Chapter 3
Using Classes and Objects

Java Software Solutions
Foundations of Program Design
Seventh Edition

John Lewis
William Loftus
Using Classes and Objects

• We can create more interesting programs using predefined classes and related objects

• Chapter 3 focuses on:
  – object creation and object references
  – the String class and its methods
  – the Java API class library
  – the Random and Math classes
  – formatting output
  – enumerated types
  – wrapper classes
  – graphical components and containers
  – labels and images
Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Components and Containers
Images
Creating Objects

• A variable **holds** either a **primitive value** 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
- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

```java
title = new String ("Java Software Solutions");
```

This calls the String *constructor*, which is a special method that sets up the object
Invoking Methods

• We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

  ```java
  numChars = 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
Declarations

```java
int num1;
String name1;
```

- The first declaration creates a variable that **holds an integer**, while the second holds a **reference to a String**

- Initially the two variables do not contain any data (which is not the same as containing `null`) – you need to **instantiate** or **assign** the variables.

```
  num1 -
  name1 -
```
References

• Note that a primitive variable contains the value itself, but an object variable contains the address of the object (object reference)

• 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

```java
num1 = 38;
name1 = new String("Steve Jobs");
```

![Diagram with variables num1 and name1, and their values: num1 = 38, name1 pointing to "Steve Jobs"]/
Quiz

• Is there any semantic difference in these two code snippets?

```java
int num1 = 3;
String name1 = "Tim Cook";

int num1;
String name1;
num1 = 3;
name1 = new String("Tim Cook");
```
Assignment Revisited

• 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
Reference Assignment

• For object references, assignment copies the address:

Before:

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

After:

\[
\text{Evaluating a variable that stores an object returns the address of the object.}
\]
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 (state) 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.
Quiz

• Write a declaration for a `String` variable called `author`, and initialize to the string "Fred Brooks"

• Draw a graphic representation of the variable and its value (similar to those used in the previous slides)
Quiz

• Write a declaration for a `String` variable called `author`, and initialize to the string "Fred Brooks"

• Draw a graphic representation of the variable and its value

```java
String author = new String("Fred Brooks");
OR
String author = "Fred Brooks";
```

```
author  ➔  "Fred Brooks"
```
Quiz

• What is the output of these statements?

```java
String person1 = "bill";
String person2 = person1;
person2 = "john";
System.out.println(person1);
```
Quiz

- What is the output of these statements?

```java
String person1 = "bill";
String person2 = person1;
person2 = "john";
System.out.println(person1);

> bill
```
Quiz

If two variables contain aliases of the same object then:

• A) the object may be modified using either alias
• B) the object cannot be modified unless there's but a single reference to it
• C) a third alias is created if/when the object is modified
• D) the object will become an "orphan" if both variables are set to null
• E) answers A and D are correct
Quiz

If two variables contain aliases of the same object then:
• A) the object may be modified using either alias
• B) the object cannot be modified unless there's but a single reference to it
• C) a third alias is created if/when the object is modified
• D) the object will become an "orphan" if both variables are set to null
• E) answers A and D are correct
The String Class

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

```java
    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

• Therefore 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
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
public class StringMutation
{
    public static void main (String[] args)
    {
        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.");
        mutation2 = mutation1.toUpperCase();
        mutation3 = mutation2.replace ('E', 'X');
        mutation4 = mutation3.substring (3, 30);
        
        continued
continued

    // 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

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

```java
System.out.println("Mutated length: "+ mutation4.length());
}
```
Quick Check

What output is produced by the following?

```java
String str = "Space, the final frontier.";
System.out.println (str.length());
System.out.println (str.substring(7));
System.out.println (str.toUpperCase());
System.out.println (str.length());
```
Quick Check

What output is produced by the following?

```java
String str = "Space, the final frontier.";
System.out.println (str.length());
System.out.println (str.substring(7));
System.out.println (str.toUpperCase());
System.out.println (str.length());
```

```
26
the final frontier.
SPACE, THE FINAL FRONTIER.
26
```
Quiz

These two ways of setting up a String yield identical results:

a) `String string = new String("123.45");`
b) `String string = "" + 123.45;`

TRUE or FALSE?
Quiz

• These two ways of setting up a String yield identical results:

a) String string = new String("123.45");
b) String string = "" + 123.45;

TRUE
Quiz

• Write a declaration for a String variable called `change` and initialize it to the characters stored in another String object called `original` with all 'e' characters changed to 'j'.
Quiz

• Write a declaration for a String variable called change and initialize it to the characters stored in another String object called original with all 'e' characters changed to 'j'.

String change = original.replace ('e', 'j');
Outline

Creating Objects
The String Class
The Random and Math Classes
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
The Java API

• The Java class library is sometimes referred to as the Java API

• API stands for *Application Programming Interface*

• Clusters of related classes are sometimes referred to as specific APIs:
  – The Swing API
  – The Database API
The Java API

• Get comfortable navigating the online Java API documentation
Packages

• For purposes of accessing them, classes in the Java API are organized into packages

• These often overlap with specific APIs

• Examples:

<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
RandomNumbers.java
Author: Lewis/Loftus

Demonstrates the creation of pseudo-random numbers using the Random class.

import java.util.Random;

public class RandomNumbers {
    public static void main (String[] args) {
        Random generator = new Random();
        int num1;
        float num2;

        num1 = generator.nextInt();
        System.out.println("A random integer: " + num1);

        num1 = generator.nextInt(10);
        System.out.println("From 0 to 9: " + num1);
    }
}

continued
num1 = generator.nextInt(10) + 1;
System.out.println("From 1 to 10: " + num1);

num1 = generator.nextInt(15) + 20;
System.out.println("From 20 to 34: " + 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);
}
}
Sample Run

A random integer: 672981683
From 0 to 9: 0
From 1 to 10: 3
From 20 to 34: 30
From -10 to 9: -4
A random float (between 0-1): 0.18538326
From 1 to 6: 3

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);
Quick Check

Given a Random object named gen, what range of values are produced by the following expressions?

- `gen.nextInt(25)`
- `gen.nextInt(6) + 1`
- `gen.nextInt(100) + 10`
- `gen.nextInt(50) + 100`
- `gen.nextInt(10) - 5`
- `gen.nextInt(22) + 12`
Quick Check

Given a `Random` object named `gen`, what range of values are produced by the following expressions?

<table>
<thead>
<tr>
<th>Expression</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gen.nextInt(25)</code></td>
<td>0 to 24</td>
</tr>
<tr>
<td><code>gen.nextInt(6) + 1</code></td>
<td>1 to 6</td>
</tr>
<tr>
<td><code>gen.nextInt(100) + 10</code></td>
<td>10 to 109</td>
</tr>
<tr>
<td><code>gen.nextInt(50) + 100</code></td>
<td>100 to 149</td>
</tr>
<tr>
<td><code>gen.nextInt(10) - 5</code></td>
<td>-5 to 4</td>
</tr>
<tr>
<td><code>gen.nextInt(22) + 12</code></td>
<td>12 to 33</td>
</tr>
</tbody>
</table>
Quick Check

Write an expression that produces a random integer in the following ranges:

<table>
<thead>
<tr>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 12</td>
</tr>
<tr>
<td>1 to 20</td>
</tr>
<tr>
<td>15 to 20</td>
</tr>
<tr>
<td>-10 to 0</td>
</tr>
</tbody>
</table>
Quick Check

Write an expression that produces a random integer in the following ranges:

<table>
<thead>
<tr>
<th>Range</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 12</td>
<td><code>gen.nextInt(13)</code></td>
</tr>
<tr>
<td>1 to 20</td>
<td><code>gen.nextInt(20) + 1</code></td>
</tr>
<tr>
<td>15 to 20</td>
<td><code>gen.nextInt(6) + 15</code></td>
</tr>
<tr>
<td>-10 to 0</td>
<td><code>gen.nextInt(11) - 10</code></td>
</tr>
</tbody>
</table>
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 are invoked through the class name – no object of the **Math** class is needed

  \[ \text{value} = \text{Math.cos}(90) + \text{Math.sqrt}(\text{delta}); \]

• We discuss static methods further in Chapter 7

• **See Quadratic.java**
import java.util.Scanner;

public class Quadratic {
    // Determines the roots of a quadratic equation.
    public static void main (String[] args) {
        int a, b, c; // ax^2 + bx + c
        double discriminant, root1, root2;

        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter the coefficient of x squared: ");
        a = scan.nextInt();
        continued
System.out.print("Enter the coefficient of x: ");
b = scan.nextInt();

System.out.print("Enter the constant: ");
c = scan.nextInt();

// Use the quadratic formula to compute the roots.
// Assumes a positive discriminant.

discriminant = Math.pow(b, 2) - (4 * a * c);
root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

System.out.println("Root #1: "+ root1);
System.out.println("Root #2: "+ root2);
System.out.print("Enter the coefficient of x squared: ");
b = scan.nextInt();
System.out.print("Enter the coefficient of x: ");
c = scan.nextInt();

// Use the quadratic formula to compute the roots.
// Assumes a positive discriminant.

discriminant = Math.pow(b, 2) - (4 * a * c);
root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

System.out.println("Root #1: "+ root1);
System.out.println("Root #2: "+ root2);
}
Quiz

What will be displayed by this command:
System.out.println(Math.pow(3, 3-1));

- A) 9
- B) 8
- C) 6
- D) 4
- E) 27
Quiz

• What will be displayed by this command:
  \texttt{System.out.println(Math.pow(3, 3-1));}

  – A) 9 because 3-1 evaluates to 2 and Math.pow(3,2) is \(3^2\) which is 9
  – B) 8
  – C) 6
  – D) 4
  – E) 27
Outline

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

• It is often necessary to format output 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**
  
  ```java
  getCurrencyInstance()
  ```

  ```java
  getPercentInstance()
  ```

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

- See `Purchase.java`
import java.util.Scanner;
import java.text.NumberFormat;

public class Purchase
{

    public static void main(String[] args)
    {
        final double TAX_RATE = 0.06; // 6% sales tax

        int quantity;
        double subtotal, tax, totalCost, unitPrice;

        Scanner scan = new Scanner(System.in);

        continued
NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
NumberFormat fmt2 = NumberFormat.getPercentInstance();

System.out.print ("Enter the quantity: ");
quantity = scan.nextInt();

System.out.print ("Enter the unit price: ");
unitPrice = scan.nextDouble();

subtotal = quantity * unitPrice;
tax = subtotal * TAX_RATE;
totalCost = subtotal + tax;

// Print output with appropriate formatting
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));
NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
NumberFormat fmt2 = NumberFormat.getPercentInstance();

System.out.print("Enter the quantity: ");
quantity = scan.nextInt();

System.out.print("Enter the unit price: ");
unitPrice = scan.nextDouble();

subtotal = quantity * unitPrice;
tax = subtotal * TAX_RATE;
totalCost = subtotal + tax;

// Print output with appropriate formatting
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));
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
import java.util.Scanner;
import java.text.DecimalFormat;

public class CircleStats
{
    // Calculates the area and circumference of a circle given its radius.
    public static void main (String[] args)
    {
        int radius;
        double area, circumference;

        Scanner scan = new Scanner (System.in);

        continued
System.out.print("Enter the circle's radius: ");
radius = scan.nextInt();

area = Math.PI * Math.pow(radius, 2);
circumference = 2 * Math.PI * radius;

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

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

Enter the circle's radius: 5
The circle's area: 78.54
The circle's circumference: 31.416
Outline

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

• Java allows you to define an enumerated type, which can then be used to declare variables.

• An enumerated type declaration lists all possible values (objects) for a variable of that type (class).

• The values are identifiers of your own choosing.

• The following declaration creates an enumerated type called Season:

```java
class enum Season {winter, spring, summer, fall};
```

• Any number of values can be listed.
Enumerated Types

• Once a type is defined, a variable of that type can be declared:

    Season time;

• And it can be assigned a value:

    time = Season.fall;

• The values are referenced through the name of the type

• Enumerated types are *type-safe* – you cannot assign any value other than those listed
Ordinal Values

• Internally, each value of an enumerated type is stored as an integer, called its \textit{ordinal value}.

• The first value in an enumerated type has an ordinal value of zero, the second one, and so on.

• However, you cannot assign a numeric value to an enumerated type, even if it corresponds to a valid ordinal value.
Enumerated Types

• The declaration of an enumerated type is a special type of class, and each variable of that type is (stores a reference to) an object.

• The **ordinal** method returns the ordinal value of the **object**

  \[\text{time.\text{ordinal}()}\]

• The **name** method returns the name of the identifier corresponding to the object's value

  \[\text{time.\text{name}()}\]

• See **IceCream.java**
public class IceCream
{
    enum Flavor {vanilla, chocolate, strawberry, fudgeRipple, coffee, rockyRoad, mintChocolateChip, cookieDough}

    public static void main (String[] args)
    {
        Flavor cone1, cone2, cone3;

        cone1 = Flavor.rockyRoad;
        cone2 = Flavor.chocolate;

        System.out.println ("cone1 value: " + cone1);
        System.out.println ("cone1 ordinal: " + cone1.ordinal());
        System.out.println ("cone1 name: " + cone1.name());
    }
}
System.out.println ();
System.out.println ("cone2 value: " + cone2);
System.out.println ("cone2 ordinal: " + cone2.ordinal());
System.out.println ("cone2 name: " + cone2.name());

cone3 = cone1;

System.out.println ();
System.out.println ("cone3 value: " + cone3);
System.out.println ("cone3 ordinal: " + cone3.ordinal());
System.out.println ("cone3 name: " + cone3.name());
```java
continued

    cone3 = cone1;
    System.out.println();
    System.out.println("cone3 value: " + cone3);
    System.out.println("cone3 ordinal: " + cone3.ordinal());
    System.out.println("cone3 name: " + cone3.name());
}
```

**Output**

```
cone1 value: rockyRoad
cone1 ordinal: 5
cone1 name: rockyRoad
cone2 value: chocolate
cone2 ordinal: 1
cone2 name: chocolate
cone3 value: rockyRoad
cone3 ordinal: 5
cone3 name: rockyRoad
```
Quiz

- Consider the following enumeration

```c
enum Speed { FAST, MEDIUM, SLOW };
```

A) The ordinal value of MEDIUM is ?
B) The ordinal value of SLOW is ?
C) The name of the Speed enumeration whose ordinal value is zero is ?
D) `Speed sp = FAST;` is syntactically correct: ?
Quiz

• Consider the following enumeration

```
enum Speed { FAST, MEDIUM, SLOW };
```

A) The ordinal value of MEDIUM is 1
B) The ordinal value of SLOW is 2
C) The name of the Speed enumeration whose ordinal value is zero is FAST
D) `Speed sp = FAST;` is syntactically correct: NO
Outline

Creating Objects
The String Class
The Random and Math Classes
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>
</tbody>
</table>
Wrapper Classes

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

```
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 BUT ONLY 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);
  ```

• They 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

Copyright © 2012 Pearson Education, Inc.
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
Quick Check

Are the following assignments valid? Explain.

```java
Double value = 15.75;

Character ch = new Character('T');
char myChar = ch;
```
Quick Check

Are the following assignments valid? Explain.

```java
Double value = 15.75;
Yes. The double literal is **autoboxed** into a Double object.

Character ch = new Character('T');
char myChar = ch;

Yes, the char in the object is **unboxed** before the assignment.
```
Quiz

In addition to their usage providing a mechanism to convert (to box) primitive data into objects, what else do the wrapper classes provide?
A) enumerations
B) static constants
C) arrays to contain the data
D) exceptions
E) none of the above
Quiz

In addition to their usage providing a mechanism to convert (to box) primitive data into objects, what else do the wrapper classes provide?
A) enumerations
B) static constants
C) arrays to contain the data
D) exceptions
E) none of the above

The wrapper classes also provide static constants, like MIN_VALUE and MAX_VALUE (the smallest and largest ints).
Outline

Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Components and Containers
Images
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 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 (like a frame or another panel) to be displayed
GUI Containers

• A GUI container can be classified as either heavyweight or lightweight

• A *heavyweight container* is one that is managed by the underlying operating system

• A *lightweight container* is managed by the Java program itself

• A *frame* is a heavyweight container and a *panel* is a lightweight container
Labels

• A *label* is a GUI component that displays a line of text and/or an image

• 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

• This program is not interactive, but the frame can be repositioned and resized

• See *Authority.java*
import java.awt.*;
import javax.swing.*;

public class Authority
{
    // Displays some words of wisdom.
    public static void main (String[] args)
    {
        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));
    }
}

continued
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);
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*
import java.awt.*;
import javax.swing.*;

public class NestedPanels
{
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Nested Panels");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        // Set up first subpanel
        JPanel subPanel1 = new JPanel();
        subPanel1.setPreferredSize (new Dimension(150, 100));
        subPanel1.setBackground (Color.green);
        JLabel label1 = new JLabel ("One");
        subPanel1.add (label1);
    }

    continued
// Set up second subpanel
JPanel subPanel2 = new JPanel();
subPanel2.setPreferredSize (new Dimension(150, 100));
subPanel2.setBackground (Color.red);
JLabel label2 = new JLabel ("Two");
subPanel2.add (label2);

// Set up primary panel
JPanel primary = new JPanel();
primary.setBackground (Color.blue);
primary.add (subPanel1);
primary.add (subPanel2);

frame.getContentPane().add(primary);
frame.pack();
frame.setVisible(true);
// Set up second subpanel
JPanel subPanel2 = new JPanel();
subPanel2.setPreferredSize(new Dimension(150, 100));
subPanel2.setBackground(Color.red);
JLabel label2 = new JLabel("Two");
subPanel2.add(label2);

// Set up primary panel
JPanel primary = new JPanel();
primary.setBackground(Color.blue);
primary.add(subPanel1);
primary.add(subPanel2);
frame.getContentPane().add(primary);
frame.pack();
frame.setVisible(true);
Outline

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

- Images can be displayed in a Java program in various ways

- 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, an image, or both at the same time
Images

- The `ImageIcon` class is used to represent the image that is stored in a label

- If text is also included, 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`
import java.awt.*;
import javax.swing.*;

public class LabelDemo{
    public static void main(String[] args){
        JFrame frame = new JFrame("Label Demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        ImageIcon icon = new ImageIcon("devil.gif");

        JLabel label1, label2, label3;

        label1 = new JLabel("Devil Left", icon, SwingConstants.CENTER);
        continued
continued

```java
label2 = new JLabel("Devil Right", icon, SwingConstants.CENTER);
label2.setHorizontalTextPosition(SwingConstants.LEFT);
label2.setVerticalTextPosition(SwingConstants.BOTTOM);

label3 = new JLabel("Devil Above", icon, SwingConstants.CENTER);
label3.setHorizontalTextPosition(SwingConstants.CENTER);
label3.setVerticalTextPosition(SwingConstants.BOTTOM);

JPanel panel = new JPanel();
panel.setBackground(Color.cyan);
panel.setPreferredSize(new Dimension(200, 250));
panel.add(label1);
panel.add(label2);
panel.add(label3);

frame.getContentPane().add(panel);
frame.pack();
frame.setVisible(true);
```
label2 = new JLabel("Devil Right", icon, SwingConstants.CENTER);
label2.setHorizontalTextPosition(SwingConstants.LEFT);
label2.setVerticalTextPosition(SwingConstants.BOTTOM);

label3 = new JLabel("Devil Above", icon, SwingConstants.CENTER);
label3.setHorizontalTextPosition(SwingConstants.CENTER);
label3.setVerticalTextPosition(SwingConstants.BOTTOM);

JPanel panel = new JPanel();
panel.setBackground(Color.cyan);
panel.setPreferredSize(new Dimension(200, 250));
panel.add(label1);
panel.add(label2);
panel.add(label3);

frame.getContentPane().add(panel);
frame.pack();
frame.setVisible(true);
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