Anyone who has never made a mistake has never tried anything new.
A. Einstein
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
  – JavaFX graphics API
  – shape classes
Outline

Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Introduction to JavaFX
Shapes and Color
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 **stantiate** 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 of object reference]
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:

name1 → "Steve Jobs"
name2 → "Steve Wozniak"

name2 = name1;

After:

name1 → "Steve Jobs"
name2 → "Steve Jobs"

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
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
Outline

Creating Objects

The String Class

The Random and Math Classes

Formatting Output

Enumerated Types

Wrapper Classes

Introduction to JavaFX

Shapes and Color
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
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
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 + ", except from vending machines.");
        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
// 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

    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());
```
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'.

Copyright © 2012 Pearson Education, Inc.
• 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');
Quiz*

• What will be printed?
• System.out.println("\".length());
• System.out.println("a\".length() + \"b".length());
• System.out.println("a\u0022.length() + \u0022b".length());

\u0022 is the Unicode escape for double quote ("")
Quiz*

- What will be printed?
- `System.out.println("\".length());`
- `System.out.println("a\".length() + \"b\".length());`
- `System.out.println("a\u0022.length() + \u0022b\".length());`

\u0022 is the Unicode escape for double quote (")
- nothing; compilation error
- 16
- 2
Outline

Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Introduction to JavaFX
Shapes and Color
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 JavaFX API
  – The Database API
The Java API

- Get comfortable using 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.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>javafx.scene.shape</td>
<td>Graphical shapes</td>
</tr>
<tr>
<td>javafx.scene.control</td>
<td>GUI controls</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.util.Scanner

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

  import java.util.Scanner;

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

  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**
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;
        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);
}
continued

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

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

• Are the two statements after the Random object creation semantically equivalent?

Random rnd = new Random();
System.out.println(rnd.nextInt(20));
System.out.println(rnd.nextInt()%20);
Quick Check

• Are the two statements after the Random object creation semantically equivalent?

Random rnd = new Random();
System.out.println(rnd.nextInt(20));
System.out.println(rnd.nextInt()%20);

No, because `rnd.nextInt()` can be a negative number and the remainder of the division of a negative number is negative:
Es. \(-10\%3 = -1\) \((-10/3 = -3)\)
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);
}
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

// 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) 9.0
  – B) 8.0
  – C) 6.0
  – D) 4.0
  – E) 27.0
Quiz

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

  – A) 9.0 because 3-1 evaluates to 2 and Math.pow(3,2) is $3^2$ which is 9.0
  – B) 8.0
  – C) 6.0
  – D) 4.0
  – E) 27.0
Quiz

• What is printed? The last two are difficult 😊

```java
System.out.println(Math.abs(Math.pow(-5, 3)));
System.out.println(Math.cos(0.0));
System.out.println(Math.log10(100));
System.out.println(Math.addExact(100, (int) Math.pow(10, 35)));
System.out.println(Math.getExponent(100));
```

Look in the API the description of `Math.addExact(int a, int b)` and `Math.getExponent(double a)`. 
Quiz

• What is printed?

```java
System.out.println(Math.abs(Math.pow(-5, 3)));
System.out.println(Math.cos(0.0));
System.out.println(Math.log10(100));
//System.out.println(Math.addExact(100, (int) Math.pow(10, 35)));
System.out.println(Math.getExponent(100));
```

125.0
1.0
2.0

If the fourth is executed we have: Exception in thread "main"
`java.lang.ArithmeticException: integer overflow`

6

100 = 100/64 * 2^6 = 1.5625_{10} * 2^6 = 1.1001_2 * 2^6
Outline

Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Introduction to JavaFX
Shapes and Color
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()
ggetPercentInstance()
```

• 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 {
    // 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);
        // continued
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
//********************************************************************
//  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
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));

}
System.out.println("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));
Exercise

• Write code statements that prompt for and read a double value from the user, and then print the result of raising that value to the fourth power. Output the results to 3 decimal places.
Solution

Scanner scan = new Scanner(System.in);
DecimalFormat fmt = new DecimalFormat("0.###");
System.out.println("Enter a value: ");
double numb = scan.nextDouble();
System.out.println(fmt.format(Math.pow(numb, 4)));

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
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 *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.

  ```java
  time.ordinal()
  ```

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

  ```java
  time.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());
    }
}

continued
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());
    }
}
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());
} }
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

```c
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
Introduction to JavaFX
Shapes and Color
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
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.

```
Double value = 15.75;

Character ch = new Character('T');
char myChar = ch;
```
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.
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).
Type Comparison Operator

- The `instanceof` operator compares an object to a specified type.
- You can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular `interface`.

```java
Integer obj1 = new Integer(3);
System.out.println("obj1 instanceof Integer: "+ (obj1 instanceof Integer));
```

- Prints:
  - `obj1 instanceof Integer: true`
import java.util.Random;

public class InstanceOfExample {

    public static void main(String[] args) {
        Integer obj1 = new Integer(3);
        String obj2 = "pippo";
        Object obj3 = new Random();

        System.out.println("obj1 instanceof Integer: "+ (obj1 instanceof Integer));
        // System.out.println("obj1 instanceof String: "+ (obj1 instanceof String));
        System.out.println("obj2 instanceof Object: "+ (obj2 instanceof Object));
        System.out.println("obj2 instanceof String: "+ (obj2 instanceof String));
        System.out.println("obj3 instanceof String: "+ (obj3 instanceof String));
        System.out.println("obj3 instanceof Random: "+ (obj3 instanceof Random));
    }
}
import java.util.Random;
public class InstanceOfExample {

    public static void main(String[] args) {
        Integer obj1 = new Integer(3);
        String obj2 = "pippo";
        Object obj3 = new Random();

        System.out.println("obj1 instanceof Integer: "+ (obj1 instanceof Integer));
        // System.out.println("obj1 instanceof String: "+ (obj1 instanceof String));
        System.out.println("obj1 instanceof Object: "+ (obj2 instanceof Object));
        System.out.println("obj2 instanceof String: "+ (obj2 instanceof String));
        System.out.println("obj3 instanceof String: "+ (obj3 instanceof String));
        System.out.println("obj3 instanceof Random: "+ (obj3 instanceof Random));
    }
}
Outline

Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Introduction to JavaFX
Shapes and Color
Intro to JavaFX

• The 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

• We’ll now begin to explore programs that use graphics and graphical user interfaces (GUIs)

• Support for these programs will come from the JavaFX API

• JavaFX has replaced older approaches (AWT and Swing)
Intro to JavaFX

• JavaFX programs extend the `Application` class, inheriting core graphical functionality

• A JavaFX program has a `start` method

• The `main` method is only needed to launch the JavaFX application

• The `start` method accepts the primary stage (window) used by the program as a parameter

• JavaFX embraces a theatre analogy

• See `HelloJavaFX.java`
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HelloJavaFX extends Application
{
    //---
    // Creates and displays two Text objects in a JavaFX window.
    //---
    public void start(Stage primaryStage)
    {
        Text hello = new Text(50, 50, "Hello, JavaFX!");
        Text question = new Text(120, 80, "How's it going?");

        Group root = new Group(hello, question);
        Scene scene = new Scene(root, 300, 120, Color.LIGHTGREEN);

        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

continued
continued

    primaryStage.setTitle("A JavaFX Program");
    primaryStage.setScene(scene);
    primaryStage.show();
}

//-------------------------------------------------------------------------------
// Launches the JavaFX application. This method is not required
// in IDEs that launch JavaFX applications automatically.
//-------------------------------------------------------------------------------
public static void main(String[] args)
{
    launch(args);
}
}
continued

primaryStage.setTitle("A JavaFX Program");
primaryStage.setScene(scene);
primaryStage.show();
}

//-------------------------------------------------------------------------------
// Launches the JavaFX application. This method is not required
// in IDEs that launch JavaFX applications automatically.
//-------------------------------------------------------------------------------
public static void main(String[] args)
{
    launch(args);
}
}
Intro to JavaFX

• In this example, two Text objects are added to a Group

• The group serves as the root node of a Scene

• The scene is displayed on the primary Stage (window)

• The size and background color of the scene can be set when the Scene object is created

• The position of each Text object is specified explicitly (in this case)
Intro to JavaFX

- The origin of the Java coordinate system is in the upper left corner
- All visible points have positive coordinates
Outline

Creating Objects
The String Class
The Random and Math Classes
Formatting Output
Enumerated Types
Wrapper Classes
Introduction to JavaFX
Shapes and Color
Basic Shapes

• JavaFX shapes are represented by classes in the javafx.scene.shape package

• A line segment is defined by the Line class, whose constructor accepts the coordinates of the two endpoints:

   Line(startX, startY, endX, endY)

• For example:

   Line myLine = new Line(10, 20, 300, 80);
Basic Shapes

• A rectangle is specified by its upper left corner and its width and height:

\[ \text{Rectangle}(x, y, \text{width}, \text{height}) \]

\[ \text{Rectangle} \ r = \text{new Rectangle}(30, 50, 200, 70); \]

• A circle is specified by its center point and radius:

\[ \text{Circle}(\text{centerX}, \text{centerY}, \text{radius}) \]

\[ \text{Circle} \ c = \text{new Circle}(100, 150, 40); \]
Basic Shapes

• An ellipse is specified by its center point and its radius along the x and y axis:

```java
Ellipse(centerX, centerY, radiusX, radiusY)
```

```java
Ellipse e = new Ellipse(100, 50, 80, 30);
```

• Shapes are drawn in the order in which they are added to the group

• The stroke and fill of each shape can be set

• See Einstein.java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class Einstein extends Application {

  // Creates and displays several shapes.
  public void start(Stage primaryStage) {
    Line line = new Line(35, 60, 150, 170);

    Circle circle = new Circle(100, 65, 20);
    circle.setFill(Color.BLUE);

  }

  continued
Rectangle rect = new Rectangle(60, 70, 250, 60);
rect.setStroke(Color.RED);
rect.setStrokeWidth(2);
rect.setFill(null);

Ellipse ellipse = new Ellipse(200, 100, 150, 50);
ellipse.setFill(Color.PALEGREEN);

Text quote = new Text(120, 100, "Out of clutter, find " + "simplicity.\n-- Albert Einstein");

Group root = new Group(ellipse, rect, circle, line, quote);
Scene scene = new Scene(root, 400, 200);

primaryStage.setTitle("Einstein");
primaryStage.setScene(scene);
primaryStage.show();

// We will typically exclude the main method. Use it to launch
// the application if needed.
Rectangle rect = new Rectangle(60, 70, 250, 60);
rect.setStroke(Color.RED);
rect.setStrokeWidth(2);
rect.setFill(null);

Ellipse ellipse = new Ellipse(200, 100, 150, 50);
ellipse.setFill(Color.PALEGREEN);

Text quote = new Text(120, 100, "Out of clutter, find simplicity.  
\n-- Albert Einstein");

Group root = new Group(ellipse, rect, circle, line, quote);

Scene scene = new Scene(root, 400, 200);

primaryStage.setTitle("Einstein");
primaryStage.setScene(scene);
primaryStage.show();

// We will typically exclude the main method. Use it to launch  
// the application if needed.
Basic Shapes

• Groups can be nested within groups

• *Translating* a shape or group shifts its position along the x or y axis

• A shape or group can be rotated using the `setRotate` method

• See `Snowman.java`
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;

class Snowman extends Application {

    @Override
    public void start(Stage primaryStage) {
        Ellipse base = new Ellipse(80, 210, 80, 60);
        base.setFill(Color.WHITE);

        Ellipse middle = new Ellipse(80, 130, 50, 40);
        middle.setFill(Color.WHITE);
    }
}

continued
Circle head = new Circle(80, 70, 30);
head.setFill(Color.WHITE);

Circle rightEye = new Circle(70, 60, 5);
Circle leftEye = new Circle(90, 60, 5);
Line mouth = new Line(70, 80, 90, 80);

Circle topButton = new Circle(80, 120, 6);
topButton.setFill(Color.RED);
Circle bottomButton = new Circle(80, 140, 6);
bottomButton.setFill(Color.RED);

Line leftArm = new Line(110, 130, 160, 130);
leftArm.setStrokeWidth(3);
Line rightArm = new Line(50, 130, 0, 100);
rightArm.setStrokeWidth(3);

Rectangle stovepipe = new Rectangle(60, 0, 40, 50);
Rectangle brim = new Rectangle(50, 45, 60, 5);
Group hat = new Group(stovepipe, brim);
hat.setTranslateX(10);
hat.setRotate(15);
Group snowman = new Group(base, middle, head, leftEye, rightEye, mouth, topButton, bottomButton, leftArm, rightArm, hat);
snowman.setTranslateX(170);
snowman.setTranslateY(50);

Circle sun = new Circle(50, 50, 30);
sun.setFill(Color.GOLD);

Rectangle ground = new Rectangle(0, 250, 500, 100);
ground.setFill(Color.STEELBLUE);

Group root = new Group(ground, sun, snowman);
Scene scene = new Scene(root, 500, 350, Color.LIGHTBLUE);

primaryStage.setTitle("Snowman");
primaryStage.setScene(scene);
primaryStage.show();
Group snowman = new Group(base, middle, head, leftEye, rightEye, mouth, topButton, bottomButton, leftArm, rightArm, hat);
snowman.setTranslateX(170);
snowman.setTranslateY(50);
Circle sun = new Circle(50, 50, 30);
sun.setFill(Color.GOLD);
Rectangle ground = new Rectangle(0, 250, 500, 100);
ground.setFill(Color.STEELBLUE);
Group root = new Group(ground, sun, snowman);
Scene scene = new Scene(root, 500, 350, Color.LIGHTBLUE);
primaryStage.setTitle("Snowman");
primaryStage.setScene(scene);
primaryStage.show();
Basic Shapes

• Without translating (shifting) the snowman’s position:
Representing Color

• A color in Java is represented by a `Color` object

• A color object holds three numbers called an RGB value, which stands for Red-Green-Blue

• Each number represents the contribution of that color

• This is how the human eye works

• Each number in an RGB value is in the range 0 to 255
Representing Color

• A color with an RGB value of 255, 255, 0 has a full contribution of red and green, but no blue, which is a shade of yellow

• The static `rgb` method in the `Color` class returns a `Color` object with a specific RGB value:

```java
Color purple = Color.rgb(183, 44, 150);
```

• The `color` method uses percentages:

```java
Color maroon = Color.color(0.6, 0.1, 0.0);
```
Representing Color

• For convenience, several `Color` objects have been predefined, such as:

  ```java
  Color.BLACK 0, 0, 0
  Color.WHITE 255, 255, 255
  Color.CYAN 0, 255, 255
  Color.PINK 255, 192, 203
  Color.GRAY 128, 128, 128
  ```

• See the online documentation of the `Color` class for a full list of predefined colors.
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
  – JavaFX graphics API
  – shape classes
Quiz

• What is printed?

String m = “maria”;
String n = “nicola”;
m = n;
System.out.println(m);
System.out.println(n+n == m+m);
Quiz

• What is printed?

String m = "maria";
String n = "nicola";
m = n;
System.out.println(m);
System.out.println(n+n == m+m);

nicola
false
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");
```
Quiz

• Write a statement that prints the number of characters in a String object called `overview`.
• What output is produced by the following code fragment?

```java
String m1, m2, m3;
m1 = "Quest for the Holy Grail";
m2 = m1.toLowerCase();
m3 = m1 + " " + m2;
System.out.println(m3.replace('h', 'z'));```

Copyright © 2012 Pearson Education, Inc.
Quiz

• Write a statement that prints the number of characters in a String object called `overview`.

```java
System.out.println(overview.length());
```

• What output is produced by the following code fragment?

```java
String m1, m2, m3;
m1 = "Quest for the Holy Grail";
m2 = m1.toLowerCase();
m3 = m1 + " " + m2;
System.out.println(m3.replace('h', 'z'));
```

Quest for tze Holy Grail quest for tze zoly grail
Quiz

• What is the effect of the following import statement?
  import java.awt.*;

• Write an assignment statement that computes the square root of the sum of num1 and num2 and assigns the result to num3. Hint: use the sqrt static method of the class Math.
Quiz

• What is the effect of the following import statement?
  
  import java.awt.*;
  
  – This statement allows the program in which it is written to access all classes (because of the wildcard *) in the package java.awt without any further reference to the package name.

• Write an assignment statement that computes the square root of the sum of num1 and num2 and assigns the result to num3. Hint: use the sqrt static method of the class Math.
  
  num3 = Math.sqrt(num1 + num2);
Quiz

• Write a declaration for an enumerated type that represents the days of the week.
Quiz

• Write a declaration for an enumerated type that represents the days of the week.

```java
enum Days {sunday, monday, tuesday, wednesday, thursday, friday, saturday}
```