Primitive Types, Strings, and Console I/O

Chapter 2
Variables and Values

• *Variables* store data such as numbers and letters.
  – Think of them as places to store data.
  – They are implemented as memory locations.
• The data stored by a variable is called its *value*.
  – The value is stored in the memory location.
• Its value can be changed.
Variables and Values, cont.

```java
public class EggBasket {
    public static void main (String[] args) {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        eggsPerBasket = 6;
        numberOfBaskets = 10;
        totalEggs = numberOfBaskets * eggsPerBasket;
        ...
    }
}
```
Variables and Values, cont.

• variables
  
  numberOfBaskets
  eggsPerBasket
  totalEggs

• assigning values
  
  eggsPerBasket = 6;
  eggsPerBasket = eggsPerBasket - 2;
Naming and Declaring Variables

• Choose names that are helpful such as *count* or *speed*, but not *c* or *s*.

• When you *declare* a variable, you provide its name and type.
  
  ```java
  int numberOfBaskets;
  ```

• A variable’s *type* determines what kinds of values it can hold (*int*, *double*, *char*, etc.).

• A variable must be declared before it is used.
Syntax and Examples

• syntax
  
  \texttt{type variable\_1, variable\_2, ...;}

• examples
  
  \texttt{int styleChoice, numberOfChecks; double balance, interestRate; char jointOrIndividual;}
Types in Java

• A class type is used for a class of objects and has both data and methods.
  - “Career in Computer Science” is a value of class type String

• A primitive type is used for simple, nondecomposable values such as an individual number or individual character.
  - int, double, and char are primitive types.
Naming Conventions

• Class types begin with an uppercase letter (e.g. `String`).
• Primitive types begin with a lowercase letter (e.g. `int`).
• Variables of both class and primitive types begin with a lowercase letters (e.g. `myName`, `myBalance`, `totalEggs`).
  – Multiword names are “punctuated” using uppercase letters (“camel” notation).
Where to Declare Variables

- Declare a variable
  - just before it is used or
  - at the beginning of the section of your program that is enclosed in {}.

```java
public static void main(String[] args) {
  /* declare variables here */

  ...  

  }
```
Java Identifiers

• An *identifier* is a name, such as the name of a variable.

• Identifiers may contain only
  – letters
  – digits (0 through 9)
  – the underscore character (_)
  – and the dollar sign symbol ($) which has a special meaning

but the first character cannot be a digit.
Java Identifiers, cont.

- Identifiers may not contain any spaces, dots (.), asterisks (*), or other characters:
  
  7-11 netscape.com util.* (not allowed)

- Identifiers can be arbitrarily long.
- Since Java is case sensitive, stuff, Stuff, and STUFF are different identifiers.
Keywords or Reserved Words

• Words such as `while` are called *keywords* or *reserved words* and have special, predefined meanings.
• Keywords cannot be used as identifiers.
• See Appendix 1 for a complete list of Java keywords.
• Other keywords: `int, public, class`
Primitive Types

• four integer types (byte, short, int, and long)
  - int is most common
• two floating-point types (float and double)
  - double is more common
• one character type (char)
• one boolean type (boolean)
## Primitive Types, cont.

<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>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>−2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>integer</td>
<td>8 bytes</td>
<td>−9223372036854775808 to 9223372036854775807</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>±1.76769313486231570 × 10^{+308} to ±4.94065645841246544 × 10^{-324}</td>
</tr>
<tr>
<td>char</td>
<td>single character (Unicode)</td>
<td>2 bytes</td>
<td>all Unicode characters</td>
</tr>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>1 bit</td>
<td>not applicable</td>
</tr>
</tbody>
</table>

Display 2.2

Primitive Types
Examples of Primitive Values

- **integer types**
  
  0  -1  365  12000

- **floating-point types**
  
  0.99  -22.8  3.14159  5.0

- **character type**
  
  'a'  'A'  '#'  ' ' (blank)

- **boolean type**
  
  true  false
Assignment Statements

• An assignment statement is used to assign a value to a variable.
  \[\text{answer} = 42;\]

• The “equal sign” is called the \textit{assignment operator}.

• We say, “The variable named \texttt{answer} is assigned a value of 42,” or more simply, “\texttt{answer} is assigned 42.”
Assignment Statements, cont.

• Syntax

\[ \text{variable} = \text{expression}; \]

where \text{expression} can be another variable, a literal or constant (such as a number), or something more complicated which combines variables and literals using \textit{operators} (such as + and -)
Assignment Examples

amount = 3.99;
firstInitial = 'W';
score = numberOfCards + handicap;
eggsPerBasket = eggsPerBasket - 2;
Assignment Evaluation

• The expression on the right-hand side of the assignment operator (=) is evaluated first.
• The result is used to set the value of the variable on the left-hand side of the assignment operator.

    score = numberOfCards + handicap;
    eggsPerBasket = eggsPerBasket - 2;
Specialized Assignment Operators

• Assignment operators can be combined with arithmetic operators (including -, *, /, and %, discussed later).

```java
amount = amount + 5;
```

can be written as

```java
amount += 5;
```
yielding the same result.
Simple Screen Output

System.out.println("The count is " + count);
outputs the string literal "The count is" followed by the current value of the variable count.
Simple Input

• Sometimes the information needed for a computation is obtained from the user at run time.

• Keyboard input requires

  import java.util.*  

  at the beginning of the file.
Simple Input, cont.

- Data can be entered from the keyboard using
  
  ```java
  Scanner keyboard =
  new Scanner(System.in);
  ```

  followed, for example, by
  
  ```java
  eggsPerBasket = keyboard.nextInt();
  ```

  which reads one `int` value from the keyboard
  and assigns it to `eggsPerBasket`.
Simple Input, cont.

- class EggBasket2

```java
import java.util.*;

public class EggBasket2 {
    public static void main(String[] args) {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the number of eggs in each basket:");
        eggsPerBasket = keyboard.nextInt();
        System.out.println("Enter the number of baskets:");
        numberOfBaskets = keyboard.nextInt();
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println("If you have");
        System.out.println("the total number of eggs is \(\) \+ totalEggs);";
        System.out.println("Now we take two eggs out of each basket.");
        eggsPerBasket = eggsPerBasket - 2;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println("You now have");
        System.out.println("the new total number of eggs is " + totalEggs);
    }
}
```

Sample Screen Dialog

Enter the number of eggs in each basket:
6
Enter the number of baskets:
10
If you have 6 eggs per basket and 10 baskets, then the total number of eggs is 60
Now we take two eggs out of each basket.
You now have 4 eggs per basket and 10 baskets.
The new total number of eggs is 40
Number Constants

• Literal expressions such as $2, 3.7, \text{ or } 'y'$ are called *constants*.
• Integer constants can be preceded by a + or - sign, but cannot contain commas.
• Floating-point constants can be written
  – with digits after a decimal point or
  – using *e* notation.
e Notation

- e notation is also called scientific notation or floating-point notation.
- examples
  - 865000000.0 can be written as 8.65e8
  - 0.000483 can be written as 4.83e-4
- The number in front of the e does not need to contain a decimal point.
Assignment Compatibilities

• Java is said to be *strongly typed*.  
  – You can’t, for example, assign a floating point value to a variable declared to store an integer.

• Sometimes conversions between numbers are possible.

```java
doubleVariable = 7;
```

*is possible even if* `doubleVariable` *is of type `double`, for example.*
Assignment Compatibilities, cont.

• A value of one type can be assigned to a variable of any type further to the right

  byte --> short --> int --> long
  --> float --> double

  but not to a variable of any type further to the left.

• You can assign a value of type char to a variable of type int.
Type Casting

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.

- For example,

```java
double distance;
distance = 9.7;
int points;
points = (int)distance;

(illegal without (int))
```
Type Casting, cont.

- The value of \((\text{int})\text{distance}\) is 9, but the value of \(\text{distance}\), both before and after the cast, is 9.7.

- Any nonzero value to the right of the decimal point is \textit{truncated} rather than \textit{rounded}. 
Initializing Variables

• A variable that has been declared, but not yet given a value is said to be \textit{uninitialized}.
• Uninitialized class variables have the value \texttt{null}.
• Uninitialized primitive variables may have a default value.
• It’s good practice not to rely on a default value.
Initializing Variables, cont.

• To protect against an uninitialized variable (and to keep the compiler happy), assign a value at the time the variable is declared.

• Examples:

```c
int count = 0;
char grade = 'A';
```
Imprecision in Floating-Point Numbers

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence $\frac{1.0}{3.0}$ is slightly less than $\frac{1}{3}$.
- $\frac{1.0}{3.0} + \frac{1.0}{3.0} + \frac{1.0}{3.0}$ is slightly less than $1$. 
Arithmetic Operations

- Arithmetic expressions can be formed using the +, -, *, and / operators together with variables or numbers referred to as operands.
  - When both operands are of the same type, the result is of that type.
  - When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.
Arithmetic Operations, cont.

• Example

If \( \text{hoursWorked} \) is an \( \text{int} \) to which the value 40 has been assigned, and \( \text{payRate} \) is a \( \text{double} \) to which 8.25 has been assigned

\[
\text{hoursWorked} \times \text{payRate}
\]

is a \( \text{double} \) with a value of 330.0.
Arithmetic Operations, cont.

• Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  – The result of one step produces one of the operands to be used in the next step.

• example

  balance + (balance * rate)
Arithmetic Operations, cont.

- if at least one of the operands is a floating-point type and the rest are integers, the result will be a floating point type.
- The result is the rightmost type from the following list that occurs in the expression.

byte --> short --> int --> long
--> float --> double
The Division Operator

• The division operator (/) behaves as expected if one of the operands is a floating-point type.

• When both operands are integer types, the result is truncated, not rounded.
  – Hence, 99/100 has a value of 0.
The \texttt{mod} Operator

- The \texttt{mod} (\%) operator is used with operators of integer type to obtain the remainder after integer division.
- 14 divided by 4 is 3 \textit{with a remainder of 2}.
  - Hence, $14 \mod 4$ is equal to 2.
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer.
Parentheses and Precedence

• Parentheses can communicate the order in which arithmetic operations are performed
• examples:
  
  (cost + tax) * discount
  cost + (tax * discount)

• Without parentheses, an expression is evaluated according to the rules of precedence.
Precedence Rules

*Highest Precedence*

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

*Lowest Precedence*

Display 2.4

Precedence Rules
# Sample Expressions

<table>
<thead>
<tr>
<th>Ordinary Mathematical Expression</th>
<th>Java Expression (Preferred Form)</th>
<th>Equivalent Fully Parenthesized Java Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(rate^2 + \delta)</td>
<td>(rate*rate + \delta)</td>
<td>((rate*rate) + \delta)</td>
</tr>
<tr>
<td>(2(salary + bonus))</td>
<td>(2*(salary + bonus))</td>
<td>(2*(salary + bonus))</td>
</tr>
<tr>
<td>(\frac{1}{time + 3mass})</td>
<td>(1/(time + 3*mass))</td>
<td>(1/(time + (3*mass)))</td>
</tr>
<tr>
<td>(\frac{a - 7}{t + 9v})</td>
<td>((a - 7)/(t + 9*v))</td>
<td>((a - 7)/(t + (9*v)))</td>
</tr>
</tbody>
</table>

Display 2.5

Arithmetic Expressions in Java
Increment (and Decrement) Operators

• used to increase (or decrease) the value of a variable by 1
• easy to use
• the increment operator
  \[ \text{count++ or ++count} \]
• the decrement operator
  \[ \text{count-- or --count} \]
Increment (and Decrement) Operators

• equivalent operations

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

```c
count--;  
--count;  
count = count - 1;
```
Increment (and Decrement) Operators in Expressions

- after executing
  
  ```java
  int m = 4;
  int result = 3 * (++m)
  
  result has a value of 15 and m has a value of 5
  ```

- after executing
  
  ```java
  int m = 4;
  int result = 3 * (m++)
  
  result has a value of 12 and m has a value of 5
  ```
The Class String

- We’ve used constants of type String already.
  “Enter a whole number from 1 to 99.”
- A value of type String is a sequence of characters treated as a single item.
Declaring and Printing Strings

• declaring
  
  String greeting;
  greeting = “Hello!”;

  or

  String greeting = “Hello!”;

  or

  String greeting = new String(“Hello!”);

• printing

  System.out.println(greeting);
Concatenation of Strings

• Two strings are *concatenated* using the + operator.
  
  String greeting = "Hello";
  String sentence;
  sentence = greeting + " officer";
  System.out.println(sentence);

• Any number of strings can be concatenated using the + operator.
Concatenating Strings and Integers

String solution;

solution = "The temperature is " + 72;
System.out.println (solution);

The temperature is 72
Classes

• A class is a type used to produce objects.
• An object is an entity that stores data and can take actions defined by methods.
• An object of the String class stores data consisting of a sequence of characters.
• The length() method returns the number of characters in a particular String object.

```java
int howMany = solution.length();
```
Objects, Methods, and Data

• Objects within a class
  – have the same methods
  – have the same kind(s) of data but the data can have different values.

• Primitive types have values, but no methods.
The Method `length()`

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```java
int count = solution.length();
System.out.println(solution.length());
spaces = solution.length() + 3;
```
Positions in a `String`

- positions start with 0, not 1.
  - The `J` in “Java is fun.” is in position 0
Positions in a String, cont.

- A position is referred to an an index.
  - The ‘f’ in “Java is fun.” is at index 8.

The twelve characters in the string "Java is fun." have indices 0 through 11. The index of each character is shown above it.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>a</td>
<td>v</td>
<td>a</td>
<td>i</td>
<td>s</td>
<td></td>
<td>f</td>
<td>u</td>
<td>n</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

Note that the blanks and the period count as characters in the string.

Display 2.8
String Indices
(Not) Changing String Objects

• No methods allow you to change the value of a String object.

• But you can change the value of a String variable.

```java
String pause = " Hmm ";
pause = pause.trim();
pause = pause + " mmm!";
pause = "Ahhh";
```

value of
pause

Hmm
Hmm
Hmmmmmm!
Ahhh
Escape Characters

• How would you print

   "Java" refers to a language.

• The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

   System.out.println("\"Java\" refers to a language.");
Escape Characters

\" 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. Add whitespace up to the next tab stop.

Display 2.10
Escape Characters

• Each escape sequence is a single character even though it is written with two symbols.
Examples

System.out.println("abc\def");
abc\def
System.out.println("new
line");
new
line
char singleQuote = \\\\\nSystem.out.println(singleQuote);
\"
The Unicode Character Set

• Most programming languages use the ASCII character set.
• Java uses the *Unicode* character set which includes the ASCII character set.
• The Unicode character set includes characters from many different alphabets (but you probably won’t use them).
Screen Output

- We’ve seen several examples of screen output already.
- `System.out` is an object that is part of Java.
- `println()` is one of the methods available to the `System.out` object.
Screen Output, cont.

- The concatenation operator (+) is useful when everything does not fit on one line.

  System.out.println("When everything " +
  "does not fit on one line, use the" +
  " concatenation operator (\'/\'+\'/\')");

- Do not break the line except immediately before or after the concatenation operator (+).
Screen Output, cont.

• Alternatively, use `print()`

```
System.out.print("When everything ");
System.out.print("does not fit on ");
System.out.print("one line, use the ");
System.out.print("\"print\" ");
System.out.println("statement");
```

*ending with a `println()`.*
Screen Output, cont.

- **syntax**
  ```java
  System.out.println(output_1 + output_2 + ... + output_n);
  ```

- **example**
  ```java
  System.out.println (1967 + " " + "Oldsmobile" + " " + 442);
  1967 Oldsmobile 442
  ```
Keyboard Input

• Java 5.0 has facilities for handling keyboard input.
• These facilities are provided by the Scanner class in the java.util package.
  – A package is a library of classes.
Using the Scanner Class

• Near the beginning of your program, insert
  import java.util.*

• Create an object of the Scanner class
  Scanner keyboard =
      new Scanner (System.in)

• Read data (an int or a double, for example)
  int n1 = keyboard.nextInt();
  double d1 = keyboard.nextDouble();
Some Scanner Class Methods

• syntax

\[
\text{Int\_Variable} = \text{Object\_Name}\text{.nextInt();}
\]
\[
\text{Double\_Variable} = \text{Object\_Name}\text{.nextDouble();}
\]
\[
\text{String\_Variable} = \text{Object\_Name}\text{.next();}
\]
  \hspace{1em} // reads word
\[
\text{String\_Variable} = \text{Object\_Name}\text{.nextLine();}
\]
  \hspace{1em} // reads rest of line
Some Scanner Class Methods, cont.

- **examples**
  ```java
  int count = keyboard.nextInt();
  double distance = keyboard.nextDouble();
  String word = keyboard.next();
  String wholeLine = keyboard.nextLine();
  ```

- **Remember to prompt the user for input, e.g.**
  ```java
  System.out.print(“Enter an integer: “);
  ```
nextLine() Method Caution

- The `nextLine()` method reads the remainder of the current line, even if it is empty.
The Empty String

• A string can have any number of characters, including zero.
• The string with zero characters is called the empty string.
• The empty string is useful and can be created in many ways including
  
  String s3 = "";
Documentation and Style: Outline

- Meaningful Names
- Self-Documentation and 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 Names for Variables

- 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.
Documentation and 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, cont.

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

/* the simplex method is used to calculate the answer*/
Comments, cont.

• 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

- class CircleCalculation

```java
import java.util.*;

/**
 * Program to determine area of a circle.
 * Author: Jane Q. Programmer.
 * E-mail Address: Jane@SomeMachine.etc.etc.
 * Programming Assignment 2,
 * Last Changed: October 7, 2006.
 */

public class CircleCalculation
{
    public static void main(String[] args)
    {
        double radius; //in inches
        double area; //in square inches
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the radius of a circle in inches:");
        radius = keyboard.nextDouble();
        area = 3.14159 * radius * radius;
        System.out.println("A circle of radius "+radius+" inches");
        System.out.println("has an area of "+area+" square inches.");
    }
}
```

Sample Screen Dialog

```
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.
```
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, cont.

- Indentation does not change the behavior of the program.
- Improper indentation can miscommunicate the behavior of the program.
Named Constants

• To avoid confusion, always name constants (and variables).
  
  \[
  \text{circumference} = \pi \times \text{radius};
  \]
  
  is clearer than
  
  \[
  \text{circumference} = 3.14159 \times 6.023;
  \]

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

• 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 = 6.5;

• If a literal (such as 6.5) 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

- class CircleCalculation2

```java
import java.util.*;
/**
  Program to determine area of a circle.
  Author: Jane Q. Programmer.
  E-mail Address: janeq@somemachine.etc.etc.
  Programming Assignment 2.
  Last Changed: October 7, 2006.
*/
public class CircleCalculation2
{
    public static final double PI = 3.14159;
    public static void main(String[] args)
    {
        double radius; //in inches
        double area; //in square inches
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the radius of a circle in inches:");
        radius = keyboard.nextDouble();
        area = 3.14159 * radius * radius;
        System.out.println("A circle of radius "+radius+" inches has an area of "+area+" square inches.");
    }
}
```

Sample Screen Dialog

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.
(optional) Graphics Supplement: Outline

- Style Rules Applied to a Graphics Applet
- JOptionPane
- Inputting Numeric Types
- Multi-Line Output Windows
Style Rules Applied to a Graphics Applet

- class HappyFace

```java
import java.awt.*;
import java.awt.*;

//
// Applet that displays a happy face.
// Author: Jane Q. Programmer.
// E-mail Address: janep@homemachine.etc.etc.
// Programming Assignment 3.
// Last Changed: October 9, 2006.
//
// public class HappyFace extends Applet
// {
//   public static final int FACE_DIAMETER = 200;
//   public static final int X_FACE = 100;
//   public static final int Y_FACE = 50;
//   public static final int Y_EYE = 10;
//   public static final int EYE_HEIGHT = 20;
//   public static final int X_RIGHT_EYE = 155;
//   public static final int X_LEFT_EYE = 150;
//   public static final int Y_LEFT_EYE = Y_RIGHT_EYE;
//   public static final int MOUTH_WIDTH = 100;
//   public static final int MOUTH_HEIGHT = 50;
//   public static final int Y_MOUTH = 150;
//   public static final int X_MOUTH = 175;
//   public static final int MOUTH_START_ANGLE = 180;
//   public static final int MOUTH_DEGREES_SHOWN = 180;
//   public void paint(Graphics canvas) {
//     // Draw face outline:
//     canvas.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
//     // Draw eyes:
//     canvas.fillOval(X_RIGHT_EYE, Y_RIGHT_EYE, EYE_WIDTH, EYE_HEIGHT);
//     canvas.fillOval(X_LEFT_EYE, Y_LEFT_EYE, EYE_WIDTH, EYE_HEIGHT);
//     // Draw mouth:
//     canvas.drawArc(X_MOUTH, Y_MOUTH, MOUTH_WIDTH, MOUTH_HEIGHT,
//                   MOUTH_START_ANGLE, MOUTH_DEGREES_SHOWN);
//   }
```

Display 2.16
Refresh Using Default Colors and Connectors
Style Rules Applied to a Graphics Applet, cont.

• Named constants make it easier to find values.
• Comments and named constants make changing the code much easier.
• Named constants protect against changing the wrong value.
JOptionPane

• class JOptionPaneDemo

```java
import javax.swing.*;

public class JOptionPaneDemo {
    public static void main(String[] args) {
        String appleString;
        appleString = JOptionPane.showInputDialog("Enter number of apples:");
        int appleCount = Integer.parseInt(appleString);

        String orangeString;
        orangeString = JOptionPane.showInputDialog("Enter number of oranges:");
        int orangeCount = Integer.parseInt(orangeString);

        int totalFruitCount;
        totalFruitCount = appleCount + orangeCount;

        JOptionPane.showMessageDialog(null, "The total number of fruits = " + totalFruitCount);
        System.exit(0);
    }
}
```

Window 1

Window 2

Window 3

Display 2.17
A Program with JOptionPane I/O
JOptionPane, cont.

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

- **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

```java
appleCount = Integer.parseInt(appleString);
```
JOptionPane, cont.

- Output is displayed using the `showMessageDialog` method.
  ```java
  JOptionPane.showMessageDialog(null, "The total number of fruits = " + totalFruitCount);
  ```
JOptionPane, cont.

• syntax
  – input
    
    ```java
    String_Variable =
    JOptionPane.showMessageDialog(String);
    ```
  – output
    
    ```java
    JOptionPane.showMessageDialog(null, String);
    ```
• System.exit(0) ends the program.
JOptionPane Cautions

- If the input is not in the correct format, the program will crash.
- If you omit the last line (System.exit(0)), the program will not end, even when the OK button in the output window is clicked.
- Always label any output.
Inputting Numeric Types

- `JOptionPane.showInputDialog` can be used to input any of the numeric types.
  - Simply convert the input string to the appropriate numeric type.

<table>
<thead>
<tr>
<th>Type Name</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>

To convert a value of type `String` to a value of the type given in the first column, use the method given in the second column. Each of the methods in the second column returns a value of the type given in the first column. The `String_To_Convert` must be a correct string representation of a value of the type given in the first column. For example, to convert to an `int`, the `String_To_Convert` must be a whole number (in the range of the type `int`) that is written in the usual way without any decimal point.

Display 2.18

Methods for Converting Strings to Numbers
Multi-Line Output Windows

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

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

Display 2.19
A Multiline Output Window
Programming Example

- class ChangeMakerWindow

```java
import javax.swing.*;
public class ChangeMakerWindow
{
    public static void main(String[] args)
    {
        String amountString = JOptionPane.showInputDialog(
            "Enter a whole number from 1 to 99, \n" + "I will output a combination of coins\n" + "that equals that amount of change.\n";
        int amount, originalAmount, quarters, dimes, nickels, pennies;
        amount = Integer.parseInt(amountString);
        originalAmount = amount;
        quarters = amount/25;
        amount = amount%25;
        dimes = amount/10;
        amount = amount%10;
        nickels = amount/5;
        amount = amount%5;
        pennies = amount;
        JOptionPane.showMessageDialog(null,
            originalAmount + " cents in coins can be given as: \n" + quarters + " quarters\n" + dimes + " dimes\n" + nickels + " nickels and\n" + pennies + " pennies\n";
        System.exit(0);
    }
}
```

Chapter 2 124
Programming Example, cont.

**Input Window**

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

**Output Window**

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

Display 2.20
Change Program with I/O Windows
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, cont.

• You have learned about simple keyboard input and screen output.

• (optional) You have learned about windows-based input and output using the `JOptionPane` class.