Basic Computation

Chapter 2
Documentation and Style: Outline

• Meaningful Names
• Comments
• Indentation
• Named Constants
Documentation and Style

• Most programs are modified over time to respond to new requirements.
• Programs which are easy to read and understand are easy to modify.
• Even if it will be used only once, you have to read it in order to debug it.
Meaningful Variable Names

• A variable's name should suggest its use.
• Observe conventions in choosing names for variables.
  ▪ Use only letters and digits.
  ▪ "Punctuate" using uppercase letters at word boundaries (e.g. `taxRate`).
  ▪ Start variables with lowercase letters.
  ▪ Start class names with uppercase letters.
Comments

• The best programs are self-documenting.
  ▪ Clean style
  ▪ Well-chosen names
• Comments are written into a program as needed explain the program.
  ▪ They are useful to the programmer, but they are ignored by the compiler.
Comments

• A comment can begin with //.
• Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.

  double radius; //in centimeters
Comments

• A comment can begin with /* and end with */
• Everything between these symbols is treated as a comment and is ignored by the compiler.

    /**
     This program should only be used on alternate Thursdays, except during leap years, when it should only be used on alternate Tuesdays.
     */
Comments

• A javadoc comment, begins with /*** and ends with */.

• It can be extracted automatically from Java software.

    /*** method change requires the number of coins to be nonnegative */
When to Use Comments

• Begin each program file with an explanatory comment
  ▪ What the program does
  ▪ The name of the author
  ▪ Contact information for the author
  ▪ Date of the last modification.

• Provide only those comments which the expected reader of the program file will need in order to understand it.
Comments Example

- View sample program

```java
class CircleCalculation, listing 2.7
```

Enter the radius of a circle in inches:
2.5
A circle of radius 2.5 inches has an area of 19.6349375 square inches.

Sample Screen Output
Indentation

• Indentation should communicate nesting clearly.
• A good choice is four spaces for each level of indentation.
• Indentation should be consistent.
• Indentation should be used for second and subsequent lines of statements which do not fit on a single line.
Indentation

• Indentation does not change the behavior of the program.
• Proper indentation helps communicate to the human reader the nested structures of the program
Using Named Constants

• To avoid confusion, always name constants (and variables).
  
  \[
  \text{area} = \pi \times \text{radius} \times \text{radius};
  \]
  
is clearer than

  \[
  \text{area} = 3.14159 \times \text{radius} \times \text{radius};
  \]

• Place constants near the beginning of the program.
Named Constants

• Once the value of a constant is set (or changed by an editor), it can be used (or reflected) throughout the program.

    public static final double INTEREST_RATE = 6.65;

• If a literal (such as 6.65) is used instead, every occurrence must be changed, with the risk than another literal with the same value might be changed unintentionally.
Declaring Constants

• Syntax

    public static final
    Variable_Type = Constant;

• Examples

    public static final double
    PI = 3.14159;

    public static final String MOTTO = "The customer is always right.";

• By convention, uppercase letters are used for constants.
Named Constants

• View sample program

```java
class CircleCalculation2, listing 2.8
```

Enter the radius of a circle in inches:
2.5
A circle of radius 2.5 inches has an area of 19.6349375 square inches.
JOptionPane

- View sample program class `JOptionPaneDemo`, listing 2.11
**JOptionPane**

- **JOptionPane** can be used to construct windows that interact with the user.
- The **JOptionPane** class is imported by
  ```java
  import javax.swing.JApplet;
  ```
- The **JOptionPane** class produces windows for obtaining input or displaying output.
JOptionPane

- Use `showInputDialog()` for input.
- Only string values can be input.
- To convert an input value from a string to an integer use the `parseInt()` method from the `Integer` class, use

```
appleCount = Integer.parseInt(appleString);
```
JOptionPane

• Output is displayed using the `showMessageDialog` method.

```java
JOptionPane.showMessageDialog(null, "The total number of fruits = " + totalFruitCount);
```
JOptionPane

• Syntax
  ▪ Input
    ```java
    String_Variable = JOptionPane.showInputDialog
                     (String_Expression);
    ```
  ▪ Output
    ```java
    JOptionPane.showMessageDialog
                 (null, String_Expression);
    ```
• `System.exit(0)` ends the program.
**JOptionPane Caution**

- If the input is not in the correct format, the program will crash.
Inputting Numeric Types

- `JOptionPane.showInputDialog` can be used to input any of the numeric types.
- Figure 2.8 Methods for converting strings to numbers

<table>
<thead>
<tr>
<th>Result Type</th>
<th>Method for Converting</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td><code>Byte.parseByte(String_To_Convert)</code></td>
</tr>
<tr>
<td>short</td>
<td><code>Short.parseShort(String_To_Convert)</code></td>
</tr>
<tr>
<td>int</td>
<td><code>Integer.parseInt(String_To_Convert)</code></td>
</tr>
<tr>
<td>long</td>
<td><code>Long.parseLong(String_To_Convert)</code></td>
</tr>
<tr>
<td>float</td>
<td><code>Float.parseFloat(String_To_Convert)</code></td>
</tr>
<tr>
<td>double</td>
<td><code>Double.parseDouble(String_To_Convert)</code></td>
</tr>
</tbody>
</table>
Multi-Line Output Windows

• To output multiple lines using the method `JOptionPane.showMessageDialog`, insert the new line character '
' into the string used as the second argument.

```java
JOptionPane.showMessageDialog(null,
   "The number of apples\n" +
   "plus the number of oranges\n" +
   "is equal to " + totalFruit);
```
Multi-Line Output Windows

• Figure 2.9 A dialog window containing multiline output
Programming Example

- View sample program
  class ChangeMakerWindow, listing 2.12

Sample Screen Output

Input

Enter a whole number from 1 to 99.
I will output a combination of coins that equals that amount of change.

87

Message

87 cents in coins can be given as:
3 quarters
1 dimes
0 nickels and
2 pennies

OK
Summary

• You have become familiar with Java primitive types (numbers, characters, etc.).
• You have learned about assignment statements and expressions.
• You have learned about strings.
• You have become familiar with classes, methods, and objects.
Summary

• You have learned about simple keyboard input and screen output.
• You have learned about windows-based input and output using the `JOptionPane` class.