

Chapter 3: Using Classes and Objects

Java Software Solutions
Foundations of Program Design
Sixth Edition

by
Lewis & Loftus



Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

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 standard class library
 - the `Random` and `Math` classes
 - formatting output
 - enumerated types *ej obligatoriskt*
 - wrapper classes *ej obligatoriskt*
 - graphical components and containers
 - labels and images

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-2

Creating Objects

- A variable holds either a primitive type or a *reference* to an object
- A class name can be used as a type to declare an *object reference variable*

```
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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-3

Creating Objects

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

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

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

Creating an object is called *instantiation*

An object is an *instance* of a particular class

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-4

Invoking Methods

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

```
count = title.length()
```

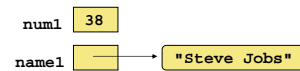
- A method may *return a value*, which can be used in an assignment or expression
- A method invocation can be thought of as asking an object to perform a service

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-5

References

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



Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-6

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
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-7

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"
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-8

Aliases

- Two or more references that refer to the same object are called *aliases* of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-9

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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-10

The String Class

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

```
title = "Java Software Solutions";
```
- This is special syntax that works only for strings
- Each string literal (enclosed in double quotes) represents a `String` object

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-11

String Methods

- Once a `String` object has been created, neither its value nor its length can be changed
- Thus we say that an object of the `String` class is *immutable*
- However, several methods of the `String` class return new `String` objects that are modified versions of the original
- See the list of `String` methods on page 119 and in Appendix M

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-12

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](#)

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-13

Class Libraries

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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-14

Packages

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

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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-15

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.*;
```

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-16

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:

```
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 © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-17

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](#)

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-18

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 © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-19

The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)
- Static methods can be invoked through the class name – no object of the `Math` class is needed

```
value = Math.cos(90) + Math.sqrt(delta);
```
- See [Quadratic.java](#)
- We discuss static methods further in Chapter 6

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-20

Formatting Output

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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-21

Formatting Output

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

```
getCurrencyInstance()
```

```
getPercentInstance()
```

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format
- See [Purchase.java](#)

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-22

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 © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-23

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 specified through the name of the type
- Enumerated types are *type-safe* – you cannot assign any value other than those listed

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-24

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 © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-25

Enumerated Types

- The declaration of an enumerated type is a special type of class, and 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 © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-26

Wrapper Classes

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

<u>Primitive Type</u>	<u>Wrapper Class</u>
<code>byte</code>	<code>Byte</code>
<code>short</code>	<code>Short</code>
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>float</code>	<code>Float</code>
<code>double</code>	<code>Double</code>
<code>char</code>	<code>Character</code>
<code>boolean</code>	<code>Boolean</code>
<code>void</code>	<code>Void</code>

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-27

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 of other objects
- Primitive values could not be stored in such containers, but wrapper objects could be

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-28

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);
```

- The wrapper classes often contain useful constants as well
- For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-29

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)

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-30

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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-31

GUI Containers

- A *GUI container* is a component that is used to hold and organize other components
- A *frame* is a container that is used to display a GUI-based Java application
- A frame is displayed as a separate window with a title bar – it can be repositioned and resized on the screen as needed
- A *panel* is a container that cannot be displayed on its own but is used to organize other components
- A panel must be added to another container to be displayed

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-32

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
- Occasionally this distinction is important
- A frame is a heavyweight container and a panel is a lightweight container

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-33

Labels

- A *label* is a GUI component that displays a line of text
- Labels are usually used to display information or identify other components in the interface
- Let's look at a program that organizes two labels in a panel and displays that panel in a frame
- See [Authority.java](#)
- This program is not interactive, but the frame can be repositioned and resized

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-34

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](#)

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-35

Images

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

Copyright © 2009 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3-36

Images

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