Chapter 2: Objects and Primitive Data

Presentation slides for

Java Software Solutions

for AP* Computer Science A
2nd Edition

by John Lewis, William Loftus, and Cara Cocking

Java Software Solutions is published by Addison-Wesley

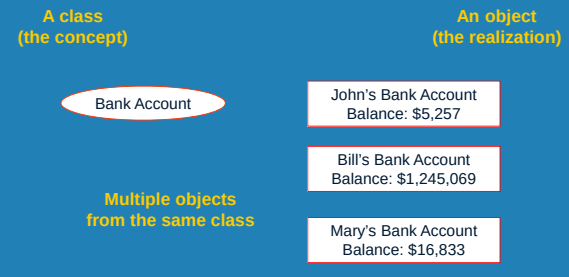
Introduction to Objects

- An *object* represents something with which we can interact in a program
- An object provides a collection of services that we can tell it to perform for us
- The services are defined by methods in a *class* that defines the object
- A class represents a concept, and an object represents the embodiment of a class
- A class can be used to create multiple objects

Object-Oriented Programming

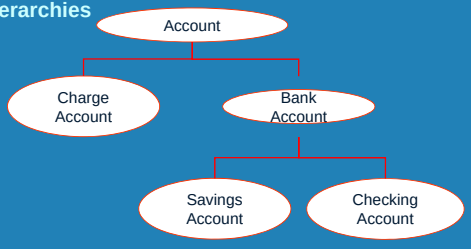
- The following concepts are important to object-oriented programming:
  - object
  - attribute
  - method
  - class
  - encapsulation
  - inheritance
  - polymorphism

Objects and Classes



Inheritance

- One class can be used to derive another via *inheritance*
- Classes can be organized into inheritance hierarchies



Using Objects

- The `System.out` object represents a destination to which we can send output
- In the `Lincoln` program, we invoked the `println` method of the `System.out` object:

```
System.out.println ("Whatever you are, be a good one.");
```

object — method — information provided to the method (parameters)

The print Method

- The `System.out` object provides another service as well
- The `print` method is similar to the `println` method, except that it does not advance to the next line
- Therefore anything printed after a `print` statement will appear on the same line
- See Countdown.java (page 61)

Abstraction

- An *abstraction* hides (or suppresses) the right details at the right time
- An object is abstract in that we don't have to think about its internal details in order to use it
- For example, we don't have to know how the `println` method works in order to invoke it
- A human being can manage only seven (plus or minus 2) pieces of information at one time
- But if we group information into chunks (such as objects) we can manage many complicated pieces at once
- Classes and objects help us write complex software

Character Strings

- Every character string is an object in Java, defined by the `String` class
- Every string literal, delimited by double quotation marks, represents a `String` object
- The *string concatenation operator* (+) is used to append one string to the end of another
- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program
- See Facts.java (page 64)

String Concatenation

- The plus operator (+) is also used for arithmetic addition
- The function that the + operator performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right
- Parentheses can be used to force the operation order
- See Addition.java (page 66)

Escape Sequences

- What if we wanted to print a double quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\), which indicates that the character(s) that follow should be treated in a special way

```
System.out.println ("I said \"Hello\" to you.");
```


Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

- See Roses.java (page 67)

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying the variable's name and the type of information that it will hold


```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in `total` is overwritten
- You can assign only a value to a variable that is consistent with the variable's declared type
- See Geometry.java (page 70)

Variables

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used
- See PianoKeys.java (page 69)

Constants

- A constant is an identifier that is similar to a variable except that it holds one value while the program is active
- The compiler will issue an error if you try to change the value of a constant during execution
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

- Constants:
  - give names to otherwise unclear literal values
  - facilitate updates of values used throughout a program
  - prevent inadvertent attempts to change a value

Primitive Data

- There are exactly eight primitive data types in Java
- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`
- Only three are in the AP subset: `int`, `double`, and `boolean`

Numeric Primitive Data

- The difference between the numeric primitive types is their size and the values they can store.
- The `int` type stores only whole numbers while `double` includes a decimal place.

Type	Storage	Min Value	Max Value
`int`	32 bits	-2,147,483,648	2,147,483,647
`double`	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	

Boolean

- A `boolean` value represents a true or false condition
- A boolean also can be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```


Characters

- A `char` variable stores a single character from the *Unicode character set*
- A *character set* is an ordered list of characters, and each character corresponds to a unique number
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages
- Character literals are delimited by single quotes:

```
'a'   'X'   '7'   '$'   ','   '\n'
```

Characters

- The *ASCII character set* is older and smaller than Unicode, but is still quite popular
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	A, B, C, …
lowercase letters	a, b, c, …
punctuation	period, semi-colon, …
digits	0, 1, 2, …
special symbols	&, |, \, …
control characters	carriage return, tab, ...

Arithmetic Expressions

- An *expression* is a combination of one or more operands and their operators
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

Division and Remainder

- If both operands to the division operator (*/*) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3      equals?   4
8 / 12      equals?   0
```

- The remainder operator (%) returns the remainder after dividing the second operand into the first

```
14 % 3      equals?   2
8 % 12      equals?   8
```


Operator Precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can be used to force the evaluation order

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4

a + b * c - d / e
  3   1   4   2

a / (b + c) - d % e
  2   1   4   3

a / (b * (c + (d - e)))
  4   3   2   1
```


Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4     1   3     2
```

Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```

Then the result is stored back into count (overwriting the original value)

Data Conversions

- Sometimes it is convenient to convert data from one type to another
- For example, we may want to treat an integer as a floating point value during a computation
- Conversions must be handled carefully to avoid losing information
- *Widening conversions* are safest because they usually do not lose information (`int` to `double`)
- *Narrowing conversions* can lose information (`double` to `int`)

Data Conversions

- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
  - Only widening conversions can happen via assignment
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands

Data Conversions

- *Casting* is the most powerful, and dangerous, technique for conversion
  - Both widening and narrowing conversions can be accomplished by explicitly casting a value
  - To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

```
result = (double) total / count;
```


Enumerated Types

- An enumerated type represents values that come from a small, fixed set, such as the seasons of the year.
- Enumerated types are specified using `enum` :

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

- Now variables of type `Season` can be declared

```
Season time;
```

- and used

```
time = Season.spring;
```


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

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

Creating Objects

- 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
- Once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
title.length()
```


String Methods

- The `String` class has several methods that are useful for manipulating strings
- Many of the methods *return a value*, such as an integer or a new `String` object
- See the list of `String` methods on page 84
- See StringMutation.java (page 86)

Wrapper Classes

- A *wrapper class* represents a particular primitive type
- For example

```
Integer ageObj = new Integer (20);
```

uses the `Integer` class to create an object which effectively represents the integer 20 as an object

- This is useful when a program requires an object instead of a primitive type
- *Autoboxing* automatically converts between wrapper classes and primitive types, so that the following is also valid:

```
Integer ageObj = 20;
```

- Methods on the `Integer` and `Double` wrapper classes are shown on page 87

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
- The `System` class and the `String` class are part of the Java standard class library
- Other class libraries can be obtained through third party vendors, or you can create them yourself

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

The import Declaration

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

```
java.util.Random
```

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

```
import java.util.Random;
```

- 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
- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- See RandomNumbers.java (page 93)

Class Methods

- Some methods can be invoked through the class name, instead of through an object of the class
- These methods are called *class methods* or *static methods*
- The `Math` class contains many static methods, providing various mathematical functions, such as absolute value, trigonometry functions, square root, etc.

```
temp = Math.cos(90) + Math.sqrt(delta);
```


Interactive Programs

- The `Scanner` class is used to get input from the user, allowing a program to be interactive
- It is part of the `java.util` package
- First a `Scanner` object is created

```
Scanner scan = new Scanner (System.in);
```

- Then various methods can be used to read different types of data from the keyboard

```
int num = scan.nextInt();
```

- See Echo.java (page 97)
- See Quadratic.java (page 98)

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 Price.java (page 100)

Formatting Output

- The `DecimalFormat` class can be used to format a floating point value in generic ways
- For example, you can specify that the number should be printed to three decimal places
- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number
- See CircleStats.java (page 102)

Applets

- A Java application is a stand-alone program with a `main` method (like the ones we've seen so far)
- A Java *applet* is a program that is intended to transported over the Web and executed using a web browser
- An applet also can be executed using the appletviewer tool of the Java Software Development Kit
- An applet doesn't have a `main` method
- Instead, there are several special methods that serve specific purposes

Applets

- The `paint` method, for instance, is executed automatically and is used to draw the applet's contents
- The `paint` method accepts a parameter that is an object of the `Graphics` class
- A `Graphics` object defines a *graphics context* on which we can draw shapes and text
- The `Graphics` class has several methods for drawing shapes

Applets

- The class that defines an applet *extends* the `Applet` class
- This makes use of *inheritance*, which is explored in more detail in Chapter 7
- See Einstein.java (page 105)
- An applet is embedded into an HTML file using a tag that references the bytecode file of the applet class
- The bytecode version of the program is transported across the web and executed by a Java interpreter that is part of the browser

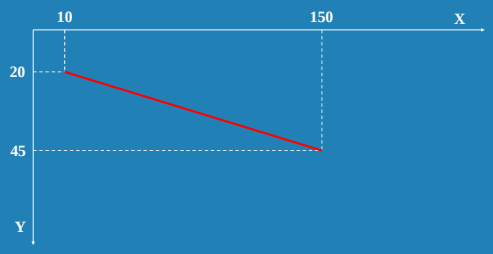
The HTML applet Tag

```
<html>

   <head>
      <title>The Einstein Applet</title>
   </head>

   <body>
      <applet code="Einstein.class" width=350 height=175>
      </applet>
   </body>

</html>
```

Drawing Shapes

- Let's explore some of the methods of the `Graphics` class that draw shapes in more detail
- A shape can be filled or unfilled, depending on which method is invoked
- The method parameters specify coordinates and sizes
- Recall from Chapter 1 that the Java coordinate system has the origin in the top left corner
- Shapes with curves, like an oval, are usually drawn by specifying the shape's *bounding rectangle*
- An arc can be thought of as a section of an oval

Drawing a Line

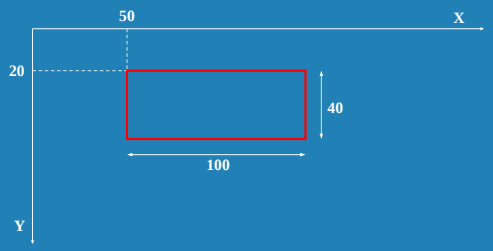



```
page.drawLine (10, 20, 150, 45);
```

or

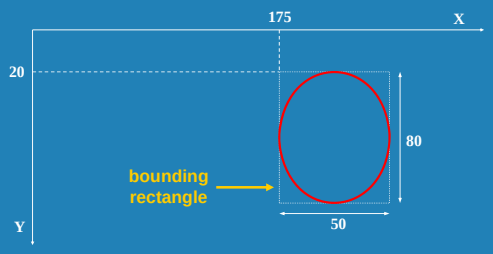
```
page.drawLine (150, 45, 10, 20);
```


Drawing a Rectangle




```
page.drawRect (50, 20, 100, 40);
```


Drawing an Oval




```
page.drawOval (175, 20, 50, 80);
```

The Color Class

- A color is defined in a Java program using an object created from the `Color` class
- The `Color` class also contains several static predefined colors, including:

Object	RGB Value
`Color.black`	`0, 0, 0`
`Color.blue`	`0, 0, 255`
`Color.cyan`	`0, 255, 255`
`Color.orange`	`255, 200, 0`
`Color.white`	`255, 255, 255`
`Color.yellow`	`255, 255, 0`

The Color Class

- Every drawing surface has a *background color*
- Every graphics context has a current *foreground color*
- Both can be set explicitly
- See Snowman.java (page110)

Summary

- Chapter 2 has focused on:
  - predefined objects
  - primitive data
  - the declaration and use of variables
  - expressions and operator precedence
  - creating and using objects
  - class libraries
  - Java applets
  - drawing shapes