Using Classes and Objects

• To create interesting/powerful programs, we can (and need to)
  ▪ use predefined classes and related objects (chap. 3) (such as Scanner, System.out, String, Random, Graphics…)
  ▪ write and use our own classes (chap. 4)
    • Extending (inheriting) other classes (chap. 8)

• Chapter 3 focuses on:
  ▪ object creation and object references
  ▪ the String class and its methods
  ▪ the Java standard class library
  ▪ the Random and Math classes
  ▪ formatting output
  ▪ enumerated types (skipped)
  ▪ wrapper classes (skipped)
  ▪ graphical components and containers
  ▪ labels and images
Creating Objects

- A variable holds either a primitive type or a reference to an object

- A class name can be used as a type to declare an object reference variable
  
  ```java
  String title;
  ```

- No object is created with this declaration

- An object reference variable holds the address of an object

- The object itself must be created separately
Creating Objects

- Generally, we use the `new` operator to create an object
  
  \[
  \text{title} = \text{new String ("Java Software Solutions");}
  \]
  
  This calls the String `constructor`, which is a special method that sets up the object

- Creating an object is called *instantiation*

- An object is an *instance* of a particular class

Invoking Methods

- We’ve seen that once an object has been instantiated, we can use the `dot operator` to invoke its methods
  
  \[
  \text{count} = \text{title.length();}
  \]

- A method may *return a value*, which can be used in an assignment or expression

- A method invocation can be thought of as asking an object to perform a *service (function)*
References

- Note that a primitive variable contains the value itself, but an object variable contains the address of the object.
- An object reference can be thought of as a pointer to the location of the object.
- Rather than dealing with arbitrary addresses, we often depict a reference graphically.

![Reference Diagram]

Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable.
- For primitive types:

  ![Assignment Diagram]
Reference Assignment

- For object references, assignment copies the address:

  Before:

  \[
  \text{name1} \rightarrow "Steve Jobs"
  \]

  \[
  \text{name2} \rightarrow "Steve Wozniak"
  \]

  \[
  \text{name2} = \text{name1};
  \]

  After:

  \[
  \text{name1} \rightarrow "Steve Jobs"
  \]

  \[
  \text{name2} \rightarrow "Steve Jobs"
  \]

Aliases

- Two or more references that refer to the same object are called aliases of each other

- That creates an interesting situation: one object can be accessed using multiple reference variables

- Aliases can be useful, but should be managed carefully

- Changing an object through one reference changes it for all of its aliases, because there is really only one object
Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program
- The object is useless, and therefore is called garbage
- Java performs automatic garbage collection periodically, returning an object's memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection

Outline

Creating Objects
The String Class
Packages
Formatting Output
Enumerated Types
Wrapper Classes
Components and Containers
Images
The String Class

• Because strings are so common, we don't have to use the new operator to create a String object

   title = "Java Software Solutions";

• This is special syntax that works only for strings

• Each string literal (enclosed in double quotes) represents a String object

String Methods

• Once a String object has been created, neither its value nor its length can be changed

• Thus we say that an object of the String class is immutable

• However, several methods of the String class return new String objects that are modified versions of the original

• See the list of String methods on page 119 and in Appendix M
String Indexes

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric index
- The indexes begin at zero in each string
- In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4
- See StringMutation.java (page 120)

StringMutation.java (segments)

```
String phrase = "Change is inevitable";
String mutation1, mutation2, mutation3, mutation4;

System.out.println ("Original string: ", phrase);
System.out.println ("Length of string: ", phrase.length());

mutation1 = phrase.concat (", except from vending machines.");
motion2 = mutation1.toUpperCase();
motion3 = mutation2.replace ('E', 'X');
motion4 = mutation3.substring (3, 30);

// Print each mutated string
System.out.println ("Mutation #1: ", mutation1);
System.out.println ("Mutation #2: ", mutation2);
System.out.println ("Mutation #3: ", mutation3);
System.out.println ("Mutation #4: ", mutation4);

System.out.println ("Mutated length: ", mutation4.length());
```
Output from StringMutation.java

Original string: "Change is inevitable"

Length of string: 20

Mutation #1: Change is inevitable, except from vending machines.

Mutation #2: CHANGE IS INEVITABLE, EXCEPT FROM VENDING MACHINES.

Mutation #3: CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS.

Mutation #4: NGX IS INXVITABLX, XXCXPT F

Mutated length: 27

Outline

Creating Objects
The String Class
Packages
Formatting Output
Enumerated Types
Wrapper Classes
Components and Containers
Images
Class Libraries

- A *class library* is a collection of classes that we can use when developing programs.
- The *Java standard class library* is part of any Java development environment.
- Its classes are not part of the Java language per se, but we rely on them heavily.
- Various classes we've already used (*System*, *Scanner*, *String*) are part of the Java standard class library.
- Other class libraries can be obtained through third party vendors, or you can create them yourself.

Packages

- The classes of the Java standard class library are organized into *packages*.
- Some of the packages in the standard class library are:

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>General support</td>
</tr>
<tr>
<td>java.applet</td>
<td>Creating applets for the web</td>
</tr>
<tr>
<td>java.awt</td>
<td>Graphics and graphical user interfaces</td>
</tr>
<tr>
<td>javax.swing</td>
<td>Additional graphics capabilities</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>javax.xml.parsers</td>
<td>XML document processing</td>
</tr>
</tbody>
</table>
The import Declaration

• When you want to use a class from a package, you could use its *fully qualified name*
  
  ```java
  java.util.Scanner
  ```

• Or you can *import* the class, and then use just the class name
  
  ```java
  import java.util.Scanner;
  ```

• To import all classes in a particular package, you can use the *wildcard character*
  
  ```java
  import java.util.*;
  ```

The import Declaration

• All classes of the *java.lang* package are imported automatically into all programs

• It's as if all programs contain the following line:
  
  ```java
  import java.lang.*;
  ```

• That's why we didn't have to import the *System* or *String* classes explicitly in earlier programs

• The *Scanner* class, on the other hand, is part of the *java.util* package, and therefore must be imported
The Random Class

- The Random class is part of the java.util package
- It provides methods that generate pseudorandom numbers
- A Random object performs complicated calculations based on a seed value to produce a stream of seemingly random values
- See RandomNumbers.java (page 125)

RandomNumbers.java (segments)

```java
import java.util.Random;
...
Random generator = new Random();
...
num1 = generator.nextInt();
System.out.println("A random integer: "+num1);
num1 = generator.nextInt(10);
System.out.println("From 0 to 9: "+num1);
num1 = generator.nextInt(10) + 1;
System.out.println("From 1 to 10: "+num1);
...
num1 = generator.nextInt(20) - 10;
System.out.println("From -10 to 9: "+num1);
num2 = generator.nextFloat();
System.out.println("A random float (between 0-1): "+num2);
num2 = generator.nextFloat() * 6; // 0.0 to 5.999999
num1 = (int)num2 + 1;
System.out.println("From 1 to 6: "+num1);
```

The Math Class

- The Math class is part of the java.lang package
- The Math class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

The Math Class

- The methods of the Math class are static methods (also called class methods)
- Static methods can be invoked through the class name – no object of the Math class is needed
  
  \[
  \text{value} = \text{Math.cos}(90) + \text{Math.sqrt}(\text{delta});
  \]
- See Quadratic.java (page 129)
  
  \[
  \text{root1} = ((-1 * b) + \text{Math.sqrt}(\text{discriminant})) / (2 * a);
  \]
- We discuss static methods further in Chapter 6
Formatting Output

- It is often necessary to format values in certain ways so that they can be presented properly
- The Java standard class library contains classes that provide formatting capabilities
- The `NumberFormat` class allows you to format values as currency or percentages
- The `DecimalFormat` class allows you to format values based on a pattern
- Both are part of the `java.text` package
Formatting Output

- The `NumberFormat` class has static methods that return a formatter object
  
  `getCurrencyInstance()`
  `getPercentInstance()`

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format

- See Purchase.java (page 131)

---

Purchase.java (segments)

```java
import java.util.Scanner;
import java.text.NumberFormat;
...
final double TAX_RATE = 0.06;  // 6% sales tax
int quantity;
double subtotal, tax, totalCost, unitPrice;
...
NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
NumberFormat fmt2 = NumberFormat.getPercentInstance();

//enter the quantity, unit price and compute the cost
...
System.out.println("Subtotal: " + fmt1.format(subtotal));
System.out.println("Tax: " + fmt1.format(tax) + " at " + fmt2.format(TAX_RATE));
System.out.println("Total: " + fmt1.format(totalCost));
```
Example output from Purchase.java

Enter the quantity: 5
Enter the unit price: 3.87
Subtotal: $19.35
Tax: $1.16 at 6%
Total: $20.51

Formatting Output

• The DecimalFormat class can be used to format a floating point value in various ways
• For example, you can specify that the number should be truncated to three decimal places
• The constructor of the DecimalFormat class takes a string that represents a pattern for the formatted number
• See CircleStats.java (page 134)
import java.util.Scanner;
import java.text.DecimalFormat;

// enter the radius, and compute the circle area and circumference
area = Math.PI * Math.pow(radius, 2);

// Round the output to three decimal places
DecimalFormat fmt = new DecimalFormat("0.###");

System.out.println("The circle\'s area: " + fmt.format(area));

Example output:
Enter the circle\'s radius: 5
The circle\'s area: 78.54         (for 78.540, trailing zeros not printed)
The circle\'s circumference: 31.416
Enumerated Types

- Java allows you to define an enumerated type, which can then be used to declare variables
- An enumerated type establishes all possible values for a variable of that type
- The values are identifiers of your own choosing
- The following declaration creates an enumerated type called Season
  ```java
  enum Season {winter, spring, summer, fall};
  ```
- Any number of values can be listed

Outline

Creating Objects
The String Class
Packages
Formatting Output
Enumerated Types
Wrapper Classes
Components and Containers
Images
Wrapper Classes

• The java.lang package contains wrapper classes that correspond to each primitive type:

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>void</td>
<td>Void</td>
</tr>
</tbody>
</table>
Graphical Applications

- Except for the applets seen in Chapter 2, the example programs we've explored thus far have been text-based.
- They are called **command-line applications**, which interact with the user using simple text prompts.
- Let's examine some **Java applications** that have **graphical components**.
- These components will serve as a foundation to programs that have true graphical user interfaces (GUIs).

GUI Components

- A **GUI component** is an object that represents a screen element such as a button or a text field.
- GUI-related classes are defined primarily in the **java.awt** and the **javax.swing** packages.
- The **Abstract Windowing Toolkit (AWT)** was the original Java GUI package.
- The **Swing** package provides additional and more versatile components.
- Both packages are needed to create a Java GUI-based program.
GUI Containers

• A GUI container is a component that is used to hold and organize other components

• A frame is a container that is used to display a GUI-based Java application

• A frame is displayed as a separate window with a title bar – it can be repositioned and resized on the screen as needed

• A panel is a container that cannot be displayed on its own but is used to organize other components

• A panel must be added to another container to be displayed

Labels

• A label is a GUI component that displays a line of text

• Labels are usually used to display information or identify other components in the interface

• Let's look at a program that organizes two labels in a panel and displays that panel in a frame

• See Authority.java (page 143)

• This program is not interactive, but the frame can be repositioned and resized
Authority.java (segments)

```java
import java.awt.*;
import javax.swing.*;
...
JFrame frame = new JFrame("Authority");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel primary = new JPanel();
    primary.setBackground(Color.yellow);
    primary.setPreferredSize(new Dimension(250, 75));
    JLabel label1 = new JLabel("Question authority,");
    JLabel label2 = new JLabel("but raise your hand first.");
    primary.add(label1);
    primary.add(label2);
    frame.getContentPane().add(primary);
    frame.pack();
    frame.setVisible(true);
```

Nested Panels

- Containers that contain other components make up the *containment hierarchy* of an interface
- This hierarchy can be as intricate as needed to create the visual effect desired
- The following example nests two panels inside a third panel – note the effect this has as the frame is resized
- See [NestedPanels.java](#) (page 145)
Images

- Images are often used in a program with a graphical interface
- Java can manage images in both JPEG and GIF formats
- As we've seen, a JLabel object can be used to display a line of text
- It can also be used to display an image
- That is, a label can be composed of text, and image, or both at the same time
Images

- The ImageIcon class is used to represent an image that is stored in a label
- The position of the text relative to the image can be set explicitly
- The alignment of the text and image within the label can be set as well
- See LabelDemo.java (page 147)

Summary

- Chapter 3 focused on:
  - object creation and object references
  - the String class and its methods
  - the Java standard class library
  - the Random and Math classes
  - formatting output
  - enumerated types
  - wrapper classes
  - graphical components and containers
  - labels and images