Chapter 2: Objects and Primitive Data

Presentation slides for

Java Software Solutions
Foundations of Program Design
Second Edition

by John Lewis and William Loftus

Java Software Solutions is published by Addison-Wesley-Longman

Presentation slides are copyright 1999 by John Lewis and William Loftus. All rights reserved.
Instructors using the textbook may use and modify these slides for pedagogical purposes.
Objects and Primitive Data

- We can now explore some more fundamental programming concepts

- Chapter 2 focuses on:
  - predefined objects
  - primitive data
  - the declaration and use of variables
  - expressions and operator precedence
  - class libraries
  - Java applets
  - drawing shapes
Introduction to Objects

- Initially, we can think of an object as 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.
- In the Lincoln program, we invoked the `println` method of the `System.out` object:

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

Object method

Information provided to the method (parameters)
The println and print Methods

- 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 53)
Abstraction

- An abstraction hides (or ignores) the right details at the right time
- An object is abstract in that we don't really have to think about its internal details in order to use it
- We don't have to know how the `println` method works in order to invoke it
- A human being can only manage 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
- Therefore, we can write complex software by organizing it carefully into classes and objects
The String Class

- 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 56)
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 58)
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:

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>

See [Roses.java](#) (page 59)
Variables

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

```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration
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 60)
Assignment

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

\[
\text{total} = 55;
\]

- 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 only assign a value to a variable that is consistent with the variable's declared type
- See Geometry.java (page 62)
A constant is an identifier that is similar to a variable except that it holds one value for its entire existence.

The compiler will issue an error if you try to change a constant.

In Java, we use the `final` modifier to declare a constant:

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

Constants:
- give names to otherwise unclear literal values
- facilitate changes to the code
- prevent inadvertent errors
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
The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10(^{18})</td>
<td>&gt; 9 x 10(^{18})</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10(^{38}) with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10(^{308}) with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>
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'    '$'    ','    '
`
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, ...
A **boolean** value represents a true or false condition

A boolean can also 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

```java
boolean done = false;
```
Arithmetic Expressions

- An expression is a combination of operators and operands
- Arithmetic expressions compute numeric results and make use of the arithmetic operators:
  
  Addition: +
  Subtraction: -
  Multiplication: *
  Division: /
  Remainder: %

- If either or both operands to 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)

\[
14 \div 3 \quad \text{equals?} \quad 4
\]

\[
8 \div 12 \quad \text{equals?} \quad 0
\]

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

\[
14 \mod 3 \quad \text{equals?} \quad 2
\]

\[
8 \mod 12 \quad \text{equals?} \quad 8
\]
Operator Precedence

- Operators can be combined into complex expressions
  
  \[ \text{result} = \text{total} + \text{count} / \text{max} - \text{offset}; \]

- 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 always be used to force the evaluation order
Operator Precedence

What is the order of evaluation in the following expressions?

\[
\begin{align*}
\text{a + b + c + d + e} & \quad 1 \quad 2 \quad 3 \quad 4 \\
\text{a + b * c - d / e} & \quad 3 \quad 1 \quad 4 \quad 2 \\
\text{a / (b + c) - d \% e} & \quad 2 \quad 1 \quad 4 \quad 3 \\
\text{a / (b * (c + (d - e))))} & \quad 4 \quad 3 \quad 2 \quad 1
\end{align*}
\]
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

$$\text{answer} = \frac{\text{sum}}{4} + \text{MAX} \times \text{lowest};$$

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 tend to go from a small data type to a larger one (such as a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`
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 = (float) total / count;
```
Creating Objects

- A variable either holds a primitive type, or it holds 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 has been created with this declaration
- An object reference variable holds the address of an object
- The object itself must be created separately
Creating Objects

- We use the `new` operator to create an object

```java
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

  ```java
  title = "Java Software Solutions";
  ```

- This is special syntax that only works for strings

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

  ```java
  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 75 and in Appendix M.

- See **StringMutation.java** (page 77).
A class library is a collection of classes that we can use when developing programs.

There is a *Java standard class library* that is part of any Java development environment.

These 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.
The classes of the Java standard class library are organized into packages.

Some of the packages in the standard class library are:

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>General support</td>
</tr>
<tr>
<td>java.applet</td>
<td>Creating applets for the web</td>
</tr>
<tr>
<td>java.awt</td>
<td>Graphics and graphical user interfaces</td>
</tr>
<tr>
<td>javax.swing</td>
<td>Additional graphics capabilities and components</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
</tbody>
</table>
The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*
  
  ```java
  java.util.Random
  ```

- Or you can *import* the class, then just use the class name
  
  ```java
  import java.util.Random;
  ```

- To import all classes in a particular package, you can use the *wildcard* character
  
  ```java
  import java.util.*;
  ```
The import Declaration

- All classes of the java.lang package are automatically imported into all programs
- That's why we didn't have to explicitly import the System or String classes in earlier programs

- The Random class is part of the java.util package
- It provides methods that generate pseudo-random numbers
- We often have to scale and shift a number into an appropriate range for a particular purpose
- See RandomNumbers.java (page 82)
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.

  \[
  \text{temp} = \text{Math.cos}(90) + \text{Math.sqrt}(\text{delta});
  \]
The Keyboard Class

- The `Keyboard` class is NOT part of the Java standard class library.
- It is provided by the authors of the textbook to make reading input from the keyboard easy.
- Details of the `Keyboard` class are explored in Chapter 8.
- For now we will simply make use of it.
- The `Keyboard` class is part of a package called `cs1`, and contains several static methods for reading particular types of data.
- See `Echo.java` (page 86).
- See `Quadratic.java` (page 87).
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 89)
The `DecimalFormat` class can be used to format a floating point value in generic ways.

For example, you can specify that the number 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 91)
Applets

- A Java application is a stand-alone program with a `main` method (like the ones we've seen so far)
- An *applet* is a Java program that is intended to be transported over the web and executed using a web browser
- An applet can also 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
- The `paint` method, for instance, is automatically executed and is used to draw the applets 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

The class that defines the applet `extends` the Applet class

This makes use of `inheritance`, an object-oriented concept explored in more detail in Chapter 7

See `Einstein.java` (page 93)
Applets

- An applet is embedded into an HTML file using a tag that references the bytecode file of the applet class.
- It is actually the bytecode version of the program that is transported across the web.
- The applet is executed by a Java interpreter that is part of the browser.
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 upper left corner.

Many shapes with curves, like an oval, are drawn by specifying its bounding rectangle.

An arc can be thought of as a section of an oval.
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
- Every graphics context has a current foreground color
- Every drawing surface has a background color

See `Snowman.java` (page 99-100)