Chapter 3:
Using Classes and Objects

Java Software Solutions
Foundations of Program Design
Sixth Edition

by
Lewis & Loftus

Coming up: Creating Objects Revisited
Creating Objects Revisited

- 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

Coming up: Creating Objects
Creating Objects

- Generally, we use the `new` operator to create an object

```java
title = 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
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.

```
num1  38
name1  "Steve Jobs"
```
The act of assignment takes a copy of a value and stores it in a variable

For primitive types:

Before:

```
num1  38
num2  96
```

```
num2 = num1;
```

After:

```
num1  38
num2  38
```

Coming up: Reference Assignment
Reference Assignment

- For object references, assignment copies the address:

Before:

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

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

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

After:

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

See: ReferenceExample.java
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.

When you pass a reference to a method, think of it as an alias, because it is one!

Coming up: Garbage Collection
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.

Let’s look at the GarbageCollectionTester.java
Strings are different than most other objects. I was able to change the internal information in the Rectangle, but not in a String.

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.

Coming up: String Indexes
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
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.

- It is provided by Sun and we will use it 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.

Coming up: Packages
Java class libraries 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>

See: Javadocs for more packages.

Coming up: The import Declaration
When you want to use a class from a package, you could use its **fully qualified name**

```java
java.util.Scanner myScanner = new java.util.Scanner(System.in);
```

Or you can **import** the class, and then use just the class name

```java
import java.util.Scanner;
Scanner myScanner = new Scanner(System.in);
```

To import all classes in a particular package, you can use the **`*`** wildcard character

```java
import java.util.*;
```

**NOTE:** The fully qualified name of any class is the list of packages followed by the class name
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
Packages and Directories

- **myFile**
  - \dir1\otherFile.java
  - \dir2\anotherFile.java
  - \dir3\dir4\lastFile.java

- The package for each file maps to the directory structure it is in

- **myFile** is not in a package

- **otherFile** is in package “dir1” and MUST include the statement “package dir1;”

- **lastFile** is “package dir2.dir3.dir4;”

What is the fully qualified name of the lastFile class?
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
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.

Coming up: Formatting Output
Formatting Output

- The `NumberFormat` class has static methods that return a formatter object:
  
  ```java
  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`
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`
## 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>

Coming up: Wrapper Classes
Wrapper Classes

- The following declaration creates an `Integer` object which represents the integer 40 as an object


```java
Integer age = new Integer(40);
```

- An object of a wrapper class can be used in any situation where a primitive value will not suffice

- For example, some objects serve as containers of other objects

- Primitive values could not be stored in such containers, but wrapper objects could be
Wrapper Classes

- Wrapper classes also contain static methods that help manage the associated type

- For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:

  ```java
  num = Integer.parseInt(str);
  ```

- The wrapper classes often contain useful constants as well

- For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values

Coming up: Autoboxing
Autoboxing

- **Autoboxing** is the automatic conversion of a primitive value to a corresponding wrapper object:

  ```java
  Integer obj;
  int num = 42;
  obj = num;
  ```

- The assignment creates the appropriate `Integer` object

- The reverse conversion (called *unboxing*) also occurs automatically as needed