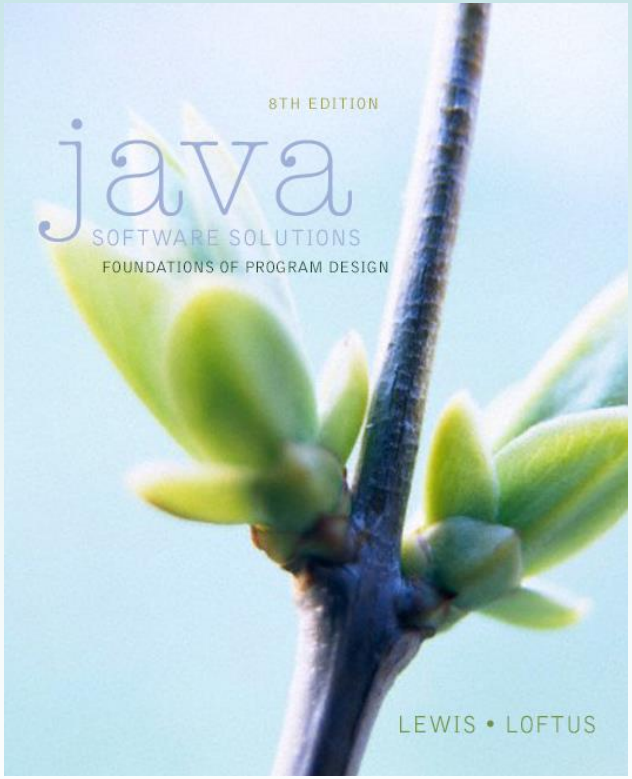
Chapter 3
Using Classes and Objects



Java Software Solutions

Foundations of Program Design

8th Edition

revised 01/10/2017

John Lewis
William Loftus

Addison-Wesley
is an imprint of
PEARSON

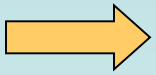
Copyright © 2014 Pearson Education, Inc.

Using Classes and 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

Copyright © 2014 Pearson Education, Inc.

Outline



Creating Objects

The String Class

The Random and Math Classes

Formatting Output

☺ **Enumerated Types**

Wrapper Classes

☺ **Components and Containers**

☺ **Images**

Copyright © 2014 Pearson Education, Inc.

Creating Objects

- A variable holds either
  - a primitive value or
  - a reference to an object
- Any one variable can only hold the type it was declared to hold
  - `int sum;` // primitive type int
  - `Scanner scan;` // object reference

Creating Objects

- To declare an *object reference variable* use the class name followed by the variable's name

```
String title;
```

- No object is created with this declaration
- An object reference variable holds the address of an object (it points to an object)
- The object itself must be created separately
- A variable can point to different objects at different times

Creating Objects

- Use the **`new`** operator to create an object
- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

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

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

Copyright © 2014 Pearson Education, Inc.

Invoking Methods

- Once an object has been instantiated, use the *dot operator* to invoke (to call) (to run) its methods

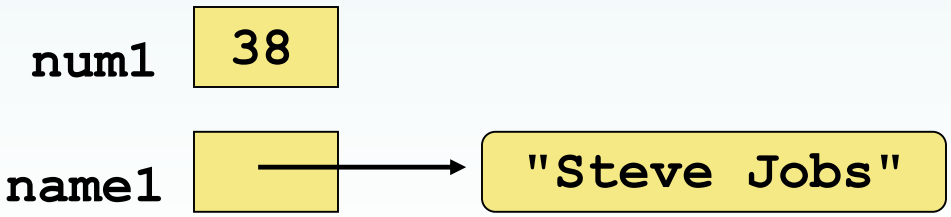
```
numChars = title.length()
```

- A method may *return a value*, which can be used in an assignment or expression
  - the value is either a primitive type, or an object reference
- A method invocation asks an object to perform a service

Copyright © 2014 Pearson Education, Inc.

References

- A primitive variable contains a value
- An object reference variable contains the address of an object
- An object reference can be thought of as a pointer to the object
- We often depict a reference graphically as a arrow:

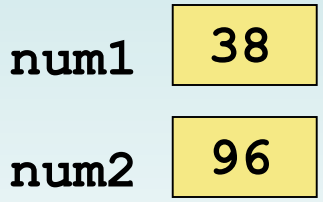


Copyright © 2014 Pearson Education, Inc.

Assignment Revisited

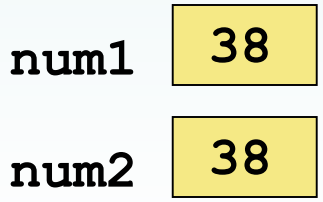
- Assignment makes a copy of a value and stores it in a variable
- For primitive types:

Before:




```
num2 = num1;
```

After:

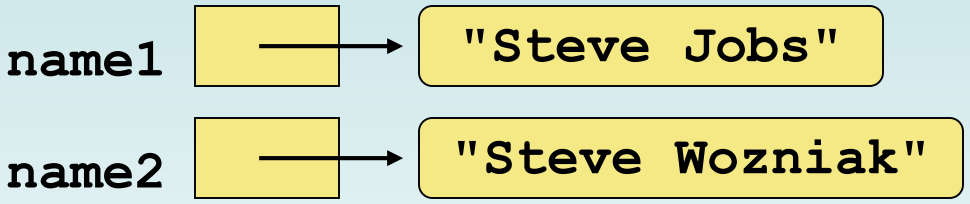


Copyright © 2014 Pearson Education, Inc.

Reference Assignment

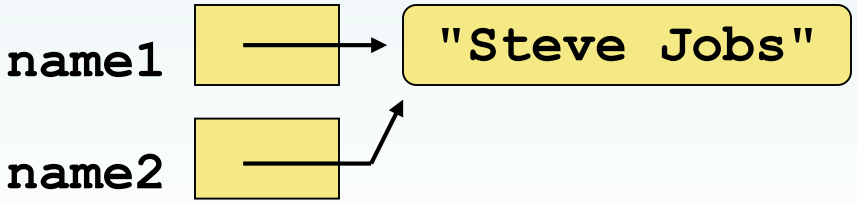
- For object references, assignment copies the address:

Before:




```
name2 = name1;
```

After:



Copyright © 2014 Pearson Education, Inc.

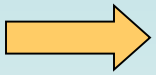
Aliases

- Two or more references that refer to the same object are called *aliases* of each other
  - Machine Gun Kelly, George Barnes
  - Lady Gaga, Stefani Germanotti
- 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 useless objects' memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection

Outline



Creating Objects

The String Class

The Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

☺ **Components and Containers**

☺ **Images**

Copyright © 2014 Pearson Education, Inc.

The String Class

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

```
String title;

title = "Java Software Solutions";
```

- This puts a reference to the string in `title`
- This is special syntax that works only for strings
- A string literal (enclosed in double quotes) corresponds to a `String` object

Copyright © 2014 Pearson Education, Inc.

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

Copyright © 2014 Pearson Education, Inc.

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*
- Indexes begin at zero
- In the string `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4
- See `StringMutation.java`

Copyright © 2014 Pearson Education, Inc.

Methods

- `toUpperCase()`
  - creates a new string, based on the original string, but with upper case characters
- `replace(char oldChar, char newChar)`
  - creates a new string resulting from replacing all occurrences of oldChar in this string with newChar.
- `substring(int beginIndex)`
  - creates a new string using chars from beginIndex to the end
- `substring(int beginIndex, int endIndex)`
  - creates a new string using chars from beginIndex to endIndex-1

```
//********************************************************************
//  StringMutation.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the String class and its methods.
//********************************************************************

public class StringMutation
{
   //-----------------------------------------------------------------
   //  Prints a string and various mutations of it.
   //-----------------------------------------------------------------
   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);
```

Copyright © 2014 Pearson Education, Inc.

```
        // 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());
    }
}
```

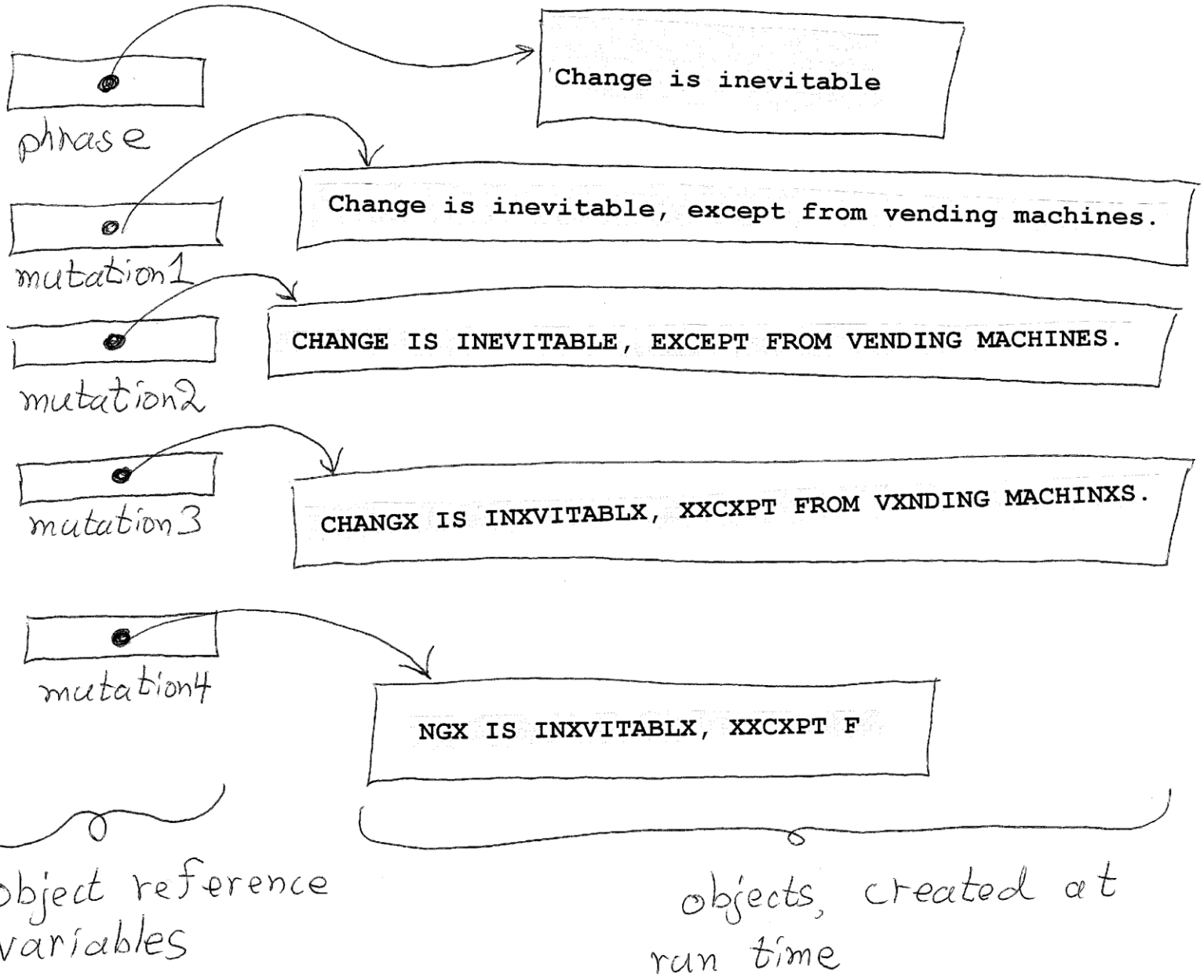
Copyright © 2014 Pearson Education, Inc.

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
```

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

Copyright © 2014 Pearson Education, Inc.


phrase
Change is inevitable
mutation1
Change is inevitable, except from vending machines.
mutation2
CHANGE IS INEVITABLE, EXCEPT FROM VENDING MACHINES.
mutation3
CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS.
mutation4
NGX IS INXVITABLX, XXCXPT F
object reference variables
objects, created at run time

Quick Check

What output is produced by the following?

```
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());
```

Copyright © 2014 Pearson Education, Inc.

Quick Check

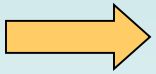
What output is produced by the following?

```
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
```

Copyright © 2014 Pearson Education, Inc.

Outline



Creating Objects

The String Class

The Random and Math Classes

Formatting Output

☺ **Enumerated Types**

Wrapper Classes

☺ **Components and Containers**

☺ **Images**

Copyright © 2014 Pearson Education, Inc.

Class Libraries

- A *class library* is a collection of classes used to develop programs
- The *Java standard class library* is part of the Java development environment
- Its classes are not part of the Java language, but are often used
- Various classes we've already used (`System`, `Scanner`, `String`) are part of the Java standard class library

Copyright © 2014 Pearson Education, Inc.

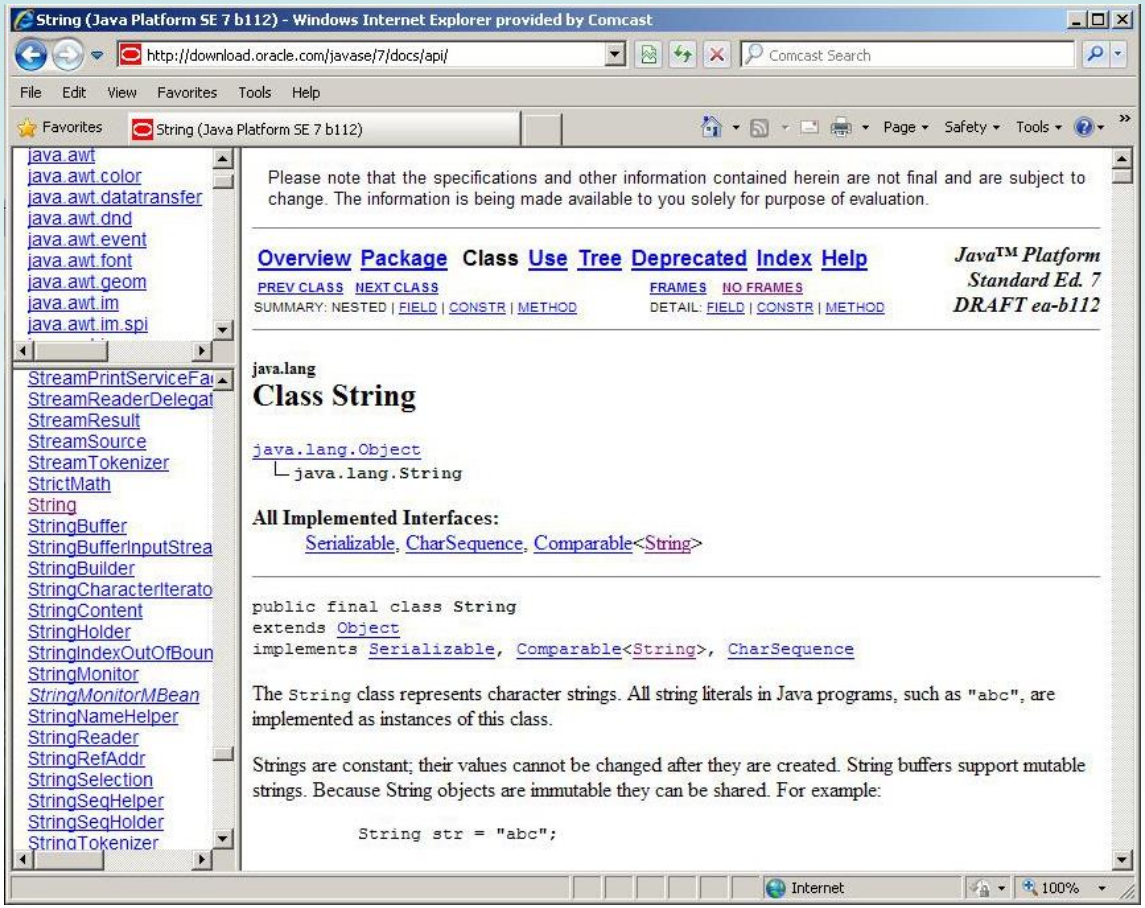
The Java API

- The Java class library is sometimes referred to as the Java API
- API stands for **A**pplication **P**rogramming **I**nterface
- Clusters of related classes are sometimes referred to as specific APIs:
  - The Swing API
  - The Database API

Copyright © 2014 Pearson Education, Inc.

The Java API

- Get comfortable navigating the online Java API documentation



Copyright © 2014 Pearson Education, Inc.

Packages

- Classes in the Java API are organized into *packages*
- Examples:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

Copyright © 2014 Pearson Education, Inc.

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner scan;
```

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

```
import java.util.Scanner;

Scanner scan;
```

- To import all classes in a particular package, use the * wildcard character

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

Copyright © 2014 Pearson Education, Inc.

The import Declaration

- All classes of the `java.lang` package are imported automatically into programs
- It's as if all programs contain the following line:

```
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

Copyright © 2014 Pearson Education, Inc.

The Random Class

- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudo random numbers
- A `Random` object performs calculations based on a *seed value* to produce a stream of seemingly random values
- See `RandomNumbers.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  RandomNumbers.java       Author: Lewis/Loftus
//
//  Demonstrates the creation of pseudo-random numbers using the
//  Random class.
//********************************************************************

import java.util.Random;

public class RandomNumbers
{
   //-----------------------------------------------------------------
   //  Generates random numbers in various ranges.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Random generator = new Random();
      int num1;
      double num2;

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

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

Copyright © 2014 Pearson Education, Inc.

```
        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.nextDouble();
        System.out.println("A random double (between 0-1): " + num2);

        num2 = generator.nextDouble() * 6;  // 0.0 to 5.999999
        num1 = (int)num2 + 1;
        System.out.println("From 1 to 6: " + num1);
    }
}
```

Copyright © 2014 Pearson Education, Inc.

continued

```
        num1
        Syst

        num1
        Syst

        num1
        Syst

        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 double (between 0-1): 0.18538326
From 1 to 6: 3
```

Copyright © 2014 Pearson Education, Inc.

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
```

Copyright © 2014 Pearson Education, Inc.

Quick Check

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

	Range
`gen.nextInt(25)`	`0 to 24`
`gen.nextInt(6) + 1`	`1 to 6`
`gen.nextInt(100) + 10`	`10 to 109`
`gen.nextInt(50) + 100`	`100 to 149`
`gen.nextInt(10) – 5`	`-5 to 4`
`gen.nextInt(22) + 12`	`12 to 33`

Copyright © 2014 Pearson Education, Inc.

Quick Check

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

Range
`0 to 12`
`1 to 20`
`15 to 20`
`-10 to 0`

Copyright © 2014 Pearson Education, Inc.

Quick Check

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

Range	
`0 to 12`	`gen.nextInt(13)`
`1 to 20`	`gen.nextInt(20) + 1`
`15 to 20`	`gen.nextInt(6) + 15`
`-10 to 0`	`gen.nextInt(11) - 10`

Copyright © 2014 Pearson Education, Inc.

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

Copyright © 2014 Pearson Education, Inc.

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

```
value = Math.cos(0.45) + Math.sqrt(delta);
```

- Arguments to trig methods are in radians
- Most methods expect (and return) double
- See `Quadratic.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Quadratic.java       Author: Modified from the textbook
//
//  Demonstrates the use of the Math class to perform a calculation
//  based on user input.
//********************************************************************

import java.util.Scanner;

public class Quadratic
{
   //-----------------------------------------------------------------
   //  Determines the roots of a quadratic equation.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      double 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.nextDouble();
```

Copyright © 2014 Pearson Education, Inc.

```
        System.out.print("Enter the coefficient of x: ");
        b = scan.nextDouble();

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

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

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

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

This program assumes that the discriminant is positive.
A better program would deal with all cases.

Copyright © 2014 Pearson Education, Inc.

continued

```
        System
        b = sc

        System
        c = sc

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

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

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

Sample Run

```
Enter the coefficient of x squared: 3
Enter the coefficient of x: 8
Enter the constant: 4
Root #1: -0.6666666666666666
Root #2: -2.0
```

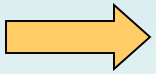
Copyright © 2014 Pearson Education, Inc.

Outline

Creating Objects

The String Class

The Random and Math Classes



Formatting Output

☺ **Enumerated Types**

Wrapper Classes

☺ **Components and Containers**

☺ **Images**

Copyright © 2014 Pearson Education, Inc.

Formatting Output

- It is often nice to format output values
- The Java standard class library contains classes that format numbers
- The `NumberFormat` class formats values as currency or as percentages
- The `DecimalFormat` class formats values based on a pattern
- Both are part of the `java.text` package

Copyright © 2014 Pearson Education, Inc.

Formatting Output

- The `NumberFormat` class has static methods that return a formatter object

```
getCurrencyInstance()

getPercentInstance()
```

- `new` is not used to create the formatter object
- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Purchase.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the NumberFormat class to format output.
//********************************************************************

import java.util.Scanner;
import java.text.NumberFormat;

public class Purchase
{
   //-----------------------------------------------------------------
   //  Calculates the final price of a purchased item using values
   //  entered by the user.
   //-----------------------------------------------------------------
   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);
```

Copyright © 2014 Pearson Education, Inc.

```
      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));
   }
}
```

Copyright © 2014 Pearson Education, Inc.

continued

```
        NumberFormat f                                    tance();
        NumberFormat f                                    ance();

        System.out.pri
        quantity = sca

        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));
    }
}
```

Sample Run

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

Copyright © 2014 Pearson Education, Inc.

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`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  CircleStats.java       Author: Lewis/Loftus
//
//  Demonstrates the formatting of decimal values using the
//  DecimalFormat class.
//********************************************************************

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

Copyright © 2014 Pearson Education, Inc.

continued

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

        area = Math.PI * radius * radius ;
        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));
    }
}
```

Copyright © 2014 Pearson Education, Inc.

Sample Run

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

continued

```
        System.ou
        radius =

        area = Math.PI * radius * radius ;
        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));
    }
}
```

Copyright © 2014 Pearson Education, Inc.

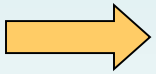
Outline

Creating Objects

The String Class

The Random and Math Classes

Formatting Output



☺ Enumerated Types

Wrapper Classes

☺ Components and Containers

☺ Images

Copyright © 2014 Pearson Education, Inc.

☺ Enumerated Types

- An *enumerated type* can be used to declare variables
- An enumerated type declaration lists all possible values for a variable of that type
- The values are identifiers of your own choosing
- To create an enumerated type called `Season`

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

- Any number of values can be listed

Copyright © 2014 Pearson Education, Inc.

☺ Enumerated Types

- A declare a variable:

```
Season time;
```

- Assign 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

Copyright © 2014 Pearson Education, Inc.

☺ Ordinal Values

- Internally, each value of an enumerated type is stored as an integer, called its *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

Copyright © 2014 Pearson Education, Inc.

☺ Enumerated Types

- An enumerated type is a special type of class
  - each variable of that type is an object
- The `ordinal` method returns the ordinal value of the object
- The `name` method returns the name of the identifier corresponding to the object's value
- See `IceCream.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  IceCream.java       Author: Lewis/Loftus
//
//  Demonstrates the use of enumerated types.
//********************************************************************

public class IceCream
{
   enum Flavor {vanilla, chocolate, strawberry, fudgeRipple, coffee,
                rockyRoad, mintChocolateChip, cookieDough}

   //-----------------------------------------------------------------
   //  Creates and uses variables of the Flavor type.
   //-----------------------------------------------------------------
   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());
```

continued

Copyright © 2014 Pearson Education, Inc.

continued

```
        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());
    }
}
```

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
```

Copyright © 2014 Pearson Education, Inc.

Outline

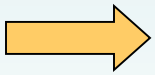
Creating Objects

The String Class

The Random and Math Classes

Formatting Output

☺ **Enumerated Types**



Wrapper Classes

☺ **Components and Containers**

☺ **Images**

Copyright © 2014 Pearson Education, Inc.

Wrapper Classes

- The `java.lang` package contains *wrapper classes* that correspond to each primitive type:

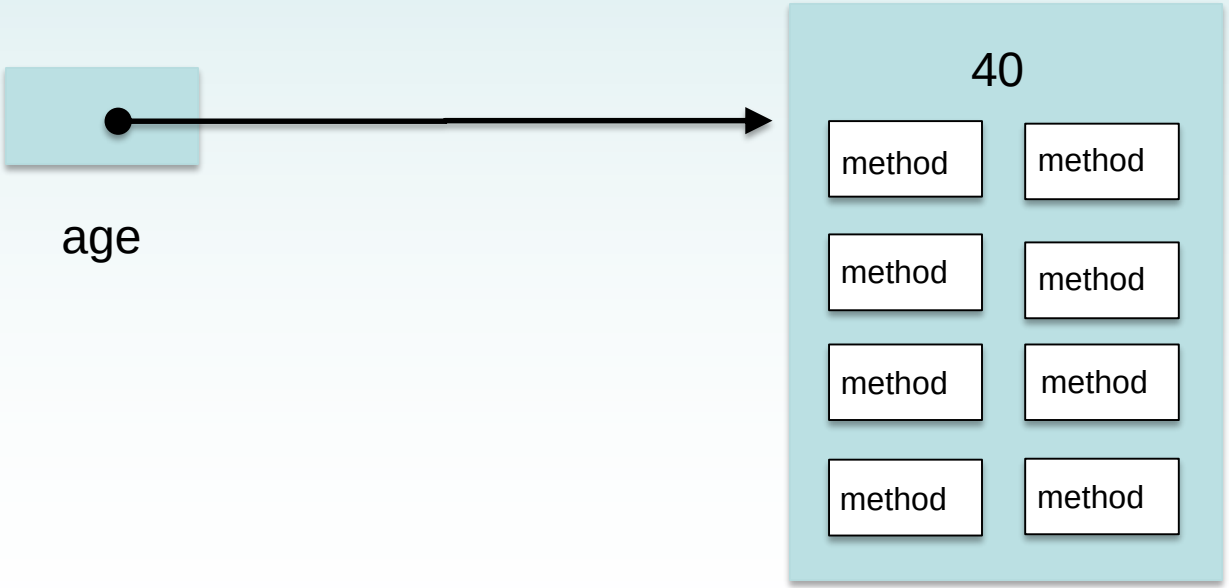
Primitive Type	Wrapper Class
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

Wrapper Classes

- A wrapper holds a primitive value along with methods and constants

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

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



Wrapper Classes

- Use a wrapper class when a primitive value will not work
- For example, some objects contain references to other objects
- Primitive values could not be stored in such containers, but wrapper objects could be

Copyright © 2014 Pearson Education, Inc.

Wrapper Classes

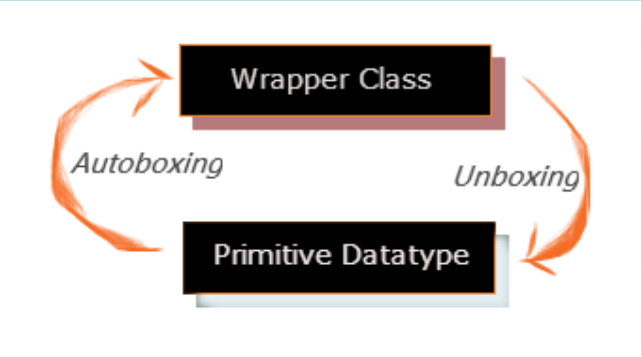
- Wrapper classes also contain static methods that help manage the associated type
- For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:

```
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 © 2014 Pearson Education, Inc.

Autoboxing



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

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

- The assignment creates the appropriate `Integer` object
- The reverse conversion (called *unboxing*) also occurs automatically as needed

Copyright © 2014 Pearson Education, Inc.

Quick Check

Are the following assignments valid? Explain.

```
Double value = 15.75;
```

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

Copyright © 2014 Pearson Education, Inc.

Quick Check

Are the following assignments valid? Explain.

```
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.

Copyright © 2014 Pearson Education, Inc.

Outline

Creating Objects

The String Class

The Random and Math Classes

Formatting Output

☺ **Enumerated Types**

Wrapper Classes

☺ **Components and Containers**

☺ **Images**

Copyright © 2014 Pearson Education, Inc.

☺ Graphical Applications

- Except for the applets, example programs have been text-based
- They are *command-line applications*, which interact with the user using text prompts
- Java applications can have graphical components
- These components are used in *graphical user interfaces* (GUIs)

Copyright © 2014 Pearson Education, Inc.

☺ GUI Components

- A *GUI component* is an object that represents a screen element such as a button, a text field, or a panel
  - At run-time, there are software objects that the program manipulates
  - The software objects are displayed on the screen
  - Don’t confuse the display (which is just a picture) with the object (which is a software object, with data and methods)
    - Clicking on a button (on the screen) invokes a method of a Button object (in main storage)

Copyright © 2014 Pearson Education, Inc.

☺ GUI Components

- GUI-related classes are defined 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

Copyright © 2014 Pearson Education, Inc.

☺ 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

Copyright © 2014 Pearson Education, Inc.

☺ 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 frame is a heavyweight container
- A *lightweight container* is managed by the Java program itself
  - A panel is a lightweight container
- Occasionally this distinction is important

Copyright © 2014 Pearson Education, Inc.

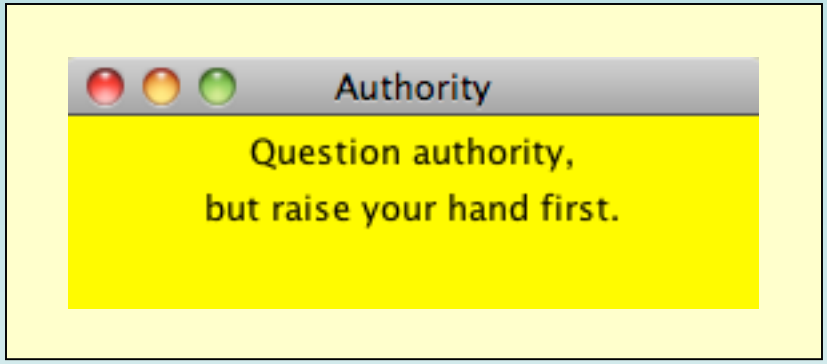
☺ Labels

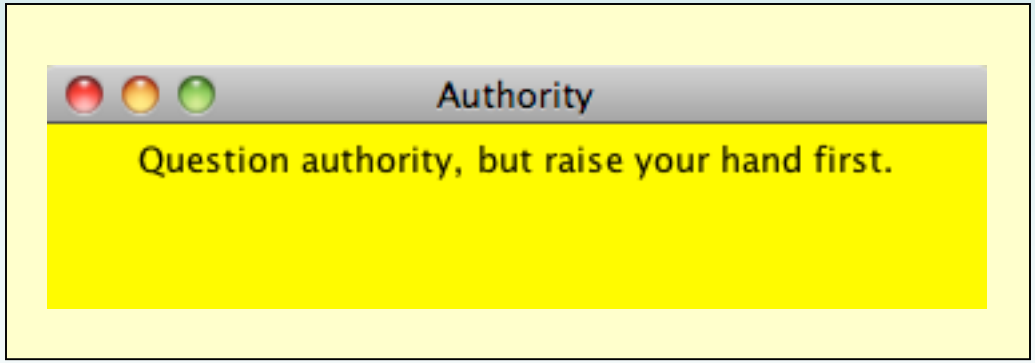
- A *label* is a GUI component that displays a line of text and/or an image
- Labels display information or identify other components
- See `Authority.java`
  - puts two labels in a panel and displays that panel in a frame
- This program is not interactive, but the frame can be repositioned and resized

Copyright © 2014 Pearson Education, Inc.

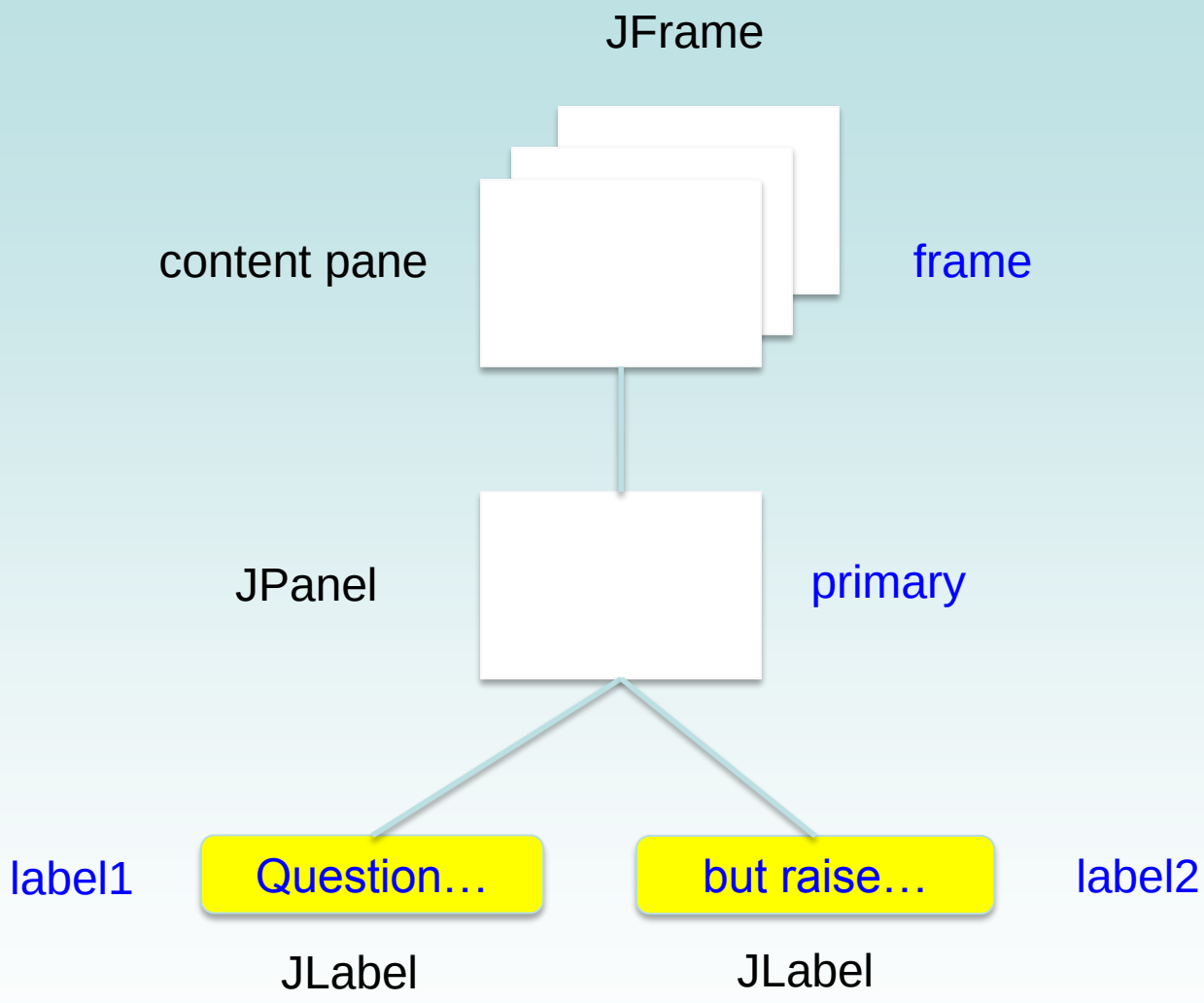
```
import java.awt.*;
import javax.swing.*;

public class Authority
{
   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));
      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);
   }
}
```

Copyright © 2014 Pearson Education, Inc.





Copyright © 2014 Pearson Education, Inc.



Copyright © 2014 Pearson Education, Inc.

☺ Details

- A frame has several panes. GUI components should be added to the content pane.
- Adding a component to a container is how you put it into the container
  - Visually, this means the component will be displayed inside the area used to display the container
  - The Layout Manager decides where components are displayed
    - Our program uses a default layout manager
    - You can pick a different layout manager

☺ More Details

- Add a panel to a frame, then add further components to the panel.
- The `setPreferredSize()` method gives the initial size of a component
  - The user can click and drag to change the size
- The `frame.pack()` method compacts the display
- The `frame.setVisible()` method asks to display the result.
  - Forget this, and you see nothing.

☺ Close Button X

```
frame.setDefaultCloseOperation( option )
```

- Says what happens when the user clicks the close button.
- Usually option should be:

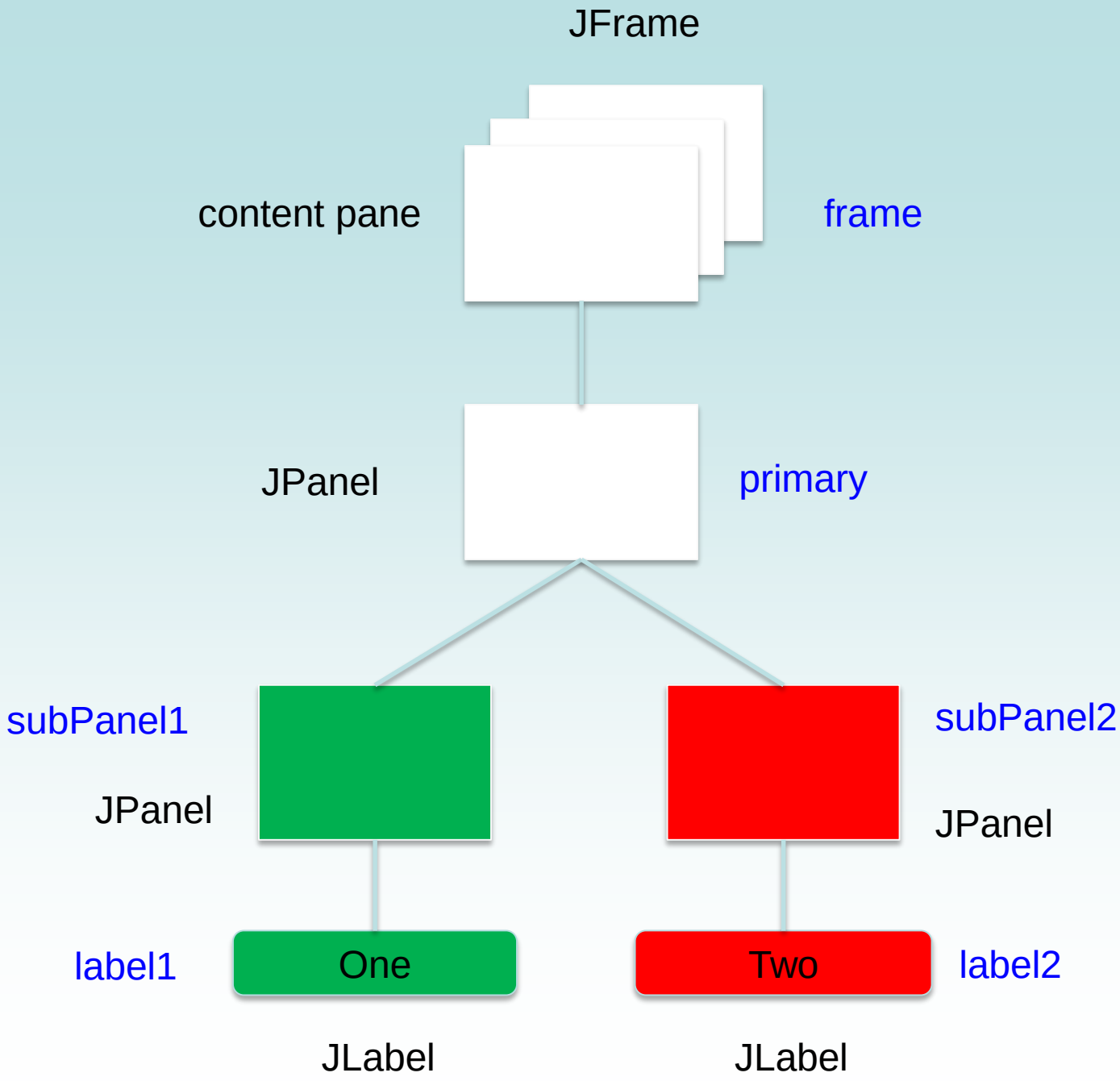
```
JFrame.EXIT_ON_CLOSE
```

- If you forget this, the program keeps running, but the visual display vanishes
  - The visual display and the component objects are different things
  - The component objects can exist without being displayed

☺ 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 when the frame is resized
  - the layout manager decides where to put components and how much padding to put around them
- See `NestedPanels.java`

Copyright © 2014 Pearson Education, Inc.


JFrame
content pane
frame
JPanel
primary
subPanel1
JPanel
subPanel2
JPanel
label1
One
Two
label2
JLabel
JLabel

```
//********************************************************************
//  NestedPanels.java       Author: Lewis/Loftus
//
//  Demonstrates a basic componenet hierarchy.
//********************************************************************

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

public class NestedPanels
{
   //-----------------------------------------------------------------
   //  Presents two colored panels nested within a third.
   //-----------------------------------------------------------------
   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);
```

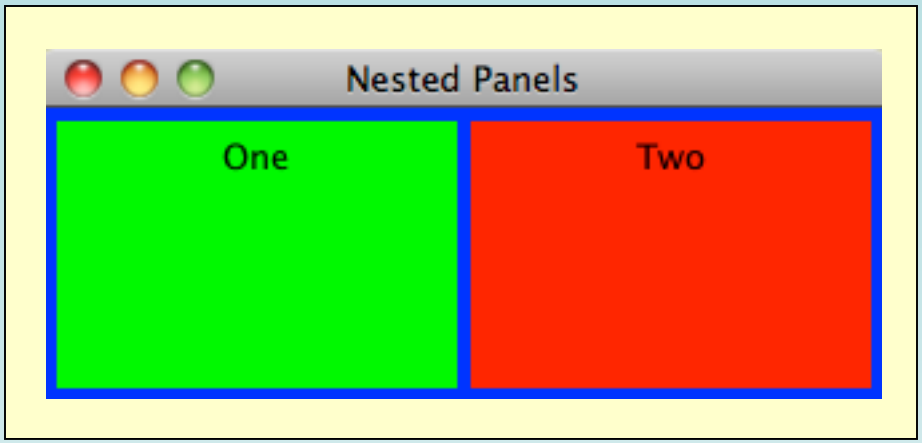
continued

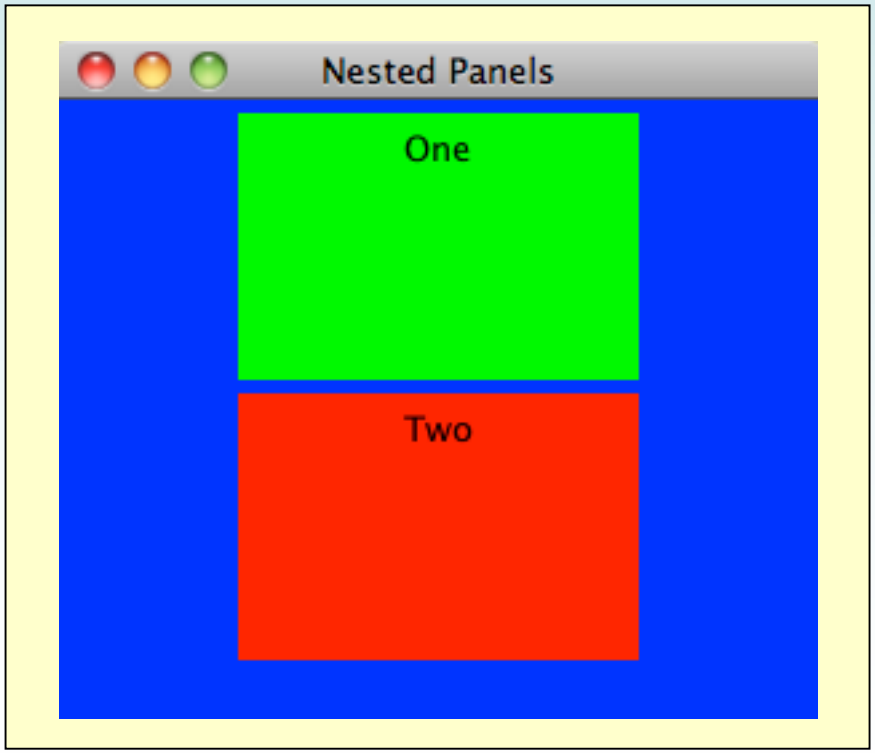
Copyright © 2014 Pearson Education, Inc.

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);
    }
}
```

Copyright © 2014 Pearson Education, Inc.





Copyright © 2014 Pearson Education, Inc.

☺ Outline

Creating Objects

The String Class

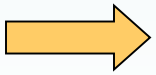
The Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

☺ Components and Containers



☺ Images

Copyright © 2014 Pearson Education, Inc.

☺ Images

- Images can be displayed in a Java program in various ways
- A `JLabel` object can display a line of text
- It can also display an image
  - Use the **`ImageIcon`** class for the image
  - Initialize with a gif or jpeg image
- A label can have text, an image, or both at the same time

Copyright © 2014 Pearson Education, Inc.

☺ Images

- Use the **`ImageIcon`** class for the image in a label
- You can set the position of the text relative to the image
- You can set the alignment of the text and image
- See `LabelDemo.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  LabelDemo.java       Author: Lewis/Loftus
//
//  Demonstrates the use of image icons in labels.
//********************************************************************

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

public class LabelDemo
{
   //-----------------------------------------------------------------
   //  Creates and displays the primary application frame.
   //-----------------------------------------------------------------
   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

Copyright © 2014 Pearson Education, Inc.

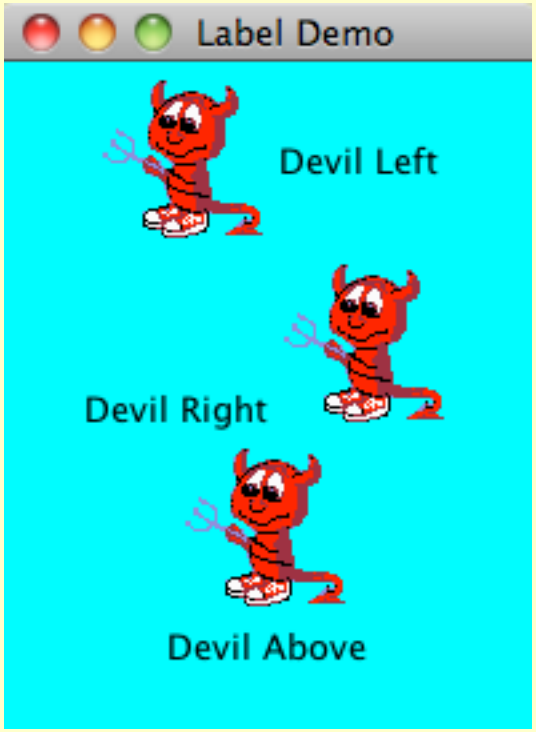
continued

```
        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);
    }
}
```

Copyright © 2014 Pearson Education, Inc.

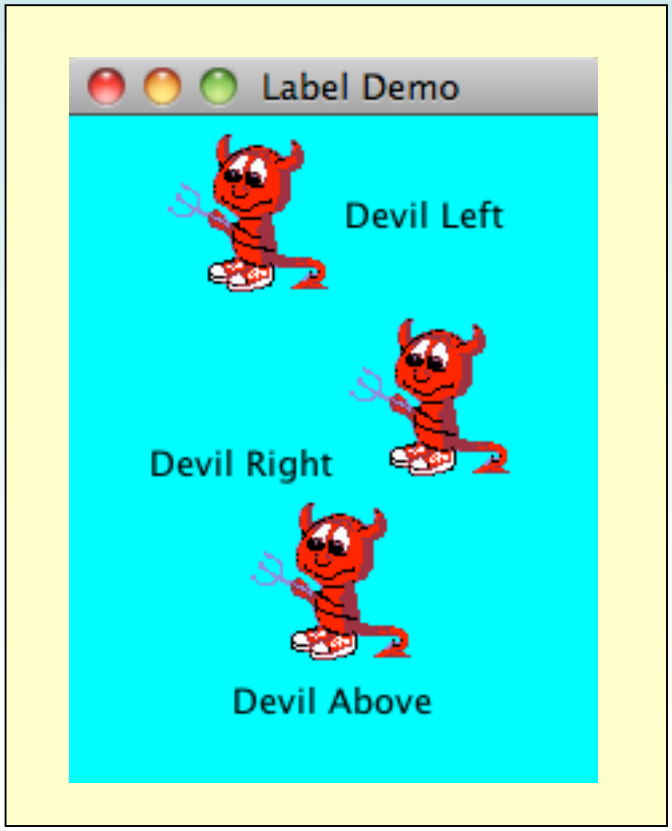
continued

```
        label2 = new JL                                    ingConstants.CENTER);
        label2.setHori                                     ants.LEFT);
        label2.setVerti                                    ts.BOTTOM);

        label3 = new JL                                    ingConstants.CENTER);
        label3.setHori                                     ants.CENTER);
        label3.setVerti                                    ts.BOTTOM);

        JPanel panel =
        panel.setBackg
        panel.setPrefe                                     250));
        panel.add(label
        panel.add(label
        panel.add(label

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



Copyright © 2014 Pearson Education, Inc.

```
// center the entire label (text and image) in the panel
label2 = new JLabel ("Devil Right", icon, SwingConstants.CENTER);

// put the text to the left inside the label
label2.setHorizontalTextPosition (SwingConstants.LEFT);

// put the text at the bottom of the label
label2.setVerticalTextPosition (SwingConstants.BOTTOM);
```



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

Copyright © 2014 Pearson Education, Inc.