE GAMBIA=Banjul, CAMEROON=Yaounde}
*///:~
```

We added the lines surrounded by comments in the **put( )** method, and moved the creation of the **pair** object so it prints when the collision occurs. Everything else is the same.

Exercise 21

```
//: containers/E21_LeftToReader.java
/****************** Exercise 21 *****************
 * Modify SimpleHashMap so that it reports the
```

```
 * number of "probes" necessary when collisions
 * occur. That is, how many calls to next() must
 * be made on the Iterators that walk the
 * LinkedLists looking for matches?
 *********************************************/
package containers;

public class E21_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Exercise left to reader");
  }
} ///:~
```

Hint: Modify the **get( )** method the same way as **put( )** in Exercise 20.

Exercise 22

```
//: containers/E22_SimpleHashMapClearRemove.java
/****************** Exercise 22 *****************
 * Implement the clear() and remove() methods for
 * SimpleHashMap.
 *********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

class SimpleHashMap3<K,V> extends SimpleHashMap<K,V> {
  @SuppressWarnings("unchecked")
  public void clear() {
    // Effectively erase everything by allocating
    // a new empty array of buckets:
    buckets = new LinkedList[SIZE];
  }
  public V remove(Object key) {
    // Code is copied from get(), except that
    // before returning the value, the Map.Entry is
    // removed from the list:
    int index = Math.abs(key.hashCode()) % SIZE;
    if(buckets[index] == null) return null;
    Iterator<MapEntry<K,V>> it = buckets[index].iterator();
    while(it.hasNext()) {
      MapEntry<K,V> iPair = it.next();
      if(iPair.getKey().equals(key)) {
        // Changes are here:
        V value = iPair.getValue();
        // Removes the last fetched value:
```

```
        it.remove();
        return value;
      }
    }
    return null;
  }
}

public class E22_SimpleHashMapClearRemove {
  public static void main(String[] args) {
    SimpleHashMap3<String,String> m =
      new SimpleHashMap3<String,String>();
    m.putAll(Countries.capitals(10));
    System.out.println(m);
    System.out.println("m.get(\"BURUNDI\") = "
      + m.get("BURUNDI"));
    m.remove("BURUNDI");
    System.out.println(
      "After removal, m.get(\"BURUNDI\") = "
      + m.get("BURUNDI"));
    m.clear();
    System.out.println("After clearing: " + m);
  }
} /* Output:
{ANGOLA=Luanda, CHAD=N'djamena, CAPE VERDE=Praia,
ALGERIA=Algiers, BURKINA FASO=Ouagadougou, CENTRAL AFRICAN
REPUBLIC=Bangui, BENIN=Porto-Novo, BOTSWANA=Gaberone,
BURUNDI=Bujumbura, CAMEROON=Yaounde}
m.get("BURUNDI") = Bujumbura
After removal, m.get("BURUNDI") = null
After clearing: {}
*///:~
```

Exercise 23

```
//: containers/E23_FullSimpleHashMap.java
/****************** Exercise 23 *****************
 * Implement the rest of the Map interface for
 * SimpleHashMap.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class SimpleHashMap4<K,V> extends SimpleHashMap3<K,V> {
```

```
  public int size() {
    // Could rely on the inherited implementation, which
    // returns entrySet().size(), but this is faster:
    int sz = 0;
    for(LinkedList<MapEntry<K,V>> bucket : buckets)
      if(bucket != null)
        sz += bucket.size();
    return sz;
  }
  public boolean isEmpty() {
    // Could just say return size() == 0;
    // but this is faster:
    for(LinkedList<MapEntry<K,V>> bucket : buckets)
      if(bucket != null)
        return false;
    return true;
  }
  public boolean containsKey(Object key) {
    // A slight modification of the previous method:
    for(LinkedList<MapEntry<K,V>> bucket : buckets) {
      if(bucket == null) continue;
      for(MapEntry<K,V> mPair : bucket)
        if(mPair.getKey().equals(key))
          return true;
    }
    return false;
  }
}

public class E23_FullSimpleHashMap {
  public static void main(String args[]) {
    SimpleHashMap4<String,String>
      m = new SimpleHashMap4<String,String>(),
      m2 = new SimpleHashMap4<String,String>();
    m.putAll(Countries.capitals(10));
    m2.putAll(Countries.capitals(10));
    print("m.size() = " + m.size());
    print("m.isEmpty() = " + m.isEmpty());
    print("m.equals(m2) = " + m.equals(m2));
    print("m.containsKey(\"BENIN\") = " +
      m.containsKey("BENIN"));
    print("m.containsKey(\"MARS\") = " +
      m.containsKey("MARS"));
    print("m.keySet() = " + m.keySet());
    print("m.values() = " + m.values());
  }
} /* Output:
```

```
m.size() = 10
m.isEmpty() = false
m.equals(m2) = true
m.containsKey("BENIN") = true
m.containsKey("MARS") = false
m.keySet() = [ANGOLA, CHAD, CAPE VERDE, ALGERIA, BURKINA
FASO, CENTRAL AFRICAN REPUBLIC, BENIN, BOTSWANA, BURUNDI,
CAMEROON]
m.values() = [Luanda, N'djamena, Praia, Algiers,
Ouagadougou, Bangui, Porto-Novo, Gaberone, Bujumbura,
Yaounde]
*///:~
```

Exercise 24

```
//: containers/E24_SimpleHashSet.java
/****************** Exercise 24 *****************
 * Following the example in SimpleHashMap.java,
 * create and test a SimpleHashSet.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

class SimpleHashSet<K> extends AbstractSet<K> {
  private final static int SIZE = 997;
  @SuppressWarnings("unchecked")
  private LinkedList<K>[] buckets = new LinkedList[SIZE];
  public boolean add(K key) {
    int index = Math.abs(key.hashCode()) % SIZE;
    if(buckets[index] == null)
      buckets[index] = new LinkedList<K>();
    LinkedList<K> bucket = buckets[index];
    ListIterator<K> it = bucket.listIterator();
    while(it.hasNext())
      if(it.next().equals(key))
        return false;
    // Sets do not permit duplicates and one
    // was already in the set.
    it.add(key);
    return true;
  }
  public boolean contains(Object key) {
    int index = Math.abs(key.hashCode()) % SIZE;
    if(buckets[index] == null) return false;
    Iterator<K> it = buckets[index].iterator();
```

```
    while(it.hasNext())
      if(it.next().equals(key))
        return true;
    return false;
  }
  public int size() {
    int sz = 0;
    for(LinkedList<K> bucket : buckets) {
      if(bucket != null)
      sz += bucket.size();
    }
    return sz;
  }
  public Iterator<K> iterator() {
    return new Iterator<K>() {
      private int count;
      private boolean canRemove;
      private int index1, index2;
      public boolean hasNext() { return count < size(); }
      public K next() {
        if(hasNext()) {
          canRemove = true;
          ++count;
          for(;;) {
            // Position of the next non-empty bucket
            while(buckets[index1] == null)
              ++index1;
            // Position of the next item to return
            try {
              return buckets[index1].get(index2++);
            } catch(IndexOutOfBoundsException e) {
              // Continue search from the next bucket
              ++index1;
              index2 = 0;
            }
          }
        }
        throw new NoSuchElementException();
      }
      public void remove() {
        if(canRemove) {
          canRemove = false;
          buckets[index1].remove(--index2);
          if(buckets[index1].isEmpty())  // Housekeeping...
            buckets[index1++] = null;
          --count;
        } else
```

```
            throw new IllegalStateException();
        }
      };
    }
}

public class E24_SimpleHashSet {
  public static void main(String[] args) {
    SimpleHashSet<String> m = new SimpleHashSet<String>();
    m.addAll(Countries.names(10));
    m.addAll(Countries.names(10));
    System.out.println("m = " + m);
    System.out.println("m.size() = " + m.size());
    Iterator<String> it = m.iterator();
    System.out.println("it.next()= "+ it.next());
    it.remove();
    System.out.println("it.next()= "+ it.next());
    System.out.println("m = " + m);
    m.remove("ALGERIA");
    System.out.println("m = " + m);
  }
} /* Output:
m = [BOTSWANA, BENIN, ALGERIA, BURKINA FASO, CAMEROON,
CENTRAL AFRICAN REPUBLIC, CAPE VERDE, CHAD, ANGOLA, BURUNDI]
m.size() = 10
it.next()= BOTSWANA
it.next()= BENIN
m = [BENIN, ALGERIA, BURKINA FASO, CAMEROON, CENTRAL AFRICAN
REPUBLIC, CAPE VERDE, CHAD, ANGOLA, BURUNDI]
m = [BENIN, BURKINA FASO, CAMEROON, CENTRAL AFRICAN
REPUBLIC, CAPE VERDE, CHAD, ANGOLA, BURUNDI]
*///:~
```

This contains keys but not values. Also, we declare different methods in **Set** than in **AbstractSet**; the above methods are the only ones necessary to implement the **SimpleHashSet**. Pay special attention to the **iterator( )** method.

Exercise 25

```
//: containers/E25_FullSimpleHashMap2.java
/****************** Exercise 25 *****************
 * Instead of using a ListIterator for each bucket,
 * modify MapEntry so it is a self-contained
 * singly-linked list (each MapEntry should have a
 * forward link to the next MapEntry). Modify the
 * rest of the code in SimpleHashMap.java so
```

```
 * this new approach works correctly.
 *********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class SimpleHashMap5<K,V> extends AbstractMap<K,V> {
  static class Entry<K,V> implements Map.Entry<K,V> {
    private K key;
    private V value;
    Entry<K,V> next;
    public Entry(K key, V value) {
      this.key = key;
      this.value = value;
    }
    public K getKey() { return key; }
    public V getValue() { return value; }
    public V setValue(V v) {
      V result = value;
      value = v;
      return result;
    }
    public int hashCode() {
      return key.hashCode() ^
        (value == null ? 0 : value.hashCode());
    }
    public boolean equals(Object o) {
      if(o instanceof Entry) {
        @SuppressWarnings("unchecked")
        Entry<K,V> e = (Entry<K,V>)o;
        Object key1 = getKey();
        Object key2 = e.getKey();
        if(key1.equals(key2)) {
          Object val1 = getValue();
          Object val2 = e.getValue();
          return
            val1 == null ? val2 == null : val1.equals(val2);
        }
      }
      return false;
    }
    public String toString() { return key + "=" + value; }
  }
  static final int SIZE = 997;
  @SuppressWarnings("unchecked")
  Entry<K,V>[] buckets = new Entry[SIZE];
```

```
  public V put(K key, V value) {
    V oldValue = null;
    int index = Math.abs(key.hashCode()) % SIZE;
    Entry<K,V> newPair = new Entry<K,V>(key, value);
    if(buckets[index] == null)
      buckets[index] = newPair;
    Entry<K,V> prevPair = null;        // Previous element
    boolean found = false;
    for(Entry<K,V> pair = buckets[index] ; pair != null;
      pair = pair.next) {
      if(pair.getKey().equals(key)) {
        oldValue = pair.getValue();
        // Replace old with new
        if(prevPair != null)
          prevPair.next = newPair;
        else
          buckets[index] = newPair;
        newPair.next = pair.next;
        found = true;
        break;
      }
      prevPair = pair;
    }
    if(!found)
      prevPair.next = newPair;
    return oldValue;
  }
  public V get(Object key) {
    int index = Math.abs(key.hashCode()) % SIZE;
    for(Entry<K,V> pair = buckets[index]; pair != null;
      pair = pair.next)
      if(pair.getKey().equals(key))
        return pair.getValue();
    return null;
  }
  public Set<Map.Entry<K,V>> entrySet() {
    Set<Map.Entry<K,V>> set = new HashSet<Map.Entry<K,V>>();
    for(Entry<K,V> bucket : buckets) {
      for(Entry<K,V> pair = bucket; pair != null;
        pair = pair.next)
        set.add(pair);
    }
    return set;
  }
  @SuppressWarnings("unchecked")
  public void clear() {
    // Effectively erase everything by allocating
```

```
    // a new empty array of buckets:
    buckets = new Entry[SIZE];
  }
  public V remove(Object key) {
    // Code is copied from get(), except that
    // before returning the value, the Map.Entry is
    // removed from the "list":
    int index = Math.abs(key.hashCode()) % SIZE;
    Entry<K,V> prevPair = null;        // Previous element
    for(Entry<K,V> pair = buckets[index] ; pair != null;
      pair = pair.next) {
      if(pair.getKey().equals(key)) {
        V value = pair.getValue();
        if(prevPair != null)
          prevPair.next = pair.next;
        else
          buckets[index] = null;
        return value;
      }
    }
    return null;
  }
  public int size() {
    // Could rely on the inherited implementation, which
    // returns entrySet().size(), but this is faster:
    int sz = 0;
    for(Entry<K,V> bucket : buckets)
      for(Entry<K,V> pair = bucket; pair != null;
        pair = pair.next)
        sz++;
    return sz;
  }
  public boolean isEmpty() {
    // Could just say return size() == 0;
    // but this is faster:
    for(Entry<K,V> bucket : buckets)
      if(bucket != null)
        return false;
    return true;
  }
  public boolean containsKey(Object key) {
    // A slight modification of the previous method:
    for(Entry<K,V> bucket : buckets) {
      for(Entry<K,V> pair = bucket; pair != null;
        pair = pair.next)
        if(pair.getKey().equals(key))
          return true;
```

```
    }
    return false;
  }
}

public class E25_FullSimpleHashMap2 {
  public static void main(String args[]) {
    SimpleHashMap5<String,String> m =
      new SimpleHashMap5<String,String>(),
    m2 = new SimpleHashMap5<String,String>();
    m.putAll(Countries.capitals(10));
    m2.putAll(Countries.capitals(10));
    print("m.size() = " + m.size());
    print("m.isEmpty() = " + m.isEmpty());
    print("m.equals(m2) = " + m.equals(m2));
    print("m.containsKey(\"BENIN\") = " +
      m.containsKey("BENIN"));
    print("m.containsKey(\"MARS\") = " +
      m.containsKey("MARS"));
    print("m.keySet() = " + m.keySet());
    print("m.values() = " + m.values());
    m.remove("ALGERIA");
    print("m = " + m);
  }
} /* Output:
m.size() = 10
m.isEmpty() = false
m.equals(m2) = true
m.containsKey("BENIN") = true
m.containsKey("MARS") = false
m.keySet() = [ANGOLA, CHAD, CAPE VERDE, ALGERIA, BURKINA
FASO, CENTRAL AFRICAN REPUBLIC, BENIN, BOTSWANA, BURUNDI,
CAMEROON]
m.values() = [Luanda, N'djamena, Praia, Algiers,
Ouagadougou, Bangui, Porto-Novo, Gaberone, Bujumbura,
Yaounde]
m = {ANGOLA=Luanda, CHAD=N'djamena, CAPE VERDE=Praia,
BURKINA FASO=Ouagadougou, CENTRAL AFRICAN REPUBLIC=Bangui,
BENIN=Porto-Novo, BOTSWANA=Gaberone, BURUNDI=Bujumbura,
CAMEROON=Yaounde}
*///:~
```

Exercise 26

```
//: containers/E26_CountedString2.java
/****************** Exercise 26 *****************
```

```
 * Add a char field to CountedString that is also
 * initialized in the constructor, and modify the
 * hashCode() and equals() methods to include
 * the value of this char.
 *********************************************/
package containers;
import java.util.*;
import static net.mindview.util.Print.*;

class CountedString2 {
  private static List<String> created =
    new ArrayList<String>();
  private String s;
  private char c;
  private int id;
  public CountedString2(String str, char ci) {
    s = str;
    c = ci;
    created.add(s);
    // id is the total number of instances
    // of this string in use by CountedString2:
    for(String s2 : created)
      if(s2.equals(s))
        id++;
  }
  public String toString() {
    return "String: " + s + " id: " + id + " char: " + c +
      " hashCode(): " + hashCode();
  }
  public int hashCode() {
    // The very simple approach:
    // return s.hashCode() * id;
    // Using Joshua Bloch's recipe:
    int result = 17;
    result = 37 * result + s.hashCode();
    result = 37 * result + id;
    result = 37 * result + c;
    return result;
  }
  public boolean equals(Object o) {
    return o instanceof CountedString2 &&
      s.equals(((CountedString2)o).s) &&
      id == ((CountedString2)o).id &&
      c == ((CountedString2)o).c;
  }
}
```

```
public class E26_CountedString2 {
  public static void main(String[] args) {
    Map<CountedString2,Integer> map =
      new HashMap<CountedString2,Integer>();
    CountedString2[] cs = new CountedString2[5];
    for(int i = 0; i < cs.length; i++) {
      cs[i] = new CountedString2("hi", 'c');
      map.put(cs[i], i); // Autobox int -> Integer
    }
    print(map);
    for(CountedString2 cstring : cs) {
      print("Looking up " + cstring);
      print(map.get(cstring));
    }
  }
} /* Output:
{String: hi id: 2 char: c hashCode(): 5418675=1, String: hi
id: 4 char: c hashCode(): 5418749=3, String: hi id: 5 char:
c hashCode(): 5418786=4, String: hi id: 1 char: c
hashCode(): 5418638=0, String: hi id: 3 char: c hashCode():
5418712=2}
Looking up String: hi id: 1 char: c hashCode(): 5418638
0
Looking up String: hi id: 2 char: c hashCode(): 5418675
1
Looking up String: hi id: 3 char: c hashCode(): 5418712
2
Looking up String: hi id: 4 char: c hashCode(): 5418749
3
Looking up String: hi id: 5 char: c hashCode(): 5418786
4
*///:~
```

Notice the addition of **char c** in **toString( )**, **hashCode( )** and **equals( )**.

Exercise 27

```
//: containers/E27_CountedString3.java
/****************** Exercise 27 *****************
 * Modify the hashCode() in CountedString.java by
 * removing the combination with id, and demonstrate
 * that CountedString still works as a key. What is
 * the problem with this approach?
 ************************************************/
package containers;
import java.util.*;
```

```
import static net.mindview.util.Print.*;

class CountedString3 {
  private static List<String> created =
    new ArrayList<String>();
  private String s;
  private int id = 0;
  public CountedString3(String str) {
    s = str;
    created.add(s);
    // id is the total number of instances
    // of this string in use by CountedString3:
    for(String s2 : created)
      if(s2.equals(s))
        id++;
  }
  public String toString() {
    return "String: " + s + " id: " + id +
      " hashCode(): " + hashCode();
  }
  public int hashCode() {
    // The very simple approach:
    // return s.hashCode();
    // Using Joshua Bloch's recipe:
    int result = 17;
    result = 37 * result + s.hashCode();
    // result = 37 * result + id;
    return result;
  }
  public boolean equals(Object o) {
    return o instanceof CountedString3 &&
      s.equals(((CountedString3)o).s) &&
      id == ((CountedString3)o).id;
  }
}

public class E27_CountedString3 {
  public static void main(String[] args) {
    Map<CountedString3,Integer> map =
      new HashMap<CountedString3,Integer>();
    CountedString3[] cs = new CountedString3[5];
    for(int i = 0; i < cs.length; i++) {
      cs[i] = new CountedString3("hi");
      map.put(cs[i], i); // Autobox int -> Integer
    }
    for(CountedString3 cstring : cs) {
      print("Looking up " + cstring);
```

```
      print(map.get(cstring));
    }
  }
} /* Output:
Looking up String: hi id: 1 hashCode(): 3958
0
Looking up String: hi id: 2 hashCode(): 3958
1
Looking up String: hi id: 3 hashCode(): 3958
2
Looking up String: hi id: 4 hashCode(): 3958
3
Looking up String: hi id: 5 hashCode(): 3958
4
*///:~
```

The only change to the original program is the line:

```
    // result = 37 * result + id;
```

Prior to this line, each object has a unique hash code. After commenting this line, each object produces the same hash code. The lookup still produces the correct values, but each object hashes to the same one. We lose the performance advantage of hashing when we must use **equals( )** to compare values until we find the right object.

Exercise 28

```
//: containers/E28_Tuple.java
/****************** Exercise 28 *****************
 * Modify net/mindview/util/Tuple.java to make it
 * a general-purpose class by adding hashCode(),
 * equals(), and implementing Comparable for each
 * type of Tuple.
 ***********************************************/
package containers;
import static net.mindview.util.Print.*;

class Tuple {
  public static class T2<A,B>
  implements Comparable<T2<A,B>> {
    private final A first;
    private final B second;
    public T2(A a, B b) {
      first = a;
      second = b;
```

```
    }
    public String toString() {
      return "(" + first + ", " + second + ")";
    }
    public int hashCode() {
      int result = 17;
      if(first != null)
        result = result * 37 + first.hashCode();
      if(second != null)
        result = result * 37 + second.hashCode();
      return result;
    }
    public boolean equals(Object obj) {
      if(obj instanceof T2) {
        @SuppressWarnings("unchecked")
        T2<A,B> o = (T2<A,B>)obj;
        return (first == null ?
          o.first == null : first.equals(o.first)) &&
          (second == null ?
          o.second == null : second.equals(o.second));
      }
      return false;
    }
    @SuppressWarnings("unchecked")
    public int compareTo(T2<A,B> o) {
      int res = ((Comparable<A>)first).compareTo(o.first);
      if(res != 0) return res;
      return ((Comparable<B>)second).compareTo(o.second);
    }
    public A getFirst() { return first; }
    public B getSecond() { return second; }
  }
  public static class T3<A,B,C>
  implements Comparable<T3<A,B,C>> {
    private final T2<A,B> tuple;
    private final C third;
    public T3(A a, B b, C c) {
      tuple = new T2<A,B>(a, b);
      third = c;
    }
    public String toString() {
      return "(" + tuple.getFirst() + ", " +
        tuple.getSecond() + ", " + third +")";
    }
    public int hashCode() {
      int result = tuple.hashCode();
      if(third != null)
```

```
        result = result * 37 + third.hashCode();
      return result;
    }
    public boolean equals(Object obj) {
      if(obj instanceof T3) {
        @SuppressWarnings("unchecked")
        T3<A,B,C> o = (T3<A,B,C>)obj;
        return (third == null ?
          o.third == null : third.equals(o.third)) &&
          tuple.equals(o.tuple);
      }
      return false;
    }
    @SuppressWarnings("unchecked")
    public int compareTo(T3<A,B,C> o) {
      int res = tuple.compareTo(o.tuple);
      if(res != 0) return res;
      return ((Comparable<C>)third).compareTo(o.third);
    }
    public A getFirst() { return tuple.getFirst(); }
    public B getSecond() { return tuple.getSecond(); }
    public C getThird() { return third; }
  }
  public static class T4<A,B,C,D>
  implements Comparable<T4<A,B,C,D>> {
    private final T3<A,B,C> tuple;
    private final D fourth;
    public T4(A a, B b, C c, D d) {
      tuple = new T3<A,B,C>(a, b, c);
      fourth = d;
    }
    public String toString() {
      return "(" + tuple.getFirst() + ", " +
        tuple.getSecond() + ", " + tuple.getThird() +
        ", " + fourth + ")";
    }
    public int hashCode() {
      int result = tuple.hashCode();
      if(fourth != null)
        result = result * 37 + fourth.hashCode();
      return result;
    }
    public boolean equals(Object obj) {
      if(obj instanceof T4) {
        @SuppressWarnings("unchecked")
        T4<A,B,C,D> o = (T4<A,B,C,D>)obj;
        return (fourth == null ?
```

```
          o.fourth == null : fourth.equals(o.fourth)) &&
          tuple.equals(o.tuple);
      }
      return false;
    }
    @SuppressWarnings("unchecked")
    public int compareTo(T4<A,B,C,D> o) {
      int res = tuple.compareTo(o.tuple);
      if(res != 0) return res;
      return ((Comparable<D>)fourth).compareTo(o.fourth);
    }
    public A getFirst() { return tuple.getFirst(); }
    public B getSecond() { return tuple.getSecond(); }
    public C getThird() { return tuple.getThird(); }
    public D getFourth() { return fourth; }
  }
  public static class T5<A,B,C,D,E>
  implements Comparable<T5<A,B,C,D,E>> {
    private final T4<A,B,C,D> tuple;
    private final E fifth;
    public T5(A a, B b, C c, D d, E e) {
      tuple = new T4<A,B,C,D>(a, b, c, d);
      fifth = e;
    }
    public String toString() {
      return "(" + tuple.getFirst() + ", " +
        tuple.getSecond() + ", " + tuple.getThird() +
        ", " + tuple.getFourth() + ", " + fifth + ")";
    }
    public int hashCode() {
      int result = tuple.hashCode();
      if(fifth != null)
        result = result * 37 + fifth.hashCode();
      return result;
    }
    public boolean equals(Object obj) {
      if(obj instanceof T5) {
        @SuppressWarnings("unchecked")
        T5<A,B,C,D,E> o = (T5<A,B,C,D,E>)obj;
        return (fifth == null ?
          o.fifth == null : fifth.equals(o.fifth)) &&
          tuple.equals(o.tuple);
      }
      return false;
    }
    @SuppressWarnings("unchecked")
    public int compareTo(T5<A,B,C,D,E> o) {
```

```
      int res = tuple.compareTo(o.tuple);
      if(res != 0) return res;
      return ((Comparable<E>)fifth).compareTo(o.fifth);
    }
    public A getFirst() { return tuple.getFirst(); }
    public B getSecond() { return tuple.getSecond(); }
    public C getThird() { return tuple.getThird(); }
    public D getFourth() { return tuple.getFourth(); }
    public E getFifth() { return fifth; }
  }
  public static <A,B> T2<A,B> tuple(A a, B b) {
    return new T2<A,B>(a, b);
  }
  public static <A,B,C> T3<A,B,C> tuple(A a, B b, C c) {
    return new T3<A,B,C>(a, b, c);
  }
  public static <A,B,C,D> T4<A,B,C,D>
  tuple(A a, B b, C c, D d) {
    return new T4<A,B,C,D>(a, b, c, d);
  }
  public static <A,B,C,D,E>
  T5<A,B,C,D,E> tuple(A a, B b, C c, D d, E e) {
    return new T5<A,B,C,D,E>(a, b, c, d, e);
  }
}

public class E28_Tuple {
  public static void main(String[] args) {
    Tuple.T5<String,Integer,Boolean,Short,Long>
      t5_1 = Tuple.tuple("a", 1, false, (short)2, 3L),
      t5_2 = Tuple.tuple("b", 2, true, (short)3, 4L);
    print("t5_1 = " + t5_1);
    print("t5_2 = " + t5_2);
    print("t5_1.equals(t5_1) = " + t5_1.equals(t5_1));
    print("t5_1.equals(t5_2) = " + t5_1.equals(t5_2));
    print("t5_1.compareTo(t5_1) = " + t5_1.compareTo(t5_1));
    print("t5_1.compareTo(t5_2) = " + t5_1.compareTo(t5_2));
  }
} /* Output:
t5_1 = (a, 1, false, 2, 3)
t5_2 = (b, 2, true, 3, 4)
t5_1.equals(t5_1) = true
t5_1.equals(t5_2) = false
t5_1.compareTo(t5_1) = 0
t5_1.compareTo(t5_2) = -1
*///:~
```

We refactor the code in **net/mindview/util/Tuple.java** to incorporate the **Comparable** for each type of **Tuple**. When a class extends a concrete **Comparable** class (as **T3<A,B,C>** extends **T2<A,B>** in the original program), and adds a significant field, there is no way to simultaneously implement **compareTo( )** correctly. Therefore we use composition instead of inheritance. The same holds true for **equals( )**. For more information see *Effective Java* by Joshua Bloch (Addison-Wesley, 2001). Our solution follows Bloch's recommendations.

Exercise 29

```
//: containers/E29_ListPerformance2.java
// {Args: 100 500} Small to keep build testing short
/****************** Exercise 29 *****************
 * Modify ListPerformance.java so that the Lists hold
 * String objects instead of Integers. Use a Generator
 * from the Arrays chapter to create test values.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

public class E29_ListPerformance2 {
  static Generator<String> gen =
    new CountingGenerator.String();
  static Random rand = new Random();
  static int reps = 1000;
  static List<Test<List<String>>> tests =
    new ArrayList<Test<List<String>>>();
  static List<Test<LinkedList<String>>> qTests =
    new ArrayList<Test<LinkedList<String>>>();
  static {
    tests.add(new Test<List<String>>("add") {
      int test(List<String> list, TestParam tp) {
        int loops = tp.loops;
        int listSize = tp.size;
        for(int i = 0; i < loops; i++) {
          list.clear();
          for(int j = 0; j < listSize; j++)
            list.add(gen.next());
        }
        return loops * listSize;
      }
    });
    tests.add(new Test<List<String>>("get") {
```

```
      int test(List<String> list, TestParam tp) {
        int loops = tp.loops * reps;
        int listSize = list.size();
        for(int i = 0; i < loops; i++)
          list.get(rand.nextInt(listSize));
        return loops;
      }
    });
    tests.add(new Test<List<String>>("set") {
      int test(List<String> list, TestParam tp) {
        int loops = tp.loops * reps;
        int listSize = list.size();
        for(int i = 0; i < loops; i++)
          list.set(rand.nextInt(listSize), "Hello");
        return loops;
      }
    });
    tests.add(new Test<List<String>>("iteradd") {
      int test(List<String> list, TestParam tp) {
        final int LOOPS = 1000000;
        int half = list.size() / 2;
        ListIterator<String> it = list.listIterator(half);
        for(int i = 0; i < LOOPS; i++)
          it.add("Hello");
        return LOOPS;
      }
    });
    tests.add(new Test<List<String>>("insert") {
      int test(List<String> list, TestParam tp) {
        int loops = tp.loops;
        for(int i = 0; i < loops; i++)
          // Minimize random-access cost
          list.add(5, "Hello");
        return loops;
      }
    });
    tests.add(new Test<List<String>>("remove") {
      int test(List<String> list, TestParam tp) {
        int loops = tp.loops;
        int size = tp.size;
        for(int i = 0; i < loops; i++) {
          list.clear();
          list.addAll(CollectionData.list(
            new CountingGenerator.String(), size));
          while(list.size() > 5)
            list.remove(5); // Minimize random-access cost
        }
```

```
        return loops * size;
      }
    });
    // Tests for queue behavior:
    qTests.add(new Test<LinkedList<String>>("addFirst") {
      int test(LinkedList<String> list, TestParam tp) {
        int loops = tp.loops;
        int size = tp.size;
        for(int i = 0; i < loops; i++) {
          list.clear();
          for(int j = 0; j < size; j++)
            list.addFirst("Hello");
        }
        return loops * size;
      }
    });
    qTests.add(new Test<LinkedList<String>>("addLast") {
      int test(LinkedList<String> list, TestParam tp) {
        int loops = tp.loops;
        int size = tp.size;
        for(int i = 0; i < loops; i++) {
          list.clear();
          for(int j = 0; j < size; j++)
            list.addLast("Hello");
        }
        return loops * size;
      }
    });
    qTests.add(
      new Test<LinkedList<String>>("rmFirst") {
        int test(LinkedList<String> list, TestParam tp) {
          int loops = tp.loops;
          int size = tp.size;
          for(int i = 0; i < loops; i++) {
            list.clear();
            list.addAll(CollectionData.list(
              new CountingGenerator.String(), size));
            while(list.size() > 0)
              list.removeFirst();
          }
          return loops * size;
        }
      });
    qTests.add(new Test<LinkedList<String>>("rmLast") {
      int test(LinkedList<String> list, TestParam tp) {
        int loops = tp.loops;
        int size = tp.size;
```

```
        for(int i = 0; i < loops; i++) {
          list.clear();
          list.addAll(CollectionData.list(
              new CountingGenerator.String(), size));
          while(list.size() > 0)
            list.removeLast();
        }
        return loops * size;
      }
    });
  }
  static class ListTester extends Tester<List<String>> {
    public ListTester(List<String> container,
        List<Test<List<String>>> tests) {
      super(container, tests);
    }
    // Fill to the appropriate size before each test:
    @Override protected List<String> initialize(int size){
      container.clear();
      container.addAll(CollectionData.list(
        new CountingGenerator.String(), size));
      return container;
    }
    // Convenience method:
    public static void run(List<String> list,
        List<Test<List<String>>> tests) {
      new ListTester(list, tests).timedTest();
    }
  }
  public static void main(String[] args) {
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    // Can only do these two tests on an array:
    Tester<List<String>> arrayTest =
      new Tester<List<String>>(null, tests.subList(1, 3)){
        // This will be called before each test. It
        // produces a non-resizeable array-backed list:
        @Override protected
        List<String> initialize(int size) {
          String[] sa = Generated.array(String.class,
            new CountingGenerator.String(), size);
          return Arrays.asList(sa);
        }
      };
    arrayTest.setHeadline("Array as List");
    arrayTest.timedTest();
    Tester.defaultParams = TestParam.array(
```

```
      10, 5000, 100, 5000, 1000, 1000, 10000, 200);
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    ListTester.run(new ArrayList<String>(), tests);
    ListTester.run(new LinkedList<String>(), tests);
    ListTester.run(new Vector<String>(), tests);
    Tester.fieldWidth = 12;
    Tester<LinkedList<String>> qTest =
      new Tester<LinkedList<String>>(
        new LinkedList<String>(), qTests);
    qTest.setHeadline("Queue tests");
    qTest.timedTest();
  }
} /* Output: (Sample)
--- Array as List ---
 size     get     set
  100     106     104
-------------------- ArrayList --------------------
 size     add     get     set iteradd  insert  remove
  100     541      97      99     322     791     748
-------------------- LinkedList --------------------
 size     add     get     set iteradd  insert  remove
  100     471     175     159     447     305     546
---------------------- Vector ----------------------
 size     add     get     set iteradd  insert  remove
  100     485     126     191     310     905     881
------------------- Queue tests --------------------
 size    addFirst     addLast     rmFirst      rmLast
  100          69          78         516         504
*///:~
```

Exercise 30

```
//: containers/E30_SortPerformance.java
// {Args: 1000 500}
/****************** Exercise 30 *****************
 * Compare the performance of Collections.sort()
 * between an ArrayList and a LinkedList.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

public class E30_SortPerformance {
  static List<Test<List<Integer>>> tests =
    new ArrayList<Test<List<Integer>>>();
```

```
  static {
    tests.add(new Test<List<Integer>>("sort") {
      int test(List<Integer> list, TestParam tp) {
        for(int i = 0; i < tp.loops; i++) {
          list.clear();
          list.addAll(CollectionData.list(
            new RandomGenerator.Integer(), tp.size));
          Collections.sort(list);
        }
        return tp.loops;
      }
    });
  }
  static class ListTester extends Tester<List<Integer>> {
    public ListTester(List<Integer> container,
        List<Test<List<Integer>>> tests) {
      super(container, tests);
    }
    // Fill to the appropriate size before each test:
    @Override protected List<Integer> initialize(int size){
      container.clear();
      container.addAll(CollectionData.list(
        new RandomGenerator.Integer(), size));
      return container;
    }
    // Convenience method:
    public static void run(List<Integer> list,
        List<Test<List<Integer>>> tests) {
      new ListTester(list, tests).timedTest();
    }
  }
  public static void main(String[] args) {
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    ListTester.run(new ArrayList<Integer>(), tests);
    ListTester.run(new LinkedList<Integer>(), tests);
  }
} /* Output: (Sample)
- ArrayList -
 size    sort
 1000  529815
- LinkedList -
 size    sort
 1000  527436
*///:~
```

Both maps perform similarly in **Collections.sort( )**, which makes sense given the way we implement it (for more details consult the JDK).

Don't be fooled by benchmarking results. *TIJ4*'s *Microbenchmarking dangers* section (see the index) explains some common traps to avoid.

Exercise 31

```
//: containers/E31_StringContainer.java
/****************** Exercise 31 ******************
 * Create a container that encapsulates an array of
 * String, and that only allows adding Strings and
 * getting Strings, so that there are no casting
 * issues during use. If the internal array isn't big
 * enough for the next add, your container should
 * automatically resize it. In main(), compare the
 * performance of your container with an
 * ArrayList<String>.
 *********************************************/
package containers;
import java.util.*;

class StringContainer {
  private String[] array;
  private int index;
  private static final int INCR = 255;
  public StringContainer() {
    array = new String[10];
  }
  public StringContainer(int sz) {
    array = new String[sz];
  }
  public void add(String s) {
    if(index >= array.length) {
      String[] tmp = new String[array.length + INCR];
      for(int i = 0; i < array.length; i++)
        tmp[i] = array[i];
      index = array.length;
      array = tmp;
    }
    array[index++] = s;
  }
  public String get(int i) { return array[i]; }
  public int size() { return index; }
}
```

```
public class E31_StringContainer {
  static final int LOOPS = 10000;
  static List<Test<List<String>>> alTests =
    new ArrayList<Test<List<String>>>();
  static List<Test<StringContainer>> scTests =
    new ArrayList<Test<StringContainer>>();
  static {
    alTests.add(new Test<List<String>>("addget") {
      int test(List<String> list, TestParam tp) {
        for(int i = 0; i < LOOPS; i++) {
          list.add(Integer.toString(i));
          list.get(i);
        }
        return LOOPS;
      }
    });
    scTests.add(new Test<StringContainer>("addget") {
      int test(StringContainer sc, TestParam tp) {
        for(int i = 0; i < LOOPS; i++) {
          sc.add(Integer.toString(i));
          sc.get(i);
        }
        return LOOPS;
      }
    });
  }
  public static void main(String[] args) {
    // Parameters are also hard-coded inside tests.
    Tester.defaultParams = TestParam.array(LOOPS, 1);
    Tester.run(new ArrayList<String>(LOOPS), alTests);
    Tester.run(new StringContainer(LOOPS), scTests);
  }
} /* Output: (Sample)
- ArrayList -
 size  addget
10000    1266
 StringContainer
 size  addget
10000     574
*///:~
```

This simple though somewhat inaccurate test is from *TIJ4*. Note that the **ArrayList** performance is close to what you might hand-code; this emphasizes that you shouldn't waste time writing containers yourself when you can use the standard library. Wait to optimize code until you find the bottleneck by thoroughly profiling your application.

Exercise 32

```
//: containers/E32_IntContainer.java
/****************** Exercise 32 ******************
 * Repeat the previous exercise for a container of
 * int, and compare the performance to an
 * ArrayList<Integer>. In your performance comparison,
 * include the process of incrementing each object
 * in the container.
 ***********************************************/
package containers;
import java.util.*;

class IntContainer {
  private int[] array;
  private int index;
  private static final int INCR = 255;
  public IntContainer() {
    array = new int[10];
  }
  public IntContainer(int sz) {
    array = new int[sz];
  }
  public void add(int i) {
    if(index >= array.length) {
      int[] tmp = new int[array.length + INCR];
      for(int j = 0; j < array.length; j++)
        tmp[j] = array[j];
      index = array.length;
      array = tmp;
    }
    array[index++] = i;
  }
  public int get(int i) { return array[i]; }
  public void set(int i, int val) { array[i] = val; }
  public int size() { return index; }
}

public class E32_IntContainer {
  static final int LOOPS = 10000;
  static List<Test<List<Integer>>> alTests =
    new ArrayList<Test<List<Integer>>>();
  static List<Test<IntContainer>> icTests =
    new ArrayList<Test<IntContainer>>();
  static {
    alTests.add(new Test<List<Integer>>("addget") {
```

```
      int test(List<Integer> list, TestParam tp) {
        for(int i = 0; i < LOOPS; i++) {
          list.add(i);
          list.set(i, list.get(i) + 1);
        }
        return LOOPS;
      }
    });
    icTests.add(new Test<IntContainer>("addget") {
      int test(IntContainer ic, TestParam tp) {
        for(int i = 0; i < LOOPS; i++) {
          ic.add(i);
          ic.set(i, ic.get(i) + 1);
        }
        return LOOPS;
      }
    });
  }
  public static void main(String[] args) {
    // Parameters are also hard-coded inside tests.
    Tester.defaultParams = TestParam.array(LOOPS, 1);
    Tester.run(new ArrayList<Integer>(LOOPS), alTests);
    Tester.run(new IntContainer(LOOPS), icTests);
  }
} /* Output: (Sample)
- ArrayList -
 size  addget
10000    8975
 IntContainer
 size  addget
10000     491
*///:~
```

IntContainer is significantly faster than **ArrayList**.

Exercise 33

```
//: containers/E33_ListPerformance3.java
// {RunByHand} (Takes too long during the build process)
/******************** Exercise 33 ************************
 * Create a FastTraversalLinkedList that internally uses a
 * LinkedList for rapid insertions and removals, and an
 * ArrayList for rapid traversals and get() operations.
 * Test it by modifying ListPerformance.java.
 *********************************************************/
package containers;
```

```
import java.util.*;
import net.mindview.util.*;

class FastTraversalLinkedList<T> extends AbstractList<T> {
  private class FlaggedArrayList {
    private FlaggedLinkedList companion;
    boolean changed = false;
    private ArrayList<T> list = new ArrayList<T>();
    public void addCompanion(FlaggedLinkedList other) {
      companion = other;
    }
    private void synchronize() {
      if(companion.changed) {
        list = new ArrayList<T>(companion.list);
        companion.changed = false;
      }
    }
    public T get(int index) {
      synchronize();
      return list.get(index);
    }
    public int size() {
      synchronize();
      return list.size();
    }
    public T remove(int index) {
      synchronize();
      changed = true;
      return list.remove(index);
    }
    public boolean remove(Object item) {
      synchronize();
      changed = true;
      return list.remove(item);
    }
    // Always broadcasted to the companion container, too.
    public void clear() {
      list.clear();
      changed = false;
    }
  }
  private class FlaggedLinkedList {
    private FlaggedArrayList companion;
    boolean changed = false;
    LinkedList<T> list = new LinkedList<T>();
    public void addCompanion(FlaggedArrayList other) {
      companion = other;
```

```
    }
    private void synchronize() {
      if(companion.changed) {
        list = new LinkedList<T>(companion.list);
        companion.changed = false;
      }
    }
    public void add(int index, T item) {
      synchronize();
      changed = true;
      list.add(index, item);
    }
    public boolean add(T item) {
      synchronize();
      changed = true;
      return list.add(item);
    }
    public Iterator<T> iterator() {
      synchronize();
      return list.iterator();
    }
    // Always broadcasted to the companion container, too.
    public void clear() {
      list.clear();
      changed = false;
    }
  }
  private FlaggedArrayList aList =
    new FlaggedArrayList();
  private FlaggedLinkedList lList =
    new FlaggedLinkedList();
  // Connect the two so they can synchronize:
  {
    aList.addCompanion(lList);
    lList.addCompanion(aList);
  }
  public int size() { return aList.size(); }
  public T get(int arg) { return aList.get(arg); }
  public void add(int index, T item) {
    lList.add(index, item);
  }
  public boolean add(T item) {
    return lList.add(item);
  }
  // Through testing, we discovered that the ArrayList is
  // actually much faster for removals than the LinkedList:
  public T remove(int index) {
```

```
    return aList.remove(index);
  }
  public boolean remove(Object item) {
    return aList.remove(item);
  }
  public Iterator<T> iterator() {
    return lList.iterator();
  }
  public void clear() {
    aList.clear();
    lList.clear();
  }
}

public class E33_ListPerformance3 {
  static Random rand = new Random();
  static int reps = 1000;
  static List<Test<List<Integer>>> tests =
    new ArrayList<Test<List<Integer>>>();
  static {
    tests.add(new Test<List<Integer>>("iter") {
      int test(List<Integer> list, TestParam tp) {
        for(int i = 0; i < tp.loops; i++) {
          Iterator<Integer> it = list.iterator();
          while(it.hasNext())
            it.next(); // Produces value
        }
        return tp.loops;
      }
    });
    tests.add(new Test<List<Integer>>("get") {
      int test(List<Integer> list, TestParam tp) {
        int loops = tp.loops * reps;
        int listSize = list.size();
        for(int i = 0; i < loops; i++)
          list.get(rand.nextInt(listSize));
        return loops;
      }
    });
    tests.add(new Test<List<Integer>>("insert") {
      int test(List<Integer> list, TestParam tp) {
        int loops = tp.loops;
        for(int i = 0; i < loops; i++)
          list.add(5, 47); // Minimize random-access cost
        return loops;
      }
    });
```

```
    tests.add(new Test<List<Integer>>("remove_I ") {
      int test(List<Integer> list, TestParam tp) {
        int count = 0;
        for(int i = list.size() / 2 ; i < list.size();
            i++) {
          ++count;
          list.remove(i);
        }
        return count;
      }
    });
    tests.add(new Test<List<Integer>>("remove_O") {
      int test(List<Integer> list, TestParam tp) {
        int count = 0;
        for(int i = list.size() / 2 ; i < list.size();
            i++) {
          ++count;
          list.remove(list.get(i));
        }
        return count;
      }
    });
  }
  static class ListTester extends Tester<List<Integer>> {
    public ListTester(List<Integer> container,
        List<Test<List<Integer>>> tests) {
      super(container, tests);
    }
    // Fill to the appropriate size before each test:
    @Override protected List<Integer> initialize(int size){
      container.clear();
      container.addAll(new CountingIntegerList(size));
      return container;
    }
    // Convenience method:
    public static void run(List<Integer> list,
        List<Test<List<Integer>>> tests) {
      new ListTester(list, tests).timedTest();
    }
  }
  public static void main(String[] args) {
    ListTester.run(new ArrayList<Integer>(), tests);
    ListTester.run(new LinkedList<Integer>(), tests);
    ListTester.run(
      new FastTraversalLinkedList<Integer>(), tests);
  }
} ///:~
```

When you perform an operation on either **FlaggedArrayList** or **FlaggedLinkedList**, it checks for changes in the other and self-updates, setting or clearing both lists' **changed** flags. You can see the two lists connected in the instance initialization clause.

We implement only the methods used in the tests in **FlaggedArrayList** and **FlaggedLinkedList**. The **FastTraversalLinkedList** forwards method calls to the most efficient list.

The insertion and removal tests don't use the iterator (which would confound the results). Also, we distinguish "remove at location" as 'remove_I' and "remove by object" as 'remove_O'. The results of one run are as follows:

```
----------------- ArrayList -----------------
 size    iter     get  insertremove_I remove_O
   10    1302     106    4231   12329   16513
  100    4104      96    3701    1005   47717
 1000   33924      95    4846    1108    9171
10000  339132      92   14295    4124   85110
----------------- LinkedList -----------------
 size    iter     get  insertremove_I remove_O
   10    1104     125     676  118204  120866
  100    2489     165      68     783   34911
 1000   24384     769      69    1071   10800
10000  245605   13097      87   14333   98771
---------- FastTraversalLinkedList ----------
 size    iter     get  insertremove_I remove_O
   10     512     108     235   26553   20298
  100    2491     108     195    1553    2008
 1000   24158     107      80    1292   12888
10000  243113     103      97    4062   94922
```

Surprisingly, removing an element in the middle of an **ArrayList** turns out to be far less damaging than removing one from the middle of a **LinkedList**. Iteration is somewhat better in **LinkedList**, which is noticeably faster only with insertions.

FastTraversalLinkedList's repeated insertion and removal entails list copying which can negatively affect overhead.

Exercise 34

```
//: containers/E34_StringSetPerformance.java
// {Args: 1000 500} Medium to keep build testing reasonable
/******************** Exercise 34 ************************
 * Modify SetPerformance.java so that the Sets hold String
```

```
 * objects instead of Integers. Use a Generator from the
 * Arrays chapter to create test values.
 *****************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

public class E34_StringSetPerformance {
  static List<Test<Set<String>>> tests =
    new ArrayList<Test<Set<String>>>();
  static {
    tests.add(new Test<Set<String>>("add") {
      int test(Set<String> set, TestParam tp) {
        Generator<String> gen;
        int loops = tp.loops;
        int size = tp.size;
        for(int i = 0; i < loops; i++) {
          set.clear();
          // Always starts the sequence from the beginning.
          gen = new CountingGenerator.String();
          for(int j = 0; j < size; j++)
            set.add(gen.next());
        }
        return loops * size;
      }
    });
    tests.add(new Test<Set<String>>("contains") {
      int test(Set<String> set, TestParam tp) {
        Generator<String> gen =
          new CountingGenerator.String(5);
        int loops = tp.loops;
        int span = tp.size * 2;
        for(int i = 0; i < loops; i++)
          for(int j = 0; j < span; j++)
            set.contains(gen.next());
        return loops * span;
      }
    });
    tests.add(new Test<Set<String>>("iterate") {
      int test(Set<String> set, TestParam tp) {
        int loops = tp.loops * 10;
        for(int i = 0; i < loops; i++) {
          Iterator<String> it = set.iterator();
          while(it.hasNext())
            it.next();
        }
        return loops * set.size();
```

```
      }
    });
  }
  public static void main(String[] args) {
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    Tester.fieldWidth = 10;
    Tester.run(new TreeSet<String>(), tests);
    Tester.run(new HashSet<String>(), tests);
    Tester.run(new LinkedHashSet<String>(), tests);
  }
} /* Output: (Sample)
------------- TreeSet -------------
 size       add  contains   iterate
 1000       687       497        58
------------- HashSet -------------
 size       add  contains   iterate
 1000       517       360        66
---------- LinkedHashSet ----------
 size       add  contains   iterate
 1000       528       338       110
*///:~
```

Exercise 35

```
//: containers/E35_MapPerformance2.java
// {Args: 100 5000} Small to keep build testing short
/******************** Exercise 35 ************************
 * Modify MapPerformance.java to include tests of SlowMap.
 *********************************************************/
package containers;
import java.util.*;

public class E35_MapPerformance2 {
  static List<Test<Map<Integer,Integer>>> tests =
    new ArrayList<Test<Map<Integer,Integer>>>();
  static {
    tests.add(new Test<Map<Integer,Integer>>("put") {
      int test(Map<Integer,Integer> map, TestParam tp) {
        int loops = tp.loops;
        int size = tp.size;
        for(int i = 0; i < loops; i++) {
          map.clear();
          for(int j = 0; j < size; j++)
            map.put(j, j);
        }
```

```
        return loops * size;
      }
    });
    tests.add(new Test<Map<Integer,Integer>>("get") {
      int test(Map<Integer,Integer> map, TestParam tp) {
        int loops = tp.loops;
        int span = tp.size * 2;
        for(int i = 0; i < loops; i++)
          for(int j = 0; j < span; j++)
            map.get(j);
        return loops * span;
      }
    });
    tests.add(new Test<Map<Integer,Integer>>("iterate") {
      int test(Map<Integer,Integer> map, TestParam tp) {
        int loops = tp.loops * 10;
        for(int i = 0; i < loops; i ++) {
          Iterator<Map.Entry<Integer, Integer>> it =
            map.entrySet().iterator();
          while(it.hasNext())
            it.next();
        }
        return loops * map.size();
      }
    });
  }
  public static void main(String[] args) {
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    Tester.run(new TreeMap<Integer,Integer>(), tests);
    Tester.run(new HashMap<Integer,Integer>(), tests);
    Tester.run(new LinkedHashMap<Integer,Integer>(),tests);
    Tester.run(
      new IdentityHashMap<Integer,Integer>(), tests);
    Tester.run(new WeakHashMap<Integer,Integer>(), tests);
    Tester.run(new Hashtable<Integer,Integer>(), tests);
    Tester.run(new SlowMap2<Integer,Integer>(), tests);
  }
} /* Output: (Sample)
---------- TreeMap ----------
 size     put     get iterate
  100     529     177      34
---------- HashMap ----------
 size     put     get iterate
  100     178      43      64
------- LinkedHashMap -------
 size     put     get iterate
```

```
  100     221      55      28
------ IdentityHashMap ------
 size     put     get iterate
  100     270     222      47
-------- WeakHashMap --------
 size     put     get iterate
  100     288     174      83
--------- Hashtable ---------
 size     put     get iterate
  100     163     102      49
---------- SlowMap2 ----------
 size     put     get iterate
  100    2078    1396      79
*///:~
```

Here, we use the **SlowMap2** class, the more solid **Map** implementation. Its performance degrades sharply as a function of size, though. Try running the test with "size 1000." **SlowMap2** clearly lives up to its name.

Exercise 36

```
//: containers/E36_MapEntrySlowMap.java
// {Args: 100 5000} Small to keep build testing short
/****************** Exercise 36 *****************
 * Modify SlowMap so that instead of two ArrayLists,
 * it holds a single ArrayList of MapEntry objects.
 * Verify that the modified version works correctly.
 * Using MapPerformance.java, test the speed of your
 * new Map. Now change the put() method so that it
 * performs a sort() after each pair is entered, and
 * modify get() to use Collections.binarySearch() to
 * look up the key. Compare the performance of the new
 * version with the old ones.
 *********************************************/
package containers;
import java.util.*;

// Does not support 'null' keys.
@SuppressWarnings("unchecked")
class MapEntrySlowMap1<K,V> extends AbstractMap<K,V> {
  protected List<MapEntry<K,V>> entries =
    new ArrayList<MapEntry<K,V>>();
  // Returns the entry matching the specified key, or null.
  protected MapEntry<K,V> getEntry(Object key) {
    for(MapEntry<K,V> entry : entries)
      if(entry.getKey().equals(key))
```

```
        return entry;
    return null;
  }
  @Override public V put(K key, V value) {
    if(key == null)
      throw new NullPointerException();
    MapEntry<K,V> oldEntry = getEntry(key);
    V oldValue = null;

    if(oldEntry == null)
      entries.add(new MapEntry<K,V>(key, value));
    else {
      oldValue = oldEntry.getValue();
      oldEntry.setValue(value);
    }
    return oldValue;
  }
  @Override public V get(Object key) {
    if(key == null)
      throw new NullPointerException();
    MapEntry<K,V> entry = getEntry(key);
    return entry == null ? null : entry.getValue();
  }
  private EntrySet entrySet = new EntrySet();
  @Override public Set<Map.Entry<K,V>> entrySet() {
    return entrySet;
  }
  // Uses the 'Flyweight' pattern
  private class EntrySet
  extends AbstractSet<Map.Entry<K,V>> {
    @Override public Iterator<Map.Entry<K,V>> iterator() {
      return new Iterator<Map.Entry<K,V>>() {
        private int index = -1;
        boolean canRemove;
        public boolean hasNext() {
          return index < entries.size() - 1;
        }
        public Map.Entry<K,V> next() {
          canRemove = true;
          ++index;
          return entries.get(index);
        }
        public void remove() {
          if(!canRemove)
            throw new IllegalStateException();
          canRemove = false;
          entries.remove(index--);
```

```
        }
      };
    }
    @Override public boolean contains(Object o) {
      if(o instanceof MapEntry) {
        MapEntry<K,V> e = (MapEntry<K,V>)o;
        return e.equals(getEntry(e.getKey()));
      }
      return false;
    }
    @Override public boolean remove(Object o) {
      if(contains(o)) {
        entries.remove((MapEntry<K,V>)o);
        return true;
      }
      return false;
    }
    @Override public int size() { return entries.size(); }
    @Override public void clear() { entries.clear(); }
  }
}

@SuppressWarnings("unchecked")
class MapEntryComp<K,V> implements
Comparator<MapEntry<K,V>> {
  public int compare(MapEntry<K,V> o1, MapEntry<K,V> o2) {
    Comparable<K> c1 = (Comparable<K>)o1.getKey();
    return c1.compareTo(o2.getKey());
  }
}

@SuppressWarnings("unchecked")
class MapEntrySlowMap2<K,V> extends MapEntrySlowMap1<K,V> {
  final MapEntryComp<K,V> comp = new MapEntryComp<K,V>();
  // Returns the entry matching the specified key, or null.
  @Override protected MapEntry<K,V> getEntry(Object key) {
    MapEntry<K,V> searchEntry =
      new MapEntry<K,V>((K)key, null);
    int index =
      Collections.binarySearch(entries, searchEntry, comp);
    if(index >= 0)
      return entries.get(index);
    return null;
  }
  @Override public V put(K key, V value) {
    if(key == null)
      throw new NullPointerException();
```

```
    MapEntry<K,V> oldEntry = getEntry(key);
    V oldValue = null;
    if(oldEntry == null) {
      entries.add(new MapEntry<K,V>(key, value));
      Collections.sort(entries, comp);
    } else {
      oldValue = oldEntry.getValue();
      oldEntry.setValue(value);
    }
    return oldValue;
  }
}

public class E36_MapEntrySlowMap {
  public static void main(String[] args) {
    // Testing correctness...
    System.out.println("Testing MapEntrySlowMap1");
    E16_SlowMapTest.test(
      new MapEntrySlowMap1<Integer,String>());
    System.out.println("Testing MapEntrySlowMap2");
    E16_SlowMapTest.test(
      new MapEntrySlowMap2<Integer,String>());
    // Measuring performance...
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    Tester.run(new TreeMap<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new HashMap<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new LinkedHashMap<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(
      new IdentityHashMap<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new WeakHashMap<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new Hashtable<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new SlowMap2<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new MapEntrySlowMap1<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new MapEntrySlowMap2<Integer,Integer>(),
      E35_MapPerformance2.tests);
  }
} /* Output: (Sample)
Testing MapEntrySlowMap1
```

```
MapEntrySlowMap1
Size = 25, Keys: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Values: [A0, B0, C0, D0, E0, F0, G0, H0, I0, J0, K0, L0, M0,
N0, O0, P0, Q0, R0, S0, T0, U0, V0, W0, X0, Y0]
{0=A0, 1=B0, 2=C0, 3=D0, 4=E0, 5=F0, 6=G0, 7=H0, 8=I0, 9=J0,
10=K0, 11=L0, 12=M0, 13=N0, 14=O0, 15=P0, 16=Q0, 17=R0,
18=S0, 19=T0, 20=U0, 21=V0, 22=W0, 23=X0, 24=Y0}
map.containsKey(11): true
map.get(11): L0
map.containsValue("F0"): true
First key in map: 0
Size = 24, Keys: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
map.isEmpty(): true
map.isEmpty(): true
Testing MapEntrySlowMap2
MapEntrySlowMap2
Size = 25, Keys: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Values: [A0, B0, C0, D0, E0, F0, G0, H0, I0, J0, K0, L0, M0,
N0, O0, P0, Q0, R0, S0, T0, U0, V0, W0, X0, Y0]
{0=A0, 1=B0, 2=C0, 3=D0, 4=E0, 5=F0, 6=G0, 7=H0, 8=I0, 9=J0,
10=K0, 11=L0, 12=M0, 13=N0, 14=O0, 15=P0, 16=Q0, 17=R0,
18=S0, 19=T0, 20=U0, 21=V0, 22=W0, 23=X0, 24=Y0}
map.containsKey(11): true
map.get(11): L0
map.containsValue("F0"): true
First key in map: 0
Size = 24, Keys: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
map.isEmpty(): true
map.isEmpty(): true
---------- TreeMap ----------
 size     put     get iterate
  100    1564     842     148
---------- HashMap ----------
 size     put     get iterate
  100     499     213     168
------- LinkedHashMap -------
 size     put     get iterate
  100     704     293     115
------ IdentityHashMap ------
 size     put     get iterate
  100     598     580     181
-------- WeakHashMap --------
 size     put     get iterate
```

```
  100     687     317     220
--------- Hashtable ---------
 size     put     get iterate
  100     460     265     219
---------- SlowMap2 ----------
 size     put     get iterate
  100    6238    6228     363
------ MapEntrySlowMap1 ------
 size     put     get iterate
  100   10334   15029     195
------ MapEntrySlowMap2 ------
 size     put     get iterate
  100   28463    1484     196
*///:~
```

The **MapEntrySlowMap2**'s **get( )** method performs well, but **put( )** is disastrous. Notice that the **@SuppressWarnings("unchecked")** annotations appear immediately before class definitions; always confine these annotations by placing them near the problem.

Exercise 37

```
//: containers/E37_SimpleHashMapArrays.java
// {Args: 100 5000} Small to keep build testing short
/****************** Exercise 37 *****************
 * Modify SimpleHashMap to use ArrayLists instead
 * of LinkedLists. Modify MapPerformance.java to
 * compare the performance of the two
 * implementations.
 ***********************************************/
package containers;
import java.util.*;

class SimpleHashMap6<K,V> extends AbstractMap<K,V> {
  static final int SIZE = 997;
  @SuppressWarnings("unchecked")
  ArrayList<MapEntry<K,V>>[] buckets =
    new ArrayList[SIZE];
  public V put(K key, V value) {
    V oldValue = null;
    int index = Math.abs(key.hashCode()) % SIZE;
    if(buckets[index] == null)
      buckets[index] = new ArrayList<MapEntry<K,V>>();
    ArrayList<MapEntry<K,V>> bucket = buckets[index];
    MapEntry<K,V> pair = new MapEntry<K,V>(key, value);
    boolean found = false;
```

```
    ListIterator<MapEntry<K,V>> it = bucket.listIterator();
    while(it.hasNext()) {
      MapEntry<K,V> iPair = it.next();
      if(iPair.getKey().equals(key)) {
        oldValue = iPair.getValue();
        it.set(pair); // Replace old with new
        found = true;
        break;
      }
    }
    if(!found)
      buckets[index].add(pair);
    return oldValue;
  }
  public V get(Object key) {
    int index = Math.abs(key.hashCode()) % SIZE;
    if(buckets[index] == null) return null;
    for(MapEntry<K,V> iPair : buckets[index])
      if(iPair.getKey().equals(key))
        return iPair.getValue();
    return null;
  }
  public Set<Map.Entry<K,V>> entrySet() {
    Set<Map.Entry<K,V>> set= new HashSet<Map.Entry<K,V>>();
    for(ArrayList<MapEntry<K,V>> bucket : buckets) {
      if(bucket == null) continue;
      for(MapEntry<K,V> mpair : bucket)
        set.add(mpair);
    }
    return set;
  }
  @SuppressWarnings("unchecked")
  public void clear() {
    // Effectively erase everything by allocating
    // a new empty array of buckets:
    buckets = new ArrayList[SIZE];
  }
  public V remove(Object key) {
    // Code is copied from get(), except that
    // before returning the value, the Map.Entry is
    // removed from the list:
    int index = Math.abs(key.hashCode()) % SIZE;
    if(buckets[index] == null) return null;
    Iterator<MapEntry<K,V>> it = buckets[index].iterator();
    while(it.hasNext()) {
      MapEntry<K,V> iPair = it.next();
      if(iPair.getKey().equals(key)) {
```

```
        // Changes are here:
        V value = iPair.getValue();
        // Removes the last fetched value:
        it.remove();
        return value;
      }
    }
    return null;
  }
  public int size() {
    // Could rely on the inherited implementation, which
    // returns entrySet().size(), but this is faster:
    int sz = 0;
    for(ArrayList<MapEntry<K,V>> bucket : buckets)
      if(bucket != null)
        sz += bucket.size();
    return sz;
  }
  public boolean isEmpty() {
    // Could just say return size() == 0;
    // but this is faster:
    for(ArrayList<MapEntry<K,V>> bucket : buckets)
      if(bucket != null)
        return false;
    return true;
  }
  public boolean containsKey(Object key) {
    // A slight modification of the previous method:
    for(ArrayList<MapEntry<K,V>> bucket : buckets) {
      if(bucket == null) continue;
      for(MapEntry<K,V> mPair : bucket)
        if(mPair.getKey().equals(key))
          return true;
    }
    return false;
  }
}

public class E37_SimpleHashMapArrays {
  public static void main(String[] args) {
    if(args.length > 0)
      Tester.defaultParams = TestParam.array(args);
    Tester.run(new SimpleHashMap4<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new SimpleHashMap5<Integer,Integer>(),
      E35_MapPerformance2.tests);
    Tester.run(new SimpleHashMap6<Integer,Integer>(),
```

```
      E35_MapPerformance2.tests);
  }
} /* Output: (Sample)
------- SimpleHashMap4 -------
 size     put     get iterate
  100     293      92     419
------- SimpleHashMap5 -------
 size     put     get iterate
  100     118      54    1389
------- SimpleHashMap6 -------
 size     put     get iterate
  100     278      92     407
*///:~
```

Earlier exercises explain the **SimpleHashMap4** and **SimpleHashMap5** classes. **SimpleHashMap5** uses its own singly-linked list, like **HashMap** from the JDK. Check the source code for **HashMap** in the JDK to see how chaining works. Sometimes your own data structure is more efficient than general purpose **List** objects.

The iteration in **SimpleHashMap5** is clearly not optimized.

Exercise 38

```
//: containers/E38_HashMapLoadFactor.java
// {RunByHand}
/****************** Exercise 38 *****************
 * Look up the HashMap class in the JDK documentation.
 * Create a HashMap, fill it with elements, and
 * determine the load factor. Test the lookup speed
 * with this map, then attempt to increase the speed
 * by making a new HashMap with a larger initial
 * capacity and copying the old map into the new one,
 * then run your lookup speed test again on the new map.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

public class E38_HashMapLoadFactor {
  public static
  void testGet(Map<String,String> filledMap, int n) {
    for(int tc = 0; tc < 1000000; tc++)
      for(int i = 0; i < Countries.DATA.length; i++) {
        String key = Countries.DATA[i][0];
```

```
        filledMap.get(key);
      }
  }
  public static void main(String args[]) {
    // Initial capacity 16:
    HashMap<String,String> map1 =
      new HashMap<String,String>();
    // Fill to less than threshold:
    map1.putAll(Countries.capitals(11));
    // Initial capacity 32:
    HashMap<String,String> map2 =
      new HashMap<String,String>(32);
    map2.putAll(map1);
    long t1 = System.currentTimeMillis();
    testGet(map1, 11);
    long t2 = System.currentTimeMillis();
    print("map1 : " + (t2 - t1));
    t1 = System.currentTimeMillis();
    testGet(map2, 11);
    t2 = System.currentTimeMillis();
    print("map2 : " + (t2 - t1));
  }
} ///:~
```

From *TIJ4*:

***Load factor**:* size/capacity. A load factor of 0 is an empty table, 0.5 is a half-full table, etc.; **map1**'s load factor before executing the test is 11/16 = 0.69, and **map2**'s is 11/32 = 0.34.

There are no **public** methods to access capacity, which you must calculate using data from the map. Either call the appropriate **HashMap** constructor or look up the default source code in the JDK, as we do here. The default constructor has initial values of:

initialCapacity = 16

loadFactor = .75

The source code also has the **threshold** — the value at which **rehash( )** is called to reorganize the table. From the JDK:

```
threshold = (int)(initialCapacity * loadFactor);
```

The table size increases when the number of elements is greater than or equal to (int)(0.75 * 16) = 12. So when we add a twelfth element the table rehashes.

Use **HashMap(int initialCapacity)**, a one-argument constructor, to make a **HashMap** with a larger initial capacity:

```
HashMap<String,String> map = new HashMap<String,String>(32);
```

Now we have a **HashMap** with double the default initial capacity, and a threshold of (int)(0.75 * 32) = 24.

The output from one run is:

```
map1 : 8469
map2 : 7671
```

Lookups make the map with a higher load factor less efficient but higher initial capacity minimizes (or even eliminates) **rehash** operations.

Exercise 39

```
//: containers/E39_SimpleHashMapRehash.java
/****************** Exercise 39 *****************
 * Invoke a private rehash() method in SimpleHashMap when
 * the load factor exceeds 0.75. During rehash, determine
 * the new number of buckets by finding the first prime
 * number greater than twice the original number of buckets.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

@SuppressWarnings("unchecked")
class SimpleHashMap7<K,V> extends SimpleHashMap<K,V> {
  private int count; // Number of elements
  private static final double loadFactor = 0.75;
  // Use a prime initial capacity; the JDK uses a number,
  // which is a power of 2:
  private final static int initialCapacity = 11;
  private int capacity = initialCapacity;
  private int threshold = (int)(capacity * loadFactor);
  { buckets = new LinkedList[capacity]; }
  @Override public V put(K key, V value) {
    V oldValue = null;
    int index = Math.abs(key.hashCode()) % capacity;
    if(buckets[index] == null)
      buckets[index] = new LinkedList<MapEntry<K,V>>();
    LinkedList<MapEntry<K,V>> bucket = buckets[index];
    MapEntry<K,V> pair = new MapEntry<K,V>(key, value);
    boolean found = false;
```

```
    ListIterator<MapEntry<K,V>> it = bucket.listIterator();
    while(it.hasNext()) {
      MapEntry<K,V> iPair = it.next();
      if(iPair.getKey().equals(key)) {
        oldValue = iPair.getValue();
        it.set(pair); // Replace old with new
        found = true;
        break;
      }
    }
    if(!found) {
      if(count >= threshold)
        rehash();
      if(buckets[index] == null)
        buckets[index] = new LinkedList<MapEntry<K,V>>();
      buckets[index].add(pair);
      ++count;
    }
    return oldValue;
  }
  private boolean isPrime(int candidate) {
    for(int j = 2; j < candidate; j++)
      if(candidate % j == 0) return false;
    return true;
  }
  private int nextPrime(int candidate) {
    while(!isPrime(candidate))
      candidate++;
    return candidate;
  }
  private void rehash() {
    // Points to a new Set of the entries, so it
    // won't be bothered by what we're about to do:
    Iterator<Map.Entry<K,V>> it = entrySet().iterator();
    // Get next prime capacity:
    capacity = nextPrime(capacity * 2);
    System.out.println(
      "Rehashing, new capacity = " + capacity);
    buckets = new LinkedList[capacity];
    threshold = (int)(capacity * loadFactor);
    count = 0;
    // Fill new buckets (crude, but it works):
    while(it.hasNext()) {
      Map.Entry<K,V> me = it.next();
      put(me.getKey(), me.getValue());
    }
  }
```

```
}

public class E39_SimpleHashMapRehash {
  public static void main(String[] args) {
    SimpleHashMap7<String,String> m =
      new SimpleHashMap7<String,String>();
    m.putAll(Countries.capitals(50));
    System.out.println(m);
  }
} /* Output:
Rehashing, new capacity = 23
Rehashing, new capacity = 47
Rehashing, new capacity = 97
{CHAD=N'djamena, BISSAU=Bissau, MOROCCO=Rabat,
LIBERIA=Monrovia, GUINEA=Conakry, LESOTHO=Maseru,
BENIN=Porto-Novo, NAMIBIA=Windhoek, GABON=Libreville,
ETHIOPIA=Addis Ababa, SOUTH AFRICA=Pretoria/Cape Town, SAO
TOME E PRINCIPE=Sao Tome, COMOROS=Moroni, ANGOLA=Luanda,
CAPE VERDE=Praia, THE GAMBIA=Banjul,
MADAGASCAR=Antananarivo, CONGO=Brazzaville,
BURUNDI=Bujumbura, MALI=Bamako, DJIBOUTI=Dijibouti,
SOMALIA=Mogadishu, EQUATORIAL GUINEA=Malabo, SUDAN=Khartoum,
SWAZILAND=Mbabane, EGYPT=Cairo, GHANA=Accra, MAURITIUS=Port
Louis, CENTRAL AFRICAN REPUBLIC=Bangui, SEYCHELLES=Victoria,
COTE D'IVOIR (IVORY COAST)=Yamoussoukro, KENYA=Nairobi,
RWANDA=Kigali, ALGERIA=Algiers, BOTSWANA=Gaberone,
NIGER=Niamey, ERITREA=Asmara, LIBYA=Tripoli, TOGO=Lome,
NIGERIA=Abuja, MOZAMBIQUE=Maputo, MAURITANIA=Nouakchott,
BURKINA FASO=Ouagadougou, UGANDA=Kampala, SENEGAL=Dakar,
CAMEROON=Yaounde, TANZANIA=Dodoma, SIERRA LEONE=Freetown,
MALAWI=Lilongwe, TUNISIA=Tunis}
*///:~
```

The solution is more simple than efficient. Some information comes from the source code for **java.util.HashMap** in the JDK.

Note the new fields at the top of the class: **count** keeps track of the number of elements; **loadFactor** is fixed at the default for **HashMap**, 0.75; **initialCapacity** is 11; **capacity** is the current number of buckets (called **SIZE** in **SimpleHashMap.java** in *TIJ4*); and **threshold** determines when to call **rehash( )**.

We also add code in **put( )** to call **rehash( )**, while the other methods do not change. At the bottom of the class there are three new methods: **isPrime( )** and **nextPrime( )** generate prime numbers; and **rehash( )** itself.

rehash( ) begins by getting an **Iterator** to the **entrySet( )** (a set of all keys and values) for this map. The code for **entrySet( )** makes a new collection to hold the objects, so the objects are not garbage-collected when we change the **bucket** container.

First we increase the buckets to a new prime number that is more than double the current number. This doubles the **capacity**, then gives it to **nextPrime( )**, which calls **isPrime( )** to see if the number is prime, and if not, increments the **candidate** and repeats its test; the process is not very efficient. For further exercise, create a lookup table for prime numbers to use in the map.

New **capacity** creates a new **bucket** array, the new **threshold** is calculated, and the **entrySet( )** iterator and **put( )** reload the elements into the hash table—again, effective but inefficient. (See the **java.util.HashMap** source code for a better approach.)

Exercise 40

```
//: containers/E40_ComparableClass.java
/****************** Exercise 40 *****************
 * Create a class containing two String objects
 * and make it Comparable so that the comparison
 * only cares about the first String. Fill an array
 * and an ArrayList with objects of your class,
 * using the RandomGenerator generator. Demonstrate that
 * sorting works properly. Now make a Comparator that
 * only cares about the second String, and demonstrate
 * that sorting works properly. Also perform a binary
 * search using your Comparator.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class TwoString implements Comparable<TwoString> {
  String s1, s2;
  public TwoString(String s1i, String s2i) {
    s1 = s1i;
    s2 = s2i;
  }
  public String toString() {
    return "[s1 = " + s1 + ", s2 = " + s2 + "]";
  }
```

```
  public int compareTo(TwoString rv) {
    String rvi = rv.s1;
    return s1.compareTo(rvi);
  }
  private static RandomGenerator.String gen =
    new RandomGenerator.String();
  public static Generator<TwoString> generator() {
    return new Generator<TwoString>() {
      public TwoString next() {
        return new TwoString(gen.next(), gen.next());
      }
    };
  }
}

class CompareSecond implements Comparator<TwoString> {
  public int compare(TwoString sc1, TwoString sc2) {
    return sc1.s2.compareTo(sc2.s2);
  }
}

public class E40_ComparableClass {
  public static void main(String[] args) {
    TwoString[] array = new TwoString[10];
    Generated.array(array, TwoString.generator());
    print("before sorting, array = " +
      Arrays.asList(array));
    Arrays.sort(array);
    print("\nafter sorting, array = " +
      Arrays.asList(array));
    Arrays.sort(array, new CompareSecond());
    print(
      "\nafter sorting with CompareSecond, array = " +
      Arrays.asList(array));
    ArrayList<TwoString> list = new ArrayList<TwoString>(
      CollectionData.list(TwoString.generator(), 10));
    TwoString zeroth = list.get(0);
    print("\nbefore sorting, list = " + list);
    Collections.sort(list);
    print("\nafter sorting, list = " + list);
    Comparator<TwoString> comp = new CompareSecond();
    Collections.sort(list, comp);
    print(
      "\nafter sorting with CompareSecond, list = "
      + list);
    int index =
      Collections.binarySearch(list, zeroth, comp);
```

```
    print("\nFormer zeroth element " +
      zeroth + " now located at " + index);
  }
} /* Output:
before sorting, array = [[s1 = YNzbrny, s2 = GcFOWZn], [s1 =
TcQrGse, s2 = GZMmJMR], [s1 = oEsuEcU, s2 = OneOEdL], [s1 =
smwHLGE, s2 = ahKcxrE], [s1 = qUCBbkI, s2 = naMesbt], [s1 =
WHkjUrU, s2 = kZPgwsq], [s1 = PzDyCyR, s2 = FJQAHxx], [s1 =
HvHqXum, s2 = cXZJoog], [s1 = oYWMNvq, s2 = euTpnXs], [s1 =
gqiaxxE, s2 = AJJmzMs]]

after sorting, array = [[s1 = HvHqXum, s2 = cXZJoog], [s1 =
PzDyCyR, s2 = FJQAHxx], [s1 = TcQrGse, s2 = GZMmJMR], [s1 =
WHkjUrU, s2 = kZPgwsq], [s1 = YNzbrny, s2 = GcFOWZn], [s1 =
gqiaxxE, s2 = AJJmzMs], [s1 = oEsuEcU, s2 = OneOEdL], [s1 =
oYWMNvq, s2 = euTpnXs], [s1 = qUCBbkI, s2 = naMesbt], [s1 =
smwHLGE, s2 = ahKcxrE]]

after sorting with CompareSecond, array = [[s1 = gqiaxxE, s2
= AJJmzMs], [s1 = PzDyCyR, s2 = FJQAHxx], [s1 = TcQrGse, s2
= GZMmJMR], [s1 = YNzbrny, s2 = GcFOWZn], [s1 = oEsuEcU, s2
= OneOEdL], [s1 = smwHLGE, s2 = ahKcxrE], [s1 = HvHqXum, s2
= cXZJoog], [s1 = oYWMNvq, s2 = euTpnXs], [s1 = WHkjUrU, s2
= kZPgwsq], [s1 = qUCBbkI, s2 = naMesbt]]

before sorting, list = [[s1 = slJrLvp, s2 = fFvHVEE], [s1 =
qjncLdZ, s2 = kpTkWng], [s1 = JjhcYVz, s2 = PCQQBnK], [s1 =
SOSdgHz, s2 = IVPJWjR], [s1 = BqtXbJV, s2 = IMLPoVg], [s1 =
qUocIBR, s2 = BnFGofp], [s1 = TQzGSrL, s2 = rcVwhKO], [s1 =
zqvPZdC, s2 = ohnYVma], [s1 = opwBTwV, s2 = gLhnfDa], [s1 =
tviIOeV, s2 = gTALcXW]]

after sorting, list = [[s1 = BqtXbJV, s2 = IMLPoVg], [s1 =
JjhcYVz, s2 = PCQQBnK], [s1 = SOSdgHz, s2 = IVPJWjR], [s1 =
TQzGSrL, s2 = rcVwhKO], [s1 = opwBTwV, s2 = gLhnfDa], [s1 =
qUocIBR, s2 = BnFGofp], [s1 = qjncLdZ, s2 = kpTkWng], [s1 =
slJrLvp, s2 = fFvHVEE], [s1 = tviIOeV, s2 = gTALcXW], [s1 =
zqvPZdC, s2 = ohnYVma]]

after sorting with CompareSecond, list = [[s1 = qUocIBR, s2
= BnFGofp], [s1 = BqtXbJV, s2 = IMLPoVg], [s1 = SOSdgHz, s2
= IVPJWjR], [s1 = JjhcYVz, s2 = PCQQBnK], [s1 = slJrLvp, s2
= fFvHVEE], [s1 = opwBTwV, s2 = gLhnfDa], [s1 = tviIOeV, s2
= gTALcXW], [s1 = qjncLdZ, s2 = kpTkWng], [s1 = zqvPZdC, s2
= ohnYVma], [s1 = TQzGSrL, s2 = rcVwhKO]]
```

```
Former zeroth element [s1 = slJrLvp, s2 = fFvHVEE] now
located at 4
*///:~
```

Exercise 41

```
//: containers/E41_HashedComparable.java
/****************** Exercise 41 *****************
 * Modify the class in Exercise 40 so
 * that it works with HashSets and as a key in
 * HashMaps.
 ************************************************/
package containers;
import java.util.*;
import net.mindview.util.*;

class TwoString2 implements Comparable<TwoString2> {
  String s1, s2;
  public TwoString2(String s1i, String s2i) {
    s1 = s1i;
    s2 = s2i;
  }
  public String toString() {
    return "[s1 = " + s1 + ", s2 = " + s2 + "]";
  }
  public int compareTo(TwoString2 rv) {
    String rvi = rv.s1;
    return s1.compareTo(rvi);
  }
  public int hashCode() {
    // Since the comparisons are based on s1,
    // use s1's hashCode:
    return s1.hashCode();
  }
  public boolean equals(Object obj) {
    return obj instanceof TwoString2 &&
      ((TwoString2)obj).s1.equals(s1);
  }
  private static RandomGenerator.String gen =
    new RandomGenerator.String();
  public static Generator<TwoString2> generator() {
    return new Generator<TwoString2>() {
      public TwoString2 next() {
        return new TwoString2(gen.next(), gen.next());
      }
    };
```

```
  }
}

public class E41_HashedComparable {
  public static void main(String[] args) {
    HashSet<TwoString2> hs = new HashSet<TwoString2>();
    HashMap<TwoString2,Integer> hm =
      new HashMap<TwoString2,Integer>();
    Generator<TwoString2> gen = TwoString2.generator();
    hs.addAll(CollectionData.list(gen, 20));
    for(int i = 0; i < 20; i++)
      hm.put(gen.next(), i);
    System.out.println("hs = " + hs);
    System.out.println("hm = " + hm);
  }
} /* Output:
hs = [[s1 = tviIOeV, s2 = gTALcXW], [s1 = SOSdgHz, s2 =
IVPJWjR], [s1 = gqiaxxE, s2 = AJJmzMs], [s1 = opwBTwV, s2 =
gLhnfDa], [s1 = qUocIBR, s2 = BnFGofp], [s1 = oYWMNvq, s2 =
euTpnXs], [s1 = qjncLdZ, s2 = kpTkWng], [s1 = slJrLvp, s2 =
fFvHVEE], [s1 = BqtXbJV, s2 = IMLPoVg], [s1 = HvHqXum, s2 =
cXZJoog], [s1 = TcQrGse, s2 = GZMmJMR], [s1 = oEsuEcU, s2 =
OneOEdL], [s1 = WHkjUrU, s2 = kZPgwsq], [s1 = zqvPZdC, s2 =
ohnYVma], [s1 = PzDyCyR, s2 = FJQAHxx], [s1 = JjhcYVz, s2 =
PCQQBnK], [s1 = YNzbrny, s2 = GcFOWZn], [s1 = qUCBbkI, s2 =
naMesbt], [s1 = TQzGSrL, s2 = rcVwhKO], [s1 = smwHLGE, s2 =
ahKcxrE]]
hm = {[s1 = JYmXfyL, s2 = LXdaCcH]=13, [s1 = vWlnklx, s2 =
ieETLdl]=12, [s1 = kkXmZpE, s2 = lUnpFsT]=0, [s1 = YOzPgpj,
s2 = HZAeudW]=19, [s1 = eRzDqLz, s2 = lEcBfEA]=16, [s1 =
DPbmXRQ, s2 = TmKrsgn]=14, [s1 = hLIpVVl, s2 = pNVvUuk]=15,
[s1 = vAusWxO, s2 = ikOSUvL]=9, [s1 = BcatxLS, s2 =
lXXnWSa]=11, [s1 = fHggOPz, s2 = kihQiHq]=5, [s1 = qYNYZGi,
s2 = IhHLLNA]=1, [s1 = WVdlNmu, s2 = TJqCrcf]=6, [s1 =
CJIkjyU, s2 = ZPzWRSy]=10, [s1 = xcwSwRL, s2 = RRDDPuI]=18,
[s1 = hfFkVrA, s2 = gepwlXE]=2, [s1 = PKNjWUl, s2 =
MJOTlae]=3, [s1 = haoyWJS, s2 = yIPMqvD]=7, [s1 = CmEtqPh,
s2 = yXPQvPa]=17, [s1 = GpArCjV, s2 = SpNQlmO]=8, [s1 =
olwrSJY, s2 = FSRWuZJ]=4}
*///:~
```

You must add **hashCode( )** and **equals( )** to use a class as a key in a hashed container.

Exercise 42

```
//: containers/E42_AlphaComparable.java
/****************** Exercise 42 *****************
 * Modify Exercise 40 so that an alphabetic sort
 * is used.
 ***********************************************/
package containers;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class TwoStringAlphabetic
implements Comparable<TwoStringAlphabetic> {
  String s1, s2;
  public TwoStringAlphabetic(String s1i, String s2i) {
    s1 = s1i;
    s2 = s2i;
  }
  public String toString() {
    return "[s1 = " + s1 + ", s2 = " + s2 + "]";
  }
  public int compareTo(TwoStringAlphabetic rv) {
    String rvi = rv.s1;
    return s1.toLowerCase().compareTo(rvi.toLowerCase());
  }
  private static RandomGenerator.String gen =
    new RandomGenerator.String();
  public static Generator<TwoStringAlphabetic> generator() {
    return new Generator<TwoStringAlphabetic>() {
      public TwoStringAlphabetic next() {
        return new TwoStringAlphabetic(
          gen.next(), gen.next());
      }
    };
  }
}

class CompareSecondAlphabetic
implements Comparator<TwoStringAlphabetic> {
  public int compare(TwoStringAlphabetic sc1,
    TwoStringAlphabetic sc2) {
    return sc1.s1.toLowerCase().compareTo(
      sc2.s1.toLowerCase());
  }
}
```

```
public class E42_AlphaComparable {
  public static void main(String[] args) {
    TwoStringAlphabetic [] array =
      new TwoStringAlphabetic [10];
    Generated.array(array, TwoStringAlphabetic.generator());
    print("before sorting, array = " +
      Arrays.asList(array));
    Arrays.sort(array);
    print("\nafter sorting, array = " +
      Arrays.asList(array));
    Arrays.sort(array, new CompareSecondAlphabetic());
    print("\nafter sorting with CompareSecondAlphabetic," +
      " array = " +
      Arrays.asList(array));
    ArrayList<TwoStringAlphabetic> list =
      new ArrayList<TwoStringAlphabetic>(
        CollectionData.list(
          TwoStringAlphabetic.generator(), 10));
    TwoStringAlphabetic zeroth = list.get(0);
    print("\nbefore sorting, list = " + list);
    Collections.sort(list);
    print("\nafter sorting, list = " + list);
    Comparator<TwoStringAlphabetic> comp =
      new CompareSecondAlphabetic();
    Collections.sort(list, comp);
    print("\nafter sorting with CompareSecondAlphabetic," +
     " list = "
      + list);
    int index =
      Collections.binarySearch(list, zeroth, comp);
    print("\nFormer zeroth element " +
      zeroth + " now located at " + index);
  }
} /* Output:
before sorting, array = [[s1 = YNzbrny, s2 = GcFOWZn], [s1 =
TcQrGse, s2 = GZMmJMR], [s1 = oEsuEcU, s2 = OneOEdL], [s1 =
smwHLGE, s2 = ahKcxrE], [s1 = qUCBbkI, s2 = naMesbt], [s1 =
WHkjUrU, s2 = kZPgwsq], [s1 = PzDyCyR, s2 = FJQAHxx], [s1 =
HvHqXum, s2 = cXZJoog], [s1 = oYWMNvq, s2 = euTpnXs], [s1 =
gqiaxxE, s2 = AJJmzMs]]

after sorting, array = [[s1 = gqiaxxE, s2 = AJJmzMs], [s1 =
HvHqXum, s2 = cXZJoog], [s1 = oEsuEcU, s2 = OneOEdL], [s1 =
oYWMNvq, s2 = euTpnXs], [s1 = PzDyCyR, s2 = FJQAHxx], [s1 =
qUCBbkI, s2 = naMesbt], [s1 = smwHLGE, s2 = ahKcxrE], [s1 =
```

```
TcQrGse, s2 = GZMmJMR], [s1 = WHkjUrU, s2 = kZPgwsq], [s1 =
YNzbrny, s2 = GcFOWZn]]

after sorting with CompareSecondAlphabetic, array = [[s1 =
gqiaxxE, s2 = AJJmzMs], [s1 = HvHqXum, s2 = cXZJoog], [s1 =
oEsuEcU, s2 = OneOEdL], [s1 = oYWMNvq, s2 = euTpnXs], [s1 =
PzDyCyR, s2 = FJQAHxx], [s1 = qUCBbkI, s2 = naMesbt], [s1 =
smwHLGE, s2 = ahKcxrE], [s1 = TcQrGse, s2 = GZMmJMR], [s1 =
WHkjUrU, s2 = kZPgwsq], [s1 = YNzbrny, s2 = GcFOWZn]]

before sorting, list = [[s1 = slJrLvp, s2 = fFvHVEE], [s1 =
qjncLdZ, s2 = kpTkWng], [s1 = JjhcYVz, s2 = PCQQBnK], [s1 =
SOSdgHz, s2 = IVPJWjR], [s1 = BqtXbJV, s2 = IMLPoVg], [s1 =
qUocIBR, s2 = BnFGofp], [s1 = TQzGSrL, s2 = rcVwhKO], [s1 =
zqvPZdC, s2 = ohnYVma], [s1 = opwBTwV, s2 = gLhnfDa], [s1 =
tviIOeV, s2 = gTALcXW]]

after sorting, list = [[s1 = BqtXbJV, s2 = IMLPoVg], [s1 =
JjhcYVz, s2 = PCQQBnK], [s1 = opwBTwV, s2 = gLhnfDa], [s1 =
qjncLdZ, s2 = kpTkWng], [s1 = qUocIBR, s2 = BnFGofp], [s1 =
slJrLvp, s2 = fFvHVEE], [s1 = SOSdgHz, s2 = IVPJWjR], [s1 =
TQzGSrL, s2 = rcVwhKO], [s1 = tviIOeV, s2 = gTALcXW], [s1 =
zqvPZdC, s2 = ohnYVma]]

after sorting with CompareSecondAlphabetic, list = [[s1 =
BqtXbJV, s2 = IMLPoVg], [s1 = JjhcYVz, s2 = PCQQBnK], [s1 =
opwBTwV, s2 = gLhnfDa], [s1 = qjncLdZ, s2 = kpTkWng], [s1 =
qUocIBR, s2 = BnFGofp], [s1 = slJrLvp, s2 = fFvHVEE], [s1 =
SOSdgHz, s2 = IVPJWjR], [s1 = TQzGSrL, s2 = rcVwhKO], [s1 =
tviIOeV, s2 = gTALcXW], [s1 = zqvPZdC, s2 = ohnYVma]]

Former zeroth element [s1 = slJrLvp, s2 = fFvHVEE] now
located at 5
*///:~
```

I/O

Exercise 1

```
//: io/E01_SearchWords.java
// {Args: java E01_SearchWords}
/****************** Exercise 1 *****************
 * Modify DirList.java (or one of its variants) so
 * that the FilenameFilter opens and reads each file
 * (using the net.mindview.util.TextFile utility) and
 * accepts the file based on whether any of the
 * trailing arguments on the command line exist in
 * that file.
 ***********************************************/
package io;
import java.io.*;
import java.util.*;
import net.mindview.util.TextFile;

public class E01_SearchWords {
  public static void main(final String[] args) {
    File path = new File(".");
    String[] list;
    if(args.length == 0)
      list = path.list();
    else
      list = path.list(new FilenameFilter() {
        private String ext = args[0].toLowerCase();
        public boolean accept(File dir, String name) {
          // Only analyze source files with the specified
          // extension (given as the first command line
          // argument)
          if(name.toLowerCase().endsWith(ext)) {
            // Only filter upon file extension?
            if(args.length == 1)
              return true;
            Set<String> words = new HashSet<String>(
              new TextFile(new File(
                dir, name).getAbsolutePath(), "\\W+"));
            for(int i = 1; i < args.length; i++)
              if(words.contains(args[i]))
                return true;
```

```
          }
          return false;
        }
      });
    Arrays.sort(list, String.CASE_INSENSITIVE_ORDER);
    for(String dirItem : list)
      System.out.println(dirItem);
  }
} /* Output:
E01_SearchWords.java
E02_SortedDirList.java
E03_DirSize.java
*///:~
```

The first command line argument is the file extension, and the rest are the words to search for. If you do not provide any information on the command line, the program will list the contents of the current folder.

The **net.mindview.util.TextFile** utility constructor expects a pathname string as the first argument. The program uses the **getAbsolutePath( )** method of the **File** class to produce one (see the JDK for more information).

Exercise 2

```
//: io/E02_SortedDirList.java
/****************** Exercise 2 ******************
 * Create a class called SortedDirList with a
 * constructor that takes a File object and builds
 * a sorted directory list from the files at that
 * File. Add to this class two overloaded list()
 * methods: the first produces the whole list, and
 * the second produces the subset of the list that
 * matches its argument (which is a regular
 * expression).
 *********************************************/
package io;
import java.io.*;
import java.util.regex.*;
import java.util.*;
import static net.mindview.util.Print.*;

class SortedDirList {
  private File path;
  public SortedDirList() { path = new File("."); }
  public SortedDirList(File path) { this.path = path; }
  public String[] list() {
```

```
    String[] list = path.list();
    Arrays.sort(list, String.CASE_INSENSITIVE_ORDER);
    return list;
  }
  public String[] list(final String fn_regex) {
    String[] list =
      path.list(new FilenameFilter() {
        private Pattern pattern = Pattern.compile(fn_regex);
        public boolean accept(File dir, String name) {
          return pattern.matcher(name).matches();
        }
      });
    Arrays.sort(list, String.CASE_INSENSITIVE_ORDER);
    return list;
  }
}

public class E02_SortedDirList {
  public static void main(String args[]) {
    // Default constructor == current directory
    SortedDirList dir = new SortedDirList();
    print(Arrays.asList(dir.list("E0[12]_.*\\.java")));
  }
} /* Output:
[E01_SearchWords.java, E02_SortedDirList.java]
*///:~
```

Much of this is a rewrite of **DirList.java** into a reusable class.

Exercise 3

```
//: io/E03_DirSize.java
// {Args: "E0[12]_.*\.java"}
/****************** Exercise 3 *****************
 * Modify DirList.java (or one of its variants) so
 * that it sums up the file sizes of the selected
 * files.
 ***********************************************/
package io;
import java.io.*;
import java.util.regex.*;
import java.util.*;
import static net.mindview.util.Print.*;

public class E03_DirSize {
  public static void main(final String[] args) {
```

```
    File path = new File(".");
    String[] list;
    if(args.length == 0)
      list = path.list();
    else
      list = path.list(new FilenameFilter() {
        private Pattern pattern = Pattern.compile(args[0]);
        public boolean accept(File dir, String name) {
          return pattern.matcher(name).matches();
        }
      });
    Arrays.sort(list, String.CASE_INSENSITIVE_ORDER);
    long total = 0;
    long fs;
    for(String dirItem : list) {
      fs = new File(path, dirItem).length();
      print(dirItem + ", " + fs + " byte(s)");
      total += fs;
    }
    print("======================");
    print(list.length + " file(s), " + total + " bytes");
  }
} /* Output:
E01_SearchWords.java, 1731 byte(s)
E02_SortedDirList.java, 1606 byte(s)
======================
2 file(s), 3337 bytes
*///:~
```

The **File.length( )** method reads the number of bytes in the file.

Exercise 4

```
//: io/E04_DirSize2.java
// {Args: ".*\.java"} All java files
/****************** Exercise 4 *****************
 * Use Directory.walk() to sum the sizes of all
 * files in a directory tree whose names match a
 * particular regular expression.
 ***********************************************/
package io;
import java.io.*;
import net.mindview.util.*;

public class E04_DirSize2 {
  public static void main(String[] args) {
```

```
    Directory.TreeInfo ti;
    if(args.length == 0)
      ti = Directory.walk("../object");
    else
      ti = Directory.walk("../object", args[0]);
    long total = 0;
    for(File file : ti)
      total += file.length();
    System.out.println(
      ti.files.size() + " file(s), " + total + " bytes");
  }
} /* Output: (Sample)
16 file(s), 11647 bytes
*///:~
```

The program sums the sizes of all Java files representing solutions of exercises from the *Everything is an Object* chapter of *TIJ4*.

Exercise 5

```
//: io/E05_ProcessFiles2.java
/****************** Exercise 5 *****************
 * Modify ProcessFiles.java so that it matches a
 * regular expression rather than a fixed
 * extension.
 ***********************************************/
package io;
import java.io.*;
import net.mindview.util.*;

class ProcessFiles2 {
  private ProcessFiles.Strategy strategy;
  private String regex;
  public ProcessFiles2(ProcessFiles.Strategy strategy,
                       String regex) {
    this.strategy = strategy;
    this.regex = regex;
  }
  public void start(String[] args) {
    try {
      if(args.length == 0)
        processDirectoryTree(new File("."));
      else
        for(String arg : args) {
          File fileArg = new File(arg);
          if(fileArg.isDirectory())
```

```
            processDirectoryTree(fileArg);
          else
            if(arg.matches(regex))
              strategy.process(fileArg.getCanonicalFile());
        }
    } catch(IOException e) {
      throw new RuntimeException(e);
    }
  }
  public void
  processDirectoryTree(File root) throws IOException {
    for(File file : Directory.walk(
        root.getAbsolutePath(), regex))
      strategy.process(file.getCanonicalFile());
  }
}

public class E05_ProcessFiles2 {
  // Demonstration of how to use it:
  public static void main(String[] args) {
    new ProcessFiles2(new ProcessFiles.Strategy() {
      public void process(File file) {
        System.out.println(file);
      }
    }, ".*\\.java").start(args);
  }
} /* (Execute to see output) *///:~
```

The program lists all the Java source-code files in given directories.

Exercise 6

```
//: io/E06_ProcessFiles3.java
// {Args: . 1/1/06}
/****************** Exercise 6 *****************
 * Use ProcessFiles to find all the Java
 * source-code files in a particular directory
 * subtree that have been modified after a
 * particular date.
 ***********************************************/
package io;
import java.io.*;
import java.text.*;
import java.util.*;
import net.mindview.util.*;
```

```
public class E06_ProcessFiles3 {
  public static void main(String[] args) {
    DateFormat df = DateFormat.getDateInstance(
      DateFormat.SHORT, Locale.US);
    if(args.length != 2) {
      System.err.println(
       "Usage: java E06_ProcessFiles3 path date");
      return;
    }
    long tmp = 0;
    try {
      df.setLenient(false);
      tmp = df.parse(args[1]).getTime();
    } catch(ParseException pe) {
      pe.printStackTrace();
      return;
    }
    final long modTime = tmp;
    new ProcessFiles(new ProcessFiles.Strategy() {
      public void process(File file) {
        if(modTime < file.lastModified())
          System.out.println(file);
      }
    }, "java").start(new String[] {args[0]});
  }
} /* (Execute to see output) *///:~
```

The program calls **DateFormat.getDateInstance( )** for the date formatter with the right style (short for "M/d/yy") for the US locale. The formatter then parses input, converting a string into a **Date** object. For strict adherence to the specified format, you must call **setLenient(false)** before invoking **parse( )**. Finally, **getTime( )** finds the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by the **Date** object. The program throws a **ParseException** for invalid input.

Exercise 7

```
//: io/E07_FileIntoList.java
/****************** Exercise 7 *****************
 * Open a text file so that you can read the file
 * one line at a time. Read each line as a String
 * and place that String object into a LinkedList.
 * Print all of the lines in the LinkedList in reverse
 * order.
 ***********************************************/
```

```
package io;
import java.io.*;
import java.util.*;

public class E07_FileIntoList {
  // Throw exceptions to console:
  public static List<String>
  read(String filename) throws IOException {
    // Reading input by lines:
    BufferedReader in = new BufferedReader(
      new FileReader(filename));
    String s;
    List<String> list = new LinkedList<String>();
    while((s = in.readLine())!= null)
      list.add(s);
    in.close();
    return list;
  }
  public static void main(String[] args)
  throws IOException {
    List<String> list = read("E07_FileIntoList.java");
    for(ListIterator<String> it =
        list.listIterator(list.size()); it.hasPrevious();)
      System.out.println(it.previous());
  }
} /* (Execute to see output) *///:~
```

The output is the above file in reverse order.

Exercise 8

```
//: io/E08_CommandLine.java
// {Args: E08_CommandLine.java}
/****************** Exercise 8 *****************
 * Modify Exercise 7 so that the name of the file
 * you read is provided as a command-line argument.
 ***********************************************/
package io;
import java.util.*;

public class E08_CommandLine {
  public static void main(String[] args)
  throws java.io.IOException {
    if(args.length != 1) {
      System.err.println(
        "Usage: java E08_CommandLine file");
```

```
      return;
    }
    List<String> list = E07_FileIntoList.read(args[0]);
    for(ListIterator<String> it =
        list.listIterator(list.size()); it.hasPrevious();)
      System.out.println(it.previous());
  }
} /* (Execute to see output) *///:~
```

Exercise 9

```
//: io/E09_UpperCase.java
// {Args: E09_UpperCase.java}
/****************** Exercise 9 *****************
 * Modify Exercise 8 to force all the lines in
 * the LinkedList to uppercase and send the results
 * to System.out.
 ***********************************************/
package io;
import java.util.*;

public class E09_UpperCase {
  public static void main(String[] args)
  throws java.io.IOException {
    if(args.length != 1) {
      System.err.println(
        "Usage: java E09_UpperCase file");
      return;
    }
    List<String> list = E07_FileIntoList.read(args[0]);
    for(ListIterator<String> it =
        list.listIterator(list.size()); it.hasPrevious();)
      System.out.println(it.previous().toUpperCase());
  }
} /* (Execute to see output) *///:~
```

Exercise 10

```
//: io/E10_FindWords.java
// {Args: E10_FindWords.java import public}
/****************** Exercise 10 *****************
 * Modify Exercise 8 to take additional command-line
 * arguments of words to find in the file. Print
 * all lines in which any of the words match.
 ***********************************************/
```

```
package io;
import java.util.*;

public class E10_FindWords {
  public static void main(String[] args)
  throws java.io.IOException {
    if(args.length < 2) {
      System.err.println(
        "Usage: java E10_FindWords file words");
      return;
    }
    Set<String> words = new HashSet<String>();
    for(int i = 1; i < args.length; i++)
      words.add(args[i]);
    List<String> list = E07_FileIntoList.read(args[0]);
    for(ListIterator<String> it =
        list.listIterator(list.size()); it.hasPrevious();) {
      String candidate = it.previous();
      for(String word : words)
        if(candidate.indexOf(word) != -1) {
          System.out.println(candidate);
          break;
        }
    }
  }
} /* (Execute to see output) *///:~
```

Set automatically eliminates duplicate candidate words from the command line. The **break** statement prevents the line from being printed more than once, if more than one candidate word is present in the line.

Exercise 11

```
//: io/E11_GreenhouseControls2.java
// {Args: 5000000}
/****************** Exercise 11 *************************
 * (Intermediate) In the
 * innerclasses/GreenhouseController.java example,
 * GreenhouseController contains a hard-coded set of events.
 * Change the program so that it reads the events and their
 * relative times from a text file. (Challenging: Use a
 * Factory Method design pattern to build the events—see
 * Thinking in Patterns (with Java) at www.MindView.net.)
 ********************************************************/
package io;
import java.util.*;
```

```
import java.io.*;
import java.lang.reflect.*;
import innerclasses.controller.*;

class GreenhouseControls2 extends GreenhouseControls {
  class Restart extends Event {
    private Event[] eventList;
    public Restart(long delayTime) { super(delayTime); }
    public void action() {
      for(Event e : eventList) {
        e.start(); // Rerun each event
        addEvent(e);
      }
      start();
      addEvent(this); // Rerun this Event
    }
    public String toString() {
      return "Restarting system";
    }
    public void setEventList(Event[] eventList) {
      this.eventList = eventList;
    }
  }
  class GHEventFactory {
    LinkedList<EventCreator> events =
      new LinkedList<EventCreator>();
    class EventCreator {
      Constructor<Event> ctor;
      long offset;
      public EventCreator(Constructor<Event> ctor,
        long offset) {
        this.ctor = ctor;
        this.offset = offset;
      }
    }
    @SuppressWarnings("unchecked")
    public GHEventFactory(String eventFile) {
      try {
        BufferedReader in =
          new BufferedReader(
            new FileReader(eventFile));
        String s;
        while((s = in.readLine())!= null) {
          int colon = s.indexOf(':');
          // Must use '$' instead of '.' to
          // describe inner classes:
          String className = s.substring(0, colon).trim();
```

```
          Class<?> outer = className.equals("Restart") ?
            GreenhouseControls2.class :
            GreenhouseControls.class;
          String type =
            outer.getSimpleName() + "$" + className;
          long offset = Long.parseLong(
            s.substring(colon + 1).trim());
          // Use Reflection to find and call
          // the right constructor:
          Class<Event> eventClass =
            (Class<Event>)Class.forName(type);
          // Inner class constructors implicitly
          // take the outer-class object as a
          // first argument:
          Constructor<Event> ctor =
            eventClass.getConstructor(
              new Class<?>[] { outer, long.class });
          events.add(new EventCreator(ctor, offset));
        }
      } catch(Exception e) {
        throw new RuntimeException(e);
      }
    }
    Iterator<Event> iterator() {
      return new Iterator<Event>() {
        Iterator<EventCreator> it = events.iterator();
        public boolean hasNext() {
          return it.hasNext();
        }
        public Event next() {
          EventCreator ec = it.next();
          Event returnVal = null;
          try {
            returnVal = ec.ctor.newInstance(
              new Object[] {
                GreenhouseControls2.this, ec.offset
              });
          } catch(Exception e) {
            throw new RuntimeException(e);
          }
          return returnVal;
        }
        public void remove() {
          throw new UnsupportedOperationException();
        }
      };
    }
```

```
  }
  GHEventFactory gheFactory;
  public GreenhouseControls2(String initFile) {
    gheFactory = new GHEventFactory(initFile);
    // Now we need some logic to setup the system.
    // The restart event requires a special attention.
    LinkedList<Event> restartableEvents =
      new LinkedList<Event>();
    Iterator<Event> it = gheFactory.iterator();
    while(it.hasNext()) {
      Event e = it.next();
      if(e instanceof Bell || e instanceof Restart)
        continue;
      restartableEvents.add(e);
    }
    it = gheFactory.iterator();
    while(it.hasNext()) {
      Event e = it.next();
      addEvent(e);
      if(e instanceof Restart)
        ((Restart)e).setEventList(
          restartableEvents.toArray(new Event[0]));
    }
  }
}

public class E11_GreenhouseControls2 {
  public static void main(String[] args) {
    GreenhouseControls2 gc =
      new GreenhouseControls2("GreenhouseConfig.dat");
    try {
      if(args.length == 1)
        gc.addEvent(new GreenhouseControls.Terminate(
          Long.parseLong(args[0])));
    } catch(NumberFormatException e) {
      System.err.println("Terminate event is not added!");
      e.printStackTrace();
    }
    gc.run();
  }
} /* Output: (Sample)
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Bing!
```

```
Thermostat on day setting
Restarting system
Bing!
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Bing!
Thermostat on day setting
Restarting system
Bing!
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Terminating
*///:~
```

The configuration file consists of the class name, a colon, and the time offset. The '**//:!**' on the first line causes the extraction program to remove the first and last lines, so they do not appear in the file included in the code distribution for *TIJ4*:

```
//:! io/GreenhouseConfig.dat
ThermostatNight: 100000
LightOn: 200000
LightOff: 400000
WaterOn: 600000
WaterOff: 800000
Bell: 1000000
ThermostatDay: 1400000
Restart: 2000000
///:~
```

The **GHEventFactory** class contains a **LinkedList** to hold an **EventCreator** object for each line in the configuration file. Each **EventCreator** holds a **Constructor** and the **offset** value for each event. The **Constructor** is a **java.lang.reflection** object that represents a constructor for a class, which you can use to dynamically create new objects. The trick is in producing the **Constructor** object, which we see in the **GHEventFactory** constructor.

The **GHEventFactory** constructor takes a text configuration file argument. It opens that file, finds the colon in each line, and divides the line on the colon into the class name and offset value. Since these are inner classes, you must prepend **GreenhouseControls** to qualify the name, but never separate it with a '**.**'

because **Class.forName( )** actually looks for the filename to load, so you must separate the names with the '**$**' (which separates class names in the compiler-generated internal name for an inner class).

Class.forName( ) uses the string we provide to produce a reference to the **Class** object. The method **getConstructor( )** produces the **Constructor** object, but needs an argument list to match with the appropriate constructor. Give it an array of **Class** objects matching your argument list; then you can add the **EventCreator**.

The program could throw several exceptions here, which we convert to **RuntimeException**s. If anything fails, it reports everything to the console.

We use an iterator to move through the **EventCreator** objects and produce new **Event** objects, which we define as anonymous inner classes built on top of the **events LinkedList**. Every time you call **next( )**, the iterator fetches the next **EventCreator** in the list and uses its **Constructor** and **offset** to build a new **Event** object. The **newInstance** method is called on the **Constructor** object; it requires the correct number and type of arguments passed to it as a dynamically created array of **Object**s. The offset is added to the current time each time you call **next( )**.

We initialize the **Restart** class with a new **setEventList( )** method. During startup it hands eligible events to the matching **Restart** object (e.g., the **Bell** restarts itself automatically).

Exercise 12

```
//: io/E12_LineNumber.java
// {Args: E12_LineNumber.java E12_LineNumber.out}
/****************** Exercise 12 *****************
 * Modify Exercise 8 to also open a text file so
 * you can write text into it. Write the lines in the
 * LinkedList, along with line numbers (do not
 * attempt to use the "LineNumber" classes), out
 * to the file.
 ***********************************************/
package io;
import java.io.*;
import java.util.*;

public class E12_LineNumber {
  public static void main(String[] args)
  throws java.io.IOException {
    if(args.length != 2) {
```

```
      System.err.println(
        "Usage: java E12_LineNumber infile outfile");
      return;
    }
    List<String> list = E07_FileIntoList.read(args[0]);
    PrintWriter out =
      new PrintWriter(
        new BufferedWriter(
          new FileWriter(args[1])));
    // We need to count backwards...
    int line = list.size();
    for(ListIterator<String> it =
        list.listIterator(list.size()); it.hasPrevious();)
      out.println(line-- + ": " + it.previous());
    out.close();
  }
} ///:~
```

Exercise 13

```
//: io/E13_CountLines.java
/****************** Exercise 13 *****************
 * Modify BasicFileOutput.java so that it uses
 * LineNumberReader to keep track of the line
 * count. Note that it's much easier to just keep
 * track programmatically.
 ************************************************/
package io;
import java.io.*;

public class E13_CountLines {
  static String file = "E13_CountLines.out";
  public static void main(String[] args)
  throws IOException {
    // LineNumberReader is inherited from
    // BufferedReader so we don't need to
    // explicitly buffer it:
    LineNumberReader in =
      new LineNumberReader(
        new FileReader("E13_CountLines.java"));
    PrintWriter out = new PrintWriter(
      new BufferedWriter(new FileWriter(file)));
    String s;
    while((s = in.readLine()) != null )
      out.println(in.getLineNumber() + ": " + s);
    out.close();
```

```
    // Show the stored file:
    System.out.println(E07_FileIntoList.read(file));
  }
} /* (Execute to see output) *///:~
```

LineNumberReader counts from one, while most counting begins at zero.

Exercise 14

```
//: io/E14_BufferPerformance.java
/****************** Exercise 14 *****************
 * Starting with BasicFileOutput.java, write a
 * program that compares the performance of writing
 * to a file when using buffered and unbuffered I/O.
 ***********************************************/
package io;
import java.io.*;
import java.util.*;

public class E14_BufferPerformance {
  static String file = "E14_BufferPerformance.out";
  public static void main(String[] args)
  throws IOException {
    List<String> list = E07_FileIntoList.read(
      "E14_BufferPerformance.java");
    PrintWriter out = new PrintWriter(
      new BufferedWriter(new FileWriter(file)));
    int lineCount = 1;
    long t1 = System.currentTimeMillis();
    for(String s : list) {
      for(int i = 0; i < 10000; i++)
        out.println(lineCount + ": " + s);
      lineCount++;
    }
    long t2 = System.currentTimeMillis();
    System.out.println("buffered: " + (t2 - t1));
    out.close();
    out = new PrintWriter(new FileWriter(file));
    lineCount = 1;
    t1 = System.currentTimeMillis();
    for(String s : list) {
      for(int i = 0; i < 10000; i++)
        out.println(lineCount + ": " + s);
      lineCount++;
    }
    t2 = System.currentTimeMillis();
```

```
    System.out.println("unbuffered: " + (t2 - t1));
    out.close();
  }
} /* Output: (Sample)
buffered: 3385
unbuffered: 4196
*///:~
```

Exercise 15

```
//: io/E15_StoringAndRecoveringAllData.java
/****************** Exercise 15 *****************
 * Look up DataOutputStream and DataInputStream in
 * the JDK documentation. Starting with
 * StoringAndRecoveringData.java, create a program
 * that stores and then retrieves all the different
 * possible types provided by the DataOutputStream
 * and DataInputStream classes. Verify that the
 * values are stored and retrieved accurately.
 ***********************************************/
package io;
import java.io.*;
import static net.mindview.util.Print.*;

public class E15_StoringAndRecoveringAllData {
  public static void main(String[] args)
  throws IOException {
    DataOutputStream out = new DataOutputStream(
      new BufferedOutputStream(
        new FileOutputStream("Data.txt")));
    out.writeBoolean(true);
    out.writeByte(100);
    out.writeByte(255);
    out.writeChar('A');
    out.writeFloat(1.41413f);
    out.writeLong(1000000000L);
    out.writeInt(100000);
    out.writeShort(30000);
    out.writeShort(65535);
    out.writeDouble(3.14159);
    out.writeUTF("That was pi");
    out.close();
    DataInputStream in = new DataInputStream(
      new BufferedInputStream(
        new FileInputStream("Data.txt")));
    print(in.readBoolean());
```

```
    print(in.readByte());
    print(in.readUnsignedByte());
    print(in.readChar());
    print(in.readFloat());
    print(in.readLong());
    print(in.readInt());
    print(in.readShort());
    print(in.readUnsignedShort());
    print(in.readDouble());
    // Only readUTF() will recover the
    // Java-UTF String properly:
    print(in.readUTF());
  }
} /* Output:
true
100
255
A
1.41413
1000000000
100000
30000
65535
3.14159
That was pi
*///:~
```

This program demonstrates the appropriate methods for writing/reading the basic data types in Java. Observe, however, that there are no equivalent methods for **readUnsignedByte( )** and **readUnsignedShort( )** in **DataOutputStream**, another reason to be careful when using this technique for storing and retrieving complex data structures. Sometimes you need to know both the type and the range of stored data, though we don't advise that you hard code all this inside a program.

Exercise 16

```
//: io/E16_UsingAllRandomAccessFile.java
/****************** Exercise 16 *****************
 * Look up RandomAccessFile in the JDK
 * documentation. Starting with
 * UsingRandomAccessFile.java, create a program
 * that stores and then retrieves all the different
 * possible types provided by the RandomAccessFile
 * class. Verify that the values are stored and
```

```
 * retrieved accurately.
 *********************************************/
package io;
import java.io.*;
import static net.mindview.util.Print.*;

public class E16_UsingAllRandomAccessFile {
  static String file = "rtest.dat";
  static void display() throws IOException {
    RandomAccessFile rf = new RandomAccessFile(file, "r");
    print(rf.readBoolean());
    print(rf.readByte());
    print(rf.readUnsignedByte());
    print(rf.readChar());
    print(rf.readFloat());
    print(rf.readLong());
    print(rf.readInt());
    print(rf.readShort());
    print(rf.readUnsignedShort());
    print(rf.readDouble());
    print(rf.readUTF());
    rf.close();
  }
  public static void main(String[] args)
  throws IOException {
    RandomAccessFile rf = new RandomAccessFile(file, "rw");
    rf.writeBoolean(true);
    rf.writeByte(100);
    rf.writeByte(255);
    rf.writeChar('A');
    rf.writeFloat(1.41413f);
    rf.writeLong(1000000000L);
    rf.writeInt(100000);
    rf.writeShort(30000);
    rf.writeShort(65535);
    rf.writeDouble(3.14159);
    rf.writeUTF("The end of the file");
    rf.close();
    display();
    rf = new RandomAccessFile(file, "rw");
    rf.seek(3); // 1 boolean + 2 bytes
    rf.writeChar('B');
    rf.close();
    display();
  }
} /* Output:
true
```

```
100
255
A
1.41413
1000000000
100000
30000
65535
3.14159
The end of the file
true
100
255
B
1.41413
1000000000
100000
30000
65535
3.14159
The end of the file
*///:~
```

Exercise 17

```
//: io/E17_CharactersInfo.java
/****************** Exercise 17 *****************
 * Using TextFile and a Map<Character,Integer>,
 * create a program that counts the occurrence of
 * all the different characters in a file. (So if
 * there are 12 occurrences of the letter 'a' in
 * the file, the Integer associated with the Character
 * containing 'a' in the Map contains '12').
 ***********************************************/
package io;
import java.util.*;
import net.mindview.util.*;

public class E17_CharactersInfo {
  public static void main(String[] args) {
    Map<Character,Integer> charsStat =
      new HashMap<Character,Integer>();
    for(String word :
      new TextFile("E17_CharactersInfo.java", "\\W+"))
      for(int i = 0; i < word.length(); i++) {
        Character ch = word.charAt(i);
```

```
        Integer freq = charsStat.get(ch);
        charsStat.put(ch, freq == null ? 1 : freq + 1);
      }
    List<Character> keys = Arrays.asList(
      charsStat.keySet().toArray(new Character[0]));
    Collections.sort(keys);
    for(Character key : keys)
      System.out.println(key + " => " + charsStat.get(key));
  }
} /* Output: (Sample)
0 => 2
1 => 8
2 => 2
7 => 4
A => 3
C => 12
E => 4
...
u => 10
v => 5
w => 8
x => 3
y => 10
*///:~
```

Exercise 18

```
//: io/E18_TextFile2.java
/****************** Exercise 18 *****************
 * Modify TextFile.java so that it passes
 * IOExceptions out to the caller.
 ***********************************************/
package io;
import java.io.*;
import java.util.*;

class TextFile2 extends ArrayList<String> {
  // Read a file as a single string:
  public static String read(String fileName)
  throws IOException {
    StringBuilder sb = new StringBuilder();
    BufferedReader in= new BufferedReader(new FileReader(
      new File(fileName).getAbsoluteFile()));
    try {
      String s;
      while((s = in.readLine()) != null) {
```

```
        sb.append(s);
        sb.append("\n");
      }
    } finally {
      in.close();
    }
    return sb.toString();
  }
  // Write a single file in one method call:
  public static void write(String fileName, String text)
  throws IOException {
    PrintWriter out = new PrintWriter(
      new BufferedWriter(new FileWriter(
        new File(fileName).getAbsoluteFile())));
    try {
      out.print(text);
    } finally {
      out.close();
    }
  }
  // Read a file, split by any regular expression:
  public TextFile2(String fileName, String splitter)
  throws IOException {
    super(Arrays.asList(read(fileName).split(splitter)));
    // Regular expression split() often leaves an empty
    // String at the first position:
    if(get(0).equals("")) remove(0);
  }
  // Normally read by lines:
  public TextFile2(String fileName) throws IOException {
    this(fileName, "\n");
  }
  public void write(String fileName) throws IOException {
    PrintWriter out = new PrintWriter(
      new BufferedWriter(new FileWriter(
        new File(fileName).getAbsoluteFile())));
    try {
      for(String item : this)
        out.println(item);
    } finally {
      out.close();
    }
  }
}

public class E18_TextFile2 {
  public static void main(String[] args)
```

```
  throws IOException {
    String file =
      TextFile2.read("E18_TextFile2.java");
    TextFile2.write("test.txt", file);
    TextFile2 text = new TextFile2("test.txt");
    text.write("test2.txt");
    // Break into unique sorted list of words:
    TreeSet<String> words = new TreeSet<String>(
      new TextFile2("E18_TextFile2.java", "\\W+"));
    // Display the capitalized words:
    System.out.println(words.headSet("a"));
  }
} /* Output: (95% match)
[0, 18, ArrayList, Arrays, Break, BufferedReader,
BufferedWriter, Display, E18_TextFile2, Exercise, File,
FileReader, FileWriter, IOException, IOExceptions, Modify,
Normally, PrintWriter, Read, Regular, String, StringBuilder,
System, TextFile, TextFile2, TreeSet, W, Write]
*///:~
```

Exercise 19

```
//: io/E19_BytesInfo.java
/****************** Exercise 19 *****************
 * Using BinaryFile and a Map<Byte,Integer>, create
 * a program that counts the occurrence of all the
 * different bytes in a file.
 ***********************************************/
package io;
import java.io.*;
import java.util.*;
import net.mindview.util.*;

public class E19_BytesInfo {
  public static void main(String[] args)
  throws IOException {
    Map<Byte,Integer> bytesStat =
      new HashMap<Byte,Integer>();
    for(Byte bt : BinaryFile.read("E19_BytesInfo.class")) {
      Integer freq = bytesStat.get(bt);
      bytesStat.put(bt, freq == null ? 1 : freq + 1);
    }
    List<Byte> keys =
      new ArrayList<Byte>(bytesStat.keySet());
    Collections.sort(keys);
    for(Byte key : keys)
```

```
      System.out.println(key + " => " + bytesStat.get(key));
  }
} /* Output: (Sample)
-124 => 2
-103 => 1
-94 => 1
-89 => 3
-84 => 1
...
116 => 73
117 => 22
118 => 42
119 => 2
120 => 5
121 => 10
*///:~
```

Since there is no unsigned **byte** in Java, the output contains negative values.

Exercise 20

```
//: io/E20_ClassSignatureChecker.java
/****************** Exercise 20 *****************
 * Using Directory.walk() and BinaryFile, verify
 * that all .class files in a directory tree begin
 * with the hex characters 'CAFEBABE'.
 ************************************************/
package io;
import java.io.*;
import net.mindview.util.*;

public class E20_ClassSignatureChecker {
  final static byte[] signature =
    {(byte)202, (byte)254, (byte)186, (byte)190};
  public static void main(String[] args)
  throws IOException {
    String dir = ".";
    if(args.length == 1)
      dir = args[0];
    for(File file : Directory.walk(dir, ".*\\.class")) {
      byte[] bt = BinaryFile.read(file);
      for(int i = 0; i< signature.length; i++)
        if(bt[i] != signature[i]) {
          System.err.println(file + " is corrupt!");
          break;
        }
```

```
    }
  }
} ///:~
```

Exercise 21

```
//: io/E21_UpperCaseEcho.java
// {RunByHand}
/****************** Exercise 21 *****************
 * Write a program that takes standard input and
 * capitalizes all characters, then puts the results
 * on standard output. Redirect the contents of a
 * file into this program (the process of
 * redirection will vary depending on your operating
 * system).
 ************************************************/
package io;
import java.io.*;

public class E21_UpperCaseEcho {
  public static void main(String[] args)
  throws IOException {
    BufferedReader stdin = new BufferedReader(
      new InputStreamReader(System.in));
    String s;
    while((s = stdin.readLine()) != null && s.length() != 0)
      System.out.println(s.toUpperCase());
    // An empty line or Ctrl-Z terminates the program
  }
} ///:~
```

Exercise 22

```
//: io/E22_OSExecuteDemo.java
/****************** Exercise 22 *****************
 * Modify OSExecute.java so that, instead of
 * printing the standard output stream, it returns
 * the results of executing the program as a List
 * of Strings. Demonstrate the use of this new
 * version of the utility.
 ************************************************/
package io;
import java.io.*;
import java.util.*;
import net.mindview.util.*;
```

```
class OSExecute2 {
  public static List<String> command(String command) {
    boolean err = false;
    List<String> output = new LinkedList<String>();
    try {
      Process process =
        new ProcessBuilder(command.split(" ")).start();
      BufferedReader results = new BufferedReader(
        new InputStreamReader(process.getInputStream()));
      String s;
      while((s = results.readLine()) != null)
        output.add(s);
      BufferedReader errors = new BufferedReader(
        new InputStreamReader(process.getErrorStream()));
      // Report errors and return nonzero value
      // to calling process if there are problems:
      while((s = errors.readLine()) != null) {
        System.err.println(s);
        err = true;
      }
    } catch(IOException e) {
      if(!command.startsWith("CMD /C"))
        return command("CMD /C " + command);
      throw new RuntimeException(e);
    } catch(Exception e) {
      throw new RuntimeException(e);
    }
    if(err)
      throw new OSExecuteException("Errors executing " +
        command);
    return output;
  }
}

public class E22_OSExecuteDemo {
  public static void main(String[] args) {
    List<String> result =
      OSExecute2.command("javap E22_OSExecuteDemo");
    for(String s : result)
      System.out.println(s);
  }
} /* Output:
Compiled from "E22_OSExecuteDemo.java"
public class io.E22_OSExecuteDemo extends java.lang.Object{
    public io.E22_OSExecuteDemo();
    public static void main(java.lang.String[]);
```

```
}
*///:~
```

Exercise 23

```
//: io/E23_PrintCharBuffer.java
// {RunByHand}
/********************* Exercise 23 ***********************
 * Create and test a utility method to print the contents
 * of a CharBuffer up to the point where the characters
 * are no longer printable.
 *********************************************************/
package io;
import java.nio.*;
import java.util.*;

public class E23_PrintCharBuffer {
  static BitSet isPrintable = new BitSet(127);
  static String encoding =
    System.getProperty("file.encoding");
  static {
    // Assume an encoding that obeys ASCII eg.ISO-8859-1.
    // Characters 32 to 127 represent printable characters.
    for(int i = 32; i <= 127; i++)
      isPrintable.set(i);
  }
  // Set the position to the last printable character
  public static void setPrintableLimit(CharBuffer cb) {
    cb.rewind();
    while(isPrintable.get(cb.get()));
    cb.limit(cb.position() - 1);
    cb.rewind();
  }
  public static void main(String[] args) {
    System.out.println("Default Encoding is: " + encoding);
    CharBuffer buffer =
      ByteBuffer.allocate(16).asCharBuffer();
    buffer.put("ABCDE" + (char) 0x01 + "FG");
    buffer.rewind();
    System.out.println(buffer); // Print everything
    setPrintableLimit(buffer);
    System.out.println(buffer); // Print printable
  }
} ///:~
```

Exercise 24

```
//: io/E24_DoubleBufferDemo.java
/********************* Exercise 24 ***********************
 * Modify IntBufferDemo.java to use doubles.
 *********************************************************/
package io;
import java.nio.*;

public class E24_DoubleBufferDemo {
  private static final int BSIZE = 1024;
  public static void main(String[] args) {
    ByteBuffer bb = ByteBuffer.allocate(BSIZE);
    DoubleBuffer db = bb.asDoubleBuffer();
    // Store an array of double:
    db.put(
      new double[]{ 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 });
    // Absolute location read and write:
    System.out.println(db.get(3));
    db.put(3, 0.3);
    db.flip();
    while(db.hasRemaining()) {
      double d = db.get();
      System.out.println(d);
    }
  }
} /* Output:
1.3
1.0
1.1
1.2
0.3
1.4
1.5
1.6
*///:~
```

Exercise 25

```
//: io/E25_AllocateDirect.java
// {RunByHand}
/******************** Exercise 25 ************************
 * Experiment with changing the ByteBuffer.allocate()
 * statements in the examples in this chapter to
 * ByteBuffer.allocateDirect(). Demonstrate performance
```

```
 * differences, but also notice whether the startup time
 * of the programs noticeably changes.
 ******************************************************/
package io;
import java.nio.*;
import java.nio.channels.*;
import java.io.*;
import java.nio.charset.*;

abstract class CompareAllocations {
  private String name;
  protected ByteBuffer buffer;
  private int size;
  public CompareAllocations(String name, int size) {
    this.name = name;
    this.size = size;
  }
  public void runComparison() {
    System.out.println("Program Name: <" + name + ">");
    try {
      long startTime = System.nanoTime();
      directAllocate();
      long endTime = System.nanoTime();
      System.out.println(
        "Direct Allocation Cost for buffer of size: <"
        + size + "> is <" + (endTime - startTime) + ">");
      startTime = System.nanoTime();
      execute();
      endTime = System.nanoTime();
      System.out.println(
        "Execution cost using direct buffer: <"
          + (endTime - startTime) + ">");
      startTime = System.nanoTime();
      indirectAllocate();
      endTime = System.nanoTime();
      System.out.println(
        "Indirect Allocation Cost for buffer of size: <"
          + size + "> is <" + (endTime - startTime) + ">");
      startTime = System.nanoTime();
      execute();
      endTime = System.nanoTime();
      System.out.println(
        "Execution cost using indirect buffer: <"
          + (endTime - startTime) + ">");
    } catch (IOException e) {
      throw new RuntimeException(e);
    }
```

```
  }
  public void directAllocate() {
    buffer = ByteBuffer.allocateDirect(size);
  }
  public abstract void execute() throws IOException;
  public void indirectAllocate() {
    buffer = ByteBuffer.allocate(size);
  }
}

public class E25_AllocateDirect {
  public static void main(String[] args) {
    CompareAllocations[] comparisons = {
      new CompareAllocations("GetChannel", 8192) {
        public void execute() throws IOException {
          FileChannel fc =
            new FileInputStream("E25_AllocateDirect.java")
            .getChannel();
          fc.read(buffer);
          buffer.flip();
          while(buffer.hasRemaining())
            buffer.get();
        }
      },
      new CompareAllocations("ChannelCopy", 16384) {
        public void execute() throws IOException {
          FileChannel in =
            new FileInputStream("E25_AllocateDirect.java")
            .getChannel(),
            out = new FileOutputStream("temp.txt")
            .getChannel();
          while(in.read(buffer) != -1) {
            buffer.flip(); // Prepare for writing
            out.write(buffer);
            buffer.clear();  // Prepare for reading
          }
        }
      },
      new CompareAllocations("BufferToText", 8192) {
        public void execute() throws IOException {
          FileChannel fc =
              new FileOutputStream("data2.txt")
              .getChannel();
          fc.write(ByteBuffer.wrap("Some text".getBytes()));
          fc.close();
          fc = new FileInputStream("data2.txt")
               .getChannel();
```

```
        fc.read(buffer);
        buffer.flip();
        buffer.asCharBuffer().toString();
        // Decode using this system's default Charset:
        buffer.rewind();
        Charset.forName(
          System.getProperty("file.encoding"))
          .decode(buffer);
        fc = new FileOutputStream("data2.txt")
            .getChannel();
        fc.write(ByteBuffer.wrap(
          "Some text".getBytes("UTF-16BE")));
        fc.close();
        // Now try reading again:
        fc = new FileInputStream("data2.txt")
            .getChannel();
        buffer.clear();
        fc.read(buffer);
        buffer.flip();
        buffer.asCharBuffer().toString();
        // Use a CharBuffer to write through:
        fc = new FileOutputStream("data2.txt")
            .getChannel();
        buffer.clear();
        buffer.asCharBuffer().put("Some text");
        fc.write(buffer);
        fc.close();
        // Read and display:
        fc = new FileInputStream("data2.txt")
            .getChannel();
        buffer.clear();
        fc.read(buffer);
        buffer.flip();
        buffer.asCharBuffer().toString();
      }
    },
    new CompareAllocations("GetData", 1024) {
      public void execute() throws IOException {
        // Store and read a char array:
        buffer.asCharBuffer().put("Howdy!");
        // Store and read a short:
        buffer.asShortBuffer().put((short)471142);
        buffer.getShort();
        buffer.rewind();
        // Store and read an int:
        buffer.asIntBuffer().put(99471142);
        buffer.getInt();
```

```
        buffer.rewind();
        // Store and read a long:
        buffer.asLongBuffer().put(99471142);
        buffer.getLong();
        buffer.rewind();
        // Store and read a float:
        buffer.asFloatBuffer().put(99471142);
        buffer.getFloat();
        buffer.rewind();
        // Store and read a double:
        buffer.asDoubleBuffer().put(99471142);
        buffer.getDouble();
        buffer.rewind();
      }
    },
    new CompareAllocations("IntBufferDemo", 1024) {
      public void execute() throws IOException {
        IntBuffer ib = buffer.asIntBuffer();
        // Store an array of int:
        ib.put(
          new int[] { 11, 42, 47, 99, 143, 811, 1016 });
        // Absolute location read and write:
        ib.get(3);
        ib.put(3, 1811);
        ib.flip();
        while(ib.hasRemaining()) {
          int i = ib.get();
        }
      }
    },
    new CompareAllocations("UsingBuffers", 32) {
      public void execute() throws IOException {
        char[] data = "UsingBuffers".toCharArray();
        CharBuffer cb = buffer.asCharBuffer();
        cb.put(data);
        cb.rewind();
        symmetricScramble(cb);
        cb.rewind();
        symmetricScramble(cb);
        cb.rewind();
      }
     private void symmetricScramble(CharBuffer buffer) {
        while(buffer.hasRemaining()) {
          buffer.mark();
          char c1 = buffer.get();
          char c2 = buffer.get();
          buffer.reset();
```

```
            buffer.put(c2).put(c1);
          }
        }
      }
    };
    for(int i = 0; i < comparisons.length; i++)
      comparisons[i].runComparison();
  }
} ///:~
```

Direct buffers increase program performance in bulk operations, but typically have somewhat higher allocation and deallocation costs than non-direct buffers.

Exercise 26

```
//: io/E26_JGrepMM.java
// {Args: E26_JGrepMM.java \b[Ssct]\w+}
/********************** Exercise 26 *********************
 * Modify strings/JGrep.java to use Java nio memory-mapped
 * files.
 *******************************************************/
package io;
import java.io.*;
import java.util.regex.*;
import java.nio.channels.*;
import java.nio.*;
import java.nio.charset.*;
import static net.mindview.util.Print.*;

public class E26_JGrepMM {
  public static void main(String[] args) throws Exception {
    if(args.length < 2) {
      print("Usage: java E26_JGrepMM file regex");
      System.exit(0);
    }
    Pattern p = Pattern.compile(args[1]);
    FileChannel fc =
      new FileInputStream(args[0]).getChannel();
    ByteBuffer buffer =
      fc.map(FileChannel.MapMode.READ_ONLY, 0, fc.size());
    CharBuffer cb = Charset.forName(
      System.getProperty("file.encoding")).decode(buffer);
    String[] fileAsArray = cb.toString().split("\n");
    int index = 0;
    Matcher m = p.matcher("");
    for(String line : fileAsArray) {
```

```
      m.reset(line);
      while(m.find())
        print(index++ + ": " + m.group() +
          ": " + m.start());
    }
    fc.close();
  }
} /* Output: (Sample)
0: Ssct: 30
1: strings: 10
2: to: 29
3: channels: 16
4: charset: 16
5: static: 7
6: class: 7
7: static: 9
8: String: 26
9: throws: 41
10: System: 6
11: compile: 24
12: size: 50
13: cb: 15
14: System: 6
15: String: 4
16: cb: 27
17: toString: 30
18: split: 41
19: String: 8
20: start: 19
21: close: 7
*///:~
```

This program demonstrates how to set up a Java **nio** memory-mapped file.

Exercise 27

```
//: io/E27_ObjectSerialization.java
/********************* Exercise 27 *********************
 * Create a Serializable class containing a reference to an
 * object of a second Serializable class. Create an instance
 * of your class, serialize it to disk, then restore it and
 * verify that the process worked correctly.
 *******************************************************/
package io;
import java.io.*;
import static net.mindview.util.Print.*;
```

```
class Thing1 implements Serializable {
  private Thing2 next;
  public Thing1(int id) { next = new Thing2(id); }
  public String toString() {
    StringBuilder result =
      new StringBuilder("Thing1(Thing2(");
    result.append(next);
    result.append("))");
    return result.toString();
  }
}

class Thing2 implements Serializable {
  private int id;
  public Thing2(int id) { this.id = id; }
  public String toString() { return Integer.toString(id); }
}

public class E27_ObjectSerialization {
  public static void main(String[] args)
  throws ClassNotFoundException, IOException {
    Thing1 t1 = new Thing1(1);
    print("t1 = " + t1);
    ObjectOutputStream out = new ObjectOutputStream(
      new FileOutputStream("thing1.out"));
    out.writeObject("Thing1 storage\n");
    out.writeObject(t1);
    out.close(); // Also flushes output
    ObjectInputStream in = new ObjectInputStream(
      new FileInputStream("thing1.out"));
    String s = (String)in.readObject();
    Thing1 t2 = (Thing1)in.readObject();
    print(s + "t2 = " + t2);
    ByteArrayOutputStream bout =
      new ByteArrayOutputStream();
    ObjectOutputStream out2 = new ObjectOutputStream(bout);
    out2.writeObject("Thing1 storage\n");
    out2.writeObject(t1);
    out2.flush();
    ObjectInputStream in2 = new ObjectInputStream(
      new ByteArrayInputStream(bout.toByteArray()));
    s = (String)in2.readObject();
    Thing1 t3 = (Thing1)in2.readObject();
    print(s + "t3 = " + t3);
  }
} /* Output:
```

```
t1 = Thing1(Thing2(1))
Thing1 storage
t2 = Thing1(Thing2(1))
Thing1 storage
t3 = Thing1(Thing2(1))
*///:~
```

Exercise 28

```
//: io/E28_BlipCheck.java
// {RunByHand}
/****************** Exercise 28 *****************
 * In Blips.java, copy the file and rename it to
 * BlipCheck.java and rename the class Blip2 to
 * BlipCheck (making it public and removing the
 * public scope from the class Blips in the
 * process). Remove the //! marks in the file and
 * execute the program including the offending
 * lines. Next, comment out the default
 * constructor for BlipCheck. Run it and explain
 * why it works. Note that after compiling, you
 * must execute the program with "java Blips"
 * because the main() method is still in class
 * Blips.
 *********************************************/
package io;
import java.io.*;
import static net.mindview.util.Print.*;

class Blip1 implements Externalizable {
  public Blip1() {
    print("Blip1 Constructor");
  }
  public void writeExternal(ObjectOutput out)
      throws IOException {
    print("Blip1.writeExternal");
  }
  public void readExternal(ObjectInput in)
     throws IOException, ClassNotFoundException {
    print("Blip1.readExternal");
  }
}

public class E28_BlipCheck implements Externalizable {
//  E28_BlipCheck() {
//    print("BlipCheck Constructor");
```

```
//  }
  public void writeExternal(ObjectOutput out)
      throws IOException {
    print("BlipCheck.writeExternal");
  }
  public void readExternal(ObjectInput in)
     throws IOException, ClassNotFoundException {
    print("BlipCheck.readExternal");
  }
  public static void main(String[] args) throws Exception {
    // To make it run with Ant.
    Blips.main(args);
  }
}

class Blips {
  // Throw exceptions to console:
  public static void main(String[] args)
  throws IOException, ClassNotFoundException {
    print("Constructing objects:");
    Blip1 b1 = new Blip1();
    E28_BlipCheck b2 = new E28_BlipCheck();
    ObjectOutputStream o =
      new ObjectOutputStream(
        new FileOutputStream("Blips.out"));
    print("Saving objects:");
    o.writeObject(b1);
    o.writeObject(b2);
    o.close();
    // Now get them back:
    ObjectInputStream in =
      new ObjectInputStream(
        new FileInputStream("Blips.out"));
    print("Recovering b1:");
    b1 = (Blip1)in.readObject();
    // OOPS! Throws an exception:
    print("Recovering b2:");
    b2 = (E28_BlipCheck)in.readObject();
  }
} ///:~
```

When we remove the **//!'s**, the output is:

```
Constructing objects:
Blip1 Constructor
BlipCheck Constructor
Saving objects:
```

```
Blip1.writeExternal
BlipCheck.writeExternal
Recovering b1:
Blip1 Constructor
Blip1.readExternal
Recovering b2:
Exception in thread "main" java.io.InvalidClassException:
E28_BlipCheck; no valid constructor
        at java.io.ObjectStreamClass.<init>(Unknown Source)
        at java.io.ObjectStreamClass.lookup(Unknown Source)
        at java.io.ObjectOutputStream.writeObject0(Unknown
Source)
        at java.io.ObjectOutputStream.writeObject(Unknown
Source)
        at Blips.main(E28_BlipCheck.java:64)
        at E28_BlipCheck.main(E28_BlipCheck.java:48)
```

When we comment out the constructor (as above) the output is:

```
Constructing objects:
Blip1 Constructor
Saving objects:
Blip1.writeExternal
BlipCheck.writeExternal
Recovering b1:
Blip1 Constructor
Blip1.readExternal
Recovering b2:
BlipCheck.readExternal
```

Eliminating the explicit default constructor allows the compiler to generate the default constructor, which removes the exception. When the compiler synthesizes the default constructor for a **public** class, it makes that constructor **public**, so it works as an **Externalizable** object.

Exercise 29

```
//: io/E29_Blip3Test.java
/****************** Exercise 29 *****************
 * In Blip3.java, comment out the two lines after
 * the phrases "You must do this:" and run the
 * program. Explain the result and why it differs
 * from when the two lines are in the program.
 ***********************************************/
package io;
import java.io.*;
```

```
import static net.mindview.util.Print.*;

class Blip3B extends Blip3 {
  public Blip3B() {}
  public Blip3B(String x, int a) { super(x, a); }
  @Override public void writeExternal(ObjectOutput out)
  throws IOException {
    print("Blip3B.writeExternal");
    // You must do this:
//    out.writeObject(s);
//    out.writeInt(i);
  }
  @Override public void readExternal(ObjectInput in)
  throws IOException, ClassNotFoundException {
    print("Blip3B.readExternal");
    // You must do this:
//    s = (String)in.readObject();
//    i = in.readInt();
  }
}

public class E29_Blip3Test {
  public static void main(String[] args)
  throws IOException, ClassNotFoundException {
    print("Constructing objects:");
    Blip3B b3 = new Blip3B("A String ", 47);
    print(b3);
    ObjectOutputStream o = new ObjectOutputStream(
      new FileOutputStream("Blip3B.out"));
    print("Saving object:");
    o.writeObject(b3);
    o.close();
    // Now get it back:
    ObjectInputStream in = new ObjectInputStream(
      new FileInputStream("Blip3B.out"));
    print("Recovering b3:");
    b3 = (Blip3B)in.readObject();
    print(b3);
  }
} /* Output:
Constructing objects:
Blip3(String x, int a)
A String 47
Saving object:
Blip3B.writeExternal
Recovering b3:
Blip3 Constructor
```

```
Blip3B.readExternal
null0
*///:~
```

writeObject( ) calls the **writeExternal( )** method , and **readObject( )** calls **readExternal( )**. If these do not explicitly write and read the relevant parts of the object, they aren't stored or retrieved, so the object gets only the default values for its fields, **null** and **0**.

Exercise 30

```
//: io/E30_RepairCADState.java
/****************** Exercise 30 *****************
 * Repair the program CADState.java as described in
 * the text.
 ************************************************/
package io;
import java.io.*;
import java.util.*;

abstract class Shape implements Serializable {
  public static final int RED = 1, BLUE = 2, GREEN = 3;
  private int xPos, yPos, dimension;
  private static Random rand = new Random(47);
  private static int counter = 0;
  public abstract void setColor(int newColor);
  public abstract int getColor();
  public Shape(int xVal, int yVal, int dim) {
    xPos = xVal;
    yPos = yVal;
    dimension = dim;
  }
  public String toString() {
    return getClass() +
      "color[" + getColor() + "] xPos[" + xPos +
      "] yPos[" + yPos + "] dim[" + dimension + "]\n";
  }
  public static Shape randomFactory() {
    int xVal = rand.nextInt(100);
    int yVal = rand.nextInt(100);
    int dim = rand.nextInt(100);
    switch(counter++ % 3) {
      default:
      case 0: return new Circle(xVal, yVal, dim);
      case 1: return new Square(xVal, yVal, dim);
      case 2: return new Line(xVal, yVal, dim);
```

```
    }
  }
}

class Circle extends Shape {
  private static int color;
  public static void
  serializeStaticState(ObjectOutputStream os)
  throws IOException { os.writeInt(color); }
  public static void
  deserializeStaticState(ObjectInputStream os)
  throws IOException { color = os.readInt(); }
  public Circle(int xVal, int yVal, int dim) {
    super(xVal, yVal, dim);
  }
  public void setColor(int newColor) { color = newColor; }
  public int getColor() { return color; }
}

class Square extends Shape {
  private static int color;
  public static void
  serializeStaticState(ObjectOutputStream os)
  throws IOException { os.writeInt(color); }
  public static void
  deserializeStaticState(ObjectInputStream os)
  throws IOException { color = os.readInt(); }
  public Square(int xVal, int yVal, int dim) {
    super(xVal, yVal, dim);
  }
  public void setColor(int newColor) { color = newColor; }
  public int getColor() { return color; }
}

class Line extends Shape {
  private static int color;
  public static void
  serializeStaticState(ObjectOutputStream os)
  throws IOException { os.writeInt(color); }
  public static void
  deserializeStaticState(ObjectInputStream os)
  throws IOException { color = os.readInt(); }
  public Line(int xVal, int yVal, int dim) {
    super(xVal, yVal, dim);
  }
  public void setColor(int newColor) { color = newColor; }
  public int getColor() { return color; }
```

```
}

public class E30_RepairCADState {
  @SuppressWarnings("unchecked")
  public static void main(String[] args) throws Exception {
    List<Shape> shapes = new ArrayList<Shape>();
    // Make some shapes:
    for(int i = 0; i < 10; i++)
      shapes.add(Shape.randomFactory());
    // Set all the static colors to GREEN:
    for(int i = 0; i < 10; i++)
      ((Shape)shapes.get(i)).setColor(Shape.GREEN);
    // Save the state vector:
    ObjectOutputStream out = new ObjectOutputStream(
      new FileOutputStream("CADState.out"));
    Circle.serializeStaticState(out);
    Square.serializeStaticState(out);
    Line.serializeStaticState(out);
    out.writeObject(shapes);
    // Display the shapes:
    System.out.println(shapes);
    // Now read the file back in:
    ObjectInputStream in = new ObjectInputStream(
      new FileInputStream("CADState.out"));
    // Read in the same order they were written:
    Circle.deserializeStaticState(in);
    Square.deserializeStaticState(in);
    Line.deserializeStaticState(in);
    shapes = (List<Shape>)in.readObject();
    System.out.println(shapes);
  }
} /* Output:
[class io.Circlecolor[3] xPos[58] yPos[55] dim[93]
, class io.Squarecolor[3] xPos[61] yPos[61] dim[29]
, class io.Linecolor[3] xPos[68] yPos[0] dim[22]
, class io.Circlecolor[3] xPos[7] yPos[88] dim[28]
, class io.Squarecolor[3] xPos[51] yPos[89] dim[9]
, class io.Linecolor[3] xPos[78] yPos[98] dim[61]
, class io.Circlecolor[3] xPos[20] yPos[58] dim[16]
, class io.Squarecolor[3] xPos[40] yPos[11] dim[22]
, class io.Linecolor[3] xPos[4] yPos[83] dim[6]
, class io.Circlecolor[3] xPos[75] yPos[10] dim[42]
]
[class io.Circlecolor[3] xPos[58] yPos[55] dim[93]
, class io.Squarecolor[3] xPos[61] yPos[61] dim[29]
, class io.Linecolor[3] xPos[68] yPos[0] dim[22]
, class io.Circlecolor[3] xPos[7] yPos[88] dim[28]
```

```
, class io.Squarecolor[3] xPos[51] yPos[89] dim[9]
, class io.Linecolor[3] xPos[78] yPos[98] dim[61]
, class io.Circlecolor[3] xPos[20] yPos[58] dim[16]
, class io.Squarecolor[3] xPos[40] yPos[11] dim[22]
, class io.Linecolor[3] xPos[4] yPos[83] dim[6]
, class io.Circlecolor[3] xPos[75] yPos[10] dim[42]
]
*///:~
```

We retrieve the stored color for each type, rather than the default values.

Exercise 31

```
//: io/E31_PeopleWithAddresses.java
// {Requires: nu.xom.Node; You must install
// the XOM library from http://www.xom.nu }
/****************** Exercise 31 *****************
 * Add appropriate address information to Person.java
 * and People.java.
 ***********************************************/
package io;
import nu.xom.*;
import java.io.*;
import java.util.*;

class Person {
  private String first, last, address, city, state;
  private int zipCode;
  public Person(String first, String last, String address,
    String city, String state, int zipCode) {
    this.first = first;
    this.last = last;
    this.address = address;
    this.city = city;
    this.state = state;
    this.zipCode = zipCode;
  }
  // Produce an XML Element from this Person object:
  public Element getXML() {
    Element person = new Element("person");
    Element firstName = new Element("first");
    firstName.appendChild(first);
    Element lastName = new Element("last");
    lastName.appendChild(last);
    Element addr = new Element("address");
    addr.appendChild(address);
```

```
    Element cty = new Element("city");
    cty.appendChild(city);
    Element st = new Element("state");
    st.appendChild(state);
    Element zc = new Element("zipCode");
    zc.appendChild(Integer.toString(zipCode));
    person.appendChild(firstName);
    person.appendChild(lastName);
    person.appendChild(addr);
    person.appendChild(cty);
    person.appendChild(st);
    person.appendChild(zc);
    return person;
  }
  // Constructor to restore a Person from an XML Element:
  public Person(Element person) {
    first= person.getFirstChildElement("first").getValue();
    last = person.getFirstChildElement("last").getValue();
    address =
      person.getFirstChildElement("address").getValue();
    city = person.getFirstChildElement("city").getValue();
    state = person.getFirstChildElement("state").getValue();
    zipCode = Integer.valueOf(
      person.getFirstChildElement("zipCode").getValue());
  }
  public String toString() {
    return first + " " + last + " " + address + " " + city +
      " " + state + " " + zipCode;
  }
  // Make it human-readable:
  public static void
  format(OutputStream os, Document doc) throws Exception {
    Serializer serializer= new Serializer(os,"ISO-8859-1");
    serializer.setIndent(4);
    serializer.setMaxLength(60);
    serializer.write(doc);
    serializer.flush();
  }
}

class People extends ArrayList<Person> {
  public People(String fileName) throws Exception  {
    Document doc = new Builder().build(fileName);
    Elements elements =
      doc.getRootElement().getChildElements();
    for(int i = 0; i < elements.size(); i++)
      add(new Person(elements.get(i)));
```

```
  }
}

public class E31_PeopleWithAddresses {
  public static void main(String[] args) throws Exception {
    List<Person> people = Arrays.asList(
      new Person("Dr. Bunsen", "Honeydew", "Street 1",
        "New York", "NY", 10001),
      new Person("Gonzo", "The Great", "Street 2",
        "New York", "NY", 20002),
      new Person("Phillip J.", "Fry", "Street 3",
        "New York", "NY", 30003));
    System.out.println(people);
    Element root = new Element("people");
    for(Person p : people)
      root.appendChild(p.getXML());
    Document doc = new Document(root);
    Person.format(System.out, doc);
    Person.format(new BufferedOutputStream(
      new FileOutputStream("People.xml")), doc);
    People p = new People("People.xml");
    System.out.println(p);
  }
} /* Output:
[Dr. Bunsen Honeydew Street 1 New York NY 10001, Gonzo The
Great Street 2 New York NY 20002, Phillip J. Fry Street 3
New York NY 30003]
<?xml version="1.0" encoding="ISO-8859-1"?>
<people>
    <person>
        <first>Dr. Bunsen</first>
        <last>Honeydew</last>
        <address>Street 1</address>
        <city>New York</city>
        <state>NY</state>
        <zipCode>10001</zipCode>
    </person>
    <person>
        <first>Gonzo</first>
        <last>The Great</last>
        <address>Street 2</address>
        <city>New York</city>
        <state>NY</state>
        <zipCode>20002</zipCode>
    </person>
    <person>
        <first>Phillip J.</first>
```

```
        <last>Fry</last>
        <address>Street 3</address>
        <city>New York</city>
        <state>NY</state>
        <zipCode>30003</zipCode>
    </person>
</people>
[Dr. Bunsen Honeydew Street 1 New York NY 10001, Gonzo The
Great Street 2 New York NY 20002, Phillip J. Fry Street 3
New York NY 30003]
*///:~
```

Exercise 32

```
//: io/E32_WordsInfoXML.java
// {Requires: nu.xom.Node; You must install
// the XOM library from http://www.xom.nu }
// {RunByHand}
/****************** Exercise 32 *****************
 * Using a Map<String,Integer> and the
 * net.mindview.util.TextFile utility, write a
 * program that counts the occurrence of words in
 * a file (use "\\W+" as the second argument to the
 * TextFile constructor). Store the results as an
 * XML file.
 ***********************************************/
package io;
import nu.xom.*;
import java.io.*;
import java.util.*;
import net.mindview.util.*;

public class E32_WordsInfoXML {
  // Produce an XML Element from this Map.Entry object:
  static Element getXML(Map.Entry<String,Integer> me) {
    Element record = new Element("record");
    Element word = new Element("word");
    word.appendChild(me.getKey());
    Element freq = new Element("frequency");
    freq.appendChild(me.getValue().toString());
    record.appendChild(word);
    record.appendChild(freq);
    return record;
  }
  public static void main(String[] args) throws Exception {
    Map<String,Integer> wordsStat =
```

```
      new HashMap<String,Integer>();
    for(String word :
        new TextFile("E32_WordsInfoXML.java", "\\W+")) {
      Integer freq = wordsStat.get(word);
      wordsStat.put(word, freq == null ? 1 : freq + 1);
    }
    Element root = new Element("words");
    for(Map.Entry<String,Integer> me : wordsStat.entrySet())
      root.appendChild(getXML(me));
    Document doc = new Document(root);
    Person.format(System.out, doc);
    Person.format(
      new BufferedOutputStream(new FileOutputStream(
        "WordsInfo.xml")), doc);
  }
} ///:~
```

Exercise 33

```
//: io/E33_PreferencesDemo.java
// {RunByHand}
/****************** Exercise 33 *****************
 * Write a program that displays the current value
 * of a directory called "base directory" and
 * prompts you for a new value. Use the Preferences
 * API to store the value.
 ***********************************************/
package io;
import java.util.*;
import java.util.prefs.*;
import static net.mindview.util.Print.*;

public class E33_PreferencesDemo {
  public static void main(String[] args) throws Exception {
    Preferences prefs = Preferences
      .userNodeForPackage(E33_PreferencesDemo.class);
    String directory =
      prefs.get("base directory", "Not defined");
    print("directory: " + directory);
    Scanner sc = new Scanner(System.in);
    printnb("Enter a new directory: ");
    directory = sc.nextLine();
    prefs.put("base directory", directory);
    print("\ndirectory: " + directory);
  }
} ///:~
```

Enumerated Types

Exercise 1

```
//: enumerated/E01_TrafficLight2.java
/****************** Exercise 1 *****************
 * Use a static import to modify TrafficLight.java
 * so you don't have to qualify the enum instances.
 ***********************************************/
package enumerated;
import static net.mindview.util.Print.*;
import static enumerated.Signal.*;

public class E01_TrafficLight2 {
  Signal color = RED;
  public void change() {
    switch(color) {
      case RED:    color = GREEN;
                   break;
      case GREEN:  color = YELLOW;
                   break;
      case YELLOW: color = RED;
    }
  }
  public String toString() {
    return "The traffic light is " + color;
  }
  public static void main(String[] args) {
    E01_TrafficLight2 t = new E01_TrafficLight2();
    for(int i = 0; i < 7; i++) {
      print(t);
      t.change();
    }
  }
} /* Output:
The traffic light is RED
The traffic light is GREEN
The traffic light is YELLOW
The traffic light is RED
The traffic light is GREEN
The traffic light is YELLOW
The traffic light is RED
```

```
*///:~
```

```
//: enumerated/Signal.java
package enumerated;

public enum Signal { GREEN, YELLOW, RED, } ///:~
```

This technique is not possible if the **enum** is defined in the same file or the default package.

Exercise 2

```
//: enumerated/E02_EnumStaticImplementation.java
/****************** Exercise 2 *****************
 * Instead of implementing an interface, make
 * next() a static method. What are the benefits
 * and drawbacks of this approach?
 ***********************************************/
package enumerated;
import java.util.*;

enum CartoonCharacter {
  SLAPPY, SPANKY, PUNCHY, SILLY, BOUNCY, NUTTY, BOB;
  private static Random rand = new Random(47);
  public static CartoonCharacter next() {
    return values()[rand.nextInt(values().length)];
  }
}

public class E02_EnumStaticImplementation {
  public static void printNext() {
    System.out.print(CartoonCharacter.next() + ", ");
  }
  public static void main(String[] args) {
    for(int i = 0; i < 10; i++)
      printNext();
  }
} /* Output:
BOB, PUNCHY, BOB, SPANKY, NUTTY, PUNCHY, SLAPPY, NUTTY,
NUTTY, SLAPPY,
*///:~
```

The benefit of making the **next( )** method **static** is that you do not need an instance of the **enum** to call a method. The downside is that methods that take a **Generator** cannot accept **CartoonCharacter**.

Exercise 3

```
//: enumerated/E03_Meal.java
/****************** Exercise 3 *****************
 * Add a new Course to Course.java and demonstrate
 * that it works in Meal.java.
 ***********************************************/
package enumerated;
import net.mindview.util.*;

interface Food {
  enum Appetizer implements Food {
    SALAD, SOUP, SPRING_ROLLS;
  }
  enum Beverage implements Food {
    BEER, VINE, JUICE, COLA, WATER;
  }
  enum MainCourse implements Food {
    LASAGNE, BURRITO, PAD_THAI,
    LENTILS, HUMMOUS, VINDALOO;
  }
  enum Dessert implements Food {
    TIRAMISU, GELATO, BLACK_FOREST_CAKE,
    FRUIT, CREME_CARAMEL;
  }
  enum Coffee implements Food {
    BLACK_COFFEE, DECAF_COFFEE, ESPRESSO,
    LATTE, CAPPUCCINO, TEA, HERB_TEA;
  }
}

enum Course {
  APPETIZER(Food.Appetizer.class),
  MAINCOURSE(Food.MainCourse.class),
  BEVERAGE(Food.Beverage.class),
  DESSERT(Food.Dessert.class),
  COFFEE(Food.Coffee.class);
  private Food[] values;
  private Course(Class<? extends Food> kind) {
    values = kind.getEnumConstants();
  }
  public Food randomSelection() {
    return Enums.random(values);
  }
}
```

```
public class E03_Meal {
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++) {
      for(Course course : Course.values()) {
        Food food = course.randomSelection();
        System.out.println(food);
      }
      System.out.println("---");
    }
  }
} /* Output:
SPRING_ROLLS
VINDALOO
COLA
GELATO
TEA
---
SPRING_ROLLS
HUMMOUS
BEER
BLACK_FOREST_CAKE
BLACK_COFFEE
---
SALAD
LASAGNE
VINE
CREME_CARAMEL
LATTE
---
SOUP
HUMMOUS
VINE
TIRAMISU
ESPRESSO
---
SOUP
LASAGNE
VINE
BLACK_FOREST_CAKE
BLACK_COFFEE
---
*///:~
```

Exercise 4

```
//: enumerated/E04_Meal2.java
```

```
/****************** Exercise 4 *****************
 * Repeat the above exercise for Meal2.java.
 ***********************************************/
package enumerated;
import net.mindview.util.*;

enum Meal2 {
  APPETIZER(Food.Appetizer.class),
  MAINCOURSE(Food.MainCourse.class),
  BEVERAGE(Food.Beverage.class),
  DESSERT(Food.Dessert.class),
  COFFEE(Food.Coffee.class);
  private Food[] values;
  private Meal2(Class<? extends Food> kind) {
    values = kind.getEnumConstants();
  }
  public interface Food {
    enum Appetizer implements Food {
      SALAD, SOUP, SPRING_ROLLS;
    }
    enum MainCourse implements Food {
      LASAGNE, BURRITO, PAD_THAI,
      LENTILS, HUMMOUS, VINDALOO;
    }
    enum Beverage implements Food {
      BEER, VINE, JUICE, COLA, WATER;
    }
    enum Dessert implements Food {
      TIRAMISU, GELATO, BLACK_FOREST_CAKE,
      FRUIT, CREME_CARAMEL;
    }
    enum Coffee implements Food {
      BLACK_COFFEE, DECAF_COFFEE, ESPRESSO,
      LATTE, CAPPUCCINO, TEA, HERB_TEA;
    }
  }
  public Food randomSelection() {
    return Enums.random(values);
  }
}

public class E04_Meal2 {
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++) {
      for(Meal2 meal : Meal2.values()) {
        Meal2.Food food = meal.randomSelection();
        System.out.println(food);
```

```
      }
      System.out.println("---");
    }
  }
} /* Output:
SPRING_ROLLS
VINDALOO
COLA
GELATO
TEA
---
SPRING_ROLLS
HUMMOUS
BEER
BLACK_FOREST_CAKE
BLACK_COFFEE
---
SALAD
LASAGNE
VINE
CREME_CARAMEL
LATTE
---
SOUP
HUMMOUS
VINE
TIRAMISU
ESPRESSO
---
SOUP
LASAGNE
VINE
BLACK_FOREST_CAKE
BLACK_COFFEE
---
*///:~
```

Exercise 5

```
//: enumerated/E05_VowelsAndConsonants2.java
/****************** Exercise 5 *****************
 * Modify control/VowelsAndConsonants.java so that
 * it uses three enum types: VOWEL, SOMETIMES_A_VOWEL,
 * and CONSONANT. The enum constructor should take
 * the various letters that describe that particular
 * category. Hint: Use varargs, and remember that
```

```
 * varargs automatically creates an array for you.
 *********************************************/
package enumerated;
import java.util.*;
import static net.mindview.util.Print.*;

enum CharacterCategory {
  VOWEL('a', 'e', 'i', 'o', 'u') {
    public String toString() { return "vowel"; }
  },
  SOMETIMES_A_VOWEL('y', 'w') {
    public String toString() {
      return "sometimes a vowel";
    }
  },
  CONSONANT {
    public String toString() { return "consonant"; }
  };
  private HashSet<Character> chars =
    new HashSet<Character>();
  private CharacterCategory(Character... chars) {
    if(chars != null)
      this.chars.addAll(Arrays.asList(chars));
  }
  public static CharacterCategory getCategory(Character c) {
    if(VOWEL.chars.contains(c))
      return VOWEL;
    if(SOMETIMES_A_VOWEL.chars.contains(c))
      return SOMETIMES_A_VOWEL;
    return CONSONANT;
  }
}

public class E05_VowelsAndConsonants2 {
  public static void main(String[] args) {
    Random rand = new Random(47);
    for(int i = 0; i < 100; i++) {
      int c = rand.nextInt(26) + 'a';
      printnb((char)c + ", " + c + ": ");
      print(
        CharacterCategory.getCategory((char)c).toString());
    }
  }
} /* Output: (Sample)
y, 121: sometimes a vowel
n, 110: consonant
z, 122: consonant
```

```
b, 98: consonant
r, 114: consonant
...
h, 104: consonant
x, 120: consonant
x, 120: consonant
h, 104: consonant
v, 118: consonant
*///:~
```

The **getCategory( )** factory method returns the appropriate **enum** constant (based on the character provided): **VOWEL**, **SOMETIMES_A_VOWEL,** or **CONSONANT**. Each **enum** has a custom **toString( )** method to return the appropriate text for that **enum**. (More can be found in the *Constant-specific methods* section in *TIJ4*.)

Exercise 6

We nest **Appetizer**, **MainCourse**, **Dessert**, and **Coffee** inside **Food** rather than making them independent **enum**s that coincidentally use **Food**; this produces a clearer structure. When you see **Food.Appetizer** in the code, you know the terms are related. The clarity and comprehensibility of code are crucial, especially during maintenance. Remember, when you reduce software maintenance you increase profit.

Nesting has another benefit when you use reflection to build up the grouping **enum** (like **Course** from *TIJ4*). Try this as an additional exercise with *TIJ4*'s **enumerated/menu/Meal.java** program. Pay special attention to overcoming the JDK's constraints on the **java.lang.Enum** class.

Exercise 7

EnumSet is an abstract class with two private implementation classes: **JumboEnumSet** for types with more than 64 elements, and **RegularEnumSet** for types with up to 64 elements. The main factory method (entry point) of the **EnumSet** class is **noneOf( )**, which the other **static** methods also call. The design comes from *Patterns for Encapsulating Class Trees*. (See http://www.riehle.org/computer-science/research/1995/plop-1995-trading.html.)

TIJ4 describes **EnumSet**'s very efficient bit vector representation, whereby **JumboEnumSet** uses an array of **long**s, while **RegularEnumSet** uses a single **long**.

Exercise 8

```
//: enumerated/E08_MailForwarding.java
/****************** Exercise 8 *****************
 * Modify PostOffice.java so it has the ability to
 * forward mail.
 ***********************************************/
package enumerated;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class FMail {
  // The NO's lower the probability of random selection:
  enum GeneralDelivery {YES,NO1,NO2,NO3,NO4,NO5}
  enum Scannability {UNSCANNABLE,YES1,YES2,YES3,YES4}
  enum Readability {ILLEGIBLE,YES1,YES2,YES3,YES4}
  enum Address {INCORRECT,OK1,OK2,OK3,OK4,OK5,OK6}
  enum ReturnAddress {MISSING,OK1,OK2,OK3,OK4,OK5}
  enum ForwardAddress {MISSING,OK1,OK2,OK3,OK4,OK5}
  GeneralDelivery generalDelivery;
  Scannability scannability;
  Readability readability;
  Address address;
  ReturnAddress returnAddress;
  ForwardAddress forwardAddress;
  static long counter;
  long id = counter++;
  public String toString() { return "Mail " + id; }
  public String details() {
    return toString() +
      ", General Delivery: " + generalDelivery +
      ", Address Scanability: " + scannability +
      ", Address Readability: " + readability +
      ", Address Address: " + address +
      ", Return address: " + returnAddress +
      ", Forward address: " + forwardAddress;
  }
  // Generate test FMail:
  public static FMail randomMail() {
    FMail m = new FMail();
    m.generalDelivery= Enums.random(GeneralDelivery.class);
    m.scannability = Enums.random(Scannability.class);
    m.readability = Enums.random(Readability.class);
    m.address = Enums.random(Address.class);
    m.returnAddress = Enums.random(ReturnAddress.class);
```

```
    m.forwardAddress = Enums.random(ForwardAddress.class);
    return m;
  }
  public static Iterable<FMail> generator(final int count) {
    return new Iterable<FMail>() {
      int n = count;
      public Iterator<FMail> iterator() {
        return new Iterator<FMail>() {
          public boolean hasNext() { return n-- > 0; }
          public FMail next() { return randomMail(); }
          public void remove() { // Not implemented
            throw new UnsupportedOperationException();
          }
        };
      }
    };
  }
}

public class E08_MailForwarding {
  enum MailHandler {
    GENERAL_DELIVERY {
      boolean handle(FMail m) {
        switch(m.generalDelivery) {
          case YES:
            print("Using general delivery for " + m);
            return true;
          default: return false;
        }
      }
    },
    MACHINE_SCAN {
      boolean handle(FMail m) {
        switch(m.scannability) {
          case UNSCANNABLE: return false;
          default:
            switch(m.address) {
              case INCORRECT: return false;
              default:
                print("Delivering "+ m + " automatically");
                return true;
            }
        }
      }
    },
    VISUAL_INSPECTION {
      boolean handle(FMail m) {
```

```
        switch(m.readability) {
          case ILLEGIBLE: return false;
          default:
            switch(m.address) {
              case INCORRECT: return false;
              default:
                print("Delivering " + m + " normally");
                return true;
            }
        }
      }
    },
    FORWARD_MAIL {
      boolean handle(FMail m) {
        switch(m.forwardAddress) {
          case MISSING: return false;
          default:
            print("Forwarding " + m);
            return true;
        }
      }
    },
    RETURN_TO_SENDER {
      boolean handle(FMail m) {
        switch(m.returnAddress) {
          case MISSING: return false;
          default:
            print("Returning " + m + " to sender");
            return true;
        }
      }
    };
    abstract boolean handle(FMail m);
  }
  static void handle(FMail m) {
    for(MailHandler handler : MailHandler.values())
      if(handler.handle(m))
        return;
    print(m + " is a dead letter");
  }
  public static void main(String[] args) {
    for(FMail mail : FMail.generator(10)) {
      print(mail.details());
      handle(mail);
      print("*****");
    }
  }
```

```
} /* Output:
Mail 0, General Delivery: NO2, Address Scanability:
UNSCANNABLE, Address Readability: YES3, Address Address:
OK1, Return address: OK1, Forward address: OK5
Delivering Mail 0 normally
*****
Mail 1, General Delivery: NO4, Address Scanability:
UNSCANNABLE, Address Readability: YES2, Address Address:
INCORRECT, Return address: MISSING, Forward address: MISSING
Mail 1 is a dead letter
*****
Mail 2, General Delivery: NO3, Address Scanability: YES4,
Address Readability: YES4, Address Address: OK3, Return
address: OK4, Forward address: OK1
Delivering Mail 2 automatically
*****
Mail 3, General Delivery: NO2, Address Scanability: YES3,
Address Readability: YES1, Address Address: OK4, Return
address: OK1, Forward address: MISSING
Delivering Mail 3 automatically
*****
Mail 4, General Delivery: NO2, Address Scanability: YES3,
Address Readability: YES1, Address Address: OK5, Return
address: OK4, Forward address: OK2
Delivering Mail 4 automatically
*****
Mail 5, General Delivery: YES, Address Scanability: YES4,
Address Readability: ILLEGIBLE, Address Address: OK4, Return
address: OK4, Forward address: MISSING
Using general delivery for Mail 5
*****
Mail 6, General Delivery: NO1, Address Scanability: YES4,
Address Readability: YES4, Address Address: OK6, Return
address: OK3, Forward address: OK5
Delivering Mail 6 automatically
*****
Mail 7, General Delivery: YES, Address Scanability: YES2,
Address Readability: YES1, Address Address: INCORRECT,
Return address: OK1, Forward address: OK2
Using general delivery for Mail 7
*****
Mail 8, General Delivery: NO4, Address Scanability: YES1,
Address Readability: YES2, Address Address: INCORRECT,
Return address: OK4, Forward address: OK5
Forwarding Mail 8
*****
```

```
Mail 9, General Delivery: NO1, Address Scanability: YES4,
Address Readability: YES1, Address Address: OK5, Return
address: MISSING, Forward address: OK1
Delivering Mail 9 automatically
*****
*///:~
```

Exercise 9

```
//: enumerated/E09_PostOffice2.java
/****************** Exercise 9 *****************
 * Modify class PostOffice so that it uses an
 * EnumMap.
 ***********************************************/
package enumerated;
import java.util.*;
import static net.mindview.util.Print.*;

interface Command { boolean handle(Mail m); }

public class E09_PostOffice2 {
  static EnumMap<MailHandler,Command> em =
    new EnumMap<MailHandler,Command>(MailHandler.class);
  static {
    em.put(MailHandler.GENERAL_DELIVERY, new Command() {
      public boolean handle(Mail m) {
        switch(m.generalDelivery) {
          case YES:
            print("Using general delivery for " + m);
            return true;
          default: return false;
        }
      }
    });
    em.put(MailHandler.MACHINE_SCAN, new Command() {
      public boolean handle(Mail m) {
        switch(m.scannability) {
          case UNSCANNABLE: return false;
          default:
            switch(m.address) {
              case INCORRECT: return false;
              default:
                print("Delivering "+ m + " automatically");
                return true;
            }
        }
```

```
      }
    });
    em.put(MailHandler.VISUAL_INSPECTION, new Command() {
      public boolean handle(Mail m) {
        switch(m.readability) {
          case ILLEGIBLE: return false;
          default:
            switch(m.address) {
              case INCORRECT: return false;
              default:
                print("Delivering " + m + " normally");
                return true;
            }
        }
      }
    });
    em.put(MailHandler.RETURN_TO_SENDER, new Command() {
      public boolean handle(Mail m) {
        switch(m.returnAddress) {
          case MISSING: return false;
          default:
            print("Returning " + m + " to sender");
            return true;
        }
      }
    });
  }
  enum MailHandler {
    GENERAL_DELIVERY, MACHINE_SCAN, VISUAL_INSPECTION,
    RETURN_TO_SENDER;
  }
  static void handle(Mail m) {
    for(Command cmd : em.values())
      if(cmd.handle(m))
        return;
    print(m + " is a dead letter");
  }
  public static void main(String[] args) {
    for(Mail mail : Mail.generator(10)) {
      print(mail.details());
      handle(mail);
      print("*****");
    }
  }
} /* Output:
```

```
Mail 0, General Delivery: NO2, Address Scanability:
UNSCANNABLE, Address Readability: YES3, Address Address:
OK1, Return address: OK1
Delivering Mail 0 normally
*****
Mail 1, General Delivery: NO5, Address Scanability: YES3,
Address Readability: ILLEGIBLE, Address Address: OK5, Return
address: OK1
Delivering Mail 1 automatically
*****
Mail 2, General Delivery: YES, Address Scanability: YES3,
Address Readability: YES1, Address Address: OK1, Return
address: OK5
Using general delivery for Mail 2
*****
Mail 3, General Delivery: NO4, Address Scanability: YES3,
Address Readability: YES1, Address Address: INCORRECT,
Return address: OK4
Returning Mail 3 to sender
*****
Mail 4, General Delivery: NO4, Address Scanability:
UNSCANNABLE, Address Readability: YES1, Address Address:
INCORRECT, Return address: OK2
Returning Mail 4 to sender
*****
Mail 5, General Delivery: NO3, Address Scanability: YES1,
Address Readability: ILLEGIBLE, Address Address: OK4, Return
address: OK2
Delivering Mail 5 automatically
*****
Mail 6, General Delivery: YES, Address Scanability: YES4,
Address Readability: ILLEGIBLE, Address Address: OK4, Return
address: OK4
Using general delivery for Mail 6
*****
Mail 7, General Delivery: YES, Address Scanability: YES3,
Address Readability: YES4, Address Address: OK2, Return
address: MISSING
Using general delivery for Mail 7
*****
Mail 8, General Delivery: NO3, Address Scanability: YES1,
Address Readability: YES3, Address Address: INCORRECT,
Return address: MISSING
Mail 8 is a dead letter
*****
```

```
Mail 9, General Delivery: NO1, Address Scanability:
UNSCANNABLE, Address Readability: YES2, Address Address:
OK1, Return address: OK4
Delivering Mail 9 normally
*****
*///:~
```

Exercise 10

```
//: enumerated/E10_VendingMachine2.java
// {Args: VendingMachineInput.txt}
/****************** Exercise 10 *****************
 * Modify class VendingMachine (only) using EnumMap
 * so that one program can have multiple instances
 * of VendingMachine.
 ***********************************************/
package enumerated;
import java.util.*;
import java.util.concurrent.locks.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

class VendingMachine2 {
  // ******************************
  // *** NEWLY ADDED CODE: BEGIN ***
  // ******************************
  private static class Context {
    private State state = State.RESTING;
    private int amount;
    private Input selection;
  }
  private static Map<Machine,Context> em =
    Collections.synchronizedMap(
      new EnumMap<Machine,Context>(Machine.class));
  static {
    for(Machine m : Machine.values())
      em.put(m, new Context());
  }
  enum Machine { M1, M2, M3 }
  private static final ReentrantLock lock =
    new ReentrantLock();
  // ******************************
  // *** NEWLY ADDED CODE: END ***
  // ******************************
  private static State state;
  private static int amount;
```

```
  private static Input selection;
  enum StateDuration { TRANSIENT } // Tagging enum
  enum State {
    RESTING {
      void next(Input input) {
        switch(Category.categorize(input)) {
          case MONEY:
            amount += input.amount();
            state = ADDING_MONEY;
            break;
          case SHUT_DOWN:
            state = TERMINAL;
          default:
        }
      }
    },
    ADDING_MONEY {
      void next(Input input) {
        switch(Category.categorize(input)) {
          case MONEY:
            amount += input.amount();
            break;
          case ITEM_SELECTION:
            selection = input;
            if(amount < selection.amount())
              print("Insufficient money for " + selection);
            else state = DISPENSING;
            break;
          case QUIT_TRANSACTION:
            state = GIVING_CHANGE;
            break;
          case SHUT_DOWN:
            state = TERMINAL;
          default:
        }
      }
    },
    DISPENSING(StateDuration.TRANSIENT) {
      void next() {
        print("here is your " + selection);
        amount -= selection.amount();
        state = GIVING_CHANGE;
      }
    },
    GIVING_CHANGE(StateDuration.TRANSIENT) {
      void next() {
        if(amount > 0) {
```

```
          print("Your change: " + amount);
          amount = 0;
        }
        state = RESTING;
      }
    },
    TERMINAL { void output() { print("Halted"); } };
    private boolean isTransient = false;
    State() {}
    State(StateDuration trans) { isTransient = true; }
    void next(Input input) {
      throw new RuntimeException("Only call " +
        "next(Input input) for non-transient states");
    }
    void next() {
      throw new RuntimeException("Only call next() for " +
        "StateDuration.TRANSIENT states");
    }
    void output() { print(amount); }
  }
  // This method is now taking an extra parameter denoting
  // the ID of the VendingMachine "instance".
  static void run(Generator<Input> gen, Machine m) {
    Context ctx = em.get(m);
    while(ctx.state != State.TERMINAL) {
      lock.lock();
      state = ctx.state;
      amount = ctx.amount;
      selection = ctx.selection;
      try {
        state.next(gen.next());
        while(state.isTransient)
          state.next();
        state.output();
        ctx.state = state;
        ctx.amount = amount;
        ctx.selection = selection;
        em.put(m, ctx);
      } finally {
        lock.unlock();
      }
      Thread.yield();
    }
  }
}

public class E10_VendingMachine2 {
```

```
  public static void main(String[] args) {
    for(final VendingMachine2.Machine m :
        VendingMachine2.Machine.values()) {
      Generator<Input> gen = new RandomInputGenerator();
      if(args.length == 1)
        gen = new FileInputGenerator(args[0]);
      final Generator<Input> g = gen;
      new Thread(new Runnable() {
        public void run() { VendingMachine2.run(g, m); }
      }).start();
    }
  }
} /* Output: (Sample)
25
25
25
50
50
50
75
75
75
here is your CHIPS
0
here is your CHIPS
0
here is your CHIPS
0
100
100
100
200
200
200
here is your TOOTHPASTE
0
here is your TOOTHPASTE
0
here is your TOOTHPASTE
0
25
25
25
35
35
35
Your change: 35
```

```
0
Your change: 35
0
Your change: 35
0
25
25
25
35
35
35
Insufficient money for SODA
35
Insufficient money for SODA
35
Insufficient money for SODA
35
60
60
60
70
70
70
75
75
75
Insufficient money for SODA
75
Insufficient money for SODA
75
Insufficient money for SODA
75
Your change: 75
0
Your change: 75
0
Your change: 75
0
Halted
Halted
Halted
*///:~
```

Each instance of **VendingMachine2** has a different set of values for the variables that define its state, all grouped as the **Context** class. We associate each **VendingMachine2** instance with its context information using

EnumMap, which we fill with data during class initialization. **Machine enum** enumerates each different instance.

We alter the **run( )** method to incorporate the context switching mechanism. We run each machine in a different thread, as shown in the **main( )** method. Threading is only used as a demonstration in this solution; you don't need to know the details yet.

Exercise 11

```
//: enumerated/E11_VendingMachine3.java
// {Args: VendedItems.txt VendingMachineInput.txt}
/****************** Exercise 11 *****************
 * In a real vending machine you will want to easily
 * add and change the type of vended items, so the
 * limits imposed by an enum on Input are impractical
 * (remember that enums are for a restricted set of
 * types). Modify VendingMachine.java so that the
 * vended items are represented by a class instead
 * of being part of Input, and initialize an
 * ArrayList of these objects from a text file (using
 * net.mindview.util.TextFile).
 ***********************************************/
package enumerated;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;

// A simple data holder class
class VendedItem {
  int amount;
  String name;
  VendedItem(String name, int amount) {
    this.name = name;
    this.amount = amount;
  }
  // The data is expected to be in a format: <name> <amount>
  public static VendedItem parse(String data) {
    String[] s = data.split(" ");
    return new VendedItem(s[0], Integer.parseInt(s[1]));
  }
  private static List<VendedItem> items =
    new ArrayList<VendedItem>();
  public static void addItem(VendedItem item) {
    items.add(item);
```

```
  }
  // A very slow lookup procedure
  public static VendedItem lookup(String name) {
    for(VendedItem item : items)
      if(item.name.equals(name))
        return item;
    return null;
  }
  private static Random rand = new Random(47);
  public static VendedItem randomSelection() {
    return items.get(rand.nextInt(items.size()));
  }
}

// A class representing an input to a state machine
class ExtInput {
  Input2 input;
  VendedItem item;
  ExtInput(Input2 input, VendedItem item) {
    this.input = input;
    this.item = item;
  }
  public int amount() {
    return item != null ? item.amount : input.amount();
  }
  public String toString() {
    return item != null ? item.name : input.toString();
  }
}

enum Input2 {
  NICKEL(5), DIME(10), QUARTER(25), DOLLAR(100),
  VENDED_ITEM,
  ABORT_TRANSACTION {
    public int amount() { // Disallow
      throw new RuntimeException("ABORT.amount()");
    }
  },
  STOP { // This must be the last instance.
    public int amount() { // Disallow
      throw new RuntimeException("SHUT_DOWN.amount()");
    }
  };
  int value; // In cents
  Input2(int value) { this.value = value; }
  Input2() {}
  int amount() { return value; }; // In cents
```

```
  static Random rand = new Random(47);
  public static Input2 randomSelection() {
    // Don't include STOP:
    return values()[rand.nextInt(values().length - 1)];
  }
}

enum Category2 {
  MONEY(Input2.NICKEL, Input2.DIME, Input2.QUARTER,
    Input2.DOLLAR),
  ITEM_SELECTION(Input2.VENDED_ITEM),
  QUIT_TRANSACTION(Input2.ABORT_TRANSACTION),
  SHUT_DOWN(Input2.STOP);
  private Input2[] values;
  Category2(Input2... types) { values = types; }
  private static EnumMap<Input2,Category2> categories =
    new EnumMap<Input2,Category2>(Input2.class);
  static {
    for(Category2 c : Category2.class.getEnumConstants())
      for(Input2 type : c.values)
        categories.put(type, c);
  }
  public static Category2 categorize(Input2 input) {
    return categories.get(input);
  }
}

public class E11_VendingMachine3 {
  private static State state = State.RESTING;
  private static int amount = 0;
  private static ExtInput selection = null;
  enum StateDuration { TRANSIENT } // Tagging enum
  enum State {
    RESTING {
      void next(ExtInput input) {
        switch(Category2.categorize(input.input)) {
          case MONEY:
            amount += input.amount();
            state = ADDING_MONEY;
            break;
          case SHUT_DOWN:
            state = TERMINAL;
          default:
        }
      }
    },
    ADDING_MONEY {
```

```
      void next(ExtInput input) {
        switch(Category2.categorize(input.input)) {
          case MONEY:
            amount += input.amount();
            break;
          case ITEM_SELECTION:
            selection = input;
            if(amount < selection.amount())
              print("Insufficient money for " + selection);
            else state = DISPENSING;
            break;
          case QUIT_TRANSACTION:
            state = GIVING_CHANGE;
            break;
          case SHUT_DOWN:
            state = TERMINAL;
          default:
        }
      }
    },
    DISPENSING(StateDuration.TRANSIENT) {
      void next() {
        print("here is your " + selection);
        amount -= selection.amount();
        state = GIVING_CHANGE;
      }
    },
    GIVING_CHANGE(StateDuration.TRANSIENT) {
      void next() {
        if(amount > 0) {
          print("Your change: " + amount);
          amount = 0;
        }
        state = RESTING;
      }
    },
    TERMINAL { void output() { print("Halted"); } };
    private boolean isTransient = false;
    State() {}
    State(StateDuration trans) { isTransient = true; }
    void next(ExtInput input) {
      throw new RuntimeException("Only call " +
        "next(ExtInput input) for non-transient states");
    }
    void next() {
      throw new RuntimeException("Only call next() for " +
        "StateDuration.TRANSIENT states");
```

```
    }
    void output() { print(amount); }
  }
  static void run(Generator<ExtInput> gen) {
    while(state != State.TERMINAL) {
      state.next(gen.next());
      while(state.isTransient)
        state.next();
      state.output();
    }
  }
  public static void main(String[] args) {
    if(args.length < 1) {
      System.err.println(
        "The vended items data file is not given!");
      return;
    }
    Generator<ExtInput> gen;
    if(args.length == 2)
      gen = new FileExtInputGenerator(args[1]);
    else
      gen = new RandomExtInputGenerator();
    // Parse the vended items data file
    for(String data : new TextFile(args[0], ";"))
      VendedItem.addItem(VendedItem.parse(data.trim()));
    run(gen);
  }
}

// For a basic sanity check:
class RandomExtInputGenerator
implements Generator<ExtInput> {
  public ExtInput next() {
    return new ExtInput(Input2.randomSelection(),
      VendedItem.randomSelection());
  }
}

// Create Inputs from a file of ';'-separated strings:
class FileExtInputGenerator implements Generator<ExtInput> {
  private Iterator<String> input;
  public FileExtInputGenerator(String fileName) {
    input = new TextFile(fileName, ";").iterator();
  }
  public ExtInput next() {
    if(!input.hasNext())
      return null;
```

```
    String s = input.next().trim();
    try {
      return
        new ExtInput(Enum.valueOf(Input2.class,s), null);
    } catch(IllegalArgumentException e) {
      // B plan: probably a vended item...
      VendedItem item = VendedItem.lookup(s);
      if(item != null)
        return new ExtInput(Input2.VENDED_ITEM, item);
      throw e;  // Rethrow the catched exception
    }
  }
} /* Output:
25
50
75
here is your CHIPS
0
100
200
here is your TOOTHPASTE
0
25
35
Your change: 35
0
25
35
Insufficient money for SODA
35
60
70
75
Insufficient money for SODA
75
Your change: 75
0
Halted
*///:~
```

This program uses a text file containing the merchandise list and merchandise values. Here's the text file we use to produce the output above (**VendingMachineInput.txt**'s content remains the same):

```
//:! enumerated/VendedItems.txt
TOOTHPASTE 200;CHIPS 75;SODA 100;SOAP 50
///:~
```

We add two classes to **VendingMachine.java**: **VendedItem** is a data holder with some utility methods, like **parse( )**, with a **static ArrayList** holding all items read from **VendingMachine.java**; and **ExtInput** represents an input to a state machine. **State enum** does not require substantial change.

Annotations

Exercise 1

```
//: annotations/E01_TableCreator.java
// {Args: annotations.Member}
/****************** Exercise 01 *****************
 * Implement more SQL types in the database example.
 ***********************************************/
package annotations;
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.util.*;
import annotations.database.*;

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@interface SQLBoolean {
  String name() default "";
  Constraints constraints() default @Constraints;
}

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@interface SQLCharacter {
  int value() default 0;
  String name() default "";
  Constraints constraints() default @Constraints;
}

@DBTable(name = "MEMBER")
class Member {
  @SQLString(30) String firstName;
  @SQLString(50) String lastName;
  @SQLInteger Integer age;
  @SQLCharacter(value = 15,
  constraints = @Constraints(primaryKey = true))
  String handle;
  static int memberCount;
  @SQLBoolean Boolean isVIP;
  public String getHandle() { return handle; }
  public String getFirstName() { return firstName; }
```

```
  public String getLastName() { return lastName; }
  public String toString() { return handle; }
  public Integer getAge() { return age; }
  public Boolean isVIP() { return isVIP; }
}

public class E01_TableCreator {
  public static void main(String[] args) throws Exception {
    if(args.length < 1) {
      System.out.println("arguments: annotated classes");
      System.exit(0);
    }
    for(String className : args) {
      Class<?> cl = Class.forName(className);
      DBTable dbTable = cl.getAnnotation(DBTable.class);
      if(dbTable == null) {
        System.out.println(
          "No DBTable annotations in class " + className);
        continue;
      }
      String tableName = dbTable.name();
      // If the name is empty, use the Class name:
      if(tableName.length() < 1)
        tableName = cl.getName().toUpperCase();
      List<String> columnDefs = new ArrayList<String>();
      for(Field field : cl.getDeclaredFields()) {
        String columnName = null;
        Annotation[] anns = field.getDeclaredAnnotations();
        if(anns.length < 1)
          continue; // Not a db table column
        if(anns[0] instanceof SQLInteger) {
          SQLInteger sInt = (SQLInteger) anns[0];
          // Use field name if name not specified
          if(sInt.name().length() < 1)
            columnName = field.getName().toUpperCase();
          else
            columnName = sInt.name();
          columnDefs.add(columnName + " INT" +
            getConstraints(sInt.constraints()));
        } else if(anns[0] instanceof SQLString) {
          SQLString sString = (SQLString) anns[0];
          // Use field name if name not specified.
          if(sString.name().length() < 1)
            columnName = field.getName().toUpperCase();
          else
            columnName = sString.name();
          columnDefs.add(columnName + " VARCHAR(" +
```

```
            sString.value() + ")" +
            getConstraints(sString.constraints()));
        } else if(anns[0] instanceof SQLBoolean) {
          SQLBoolean sBol = (SQLBoolean) anns[0];
          // Use field name if name not specified
          if(sBol.name().length() < 1)
            columnName = field.getName().toUpperCase();
          else
            columnName = sBol.name();
          columnDefs.add(columnName + " BOOLEAN" +
            getConstraints(sBol.constraints()));
        } else if(anns[0] instanceof SQLCharacter) {
          SQLCharacter sChar = (SQLCharacter) anns[0];
          // Use field name if name not specified.
          if(sChar.name().length() < 1)
            columnName = field.getName().toUpperCase();
          else
            columnName = sChar.name();
          columnDefs.add(columnName + " CHARACTER(" +
            sChar.value() + ")" +
            getConstraints(sChar.constraints()));
        }
        StringBuilder createCommand = new StringBuilder(
          "CREATE TABLE " + tableName + "(");
        for(String columnDef : columnDefs)
          createCommand.append("\n    " + columnDef + ",");
        // Remove trailing comma
        String tableCreate = createCommand.substring(
          0, createCommand.length() - 1) + ");";
        System.out.println("Table Creation SQL for " +
          className + " is :\n" + tableCreate);
      }
    }
  }
  private static String getConstraints(Constraints con) {
    String constraints = "";
    if(!con.allowNull())
      constraints += " NOT NULL";
    if(con.primaryKey())
      constraints += " PRIMARY KEY";
    if(con.unique())
      constraints += " UNIQUE";
    return constraints;
  }
} /* Output:
Table Creation SQL for annotations.Member is :
CREATE TABLE MEMBER(
```

```
    FIRSTNAME VARCHAR(30));
Table Creation SQL for annotations.Member is :
CREATE TABLE MEMBER(
    FIRSTNAME VARCHAR(30),
    LASTNAME VARCHAR(50));
Table Creation SQL for annotations.Member is :
CREATE TABLE MEMBER(
    FIRSTNAME VARCHAR(30),
    LASTNAME VARCHAR(50),
    AGE INT);
Table Creation SQL for annotations.Member is :
CREATE TABLE MEMBER(
    FIRSTNAME VARCHAR(30),
    LASTNAME VARCHAR(50),
    AGE INT,
    HANDLE CHARACTER(15) PRIMARY KEY);
Table Creation SQL for annotations.Member is :
CREATE TABLE MEMBER(
    FIRSTNAME VARCHAR(30),
    LASTNAME VARCHAR(50),
    AGE INT,
    HANDLE CHARACTER(15) PRIMARY KEY,
    ISVIP BOOLEAN);
*///:~
```

We add two new SQL types: **SQLBoolean** and **SQLCharacter**; the latter represents a fixed-size string.

Exercise 2

```
//: annotations/E02_InterfaceExtractorProcessorFactory.java
// APT-based annotation processing.
/****************** Exercise 02 *****************
 * Add support for division to the interface
 * extractor.
 ************************************************/
package annotations;
import com.sun.mirror.apt.*;
import com.sun.mirror.declaration.*;
import java.util.*;

public class E02_InterfaceExtractorProcessorFactory
  implements AnnotationProcessorFactory {
  public AnnotationProcessor getProcessorFor(
    Set<AnnotationTypeDeclaration> atds,
    AnnotationProcessorEnvironment env) {
```

```
    return new InterfaceExtractorProcessor(env);
  }
  public Collection<String> supportedAnnotationTypes() {
    return
     Collections.singleton("annotations.ExtractInterface");
  }
  public Collection<String> supportedOptions() {
    return Collections.emptySet();
  }
} ///:~
```

```
//: annotations/Divisor.java
package annotations;

@ExtractInterface("IDivisor")
class Divisor {
  public int divide(int x, int y) {
    int total = 0;
    while(x >= y) {
      x = sub(x, y);
      total++;
    }
    return total;
  }
  private int sub(int x, int y) { return x - y; }
  public static void main(String[] args) {
    System.out.println(
      "2678/134 = " + new Divisor().divide(2678, 134));
  }
} /* Output:
2678/134 = 19
*///:~
```

```
//: annotations/InterfaceExtractorProcessor.java
package annotations;
import com.sun.mirror.apt.*;
import com.sun.mirror.declaration.*;
import java.io.*;
import java.util.*;

public class InterfaceExtractorProcessor
  implements AnnotationProcessor {
  private final AnnotationProcessorEnvironment env;
  private ArrayList<MethodDeclaration> interfaceMethods =
    new ArrayList<MethodDeclaration>();
  public InterfaceExtractorProcessor(
    AnnotationProcessorEnvironment env) { this.env = env; }
```

```
  public void process() {
    for(TypeDeclaration typeDecl :
      env.getSpecifiedTypeDeclarations()) {
      interfaceMethods.clear();
      ExtractInterface annot =
        typeDecl.getAnnotation(ExtractInterface.class);
      if(annot == null)
        break;
      for(MethodDeclaration m : typeDecl.getMethods())
        if(m.getModifiers().contains(Modifier.PUBLIC) &&
           !(m.getModifiers().contains(Modifier.STATIC)))
          interfaceMethods.add(m);
      if(interfaceMethods.size() > 0) {
        try {
          PrintWriter writer =
            env.getFiler().createSourceFile(annot.value());
          writer.println("package " +
            typeDecl.getPackage().getQualifiedName() +";");
          writer.println("public interface " +
            annot.value() + " {");
          for(MethodDeclaration m : interfaceMethods) {
            writer.print("  public ");
            writer.print(m.getReturnType() + " ");
            writer.print(m.getSimpleName() + " (");
            int i = 0;
            for(ParameterDeclaration parm :
              m.getParameters()) {
              writer.print(parm.getType() + " " +
                parm.getSimpleName());
              if(++i < m.getParameters().size())
                writer.print(", ");
            }
            writer.println(");");
          }
          writer.println("}");
          writer.close();
        } catch(IOException ioe) {
          throw new RuntimeException(ioe);
        }
      }
    }
  }
} ///:~
```

Divisor, like **Multiplier**, works only with positive integers. With one minor modification to the **InterfaceExtractorProcessor**, **interfaceMethods** now

clears at each iteration (and thus avoids accumulating methods from other classes). (To properly invoke **apt**, see **build.xml**.)

Exercise 3

```
//: annotations/E03_TableCreationProcessorFactory.java
/****************** Exercise 03 *****************
 * Add support for more SQL types to
 * TableCreationProcessorFactory.java.
 ************************************************/
package annotations;
import com.sun.mirror.apt.*;
import com.sun.mirror.declaration.*;
import com.sun.mirror.util.*;
import java.util.*;
import static com.sun.mirror.util.DeclarationVisitors.*;
import annotations.database.*;

public class E03_TableCreationProcessorFactory
  implements AnnotationProcessorFactory {
  public AnnotationProcessor getProcessorFor(
    Set<AnnotationTypeDeclaration> atds,
    AnnotationProcessorEnvironment env) {
    return new TableCreationProcessor(env);
  }
  public Collection<String> supportedAnnotationTypes() {
    return Arrays.asList(
      "annotations.database.DBTable",
      "annotations.database.Constraints",
      "annotations.database.SQLString",
      "annotations.database.SQLInteger",
      "annotations.SQLBoolean",
      "annotations.SQLCharacter");
  }
  public Collection<String> supportedOptions() {
    return Collections.emptySet();
  }
  private static class TableCreationProcessor
    implements AnnotationProcessor {
    private final AnnotationProcessorEnvironment env;
    private String sql = "";
    public TableCreationProcessor(
      AnnotationProcessorEnvironment env) {
      this.env = env;
    }
```

```
  public void process() {
    for(TypeDeclaration typeDecl :
      env.getSpecifiedTypeDeclarations()) {
      typeDecl.accept(getDeclarationScanner(
        new TableCreationVisitor(), NO_OP));
      sql = sql.substring(0, sql.length() - 1) + ");";
      System.out.println("creation SQL is :\n" + sql);
      sql = "";
    }
  }
  private class TableCreationVisitor
    extends SimpleDeclarationVisitor {
    public void visitClassDeclaration(
      ClassDeclaration d) {
      DBTable dbTable = d.getAnnotation(DBTable.class);
      if(dbTable != null) {
        sql += "CREATE TABLE ";
        sql += (dbTable.name().length() < 1)
          ? d.getSimpleName().toUpperCase()
          : dbTable.name();
        sql += " (";
      }
    }
    public void visitFieldDeclaration(
      FieldDeclaration d) {
      String columnName = "";
      if(d.getAnnotation(SQLInteger.class) != null) {
        SQLInteger sInt = d.getAnnotation(
            SQLInteger.class);
        // Use field name if name not specified
        if(sInt.name().length() < 1)
          columnName = d.getSimpleName().toUpperCase();
        else
          columnName = sInt.name();
        sql += "\n    " + columnName + " INT" +
          getConstraints(sInt.constraints()) + ",";
      }
      if(d.getAnnotation(SQLString.class) != null) {
        SQLString sString = d.getAnnotation(
            SQLString.class);
        // Use field name if name not specified.
        if(sString.name().length() < 1)
          columnName = d.getSimpleName().toUpperCase();
        else
          columnName = sString.name();
        sql += "\n    " + columnName + " VARCHAR(" +
          sString.value() + ")" +
```

```
            getConstraints(sString.constraints()) + ",";
        }
        if(d.getAnnotation(SQLBoolean.class) != null) {
          SQLBoolean sBol = d.getAnnotation(
              SQLBoolean.class);
          // Use field name if name not specified
          if(sBol.name().length() < 1)
            columnName = d.getSimpleName().toUpperCase();
          else
            columnName = sBol.name();
          sql += "\n    " + columnName + " BOOLEAN" +
            getConstraints(sBol.constraints()) + ",";
        }
        if(d.getAnnotation(SQLCharacter.class) != null) {
          SQLCharacter sChar = d.getAnnotation(
              SQLCharacter.class);
          // Use field name if name not specified.
          if(sChar.name().length() < 1)
            columnName = d.getSimpleName().toUpperCase();
          else
            columnName = sChar.name();
          sql += "\n    " + columnName + " CHARACTER(" +
            sChar.value() + ")" +
            getConstraints(sChar.constraints()) + ",";
        }
      }
      private String getConstraints(Constraints con) {
        String constraints = "";
        if(!con.allowNull())
          constraints += " NOT NULL";
        if(con.primaryKey())
          constraints += " PRIMARY KEY";
        if(con.unique())
          constraints += " UNIQUE";
        return constraints;
      }
    }
  }
} ///:~
```

We reuse the new SQL types introduced in Exercise 1. (The **build.xml** file shows how to properly invoke **apt**.)

The **–XclassesAsDecls** non-standard **apt** option treats both class and source files as declarations to process. If you preserve annotations in class definitions, you can list compiled classes as declarations to process instead of source files, and accelerate the whole cycle. The output is:

```
creation SQL is :
CREATE TABLE MEMBER (
    FIRSTNAME VARCHAR(30),
    LASTNAME VARCHAR(50),
    AGE INT,
    HANDLE CHARACTER(15) PRIMARY KEY,
    ISVIP BOOLEAN);
```

Exercise 4

```
//: annotations/E04_TestSetupFixture.java
/****************** Exercise 04 *****************
 * Verify that a new testObject is created before
 * each test.
 ***********************************************/
package annotations;
import java.util.*;
import net.mindview.atunit.*;
import net.mindview.util.*;

public class E04_TestSetupFixture {
  HashSet<String> testObject = new HashSet<String>();
  @Test void _t1() {
    assert testObject.isEmpty();
    testObject.add("one");
  }
  @Test void _t2() {
    assert testObject.isEmpty();
    testObject.add("one");
  }
  public static void main(String[] args) throws Exception {
    OSExecute.command("java " +
      "net.mindview.atunit.AtUnit E04_TestSetupFixture");
  }
} /* Output:
annotations.E04_TestSetupFixture
  . _t1
  . _t2
OK (2 tests)
*///:~
```

We create a new **testObject** before each test; otherwise either **_t1** or **_t2** would fail (depending on the order of execution).

Exercise 5

```
//: annotations/E05_TestSetupFixture2.java
/****************** Exercise 05 *****************
 * Modify the above example to use the inheritance
 * approach.
 ************************************************/
package annotations;
import java.util.*;
import net.mindview.atunit.*;
import net.mindview.util.*;

public class E05_TestSetupFixture2 extends HashSet<String> {
  @Test void _t1() {
    assert isEmpty();
    add("one");
  }
  @Test void _t2() {
    assert isEmpty();
    add("one");
  }
  public static void main(String[] args) throws Exception {
    OSExecute.command("java " +
      "net.mindview.atunit.AtUnit E05_TestSetupFixture2");
  }
} /* Output:
annotations.E05_TestSetupFixture2
  . _t1
  . _t2
OK (2 tests)
*///:~
```

Exercise 6

```
//: annotations/E06_LinkedListTest.java
/****************** Exercise 06 *****************
 * Test LinkedList using the approach shown in
 * HashSetTest.java.
 ************************************************/
package annotations;
import java.util.*;
import net.mindview.atunit.*;
import net.mindview.util.*;

public class E06_LinkedListTest {
```

```
  LinkedList<String> testObject = new LinkedList<String>();
  @Test void initialization() {
    assert testObject.isEmpty();
  }
  @Test void _contains() {
    testObject.add("one");
    assert testObject.contains("one");
  }
  @Test void _remove() {
    testObject.add("one");
    testObject.remove("one");
    assert testObject.isEmpty();
  }
  public static void main(String[] args) throws Exception {
    OSExecute.command("java " +
      " net.mindview.atunit.AtUnit E06_LinkedListTest");
  }
} /* Output:
annotations.E06_LinkedListTest
  . initialization
  . _remove
  . _contains
OK (3 tests)
*///:~
```

Exercise 7

```
//: annotations/E07_LinkedListTest2.java
/****************** Exercise 07 *****************
 * Use the inheritance approach to modify Exercise 6.
 ***********************************************/
package annotations;
import java.util.*;
import net.mindview.atunit.*;
import net.mindview.util.*;

public class E07_LinkedListTest2
extends LinkedList<String> {
  @Test void initialization() {
    assert isEmpty();
  }
  @Test void _contains() {
    add("one");
    assert contains("one");
  }
  @Test void _remove() {
```

```
    add("one");
    remove("one");
    assert isEmpty();
  }
  public static void main(String[] args) throws Exception {
    OSExecute.command("java " +
      " net.mindview.atunit.AtUnit E07_LinkedListTest2");
  }
} /* Output:
annotations.E07_LinkedListTest2
  . initialization
  . _remove
  . _contains
OK (3 tests)
*///:~
```

Exercise 8

```
//: annotations/E08_TestPrivateMethod.java
/****************** Exercise 08 *****************
 * Add a non-private @TestProperty method (described
 * above) to create a class with a private method.
 * Call this method in your test code.
 ***********************************************/
package annotations;
import net.mindview.atunit.*;
import net.mindview.util.*;

class PrivateMethod {
  private boolean hidden() { return true; }
  @TestProperty boolean visible() { return hidden(); }
}

public class E08_TestPrivateMethod extends PrivateMethod {
  @Test void _hidden() { assert visible(); }
  public static void main(String[] args) {
    OSExecute.command("java " +
      " net.mindview.atunit.AtUnit E08_TestPrivateMethod");
  }
} /* Output:
annotations.E08_TestPrivateMethod
  . _hidden
OK (1 tests)
*///:~
```

Exercise 9

```
//: annotations/E09_HashMapTest.java
/****************** Exercise 09 *****************
 * Write basic @Unit tests for HashMap.
 ************************************************/
package annotations;
import java.util.*;
import net.mindview.atunit.*;
import net.mindview.util.*;

public class E09_HashMapTest {
  HashMap<String,Integer> testObject =
    new HashMap<String,Integer>();
  @Test void initialization() {
    assert testObject.isEmpty();
  }
  @Test void _containsValue() {
    testObject.put("one", 1);
    assert testObject.containsValue(1);
  }
  @Test void _containsKey() {
    testObject.put("one", 1);
    assert testObject.containsKey("one");
  }
  @Test void _remove() {
    testObject.put("one", 1);
    testObject.remove("one");
    assert testObject.isEmpty();
  }
  @Test void _get() {
    testObject.put("one", 1);
    assert testObject.get("one") == 1;
  }
  @Test void _size() {
    testObject.put("one", 1);
    testObject.put("two", 2);
    assert testObject.size() == 2;
  }
  @Test void _clear() {
    testObject.put("one", 1);
    assert !testObject.isEmpty();
    testObject.clear();
    assert testObject.isEmpty();
  }
  public static void main(String[] args) throws Exception {
```

```
    OSExecute.command("java " +
      " net.mindview.atunit.AtUnit E09_HashMapTest");
  }
} /* Output:
annotations.E09_HashMapTest
  . initialization
  . _containsValue
  . _containsKey
  . _remove
  . _size
  . _clear
  . _get
OK (7 tests)
*///:~
```

Exercise 10

```
//: annotations/E10_TrafficLightTest.java
/****************** Exercise 10 *****************
 * Select an example from elsewhere in the book
 * and add @Unit tests.
 ***********************************************/
package annotations;
import net.mindview.atunit.*;
import net.mindview.util.*;

enum Signal { GREEN, YELLOW, RED, }

class TrafficLight {
  Signal color = Signal.RED;
  public void change() {
    switch(color) {
      // Note that you don't have to say Signal.RED
      // in the case statement:
      case RED:    color = Signal.GREEN;
                   break;
      case GREEN:  color = Signal.YELLOW;
                   break;
      case YELLOW: color = Signal.RED;
                   break;
    }
  }
  public String toString() {
    return "The traffic light is " + color;
  }
}
```

```
public class E10_TrafficLightTest {
  TrafficLight testObject = new TrafficLight();
  @Test void initialization() {
    assert testObject.toString().equals(str(Signal.RED));
  }
  @Test void _change() {
    testObject.change();
    assert testObject.toString().equals(str(Signal.GREEN));
    testObject.change();
    assert testObject.toString().equals(str(Signal.YELLOW));
    testObject.change();
    assert testObject.toString().equals(str(Signal.RED));
  }
  @TestProperty private static String str(Signal s) {
    return "The traffic light is " + s;
  }
  public static void main(String[] args) throws Exception {
    OSExecute.command("java " +
      " net.mindview.atunit.AtUnit E10_TrafficLightTest");
  }
} /* Output:
annotations.E10_TrafficLightTest
  . initialization
  . _change
OK (2 tests)
*///:~
```

We use the **enumerated/TrafficLight.java** program (from the *Enumerated Types* chapter) as an example.

Exercise 11

```
//: annotations/E11_AtUnit2.java
// {RunByHand}
/****************** Exercise 11 *****************
 * Add an @TestNote annotation to @Unit, so that the
 * accompanying note is simply displayed during testing.
 ************************************************/
package annotations;
import java.lang.reflect.*;
import java.io.*;
import java.util.*;
import net.mindview.util.*;
import static net.mindview.util.Print.*;
import net.mindview.atunit.*;
```

```
public class E11_AtUnit2 implements ProcessFiles.Strategy {
  static Class<?> testClass;
  static List<String> failedTests= new ArrayList<String>();
  static long testsRun = 0;
  static long failures = 0;
  public static void main(String[] args) throws Exception {
    ClassLoader.getSystemClassLoader()
      .setDefaultAssertionStatus(true); // Enable asserts
    new ProcessFiles(
      new E11_AtUnit2(), "class").start(args);
    if(failures == 0)
      print("OK (" + testsRun + " tests)");
    else {
      print("(" + testsRun + " tests)");
      print("\n>>> " + failures + " FAILURE" +
        (failures > 1 ? "S" : "") + " <<<");
      for(String failed : failedTests)
        print("  " + failed);
    }
  }
  public void process(File cFile) {
    try {
      String cName = ClassNameFinder.thisClass(
        BinaryFile.read(cFile));
      if(!cName.contains("."))
        return; // Ignore unpackaged classes
      testClass = Class.forName(cName);
    } catch(Exception e) {
      throw new RuntimeException(e);
    }
    TestMethods testMethods = new TestMethods();
    HashMap<Method,String> testNotes =
      new HashMap<Method,String>();
    Method creator = null;
    Method cleanup = null;
    for(Method m : testClass.getDeclaredMethods()) {
      testMethods.addIfTestMethod(m);
      // Check to see whether @TestNote is present, and used
      // correctly.
      TestNote testNote;
      if((testNote = m.getAnnotation(TestNote.class)) !=
         null) {
        if(m.getAnnotation(Test.class) == null)
          throw new RuntimeException("@TestNote method" +
          " must be a @Test method, too");
        testNotes.put(m, testNote.value());
```

```
      }
      if(creator == null)
        creator = checkForCreatorMethod(m);
      if(cleanup == null)
        cleanup = checkForCleanupMethod(m);
    }
    if(testMethods.size() > 0) {
      if(creator == null)
        try {
          if(!Modifier.isPublic(testClass
             .getDeclaredConstructor().getModifiers())) {
            print("Error: " + testClass +
              " default constructor must be public");
            System.exit(1);
          }
        } catch(NoSuchMethodException e) {
          // Synthesized default constructor; OK
        }
      print(testClass.getName());
    }
    for(Method m : testMethods) {
      printnb("  . " + m.getName() + " ");
      if(testNotes.containsKey(m))
        printnb(" : " + testNotes.get(m) + " ");
      try {
        Object testObject = createTestObject(creator);
        boolean success = false;
        try {
          if(m.getReturnType().equals(boolean.class))
            success = (Boolean)m.invoke(testObject);
          else {
            m.invoke(testObject);
            success = true; // If no assert fails
          }
        } catch(InvocationTargetException e) {
          // Actual exception is inside e:
          print(e.getCause());
        }
        print(success ? "" : "(failed)");
        testsRun++;
        if(!success) {
          failures++;
          failedTests.add(testClass.getName() +
            ": " + m.getName());
        }
        if(cleanup != null)
          cleanup.invoke(testObject, testObject);
```

```
      } catch(Exception e) {
        throw new RuntimeException(e);
      }
    }
  }
  static class TestMethods extends ArrayList<Method> {
    void addIfTestMethod(Method m) {
      if(m.getAnnotation(Test.class) == null)
        return;
      if(!(m.getReturnType().equals(boolean.class) ||
          m.getReturnType().equals(void.class)))
        throw new RuntimeException("@Test method" +
          " must return boolean or void");
      m.setAccessible(true); // In case it's private, etc.
      add(m);
    }
  }
  private static Method checkForCreatorMethod(Method m) {
    if(m.getAnnotation(TestObjectCreate.class) == null)
      return null;
    if(!m.getReturnType().equals(testClass))
      throw new RuntimeException("@TestObjectCreate " +
        "must return instance of Class to be tested");
    if((m.getModifiers() &
        java.lang.reflect.Modifier.STATIC) < 1)
      throw new RuntimeException("@TestObjectCreate " +
        "must be static.");
    m.setAccessible(true);
    return m;
  }
  private static Method checkForCleanupMethod(Method m) {
    if(m.getAnnotation(TestObjectCleanup.class) == null)
      return null;
    if(!m.getReturnType().equals(void.class))
      throw new RuntimeException("@TestObjectCleanup " +
        "must return void");
    if((m.getModifiers() &
        java.lang.reflect.Modifier.STATIC) < 1)
      throw new RuntimeException("@TestObjectCleanup " +
        "must be static.");
    if(m.getParameterTypes().length == 0 ||
       m.getParameterTypes()[0] != testClass)
      throw new RuntimeException("@TestObjectCleanup " +
        "must take an argument of the tested type.");
    m.setAccessible(true);
    return m;
  }
```

```
  private static Object createTestObject(Method creator) {
    if(creator != null) {
      try {
        return creator.invoke(testClass);
      } catch(Exception e) {
        throw new RuntimeException("Couldn't run " +
          "@TestObject (creator) method.");
      }
    }
    // Use the default constructor:
    try {
      return testClass.newInstance();
    } catch(Exception e) {
      throw new RuntimeException("Couldn't create a " +
        "test object. Try using a @TestObject method.");
    }
  }
} ///:~
```

```
//: annotations/TestNote.java
// The @Unit @TestNote tag.
package annotations;
import java.lang.annotation.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface TestNote {
  String value() default "";
} ///:~
```

Always pair the **@TestNote** annotation (if one is specified) with its corresponding **@Test**. Here's a demonstration:

```
//: annotations/HashSetCommentedTest.java
package annotations;
import java.util.*;
import net.mindview.atunit.*;
import net.mindview.util.*;

public class HashSetCommentedTest {
  HashSet<String> testObject = new HashSet<String>();
  @TestNote("Tests if the new HashSet is empty ")
  @Test void initialization() {
    assert testObject.isEmpty();
  }
  @TestNote("Tests the HashSet.contains(Object o) method")
  @Test void _contains() {
    testObject.add("one");
```

```
    assert testObject.contains("one");
  }
  @TestNote("Tests the HashSet.remove(Object o) method")
  @Test void _remove() {
    testObject.add("one");
    testObject.remove("one");
    assert testObject.isEmpty();
  }
  public static void main(String[] args) throws Exception {
    OSExecute.command("java " +
      " annotations.E11_AtUnit2 HashSetCommentedTest");
  }
} /* Output:
annotations.HashSetCommentedTest
  . initialization  : Tests if the new HashSet is empty
  . _remove  : Tests the HashSet.remove(Object o) method
  . _contains  : Tests the HashSet.contains(Object o) method
OK (3 tests)
*///:~
```

Concurrency

Exercise 1

```
//: concurrency/E01_Runnable.java
/****************** Exercise 1 *****************
 * Implement a Runnable. Inside run(), print a
 * message, and then call yield(). Repeat this
 * three times, and then return from run(). Put
 * a startup message in the constructor and a
 * shutdown message when the task terminates. Create
 * a number of these tasks and drive them using
 * threads.
 ***********************************************/
package concurrency;

class Printer implements Runnable {
  private static int taskCount;
  private final int id = taskCount++;
  Printer() {
    System.out.println("Printer started, ID = " + id);
  }
  public void run() {
    System.out.println("Stage 1..., ID = " + id);
    Thread.yield();
    System.out.println("Stage 2..., ID = " + id);
    Thread.yield();
    System.out.println("Stage 3..., ID = " + id);
    Thread.yield();
    System.out.println("Printer ended, ID = " + id);
  }
}

public class E01_Runnable {
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++)
      new Thread(new Printer()).start();
  }
} /* Output: (Sample)
Printer started, ID = 0
Printer started, ID = 1
Printer started, ID = 2
```

```
Printer started, ID = 3
Printer started, ID = 4
Stage 1..., ID = 0
Stage 1..., ID = 1
Stage 1..., ID = 2
Stage 1..., ID = 3
Stage 1..., ID = 4
Stage 2..., ID = 0
Stage 2..., ID = 1
Stage 2..., ID = 2
Stage 2..., ID = 3
Stage 2..., ID = 4
Stage 3..., ID = 0
Stage 3..., ID = 1
Stage 3..., ID = 2
Stage 3..., ID = 3
Stage 3..., ID = 4
Printer ended, ID = 0
Printer ended, ID = 1
Printer ended, ID = 2
Printer ended, ID = 3
Printer ended, ID = 4
*///:~
```

Exercise 2

```
//: concurrency/E02_Fibonacci.java
/****************** Exercise 2 *****************
 * Following the form of generics/Fibonacci.java,
 * create a task that produces a sequence of n
 * Fibonacci numbers, where n is provided to the
 * constructor of the task. Create a number of these
 * tasks and drive them using threads.
 ***********************************************/
package concurrency;
import java.util.*;
import net.mindview.util.*;

class Fibonacci implements Generator<Integer>, Runnable {
  private int count;
  private final int n;
  public Fibonacci(int n) { this.n = n; }
  public Integer next() { return fib(count++); }
  private int fib(int n) {
    if(n < 2) return 1;
    return fib(n-2) + fib(n-1);
```

```
  }
  public void run() {
    Integer[] sequence = new Integer[n];
    for(int i = 0; i < n; i++)
      sequence[i] = next();
    System.out.println(
      "Seq. of " + n + ": " + Arrays.toString(sequence));
  }
}

public class E02_Fibonacci {
  public static void main(String[] args) {
    for(int i = 1; i <= 5; i++)
      new Thread(new Fibonacci(i)).start();
  }
} /* Output: (Sample)
Seq. of 2: [1, 1]
Seq. of 3: [1, 1, 2]
Seq. of 4: [1, 1, 2, 3]
Seq. of 5: [1, 1, 2, 3, 5]
Seq. of 1: [1]
*///:~
```

Exercise 3

```
//: concurrency/E03_Runnable2.java
/****************** Exercise 3 *****************
 * Repeat Exercise 1 using the different types of
 * executors shown in this section.
 ***********************************************/
package concurrency;
import java.util.concurrent.*;

public class E03_Runnable2 {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < 5; i++)
      exec.execute(new Printer());
    Thread.yield();
    exec.shutdown();
    exec = Executors.newFixedThreadPool(5);
    for(int i = 0; i < 5; i++)
      exec.execute(new Printer());
    Thread.yield();
    exec.shutdown();
    exec = Executors.newSingleThreadExecutor();
```

```
    for(int i = 0; i < 5; i++)
      exec.execute(new Printer());
    Thread.yield();
    exec.shutdown();
  }
} /* Output: (Sample)
Printer started, ID = 0
Printer started, ID = 1
Printer started, ID = 2
Printer started, ID = 3
Printer started, ID = 4
Stage 1..., ID = 0
Stage 1..., ID = 1
Stage 1..., ID = 2
Stage 1..., ID = 3
Stage 1..., ID = 4
...
Stage 3..., ID = 11
Printer ended, ID = 11
Stage 1..., ID = 12
Stage 2..., ID = 12
Stage 3..., ID = 12
Printer ended, ID = 12
Stage 1..., ID = 13
Stage 2..., ID = 13
Stage 3..., ID = 13
Printer ended, ID = 13
Stage 1..., ID = 14
Stage 2..., ID = 14
Stage 3..., ID = 14
Printer ended, ID = 14
*///:~
```

The call to **shutdown( )** is an asynchronous method call to initiate an orderly shutdown procedure. The JDK states: *"After being shut down, the executor will eventually terminate, at which point no tasks are actively executing, no tasks are awaiting execution, and no new tasks can be submitted."*

Exercise 4

```
//: concurrency/E04_Fibonacci2.java
/****************** Exercise 4 *****************
 * Repeat Exercise 2 using the different types of
 * executors shown in this section.
 ***********************************************/
package concurrency;
```

```
import java.util.concurrent.*;

public class E04_Fibonacci2 {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 1; i <= 5; i++)
      exec.execute(new Fibonacci(i));
    Thread.yield();
    exec.shutdown();
    exec = Executors.newFixedThreadPool(5);
    for(int i = 1; i <= 5; i++)
      exec.execute(new Fibonacci(i));
    Thread.yield();
    exec.shutdown();
    exec = Executors.newSingleThreadExecutor();
    for(int i = 1; i <= 5; i++)
      exec.execute(new Fibonacci(i));
    Thread.yield();
    exec.shutdown();
  }
} /* Output: (Sample)
Seq. of 1: [1]
Seq. of 2: [1, 1]
Seq. of 3: [1, 1, 2]
Seq. of 4: [1, 1, 2, 3]
Seq. of 5: [1, 1, 2, 3, 5]
Seq. of 1: [1]
Seq. of 2: [1, 1]
Seq. of 3: [1, 1, 2]
Seq. of 4: [1, 1, 2, 3]
Seq. of 5: [1, 1, 2, 3, 5]
Seq. of 1: [1]
Seq. of 2: [1, 1]
Seq. of 3: [1, 1, 2]
Seq. of 4: [1, 1, 2, 3]
Seq. of 5: [1, 1, 2, 3, 5]
*///:~
```

Exercise 5

```
//: concurrency/E05_FibonacciSum.java
/****************** Exercise 5 *****************
 * Modify Exercise 2 so that the task is a Callable
 * that sums the values of all the Fibonacci numbers.
 * Create several tasks and display the results.
 ***********************************************/
```

```
package concurrency;
import java.util.*;
import java.util.concurrent.*;
import net.mindview.util.*;

class FibonacciSum
implements Generator<Integer>, Callable<Integer> {
  private int count;
  private final int n;
  public FibonacciSum(int n) { this.n = n; }
  public Integer next() { return fib(count++); }
  private int fib(int n) {
    if(n < 2) return 1;
    return fib(n-2) + fib(n-1);
  }
  public Integer call() {
    int sum = 0;
    for(int i = 0; i < n; i++)
      sum += next();
    return sum;
  }
}

public class E05_FibonacciSum {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool();
    ArrayList<Future<Integer>> results =
      new ArrayList<Future<Integer>>();
    for(int i = 1; i <= 5; i++)
      results.add(exec.submit(new FibonacciSum(i)));
    Thread.yield();
    exec.shutdown();
    for(Future<Integer> fi : results)
      try {
        System.out.println(fi.get());
      } catch(Exception e) {
        e.printStackTrace();
      }
  }
} /* Output:
1
2
4
7
12
*///:~
```

Exercise 6

```
//: concurrency/E06_SleepingTask2.java
// {Args: 5}
/****************** Exercise 6 *****************
 * Create a task that sleeps for a random amount
 * of time between 1 and 10 seconds, then displays
 * its sleep time and exits. Create and run a quantity
 * (given on the command line) of these tasks.
 ***********************************************/
package concurrency;
import java.util.*;
import java.util.concurrent.*;

class SleepingTask2 implements Runnable {
  private static Random rnd = new Random();
  private final int sleep_time = rnd.nextInt(10) + 1;
  public void run() {
    try {
      TimeUnit.SECONDS.sleep(sleep_time);
    } catch(InterruptedException e) {
      System.err.println("Interrupted: " + e);
    }
    System.out.println(sleep_time);
  }
}

public class E06_SleepingTask2 {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool();
    if(args.length != 1) {
      System.err.println(
        "Provide the quantity of tasks to run");
      return;
    }
    for(int i = 0; i < Integer.parseInt(args[0]); i++)
      exec.execute(new SleepingTask2());
    Thread.yield();
    exec.shutdown();
  }
} /* Output: (Sample)
1
1
2
6
10
```

```
*///:~
```

Exercise 7

```
//: concurrency/E07_Daemons2.java
/****************** Exercise 7 *****************
 * Experiment with different sleep times in
 * Daemons.java to see what happens.
 *********************************************/
package concurrency;
import java.util.concurrent.*;
import static net.mindview.util.Print.*;

class Daemon2 implements Runnable {
  private Thread[] t = new Thread[10];
  public void run() {
    for(int i = 0; i < t.length; i++) {
      t[i] = new Thread(new DaemonSpawn());
      t[i].start();
      printnb("DaemonSpawn " + i + " started, ");
    }
    try {
      // To better see the effect of altering main
      // application thread's sleep time.
      TimeUnit.MILLISECONDS.sleep(100);
    } catch(InterruptedException e) { /* Ignore */ }
    for(int i = 0; i < t.length; i++)
      printnb("t[" + i + "].isDaemon() = " +
        t[i].isDaemon() + ", ");
    while(true)
      Thread.yield();
  }
}

public class E07_Daemons2 {
  public static void main(String[] args) throws Exception {
    if(args.length != 1) {
      System.err.println(
        "Sleep time needs to be provided in msecs");
      return;
    }
    int sleep_time = Integer.parseInt(args[0]);
    Thread d = new Thread(new Daemon2());
    d.setDaemon(true);
    d.start();
    printnb("d.isDaemon() = " + d.isDaemon() + ", ");
```

```
    TimeUnit.MILLISECONDS.sleep(sleep_time);
  }
} /* (Execute to see output) *///:~
```

Our program accepts the sleep time (in milliseconds) as a command line argument. By providing different values (usually much less than 1 second), you see that the chief daemon thread (running the **Daemon2** task) abruptly terminates before even reaching the endless loop.

Exercise 8

```
//: concurrency/E08_MoreBasicDaemonThreads.java
/****************** Exercise 8 *****************
 * Modify MoreBasicThreads.java so that all the
 * threads are daemon threads, and verify that the
 * program ends as soon as main() is able to exit.
 ***********************************************/
package concurrency;

public class E08_MoreBasicDaemonThreads {
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++) {
      Thread t = new Thread(new LiftOff());
      t.setDaemon(true);
      t.start();
    }
    System.out.println("Waiting for LiftOff");
  }
} /* Output: (Sample)
Waiting for LiftOff
*///:~
```

Run the program and you see that the daemon threads are unable to complete their countdowns before the program terminates.

Exercise 9

```
//: concurrency/E09_SimplePrioritiesTF.java
/****************** Exercise 9 *****************
 * Modify SimplePriorities.java so that a custom
 * ThreadFactory sets the priorities of the threads.
 ***********************************************/
package concurrency;
import java.util.concurrent.*;
```

```
class SimplePriorities2 implements Runnable {
  private int countDown = 5;
  private volatile double d; // No optimization
  public String toString() {
    return Thread.currentThread() + ": " + countDown;
  }
  public void run() {
    for(;;) {
      // An expensive, interruptable operation:
      for(int i = 1; i < 100000; i++) {
        d += (Math.PI + Math.E) / (double)i;
        if(i % 1000 == 0)
          Thread.yield();
      }
      System.out.println(this);
      if(--countDown == 0) return;
    }
  }
}

class PriorityThreadFactory implements ThreadFactory {
  private final int priority;
  PriorityThreadFactory(int priority) {
    this.priority = priority;
  }
  public Thread newThread(Runnable r) {
    Thread t = new Thread(r);
    t.setPriority(priority);
    return t;
  }
}

public class E09_SimplePrioritiesTF {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool(
      new PriorityThreadFactory(Thread.MIN_PRIORITY));
    for(int i = 0; i < 5; i++)
      exec.execute(new SimplePriorities2());
    Thread.yield();
    exec.shutdown();
    exec = Executors.newCachedThreadPool(
      new PriorityThreadFactory(Thread.MAX_PRIORITY));
    exec.execute(new SimplePriorities2());
    Thread.yield();
    exec.shutdown();
  }
} /* Output: (Sample)
```

```
Thread[Thread-0,1,main]: 5
Thread[Thread-1,1,main]: 5
Thread[Thread-2,1,main]: 5
Thread[Thread-3,1,main]: 5
Thread[Thread-4,1,main]: 5
Thread[Thread-5,10,main]: 5
Thread[Thread-5,10,main]: 4
Thread[Thread-5,10,main]: 3
Thread[Thread-5,10,main]: 2
Thread[Thread-5,10,main]: 1
Thread[Thread-0,1,main]: 4
Thread[Thread-1,1,main]: 4
Thread[Thread-2,1,main]: 4
Thread[Thread-3,1,main]: 4
Thread[Thread-4,1,main]: 4
Thread[Thread-0,1,main]: 3
Thread[Thread-1,1,main]: 3
Thread[Thread-2,1,main]: 3
Thread[Thread-3,1,main]: 3
Thread[Thread-4,1,main]: 3
Thread[Thread-0,1,main]: 2
Thread[Thread-1,1,main]: 2
Thread[Thread-2,1,main]: 2
Thread[Thread-3,1,main]: 2
Thread[Thread-4,1,main]: 2
Thread[Thread-0,1,main]: 1
Thread[Thread-1,1,main]: 1
Thread[Thread-2,1,main]: 1
Thread[Thread-3,1,main]: 1
Thread[Thread-4,1,main]: 1
*///:~
```

Exercise 10

```
//: concurrency/E10_FibonacciSum2.java
/****************** Exercise 10 *****************
 * Modify Exercise 5 following the example of the
 * ThreadMethod class, so that runTask() takes an
 * argument of the number of Fibonacci numbers to sum,
 * and each time you call runTask() it returns
 * the Future produced by the call to submit().
 ************************************************/
package concurrency;
import java.util.*;
import java.util.concurrent.*;
```

```
class FibonacciSum2 {
  private static ExecutorService exec;
  private static int fib(int n) {
    if(n < 2) return 1;
    return fib(n-2) + fib(n-1);
  }
  public static synchronized
  Future<Integer> runTask(final int n) {
    assert exec != null;
    return exec.submit(new Callable<Integer>() {
      public Integer call() {
        int sum = 0;
        for(int i = 0; i < n; i++)
          sum += fib(i);
        return sum;
      }
    });
  }
  public static synchronized void init() {
    if(exec == null)
      exec = Executors.newCachedThreadPool();
  }
  public static synchronized void shutdown() {
    if(exec != null)
      exec.shutdown();
    exec = null;
  }
}

public class E10_FibonacciSum2 {
  public static void main(String[] args) {
    ArrayList<Future<Integer>> results =
      new ArrayList<Future<Integer>>();
    FibonacciSum2.init();
    for(int i = 1; i <= 5; i++)
      results.add(FibonacciSum2.runTask(i));
    Thread.yield();
    FibonacciSum2.shutdown();
    for(Future<Integer> fi : results)
      try {
        System.out.println(fi.get());
      } catch(Exception e) {
        e.printStackTrace();
      }
  }
} /* Output:
1
```

```
2
4
7
12
*///:~
```

Call **FibonacciSum2.init( )** at the beginning of each session, and the **FibonacciSum2.shutdown( )** static method at the end, after a dedicated thread finishes with **FibonacciSum2**. Notice the use of the **synchronized** keyword.

Exercise 11

```
//: concurrency/E11_RaceCondition.java
// {RunByHand}
/****************** Exercise 11 *****************
 * Create a class containing two data fields, and a
 * method that manipulates those fields in a multistep
 * process so that, during the execution of that method,
 * those fields are in an "improper state" (according to
 * some definition that you establish). Add methods to
 * read the fields, and create multiple threads to call
 * the various methods and show that the data is visible
 * in its "improper state." Fix the problem using the
 * synchronized keyword.
 *********************************************/
package concurrency;
import java.util.*;
import java.util.concurrent.*;

// The conditions, which always must hold are the following:
// 1. If the tank is EMPTY then the current_load == 0
// 2. If the tank is LOADED then the current_load >= 0
class Tank {
  enum State { EMPTY, LOADED };
  private State state = State.EMPTY;
  private int current_load = 0;
  public void validate() {
    if((state == State.EMPTY && current_load != 0) ||
       (state == State.LOADED && current_load == 0))
      throw new IllegalStateException();
  }
  public void fill() {
    state = State.LOADED;
    Thread.yield();     // Cause failure faster
    current_load = 10;  // Arbitrary value
```

```
  }
  public void drain() {
    state = State.EMPTY;
    Thread.yield();
    current_load = 0;
  }
}

class ConsistencyChecker implements Runnable {
  private static Random rnd = new Random();
  private Tank tank;
  ConsistencyChecker(Tank tank) { this.tank = tank; }
  public void run() {
    for(;;) {
      // Decide whether to fill or drain the tank
      if(rnd.nextBoolean())
        tank.fill();
      else
        tank.drain();
      tank.validate();
    }
  }
}

public class E11_RaceCondition {
  public static void main(String[] args) {
    Thread.setDefaultUncaughtExceptionHandler(
      new MyUncaughtExceptionHandler());
    System.out.println("Press Control-C to exit");
    ExecutorService exec = Executors.newCachedThreadPool();
    Tank tank = new Tank();
    for(int i = 0; i < 10; i++)
      exec.execute(new ConsistencyChecker(tank));
    Thread.yield();
    exec.shutdown();
  }
} ///:~
```

This program throws an **IllegalStateException** when it reaches an "improper state." We use a thread-safe variant of **Tank**; press Control-C to exit the program.

```
//: concurrency/E11_RaceConditionB.java
// {RunByHand}
package concurrency;
import java.util.concurrent.*;
```

```
class SafeTank extends Tank {
  public synchronized void validate() { super.validate(); }
  public synchronized void fill() { super.fill(); }
  public synchronized void drain() { super.drain(); }
}

public class E11_RaceConditionB {
  public static void main(String[] args) {
    Thread.setDefaultUncaughtExceptionHandler(
      new MyUncaughtExceptionHandler());
    System.out.println("Press Control-C to exit");
    ExecutorService exec = Executors.newCachedThreadPool();
    SafeTank tank = new SafeTank();
    for(int i = 0; i < 10; i++)
      exec.execute(new ConsistencyChecker(tank));
    Thread.yield();
    exec.shutdown();
  }
} ///:~
```

Exercise 12

```
//: concurrency/E12_AtomicityTest2.java
// {RunByHand}
/****************** Exercise 12 *****************
 * Repair AtomicityTest.java using the synchronized
 * keyword. Can you demonstrate that it is now
 * correct?
 ************************************************/
package concurrency;
import java.util.concurrent.*;

class AtomicityTest2 implements Runnable {
  private int i;
  public synchronized int getValue() { return i; }
  private synchronized void evenIncrement() { i++;  i++; }
  public void run() {
    while(true)
      evenIncrement();
  }
}

public class E12_AtomicityTest2 {
  public static void main(String[] args) {
    System.out.println("Press Control-C to exit");
    ExecutorService exec = Executors.newCachedThreadPool();
```

```
    AtomicityTest2 at = new AtomicityTest2();
    exec.execute(at);
    while(true) {
      int val = at.getValue();
      if(val % 2 != 0) {
        System.out.println(val);
        System.exit(0);
      }
    }
  }
} ///:~
```

The program now runs indefinitely, which might mean that it's working correctly. But when you assess your program based on testing alone, you risk shipping incorrect code to the customer. Formal verification is necessary for concurrent programming.

However, using some semi-formal verification, we can reasonably examine the program. Both methods are synchronized, and no race condition can occur. While one thread changes the state of an object, the other waits. Writes by one thread are always visible to the other. Accesses to the shared resource are fully serialized, with no visibility problems.

Exercise 13

```
//: concurrency/E13_SerialNumberChecker2.java
// {Args: 4}
/****************** Exercise 13 *****************
 * Repair SerialNumberChecker.java using the
 * synchronized keyword. Can you demonstrate that
 * it is now correct?
 ***********************************************/
package concurrency;
import java.util.concurrent.*;

class SerialNumberGenerator2 {
  private static int serialNumber;
  public synchronized static int nextSerialNumber() {
    return serialNumber++;
  }
}

public class E13_SerialNumberChecker2 {
  private static final int SIZE = 10;
  private static CircularSet serials =
    new CircularSet(1000);
```

```
  private static ExecutorService exec =
    Executors.newCachedThreadPool();
  static class SerialChecker implements Runnable {
    public void run() {
      while(true) {
        int serial =
          SerialNumberGenerator2.nextSerialNumber();
        if(serials.contains(serial)) {
          System.out.println("Duplicate: " + serial);
          System.exit(0);
        }
        serials.add(serial);
      }
    }
  }
  public static void main(String[] args) throws Exception {
    for(int i = 0; i < SIZE; i++)
      exec.execute(new SerialChecker());
    if(args.length > 0) {
      TimeUnit.SECONDS.sleep(new Integer(args[0]));
      System.out.println("No duplicates detected");
      System.exit(0);
    } else {
      System.err.println("Provide a sleep time in sec.");
      System.exit(1);
    }
  }
} /* Output:
No duplicates detected
*///:~
```

Again, notice that you can safely remove the **volatile** keyword from the code. Synchronization flushes main memory, so if we guard a field (in our case **serialNumber**) with **synchronized** methods or blocks, it does not need to be **volatile**.

Exercise 14

```
//: concurrency/E14_ManyTimers.java
// {Args: 100}
/****************** Exercise 14 ************************
 * Demonstrate that java.util.Timer scales to large numbers
 * by creating a program that generates many Timer objects
 * that perform some simple task when the timeout completes.
 ***************************************************/
package concurrency;
```

```
import java.util.*;
import java.util.concurrent.*;

public class E14_ManyTimers {
  public static void main(String[] args) throws Exception {
    if(args.length < 1) {
      System.err.println(
        "Usage: java E14_ManyTimers <num of timers>");
    }
    int numOfTimers = Integer.parseInt(args[0]);
    for(int i = 0; i < numOfTimers; i++) {
      new Timer().schedule(new TimerTask() {
        public void run() {
          System.out.println(System.currentTimeMillis());
        }
      }, numOfTimers - i);
    }
    // Wait for timers to expire
    TimeUnit.MILLISECONDS.sleep(2 * numOfTimers);
    System.exit(0);
  }
} /* Output: (Sample)
1133967112886
1133967112886
1133967112886
1133967112886
1133967112886
1133967112896
1133967112896
1133967112896
...
1133967112966
1133967112966
1133967112966
1133967112966
*///:~
```

Exercise 15

```
//: concurrency/E15_SyncObject.java
/****************** Exercise 15 ************************
 * Create a class with three methods containing critical
 * sections that all synchronize on the same object. Create
 * multiple tasks to demonstrate that only one of these
 * methods can run at a time. Now modify the methods so
 * that each one synchronizes on a different object and
```

```
 * show that all three methods can be running at once.
 *****************************************************/
package concurrency;
import static net.mindview.util.Print.*;

class SingleSynch {
  public synchronized void f() {
    for(int i = 0; i < 5; i++) {
      print("f()");
      Thread.yield();
    }
  }
  public synchronized void g() {
    for(int i = 0; i < 5; i++) {
      print("g()");
      Thread.yield();
    }
  }
  public synchronized void h() {
    for(int i = 0; i < 5; i++) {
      print("h()");
      Thread.yield();
    }
  }
}

class TripleSynch {
  private Object syncObjectG = new Object();
  private Object syncObjectH = new Object();
  public synchronized void f() {
    for(int i = 0; i < 5; i++) {
      print("f()");
      Thread.yield();
    }
  }
  public void g() {
    synchronized(syncObjectG) {
      for(int i = 0; i < 5; i++) {
        print("g()");
        Thread.yield();
      }
    }
  }
  public void h() {
    synchronized(syncObjectH) {
      for(int i = 0; i < 5; i++) {
        print("h()");
```

```
        Thread.yield();
      }
    }
  }
}

public class E15_SyncObject {
  public static void main(String[] args) throws Exception {
    final SingleSynch singleSync = new SingleSynch();
    final TripleSynch tripleSync = new TripleSynch();
    print("Test SingleSynch...");
    Thread t1 = new Thread() {
      public void run() {
        singleSync.f();
      }
    };
    t1.start();
    Thread t2 = new Thread() {
      public void run() {
        singleSync.g();
      }
    };
    t2.start();
    singleSync.h();
    t1.join();  // Wait for t1 to finish its work
    t2.join();  // Wait for t2 to finish its work
    print("Test TripleSynch...");
    new Thread() {
      public void run() {
        tripleSync.f();
      }
    }.start();
    new Thread() {
      public void run() {
        tripleSync.g();
      }
    }.start();
    tripleSync.h();
  }
} /* Output: (Sample)
Test SingleSynch...
h()
h()
h()
h()
h()
f()
```

```
f()
f()
f()
f()
g()
g()
g()
g()
g()
Test TripleSynch...
h()
f()
g()
h()
f()
g()
h()
f()
g()
h()
f()
g()
h()
f()
g()
*///:~
```

The output shows that all methods of **TripleSynch** are running at the same time; none are blocked by the synchronization of another. This is clearly not the case with **SingleSynch**.

Exercise 16

```
//: concurrency/E16_ExplicitSyncObject.java
/****************** Exercise 16 ************************
 * Modify Exercise 15 to use explicit Lock objects.
 ***************************************************/
package concurrency;
import java.util.concurrent.locks.*;
import static net.mindview.util.Print.*;

class ExplicitSingleSynch {
  private Lock lock = new ReentrantLock();
  public void f() {
    lock.lock();
    try {
```

```
      for(int i = 0; i < 5; i++) {
        print("f()");
        Thread.yield();
      }
    } finally {
      lock.unlock();
    }
  }
  public void g() {
    lock.lock();
    try {
      for(int i = 0; i < 5; i++) {
        print("g()");
        Thread.yield();
      }
    } finally {
      lock.unlock();
    }
  }
  public void h() {
    lock.lock();
    try {
      for(int i = 0; i < 5; i++) {
        print("h()");
        Thread.yield();
      }
    } finally {
      lock.unlock();
    }
  }
}

class ExplicitTripleSynch {
  private Lock lockF = new ReentrantLock();
  private Lock lockG = new ReentrantLock();
  private Lock lockH = new ReentrantLock();
  public void f() {
    lockF.lock();
    try {
      for(int i = 0; i < 5; i++) {
        print("f()");
        Thread.yield();
      }
    } finally {
      lockF.unlock();
    }
  }
```

```
  public void g() {
    lockG.lock();
    try {
      for(int i = 0; i < 5; i++) {
        print("g()");
        Thread.yield();
      }
    } finally {
      lockG.unlock();
    }
  }
  public void h() {
    lockH.lock();
    try {
      for(int i = 0; i < 5; i++) {
        print("h()");
        Thread.yield();
      }
    } finally {
      lockH.unlock();
    }
  }
}

public class E16_ExplicitSyncObject {
  public static void main(String[] args) throws Exception {
    final ExplicitSingleSynch singleSync =
      new ExplicitSingleSynch();
    final ExplicitTripleSynch tripleSync =
      new ExplicitTripleSynch();
    print("Test ExplicitSingleSynch...");
    Thread t1 = new Thread() {
      public void run() {
        singleSync.f();
      }
    };
    t1.start();
    Thread t2 = new Thread() {
      public void run() {
        singleSync.g();
      }
    };
    t2.start();
    singleSync.h();
    t1.join();  // Wait for t1 to finish its work
    t2.join();  // Wait for t2 to finish its work
    print("Test ExplicitTripleSynch...");
```

```
    new Thread() {
      public void run() {
        tripleSync.f();
      }
    }.start();
    new Thread() {
      public void run() {
        tripleSync.g();
      }
    }.start();
    tripleSync.h();
  }
} /* Output: (Sample)
Test ExplicitSingleSynch...
h()
h()
h()
h()
h()
f()
f()
f()
f()
f()
g()
g()
g()
g()
g()
Test ExplicitTripleSynch...
h()
f()
g()
h()
f()
g()
h()
f()
g()
h()
f()
g()
h()
f()
g()
*///:~
```

Exercise 17

```
//: concurrency/E17_RadiationCounter.java
/****************** Exercise 17 ************************
 * Create a radiation counter that can have any number of
 * remote sensors.
 ******************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class Count {
  private int count = 0;
  private Random rand = new Random(47);
  // Remove the synchronized keyword to see counting fail:
  public synchronized int increment() {
    int temp = count;
    if(rand.nextBoolean()) // Yield half the time
      Thread.yield();
    return (count = ++temp);
  }
  public synchronized int value() { return count; }
}

class Sensor implements Runnable {
  private static Random rand = new Random(47);
  private static Count count = new Count();
  private static List<Sensor> sensors =
    new ArrayList<Sensor>();
  private int number;
  private final int id;
  private static volatile boolean canceled = false;
  public static void cancel() { canceled = true; }
  public Sensor(int id) {
    this.id = id;
    sensors.add(this);
  }
  public void run() {
    while(!canceled) {
      // Simulate a random occurence of an ionizing event
      if(rand.nextInt(5) == 0) {
        synchronized(this) { ++number; }
        count.increment();
      }
      try {
```

```
        TimeUnit.MILLISECONDS.sleep(100);
      } catch(InterruptedException e) {
        print("sleep interrupted");
      }
    }
  }
  public synchronized int getValue() { return number; }
  public String toString() {
    return "Sensor " + id + ": " + getValue();
  }
  public static int getTotalCount() {
    return count.value();
  }
  public static int sumSensors() {
    int sum = 0;
    for(Sensor sensor : sensors)
      sum += sensor.getValue();
    return sum;
  }
}

public class E17_RadiationCounter {
  public static void main(String[] args) throws Exception {
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < 5; i++)
      exec.execute(new Sensor(i));
    TimeUnit.SECONDS.sleep(3);
    Sensor.cancel();
    exec.shutdown();
    if(!exec.awaitTermination(250, TimeUnit.MILLISECONDS))
      print("Some tasks were not terminated!");
    print("Total: " + Sensor.getTotalCount());
    print("Sum of Sensors: " + Sensor.sumSensors());
  }
} /* Output: (Sample)
Total: 25
Sum of Sensors: 25
*///:~
```

We follow the logic of the **concurrency/OrnamentalGarden.java** program from *TIJ4*. The *radiation counter* processes and counts the electrical pulse caused by each ionizing event. **Count** acts as an event processing unit, and **Sensor** a radiation detector. The output measures the radiation over time (hard-coded to 3 seconds).

Exercise 18

```
//: concurrency/E18_Interruption.java
/****************** Exercise 18 ************************
 * Create a non-task class with a method that calls
 * sleep() for a long interval. Create a task that calls
 * the method in the non-task class. In main(), start the
 * task, then call interrupt() to terminate it. Make sure
 * that the task shuts down safely.
 ***************************************************/
package concurrency;
import java.util.concurrent.*;

class NonTask {
  static void longMethod() throws InterruptedException {
    TimeUnit.SECONDS.sleep(60);  // Waits one minute
  }
}

class Task implements Runnable {
  public void run() {
    try {
      NonTask.longMethod();
    } catch (InterruptedException ie) {
      System.out.println(ie.toString());
    } finally {
      // Any cleanup code here...
    }
  }
}

public class E18_Interruption {
  public static void main(String[] args) throws Exception {
    Thread t = new Thread(new Task());
    t.start();
    TimeUnit.SECONDS.sleep(1);
    t.interrupt();
  }
} /* Output:
java.lang.InterruptedException: sleep interrupted
*///:~
```

Exercise 19

```
//: concurrency/E19_OrnamentalGarden2.java
```

```
/****************** Exercise 19 ************************
 * Modify OrnamentalGarden.java so that it uses
 * interrupt().
*****************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class Entrance2 implements Runnable {
  private static Count count = new Count();
  private static List<Entrance2> entrances =
    new ArrayList<Entrance2>();
  private int number;
  private final int id;
  public Entrance2(int id) {
    this.id = id;
    entrances.add(this);
  }
  public void run() {
    for(;;) {
      synchronized(this) { ++number; }
      print(this + " Total: " + count.increment());
      try {
        TimeUnit.MILLISECONDS.sleep(100);
      } catch(InterruptedException e) {
        print("Stopping " + this);
        return;
      }
    }
  }
  public synchronized int getValue() { return number; }
  public String toString() {
    return "Entrance " + id + ": " + getValue();
  }
  public static int getTotalCount() {
    return count.value();
  }
  public static int sumEntrances() {
    int sum = 0;
    for(Entrance2 entrance : entrances)
      sum += entrance.getValue();
    return sum;
  }
}

public class E19_OrnamentalGarden2 {
```

```
  public static void main(String[] args) throws Exception {
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < 5; i++)
      exec.execute(new Entrance2(i));
    TimeUnit.SECONDS.sleep(3);
    exec.shutdownNow();
    if(!exec.awaitTermination(250, TimeUnit.MILLISECONDS))
      print("Some tasks were not terminated!");
    print("Total: " + Entrance2.getTotalCount());
    print("Sum of Entrances: " + Entrance2.sumEntrances());
  }
} /* Output: (Sample)
Entrance 1: 1 Total: 2
Entrance 2: 1 Total: 3
Entrance 3: 1 Total: 4
Entrance 4: 1 Total: 5
...
Entrance 2: 31 Total: 152
Entrance 3: 31 Total: 153
Entrance 4: 31 Total: 154
Entrance 0: 31 Total: 155
Terminating Entrance 0: 31
Terminating Entrance 4: 31
Terminating Entrance 3: 31
Terminating Entrance 2: 31
Terminating Entrance 1: 31
Total: 155
Sum of Entrances: 155
*///:~
```

Notice how much neater this program is with the built-in thread interruption facility.

Exercise 20

```
//: concurrency/E20_InterruptCachedThreadPool.java
/****************** Exercise 20 ************************
 * Modify CachedThreadPool.java so that all tasks receive
 * an interrupt() before they are completed.
 ***************************************************/
package concurrency;
import java.util.concurrent.*;

class LiftOff2 implements Runnable {
  protected int countDown = 5000;
  private static int taskCount;
```

```
  private final int id = taskCount++;
  public LiftOff2() {}
  public LiftOff2(int countDown) {
    this.countDown = countDown;
  }
  public String status() {
    return "#" + id + "(" +
      (countDown > 0 ? countDown : "Liftoff!") + "), ";
  }
  public void run() {
    while(countDown-- > 0) {
      if(Thread.currentThread().isInterrupted()) {
        System.out.println("Interrupted: #" + id);
        return;
      }
      System.out.print(status());
      Thread.yield();
    }
  }
}

public class E20_InterruptCachedThreadPool {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < 5; i++)
      exec.execute(new LiftOff2());
    Thread.yield();
    exec.shutdownNow();
  }
} /* Output: (Sample)
#0(4999), #1(4999), #2(4999), #3(4999), #4(4999),
Interrupted: #0
Interrupted: #1
Interrupted: #2
Interrupted: #3
Interrupted: #4
*///:~
```

The above solution is simple: all tasks receive an **interrupt( )** before completing.

Exercise 21

```
//: concurrency/E21_ThreadCooperation.java
/****************** Exercise 21 *****************
 * Create two Runnables, one with a run() that
```

```
 * starts and calls wait(). The second class should
 * capture the reference of the first Runnable object.
 * Its run() should call notifyAll() for the first
 * task after some number of seconds have passed so
 * that the first task can display a message. Test
 * your classes using an Executor.
 *********************************************/
package concurrency;
import java.util.concurrent.*;
import static net.mindview.util.Print.*;

class Coop1 implements Runnable {
  public Coop1() { print("Constructed Coop1"); }
  public void run() {
    print("Coop1 going into wait");
    synchronized(this) {
      try {
        wait();
      } catch(InterruptedException ignore) {}
    }
    print("Coop1 exited wait");
  }
}

class Coop2 implements Runnable {
  Runnable otherTask;
  public Coop2(Runnable otherTask) {
    this.otherTask = otherTask;
    print("Constructed Coop2");
  }
  public void run() {
    print("Coop2 pausing 5 seconds");
    try {
      TimeUnit.SECONDS.sleep(5);
    } catch(InterruptedException ignore) {}
    print("Coop2 calling notifyAll");
    synchronized(otherTask) { otherTask.notifyAll(); }
  }
}

public class E21_ThreadCooperation {
  public static void main(String args[]) throws Exception {
    Runnable coop1 = new Coop1(),
             coop2 = new Coop2(coop1);
    ExecutorService exec = Executors.newCachedThreadPool();
    exec.execute(coop1);
    exec.execute(coop2);
```

```
    Thread.yield();
    exec.shutdown();
  }
} /* Output: (Sample)
Constructed Coop1
Constructed Coop2
Coop1 going into wait
Coop2 pausing 5 seconds
Coop2 calling notifyAll
Coop1 exited wait
*///:~
```

This is not a properly written concurrent program, although it runs correctly. You should not synchronize tasks based on different timings. You will explore more sophisticated solutions later in *TIJ4*.

Exercise 22

```
//: concurrency/E22_BusyWait.java
/****************** Exercise 22 *************************
 * Create an example of a busy wait. One task sleeps for a
 * while and then sets a flag to true. The second task
 * watches that flag inside a while loop (this is the busy
 * wait) and when the flag becomes true, sets it back to
 * false and reports the change to the console. Note how
 * much wasted time the program spends inside the busy wait,
 * and create a second version of the program that uses
 * wait() instead of the busy wait.
 ****************************************************/
package concurrency;
import java.util.concurrent.*;

public class E22_BusyWait {
  private static volatile boolean flag;
  private static int spins;
  public static void main(String[] args) throws Exception {
    Runnable r1 = new Runnable() {
      public void run() {
        for(;;) {
          try {
            TimeUnit.MILLISECONDS.sleep(10);
          } catch (InterruptedException e) { return; }
          flag = true;
        }
      }
    };
```

```
    Runnable r2 = new Runnable() {
      public void run() {
        for(;;) {
          // The busy-wait loop
          while(!flag &&
                !Thread.currentThread().isInterrupted())
            spins++;
          System.out.println("Spun " + spins + " times");
          spins = 0;
          flag = false;
          if(Thread.interrupted())
            return;
        }
      }
    };
    ExecutorService exec = Executors.newCachedThreadPool();
    exec.execute(r1);
    exec.execute(r2);
    TimeUnit.SECONDS.sleep(1);
    exec.shutdownNow();
  }
} /* Output: (Sample)
Spun 5546 times
Spun 6744 times
Spun 20270 times
...
Spun 35944 times
Spun 17733 times
Spun 23670 times
*///:~
```

flag facilitates communication between the two tasks. Normally, you might see this with a busy-wait:

```
while(!flag)
  ;
```

However, we are tracking the amount of activity in the busy-wait loop. Run the program to see why it's called "busy wait."

wait( ) and **notify( )** don't raise a flag because the communication occurs via the threading mechanism.

```
//: concurrency/E22_WaitNotify.java
// The second version using wait().
package concurrency;
import java.util.concurrent.*;
```

```
public class E22_WaitNotify {
  public static void main(String[] args) throws Exception {
    final Runnable r1 = new Runnable() {
      public void run() {
        for(;;) {
          try {
            TimeUnit.MILLISECONDS.sleep(100);
            synchronized(this) { notify(); }
          } catch(InterruptedException e) { return; }
        }
      }
    };
    Runnable r2 = new Runnable() {
      public void run() {
        for(;;) {
          try {
            synchronized(r1) { r1.wait(); }
          } catch(InterruptedException e) { return; }
          System.out.print("Cycled ");
        }
      }
    };
    ExecutorService exec = Executors.newCachedThreadPool();
    exec.execute(r1);
    exec.execute(r2);
    TimeUnit.SECONDS.sleep(1);
    exec.shutdownNow();
  }
} /* Output: (Sample)
Cycled Cycled Cycled Cycled Cycled Cycled Cycled Cycled
Cycled Cycled
*///:~
```

The **r2** task calls **wait( )** on **r1**, and **r1** calls **notify( )** on itself. Again, this program's "correctness" depends upon **r1**'s sleeping time; if you comment out the corresponding line, you deadlock due to a *missed signal*. You'll study advanced task control techniques later in *TIJ4*.

Exercise 23

```
//: concurrency/waxomatic/E23_WaxOMatic2.java
/********************* Exercise 23 **********************
 * Demonstrate that WaxOMatic.java works when
 * you use notify() instead of notifyAll().
*******************************************************/
package concurrency.waxomatic;
```

```
import java.util.concurrent.*;
import static net.mindview.util.Print.*;

class Car {
  private boolean waxOn;
  public synchronized void waxed() {
    waxOn = true; // Ready to buff
    notify();
  }
  public synchronized void buffed() {
    waxOn = false; // Ready for another coat of wax
    notify();
  }
  public synchronized void waitForWaxing()
  throws InterruptedException {
    while(waxOn == false)
      wait();
  }
  public synchronized void waitForBuffing()
  throws InterruptedException {
    while(waxOn == true)
      wait();
  }
}

class WaxOn implements Runnable {
  private Car car;
  public WaxOn(Car c) { car = c; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        printnb("Wax On! ");
        TimeUnit.MILLISECONDS.sleep(200);
        car.waxed();
        car.waitForBuffing();
      }
    } catch(InterruptedException e) {
      print("Exiting via interrupt");
    }
    print("Ending Wax On task");
  }
}

class WaxOff implements Runnable {
  private Car car;
  public WaxOff(Car c) { car = c; }
  public void run() {
```

```
    try {
      while(!Thread.interrupted()) {
        car.waitForWaxing();
        printnb("Wax Off! ");
        TimeUnit.MILLISECONDS.sleep(200);
        car.buffed();
      }
    } catch(InterruptedException e) {
      print("Exiting via interrupt");
    }
    print("Ending Wax Off task");
  }
}

public class E23_WaxOMatic2 {
  public static void main(String[] args) throws Exception {
    Car car = new Car();
    ExecutorService exec = Executors.newCachedThreadPool();
    exec.execute(new WaxOff(car));
    exec.execute(new WaxOn(car));
    TimeUnit.SECONDS.sleep(5); // Run for a while...
    exec.shutdownNow(); // Interrupt all tasks
  }
} /* Output: (Sample)
Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On!
Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off!
Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On!
Wax Off! Wax On! Wax Off! Wax On! Wax Off! Exiting via
interrupt
Ending Wax Off task
Exiting via interrupt
Ending Wax On task
*///:~
```

The output is equal to the previous program, which uses **notifyAll( )**. We meet the requirements of the ***notify( ) vs. notifyAll( )*** section of *TIJ4*, so using **notify( )** is fine.

Exercise 24

```
//: concurrency/E24_ProducerConsumer.java
// {Args: 1 200}
/********************** Exercise 24 ***********************
 * Solve a single-producer, single-consumer problem using
 * wait() and notifyAll(). The producer must not overflow
 * the receiver's buffer, which can happen if the producer
```

```
 * is faster than the consumer. If the consumer is faster
 * than the producer, then it must not read the same data
 * more than once. Do not assume anything about the relative
 * speeds of the producer or consumer.
*****************************************************/
package concurrency;
import java.util.*;
import java.util.concurrent.*;

// A queue for solving flow-control problems.
class FlowQueue<T> {
  private Queue<T> queue = new LinkedList<T>();
  private int maxSize;
  public FlowQueue(int maxSize) { this.maxSize = maxSize; }
  public synchronized void put(T v)
    throws InterruptedException {
    while(queue.size() >= maxSize)
      wait();
    queue.offer(v);
    maxSize++;
    notifyAll();
  }
  public synchronized T get() throws InterruptedException {
    while(queue.isEmpty())
      wait();
    T returnVal = queue.poll();
    maxSize--;
    notifyAll();
    return returnVal;
  }
}
class Item {
  private static int counter;
  private int id = counter++;
  public String toString() { return "Item " + id; }
}

// Produces Item objects
class Producer implements Runnable {
  private int delay;
  private FlowQueue<Item> output;
  public Producer(FlowQueue<Item> output, int sleepTime) {
    this.output = output;
    delay = sleepTime;
  }
  public void run() {
    for(;;) {
```

```
      try {
        output.put(new Item());
        TimeUnit.MILLISECONDS.sleep(delay);
      } catch(InterruptedException e) { return; }
    }
  }
}

// Consumes any object
class Consumer implements Runnable {
  private int delay;
  private FlowQueue<?> input;
  public Consumer(FlowQueue<?> input, int sleepTime) {
    this.input = input;
    delay = sleepTime;
  }
  public void run() {
    for(;;) {
      try {
        System.out.println(input.get());
        TimeUnit.MILLISECONDS.sleep(delay);
      } catch(InterruptedException e) { return; }
    }
  }
}

public class E24_ProducerConsumer {
  public static void main(String[] args) throws Exception {
    if(args.length < 2) {
     System.err.println("Usage java E24_ProducerConsumer" +
        " <producer sleep time> <consumer sleep time>");
      System.exit(1);
    }
    int producerSleep = Integer.parseInt(args[0]);
    int consumerSleep = Integer.parseInt(args[1]);
    FlowQueue<Item> fq = new FlowQueue<Item>(100);
    ExecutorService exec = Executors.newFixedThreadPool(2);
    exec.execute(new Producer(fq, producerSleep));
    exec.execute(new Consumer(fq, consumerSleep));
    TimeUnit.SECONDS.sleep(2);
    exec.shutdownNow();
  }
} /* Output: (Sample)
Item 0
Item 1
Item 2
Item 3
```

```
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10
*///:~
```

The queue ensures that the consumer will not read a value more than once. If the producer overfills the queue, the call to **put( )** suspends the producer's thread until the number of elements drops below **maxSize**. This solution is elegant and reusable.

Try the program with different production and consumption speeds. For example:

```
java E24_ProducerConsumer 200 1
```

Exercise 25

```
//: concurrency/E25_Restaurant.java
/********************* Exercise 25 **********************
 * In the Chef class in Restaurant.java, return from run()
 * after calling shutdownNow() and observe the difference
 * in behavior.
*******************************************************/
package concurrency;
import java.util.concurrent.*;
import static net.mindview.util.Print.*;

class Meal {
  private final int orderNum;
  public Meal(int orderNum) { this.orderNum = orderNum; }
  public String toString() { return "Meal " + orderNum; }
}

class WaitPerson implements Runnable {
  private Restaurant restaurant;
  public WaitPerson(Restaurant r) { restaurant = r; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        synchronized(this) {
          while(restaurant.meal == null)
            wait(); // ... for the chef to produce a meal
```

```
        }
        print("Waitperson got " + restaurant.meal);
        synchronized(restaurant.chef) {
          restaurant.meal = null;
          restaurant.chef.notifyAll(); // Ready for another
        }
      }
    } catch(InterruptedException e) {
      print("WaitPerson interrupted");
    }
  }
}

class Chef implements Runnable {
  private Restaurant restaurant;
  private int count = 0;
  public Chef(Restaurant r) { restaurant = r; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        synchronized(this) {
          while(restaurant.meal != null)
            wait(); // ... for the meal to be taken
        }
        if(++count == 10) {
          print("Out of food, closing");
          restaurant.exec.shutdownNow();
          return;
        }
        printnb("Order up! ");
        synchronized(restaurant.waitPerson) {
          restaurant.meal = new Meal(count);
          restaurant.waitPerson.notifyAll();
        }
        TimeUnit.MILLISECONDS.sleep(100);
      }
    } catch(InterruptedException e) {
      print("Chef interrupted");
    }
  }
}

class Restaurant {
  Meal meal;
  ExecutorService exec = Executors.newCachedThreadPool();
  WaitPerson waitPerson = new WaitPerson(this);
  Chef chef = new Chef(this);
```

```
  public Restaurant() {
    exec.execute(chef);
    exec.execute(waitPerson);
  }
  public static void main(String[] args) {
    new Restaurant();
  }
}

public class E25_Restaurant {
  public static void main(String[] args) {
    new Restaurant();
  }
} /* Output: (Sample)
Order up! Waitperson got Meal 1
Order up! Waitperson got Meal 2
Order up! Waitperson got Meal 3
Order up! Waitperson got Meal 4
Order up! Waitperson got Meal 5
Order up! Waitperson got Meal 6
Order up! Waitperson got Meal 7
Order up! Waitperson got Meal 8
Order up! Waitperson got Meal 9
Out of food, closing
WaitPerson interrupted
*///:~
```

The "Order up!" and "Chef interrupted" messages no longer appear, since the **Chef** task exits immediately.

Exercise 26

```
//: concurrency/E26_Restaurant2.java
/********************* Exercise 26 **********************
 * Add a BusBoy class to Restaurant.java. After the meal is
 * delivered, the WaitPerson should notify the BusBoy to
 * clean up.
*******************************************************/
package concurrency;
import java.util.concurrent.*;
import static net.mindview.util.Print.*;

class WaitPerson2 implements Runnable {
  private Restaurant2 restaurant;
  boolean notified;
  public WaitPerson2(Restaurant2 r) { restaurant = r; }
```

```
  public void run() {
    try {
      while(!Thread.interrupted()) {
        synchronized(this) {
          while(restaurant.meal == null)
            wait(); // ... for the chef to produce a meal
        }
        print("Waitperson got " + restaurant.meal);
        synchronized(restaurant.busBoy) {
          restaurant.busBoy.notified = true;
          restaurant.busBoy.meal = restaurant.meal;
          restaurant.busBoy.notifyAll(); // Clean up
        }
        synchronized(restaurant.chef) {
          restaurant.meal = null;
          restaurant.chef.notifyAll(); // Ready for another
        }
        synchronized(this) {
          if(!notified)
            wait(); // ... for the bus boy to clean up
          notified = false;
        }
      }
    } catch(InterruptedException e) {
      print("WaitPerson interrupted");
    }
  }
}

class BusBoy implements Runnable {
  private Restaurant2 restaurant;
  boolean notified;
  volatile Meal meal;
  public BusBoy(Restaurant2 r) { restaurant = r; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        synchronized(this) {
          if(!notified)
            wait(); // ... for meal delivery
          notified = false;
        }
        print("Busboy cleaned up " + meal);
        synchronized(restaurant.waitPerson) {
          restaurant.waitPerson.notified = true;
          restaurant.waitPerson.notifyAll();
        }
```

```
      }
    } catch(InterruptedException e) {
      print("BusBoy interrupted");
    }
  }
}

class Chef2 implements Runnable {
  private Restaurant2 restaurant;
  private int count = 0;
  public Chef2(Restaurant2 r) { restaurant = r; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        synchronized(this) {
          while(restaurant.meal != null)
            wait(); // ... for the meal to be taken
        }
        if(++count == 10) {
          print("Out of food, closing");
          restaurant.exec.shutdownNow();
        }
        printnb("Order up! ");
        synchronized(restaurant.waitPerson) {
          restaurant.meal = new Meal(count);
          restaurant.waitPerson.notifyAll();
        }
        TimeUnit.MILLISECONDS.sleep(100);
      }
    } catch(InterruptedException e) {
      print("Chef interrupted");
    }
  }
}

class Restaurant2 {
  Meal meal;
  ExecutorService exec = Executors.newCachedThreadPool();
  WaitPerson2 waitPerson = new WaitPerson2(this);
  BusBoy busBoy = new BusBoy(this);
  Chef2 chef = new Chef2(this);
  public Restaurant2() {
    exec.execute(chef);
    exec.execute(waitPerson);
    exec.execute(busBoy);
  }
}
```

```
public class E26_Restaurant2 {
  public static void main(String[] args) {
    new Restaurant2();
  }
} /* Output: (Sample)
Order up! Waitperson got Meal 1
Busboy cleaned up Meal 1
Order up! Waitperson got Meal 2
Busboy cleaned up Meal 2
Order up! Waitperson got Meal 3
Busboy cleaned up Meal 3
Order up! Waitperson got Meal 4
Busboy cleaned up Meal 4
Order up! Waitperson got Meal 5
Busboy cleaned up Meal 5
Order up! Waitperson got Meal 6
Busboy cleaned up Meal 6
Order up! Waitperson got Meal 7
Busboy cleaned up Meal 7
Order up! Waitperson got Meal 8
Busboy cleaned up Meal 8
Order up! Waitperson got Meal 9
Busboy cleaned up Meal 9
Out of food, closing
WaitPerson interrupted
BusBoy interrupted
Order up! Chef interrupted
*///:~
```

The waitperson notifies the busboy and the chef in parallel, so meal preparation and clean up are simultaneous. We use the **notified** field in the **BusBoy** and **WaitPerson** classes; without this flag, a *missed signal* may stall the program. It is vital that you understand how these entities communicate with each other, and the various scenarios that you encounter in concurrent programs.

Exercise 27

```
//: concurrency/E27_Restaurant3.java
/********************* Exercise 27 **********************
 * Modify Restaurant.java to use explicit Lock and Condition
 * objects.
*******************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;
```

```
import static net.mindview.util.Print.*;

class WaitPerson3 implements Runnable {
  private Restaurant3 restaurant;
  Lock lock = new ReentrantLock();
  Condition condition = lock.newCondition();
  public WaitPerson3(Restaurant3 r) { restaurant = r; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        lock.lock();
        try {
          while(restaurant.meal == null)
            condition.await();
        } finally {
          lock.unlock();
        }
        print("Waitperson got " + restaurant.meal);
        restaurant.chef.lock.lock();
        try {
          restaurant.meal = null;
          restaurant.chef.condition.signalAll();
        } finally {
          restaurant.chef.lock.unlock();
        }
      }
    } catch(InterruptedException e) {
      print("WaitPerson interrupted");
    }
  }
}

class Chef3 implements Runnable {
  private Restaurant3 restaurant;
  private int count;
  Lock lock = new ReentrantLock();
  Condition condition = lock.newCondition();
  public Chef3(Restaurant3 r) { restaurant = r; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        lock.lock();
        try {
          while(restaurant.meal != null)
            condition.await();
        } finally {
          lock.unlock();
```

```
        }
        if(++count == 10) {
          print("Out of food, closing");
          restaurant.exec.shutdownNow();
        }
        printnb("Order up! ");
        restaurant.waitPerson.lock.lock();
        try {
          restaurant.meal = new Meal(count);
          restaurant.waitPerson.condition.signalAll();
        } finally {
          restaurant.waitPerson.lock.unlock();
        }
        TimeUnit.MILLISECONDS.sleep(100);
      }
    } catch(InterruptedException e) {
      print("Chef interrupted");
    }
  }
}

class Restaurant3 {
  Meal meal;
  ExecutorService exec = Executors.newCachedThreadPool();
  WaitPerson3 waitPerson = new WaitPerson3(this);
  Chef3 chef = new Chef3(this);
  public Restaurant3() {
    exec.execute(chef);
    exec.execute(waitPerson);
  }
}

public class E27_Restaurant3 {
  public static void main(String[] args) {
    new Restaurant3();
  }
} /* Output: (Sample)
Order up! Waitperson got Meal 1
Order up! Waitperson got Meal 2
Order up! Waitperson got Meal 3
Order up! Waitperson got Meal 4
Order up! Waitperson got Meal 5
Order up! Waitperson got Meal 6
Order up! Waitperson got Meal 7
Order up! Waitperson got Meal 8
Order up! Waitperson got Meal 9
Out of food, closing
```

```
WaitPerson interrupted
Order up! Chef interrupted
*///:~
```

Exercise 28

```
//: concurrency/E28_TestBlockingQueues2.java
// {RunByHand}
/********************** Exercise 28 **********************
 * Modify TestBlockingQueues.java by adding a new task that
 * places LiftOff on the BlockingQueue, instead of doing it
 * in main().
*********************************************************/
package concurrency;
import java.io.*;
import java.util.concurrent.*;
import static net.mindview.util.Print.*;

class LiftOffRunner implements Runnable {
  private BlockingQueue<LiftOff> rockets;
  public LiftOffRunner(BlockingQueue<LiftOff> queue) {
    rockets = queue;
  }
  public void add(LiftOff lo) throws InterruptedException {
    rockets.put(lo);
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        LiftOff rocket = rockets.take();
        rocket.run(); // Use this thread
      }
    } catch(InterruptedException e) {
      print("Waking from take()");
    }
    print("Exiting LiftOffRunner");
  }
}

class LiftOffProducer implements Runnable {
  private LiftOffRunner runner;
  public LiftOffProducer(LiftOffRunner runner) {
    this.runner = runner;
  }
  public void run() {
    try {
```

```
      for(int i = 0; i < 5; i++)
        runner.add(new LiftOff(5));
    } catch(InterruptedException e) {
      print("Waking from put()");
    }
    print("Exiting LiftOffProducer");
  }
}

public class E28_TestBlockingQueues2 {
  private static void getkey() {
    try {
      new BufferedReader(
        new InputStreamReader(System.in)).readLine();
    } catch(java.io.IOException e) {
      throw new RuntimeException(e);
    }
  }
  private static void getkey(String message) {
    print(message);
    getkey();
  }
  private static void
  test(String msg, BlockingQueue<LiftOff> queue) {
    ExecutorService exec = Executors.newFixedThreadPool(2);
    print(msg);
    LiftOffRunner runner = new LiftOffRunner(queue);
    LiftOffProducer producer = new LiftOffProducer(runner);
    exec.execute(runner);
    exec.execute(producer);
    getkey("Press 'ENTER' (" + msg + ")");
    exec.shutdownNow();
    print("Finished " + msg + " test");
  }
  public static void main(String[] args) {
    test("LinkedBlockingQueue", // Unlimited size
      new LinkedBlockingQueue<LiftOff>());
    test("ArrayBlockingQueue",  // Fixed size
      new ArrayBlockingQueue<LiftOff>(3));
    test("SynchronousQueue",    // Size of 1
      new SynchronousQueue<LiftOff>());
  }
} ///:~
```

Exercise 29

```
//: concurrency/E29_ToastOMatic2.java
/********************** Exercise 29 ***********************
 * Modify ToastOMatic.java to create peanut butter and jelly
 * on toast sandwiches using two separate assembly lines
 * (one for peanut butter, the second for jelly, then
 * merging the two lines).
*********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class Toast {
  public enum Status {
    DRY,
    BUTTERED,
    JAMMED,
    READY {
      public String toString() {
        return
          BUTTERED.toString() + " & " + JAMMED.toString();
      }
    }
  }
  private Status status = Status.DRY;
  private final int id;
  public Toast(int idn) { id = idn; }
  public void butter() {
    status =
      (status == Status.DRY) ? Status.BUTTERED :
                               Status.READY;
  }
  public void jam() {
    status =
      (status == Status.DRY) ? Status.JAMMED :
                               Status.READY;
  }
  public Status getStatus() { return status; }
  public int getId() { return id; }
  public String toString() {
    return "Toast " + id + ": " + status;
  }
}
```

```
class ToastQueue extends LinkedBlockingQueue<Toast> {}

class Toaster implements Runnable {
  private ToastQueue toastQueue;
  private int count;
  private Random rand = new Random(47);
  public Toaster(ToastQueue tq) { toastQueue = tq; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        TimeUnit.MILLISECONDS.sleep(
          100 + rand.nextInt(500));
        // Make toast
        Toast t = new Toast(count++);
        print(t);
        // Insert into queue
        toastQueue.put(t);
      }
    } catch(InterruptedException e) {
      print("Toaster interrupted");
    }
    print("Toaster off");
  }
}

// Apply butter to toast:
class Butterer implements Runnable {
  private ToastQueue inQueue, butteredQueue;
  public Butterer(ToastQueue in, ToastQueue buttered) {
    inQueue = in;
    butteredQueue = buttered;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until next piece of toast is available:
        Toast t = inQueue.take();
        t.butter();
        print(t);
        butteredQueue.put(t);
      }
    } catch(InterruptedException e) {
      print("Butterer interrupted");
    }
    print("Butterer off");
  }
}
```

```
// Apply jam to toast:
class Jammer implements Runnable {
  private ToastQueue inQueue, jammedQueue;
  public Jammer(ToastQueue in, ToastQueue jammed) {
    inQueue = in;
    jammedQueue = jammed;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until next piece of toast is available:
        Toast t = inQueue.take();
        t.jam();
        print(t);
        jammedQueue.put(t);
      }
    } catch(InterruptedException e) {
      print("Jammer interrupted");
    }
    print("Jammer off");
  }
}

// Consume the toast:
class Eater implements Runnable {
  private ToastQueue finishedQueue;
  public Eater(ToastQueue finished) {
    finishedQueue = finished;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until next piece of toast is available:
        Toast t = finishedQueue.take();
        // Verify that all pieces are ready for consumption:
        if(t.getStatus() != Toast.Status.READY) {
          print(">>>> Error: " + t);
          System.exit(1);
        } else
          print("Chomp! " + t);
      }
    } catch(InterruptedException e) {
      print("Eater interrupted");
    }
    print("Eater off");
  }
```

```
}

// Outputs alternate inputs on alternate channels:
class Alternator implements Runnable {
  private ToastQueue inQueue, out1Queue, out2Queue;
  private boolean outTo2;  // control alternation
  public Alternator(ToastQueue in, ToastQueue out1,
          ToastQueue out2) {
    inQueue = in;
    out1Queue = out1;
    out2Queue = out2;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until next piece of toast is available:
        Toast t = inQueue.take();
        if(!outTo2)
          out1Queue.put(t);
        else
          out2Queue.put(t);
        outTo2 = !outTo2;  // change state for next time
      }
    } catch(InterruptedException e) {
      print("Alternator interrupted");
    }
    print("Alternator off");
  }
}

// Accepts toasts on either channel, and relays them on to
// a "single" successor
class Merger implements Runnable {
  private ToastQueue in1Queue, in2Queue, toBeButteredQueue,
    toBeJammedQueue, finishedQueue;
  public Merger(ToastQueue in1, ToastQueue in2,
          ToastQueue toBeButtered, ToastQueue toBeJammed,
          ToastQueue finished) {
    in1Queue = in1;
    in2Queue = in2;
    toBeButteredQueue = toBeButtered;
    toBeJammedQueue = toBeJammed;
    finishedQueue = finished;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
```

```
        // Blocks until next piece of toast is available:
        Toast t = null;
        while(t == null) {
          t = in1Queue.poll(50, TimeUnit.MILLISECONDS);
          if(t != null)
            break;
          t = in2Queue.poll(50, TimeUnit.MILLISECONDS);
        }
        // Relay toast onto the proper queue
        switch(t.getStatus()) {
          case BUTTERED:
            toBeJammedQueue.put(t);
            break;
          case JAMMED:
            toBeButteredQueue.put(t);
            break;
          default:
            finishedQueue.put(t);
        }
      }
    } catch(InterruptedException e) {
      print("Merger interrupted");
    }
    print("Merger off");
  }
}

public class E29_ToastOMatic2 {
  public static void main(String[] args) throws Exception {
    ToastQueue
      dryQueue = new ToastQueue(),
      butteredQueue = new ToastQueue(),
      toBeButteredQueue = new ToastQueue(),
      jammedQueue = new ToastQueue(),
      toBeJammedQueue = new ToastQueue(),
      finishedQueue = new ToastQueue();
    ExecutorService exec = Executors.newCachedThreadPool();
    exec.execute(new Toaster(dryQueue));
    exec.execute(new Alternator(dryQueue, toBeButteredQueue,
      toBeJammedQueue));
    exec.execute(
      new Butterer(toBeButteredQueue, butteredQueue));
    exec.execute(
      new Jammer(toBeJammedQueue, jammedQueue));
    exec.execute(new Merger(butteredQueue , jammedQueue,
      toBeButteredQueue, toBeJammedQueue, finishedQueue));
    exec.execute(new Eater(finishedQueue));
```

```
    TimeUnit.SECONDS.sleep(5);
    exec.shutdownNow();
  }
} /* (Execute to see output) *///:~
```

The finished sandwiches no longer come in predefined order because the two assembly lines work in parallel. Our solution enhances the original program from *TIJ4* with an assembly line elaboration by Doug Lea (http://gee.cs.oswego.edu/dl/cpj/assembly.html).

Lea writes: "*Polling intrinsically relies on busy-wait loops, which are intrinsically wasteful (but still sometimes less so than context-switching). Coping with this requires empirically guided decisions about how to insert sleeps, yields, or alternative actions to strike a balance between conserving CPU time and maintaining acceptable average response latencies.*" We must poll each queue in the program with a wait time of 50 milliseconds (inside the **Merger** task), because data could come from any queue, so blocking a specific one indefinitely doesn't work. The timed **poll( )** method from the JDK needs no *busy-wait* loop. We add only **Alternator** and **Merger**; **Toaster**, **Butterer**, **Jammer** and **Eater** remain the same.

Exercise 30

```
//: concurrency/E30_SendReceive.java
/********************* Exercise 30 **********************
 * Modify PipedIO.java to use a BlockingQueue instead of
 * a pipe.
*******************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class CharQueue extends LinkedBlockingQueue<Character> {}

class Sender implements Runnable {
  private Random rand = new Random(47);
  private CharQueue out = new CharQueue();
  public CharQueue getQueue() { return out; }
  public void run() {
    try {
      while(true)
        for(char c = 'A'; c <= 'z'; c++) {
          out.put(c);
          TimeUnit.MILLISECONDS.sleep(rand.nextInt(500));
```

```
        }
      } catch(InterruptedException e) {
        print(e + " Sender interrupted");
      }
    }
}

class Receiver implements Runnable {
  private CharQueue in;
  public Receiver(Sender sender) { in = sender.getQueue(); }
  public void run() {
    try {
      while(true) {
        // Blocks until characters are there:
        printnb("Read: " + in.take() + ", ");
      }
    } catch(InterruptedException e) {
      print(e + " Reader interrupted");
    }
  }
}

public class E30_SendReceive {
  public static void main(String[] args) throws Exception {
    Sender sender = new Sender();
    Receiver receiver = new Receiver(sender);
    ExecutorService exec = Executors.newCachedThreadPool();
    exec.execute(sender);
    exec.execute(receiver);
    TimeUnit.SECONDS.sleep(4);
    exec.shutdownNow();
  }
} /* Output: (Sample)
Read: A, Read: B, Read: C, Read: D, Read: E, Read: F, Read:
G, Read: H, Read: I, Read: J, Read: K, Read: L, Read: M,
Read: N, Read: O, Read: P, Read: Q,
java.lang.InterruptedException Reader interrupted
java.lang.InterruptedException: sleep interrupted Sender
interrupted
*///:~
```

Note that the **BlockingQueue**s are more robust and easier to use; you do not have to catch or handle **IOException**s, and setting up the communication channel is very straightforward.

Exercise 31

```
//: concurrency/E31_DiningPhilosophers2.java
// {Args: 5 0 deadlock 5}
/******************** Exercise 31 ************************
 * Change DeadlockingDiningPhilosophers.java so that when a
 * philosopher is done with their chopsticks, they drop them
 * into a bin. When a philosopher wants to eat, they take
 * the next two available chopsticks from the bin. Does this
 * eliminate the possibility of deadlock? Can you
 * reintroduce deadlock by simply reducing the number of
 * available chopsticks?
 *********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class Chopstick {
  private final int id;
  private boolean taken;
  public Chopstick(int ident) { id = ident; }
  public synchronized
  void take() throws InterruptedException {
    while(taken)
      wait();
    taken = true;
  }
  public synchronized void drop() {
    taken = false;
    notifyAll();
  }
  public String toString() { return "Chopstick " + id; }
}

class ChopstickQueue
extends LinkedBlockingQueue<Chopstick>{}

class ChopstickBin {
  private ChopstickQueue bin = new ChopstickQueue();
  public Chopstick get() throws InterruptedException {
    return bin.take();
  }
  public void
  put(Chopstick stick) throws InterruptedException {
    bin.put(stick);
```

```
  }
}

class Philosopher implements Runnable {
  private static Random rand = new Random(47);
  private final int id;
  private final int ponderFactor;
  private ChopstickBin bin;
  private void pause() throws InterruptedException {
    if(ponderFactor == 0) return;
    TimeUnit.MILLISECONDS.sleep(
      rand.nextInt(ponderFactor * 250));
  }
  public Philosopher(ChopstickBin bin, int ident,
    int ponder) {
    this.bin = bin;
    id = ident;
    ponderFactor = ponder;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        print(this + " " + "thinking");
        pause();
        // Get one chopstick from the bin
        Chopstick c1 = bin.get();
        print(this + " has " + c1 +
          " waiting for another one");
        // Get another chopstick from bin
        Chopstick c2 = bin.get();
        print(this + " has " + c2);
        print(this + " eating");
        pause();
        // Put the chopsticks back in bin.
        bin.put(c1);
        bin.put(c2);
      }
    } catch(InterruptedException e) {
      print(this + " " + "exiting via interrupt");
    }
  }
  public String toString() { return "Philosopher " + id; }
}

public class E31_DiningPhilosophers2 {
  public static void main(String[] args) throws Exception {
    if(args.length < 3) {
```

```
      System.err.println("usage:\n" +
        "java E31_DiningPhilosophers2 " +
        "numberOfPhilosophers ponderFactor deadlock " +
        "timeout\n" + "A nonzero ponderFactor will " +
        "generate a random sleep time during think().\n" +
        "If deadlock is not the string " +
        "'deadlock', the program will not deadlock.\n" +
        "A nonzero timeout will stop the program after " +
        "that number of seconds.");
      System.exit(1);
    }
    ChopstickBin bin = new ChopstickBin();
    int size = Integer.parseInt(args[0]);
    int ponder = Integer.parseInt(args[1]);
    for(int i = 0; i < size; i++)
      bin.put(new Chopstick(i));
    // One additional chopstick guarantees that at least
    // one philosopher can eat without blocking.
    if(!args[2].equals("deadlock"))
      bin.put(new Chopstick(size));
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < size; i++)
      exec.execute(new Philosopher(bin, i, ponder));
    if(args.length == 4)
      TimeUnit.SECONDS.sleep(Integer.parseInt(args[3]));
    else {
      print("Press 'ENTER' to quit");
      System.in.read();
    }
    exec.shutdownNow();
  }
} /* Output: (Sample)
Philosopher 1 thinking
Philosopher 1 has Chopstick 0 waiting for another one
Philosopher 1 has Chopstick 1
Philosopher 1 eating
Philosopher 1 thinking
...
Philosopher 2 has Chopstick 1 waiting for another one
Philosopher 2 has Chopstick 2
Philosopher 2 eating
Philosopher 2 thinking
Philosopher 2 has Chopstick 1 waiting for another one
Philosopher 2 has Chopstick 2
Philosopher 2 eating
Philosopher 2 thinking
Philosopher 2 has Chopstick 1 waiting for another one
```

```
Philosopher 2 has Chopstick 2
Philosopher 2 eating
Philosopher 2 thinking
Philosopher 2 has Chopstick 1 waiting for another one
Philosopher 2 has Chopstick 2
Philosopher 2 eating
Philosopher 4 has Chopstick 1 waiting for another one
Philosopher 2 thinking
Philosopher 2 has Chopstick 2 waiting for another one
Philosopher 3 exiting via interrupt
Philosopher 0 exiting via interrupt
Philosopher 1 exiting via interrupt
Philosopher 4 exiting via interrupt
Philosopher 2 exiting via interrupt
*///:~
```

We use **ChopstickQueue** to create the **ChopstickBin**, so it gains all of **LinkedBlockingQueue**'s control logic. Instead of picking up the left and right chopsticks, philosophers take them from a common bin, an activity we suspend when too few chopsticks remain. The results are the same, with the flaw that an equal number of chopsticks and philosophers can quickly deadlock, but a single extra chopstick prevents this.

Exercise 32

```
//: concurrency/E32_OrnamentalGarden3.java
/******************** Exercise 32 ************************
 * Use a CountDownLatch to solve the problem of correlating
 * the results from the Entrances in OrnamentalGarden.java.
 * Remove the unnecessary code from the new version of the
 * example.
 *********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class Entrance3 implements Runnable {
  private final CountDownLatch latch;
  private static Count count = new Count();
  private static List<Entrance3> entrances =
    new ArrayList<Entrance3>();
  private int number;
  private final int id;
  private static volatile boolean canceled;
```

```
  public static void cancel() { canceled = true; }
  public Entrance3(CountDownLatch ltc, int id) {
    latch = ltc;
    this.id = id;
    entrances.add(this);
  }
  public void run() {
    while(!canceled) {
      synchronized(this) { ++number; }
      print(this + " Total: " + count.increment());
      try {
        TimeUnit.MILLISECONDS.sleep(100);
      } catch(InterruptedException e) {
        print("sleep interrupted");
      }
    }
    latch.countDown();
    print("Stopping " + this);
  }
  public synchronized int getValue() { return number; }
  public String toString() {
    return "Entrance " + id + ": " + getValue();
  }
  public static int getTotalCount() {
    return count.value();
  }
  public static int sumEntrances() {
    int sum = 0;
    for(Entrance3 entrance : entrances)
      sum += entrance.getValue();
    return sum;
  }
}

public class E32_OrnamentalGarden3 {
  public static void main(String[] args) throws Exception {
    // All must share a single CountDownLatch object:
    CountDownLatch latch = new CountDownLatch(5);
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < 5; i++)
      exec.execute(new Entrance3(latch, i));
    TimeUnit.SECONDS.sleep(3);
    Entrance3.cancel();
    exec.shutdown();
    latch.await();  // Wait for results
    print("Total: " + Entrance3.getTotalCount());
    print("Sum of Entrances: " + Entrance3.sumEntrances());
```

```
  }
} /* Output: (Sample)
Entrance 1: 1 Total: 2
Entrance 2: 1 Total: 3
Entrance 3: 1 Total: 4
Entrance 4: 1 Total: 5
Entrance 0: 1 Total: 1
Entrance 1: 2 Total: 6
Entrance 2: 2 Total: 7
Entrance 3: 2 Total: 8
Entrance 4: 2 Total: 9
Entrance 0: 2 Total: 10
Entrance 1: 3 Total: 11
Entrance 2: 3 Total: 12
...
Terminating Entrance 2: 30
Terminating Entrance 0: 30
Terminating Entrance 3: 30
Terminating Entrance 4: 30
Terminating Entrance 1: 30
Total: 150
Sum of Entrances: 150
*///:~
```

CountDownLatch shortens and clarifies the code considerably.

Exercise 33

```
//: concurrency/E33_GreenhouseController.java
// {Args: 5000}
/******************** Exercise 33 ************************
 * Modify GreenhouseScheduler.java to use a
 * DelayQueue instead of a ScheduledExecutor.
 *********************************************************/
package concurrency;
import java.util.concurrent.*;
import static java.util.concurrent.TimeUnit.*;
import static net.mindview.util.Print.*;

abstract class Event implements Runnable, Delayed {
  protected final long delayTime;
  private long trigger;
  public Event(long delayTime) {
    this.delayTime = delayTime;
  }
  public long getDelay(TimeUnit unit) {
```

```
    return unit.convert(
      trigger - System.nanoTime(), NANOSECONDS);
  }
  public int compareTo(Delayed arg) {
    Event that = (Event)arg;
    if(trigger < that.trigger) return -1;
    if(trigger > that.trigger) return 1;
    return 0;
  }
  public void start() { // Allows restarting
    trigger = System.nanoTime() +
      NANOSECONDS.convert(delayTime, MILLISECONDS);
  }
  public abstract void run();
}

class Controller implements Runnable {
  private DelayQueue<Event> q;
  public Controller(DelayQueue<Event> q) {
    this.q = q;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        Event event = q.take();
        print(event);
        event.run();
      }
    } catch(InterruptedException e) {
      // Acceptable way to exit
    }
    print("Finished Controller");
  }
  public void addEvent(Event c) {
    c.start();
    q.put(c);
  }
}

class GreenhouseControls extends Controller {
  public GreenhouseControls(DelayQueue<Event> q) {
    super(q);
  }
  private boolean light;
  public class LightOn extends Event {
    public LightOn(long delayTime) { super(delayTime); }
    public void run() {
```

```
      // Put hardware control code here to
      // physically turn on the light.
      light = true;
    }
    public String toString() { return "Light is on"; }
  }
  public class LightOff extends Event {
    public LightOff(long delayTime) { super(delayTime); }
    public void run() {
      // Put hardware control code here to
      // physically turn off the light.
      light = false;
    }
    public String toString() { return "Light is off"; }
  }
  private boolean water;
  public class WaterOn extends Event {
    public WaterOn(long delayTime) { super(delayTime); }
    public void run() {
      // Put hardware control code here.
      water = true;
    }
    public String toString() {
      return "Greenhouse water is on";
    }
  }
  public class WaterOff extends Event {
    public WaterOff(long delayTime) { super(delayTime); }
    public void run() {
      // Put hardware control code here.
      water = false;
    }
    public String toString() {
      return "Greenhouse water is off";
    }
  }
  private String thermostat = "Day";
  public class ThermostatNight extends Event {
    public ThermostatNight(long delayTime) {
      super(delayTime);
    }
    public void run() {
      // Put hardware control code here.
      thermostat = "Night";
    }
    public String toString() {
      return "Thermostat on night setting";
```

```
    }
  }
  public class ThermostatDay extends Event {
    public ThermostatDay(long delayTime) {
      super(delayTime);
    }
    public void run() {
      // Put hardware control code here.
      thermostat = "Day";
    }
    public String toString() {
      return "Thermostat on day setting";
    }
  }
  // An example of an action() that inserts a
  // new one of itself into the event list:
  public class Bell extends Event {
    public Bell(long delayTime) { super(delayTime); }
    public void run() {
      addEvent(new Bell(delayTime));
    }
    public String toString() { return "Bing!"; }
  }
  public class Restart extends Event {
    private Event[] eventList;
    public Restart(long delayTime, Event[] eventList) {
      super(delayTime);
      this.eventList = eventList;
      for(Event e : eventList)
        addEvent(e);
    }
    public void run() {
      for(Event e : eventList) {
        addEvent(e);
      }
      addEvent(this);
    }
    public String toString() {
      return "Restarting system";
    }
  }
  public static class Terminate extends Event {
    private ExecutorService exec;
    public Terminate(long delayTime, ExecutorService e) {
      super(delayTime);
      exec = e;
    }
```

```
    public void run() { exec.shutdownNow(); }
    public String toString() { return "Terminating";  }
  }
}

public class E33_GreenhouseController {
  public static void main(String[] args) {
    ExecutorService exec = Executors.newCachedThreadPool();
    DelayQueue<Event> queue = new DelayQueue<Event>();
    GreenhouseControls gc = new GreenhouseControls(queue);
    gc.addEvent(gc.new Bell(900));
    Event[] eventList = {
      gc.new ThermostatNight(0),
      gc.new LightOn(200),
      gc.new LightOff(400),
      gc.new WaterOn(600),
      gc.new WaterOff(800),
      gc.new ThermostatDay(1400)
    };
    gc.addEvent(gc.new Restart(2000, eventList));
    if(args.length == 1)
      gc.addEvent(
        new GreenhouseControls.Terminate(
          new Integer(args[0]), exec));
    exec.execute(gc);
  }
} /* Output:
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Bing!
Thermostat on day setting
Bing!
Restarting system
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Bing!
Greenhouse water is off
Thermostat on day setting
Bing!
Restarting system
Thermostat on night setting
Light is on
```

```
Light is off
Bing!
Greenhouse water is on
Greenhouse water is off
Terminating
Finished Controller
*///:~
```

This program combines **concurrency/DelayQueueDemo.java** and **innerclasses/GreenhouseController.java** from *TIJ4*. Using **DelayQueue** instead of an ordinary **ArrayList** cleans up the code a bit.

Exercise 34

```
//: concurrency/E34_ExchangerDemo2.java
/******************* Exercise 34 ************************
 * Modify ExchangerDemo.java to use your own class instead
 * of Fat.
 ********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import net.mindview.util.*;

class ExchangerProducer<T> implements Runnable {
  private Generator<T> generator;
  private Exchanger<List<T>> exchanger;
  private List<T> holder;
  ExchangerProducer(Exchanger<List<T>> exchg,
  Generator<T> gen, List<T> holder) {
    exchanger = exchg;
    generator = gen;
    this.holder = holder;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        for(int i = 0; i < E34_ExchangerDemo2.size; i++)
          holder.add(generator.next());
        // Exchange full for empty:
        holder = exchanger.exchange(holder);
      }
    } catch(InterruptedException e) {
      // OK to terminate this way.
    }
  }
```

```
}

class ExchangerConsumer<T> implements Runnable {
  private Exchanger<List<T>> exchanger;
  private List<T> holder;
  private volatile T value;
  ExchangerConsumer(Exchanger<List<T>> ex, List<T> holder){
    exchanger = ex;
    this.holder = holder;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        holder = exchanger.exchange(holder);
        for(T x : holder) {
          value = x; // Fetch out value
          holder.remove(x); // OK for CopyOnWriteArrayList
        }
      }
    } catch(InterruptedException e) {
      // OK to terminate this way.
    }
    System.out.println("Final value: " + value);
  }
}

public class E34_ExchangerDemo2 {
  static int size = 10;
  static int delay = 5; // Seconds
  public static void main(String[] args) throws Exception {
    if(args.length > 0)
      size = new Integer(args[0]);
    if(args.length > 1)
      delay = new Integer(args[1]);
    ExecutorService exec = Executors.newCachedThreadPool();
    Exchanger<List<Integer>> xc =
      new Exchanger<List<Integer>>();
    List<Integer>
      producerList = new CopyOnWriteArrayList<Integer>(),
      consumerList = new CopyOnWriteArrayList<Integer>();
    exec.execute(new ExchangerProducer<Integer>(xc,
      new CountingGenerator.Integer(), producerList));
    exec.execute(
      new ExchangerConsumer<Integer>(xc, consumerList));
    TimeUnit.SECONDS.sleep(delay);
    exec.shutdownNow();
  }
```

```
} /* Output: (Sample)
Final value: 2903359
*///:~
```

We replace **Fat** with **Integer**, and use **CountingGenerator.Integer** to generate numbers.

Exercise 35

```
//: concurrency/E35_WebClientServerSimulation.java
// {Args: 8}
/******************** Exercise 35 ************************
 * Modify BankTellerSimulation.java so that it represents
 * Web clients making requests of a fixed number of servers.
 * The goal is to determine the load that the group of
 * servers can handle.
 *********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class WebClient {
  private final int serviceTime;
  public WebClient(int tm) { serviceTime = tm; }
  public int getServiceTime() { return serviceTime; }
  public String toString() {
    return "[" + serviceTime + "]";
  }
}

class WebClientLine extends ArrayBlockingQueue<WebClient> {
  public WebClientLine(int maxLineSize) {
    super(maxLineSize);
  }
  public String toString() {
    if(this.size() == 0)
      return "[Empty]";
    StringBuilder result = new StringBuilder();
    for(WebClient client : this)
      result.append(client);
    return result.toString();
  }
}

class WebClientGenerator implements Runnable {
```

```
  private WebClientLine clients;
  volatile int loadFactor = 1;  // Start with one client/sec
  private static Random rand = new Random(47);
  public WebClientGenerator(WebClientLine cq) {
    clients = cq;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        clients.put(new WebClient(rand.nextInt(1000)));
        TimeUnit.MILLISECONDS.sleep(1000 / loadFactor);
      }
    } catch(InterruptedException e) {
      print("WebClientGenerator interrupted");
    }
    print("WebClientGenerator terminating");
  }
}

class Server implements Runnable {
  private static int counter;
  private final int id = counter++;
  private WebClientLine clients;
  public Server(WebClientLine cq) { clients = cq; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        WebClient client = clients.take();
        TimeUnit.MILLISECONDS.sleep(
          client.getServiceTime());
      }
    } catch(InterruptedException e) {
      print(this + "interrupted");
    }
    print(this + "terminating");
  }
  public String toString() { return "Server " + id + " "; }
  public String shortString() { return "T" + id; }
}

class SimulationManager implements Runnable {
  private ExecutorService exec;
  private WebClientGenerator gen;
  private WebClientLine clients;
  private Queue<Server> servers =
    new LinkedList<Server>();
  private int adjustmentPeriod;
```

```
  // Indicates whether the queue is stable
  private boolean stable = true;
  private int prevSize;
  public SimulationManager(ExecutorService e,
    WebClientGenerator gen, WebClientLine clients,
    int adjustmentPeriod, int n) {
    exec = e;
    this.gen = gen;
    this.clients = clients;
    this.adjustmentPeriod = adjustmentPeriod;
    // Start with 'n' servers:
    for(int i= 0; i < n; i++) {
      Server server = new Server(clients);
      exec.execute(server);
      servers.add(server);
    }
  }
  public void adjustLoadFactor() {
    // This is actually a control system. By adjusting
    // the numbers, you can reveal stability issues in
    // the control mechanism.
    // If line is stable, increase the 'load factor':
    if(clients.size() > prevSize) {
      if(stable)  // Was stable previous time
        stable = false;
      else if(!stable) {  // Not stable for a second time
        print("Peak load factor: ~" + gen.loadFactor);
        exec.shutdownNow();
      }
    } else {
      print("New load factor: " + ++gen.loadFactor);
      stable = true;
    }
    prevSize = clients.size();
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        TimeUnit.MILLISECONDS.sleep(adjustmentPeriod);
        System.out.print(clients + " { ");
        for(Server server : servers)
          printnb(server.shortString() + " ");
        print("}");
        adjustLoadFactor();
      }
    } catch(InterruptedException e) {
      print(this + "interrupted");
```

```
    }
    System.out.println(this + "terminating");
  }
  public String toString() { return "SimulationManager "; }
}

public class E35_WebClientServerSimulation {
  static final int MAX_LINE_SIZE = 50;
  static final int NUM_OF_SERVERS = 3;
  static final int ADJUSTMENT_PERIOD = 1000;
  public static void main(String[] args) throws Exception {
    ExecutorService exec = Executors.newCachedThreadPool();
    WebClientLine clients =
      new WebClientLine(MAX_LINE_SIZE);
    WebClientGenerator g = new WebClientGenerator(clients);
    exec.execute(g);
    exec.execute(new SimulationManager(
      exec, g, clients, ADJUSTMENT_PERIOD, NUM_OF_SERVERS));
    if(args.length > 0) // Optional argument
      TimeUnit.SECONDS.sleep(new Integer(args[0]));
    else {
      System.out.println("Press 'ENTER' to quit");
      System.in.read();
    }
    exec.shutdownNow();
  }
} /* Output: (Sample)
[Empty] { T0 T1 T2 }
New load factor: 2
[Empty] { T0 T1 T2 }
New load factor: 3
[Empty] { T0 T1 T2 }
New load factor: 4
[Empty] { T0 T1 T2 }
New load factor: 5
[Empty] { T0 T1 T2 }
New load factor: 6
[258] { T0 T1 T2 }
[704][383] { T0 T1 T2 }
Peak load factor: ~6
Server 2 interrupted
Server 2 terminating
SimulationManager terminating
Server 1 interrupted
Server 0 interrupted
WebClientGenerator interrupted
Server 1 terminating
```

```
Server 0 terminating
WebClientGenerator terminating
*///:~
```

This is like an inversion of **BankTellerSimulation.java**, but instead of lacking a predefined number of tellers, the number of servers is fixed. We set the *load* (clients per second) by increasing the **loadFactor** until the number of clients in the queue exceeds our chosen limit. The inference engine is simple: if the client line is growing then the program has reached peak **loadFactor**. However, you must run the simulation for awhile to find this **loadFactor**.

Try changing parameters like **serviceTime** of clients to simulate different scenarios.

Exercise 36

```
//: concurrency/restaurant2/E36_RestaurantWithQueues2.java
// {Args: 5}
/******************** Exercise 36 ************************
 * Modify RestaurantWithQueues.java so there's one
 * OrderTicket object per table. Change order to
 * orderTicket, and add a Table class, with multiple
 * Customers per table.
 *********************************************************/
package concurrency.restaurant2;
import enumerated.menu.*;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

// This is consisted of many orders, and there's one ticket
// per table:
class OrderTicket {
  private static int counter;
  private final int id = counter++;
  private final Table table;
  private final List<Order> orders =
    Collections.synchronizedList(new LinkedList<Order>());
  public OrderTicket(Table table) { this.table = table; }
  public WaitPerson getWaitPerson() {
    return table.getWaitPerson();
  }
  public void placeOrder(Customer cust, Food food) {
    Order order = new Order(cust, food);
    orders.add(order);
    order.setOrderTicket(this);
```

```
  }
  public List<Order> getOrders() { return orders; }
  public String toString() {
    StringBuilder sb = new StringBuilder(
      "Order Ticket: " + id + " for: " + table + "\n");
    synchronized(orders) {
      for(Order order : orders)
        sb.append(order.toString() + "\n");
    }
    // Prune away the last added 'new-line' character
    return sb.substring(0, sb.length() - 1).toString();
  }
}

class Table implements Runnable {
  private static int counter;
  private final int id = counter++;
  private final WaitPerson waitPerson;
  private final List<Customer> customers;
  private final OrderTicket orderTicket =
    new OrderTicket(this);
  private final CyclicBarrier barrier;
  private final int nCustomers;
  private final ExecutorService exec;
  public Table(WaitPerson waitPerson, int nCustomers,
    ExecutorService e) {
    this.waitPerson = waitPerson;
    customers = Collections.synchronizedList(
      new LinkedList<Customer>());
    this.nCustomers = nCustomers;
    exec = e;
    barrier = new CyclicBarrier(nCustomers + 1,
      new Runnable() {
        public void run() {
          print(orderTicket.toString());
        }
      });
  }
  public WaitPerson getWaitPerson() { return waitPerson; }
  public void placeOrder(Customer cust, Food food) {
    orderTicket.placeOrder(cust, food);
  }
  public void run() {
    Customer customer;
    for(int i = 0; i < nCustomers; i++) {
      customers.add(customer = new Customer(this, barrier));
      exec.execute(customer);
```

```
    }
    try {
      barrier.await();
    } catch(InterruptedException ie) {
      print(this + " interrupted");
      return;
    } catch(BrokenBarrierException e) {
      throw new RuntimeException(e);
    }
    waitPerson.placeOrderTicket(orderTicket);
  }
  public String toString() {
    StringBuilder sb = new StringBuilder(
      "Table: " + id + " served by: " + waitPerson + "\n");
    synchronized(customers) {
      for(Customer customer : customers)
        sb.append(customer.toString() + "\n");
    }
    return sb.substring(0, sb.length() - 1).toString();
  }
}

// This is part of an order ticket (given to the chef):
class Order {
  private static int counter;
  private final int id;
  private volatile OrderTicket orderTicket;
  private final Customer customer;
  private final Food food;
  public Order(Customer cust, Food f) {
    customer = cust;
    food = f;
    synchronized(Order.class) { id = counter++; }
  }
  void setOrderTicket(OrderTicket orderTicket) {
    this.orderTicket = orderTicket;
  }
  public OrderTicket getOrderTicket() {
    return orderTicket;
  }
  public Food item() { return food; }
  public Customer getCustomer() { return customer; }
  public String toString() {
    return "Order: " + id + " item: " + food +
      " for: " + customer;
  }
}
```

```
// This is what comes back from the chef:
class Plate {
  private final Order order;
  private final Food food;
  public Plate(Order ord, Food f) {
    order = ord;
    food = f;
  }
  public Order getOrder() { return order; }
  public Food getFood() { return food; }
  public String toString() { return food.toString(); }
}

class Customer implements Runnable {
  private static int counter;
  private final int id;
  private final CyclicBarrier barrier;
  private final Table table;
  private int nPlates;  // Number of plates ordered
  public Customer(Table table, CyclicBarrier barrier) {
    this.table = table;
    this.barrier = barrier;
    synchronized(Customer.class) { id = counter++; }
  }
  // Only one course at a time can be received:
  private final SynchronousQueue<Plate> placeSetting =
    new SynchronousQueue<Plate>();
  public void
  deliver(Plate p) throws InterruptedException {
    // Only blocks if customer is still
    // eating the previous course:
    placeSetting.put(p);
  }
  public void run() {
    // First place an order:
    for(Course course : Course.values()) {
      Food food = course.randomSelection();
      table.placeOrder(this, food);
      ++nPlates;
    }
    try {
      barrier.await();
    } catch(InterruptedException ie) {
      print(this + " interrupted while ordering meal");
      return;
    } catch(BrokenBarrierException e) {
```

```
      throw new RuntimeException(e);
    }
    // Now wait for each ordered plate:
    for(int i = 0; i < nPlates; i++)
      try {
        // Blocks until course has been delivered:
        print(this + "eating " + placeSetting.take());
      } catch(InterruptedException e) {
        print(this + "waiting for meal interrupted");
        return;
      }
    print(this + "finished meal, leaving");
  }
  public String toString() {
    return "Customer " + id + " ";
  }
}

class WaitPerson implements Runnable {
  private static int counter;
  private final int id = counter++;
  private final Restaurant restaurant;
  final BlockingQueue<Plate> filledOrders =
    new LinkedBlockingQueue<Plate>();
  public WaitPerson(Restaurant rest) { restaurant = rest; }
  public void placeOrderTicket(OrderTicket orderTicket) {
    try {
      // Shouldn't actually block because this is
      // a LinkedBlockingQueue with no size limit:
      restaurant.orderTickets.put(orderTicket);
    } catch(InterruptedException e) {
      print(this + " placeOrderTicket interrupted");
    }
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until a course is ready
        Plate plate = filledOrders.take();
        print(this + "received " + plate +
          " delivering to " +
          plate.getOrder().getCustomer());
        plate.getOrder().getCustomer().deliver(plate);
      }
    } catch(InterruptedException e) {
      print(this + " interrupted");
    }
```

```
    print(this + " off duty");
  }
  public String toString() {
    return "WaitPerson " + id + " ";
  }
}

class Chef implements Runnable {
  private static int counter;
  private final int id = counter++;
  private final Restaurant restaurant;
  private static Random rand = new Random(47);
  public Chef(Restaurant rest) { restaurant = rest; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until an order ticket appears:
        OrderTicket orderTicket =
          restaurant.orderTickets.take();
        List<Order> orders = orderTicket.getOrders();
        synchronized(orders) {
          for(Order order : orders) {
            Food requestedItem = order.item();
            // Time to prepare order:
            TimeUnit.MILLISECONDS.sleep(rand.nextInt(500));
            Plate plate = new Plate(order, requestedItem);
            order.getOrderTicket().getWaitPerson().
              filledOrders.put(plate);
          }
        }
      }
    } catch(InterruptedException e) {
      print(this + " interrupted");
    }
    print(this + " off duty");
  }
  public String toString() { return "Chef " + id + " "; }
}

class Restaurant implements Runnable {
  private List<WaitPerson> waitPersons =
    new ArrayList<WaitPerson>();
  private List<Chef> chefs = new ArrayList<Chef>();
  private ExecutorService exec;
  private static Random rand = new Random(47);
  final BlockingQueue<OrderTicket> orderTickets =
    new LinkedBlockingQueue<OrderTicket>();
```

```
  public Restaurant(ExecutorService e, int nWaitPersons,
    int nChefs) {
    exec = e;
    for(int i = 0; i < nWaitPersons; i++) {
      WaitPerson waitPerson = new WaitPerson(this);
      waitPersons.add(waitPerson);
      exec.execute(waitPerson);
    }
    for(int i = 0; i < nChefs; i++) {
      Chef chef = new Chef(this);
      chefs.add(chef);
      exec.execute(chef);
    }
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // A new group of customers arrive; assign a
        // WaitPerson:
        WaitPerson wp = waitPersons.get(
          rand.nextInt(waitPersons.size()));
        int nCustomers = rand.nextInt(4) + 1;
        Table t = new Table(wp, nCustomers, exec);
        exec.execute(t);
        TimeUnit.MILLISECONDS.sleep(400 * nCustomers);
      }
    } catch(InterruptedException e) {
      print("Restaurant interrupted");
    }
    print("Restaurant closing");
  }
}

public class E36_RestaurantWithQueues2 {
  public static void main(String[] args) throws Exception {
    ExecutorService exec = Executors.newCachedThreadPool();
    Restaurant restaurant = new Restaurant(exec, 5, 2);
    exec.execute(restaurant);
    if(args.length > 0) // Optional argument
      TimeUnit.SECONDS.sleep(new Integer(args[0]));
    else {
      print("Press 'ENTER' to quit");
      System.in.read();
    }
    exec.shutdownNow();
  }
} /* Output: (Sample)
```

```
Order Ticket: 0 for: Table: 0 served by: WaitPerson 3
Customer 0
Customer 1
Order: 0 item: SPRING_ROLLS for: Customer 1
Order: 1 item: SPRING_ROLLS for: Customer 0
Order: 2 item: VINDALOO for: Customer 0
Order: 3 item: BURRITO for: Customer 1
Order: 4 item: GELATO for: Customer 0
Order: 5 item: CREME_CARAMEL for: Customer 1
Order: 6 item: BLACK_COFFEE for: Customer 0
Order: 7 item: TEA for: Customer 1
WaitPerson 3 received SPRING_ROLLS delivering to Customer 1
Customer 1 eating SPRING_ROLLS
WaitPerson 3 received SPRING_ROLLS delivering to Customer 0
Customer 0 eating SPRING_ROLLS
WaitPerson 3 received VINDALOO delivering to Customer 0
Customer 0 eating VINDALOO
Order Ticket: 1 for: Table: 1 served by: WaitPerson 3
Customer 2
Order: 8 item: SALAD for: Customer 2
Order: 9 item: BURRITO for: Customer 2
Order: 10 item: FRUIT for: Customer 2
Order: 11 item: TEA for: Customer 2
WaitPerson 3 received BURRITO delivering to Customer 1
Customer 1 eating BURRITO
...
Customer 12 waiting for meal interrupted
WaitPerson 3  interrupted
Customer 9 waiting for meal interrupted
Customer 13 waiting for meal interrupted
WaitPerson 3  off duty
WaitPerson 1  interrupted
Customer 8 waiting for meal interrupted
WaitPerson 2  interrupted
Customer 7 waiting for meal interrupted
WaitPerson 0  interrupted
Restaurant interrupted
WaitPerson 1  off duty
WaitPerson 2  off duty
Chef 0  interrupted
Customer 14 waiting for meal interrupted
Customer 10 waiting for meal interrupted
WaitPerson 0  off duty
WaitPerson 4  interrupted
Restaurant closing
Chef 0  off duty
WaitPerson 4  off duty
```

```
Chef 1  interrupted
Customer 11 waiting for meal interrupted
Chef 1  off duty
*///:~
```

The exercise is vague, allowing many approaches. Our solution works well, but try experimenting with others. For example, to increase restaurant productivity, eliminate the inefficiency of repeated roundtrips to deliver each plate as soon as possible, perhaps grouping food types for delivery.

The **Table** class creates a group order. Each customer at a table orders, and when they are all done, **Table** passes the order ticket to a waitperson. A **CyclicBarrier** synchronizes the tasks.

The **OrderTicket** consists of many orders, and each chef handles one ticket.

Restaurant produces **Table** tasks, so customers come in groups and occupy a particular table.

The rest of the program follows the original version.

Exercise 37

```
//: concurrency/E37_CarBuilder2.java
/******************** Exercise 37 ************************
 * Modify CarBuilder.java to add another stage to the
 * carbuilding process, which adds the exhaust system,
 * body, and fenders. As with the second stage, assume these
 * processes can be performed simultaneously by robots.
 *********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class Car2 {
  private final int id;
  private boolean
    engine = false, driveTrain = false, wheels = false,
    exhaustSystem = false, body = false, fender = false;
  public Car2(int idn)  { id = idn; }
  public Car2()  { id = -1; }
  public synchronized int getId() { return id; }
  public synchronized void addEngine() { engine = true; }
  public synchronized void addDriveTrain() {
    driveTrain = true;
```

```
  }
  public synchronized void addWheels() { wheels = true; }
  public synchronized void addExhaustSystem() {
    exhaustSystem = true;
  }
  public synchronized void addBody() { body = true; }
  public synchronized void addFender() { fender = true; }
  public synchronized String toString() {
    return "Car " + id + " [" + " engine: " + engine
      + " driveTrain: " + driveTrain
      + " wheels: " + wheels
      + " exhaust system: " + exhaustSystem
      + " body: " + body
      + " fender: " + fender + "]";
  }
}

class CarQueue extends LinkedBlockingQueue<Car2> {}

class ChassisBuilder implements Runnable {
  private CarQueue carQueue;
  private int counter = 0;
  public ChassisBuilder(CarQueue cq) { carQueue = cq; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        TimeUnit.MILLISECONDS.sleep(500);
        // Make chassis:
        Car2 c = new Car2(counter++);
        print("ChassisBuilder created " + c);
        // Insert into queue
        carQueue.put(c);
      }
    } catch(InterruptedException e) {
      print("Interrupted: ChassisBuilder");
    }
    print("ChassisBuilder off");
  }
}

class Assembler implements Runnable {
  private CarQueue chassisQueue, finishingQueue;
  private Car2 car;
  private CyclicBarrier barrier = new CyclicBarrier(4);
  private RobotPool robotPool;
  public Assembler(CarQueue cq, CarQueue fq, RobotPool rp){
    chassisQueue = cq;
```

```
    finishingQueue = fq;
    robotPool = rp;
  }
  public Car2 car() { return car; }
  public CyclicBarrier barrier() { return barrier; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until chassis is available:
        car = chassisQueue.take();
        // Hire robots to perform work (second stage):
        robotPool.hire(EngineRobot.class, this);
        robotPool.hire(DriveTrainRobot.class, this);
        robotPool.hire(WheelRobot.class, this);
        barrier.await(); // Until the robots finish
        // Hire robots to perform work (third stage):
        robotPool.hire(ExhaustSystemRobot.class, this);
        robotPool.hire(BodyRobot.class, this);
        robotPool.hire(FenderRobot.class, this);
        barrier.await();
        // Put car into finishingQueue for further work
        finishingQueue.put(car);
      }
    } catch(InterruptedException e) {
      print("Exiting Assembler via interrupt");
    } catch(BrokenBarrierException e) {
      // This one we want to know about
      throw new RuntimeException(e);
    }
    print("Assembler off");
  }
}

class Reporter implements Runnable {
  private CarQueue carQueue;
  public Reporter(CarQueue cq) { carQueue = cq; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        print(carQueue.take());
      }
    } catch(InterruptedException e) {
      print("Exiting Reporter via interrupt");
    }
    print("Reporter off");
  }
}
```

```
abstract class Robot implements Runnable {
  private RobotPool pool;
  public Robot(RobotPool p) { pool = p; }
  protected Assembler assembler;
  public Robot assignAssembler(Assembler assembler) {
    this.assembler = assembler;
    return this;
  }
  private boolean engage = false;
  public synchronized void engage() {
    engage = true;
    notifyAll();
  }
  // The part of run() that's different for each robot:
  abstract protected void performService();
  public void run() {
    try {
      powerDown(); // Wait until needed
      while(!Thread.interrupted()) {
        performService();
        assembler.barrier().await(); // Synchronize
        // We're done with that job...
        powerDown();
      }
    } catch(InterruptedException e) {
      print("Exiting " + this + " via interrupt");
    } catch(BrokenBarrierException e) {
      // This one we want to know about
      throw new RuntimeException(e);
    }
    print(this + " off");
  }
  private synchronized void
  powerDown() throws InterruptedException {
    engage = false;
    assembler = null; // Disconnect from the Assembler
    // Put ourselves back in the available pool:
    pool.release(this);
    while(engage == false)  // Power down
      wait();
  }
  public String toString() { return getClass().getName(); }
}

class EngineRobot extends Robot {
  public EngineRobot(RobotPool pool) { super(pool); }
```

```
  protected void performService() {
    print(this + " installing engine");
    assembler.car().addEngine();
  }
}

class DriveTrainRobot extends Robot {
  public DriveTrainRobot(RobotPool pool) { super(pool); }
  protected void performService() {
    print(this + " installing DriveTrain");
    assembler.car().addDriveTrain();
  }
}

class WheelRobot extends Robot {
  public WheelRobot(RobotPool pool) { super(pool); }
  protected void performService() {
    print(this + " installing Wheels");
    assembler.car().addWheels();
  }
}

class ExhaustSystemRobot extends Robot {
  public ExhaustSystemRobot(RobotPool pool) { super(pool); }
  protected void performService() {
    print(this + " installing exhausting system");
    assembler.car().addExhaustSystem();
  }
}

class BodyRobot extends Robot {
  public BodyRobot(RobotPool pool) { super(pool); }
  protected void performService() {
    print(this + " installing body");
    assembler.car().addBody();
  }
}

class FenderRobot extends Robot {
  public FenderRobot(RobotPool pool) { super(pool); }
  protected void performService() {
    print(this + " installing fender");
    assembler.car().addFender();
  }
}

class RobotPool {
```

```
  // Quietly prevents identical entries:
  private Set<Robot> pool = new HashSet<Robot>();
  public synchronized void add(Robot r) {
    pool.add(r);
    notifyAll();
  }
  public synchronized void
  hire(Class<? extends Robot> robotType, Assembler d)
  throws InterruptedException {
    for(Robot r : pool)
      if(r.getClass().equals(robotType)) {
        pool.remove(r);
        r.assignAssembler(d);
        r.engage(); // Power it up to do the task
        return;
      }
    wait(); // None available
    hire(robotType, d); // Try again, recursively
  }
  public synchronized void release(Robot r) { add(r); }
}

public class E37_CarBuilder2 {
  public static void main(String[] args) throws Exception {
    CarQueue chassisQueue = new CarQueue(),
             finishingQueue = new CarQueue();
    ExecutorService exec = Executors.newCachedThreadPool();
    RobotPool robotPool = new RobotPool();
    exec.execute(new EngineRobot(robotPool));
    exec.execute(new DriveTrainRobot(robotPool));
    exec.execute(new WheelRobot(robotPool));
    exec.execute(new ExhaustSystemRobot(robotPool));
    exec.execute(new BodyRobot(robotPool));
    exec.execute(new FenderRobot(robotPool));
    exec.execute(new Assembler(
      chassisQueue, finishingQueue, robotPool));
    exec.execute(new Reporter(finishingQueue));
    // Start everything running by producing chassis:
    exec.execute(new ChassisBuilder(chassisQueue));
    TimeUnit.SECONDS.sleep(7);
    exec.shutdownNow();
  }
} /* (Execute to see output) *///:~
```

Exercise 38

```
//: concurrency/E38_HouseBuilder.java
/******************** Exercise 38 ************************
 * Using the approach in CarBuilder.java, model the
 * house-building story that was given in this chapter.
 *********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

class House {
  private final int id;
  private boolean
    steel = false, concreteForms = false,
    concreteFoundation = false, plumbing = false,
    concreteSlab = false, framing = false;
  public House(int idn)  { id = idn; }
  public House()  { id = -1; }
  public synchronized int getId() { return id; }
  public synchronized void laySteel() { steel = true; }
  public synchronized void buildConcreteForms() {
    concreteForms = true;
  }
  public synchronized void pourConcreteFoundation() {
    concreteFoundation = true;
  }
  public synchronized void addPlumbing() {
    plumbing = true;
  }
  public synchronized void pourConcreteSlab() {
    concreteSlab = true;
  }
  public synchronized void startFraming() {
    framing = true;
  }
  public synchronized String toString() {
    return "House " + id + " [" + " steel: " + steel
      + " concreteForms: " + concreteForms
      + " concreteFoundation: " + concreteFoundation
      + " plumbing: " + plumbing
      + " concreteSlab: " + concreteSlab
      + " framing: " + framing + "]";
  }
}
```

```
class HouseQueue extends LinkedBlockingQueue<House> {}

class FootingsDigger implements Runnable {
  private HouseQueue houseQueue;
  private int counter = 0;
  public FootingsDigger(HouseQueue cq) { houseQueue = cq; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        TimeUnit.MILLISECONDS.sleep(500);
        // Dig footings:
        House h = new House(counter++);
        print("FootingsDigger created " + h);
        houseQueue.put(h);
      }
    } catch(InterruptedException e) {
      print("Interrupted: FootingsDigger");
    }
    print("FootingsDigger off");
  }
}

class HouseBuilder implements Runnable {
  private HouseQueue footingsQueue, finishingQueue;
  private House house;
  private CyclicBarrier barrier1 = new CyclicBarrier(3);
  private CyclicBarrier barrier2 = new CyclicBarrier(2);
  private TeamPool teamPool;
  private boolean secondStage = true;
  public HouseBuilder(HouseQueue hq, HouseQueue fq,
    TeamPool tp){
    footingsQueue = hq;
    finishingQueue = fq;
    teamPool = tp;
  }
  public House house() { return house; }
  public CyclicBarrier barrier() {
    return secondStage ? barrier1 : barrier2;
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        // Blocks until footings are dug:
        house = footingsQueue.take();
        teamPool.hire(SteelTeam.class, this);
        teamPool.hire(ConcreteFormsTeam.class, this);
```

```
        barrier1.await();
        secondStage = false;
        teamPool.hire(ConcreteFoundationTeam.class, this);
        barrier2.await();
        teamPool.hire(PlumbingTeam.class, this);
        barrier2.await();
        teamPool.hire(ConcreteSlabTeam.class, this);
        barrier2.await();
        teamPool.hire(FramingTeam.class, this);
        barrier2.await();
        finishingQueue.put(house);
        secondStage = true;
      }
    } catch(InterruptedException e) {
      print("Exiting HouseBuilder via interrupt");
    } catch(BrokenBarrierException e) {
      throw new RuntimeException(e);
    }
    print("HouseBuilder off");
  }
}

class Reporter2 implements Runnable {
  private HouseQueue houseQueue;
  public Reporter2(HouseQueue hq) { houseQueue = hq; }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        print(houseQueue.take());
      }
    } catch(InterruptedException e) {
      print("Exiting Reporter via interrupt");
    }
    print("Reporter off");
  }
}

abstract class Team implements Runnable {
  private TeamPool pool;
  public Team(TeamPool p) { pool = p; }
  protected HouseBuilder hb;
  public Team assignHouseBuilder(HouseBuilder hb) {
    this.hb = hb;
    return this;
  }
  private boolean engage = false;
  public synchronized void engage() {
```

```
    engage = true;
    notifyAll();
  }
  abstract protected void performService();
  public void run() {
    try {
      rest();
      while(!Thread.interrupted()) {
        performService();
        hb.barrier().await();
        rest();
      }
    } catch(InterruptedException e) {
      print("Exiting " + this + " via interrupt");
    } catch(BrokenBarrierException e) {
      throw new RuntimeException(e);
    }
    print(this + " off");
  }
  private synchronized void
  rest() throws InterruptedException {
    engage = false;
    hb = null;
    // Put ourselves back in the available pool:
    pool.release(this);
    while(engage == false)
      wait();
  }
  public String toString() { return getClass().getName(); }
}

class SteelTeam extends Team {
  public SteelTeam(TeamPool pool) { super(pool); }
  protected void performService() {
    print(this + " laying steel");
    hb.house().laySteel();
  }
}

class ConcreteFormsTeam extends Team {
  public ConcreteFormsTeam(TeamPool pool) { super(pool); }
  protected void performService() {
    print(this + " building concrete forms");
    hb.house().buildConcreteForms();
  }
}
```

```
class ConcreteFoundationTeam extends Team {
  public ConcreteFoundationTeam(TeamPool pool) {
    super(pool);
  }
  protected void performService() {
    print(this + " pouring concrete foundation");
    hb.house().pourConcreteFoundation();
  }
}

class PlumbingTeam extends Team {
  public PlumbingTeam(TeamPool pool) { super(pool); }
  protected void performService() {
    print(this + " add plumbing");
    hb.house().addPlumbing();
  }
}

class ConcreteSlabTeam extends Team {
  public ConcreteSlabTeam(TeamPool pool) { super(pool); }
  protected void performService() {
    print(this + " pour concrete slab");
    hb.house().pourConcreteSlab();
  }
}

class FramingTeam extends Team {
  public FramingTeam(TeamPool pool) { super(pool); }
  protected void performService() {
    print(this + " start framing");
    hb.house().startFraming();
  }
}

class TeamPool {
  private Set<Team> pool = new HashSet<Team>();
  public synchronized void add(Team t) {
    pool.add(t);
    notifyAll();
  }
  public synchronized void
  hire(Class<? extends Team> teamType, HouseBuilder hb)
  throws InterruptedException {
    for(Team t : pool)
      if(t.getClass().equals(teamType)) {
        pool.remove(t);
        t.assignHouseBuilder(hb);
```

```
        t.engage();
        return;
      }
    wait();
    hire(teamType, hb);
  }
  public void release(Team t) { add(t); }
}

public class E38_HouseBuilder {
  public static void main(String[] args) throws Exception {
    HouseQueue footingsQueue = new HouseQueue(),
               finishingQueue = new HouseQueue();
    ExecutorService exec = Executors.newCachedThreadPool();
    TeamPool teamPool = new TeamPool();
    exec.execute(new SteelTeam(teamPool));
    exec.execute(new ConcreteFormsTeam(teamPool));
    exec.execute(new ConcreteFoundationTeam(teamPool));
    exec.execute(new PlumbingTeam(teamPool));
    exec.execute(new ConcreteSlabTeam(teamPool));
    exec.execute(new FramingTeam(teamPool));
    exec.execute(new HouseBuilder(
      footingsQueue, finishingQueue, teamPool));
    exec.execute(new Reporter2(finishingQueue));
    exec.execute(new FootingsDigger(footingsQueue));
    TimeUnit.SECONDS.sleep(7);
    exec.shutdownNow();
  }
} /* (Execute to see output) *///:~
```

Only the names of the entities change: **Car** -> **House**, **Assembler** -> **HouseBuilder**, **Robot** -> **Team**, and **RobotPool** -> **TeamPool**. The program logic remains basically the same.

Exercise 39

```
//: concurrency/E39_FastSimulation2.java
// {RunByHand}
/******************* Exercise 39 ***********************
 * Does FastSimulation.java make reasonable assumptions?
 * Change the array to ordinary ints instead of
 * AtomicInteger and use Lock mutexes. Compare
 * performance between the two versions of the program.
 *******************************************************/
package concurrency;
import java.util.concurrent.*;
```

```
import java.util.concurrent.atomic.*;
import java.util.concurrent.locks.*;
import java.util.*;
import static net.mindview.util.Print.*;

public class E39_FastSimulation2 {
  static final int N_ELEMENTS = 10000;
  static final int N_GENES = 30;
  static final int N_EVOLVERS = 50;
  // Variant 1 (optimistic locking):
  static final AtomicInteger[][] GRID =
    new AtomicInteger[N_ELEMENTS][N_GENES];
  // Variant 2 (explicit locking):
  static final int[][] grid =
    new int[N_ELEMENTS][N_GENES];
  static final ReentrantLock[] lock =
    new ReentrantLock[N_ELEMENTS];
  // Counts the number of evolutions using 'variant 1':
  static final AtomicInteger counter1 = new AtomicInteger();
  // Counts the number of evolutions using 'variant 2':
  static final AtomicInteger counter2 = new AtomicInteger();
  static Random rand = new Random(47);
  static class Evolver1 implements Runnable {
    public void run() {
      while(!Thread.interrupted()) {
        // Randomly select an element to work on:
        int element = rand.nextInt(N_ELEMENTS);
        for(int i = 0; i < N_GENES; i++) {
          int previous = element - 1;
          if(previous < 0) previous = N_ELEMENTS - 1;
          int next = element + 1;
          if(next >= N_ELEMENTS) next = 0;
          int oldvalue = GRID[element][i].get();
          int newvalue = oldvalue +
            GRID[previous][i].get() + GRID[next][i].get();
          newvalue /= 3;
          if(!GRID[element][i]
            .compareAndSet(oldvalue, newvalue)) {
            // Some backup action...
          }
        }
        counter1.incrementAndGet();
      }
    }
  }
  static class Evolver2 implements Runnable {
    public void run() {
```

```
      while(!Thread.interrupted()) {
        // Randomly select an element to work on:
        int element = rand.nextInt(N_ELEMENTS);
        // Lock the whole row:
        lock[element].lock();
        try {
          for(int i = 0; i < N_GENES; i++) {
            int previous = element - 1;
            if(previous < 0) previous = N_ELEMENTS - 1;
            int next = element + 1;
            if(next >= N_ELEMENTS) next = 0;
            int newvalue = grid[element][i] +
              grid[previous][i] + grid[next][i];
            newvalue /= 3;
            grid[element][i] = newvalue;
          }
        } finally { lock[element].unlock(); }
        counter2.incrementAndGet();
      }
    }
  }
  public static void main(String[] args) throws Exception {
    ExecutorService exec = Executors.newCachedThreadPool();
    for(int i = 0; i < N_ELEMENTS; i++)
      for(int j = 0; j < N_GENES; j++)
        GRID[i][j] = new AtomicInteger(rand.nextInt(1000));
    for(int i = 0; i < N_ELEMENTS; i++)
      for(int j = 0; j < N_GENES; j++)
        grid[i][j] = rand.nextInt(1000);
    for(int i = 0; i < N_ELEMENTS; i++)
      lock[i] = new ReentrantLock();
    for(int i = 0; i < N_EVOLVERS; i++) {
      exec.execute(new Evolver1());
      exec.execute(new Evolver2());
    }
    TimeUnit.SECONDS.sleep(5);
    exec.shutdownNow();
    print("Variant 1: " + counter1.get());
    print("Variant 2: " + counter2.get());
  }
} /* (Execute to see output) *///:~
```

The assumptions are reasonable. Here is our output:

```
Variant 1: 3376049
Variant 2: 1362210
```

Experiment with different locking schemes; for example, lock something smaller than the whole row. Risk optimistic locking *only* if the cost of a failure is low. Otherwise, you lose the advantage gained by avoiding explicit locks in the first place.

Exercise 40

```
//: concurrency/E40_MapComparisons2.java
// {Args: 1 10 10} (Fast verification check during build)
/******************** Exercise 40 ************************
 * Following the example of ReaderWriterList.java, create
 * a ReaderWriterMap using a HashMap. Investigate its
 * performance by modifying MapComparisons.java. How does
 * it compare to a synchronized HashMap and a
 * ConcurrentHashMap?
 *********************************************************/
package concurrency;
import java.util.concurrent.locks.*;
import java.util.*;
import net.mindview.util.*;

class ReaderWriterMap<K,V> extends AbstractMap<K,V> {
  private HashMap<K,V> lockedMap;
  private ReentrantReadWriteLock lock =
    new ReentrantReadWriteLock(true);
  public ReaderWriterMap(Generator<K> genK, int size,
    V initialValue) {
    lockedMap = new HashMap<K,V>(
      MapData.map(genK, initialValue, size));
  }
  public V put(K key, V value) {
    Lock wlock = lock.writeLock();
    wlock.lock();
    try {
      return lockedMap.put(key, value);
    } finally {
      wlock.unlock();
    }
  }
  public V get(Object key) {
    Lock rlock = lock.readLock();
    rlock.lock();
    try {
      // Comment out if you would like to trace how many
      // readers are acquiring the lock simultaneously:
```

```
//    if(lock.getReadLockCount() > 1)
//      print(lock.getReadLockCount());
      return lockedMap.get(key);
    } finally {
      rlock.unlock();
    }
  }
  // Dummy implementation
  public Set<Map.Entry<K,V>> entrySet() { return null; }
}

class ReaderWriterMapTest extends MapTest {
  Map<Integer,Integer> containerInitializer() {
    return new ReaderWriterMap<Integer,Integer>(
      new CountingGenerator.Integer(), containerSize,
      1);
  }
  ReaderWriterMapTest(int nReaders, int nWriters) {
    super("ReaderWriterMap", nReaders, nWriters);
  }
}

public class E40_MapComparisons2 {
  public static void main(String[] args) {
    Tester.initMain(args);
    new SynchronizedHashMapTest(10, 0);
    new SynchronizedHashMapTest(9, 1);
    new SynchronizedHashMapTest(5, 5);
    new ConcurrentHashMapTest(10, 0);
    new ConcurrentHashMapTest(9, 1);
    new ConcurrentHashMapTest(5, 5);
    new ReaderWriterMapTest(10, 0);
    new ReaderWriterMapTest(9, 1);
    new ReaderWriterMapTest(5, 5);
    Thread.yield();
    Tester.exec.shutdown();
  }
} /* Output: (Sample)
Type                            Read time     Write time
Synched HashMap 10r 0w            4034312              0
Synched HashMap 9r 1w             9416558        1530920
readTime + writeTime =           10947478
Synched HashMap 5r 5w              119009        2369854
readTime + writeTime =            2488863
ConcurrentHashMap 10r 0w          2892266              0
ConcurrentHashMap 9r 1w           7266285         689193
readTime + writeTime =            7955478
```

```
ConcurrentHashMap 5r 5w             132698        3805790
readTime + writeTime =             3938488
ReaderWriterMap 10r 0w             6972675              0
ReaderWriterMap 9r 1w             13310350        2396394
readTime + writeTime =            15706744
ReaderWriterMap 5r 5w               204774       10277563
readTime + writeTime =            10482337
*///:~
```

ReaderWriterMap extends **AbstractMap** to make it amenable to the **MapComparisons.java** program. You see from the output that it doesn't perform well here, but sometimes it may be your best option.

Exercise 41

```
//: concurrency/E41_ActiveObjectDemo2.java
/******************** Exercise 41 ************************
 * Add a message handler to ActiveObjectDemo.java that
 * has no return value, and call this within main().
 ***********************************************************/
package concurrency;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.Print.*;

public class E41_ActiveObjectDemo2 {
  private ExecutorService ex =
    Executors.newSingleThreadExecutor();
  private Random rand = new Random(47);
  // Insert a random delay to produce the effect
  // of a calculation time:
  private void pause(int factor) {
    try {
      TimeUnit.MILLISECONDS.sleep(
        100 + rand.nextInt(factor));
    } catch(InterruptedException e) {
      print("sleep() interrupted");
    }
  }
  public Future<Integer>
  calculateInt(final int x, final int y) {
    return ex.submit(new Callable<Integer>() {
      public Integer call() {
        print("starting " + x + " + " + y);
        pause(500);
        return x + y;
```

```
      }
    });
  }
  public Future<Float>
  calculateFloat(final float x, final float y) {
    return ex.submit(new Callable<Float>() {
      public Float call() {
        print("starting " + x + " + " + y);
        pause(2000);
        return x + y;
      }
    });
  }
  // Message handler without a return value:
  public void printDocument(final String s) {
    ex.execute(new Runnable() {
      public void run() {
        print("printing document " + s);
        pause(1000);
        print("document " + s + " printed");
      }
    });
  }
  public void shutdown() { ex.shutdown(); }
  public static void main(String[] args) {
    E41_ActiveObjectDemo2 d1 = new E41_ActiveObjectDemo2();
    // Prevents ConcurrentModificationException:
    List<Future<?>> results =
      new CopyOnWriteArrayList<Future<?>>();
    for(float f = 0.0f; f < 1.0f; f += 0.2f)
      results.add(d1.calculateFloat(f, f));
    for(int i = 0; i < 5; i++) {
      results.add(d1.calculateInt(i, i));
      d1.printDocument("DOC_" + i);
    }
    print("All asynch calls made");
    while(results.size() > 0) {
      for(Future<?> f : results)
        if(f.isDone()) {
          try {
            print(f.get());
          } catch(Exception e) {
            throw new RuntimeException(e);
          }
          results.remove(f);
        }
    }
```

```
    d1.shutdown();
  }
} /* Output: (Sample)
All asynch calls made
starting 0.0 + 0.0
0.0
starting 0.2 + 0.2
starting 0.4 + 0.4
0.4
starting 0.6 + 0.6
0.8
starting 0.8 + 0.8
1.2
starting 0 + 0
1.6
printing document DOC_0
0
document DOC_0 printed
starting 1 + 1
printing document DOC_1
2
document DOC_1 printed
starting 2 + 2
printing document DOC_2
4
document DOC_2 printed
starting 3 + 3
printing document DOC_3
6
document DOC_3 printed
starting 4 + 4
printing document DOC_4
8
document DOC_4 printed
*///:~
```

Exercise 42

```
//: concurrency/E42_ActiveObjectWaxOMatic.java
/******************** Exercise 42 ************************
 * Modify WaxOMatic.java so that it implements active
 * objects.
 ***********************************************************/
package concurrency;
import java.util.concurrent.*;
import static net.mindview.util.Print.*;
```

```
class ActiveCar {
  private ExecutorService ex =
    Executors.newSingleThreadExecutor();
  private enum Action { WAX, BUFF }
  private Action lastAction = Action.BUFF;
  private boolean waxOn;
  public void wax() {
    try {
      ex.execute(waxingTask);
    } catch(RejectedExecutionException e) {}
  }
  public void buff() {
    try {
      ex.execute(buffingTask);
    } catch(RejectedExecutionException e) {}
  }
  public void shutdown() { ex.shutdown(); }
  private static void pause(int sleep_time) {
    try {
      TimeUnit.MILLISECONDS.sleep(sleep_time);
    } catch(InterruptedException e) {
      print("sleep() interrupted");
    }
  }
  private class WaxingTask implements Runnable {
    public void run() {
      if(lastAction != Action.WAX) {
        printnb("Wax On! ");
        pause(200);
        waxOn = true;
        lastAction = Action.WAX;
      }
    }
  }
  private final WaxingTask waxingTask = new WaxingTask();
  private class BuffingTask implements Runnable {
    public void run() {
      if(lastAction != Action.BUFF) {
        printnb("Wax Off! ");
        pause(200);
        waxOn = false;
        lastAction = Action.BUFF;
      }
    }
  }
  private final BuffingTask buffingTask = new BuffingTask();
```

```
}

class WaxCar implements Runnable {
  private final ActiveCar car;
  public WaxCar(ActiveCar c) { car = c; }
  public void run() { car.wax(); }
}

class BuffCar implements Runnable {
  private final ActiveCar car;
  public BuffCar(ActiveCar c) { car = c; }
  public void run() { car.buff(); }
}

public class E42_ActiveObjectWaxOMatic {
  public static void main(String[] args) throws Exception {
    ActiveCar car = new ActiveCar();
    ScheduledExecutorService exec =
      Executors.newScheduledThreadPool(2);
    exec.scheduleAtFixedRate(
      new BuffCar(car), 0, 200, TimeUnit.MILLISECONDS);
    exec.scheduleAtFixedRate(
      new WaxCar(car), 0, 200, TimeUnit.MILLISECONDS);
    TimeUnit.SECONDS.sleep(5); // Run for a while...
    exec.shutdownNow(); // Interrupt all tasks
    car.shutdown();
  }
} /* Output: (Sample)
Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On!
Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off!
Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On!
Wax Off! Wax On! Wax Off! Wax On! Wax Off! Wax On! Wax Off!
*///:~
```

Here, the *Active Object* **Car** accepts two kinds of messages: *wax* and *buff*. **WaxCar** and **BuffCar** serialize their message requests to run in parallel, so they need no locks.

The output shows that even after **WaxCar** and **BuffCar** "stop," **ActiveCar** still handles the previously posted messages. New messages arrive faster than the active object can handle them, so we take some precautions to avoid running out of memory.

You *must* catch the **RejectedExecutionException** inside **wax( )** and **buff( )** (the methods accepting messages), because the client programmer needs to interrogate the return value to know if the message has been accepted or rejected.

Graphical User Interfaces

Exercise 1

```
//: gui/E01_HelloSwing2.java
/******************** Exercise 01 ************************
 * Modify HelloSwing.java to prove to yourself that the
 * application will not close without the call to
 * setDefaultCloseOperation().
 *********************************************************/
package gui;
import javax.swing.*;

public class E01_HelloSwing2 {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Hello Swing");
    frame.setSize(300, 100);
    frame.setVisible(true);
  }
} ///:~
```

After closing the main application window, you will need to manually terminate the corresponding OS process.

Exercise 2

```
//: gui/E02_DynamicHelloLabel.java
/******************** Exercise 02 ************************
 * Modify HelloLabel.java to show that label addition is
 * dynamic, by adding a random number of labels.
 *********************************************************/
package gui;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.concurrent.*;

class DynamicHelloLabel extends JFrame {
```

```
  private static Random rnd = new Random(47);
  JLabel[] labels;
  DynamicHelloLabel() {
    super("Hello Label");
    setLayout(new FlowLayout());
    int numOfLabels = rnd.nextInt(10) + 1;
    labels = new JLabel[numOfLabels];
    for(int i = 0; i < numOfLabels; i++)
      add(labels[i] = new JLabel("label: " + i));
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300, 100);
    setVisible(true);
  }
}

public class E02_DynamicHelloLabel {
  static DynamicHelloLabel dhl;
  public static void main(String[] args) throws Exception {
    SwingUtilities.invokeLater(new Runnable() {
      public void run() { dhl = new DynamicHelloLabel(); }
    });
    TimeUnit.SECONDS.sleep(2);
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        for(int i = 0; i< dhl.labels.length; i++)
          dhl.labels[i].setText("LABEL: " + i);
      }
    });
  }
} ///:~
```

See the *Making a button* section for an explanation of why we change the default layout manager.

Exercise 3

```
//: gui/E03_SubmitSwingProgram2.java
/******************* Exercise 03 ************************
 * Modify SubmitSwingProgram.java so that it uses
 * SwingConsole.
 ********************************************************/
package gui;
import javax.swing.*;
import java.util.concurrent.*;
import static net.mindview.util.SwingConsole.*;
```

```
class SubmitSwingProgram2 extends JFrame {
  JLabel label;
  public SubmitSwingProgram2() {
    label = new JLabel("A Label");
    add(label);
  }
}

public class E03_SubmitSwingProgram2 {
  static SubmitSwingProgram2 ssp;
  public static void main(String[] args) throws Exception {
    run(ssp = new SubmitSwingProgram2(), 300, 100);
    TimeUnit.SECONDS.sleep(2);
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        ssp.label.setText("Hey! This is Different!");
      }
    });
  }
} ///:~
```

Exercise 4

```
//: gui/E04_Button1UsingDefaultLayout.java
/******************** Exercise 04 ************************
 * Verify that without the setLayout() call in
 * Button1.java, only one button will appear in the
 * resulting program.
 *********************************************************/
package gui;
import javax.swing.*;
import static net.mindview.util.SwingConsole.*;

public class E04_Button1UsingDefaultLayout extends JFrame {
  private JButton
    b1 = new JButton("Button 1"),
    b2 = new JButton("Button 2");
  public E04_Button1UsingDefaultLayout () {
    add(b1);
    add(b2);
  }
  public static void main(String[] args) {
    run(new E04_Button1UsingDefaultLayout(), 200, 100);
  }
} ///:~
```

Exercise 5

```
//: gui/E05_Button3.java
/******************** Exercise 05 ************************
 * Create an application using the SwingConsole class.
 * Include one text field and three buttons. When you press
 * each button, make different text appear in the text
 * field.
 *********************************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

public class E05_Button3 extends JFrame {
  private JButton
    b1 = new JButton("Button 1"),
    b2 = new JButton("Button 2"),
    b3 = new JButton("Button 3");
  private JTextField txt = new JTextField(10);
  private ActionListener bl = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      String name = ((JButton)e.getSource()).getText();
      txt.setText(name);
    }
  };
  public E05_Button3() {
    b1.addActionListener(bl);
    b2.addActionListener(bl);
    b3.addActionListener(bl);
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(b3);
    add(txt);
  }
  public static void main(String[] args) {
    run(new E05_Button3(), 200, 150);
  }
} ///:~
```

The text that appears in the **JTextField** is created from the label on each button.

Exercise 6

```
//: gui/E06_TestRegularExpression2.java
/******************** Exercise 6 ************************
 * Turn strings/TestRegularExpression.java into an
 * interactive Swing program that allows you to put an
 * input string in one JTextArea and a regular expression in
 * a JTextField. The results should be displayed in a second
 * JTextArea.
 ********************************************************/
package gui;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.regex.*;
import static net.mindview.util.SwingConsole.*;

public class E06_TestRegularExpression2 extends JFrame {
  private JTextArea input = new JTextArea(3, 60),
                    output = new JTextArea(3, 60);
  private JTextField expression = new JTextField(40);
  private JButton match = new JButton("Match");
  private JLabel inputL = new JLabel("Input"),
                 outputL = new JLabel("Output"),
                 expressionL = new JLabel("Expression");
  private JPanel panel1 = new JPanel(),
                 panel2 = new JPanel(),
                 panel3 = new JPanel();
  public E06_TestRegularExpression2() {
    setLayout(new GridLayout(3,1));
    panel1.add(inputL);
    panel1.add(new JScrollPane(input));
    add(panel1);
    panel2.add(expressionL);
    panel2.add(expression);
    panel2.add(match);
    add(panel2);
    panel3.add(outputL);
    panel3.add(new JScrollPane(output));
    add(panel3);
    match.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String inputS = input.getText();
        String regEx = expression.getText();
        Pattern p = Pattern.compile(regEx);
        Matcher m = p.matcher(inputS);
```

```
        String outputS = "";
        while(m.find()) {
          outputS += "Match \"" + m.group() +
            "\" at positions " + m.start() + "-" +
            (m.end() - 1) + '\n';
        }
        output.setText(outputS);
      }
    });
  }
  public static void main(String[] args) {
    run(new E06_TestRegularExpression2(), 700, 400);
  }
} ///:~
```

Here we introduce a new layout manager called **GridLayout**. When building a more complex UI, you can segregate it into different areas, then apply a panel to represent each one. We use a relatively simple UI here to demonstrate this technique.

Exercise 7

```
//: gui/E07_AllAction.java
/****************** Exercise 7 *****************
 * Create an application using SwingConsole, and
 * add all the Swing components that have an
 * addActionListener() method. (Look these up in
 * the JDK documentation from http://java.sun.com.
 * Hint: Search for addActionListener() using the
 * index.) Capture their events and display an
 * appropriate message for each inside a text field.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class AllAction extends JFrame {
  // The JMenuItem's derivatives JCheckBoxMenuItem, JMenu &
  // JRadioButtonMenuItem will not be shown separately:
  JMenuItem mi = new JMenuItem("Menu Item");
  JTextField txt = new JTextField(30);
  JButton b1 = new JButton("Button 1");
  JComboBox jcb = new JComboBox(new String[]{
    "Elements", "To", "Place", "In", "Combobox"
```

```
  });
  JFileChooser jfc = new JFileChooser(".");
  public AllAction() {
    setLayout(new FlowLayout());
    add(mi);
    add(txt);
    add(b1);
    add(jcb);
    add(jfc);
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        txt.setText("Button pressed");
      }
    });
    txt.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(
          null,
          "JTexfField ActionListener fired",
          "information",
          JOptionPane.INFORMATION_MESSAGE);
      }
    });
    jcb.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        txt.setText("JComboBox selected: " +
          jcb.getSelectedItem());
      }
    });
    jfc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        txt.setText(
          "FileChooser ActionListener fired: " +
          jfc.getSelectedFile());
      }
    });
    mi.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        txt.setText("JMenuItem selected");
      }
    });
    new Timer(5000, new ActionListener() {
      int i = 0;
      public void actionPerformed(ActionEvent e) {
        txt.setText("Timer Ticked " + i++);
      }
    }).start();
```

```
  }
}

public class E07_AllAction {
  public static void main(String args[]) {
    run(new AllAction(), 550, 400);
  }
} ///:~
```

You can fire the **ActionListener** for the **JTextField** only by pressing return (unless you can find an alternative in the documentation).

This example is a fun way to show what these components can do, and reminds us just how good Swing is.

The **javax.swing.Timer** and **java.util.Timer** are very different. We create the first without capturing the reference, but the garbage collector never collects it, as it normally would any object whose reference is unused. The constructor typically stores a reference to the object somewhere so that it won't be garbage-collected.

Exercise 8

```
//: gui/E08_Cursors.java
/****************** Exercise 8 *****************
 * Almost every Swing component is derived from
 * Component, which has a setCursor() method.
 * Look this up in the JDK documentation. Create
 * an application and change the cursor to one of
 * the stock cursors in the Cursor class.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import static net.mindview.util.SwingConsole.*;

class Cursors extends JFrame {
  JTextField txt = new JTextField(10);
  JButton b1 = new JButton("Button 1");
  Cursor hand =
    Cursor.getPredefinedCursor(
      Cursor.HAND_CURSOR);
  public Cursors() {
    setLayout(new FlowLayout());
    add(txt);
    txt.setCursor(hand);
```

```
    add(b1);
    setCursor(hand);
  }
}

public class E08_Cursors {
  public static void main(String args[]) {
    run(new Cursors(), 200, 100);
  }
} ///:~
```

The constants in the **Cursor** class are **int**s, not **Cursor** objects, so you must use either the **Cursor** constructor or **getPredefinedCursor( )** method to generate a **Cursor** object from the **int** (this seems a bit strange, but cursors are system resources which, on Windows anyway, can be used up easily—this may be the reason for the design).

Note that if you don't also set the cursor in the **JTextField**, it goes back to the default cursor.

Exercise 9

```
//: gui/E09_ShowMethods.java
/****************** Exercise 9 *****************
 * Starting with ShowAddListeners.java, create a
 * program with the full functionality of
 * typeinfo.ShowMethods.java.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.reflect.*;
import java.util.*;
import java.util.regex.*;
import static net.mindview.util.SwingConsole.*;

class ShowMethodsClean extends JFrame {
  private static Pattern qualifier =
    Pattern.compile("\\w+\\.");
  String[] n;
  JTextField
    name = new JTextField(25),
    searchFor = new JTextField(25);
  JTextArea results = new JTextArea(40, 65);
  JScrollPane scrollPane = new JScrollPane(results);
```

```
  class NameL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      String nm = name.getText().trim();
      if(nm.length() == 0) {
        results.setText("No match");
        return;
      }
      Class<?> cl;
      try {
        cl = Class.forName(nm);
      } catch(ClassNotFoundException ex) {
        results.setText("No match");
        return;
      }
      Method[] m = cl.getMethods();
      Constructor<?>[] ctor = cl.getConstructors();
      // Convert to an array of cleaned Strings:
      n = new String[m.length + ctor.length];
      for(int i = 0; i < m.length; i++)
        n[i] = m[i].toString();
      for(int i = 0; i < ctor.length; i++)
        n[i + m.length] = ctor[i].toString();
      reDisplay();
    }
  }
  void reDisplay() {
    results.setText("");
    if(searchFor.getText().trim().length() == 0)
      // Include everything:
      for(String s : n)
        results.append(s + "\n");
//      OR:
//      results.append(
//        qualifier.matcher(s).replaceAll("") + "\n");
    else {
      // Include only methods that have ALL the
      // words listed in searchFor:
      java.util.List<String> lookFor = Arrays.asList(
        searchFor.getText().split("\\s+"));
      for(String s : n) {
        Iterator<String> it = lookFor.iterator();
        boolean include = true;
        while(it.hasNext())
          if(s.indexOf(it.next()) == -1)
            include = false;
        if(include == true)
          results.append(s + "\n");
```

```
//        OR:
//        results.append(
//          qualifier.matcher(s).replaceAll("") + "\n");
      }
    }
    // Force the scrollpane back to the top:
    scrollPane.getViewport().setViewPosition(
      new Point(0, 0));
  }
  public ShowMethodsClean() {
    name.addActionListener(new NameL());
    // There is no need to parse the class file again when
    // only the search conditions have changed:
    searchFor.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        reDisplay();
      }
    });
    JPanel top1 = new JPanel();
    top1.add(new JLabel(
      "Qualified.class.name (press ENTER):"));
    top1.add(name);
    JPanel top2 = new JPanel();
    top2.add(new JLabel(
      "Words to search for (optional):"));
    top2.add(searchFor);
    JPanel top = new JPanel(new GridLayout(2,1));
    top.add(top1);
    top.add(top2);
    add(BorderLayout.NORTH, top);
    add(scrollPane);
  }
}

public class E09_ShowMethods {
  public static void main(String args[]) {
    run(new ShowMethodsClean(), 600, 400);
  }
} ///:~
```

We add a second **JTextField** called **searchFor** to hold a list of words to find. Then we copy the methods and constructors from the **Class** object into **n** (the **String**s array of all possible names), and call **reDisplay( )** to choose methods to display the names. If there are no words in the **searchFor** field, the program shows everything. If there are words in the **searchFor** field, the **String** from that field must be broken up into words using **String.split( )**. We put each

word into **java.util.List lookFor**, then use the flag **include** to examine each method/constructor, making sure it contains *all* the words in **lookFor**, and place it into the **JTextArea**. Remember, the search is case-sensitive.

The **JTextArea** can *appear* blank when elements in the output have just scrolled off the page, so we move the scroll-pane back to the top whenever **JTextArea** data is modified. You must modify the **Viewport**, and not the **JScrollPane** directly, as you can see in the line of code at the end of **reDisplay( )**.

Optionally, you can remove the pertinent comments from the code to strip off the qualifiers, and resize the main application window if the UI displays awkwardly.

Exercise 10

```
//: gui/E10_TypeableButton.java
/****************** Exercise 10 ******************
 * Create an application using SwingConsole, with a
 * JButton and a JTextField. Write and attach the
 * appropriate listener so that if the button has the
 * focus, characters typed into it will appear in the
 * JTextField.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class TypeableButton extends JFrame {
  JTextField txt = new JTextField(10);
  JButton b = new JButton("Button 1");
  public TypeableButton() {
    setLayout(new FlowLayout());
    add(txt);
    add(b);
    b.addKeyListener(new KeyAdapter() {
      public void keyTyped(KeyEvent e) {
        txt.setText(txt.getText() + e.getKeyChar());
      }
    });
  }
}
```

```
public class E10_TypeableButton {
  public static void main(String args[]) {
    run(new TypeableButton(), 200, 100);
  }
} ///:~
```

Exercise 11

```
//: gui/E11_RandomColorButton.java
/****************** Exercise 11 *****************
 * Inherit a new type of button from JButton. Each
 * time you press this button, it should change its
 * color to a randomly selected value. See
 * ColorBoxes.java (later in this chapter) for an
 * example of how to generate a random color value.
 ************************************************/
package gui;
import java.util.Random;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class RandomColorButton extends JButton {
  private static final Color[] colors = {
    Color.black, Color.blue, Color.cyan,
    Color.darkGray, Color.gray, Color.green,
    Color.lightGray, Color.magenta,
    Color.orange, Color.pink, Color.red,
    Color.white, Color.yellow
  };
  private static Random rnd = new Random(47);
  private static final Color newColor() {
    return colors[rnd.nextInt(colors.length)];
  }
  public RandomColorButton(String text) {
    super(text);
    setBackground(newColor());
    addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        setBackground(newColor());
      }
    });
  }
}
```

```
public class E11_RandomColorButton extends JFrame {
  E11_RandomColorButton() {
    add(new RandomColorButton("Random Colors"));
  }
  public static void main(String args[]) {
    run(new E11_RandomColorButton(), 150, 75);
  }
} ///:~
```

Exercise 12

```
//: gui/E12_NewEvent.java
/****************** Exercise 12 *****************
 * Monitor a new type of event in TrackEvent.java
 * by adding the new event-handling code. You'll
 * need to discover on your own the type of event
 * that you want to monitor.
 ***********************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import static net.mindview.util.SwingConsole.*;

class TrackEvent extends JFrame {
  private HashMap<String,JTextField> h =
    new HashMap<String,JTextField>();
  private String[] event = {
    "actionPerformed", "stateChanged",
    "focusGained", "focusLost", "keyPressed",
    "keyReleased", "keyTyped", "mouseClicked",
    "mouseEntered", "mouseExited", "mousePressed",
    "mouseReleased", "mouseDragged", "mouseMoved"
  };
  private MyButton
    b1 = new MyButton(Color.BLUE, "test1"),
    b2 = new MyButton(Color.RED, "test2");
  class MyButton extends JButton {
    void report(String field, String msg) {
      h.get(field).setText(msg);
    }
    ActionListener al = new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        report("actionPerformed", e.paramString());
```

```
    }
  };
  ChangeListener cl = new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
      report("stateChanged", e.toString());
    }
  };
  FocusListener fl = new FocusListener() {
    public void focusGained(FocusEvent e) {
      report("focusGained", e.paramString());
    }
    public void focusLost(FocusEvent e) {
      report("focusLost", e.paramString());
    }
  };
  KeyListener kl = new KeyListener() {
    public void keyPressed(KeyEvent e) {
      report("keyPressed", e.paramString());
    }
    public void keyReleased(KeyEvent e) {
      report("keyReleased", e.paramString());
    }
    public void keyTyped(KeyEvent e) {
      report("keyTyped", e.paramString());
    }
  };
  MouseListener ml = new MouseListener() {
    public void mouseClicked(MouseEvent e) {
      report("mouseClicked", e.paramString());
    }
    public void mouseEntered(MouseEvent e) {
      report("mouseEntered", e.paramString());
    }
    public void mouseExited(MouseEvent e) {
      report("mouseExited", e.paramString());
    }
    public void mousePressed(MouseEvent e) {
      report("mousePressed", e.paramString());
    }
    public void mouseReleased(MouseEvent e) {
      report("mouseReleased", e.paramString());
    }
  };
  MouseMotionListener mml = new MouseMotionListener() {
    public void mouseDragged(MouseEvent e) {
      report("mouseDragged", e.paramString());
    }
```

```
      public void mouseMoved(MouseEvent e) {
        report("mouseMoved", e.paramString());
      }
    };
    public MyButton(Color color, String label) {
      super(label);
      setBackground(color);
      addActionListener(al);
      addChangeListener(cl);
      addFocusListener(fl);
      addKeyListener(kl);
      addMouseListener(ml);
      addMouseMotionListener(mml);
    }
  }
  public TrackEvent() {
    setLayout(new GridLayout(event.length + 1, 2));
    for(String evt : event) {
      JTextField t = new JTextField();
      t.setEditable(false);
      add(new JLabel(evt, JLabel.RIGHT));
      add(t);
      h.put(evt, t);
    }
    add(b1);
    add(b2);
  }
}

public class E12_NewEvent {
  public static void main(String[] args) {
    run(new TrackEvent(), 700, 500);
  }
} ///:~
```

JButton has many "add listener" methods. An obvious choice is **addActionListener( )**, but we also added one for **addChangeListener( )**.

Exercise 13

```
//: gui/E13_OriginalCase.java
/****************** Exercise 13 *****************
 * Modify TextFields.java so that the characters
 * in t2 retain the original case that they were
 * typed in, instead of automatically being forced
 * to uppercase.
```

```
 *************************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class TextFields extends JFrame {
  private JButton
    b1 = new JButton("Get Text"),
    b2 = new JButton("Set Text");
  private JTextField
    t1 = new JTextField(30),
    t2 = new JTextField(30),
    t3 = new JTextField(30);
  private String s = "";
  private UpperCaseDocument ucd = new UpperCaseDocument();
  public TextFields() {
    t1.setDocument(ucd);
    ucd.addDocumentListener(new T1());
    b1.addActionListener(new B1());
    b2.addActionListener(new B2());
    t1.addActionListener(new T1A());
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(t1);
    add(t2);
    add(t3);
  }
  class T1 implements DocumentListener {
    public void changedUpdate(DocumentEvent e) {}
    public void insertUpdate(DocumentEvent e) {
      t2.setText(t1.getText());
      t3.setText("Text: "+ t1.getText());
    }
    public void removeUpdate(DocumentEvent e) {
      t2.setText(t1.getText());
    }
  }
  class T1A implements ActionListener {
    private int count = 0;
    public void actionPerformed(ActionEvent e) {
      t3.setText("t1 Action Event " + count++);
    }
```

```
  }
  class B1 implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      if(t1.getSelectedText() == null)
        s = t1.getText();
      else
        s = t1.getSelectedText();
      t1.setEditable(true);
    }
  }
  class B2 implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      ucd.setUpperCase(false);
      t1.setText("Inserted by Button 2: " + s);
      ucd.setUpperCase(true);
      t1.setEditable(false);
    }
  }
}

class UpperCaseDocument extends PlainDocument {
  private boolean upperCase /*= true*/;
  public void setUpperCase(boolean flag) {
    //  upperCase = flag;
  }
  public void
  insertString(int offset, String str, AttributeSet attSet)
  throws BadLocationException {
    if(upperCase) str = str.toUpperCase();
    super.insertString(offset, str, attSet);
  }
}

public class E13_OriginalCase {
  public static void main(String[] args) {
    run(new TextFields(), 375, 200);
  }
} ///:~
```

This exercise forces you to study the example thoroughly in order to find where to make the changes (the second and fourth lines of **UpperCaseDocument**).

Exercise 14

```
//: gui/E14_UseTextArea.java
/****************** Exercise 14 *****************
```

```
 * Modify TextPane.java to use a JTextArea instead
 * of a JTextPane.
 *********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import net.mindview.util.*;
import static net.mindview.util.SwingConsole.*;

class TextPane extends JFrame {
  private JButton b = new JButton("Add Text");
  // Only needed to change this one line:
  JTextArea tp = new JTextArea();
  private static Generator<String> sg =
    new RandomGenerator.String(7);
  public TextPane() {
    b.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        for(int i = 1; i < 10; i++)
          tp.setText(tp.getText() + sg.next() + "\n");
      }
    });
    add(new JScrollPane(tp));
    add(BorderLayout.SOUTH, b);
  }
}

public class E14_UseTextArea {
  public static void main(String[] args) {
    run(new TextPane(), 475, 425);
  }
} ///:~
```

Copying from the example in *TIJ4*, we change only the line that defines the **JTextPane**.

Exercise 15

```
//: gui/E15_CheckBoxApplication.java
/****************** Exercise 15 *****************
 * Add a check box to the application created in
 * Exercise 5, capture the event, and insert
 * different text into the text field.
 *********************************************/
package gui;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

public class E15_CheckBoxApplication extends JFrame {
  private JButton
    b1 = new JButton("Button 1"),
    b2 = new JButton("Button 2"),
    b3 = new JButton("Button 3");
  private JTextField txt = new JTextField(12);
  JCheckBox check = new JCheckBox("CheckBox");
  private ActionListener bl = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      String name = ((JButton)e.getSource()).getText();
      txt.setText(name);
    }
  };
  public E15_CheckBoxApplication() {
    b1.addActionListener(bl);
    b2.addActionListener(bl);
    b3.addActionListener(bl);
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(b3);
    add(txt);
    add(check);
    check.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JCheckBox jcb = (JCheckBox)e.getSource();
        if(jcb.isSelected())
          txt.setText("Checkbox checked");
        else
          txt.setText("Checkbox unchecked");
      }
    });
  }
  public static void main(String[] args) {
    run(new E15_CheckBoxApplication(), 200, 180);
  }
} ///:~
```

Exercise 16

```
//: gui/E16_SimplifyList.java
```

```
/****************** Exercise 16 *****************
 * Simplify List.java by passing the array to the
 * constructor and eliminating the dynamic addition
 * of elements to the list.
 ***********************************************/
package gui;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import static net.mindview.util.SwingConsole.*;

public class E16_SimplifyList extends JFrame {
  private String[] flavors = {
    "Chocolate", "Strawberry", "Vanilla Fudge Swirl",
    "Mint Chip", "Mocha Almond Fudge", "Rum Raisin",
    "Praline Cream", "Mud Pie"
  };
  private JList lst = new JList(flavors);
  private JTextArea t =
    new JTextArea(flavors.length, 20);
  private ListSelectionListener ll =
    new ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        if(e.getValueIsAdjusting()) return;
        t.setText("");
        for(Object item : lst.getSelectedValues())
          t.append(item + "\n");
      }
    };
  public E16_SimplifyList() {
    t.setEditable(false);
    setLayout(new FlowLayout());
    Border brd = BorderFactory.createMatteBorder(
      1, 1, 2, 2, Color.BLACK);
    lst.setBorder(brd);
    t.setBorder(brd);
    add(t);
    add(lst);
    lst.addListSelectionListener(ll);
  }
  public static void main(String[] args) {
    run(new E16_SimplifyList(), 250, 375);
  }
} ///:~
```

While this example mainly involves eliminating extra code, it also emphasizes that to access information about the elements in the list you must use the list model, produced by calling **getModel( )**.

Exercise 17

```
//: gui/E17_Password.java
/****************** Exercise 17 *****************
 * Create an application using SwingConsole. In the
 * JDK documentation from http://java.sun.com, find
 * the JPasswordField and add this to the program.
 * If the user types in the correct password, use
 * JOptionPane to provide a success message to the
 * user.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class Password extends JFrame {
  JPasswordField pwd = new JPasswordField(10);
  public Password() {
    setLayout(new FlowLayout());
    add(new JLabel("Type in your password:"));
    add(pwd);
    pwd.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String message = "Incorrect Password";
        JPasswordField pass =
          (JPasswordField)e.getSource();
        if("Blarth".equals(new String(pass.getPassword())))
          message = "Correct Password";
        JOptionPane.showMessageDialog(
          null,
          message,
          "information",
          JOptionPane.INFORMATION_MESSAGE);
      }
    });
  }
}

public class E17_Password {
```

```
  public static void main(String args[]) {
    run(new Password(), 200, 100);
  }
} ///:~
```

Exercise 18

```
//: gui/E18_MessageAction.java
/****************** Exercise 18 ******************
 * Modify MessageBoxes.java so that it has an
 * individual ActionListener for each button (instead
 * of matching the button text).
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class MessageAction extends JFrame {
  private JButton
    b1 = new JButton("Alert"),
    b2 = new JButton("Yes/No"),
    b3 = new JButton("Color"),
    b4 = new JButton("Input"),
    b5 = new JButton("3 Vals");
  private JTextField txt = new JTextField(15);
  public MessageAction() {
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,
          "There's a bug on you!", "Hey!",
          JOptionPane.ERROR_MESSAGE);
      }
    });
    b2.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showConfirmDialog(null,
          "or no", "choose yes",
          JOptionPane.YES_NO_OPTION);
      }
    });
    b3.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Object[] options = { "Red", "Green" };
        int sel = JOptionPane.showOptionDialog(
```

```
          null, "Choose a Color!", "Warning",
          JOptionPane.DEFAULT_OPTION,
          JOptionPane.WARNING_MESSAGE, null,
          options, options[0]);
        if(sel != JOptionPane.CLOSED_OPTION)
          txt.setText("Color Selected: " + options[sel]);
      }
    });
    b4.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String val = JOptionPane.showInputDialog(
            "How many fingers do you see?");
        txt.setText(val);
      }
    });
    b5.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Object[] selections = {"First", "Second", "Third"};
        Object val = JOptionPane.showInputDialog(
          null, "Choose one", "Input",
          JOptionPane.INFORMATION_MESSAGE,
          null, selections, selections[0]);
        if(val != null)
          txt.setText(val.toString());
      }
    });
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(b3);
    add(b4);
    add(b5);
    add(txt);
  }
}

public class E18_MessageAction {
  public static void main(String[] args) {
    run(new MessageAction(), 200, 200);
  }
} ///:~
```

Exercise 19

```
//: gui/E19_RadioMenus.java
/****************** Exercise 19 ******************
```

```
 * Modify Menus.java to use radio buttons instead
 * of check boxes on the menus.
 *********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class RadioMenus extends JFrame {
  private String[] flavors = {
    "Chocolate", "Strawberry", "Vanilla Fudge Swirl",
    "Mint Chip", "Mocha Almond Fudge", "Rum Raisin",
    "Praline Cream", "Mud Pie"
  };
  private JTextField t = new JTextField("No flavor", 30);
  private JMenuBar mb1 = new JMenuBar();
  private JMenu
    f = new JMenu("File"),
    m = new JMenu("Flavors"),
    s = new JMenu("Safety");
  private JRadioButtonMenuItem[] safety = {
    new JRadioButtonMenuItem("Guard"),
    new JRadioButtonMenuItem("Hide")
  };
  private JMenuItem[] file = { new JMenuItem("Open") };
  private JMenuBar mb2 = new JMenuBar();
  private JMenu fooBar = new JMenu("fooBar");
  private JMenuItem[] other = {
    new JMenuItem("Foo", KeyEvent.VK_F),
    new JMenuItem("Bar", KeyEvent.VK_A),
    new JMenuItem("Baz"),
  };
  private JButton b = new JButton("Swap Menus");
  class BL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      JMenuBar m = getJMenuBar();
      setJMenuBar(m == mb1 ? mb2 : mb1);
      validate(); // Refresh the frame
    }
  }
  class ML implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      JMenuItem target = (JMenuItem)e.getSource();
      String actionCommand = target.getActionCommand();
      if(actionCommand.equals("Open")) {
        String s = t.getText();
```

```
        boolean chosen = false;
        for(String flavor : flavors)
          if(s.equals(flavor))
            chosen = true;
        if(!chosen)
          t.setText("Choose a flavor first!");
        else
          t.setText("Opening " + s + ". Mmm, mm!");
      }
    }
  }
  class FL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      JMenuItem target = (JMenuItem)e.getSource();
      t.setText(target.getText());
    }
  }
  class FooL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      t.setText("Foo selected");
    }
  }
  class BarL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      t.setText("Bar selected");
    }
  }
  class BazL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      t.setText("Baz selected");
    }
  }
  class CMIL implements ItemListener {
    public void itemStateChanged(ItemEvent e) {
      JRadioButtonMenuItem target =
        (JRadioButtonMenuItem)e.getSource();
      String actionCommand = target.getActionCommand();
      if(actionCommand.equals("Guard"))
        t.setText("Guard the Ice Cream! " +
          "Guarding is " + target.isSelected());
      else if(actionCommand.equals("Hide"))
        t.setText("Hide the Ice Cream! " +
          "Is it hidden? " + target.isSelected());
    }
  }
  public RadioMenus() {
    ML ml = new ML();
```

```
    CMIL cmil = new CMIL();
    safety[0].setActionCommand("Guard");
    safety[0].setMnemonic(KeyEvent.VK_G);
    safety[0].addItemListener(cmil);
    safety[1].setActionCommand("Hide");
    safety[1].setMnemonic(KeyEvent.VK_H);
    safety[1].addItemListener(cmil);
    other[0].addActionListener(new FooL());
    other[1].addActionListener(new BarL());
    other[2].addActionListener(new BazL());
    FL fl = new FL();
    int n = 0;
    for(String flavor : flavors) {
      JMenuItem mi = new JMenuItem(flavor);
      mi.addActionListener(fl);
      m.add(mi);
      if((n++ + 1) % 3 == 0)
        m.addSeparator();
    }
    for(JRadioButtonMenuItem sfty : safety)
      s.add(sfty);
    s.setMnemonic(KeyEvent.VK_A);
    f.add(s);
    f.setMnemonic(KeyEvent.VK_F);
    for(int i = 0; i < file.length; i++) {
      file[i].addActionListener(ml);
      f.add(file[i]);
    }
    mb1.add(f);
    mb1.add(m);
    setJMenuBar(mb1);
    t.setEditable(false);
    add(t, BorderLayout.CENTER);
    b.addActionListener(new BL());
    b.setMnemonic(KeyEvent.VK_S);
    add(b, BorderLayout.NORTH);
    for(JMenuItem oth : other)
      fooBar.add(oth);
    fooBar.setMnemonic(KeyEvent.VK_B);
    mb2.add(fooBar);
  }
}

public class E19_RadioMenus {
  public static void main(String[] args) {
    run(new RadioMenus(), 300, 200);
  }
```

```
} ///:~
```

We copy the example, then replace all **JCheckBoxMenuItem**s with **JRadioButtonMenuItem**s, and all calls to **getState( )** with calls to **isSelected( )**.

Exercise 20

```
//: gui/E20_DynamicMenus.java
/****************** Exercise 20 *****************
 * Create a program that breaks a text file into
 * words. Distribute those words as labels on menus
 * and submenus.
 ***********************************************/
package gui;
import java.util.*;
import javax.swing.*;
import net.mindview.util.*;
import static net.mindview.util.SwingConsole.*;

class DynamicMenus extends JFrame {
  // You may want to change the code to use a command-line
  // parameter instead of a hard-coded value:
  private Set<String> words = new TreeSet<String>(
    new TextFile("E20_DynamicMenus.java", "\\W+"));
  private JMenuBar mb = new JMenuBar();
  // The top level menu is fixed:
  private JMenu
    // Words starting with an upper case letter:
    tm1 = new JMenu("Words Starting With UCC"),
    // Words starting with a lower case letter:
    tm2 = new JMenu("Words Starting With LCC"),
    // All other words not belonging to the above groups:
    tm3 = new JMenu("Other Words");
  DynamicMenus() {
    distributeWordsOnMenus();
    mb.add(tm1);
    mb.add(tm2);
    mb.add(tm3);
    setJMenuBar(mb);
  }
  private void distributeWordsOnMenus() {
    boolean firstInGroup;
    char currentGroup, lastGroup = (char)0;
    JMenu currentMenu;
    JMenu currentSM = null;   // The current sub-menu
```

```
    String word, nextWord = null;
    boolean callNext = true;
    for(Iterator<String> it = words.iterator();
        it.hasNext();) {
      if(callNext)
        word = it.next();
      else {
        word = nextWord;
        callNext = true;
      }
      if(word.matches("[0-9]+"))
        continue;  // Ignore plain numbers.
      currentGroup = word.charAt(0);
      firstInGroup = currentGroup != lastGroup;
      lastGroup = currentGroup;
      // Decide which top level menu to extend:
      if(Character.isLowerCase(currentGroup))
        currentMenu = tm2;
      else if(Character.isUpperCase(currentGroup))
        currentMenu = tm1;
      else
        currentMenu = tm3;
      // Decide whether this item should be put as a menu
      // item beneath the top menu, or under the
      // corresponding sub-menu:
      JMenuItem itemToAdd = new JMenuItem(word);
      if(firstInGroup) {
        // Now we need to see whether we need a new
        // sub-menu or just a plain menu item:
        if(!it.hasNext()) {
          currentMenu.add(itemToAdd);
          break;
        }
        nextWord = it.next();
        callNext = false;
        if(currentGroup != nextWord.charAt(0))
          currentMenu.add(itemToAdd);
        else {
          currentSM =
            new JMenu(Character.toString(currentGroup));
          currentSM.add(itemToAdd);
          currentMenu.add(currentSM);
        }
      } else
        currentSM.add(itemToAdd);
    }
  }
```

```
}

public class E20_DynamicMenus {
  public static void main(String[] args) {
    run(new DynamicMenus(), 400, 400);
  }
} ///:~
```

Notice that we group single words as standalone menu items (categorized by their first character) under corresponding top-level menus. The **swt/Menus.java** program in *TIJ4* shows another approach to the solution.

Exercise 21

```
//: gui/E21_SineDrawBean.java
/****************** Exercise 21 *****************
 * Modify SineWave.java to turn SineDraw into a
 * JavaBean by adding "getter" and "setter" methods.
 ************************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import static net.mindview.util.SwingConsole.*;

class SineDrawBean extends JPanel {
  private static final int SCALEFACTOR = 200;
  private int cycles;
  private int points;
  private double[] sines;
  private int[] pts;
  public SineDrawBean() { setCycles(5); }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    int maxWidth = getWidth();
    double hstep = (double)maxWidth / (double)points;
    int maxHeight = getHeight();
    pts = new int[points];
    for(int i = 0; i < points; i++)
      pts[i] =
        (int)(sines[i] * maxHeight/2 * .95 + maxHeight/2);
    g.setColor(Color.RED);
    for(int i = 1; i < points; i++) {
      int x1 = (int)((i - 1) * hstep);
      int x2 = (int)(i * hstep);
      int y1 = pts[i-1];
```

```
      int y2 = pts[i];
      g.drawLine(x1, y1, x2, y2);
    }
  }
  public void setCycles(int newCycles) {
    cycles = newCycles;
    points = SCALEFACTOR * cycles * 2;
    sines = new double[points];
    for(int i = 0; i < points; i++) {
      double radians = (Math.PI / SCALEFACTOR) * i;
      sines[i] = Math.sin(radians);
    }
    repaint();
  }
  public int getCycles() { return cycles; }
}
class SineWave extends JFrame {
  private SineDrawBean sines = new SineDrawBean();
  private JSlider adjustCycles = new JSlider(1, 30, 5);
  public SineWave() {
    add(sines);
    adjustCycles.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        sines.setCycles(
          ((JSlider)e.getSource()).getValue());
      }
    });
    add(BorderLayout.SOUTH, adjustCycles);
  }
}

public class E21_SineDrawBean {
  public static void main(String[] args) {
    run(new SineWave(), 700, 400);
  }
} ///:~
```

There's a trick to this exercise. We added only the method **getCycles( )**, but no getters and setters for **SCALEFACTOR**, **points**, **sines**, or **pts**. Those *dependent variables* are calculated on other values within the object, so you can't allow them to be set from outside of the bean.

Be careful when designing beans so that you don't mistake getters and setters for trivial access methods to variables inside an object. **setCycles( )**, for example, does much more than modify **cycles**.

Exercise 22

```
//: gui/E22_ColorMixer.java
/****************** Exercise 22 *****************
 * Create an application using SwingConsole. This
 * should have three sliders, one each for the red,
 * green, and blue values in java.awt.Color. The rest
 * of the form should be a JPanel that displays the
 * color determined by the three sliders. Also
 * include non-editable text fields that show the
 * current RGB values.
 ***********************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import static net.mindview.util.SwingConsole.*;

class ColorMixer extends JFrame {
  class ColorSlider extends JPanel {
    JSlider slider;
    JTextField value;
    public ColorSlider(String title) {
      super(new FlowLayout());
      slider = new JSlider(0, 255, 0);
      value = new JTextField(3);
      add(new JLabel(title));
      add(slider);
      value.setText("" + slider.getValue());
      add(value);
      slider.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
          value.setText("" + slider.getValue());
          reColor();
        }
      });
    }
  }
  ColorSlider
    red = new ColorSlider("red"),
    green = new ColorSlider("green"),
    blue = new ColorSlider("blue");
  JPanel color = new JPanel();
  void reColor() {
    color.setBackground(new Color(
      red.slider.getValue(),
```

```
      green.slider.getValue(),
      blue.slider.getValue()));
  }
  public ColorMixer() {
    setLayout(new GridLayout(2, 1));
    JPanel jp = new JPanel(new FlowLayout());
    jp.add(red);
    jp.add(green);
    jp.add(blue);
    add(jp);
    add(color);
    reColor();
  }
}

public class E22_ColorMixer {
  public static void main(String args[]) {
    run(new ColorMixer(), 350, 300);
  }
} ///:~
```

The **ColorSlider** inner class really simplifies and organizes the code.

As an additional exercise, add a check box that causes the sliders to jump to the next web-safe color when you release them.

Exercise 23

```
//: gui/E23_RotatingSquare.java
/****************** Exercise 23 *****************
 * Using SineWave.java as a starting point, create
 * a program that displays a rotating square on the
 * screen. One slider should control the speed of
 * rotation, and a second slider should control the
 * size of the box.
 ***********************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.geom.*;
import static net.mindview.util.SwingConsole.*;

class SquareRotate extends JPanel {
  private Rectangle2D square =
    new Rectangle2D.Float(-50f, -50f, 100f, 100f);
```

```
  private AffineTransform
    rot = new AffineTransform(),
    scale = new AffineTransform();
  private volatile int speed;
  private int boxSize;
  public SquareRotate() {
    setSpeed(5);
    setBoxSize(10);
    new Thread(new Runnable() {
      public void run() {
        for(;;) {
          SquareRotate.this.repaint();
          try {
            Thread.sleep(1000 / speed);
          } catch(InterruptedException ignore) {}
        }
      }
    }).start();
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    // Makes point (0,0) as the center of this canvas
    g2.translate(getWidth() / 2, getHeight() / 2);
    g2.scale(boxSize / 10.0, boxSize / 10.0);
    g2.setPaint(Color.blue);
    rot.rotate(Math.toRadians(20));
    g2.transform(rot);
    g2.draw(square);
  }
  public void setSpeed(int speed) { this.speed = speed; }
  public void setBoxSize(int boxSize) {
    this.boxSize = boxSize;
  }
}

class RotatingSquare extends JFrame {
  private SquareRotate sq = new SquareRotate();
  private JSlider adjustSpeed = new JSlider(1, 10, 5);
  private JSlider adjustBoxSize = new JSlider(1, 20, 10);
  public RotatingSquare() {
    add(sq);
    adjustSpeed.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        sq.setSpeed(adjustSpeed.getValue());
      }
    });
```

```
    adjustBoxSize.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        sq.setBoxSize(adjustBoxSize.getValue());
      }
    });
    JPanel sliders = new JPanel();
    sliders.setLayout(new GridLayout(2, 1));
    sliders.add(adjustSpeed);
    sliders.add(adjustBoxSize);
    add(BorderLayout.SOUTH, sliders);
  }
}

public class E23_RotatingSquare {
  public static void main(String args[]) {
    run(new RotatingSquare(), 400, 400);
  }
} ///:~
```

This solution demonstrates the Java 2D API. Consult the JDK for more information about the classes used here. Notice how the **AffineTransform** class spares you from complex mathematics by handling all aspects of rotation and scaling.

A dedicated thread controls the speed of rotation.

You can safely call the **repaint( )** method from any thread.

As an additional exercise, try to solve this without the Java 2D API and compare the two versions.

Exercise 24

```
//: gui/E24_SketchBox.java
/****************** Exercise 24 *****************
 * Remember the "sketching box" toy with two knobs,
 * one that controls the vertical movement of the
 * drawing point, and one that controls the horizontal
 * movement? Create a variation of this toy, using
 * SineWave.java to get you started. Instead of knobs,
 * use sliders. Add a button that will erase the entire
 * sketch.
 ***********************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
```

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import static net.mindview.util.SwingConsole.*;

class SketchArea extends JPanel {
  java.util.List<Point> points = new ArrayList<Point>();
  class Point {
    int x, y;
    public Point(int x, int y) {
      this.x = x;
      this.y = y;
    }
  }
  public void addPoint(int x, int y) {
    points.add(new Point(x,y));
    repaint();
  }
  public void clear() {
    points.clear();
    repaint();
  }
  Point previousPoint;
  void drawPoint(Graphics g, Point p) {
    // So that it starts from anywhere, rather
    // than drawing from 0, 0:
    if(previousPoint == null) {
      previousPoint = p;
      return;
    }
    g.drawLine(previousPoint.x, previousPoint.y,
      p.x, p.y);
    previousPoint = p;
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.red);
    previousPoint = null;
    for(Point p : points)
      drawPoint(g, p);
  }
}

class SketchBox extends JFrame {
  SketchArea sketch = new SketchArea();
  JSlider
    hAxis = new JSlider(),
```

```
    vAxis = new JSlider(JSlider.VERTICAL);
  JButton erase = new JButton("Erase");
  ChangeListener cl = new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
      sketch.addPoint(hAxis.getValue(), vAxis.getValue());
      erase.setText(
        "[Erase]   points.size() = " +
        sketch.points.size());
    }
  };
  public SketchBox() {
    add(sketch);
    hAxis.setValue(0);
    vAxis.setValue(0);
    // So vertical axis synchronizes with line:
    vAxis.setInverted(true);
    hAxis.addChangeListener(cl);
    vAxis.addChangeListener(cl);
    add(BorderLayout.SOUTH, hAxis);
    add(BorderLayout.WEST, vAxis);
    add(BorderLayout.NORTH, erase);
    erase.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        sketch.clear();
        erase.setText(
          "[Erase]   points.size() = " +
          sketch.points.size());
      }
    });
    // The width and height values are zero until
    // initial resizing (when the component is
    // first drawn). Also takes care of things
    // if it's resized:
    addComponentListener(new ComponentAdapter() {
      public void componentResized(ComponentEvent e) {
        super.componentResized(e);
        hAxis.setMaximum(sketch.getWidth());
        vAxis.setMaximum(sketch.getHeight());
      }
    });
  }
}

public class E24_SketchBox {
  public static void main(String args[]) {
    run(new SketchBox(), 700, 400);
  }
```

```
} ///:~
```

SketchArea contains a **List** of **Point** objects, each of which has an **x** and **y** coordinate. We call **paintComponent( )** to move through the list, using **drawPoint( )** to draw a line from each coordinate to the next.

The **SketchBox** has a horizontal **JSlider** (the default orientation) and a vertical one. The **ChangeListener** for both **JSlider**s adds a new **Point** at the current location and changes the text on the **JButton** to track the number of points in a drawing.

setInverted( ) on the **vAxis** draws the line in the direction of the slider.

We also use **addComponentListener( )**, passing it a **ComponentAdapter** with an overridden **componentResized( )**. This does more than make the program resize properly. In fact, you can't just call **setMaximum( )** in the constructor; the application isn't sized during construction, so **getWidth( )** and **getHeight( )** always return zero inside the constructor. We determine the initial application size after object construction, then call **componentResized( )** to set the slider maximums.

Exercise 25

```
//: gui/E25_AnimatedSine.java
/****************** Exercise 25 ******************
 * Starting with SineWave.java, create a program
 * (an application using the SwingConsole class)
 * that draws an animated sine wave that appears to
 * scroll past the viewing window like an oscilloscope,
 * driving the animation with a java.util.Timer. The
 * speed of the animation should be controlled with
 * a javax.swing.JSlider control.
 *********************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.util.TimerTask;
import static net.mindview.util.SwingConsole.*;

class ExtSineDraw extends JPanel {
  private static final int SCALEFACTOR = 200;
  private int cycles;
  private int points;
  private double[] sines;
  private int[] pts;
```

```
  private java.util.Timer timer = new java.util.Timer();
  private double offset;
  private class MyTimerTask extends TimerTask {
    public void run() {
      offset += 0.25;
      synchronized(ExtSineDraw.this) {
        for(int i = 0; i < points; i++) {
          double radians =
            (Math.PI/SCALEFACTOR) * i + offset;
          sines[i] = Math.sin(radians);
        }
      }
      repaint();
    }
  };
  private JSlider speed = new JSlider(15, 115, 65);
  ExtSineDraw() {
    super(new BorderLayout());
    setCycles(5);
    speed.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        timer.cancel();
        timer = new java.util.Timer();
        timer.scheduleAtFixedRate(
          new MyTimerTask(), 0, speed.getValue());
      }
    });
    add(BorderLayout.SOUTH, speed);
    // So that left to right is slow to fast:
    speed.setInverted(true);
    // Incorporate a 1 sec. initial delay to give time
    // for initialization:
    timer.scheduleAtFixedRate(new MyTimerTask(), 1000, 65);
  }
  public synchronized void setCycles(int newCycles) {
    cycles = newCycles;
    points = SCALEFACTOR * cycles * 2;
    sines = new double[points];
    pts = new int[points];
    for(int i = 0; i < points; i++) {
      double radians = (Math.PI / SCALEFACTOR) * i;
      sines[i] = Math.sin(radians);
    }
    repaint();
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    int maxWidth = getWidth();
    double hstep = (double)maxWidth / (double)points;
    int maxHeight = getHeight();
    synchronized(this) {
      for(int i = 0; i < points; i++)
        // Some adjustments here to compensate for
        // the added JSlider in the panel:
        pts[i] =
          (int)(sines[i] * maxHeight/2 * .89 +
          maxHeight/2 * .91);
    }
    g.setColor(Color.RED);
    for(int i = 1; i < points; i++) {
      int x1 = (int)((i - 1) * hstep);
      int x2 = (int)(i * hstep);
      int y1 = pts[i-1];
      int y2 = pts[i];
      g.drawLine(x1, y1, x2, y2);
    }
  }
}

class ExtSineWave extends JFrame {
  private ExtSineDraw sines = new ExtSineDraw();
  private JSlider adjustCycles = new JSlider(1, 30, 5);
  public ExtSineWave() {
    add(sines);
    adjustCycles.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        sines.setCycles(adjustCycles.getValue());
      }
    });
    add(BorderLayout.SOUTH, adjustCycles);
  }
}

public class E25_AnimatedSine {
  public static void main(String[] args) {
    run(new ExtSineWave(), 700, 400);
  }
} ///:~
```

Most of the changes are in **SineDraw**, which uses a **java.util.Timer** with a **run( )** method to drive the animation. We use a **BorderLayout** in the constructor instead of the default **FlowLayout**, and add the **JSlider speed** (changing the **speed** "reprograms" the timer).

The **run( )** method calculates the points as before, and also increments and adds a value **offset** each time (this value moves the sine wave forward).

We use the **synchronized** clause in **run( )** and **paintComponent( )** and synchronize **setCycles( )** to prevent threading collisions when we change the array sizes.

Exercise 26

```
//: gui/E26_MultipleSine.java
/****************** Exercise 26 *****************
 * Modify the previous exercise so that multiple
 * sine wave panels are created within the
 * application. The number of sine wave panels should
 * be controlled by command-line parameters.
 ************************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import static net.mindview.util.SwingConsole.*;

class MultiSineWave extends JFrame {
  ExtSineDraw[] sines;
  JSlider cycles = new JSlider(1, 30, 5);
  public MultiSineWave(int panels) {
    int side = Math.round(
      (float)Math.sqrt((double)panels));
    JPanel jp =
      new JPanel(new GridLayout(side, side));
    sines = new ExtSineDraw[panels];
    for(int i = 0; i < sines.length; i++) {
      sines[i] = new ExtSineDraw();
      jp.add(sines[i]);
    }
    add(jp);
    add(BorderLayout.SOUTH, cycles);
    cycles.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        for(int i = 0; i < sines.length; i++)
          sines[i].setCycles(
            ((JSlider)e.getSource()).getValue());
      }});
  }
}
```

```
public class E26_MultipleSine {
  public static void main(String[] args) {
    int panels;
    if(args.length != 0)
      panels = Integer.parseInt(args[0]);
    else
      panels = 4;
    run(new MultiSineWave(panels), 700, 400);
  }
} ///:~
```

We use the **ExtSineDraw** class from the previous exercise. This solution is fairly rudimentary: you get the argument from the command line (or use a default value of 4) and lay out the panels. When you change the number of cycles, **ChangeListener** moves through an array of **ExtSineDraw** objects and calls **setCycles( )** for each one.

We get a reasonably square array by rounding up the square root of the desired number of **ExtSineDraw** objects for both dimensions of a **GridLayout**.

Exercise 27

```
//: gui/E27_TimerAnimation.java
/********************** Exercise 27 *********************
 * Modify Exercise 25 so that the javax.swing.Timer class
 * is used to drive the animation. Note the difference
 * between this and java.util.Timer.
 *******************************************************/
package gui;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class SineDraw_T extends JPanel {
  private static final int SCALEFACTOR = 200;
  private int cycles;
  private int points;
  private double[] sines;
  private int[] pts;
  private JSlider speed = new JSlider(0, 15, 5);
  private double offset;
  private Timer timer = new Timer(500,
    new ActionListener() {
```

```
      public void actionPerformed(ActionEvent e) {
        offset += 0.25;
        for(int i = 0; i < points; i++) {
          double radians = (Math.PI/SCALEFACTOR)
            * i + offset;
          sines[i] = Math.sin(radians);
        }
        repaint();
      }
    });
  SineDraw_T() {
    super(new BorderLayout());
    setCycles(5);
    speed.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        timer.setDelay(speed.getValue() * 100);
      }
    });
    add(BorderLayout.SOUTH, speed);
    speed.setInverted(true);
    timer.setInitialDelay(1000);
    timer.start();
  }
  public void setCycles(int newCycles) {
    cycles = newCycles;
    points = SCALEFACTOR * cycles * 2;
    sines = new double[points];
    pts = new int[points];
    for(int i = 0; i < points; i++) {
      double radians = (Math.PI/SCALEFACTOR) * i;
      sines[i] = Math.sin(radians);
    }
    repaint();
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.RED);
    int maxWidth = getWidth();
    double hstep =(double)maxWidth/(double)points;
    int maxHeight = getHeight();
    for(int i = 0; i < points; i++)
      // Some adjustments here to compensate for
      // the added JSlider in the panel:
      pts[i] = (int)(sines[i] * maxHeight/2 * .94
                     + maxHeight/2 * .96);
    for(int i = 1; i < points; i++) {
      int x1 = (int)((i - 1) * hstep);
```

```
      int x2 = (int)(i * hstep);
      int y1 = pts[i-1];
      int y2 = pts[i];
      g.drawLine(x1, y1, x2, y2);
    }
  }
}

class SineWave_T extends JFrame {
  private SineDraw_T sines = new SineDraw_T();
  private JSlider cycles = new JSlider(1, 30, 5);
  public SineWave_T() {
    add(sines);
    cycles.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        sines.setCycles(cycles.getValue());
      }
    });
    add(BorderLayout.SOUTH, cycles);
  }
}

public class E27_TimerAnimation {
  public static void main(String args[]) {
    run(new SineWave_T(), 700, 400);
  }
} ///:~
```

javax.swing.Timer repeatedly calls an **ActionListener**. The code is simpler than in Exercise 25 (e.g., no need for synchronization). However, the handler must execute quickly to keep the GUI responsive, since the event-dispatching thread executes the callback. As a corollary, you should not overuse **javax.swing.Timer**.

Exercise 28

```
//: gui/E28_DiceTossing.java
/********************** Exercise 28 ********************
 * Create a dice class (just a class, without a GUI). Create
 * five dice and throw them repeatedly. Draw the curve
 * showing the sum of the dots from each throw, and show the
 * curve evolving dynamically as you throw more and more
 * times.
 *******************************************************/
package gui;
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
import java.util.Random;
import static net.mindview.util.SwingConsole.*;

class Dice {
  static Random rnd = new Random();
  int throwDice() { return rnd.nextInt(6) + 1; }
}

class Curve extends JPanel {
  private static final int MAX_DATA_POINTS = 100;
  private int[] data = new int[MAX_DATA_POINTS];
  private int offset;
  // Temporary store for drawing:
  private int[] pts = new int[MAX_DATA_POINTS];
  public void addData(int sumOfDots) {
    // Check whether we need to throw away the first half
    // of old data:
    if(offset == MAX_DATA_POINTS) {
      offset = MAX_DATA_POINTS / 2;
      System.arraycopy(data, offset, data, 0, offset);
    }
    data[offset++] = sumOfDots;
    repaint();
  }
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.GREEN);
    int maxWidth = getWidth();
    double hstep =(double)maxWidth/(double)MAX_DATA_POINTS;
    int maxHeight = getHeight();
    for(int i = 0; i < offset; i++)
      pts[i] = (int)((data[i] - 6) * maxHeight/24);
    for(int i = 1; i < offset; i++) {
      int x1 = (int)((i - 1) * hstep);
      int x2 = (int)(i * hstep);
      int y1 = pts[i-1];
      int y2 = pts[i];
      g.drawLine(x1, y1, x2, y1);
      g.drawLine(x2, y1, x2, y2);
    }
  }
}

class DiceTossing extends JFrame {
  private final Curve curve = new Curve();
```

```
  private final Dice[] dices = {new Dice(), new Dice(),
    new Dice(), new Dice(), new Dice()};
  // Simulates dice tossing:
  private Timer timer = new Timer(100,
    new ActionListener() {
      int sumOfDots;
      public void actionPerformed(ActionEvent e) {
        sumOfDots = 0;
        for(Dice dice : dices)
          sumOfDots += dice.throwDice();
        curve.addData(sumOfDots);
      }
    });
  public DiceTossing() {
    add(curve);
    // A "standard" 1 sec. initial delay:
    timer.setInitialDelay(1000);
    timer.start();
  }
}

public class E28_DiceTossing {
  public static void main(String args[]) {
    run(new DiceTossing(), 500, 300);
  }
} ///:~
```

Here, the timer drives the dice tossing simulation, and calls **addData( )** at each iteration. The **Curve** uses the generated data to draw the curve showing the sum of the dice from each throw. Everything basically follows the structure of the **SineWave.java** program.

Exercise 29

```
//: gui/E29_ColorChooser.java
/****************** Exercise 29 *****************
 * In the JDK documentation for javax.swing,
 * look up the JColorChooser. Write a program
 * with a button that brings up the color
 * chooser as a dialog.
 ************************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;
```

```
class ColorChooser extends JFrame {
  JButton b1 = new JButton("Color Chooser");
  public ColorChooser() {
    setLayout(new FlowLayout());
    add(b1);
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        JColorChooser.showDialog(
          null, "E29_ColorChooser", Color.cyan);
      }
    });
  }
}

public class E29_ColorChooser {
  public static void main(String args[]) {
    run(new ColorChooser(), 150, 75);
  }
} ///:~
```

As an additional exercise, use the color display panel from Exercise22 to show the resulting color after the dialog is closed.

Exercise 30

```
//: gui/E30_HTMLComponents.java
/****************** Exercise 30 *****************
 * "You can also use HTML text for JTabbedPane,
 * JMenuItem, JToolTip, JRadioButton, and
 * JCheckBox."
 * Write a program that shows the use of HTML text
 * on all the items from the previous paragraph.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import static net.mindview.util.SwingConsole.*;

class HTMLComponents extends JFrame {
  JMenu[] menus =
    { new JMenu("Text"), new JMenu("<html><i>HTML") };
  JMenuItem[] items = {
    new JMenuItem("Text Item"),
    new JMenuItem("<html><b><font size=+1>HTML Item") };
  public HTMLComponents() {
```

```
    setLayout(new FlowLayout());
    JTabbedPane tabs = new JTabbedPane();
    tabs.addTab("<html><font size=+1>Tab 1",
      new JCheckBox("<html><u>Check me!"));
    tabs.addTab("<html><font size=-1>Tab 2",
      new JRadioButton("<html><i>Click me!"));
    add(tabs);
    items[1].setToolTipText(
      "<html><center>Dummy Item<br><i>No action");
    menus[0].add(items[0]);
    menus[1].add(items[1]);
    JMenuBar mb = new JMenuBar();
    mb.add(menus[0]);
    mb.add(menus[1]);
    setJMenuBar(mb);
  }
}

public class E30_HTMLComponents {
  public static void main(String args[]) {
    run(new HTMLComponents(), 180, 200);
  }
} ///:~
```

Exercise 31

```
//: gui/E31_WindozeProgress.java
/****************** Exercise 31 *****************
 * Create an "asymptotic progress indicator" that
 * gets slower and slower as it approaches the
 * finish point. Add random erratic behavior so
 * it will periodically look like it's starting
 * to speed up.
 ***********************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import static net.mindview.util.SwingConsole.*;

class WindozeProgress extends JFrame {
  JProgressBar pb = new JProgressBar(0, 100);
  JLabel jl =
    new JLabel("Makin' some progress now!");
  Timer tm = new Timer(50, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```
      if(pb.getValue() == pb.getMaximum()) {
        tm.stop();
        jl.setText("That wasn't so bad, was it?");
      }
      double percent =
        (double)pb.getValue() /
        (double)pb.getMaximum();
      tm.setDelay((int)(tm.getDelay() *
        (1.0 + percent * 0.07)));
      pb.setValue(pb.getValue() + 1);
      if(percent > 0.90)
        if(Math.random() < 0.25)
          pb.setValue(pb.getValue() - 10);
    }
  });
  public WindozeProgress() {
    setLayout(new FlowLayout());
    add(pb);
    add(jl);
    pb.setValue((int)(pb.getMaximum() * 0.25));
    tm.start();
  }
}

public class E31_WindozeProgress {
  public static void main(String args[]) {
    run(new WindozeProgress(), 300, 100);
  }
} ///:~
```

We control the progress bar with a **Timer** , with a delay adjusted according to your proximity to finishing. If you're closer than 90%, the program chooses a random number between zero and one, then decrements the progress bar by 10 units if it's less than 0.25.

Exercise 32

```
//: gui/E32_SharedListener.java
/****************** Exercise 32 *****************
 * Modify Progress.java so that it does not share
 * models, but instead uses a listener to connect
 * the slider and progress bar.
 ************************************************/
package gui;
import javax.swing.*;
import java.awt.*;
```

```
import javax.swing.event.*;
import javax.swing.border.*;
import static net.mindview.util.SwingConsole.*;

class Progress extends JFrame {
  JProgressBar pb = new JProgressBar();
  JSlider sb =
    new JSlider(JSlider.HORIZONTAL, 0, 100, 60);
  public Progress() {
    setLayout(new GridLayout(2,1));
    add(pb);
    sb.setValue(0);
    sb.setPaintTicks(true);
    sb.setMajorTickSpacing(20);
    sb.setMinorTickSpacing(5);
    sb.setBorder(new TitledBorder("Slide Me"));
//    pb.setModel(sb.getModel()); // Share model
    add(sb);
    sb.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        pb.setValue(sb.getValue());
      }
    });
  }
}

public class E32_SharedListener {
  public static void main(String args[]) {
    run(new Progress(), 300, 200);
  }
} ///:~
```

We just comment out **setModel( )** and add **ChangeListener**. We also remove **ProgressMonitor** because it is irrelevant.

Exercise 33

```
//: gui/E33_ParallelCallables.java
/****************** Exercise 33 *****************
 * Modify InterruptableLongRunningCallable.java so
 * that it runs all the tasks in parallel rather
 * than sequentially.
 ************************************************/
package gui;
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.*;
import net.mindview.util.TwoTuple;
import static net.mindview.util.SwingConsole.*;

class CallableTask extends Task
implements Callable<String> {
  public String call() {
    run();
    return "Return value of " + this;
  }
}

class TaskManager<R,C extends Callable<R>>
extends ArrayList<TwoTuple<Future<R>,C>> {
  private ExecutorService exec =
    Executors.newCachedThreadPool();
  public void add(C task) {
    add(new TwoTuple<Future<R>,C>(exec.submit(task),task));
  }
  public List<R> getResults() {
    Iterator<TwoTuple<Future<R>,C>> items = iterator();
    List<R> results = new ArrayList<R>();
    while(items.hasNext()) {
      TwoTuple<Future<R>,C> item = items.next();
      if(item.first.isDone()) {
        try {
          results.add(item.first.get());
        } catch(Exception e) {
          throw new RuntimeException(e);
        }
        items.remove();
      }
    }
    return results;
  }
  public List<String> purge() {
    Iterator<TwoTuple<Future<R>,C>> items = iterator();
    List<String> results = new ArrayList<String>();
    while(items.hasNext()) {
      TwoTuple<Future<R>,C> item = items.next();
      if(!item.first.isDone()) {
        results.add("Cancelling " + item.second);
        item.first.cancel(true); // May interrupt
```

```
        items.remove();
      }
    }
    return results;
  }
}

public class E33_ParallelCallables extends JFrame {
  private JButton
    b1 = new JButton("Start Long Running Task"),
    b2 = new JButton("End Long Running Task"),
    b3 = new JButton("Get results");
  private TaskManager<String,CallableTask> manager =
    new TaskManager<String,CallableTask>();
  public E33_ParallelCallables() {
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        CallableTask task = new CallableTask();
        manager.add(task);
        System.out.println(task + " added to the queue");
      }
    });
    b2.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        for(String result : manager.purge())
          System.out.println(result);
      }
    });
    b3.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        for(TwoTuple<Future<String>,CallableTask> tt :
            manager)
          tt.second.id(); // No cast required
        for(String result : manager.getResults())
          System.out.println(result);
      }
    });
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(b3);
  }
  public static void main(String[] args) {
    run(new E33_ParallelCallables(), 200, 150);
  }
} ///:~
```

The crucial (and only) change is a different **Executor** in **TaskManager**.

Exercise 34

```
//: gui/E34_ColorStars.java
/****************** Exercise 34 *****************
 * Modify ColorBoxes.java so that it begins by
 * sprinkling points ("stars") across the canvas,
 * then randomly changes the colors of those "stars."
 ************************************************/
package gui;
import javax.swing.*;
import java.awt.*;
import java.util.concurrent.*;
import java.util.*;
import static net.mindview.util.SwingConsole.*;

class JStar extends JComponent implements Runnable {
  // The number of points on the star:
  private final int num_points;
  private final int pause;
  static Random rand = new Random();
  private Color color = new Color(0);
  // Fields used by the drawing algorithm:
  private final int[] x;
  private final int[] y;
  private final double[] dx;
  private final double[] dy;
  public JStar(int num_points, int pause) {
    this.num_points = num_points;
    this.pause = pause;
    x = new int[num_points * 2];
    y = new int[num_points * 2];
    dx = new double[num_points * 2];
    dy = new double[num_points * 2];
    // Cache calculated data:
    double delta = Math.PI / num_points;
    for(int i = 0; i < num_points * 2; i++) {
      dx[i] = Math.cos(delta * i);
      dy[i] = Math.sin(delta * i);
    }
  }
  public void paintComponent(Graphics g) {
    int width = getWidth();
    int height = getHeight();
```

```
    // The coordinates of the centre of the star:
    int centreX = width / 2;
    int centreY = height / 2;
    // The distance from the centre to points of the star
    // and to the inner core of the star, respectively:
    int outerRing, innerRing;
    if(width < height) {
      innerRing = width / 8;
      outerRing = width / 2;
    } else {
      innerRing = height / 8;
      outerRing = height / 2;
    }
    int d;
    for(int i = 0; i < num_points * 2; i++) {
      d = (i % 2 == 0) ? innerRing : outerRing;
      x[i] = centreX + (int)(d * dx[i]);
      y[i] = centreY + (int)(d * dy[i]);
    }
    g.setColor(color);
    g.fillPolygon(x ,y , num_points * 2);
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        color = new Color(rand.nextInt(0xFFFFFF));
        repaint(); // Asynchronously request a paint()
        TimeUnit.MILLISECONDS.sleep(pause);
      }
    } catch(InterruptedException e) {
      // Acceptable way to exit
    }
  }
}

public class E34_ColorStars extends JFrame {
  private static int grid = 12;
  private static int pause = 50;
  private static ExecutorService exec =
    Executors.newCachedThreadPool();
  public E34_ColorStars() {
    setLayout(new GridLayout(grid, grid));
    for(int i = 0; i < grid * grid; i++) {
      JStar js =
        new JStar(JStar.rand.nextInt(12) + 3, pause);
      add(js);
      exec.execute(js);
```

```
    }
  }
  public static void main(String[] args) {
    if(args.length > 0)
      E34_ColorStars.grid = new Integer(args[0]);
    if(args.length > 1)
      E34_ColorStars.pause = new Integer(args[1]);
    E34_ColorStars stars = new E34_ColorStars();
    run(stars, 500, 400);
  }
} ///:~
```

The **paintComponent( )** uses an algorithm to draw the stars and the **Graphics.fillPolygon( )** method to visualize them.

Exercise 35

```
//: gui/E35_LeftToReader.java
/****************** Exercise 35 ******************
 * Locate and download one or more of the free
 * GUI builder development environments available
 * on the Internet, or use a commercial product if
 * you own one. Discover what is necessary to add
 * BangBean to this environment and to use it.
 ***********************************************/
package gui;

public class E35_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Left to the reader");
  }
} ///:~
```

Exercise 36

```
//: gui/E36_LeftToReader.java
/****************** Exercise 36 ******************
 * Add Frog.class to the manifest file in this
 * section and run jar to create a JAR file containing
 * both Frog and BangBean. Now either download and
 * install the Bean Builder from Sun, or use your
 * own Beans-enabled program builder tool and add
 * the JAR file to your environment so you can test
 * the two Beans.
 ***********************************************/
```

```
package gui;

public class E36_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Left to the reader");
  }
} ///:~
```

Exercise 37

```
//: gui/valve/E37_Valve.java
/****************** Exercise 37 *****************
 * Create your own JavaBean called Valve that
 * contains two properties: a boolean called "on"
 * and an int called "level." Create a manifest file,
 * use jar to package your Bean, then load it into
 * the Bean Builder or into a Beans-enabled
 * program builder tool so that you can test it.
***********************************************/
// To test the program, run the following command
// in directory gui:
// java gui.valve.E37_Valve
// To make the jar file, run the following command
// in the directory above valve:
// jar cfm E37_Valve.jar E37_Valve.mf valve
package gui.valve;

public class E37_Valve implements java.io.Serializable {
  private boolean on;
  private int level;
  public E37_Valve() {}
  public E37_Valve(boolean on, int level) {
    this.on = on;
    this.level = level;
  }
  public boolean isOn() { return on; }
  public void setOn(boolean on) { this.on = on; }
  public int getLevel() { return level; }
  public void setLevel(int level) { this.level = level; }
  public static void main(String args[]) {
    E37_Valve v = new E37_Valve(true, 100);
    System.out.println("v.isOn() = " + v.isOn());
    System.out.println("v.getLevel() = " + v.getLevel());
  }
} ///:~
```

Note the organization of the file inside its package. Here is the manifest file:

```
//:! gui/E37_Valve.mf
Manifest-Version: 1.0

Name: valve/E37_Valve.class
Java-Bean: True
///:~
```

Run the **jar** command from the root of the code tree using the commands shown as comments in **E37_Valve.java**. Validate the resulting **jar** file by loading it into any Beans-enabled editor (we use Sun's Bean Builder).

Exercise 38

```
//: gui/E38_LeftToReader.java
/****************** Exercise 38 *****************
 * Build the "simple example of data binding syntax"
 * shown above (see the book).
 ***********************************************/
package gui;

public class E38_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Left to the reader");
  }
} ///:~
```

Exercise 39

```
//: gui/E39_LeftToReader.java
/****************** Exercise 39 *****************
 * The code download for this book does not include
 * the MP3s or JPGs shown in SongService.java. Find
 * some MP3s and JPGs, modify SongService.java to
 * include their file names, download the Flex trial
 * and build the application.
 ***********************************************/
package gui;

public class E39_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Left to the reader");
  }
} ///:~
```

Exercise 40

```
//: gui/E40_DisplayProperties2.java
// {Requires: org.eclipse.swt.widgets.Display; You must
// install the SWT library from http://www.eclipse.org }
/****************** Exercise 40 *****************
 * Modify DisplayProperties.java so that it uses
 * SWTConsole.
 ***********************************************/
package gui;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import java.io.*;
import swt.util.*;

public
class E40_DisplayProperties2 implements SWTApplication {
  public void createContents(Composite parent) {
    parent.setLayout(new FillLayout());
    Text text = new Text(parent, SWT.WRAP | SWT.V_SCROLL);
    StringWriter props = new StringWriter();
    System.getProperties().list(new PrintWriter(props));
    text.setText(props.toString());
  }
  public static void main(String [] args) {
    SWTConsole.run(new E40_DisplayProperties2(), 400, 300);
  }
} ///:~
```

Exercise 41

```
//: gui/E41_DisplayEnvironment2.java
// {Requires: org.eclipse.swt.widgets.Display; You must
// install the SWT library from http://www.eclipse.org }
/****************** Exercise 41 *****************
 * Modify DisplayEnvironment.java so that it does
 * not use SWTConsole.
 ***********************************************/
package gui;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import java.util.*;
```

```
public class E41_DisplayEnvironment2 {
  public static void main(String [] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("Display Environment");
    shell.setLayout(new FillLayout());
    Text text = new Text(shell, SWT.WRAP | SWT.V_SCROLL);
    for(Map.Entry<String, String> entry:
        System.getenv().entrySet()) {
      text.append(entry.getKey() + ": " +
        entry.getValue() + "\n");
    }
    shell.open();
    while(!shell.isDisposed())
      if(!display.readAndDispatch())
        display.sleep();
    display.dispose();
  }
} ///:~
```

Exercise 42

```
//: gui/E42_ColorStars2.java
// {Requires: org.eclipse.swt.widgets.Display; You must
// install the SWT library from http://www.eclipse.org }
/****************** Exercise 42 ******************
 * Modify swt/ColorBoxes.java so that it begins by
 * sprinkling points ("stars") across the canvas,
 * then randomly changes the colors of those "stars."
 ***********************************************/
package gui;
import swt.util.*;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import java.util.concurrent.*;
import java.util.*;
import net.mindview.util.*;

class CStar extends Canvas implements Runnable {
  class CStarPaintListener implements PaintListener {
    public void paintControl(PaintEvent e) {
      Color color = new Color(e.display, cColor);
      e.gc.setBackground(color);
```

```
      Point size = getSize();
      int width = size.x;
      int height = size.y;
      // The coordinates of the centre of the star:
      int centreX = width / 2;
      int centreY = height / 2;
      // The distance from the centre to points of the star
      // and to the inner core of the star, respectively:
      int outerRing, innerRing;
      if(width < height) {
        innerRing = width / 8;
        outerRing = width / 2;
      } else {
        innerRing = height / 8;
        outerRing = height / 2;
      }
      int d;
      int idx;
      for(int i = 0; i < num_points * 2; i++) {
        d = (i % 2 == 0) ? innerRing : outerRing;
        idx = i * 2;
        p[idx] = centreX + (int)(d * dx[i]);
        p[idx + 1] = centreY + (int)(d * dy[i]);
      }
      e.gc.fillPolygon(p);
      color.dispose();
    }
  }
  static Random rand = new Random();
  private static RGB newColor() {
    return new RGB(rand.nextInt(255),
      rand.nextInt(255), rand.nextInt(255));
  }
  private final int pause;
  private RGB cColor = newColor();
  private final int num_points;
  // Fields used by the drawing algorithm:
  private final int[] p;
  private final double[] dx;
  private final double[] dy;
  public
  CStar(Composite parent, int num_points, int pause) {
    super(parent, SWT.NONE);
    this.num_points = num_points;
    this.pause = pause;
    addPaintListener(new CStarPaintListener());
    p = new int[num_points * 4];
```

```
    dx = new double[num_points * 2];
    dy = new double[num_points * 2];
    // Cache calculated data:
    double delta = Math.PI / num_points;
    for(int i = 0; i < num_points * 2; i++) {
      dx[i] = Math.cos(delta * i);
      dy[i] = Math.sin(delta * i);
    }
  }
  public void run() {
    try {
      while(!Thread.interrupted()) {
        cColor = newColor();
        getDisplay().asyncExec(new Runnable() {
          public void run() {
            try { redraw(); } catch(SWTException e) {}
            // SWTException is OK when the parent
            // is terminated from under us.
          }
        });
        TimeUnit.MILLISECONDS.sleep(pause);
      }
    } catch(InterruptedException e) {
      // Acceptable way to exit
    } catch(SWTException e) {
      // Acceptable way to exit: our parent
      // was terminated from under us.
    }
  }
}

public class E42_ColorStars2 implements SWTApplication {
  private int grid = 12;
  private int pause = 50;
  public void createContents(Composite parent) {
    GridLayout gridLayout = new GridLayout(grid, true);
    gridLayout.horizontalSpacing = 0;
    gridLayout.verticalSpacing = 0;
    parent.setLayout(gridLayout);
    ExecutorService exec = new DaemonThreadPoolExecutor();
    for(int i = 0; i < (grid * grid); i++) {
      final CStar cs = new CStar(parent,
        CStar.rand.nextInt(12) + 3, pause);
      cs.setLayoutData(new GridData(GridData.FILL_BOTH));
      exec.execute(cs);
    }
  }
```

```
  public static void main(String[] args) {
    E42_ColorStars2 stars = new E42_ColorStars2();
    if(args.length > 0)
      stars.grid = new Integer(args[0]);
    if(args.length > 1)
      stars.pause = new Integer(args[1]);
    SWTConsole.run(stars, 500, 400);
  }
} ///:~
```

Notice how the performance degrades as you increase the grid. In this respect, the Swing version seems better.

Exercise 43

```
//: gui/E43_LeftToReader.java
/****************** Exercise 43 ******************
 * Choose any one of the Swing examples that wasn't
 * translated in this section, and translate it to
 * SWT.
 ***********************************************/
package gui;

public class E43_LeftToReader {
  public static void main(String args[]) {
    System.out.println("Left to the reader");
  }
} ///:~
```