Advanced Programming

Review
Objects and Methods

• Java is an *object-oriented programming (OOP)* language
  – Programming methodology that views a program as consisting of *objects* that interact with one another by means of actions (called *methods*)
  – Objects of the same kind are said to have the same *type* or be in the same *class*
Java Application Programs

• A Java *application program* or "regular" Java program is a class with a method named `main`
  – When a Java application program is run, the *run-time system* automatically invokes the method named `main`
  – All Java application programs start with the `main` method
A Sample Java Application Program

Display 1.1 A Sample Java Program

```java
public class FirstProgram {
    public static void main(String[] args) {
        System.out.println("Hello reader.");
        System.out.println("Welcome to Java.");

        System.out.println("Let's demonstrate a simple calculation.");
        int answer;
        answer = 2 + 2;
        System.out.println("2 plus 2 is " + answer);
    }
}
```

**Sample Dialogue**

Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
Identifiers

• *Identifier*: The name of a variable or other item (class, method, object, etc.) defined in a program
  – A Java identifier must not start with a digit, and all the characters must be letters, digits, or the underscore symbol
  – Java identifiers can theoretically be of any length
  – Java is a case-sensitive language: *Rate, rate, and RATE* are the names of three different variables
Identifiers

• Keywords and Reserved words: Identifiers that have a predefined meaning in Java
  – Do not use them to name anything else
    public    class    void    static
Naming Conventions

• Start the names of variables and methods with a lowercase letter, indicate "word" boundaries with an uppercase letter, and restrict the remaining characters to digits and lowercase letters
  
  topSpeed   bankRate1   timeOfArrival

• Start the names of classes with an uppercase letter and, otherwise, adhere to the rules above
  
  FirstProgram   MyClass   String
Variable Declarations

• Every variable in a Java program must be *declared* before it is used
  – A variable declaration tells the compiler what kind of data (type) will be stored in the variable
  – The type of the variable is followed by one or more variable names separated by commas, and terminated with a semicolon
  – Variables are typically declared just before they are used or at the start of a block (indicated by an opening brace `{ }`
  – Basic types in Java are called *primitive types*

```java
int numberOfBeans;
double oneWeight, totalWeight;
```
## Primitive Types

<table>
<thead>
<tr>
<th>TYPE NAME</th>
<th>KIND OF VALUE</th>
<th>MEMORY USED</th>
<th>SIZE RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>1 byte</td>
<td>not applicable</td>
</tr>
<tr>
<td>char</td>
<td>single character (Unicode)</td>
<td>2 bytes</td>
<td>all Unicode characters</td>
</tr>
<tr>
<td>byte</td>
<td>integer</td>
<td>1 byte</td>
<td>−128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>integer</td>
<td>2 bytes</td>
<td>−32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>4 bytes</td>
<td>−2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>integer</td>
<td>8 bytes</td>
<td>−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>floating-point number</td>
<td>4 bytes</td>
<td>−3.40282347 × 10^{+38} to −1.40239846 × 10^{-45}</td>
</tr>
<tr>
<td>double</td>
<td>floating-point number</td>
<td>8 bytes</td>
<td>±1.767569313486231570 × 10^{+308} to ±4.94065645841246544 × 10^{-324}</td>
</tr>
</tbody>
</table>
Assignment Statements With Primitive Types

– When an assignment statement is executed, the expression is first evaluated, and then the variable on the left-hand side of the equal sign is set equal to the value of the expression

$$distance = rate \times time;$$

– Note that a variable can occur on both sides of the assignment operator

$$count = count + 2;$$

– The assignment operator is automatically executed from right-to-left, so assignment statements can be chained

$$number2 = number1 = 3;$$
### Shorthand Assignment Statements

<table>
<thead>
<tr>
<th>Example:</th>
<th>Equivalent To:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>count += 2;</code></td>
<td><code>count = count + 2;</code></td>
</tr>
<tr>
<td><code>sum -= discount;</code></td>
<td><code>sum = sum - discount;</code></td>
</tr>
<tr>
<td><code>bonus *= 2;</code></td>
<td><code>bonus = bonus * 2;</code></td>
</tr>
<tr>
<td><code>time /= rushFactor;</code></td>
<td><code>time = time / rushFactor;</code></td>
</tr>
<tr>
<td><code>change %= 100;</code></td>
<td><code>change = change % 100;</code></td>
</tr>
<tr>
<td><code>amount *= count1 + count2;</code></td>
<td><code>amount = amount * (count1 + count2);</code></td>
</tr>
</tbody>
</table>
Assignment Compatibility

• In general, the value of one type cannot be stored in a variable of another type

  \[
  \text{int intVariable} = 2.99; \ //\text{Illegal}
  \]
  
  – The above example results in a type mismatch because a \textit{double} value cannot be stored in an \textit{int} variable

• However, there are exceptions to this

  \[
  \text{double doubleVariable} = 2;
  \]
  
  – For example, an \textit{int} value can be stored in a \textit{double} type
Arithmetic Operators and Expressions

• As in most languages, *expressions* can be formed in Java using variables, constants, and arithmetic operators
  – These operators are + (addition), – (subtraction), * (multiplication), / (division), and % (modulo, remainder)
  – An expression can be used anywhere it is legal to use a value of the type produced by the expression
Arithmetic Operators and Expressions

• If an arithmetic operator is combined with \texttt{int} operands, then the resulting type is \texttt{int}

• If an arithmetic operator is combined with one or two \texttt{double} operands, then the resulting type is \texttt{double}

• If different types are combined in an expression, then the resulting type is the right-most type on the following list that is found within the expression

  \texttt{byte\rightarrow short\rightarrow int\rightarrow long\rightarrow float\rightarrow double}

  – Exception: If the type produced should be \texttt{byte} or \texttt{short} (according to the rules above), then the type produced will actually be an \texttt{int}
Precedence Rules

Display 1.3 Precedence Rules

*Highest Precedence*

First: the unary operators: +, −, ++, −−, and!
Second: the binary arithmetic operators: *, /, and %
Third: the binary arithmetic operators: + and −

*Lowest Precedence*
Integer and Floating-Point Division

• When one or both operands are a floating-point type, division results in a floating-point type
  \[15.0/2\] evaluates to \(7.5\)

• When both operands are integer types, division results in an integer type
  – Any fractional part is discarded
  – The number is not rounded
  \[15/2\] evaluates to \(7\)

• Be careful to make at least one of the operands a floating-point type if the fractional portion is needed
Type Casting

- A *type cast* takes a value of one type and produces a value of another type with an "equivalent" value
  - If \( n \) and \( m \) are integers to be divided, and the fractional portion of the result must be preserved, at least one of the two must be type cast to a floating-point type *before* the division operation is performed
    
    ```
    double ans = n / (double)m;
    ```
  - Note that the desired type is placed inside parentheses immediately in front of the variable to be cast
  - Note also that the type and value of the variable to be cast does not change
More Details About Type Casting

• When type casting from a floating-point to an integer type, the number is truncated, not rounded
  – \((\text{int})2.9\) evaluates to 2, not 3
• When the value of an integer type is assigned to a variable of a floating-point type, Java performs an automatic type cast called a *type coercion*
  
  ```java
  double d = 5;
  ```
Increment and Decrement Operators

• The *increment operator* (``++``) adds one to the value of a variable
  – If `n` is equal to 2, then `n++` or `++n` will change the value of `n` to 3

• The *decrement operator* (``--``) subtracts one from the value of a variable
  – If `n` is equal to 4, then `n--` or `--n` will change the value of `n` to 3
Increment and Decrement Operators

• When either operator precedes its variable, and is part of an expression, then the expression is evaluated using the changed value of the variable
  – If $n$ is equal to 2, then $2 \times (++n)$ evaluates to 6

• When either operator follows its variable, and is part of an expression, then the expression is evaluated using the original value of the variable, and only then is the variable value changed
  – If $n$ is equal to 2, then $2 \times (n++)$ evaluates to 4
The Class `String`

- There is no primitive type for strings in Java
- The class `String` is a predefined class in Java that is used to store and process strings
- Objects of type `String` are made up of strings of characters that are written within double quotes
  - Any quoted string is a constant of type `String`
    "Live long and prosper."
- A variable of type `String` can be given the value of a `String` object
  
  ```java
  String blessing = "Live long and prosper.";
  ```
Concatenation of Strings

- **Concatenation**: Using the `+` operator on two strings in order to connect them to form one longer string
  - If `greeting` is equal to "Hello ", and `javaClass` is equal to "class", then `greeting + javaClass` is equal to "Hello class"

- Any number of strings can be concatenated together

- When a string is combined with almost any other type of item, the result is a string
  - "The answer is " + 42 evaluates to "The answer is 42"
String Methods

• The **String** class contains many useful methods for string-processing applications
  – A **String** method is called by writing a **String** object, a dot, the name of the method, and a pair of parentheses to enclose any arguments
  – If a **String** method returns a value, then it can be placed anywhere that a value of its type can be used
    ```java
    String greeting = "Hello";
    int count = greeting.length();
    System.out.println("Length is "+
greeting.length());
    ```
  – Always count from zero when referring to the *position* or *index* of a character in a string
String Indexes

Display 1.5  String Indexes

The 12 characters in the string "Java is fun." have indexes 0 through 11.

0 1 2 3 4 5 6 7 8 9 10 11
Java is fun.

Notice that the blanks and the period count as characters in the string.
Some Methods in the Class \textbf{String} (1)

\begin{itemize}
  \item \textbf{int} \texttt{length()}
  \begin{itemize}
    \item Returns the length of the calling object (which is a string) as a value of type \texttt{int}.
  \end{itemize}
  \item \textbf{boolean} \texttt{equals(Other\_String)}
  \begin{itemize}
    \item Returns true if the calling object \texttt{string} and the \texttt{Other\_String} are equal. Otherwise, returns \texttt{false}.
  \end{itemize}
\end{itemize}

\textbf{EXAMPLE}

After program executes \texttt{String greeting = "Hello";}
greeting.length() returns 6.

After program executes \texttt{String greeting = "Hello";}
greeting.equals("Hello") returns \texttt{true}
greeting.equals("Good-Bye") returns \texttt{false}
greeting.equals("hello") returns \texttt{false}

Note that case matters. "Hello" and "hello" are not equal because one starts with an uppercase letter and the other starts with a lowercase letter.

(continued)
Some Methods in the Class `String` (2)

Display 1.4 Some Methods in the Class `String`

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean equalsIgnoreCase(Other_String)</code></td>
<td>Returns true if the calling object string and the <code>Other_String</code> are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns false.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

After program executes `String name = "mary!"; greeting.equalsIgnoreCase("Mary!") returns true`

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String toLowerCase()</code></td>
<td>Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

After program executes `String greeting = "Hi Mary!"; greeting.toLowerCase() returns "hi mary!".`

(continued)
Some Methods in the Class **String** (3)

**Display 1.4** Some Methods in the Class String

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String toUpperCase()</strong></td>
<td>Returns a string with the same characters as the calling object string, but with all letter characters converted to uppercase.</td>
</tr>
<tr>
<td><strong>String trim()</strong></td>
<td>Returns a string with the same characters as the calling object string, but with leading and trailing white space removed. Whitespace characters are the characters that print as white space on paper, such as the blank (space) character, the tab character, and the new-line character '\n'.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

After program executes `String greeting = "Hi Mary!"; greeting.toUpperCase()` returns "HI MARY!".

**EXAMPLE**

After program executes `String pause = "   Hmm   "; pause.trim()` returns "Hmmm".

(continued)
## Some Methods in the Class `String`

### `charAt(Position)`

Returns the character in the calling object string at the `Position`. Positions are counted 0, 1, 2, etc.

**EXAMPLE**

After program executes `String greeting = "Hello!"; greeting.charAt(0) returns 'H', and greeting.charAt(1) returns 'e'.

### `substring(Start)`

Returns the substring of the calling object string starting from `Start` through to the end of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position `Start` is included in the value returned.

**EXAMPLE**

After program executes `String sample = "AbcdefG"; sample.substring(2) returns "cdefG".`
Some Methods in the Class **String** (5)

**Display 1.4  Some Methods in the Class String**

```java
String substring(Start, End)

Returns the substring of the calling object string starting from position Start through, but not including, position End of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position Start is included in the value returned, but the character at position End is not included.

**EXAMPLE**

After program executes String sample = "AbcdefG";
    sample.substring(2, 5) returns "cde".
```

```java
int indexOf(A_String)

Returns the index (position) of the first occurrence of the string A_String in the calling object string. Positions are counted 0, 1, 2, etc. Returns -1 if A_String is not found.

**EXAMPLE**

After program executes String greeting = "Hi Mary!";
    greeting.indexOf("Mary") returns 3, and
    greeting.indexOf("Sally") returns -1.
```
Escape Sequences

• A backslash (\) immediately preceding a character (i.e., without any space) denotes an escape sequence or an escape character
  – The character following the backslash does not have its usual meaning
  – Although it is formed using two symbols, it is regarded as a single character
Escape Sequences

Display 1.6  Escape Sequences

\" Double quote.
\' Single quote.
\ Backslash.
\n New line. Go to the beginning of the next line.
\r Carriage return. Go to the beginning of the current line.
\t Tab. White space up to the next tab stop.
String Processing

- A `String` object in Java is considered to be immutable, i.e., the characters it contains cannot be changed.
- There is another class in Java called `StringBuffer` that has methods for editing its string objects.
- However, it is possible to change the value of a `String` variable by using an assignment statement:
  ```java
  String name = "Soprano";
  name = "Anthony " + name;
  ```
Naming Constants

• Instead of using "anonymous" numbers in a program, always declare them as named constants, and use their name instead

  
  ```java
  public static final int INCHES_PER_FOOT = 12;
  public static final double RATE = 0.14;
  ```

  – This prevents a value from being changed inadvertently
  – It has the added advantage that when a value must be modified, it need only be changed in one place
  – Note the naming convention for constants: Use all uppercase letters, and designate word boundaries with an underscore character
Comments

• A *line comment* begins with the symbols `//`, and causes the compiler to ignore the remainder of the line
  – This type of comment is used for the code writer or for a programmer who modifies the code
• A *block comment* begins with the symbol pair `/*`, and ends with the symbol pair `*/`
  – The compiler ignores anything in between
  – This type of comment can span several lines
  – This type of comment provides documentation for the users of the program
Source

- Absolute Java, Savitch
  - Copyright © 2010 Pearson Addison-Wesley. All rights reserved.