Flow of Control

Chapter 3

Flow of Control

- *Flow of control* is the order in which a program performs actions.
  - Up to this point, the order has been sequential.
- A *branching statement* chooses between two or more possible actions.
- A *loop statement* repeats an action until a stopping condition occurs.

The `if-else` Statement

- A branching statement that chooses between two possible actions.
- syntax

```
if (Boolean_Expression)
  Statement_1;
else
  Statement_2;
```

The `if-else` Statement, cont.

- example

```
if (count < 3)
    total = 0;
else
    total = total + count;
```

The `if-else` Statement, cont.

- `class BankBalance`

```
import java.util.*;

public class BankBalance
{
    public static final double OVERDRAWN_PENALTY = 8.00;
    public static final double INTEREST_RATE = 0.02;//2% annually

    public static void main(String[] args)
    {
        double balance;

        System.out.print("Enter your checking account balance: $");
        Scanner keyboard =  new Scanner(System.in);
        balance = keyboard.nextDouble();
        System.out.println("Original balance $" + balance);

        if (balance >= 0)
            balance = balance + (INTEREST_RATE * balance)/12;
        else
            balance = balance - OVERDRAWN_PENALTY;

        System.out.println("After adjusting for one month");
        System.out.println("of interest and penalties,");
        System.out.println("your new balance is $" + balance);
    }
}
```

Sample Screen Dialog 1

```
Enter your checking account balance: $505.67
Original balance $505.67
After adjusting for one month
of interest and penalties,
your new balance is $506.51278
```

Sample Screen Dialog 2

```
Enter your checking account balance: $-15.53
Original balance $-15.53
After adjusting for one month
of interest and penalties,
your new balance is $-23.53
```

Display 3.1
A Program Using if-else

Compound Statements

- To include multiple statements in a branch, enclose the statements in braces.

```
if (count < 3)
    {
       total = 0;
       count = 0;
    }
```

Omitting the `else` Part

- If the `else` part is omitted and the expression after the `if` is false, no action occurs.
- syntax

```
if (Boolean_Expression)
    Statement
```

- example

```
if (weight > ideal)
    caloriesPerDay -= 500;
```

Introduction to Boolean Expressions

- The value of a *boolean expression* is either `true` or `false`.
- examples

```
time < limit
balance <= 0
```

Java Comparison Operators

Math Notation	Name	Java Notation	Java Examples
=	Equal to	==	`balance == 0` `answer == 'y'`
≠	Not equal to	!=	`income != tax` `answer != 'y'`
>	Greater than	>	`expenses > income`
≥	Greater than or equal to	>=	`points >= 60`
<	Less than	<	`pressure < max`
≤	Less than or equal to	<=	`expenses <= income`

Display 3.2

Java Comparison Operators

Compound Boolean Expressions

- Boolean expressions can be combined using the “and” (`&&`) operator.
- example

```
if ((score > 0) && (score <= 100))
...
```

- not allowed

```
if (0 < score <= 100)
...
```

Compound Boolean Expressions, cont.

- syntax

  ```
  (Sub_Expression_1) && (Sub_Expression_2)
  ```

- Parentheses often are used to enhance readability.
- The larger expression is true only when both of the smaller expressions are true.

Compound Boolean Expressions, cont.

- Boolean expressions can be combined using the “or” (`||`) operator.
- example

```
if ((quantity > 5) || (cost < 10))
...
```

- syntax

```
(Sub_Expression_1) || (Sub_Expression_2)
```

Compound Boolean Expressions, cont.

- The larger expression is true
  - when either of the smaller expressions is true
  - when both of the smaller expressions are true.
- The Java version of “or” is the *inclusive or* which allows either or both to be true.
- The *exclusive or* allows one or the other, but not both to be true.

Using ==

- `==` is appropriate for determining if two integers or characters have the same value.

  ```
  if (a == 3)
  ```

  where `a` is an integer type
- `==` is **not** appropriate for determining if two floating points values are equal. Use `<` and some appropriate tolerance instead.

  ```
  if (abs(b - c) < epsilon)
  ```

  where `b`, `c`, and `epsilon` are floating point types

Using ==, cont.

- `==` is not appropriate for determining if two objects have the same value.
  - `if (s1 == s2)`, where `s1` and `s2` refer to strings, determines only if s1 and s2 refer the a common memory location.
  - If `s1` and `s2` refer to strings with identical sequences of characters, but stored in different memory locations, `(s1 == s2)` is false.

Using ==, cont.

- To test the equality of objects of class String, use method `equals`.

```
s1.equals(s2)
```

or

```
s2.equals(s1)
```

- To test for equality ignoring case, use method `equalsIgnoreCase`.

```
("Hello".equalsIgnoreCase("hello"))
```

`equals` and `equalsIgnoreCase`

- syntax

  `String``.equals(`*`Other_String`*`)`

  `String``.equalsIgnoreCase(`*`Other_String`*`)`

Testing Strings for Equality

- `class StringEqualityDemo`

```
import java.util.*;

public class StringEqualityDemo
{
    public static void main(String[] args)
    {
        String s1, s2;

        System.out.println("Enter two lines of text:");

        Scanner keyboard = new Scanner(System.in);
        s1 = keyboard.nextLine();
        s2 = keyboard.nextLine();

        if (s1.equals(s2))
            System.out.println("The two lines are equal.");
        else
            System.out.println("The two lines are not equal.");

        if (s2.equals(s1))
            System.out.println("The two lines are equal.");
        else
            System.out.println("The two lines are not equal.");

        if (s1.equalsIgnoreCase(s2))
            System.out.println(
                        "But the lines are equal, ignoring case.");
        else
            System.out.println(
                        "Lines are not equal, even ignoring case.");
    }
}
```

These two invocations of the method `equals` *are equivalent.*

Sample Screen Dialog

```
Enter two lines of text:
Java is not coffee.
Java is NOT COFFEE.
The two lines are not equal.
The two lines are not equal.
But the lines are equal, ignoring case.
```

Display 3.3

Testing Strings for Equality

Lexicographic Order

- Lexicographic order is similar to alphabetical order, but is it based on the order of the characters in the ASCII (and Unicode) character set.
  - All the digits come before all the letters.
  - All the uppercase letters come before all the lower case letters.

Lexicographic Order, cont.

- Strings consisting of alphabetical characters can be compared using method `compareTo` and method `toUpperCase` or method `toLowerCase`.

```
String s1 = “Hello”;
String lowerS1 = s1.toLowerCase();
String s2 = “hello”;
if (s1.compareTo(s2)) == 0
    System.out.println(“Equal!”);
```

Nested Statements

- An `if-else` statement can contain any sort of statement within it.
- In particular, it can contain another `if-else` statement.
  - An `if-else` may be nested within the “if” part.
  - An `if-else` may be nested within the “else” part.
  - An `if-else` may be nested within both parts.

Nested Statements, cont.

- syntax

```
if (Boolean_Expression_1)
    if (Boolean_Expression_2)
       Statement_1;
    else
       Statement_2;
else
   if (Boolean_Expression_3)
       Statement_3;
    else
       Statement_4;
```

Nested Statements, cont.

- Each `else` is paired with the nearest unmatched `if`.
- **If used properly**, indentation communicates which `if` goes with which `else`.
- Braces can be used like parentheses to group statements.

Compound Statements

- When a list of statements is enclosed in braces (`{}`), they form a single *compound statement.*
- syntax

```
{
    Statement_1;
    Statement_2;
   …
}
```

Compound Statements, cont.

- A compound statement can be used wherever a statement can be used.
- example

```
if (total > 10)
{
   sum = sum + total;
   total = 0;
}
```

Multibranch `if-else` Statements

- syntax

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
else if (Boolean_Expression_3)
    Statement_3
else if …
else
    Default_Statement
```

Multibranch `if-else` Statements, cont.

- `class Grader`

```
import java.util.*;

public class Grader
{
    public static void main(String[] args)
    {
        int score;
        char grade;

        System.out.println("Enter your score: ");
        Scanner keyboard = new Scanner(System.in);
        score = keyboard.nextInt();

        if (score >= 90)
            grade = 'A';
        else if (score >= 80)
            grade = 'B';
        else if (score >= 70)
            grade = 'C';
        else if (score >= 60)
            grade = 'D';
        else
            grade = 'F';

        System.out.println("Score = " + score);
        System.out.println("Grade = " + grade);
    }
}
```

Sample Screen Dialog

```
Enter your score:
85
Score = 85
Grade = B
```

Display 3.4
Multibranch if-else Statement

Multibranch `if-else` Statements, cont.

- equivalent code

```
if (score >= 90)
    grade = 'A';
else if ((score >= 80) && (score < 90))
    grade = 'B';
else if ((score >= 70) && (score < 80))
    grade = 'C';
else if ((score >= 60) && (score < 70))
    grade = 'D';
else
    grade = 'F';
```

The `switch` Statement

- The `switch` statement is a mutltiway branch that makes a decision based on an *integral* (integer or character) expression.
- The `switch` statement begins with the keyword `switch` followed by an integral expression in parentheses called the *controlling expression.*

The `switch` Statement, cont.

- A list of cases follows, enclosed in braces.
- Each case consists of the keyword `case` followed by
  - a constant called the *case label*
  - a colon
  - a list of statements.
- The list of cases is searched in order for a case label matching the controlling expression.

The `switch` Statement, cont.

- The action associated with a matching case label is executed.
- If no match is found, the case labeled `default` is executed.
  - The `default` case is optional, but recommended, even if it simply prints a message.
- Repeated case labels are not allowed.

The `switch` Statement, cont.

- `class MultipleBirths`

```
import java.util.*;

public class MultipleBirths
{
    public static void main(String[] args)
    {
        int numberOfBabies;
        System.out.print("Enter number of babies: ");
        Scanner keyboard = new Scanner(System.in);
        numberOfBabies = keyboard.nextInt();

        switch (numberOfBabies)
        {
            case 1:
                System.out.println("Congratulations.");
                break;
            case 2:
                System.out.println("Wow. Twins.");
                break;
            case 3:
                System.out.println("Wow. Triplets.");
                break;
            case 4:
            case 5:
                System.out.println("Unbelieveable.");
                System.out.println(numberOfBabies + " babies");
                break;
            default:
                System.out.println("I don't believe you.");
                break;
        }
    }
}
```

controlling expression

case label

break statement

Sample Screen Dialog 1

```
Enter number of babies: 1
Congratulations.
```

Sample Screen Dialog 2

```
Enter number of babies: 3
Wow. Triplets.
```

Sample Screen Dialog 3

```
Enter number of babies: 4
Unbelievable.
4 babies
```

Sample Screen Dialog 4

```
Enter number of babies: 6
I don't believe you.
```

Display 3.5
A switch Statement

The `switch` Statement, cont.

- The action for each case typically ends with the word `break`.
- The optional `break` statement prevents the consideration of other cases.
- The controlling expression can be anything that evaluates to an integral type.

The Conditional Operator

```
if (n1 > n2)
    max = n1;
 else
    max = n2;
```

can be written as

```
max = (n1 > n2) ? n1 : n2;
```

- The `?` and `:` together are call the *conditional operator* or *ternary operator.*

The Conditional Operator, cont.

- The conditional operator is useful with print and println statements.

```
System.out.print(“You worked “ + hours +
   ((hours > 1) ? “hours” : “hour”));
```

the `while` Statement

- also called a `while` loop
- A `while` statement repeats until a controlling boolean expression becomes false.
  - If the controlling boolean expression is false initially, the `while` loop is not executed.
- The loop body typically contains a statement that ultimately causes the controlling boolean expression to become false.

the `while` Statement, cont.

- `class WhileDemo`

```
import java.util.*;

public class WhileDemo
{
    public static void main(String[] args)
    {
        int count, number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in);
        number = keyboard.nextInt();

        count = 1;
        while (count <= number)
        {
            System.out.print(count + ", ");
            count++;
        }

        System.out.println();
        System.out.println("Buckle my shoe.");
    }
}
```

Sample Screen Dialog 1

```
Enter a number:
2
1, 2,
Buckle my shoe.
```

Sample Screen Dialog 2

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

Sample Screen Dialog 3

```
Enter a number:
0

Buckle my shoe.
```

The loop body is iterated zero times.

Display 3.6
A while Loop

the `while` Statement, cont.

- syntax

```
while (Boolean_Expression)
   Body_Statement;
or
while (Boolean_Expression)
{
   First_Statement;
   Second_Statement;
   …
}
```

The `do-while` Statement

- also called a `do-while` loop
- similar to a `while` statement, except that the loop body is executed at least once
- syntax

```
do
    Body_Statement
while (Boolean_Expression);
```

  – don't forget the semicolon at the end of the while line!

The do-while Statement, cont.

- `class DoWhileDemo`

```
import java.util.*;
public class DoWhileDemo
{
    public static void main(String[] args)
    {
        int count, number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in);
        number = keyboard.nextInt();

        count = 1;
        do
        {
            System.out.print(count + ", ");
            count++;
        }while (count <= number);

        System.out.println();
        System.out.println("Buckle my shoe.");
    }
}
```

Sample Screen Dialog 1

```
Enter a number:
2
1, 2,
Buckle my shoe.
```

Sample Screen Dialog 2

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

Sample Screen Dialog 3

```
Enter a number:
0
1,
Buckle my shoe.
```

The loop body is always executed at least one time.

Display 3.8
A do–while Loop

The `do-while` Statement, cont.

- First, the loop body is executed.
- Then the boolean expression is checked.
  - As long as it is true, the loop is executed again.
  - If it is false, the loop is exited.
- equivalent `while` statement

```
Statement(s)_S1
while (Boolean_Condition)
    Statement(s)_S1
```

Infinite Loops

- A loop which repeats without ever ending is called an *infinite loop.*
- If the controlling boolean expression never becomes false, a `while` loop or a `do-while` loop will repeat without ending.

The `for` Statement

- A `for` statement executes the body of a loop a fixed number of times.
- example

```
for (count = 1; count < 5; count++)
     System.out.println(count);
System.out.println(“Done”);
```

The `for` Statement, cont.

- syntax

```
for (Initialization, Condition, Update)
    Body_Statement
```

  - *`Body_Statement`* can be either a simple statement or a compound statement in `{}`.
- corresponding `while` statement

```
Initialization
while (Condition)
    Body_Statement_Including_Update
```

Multiple Initialization, etc.

- example

```
for (n = 1, p = 1; n < 10; n++)
    p = p * n;
```

- Only one boolean expression is allowed, but it can consist of `&&`s, `||`s, and `!`s.
- Multiple update actions are allowed, too.

```
for (n = 1, p = 1; n < 10; n++, p = p * n)
```

- rarely used

The Empty `for` Statement

- What is printed by

```
int product = 1, number;
for (number = 1; number <= 10;
  number++);
    product = product * number;
System.out.println(product);
```

?

- The last semicolon in

```
for (number = 1; number <= 10; number++);
```

produces an empty `for` statement.

The Empty `while` Statement

```
int product = 1, number = 1;
while (number <= 10);
{
    product = product * number;
    number++
}
System.out.println(product);
```

- The last semicolon in

  `while (number <= 10);`

  produces an empty `while` loop body.

Choosing a Loop Statement

- If you know how many times the loop will be iterated, use a `for` loop.
- If you don’t know how many times the loop will be iterated, but
  - it could be zero, use a `while` loop
  - it will be at least once, use a `do-while` loop.
- Generally, a `while` loop is a safe choice.

The `break` Statement in Loops

- A `break` statement can be used to end a loop immediately.
- The `break` statement ends only the **innermost** loop or switch statement that contains the `break` statement.
- `break` statements make loops more difficult to understand.
- Use `break` statements sparingly (if ever).

The `break` Statement in Loops, cont.

- `class BreakDemo`

```
import java.util.*;

public class BreakDemo
{
    public static void main(String[] args)
    {
        int itemNumber;
        double amount, total;
        Scanner keyboard = new Scanner(System.in);

        System.out.println("You may buy ten items, but");
        System.out.println("the total price must not exceed $100.");

        total = 0;
        for (itemNumber = 1; itemNumber <= 10; itemNumber++)
        {
            System.out.print("Enter cost of item #"
                                        + itemNumber + ": $");
            amount = keyboard.nextDouble();
            total = total + amount;
            if (total >= 100)
            {
                System.out.println("You spent all your money.");
                break;
            }
            System.out.println("Your total so far is $" + total);
            System.out.println("You may purchase up to "
                        + (10   itemNumber) + " more items.");
        }
        System.out.println("You spent $" + total);
    }
}
```

Sample Screen Dialog

```
You may buy ten items, but
the total price must not exceed $100.
Enter cost of item #1: $90.93
Your total so far is $90.93
You may purchase up to 9 more items.
Enter cost of item #2: $10.50
You spent all your money.
You spent $101.43
```

Display 3.13
Ending a Loop with a break Statement

The `exit` Method

- Sometimes a situation arises that makes continuing the program pointless.
- A program can be terminated normally by

```
System.exit(0).
```

- example

```
if (numberOfWinners == 0)
{
   System.out.println(“cannot divide by 0”);
   System.exit(0);
}
```

Ending a Loop, cont.

- For large input lists, a *sentinel value* can be used to signal the end of the list.
  - The sentinel value must be different from all the other possible inputs.
  - A negative number following a long list of nonnegative exam scores could be suitable.

```
90
0
10
-1
```

Ending a Loop, cont.

- example - reading a list of scores followed by a sentinel value

```
int next = keyboard.nextInt();
while (next  >= 0)
{
   Process_The_Score
   next = keyboard.nextInt();
}
```

Ending a Loop, cont.

- `class ExamAverager`

```
import java.util.*;
/**
 Determines the average of a list of (nonnegative) exam scores.
 Repeats for more exams until the user says she/he is finished.
*/
public class ExamAverager
{
    public static void main(String[] args)
    {
        System.out.println("This program computes the average of");
        System.out.println("a list of (nonnegative) exam scores.");
        double sum;
        int numberOfStudents;
        double next;
        String answer;
        Scanner keyboard = new Scanner(System.in);
        do
        {
            System.out.println();
            System.out.println("Enter all the scores to be averaged.");
            System.out.println("Enter a negative number after");
            System.out.println("you have entered all the scores.");
            sum = 0;
            numberOfStudents = 0;
            next = keyboard.nextDouble();
            while (next >= 0)
            {
                sum = sum + next;
                numberOfStudents++;
                next = keyboard.nextDouble();
            }
            if (numberOfStudents > 0)
                System.out.println("The average is "
                                   + (sum/numberOfStudents));
            else
                System.out.println("No scores to average.");
            System.out.println("Want to average another exam?");
            System.out.println("Enter yes or no.");
            answer = keyboard.next();
        }while (answer.equalsIgnoreCase("yes"));
    }
}
```

Sample Screen Dialog

```
This program computes the average of
a list of (nonnegative) exam scores.

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
100
90
100
90
-1
The average is 95.0
Want to average another exam?
Enter yes or no.
yes

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
90
70
80
-1
The average is 80.0
Want to average another exam?
Enter yes or no.
no
```

Display 3.14
Nested Loops

Nested Loops

- The body of a loop can contain any kind of statements, including another loop.
- In the previous example
  - the average score was computed using a `while` loop.
  - This `while` loop was placed inside a `do-while` loop so the process could be repeated for other sets of exam scores.

Loop Bugs

- common loop bugs
  - unintended infinite loops
  - off-by-one errors
  - testing equality of floating-point numbers
- subtle infinite loops
  - The loop may terminate for some input values, but not for others.
  - For example, you can’t get out of debt when the monthly penalty exceeds the monthly payment.

The Type `boolean`

- Boolean Expressions and Variables
- Truth Tables and Precedence Rules
- Input and Output of Boolean Values

The Type `boolean`, cont.

- The type `boolean` is a primitive type with only two values: `true` and `false`.
- Boolean variables can make programs more readable.

```
if (systemsAreOK)
```

instead of

```
if((temperature <= 100) && (thrust >= 12000)
  && (cabinPressure > 30) && …)
```

Boolean Expressions and Variables

- Variables, constants, and expressions of type `boolean` all evaluate to either `true` or `false`.
- A boolean variable can be given the value of a boolean expression by using an assignment operator.

```
boolean isPositive = (number > 0);
...
if (isPositive) ...
```

Naming Boolean Variables

- Choose names such as `isPositive` or `systemsAreOk`.
- Avoid names such as `numberSign` or `systemStatus`.

Short-circuit Evaluation

- Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
  - If the first operand associated with an `||` is `true`, the expression is `true`.
  - If the first operand associated with an `&&` is `false`, the expression is `false`.
- This is called *short-circuit* or *lazy* evaluation.

Short-circuit Evaluation, cont.

- Short-circuit evaluation is not only efficient, sometimes it is essential!
- A run-time error can result, for example, from an attempt to divide by zero.

```
if ((number != 0) && (sum/number > 5))
```

- *Complete evaluation* can be achieved by substituting `&` for `&&` or `|` for `||`.

Input and Output of Boolean Values

- example

```
boolean boo = false;
System.out.println(boo);
System.out.print(“Enter a boolean value: “);
Scanner keyboard = new Scanner (System.in);
boo = keyboard.nextBoolean();
System.out.println(boo);
```

Input and Output of Boolean Values, cont.

- dialog

```
false
Enter a boolean value: true
true
```

Using a Boolean Variable to End a Loop

- example

```
boolean numbersLeftToRead = true
while (numbersLeftToRead)
{
   next = keyboard.nextInt()
   if (next < 0)
         numbersLeftToRead = false;
   else
         Process_Next_Number
}
```

(optional) Graphics Supplement: Outline

- Specifying a Drawing Color
- The `drawString` Method
- A `JOptionPane` Yes/No Window

Specifying a Drawing Color

- When drawing a shape inside an applet's `paint` method, think of the drawing being done with a pen that can change colors.
- The method `setColor` changes the color of the "pen."

```
canvas.setColor(Color.YELLOW);
```

- Drawings done later appear on top of drawings done earlier.

Specifying a Drawing Color, cont.

```
Color.BLACK
Color.BLUE
Color.CYAN
Color.DARK_GRAY
Color.GRAY
Color.GREEN
Color.LIGHT_GRAY
Color.MAGENTA
Color.ORANGE
Color.PINK
Color.RED
Color.WHITE
Color.YELLOW
```

Display 3.19

Predefined Colors

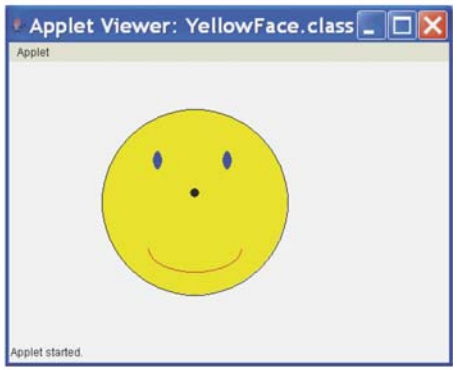
Specifying a Drawing Color, cont.

```
import javax.swing.*;
import java.awt.*;

public class YellowFace extends JApplet
{
    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 EYE_WIDTH = 10;
    public static final int EYE_HEIGHT = 20;
    public static final int X_RIGHT_EYE = 155;
    public static final int Y_RIGHT_EYE = 95;
    public static final int X_LEFT_EYE = 230;
    public static final int Y_LEFT_EYE = Y_RIGHT_EYE;

    public static final int NOSE_DIAMETER = 10;
    public static final int X_NOSE = 195;//Center of nose will be at 200
    public static final int Y_NOSE = 135;

    public static final int MOUTH_WIDTH = 100;
    public static final int MOUTH_HEIGHT = 50;
    public static final int X_MOUTH = 150;
    public static final int Y_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 circle:
        canvas.setColor(Color.YELLOW);
        canvas.fillOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
        canvas.setColor(Color.BLACK);
        canvas.drawOval(X_FACE, Y_FACE, FACE_DIAMETER, FACE_DIAMETER);
        //Draw eyes:
        canvas.setColor(Color.BLUE);
        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 nose:
        canvas.setColor(Color.BLACK);
        canvas.fillOval(X_NOSE, Y_NOSE, NOSE_DIAMETER, NOSE_DIAMETER);
        //Draw mouth:
        canvas.setColor(Color.RED);
        canvas.drawArc(X_MOUTH, Y_MOUTH, MOUTH_WIDTH, MOUTH_HEIGHT,
                       MOUTH_START_ANGLE, MOUTH_DEGREES_SHOWN);
    }
}
```

The filled yellow circle is drawn first so that the other drawings will be on top of the yellow.

Display 3.18
Adding Color

Resulting Applet



The `drawString` Method

- similar to other drawing methods, but used to "draw" text

```
canvas.drawString("Hello",10,20);
```

- syntax

```
Graphics_Object.drawString(String, X, Y);
```

A `JOptionPane` Yes/No Window

- used to present the user with a yes/no question
- The window contains
  - the question text
  - two buttons labeled `Yes` and `No`.

A JOptionPane Yes/No Window, cont.

- example

```
int answer =
  JOptionPane.showConfirmDialog(null,
  “End program?”, “Want to end?”,
  JOptionPane.YES_NO_OPTION);
if (answer == JOptionPane.YES_OPTION)
   System.exit(0);
else
   System.out.println(“once more”);
```

A `JOptionPane` Yes/No Window, cont.

Display 3.21

An Applet that Uses Looping and Branching

A `JOptionPane` Yes/No Window, cont.

- `JOptionPane.showConfirmDialog` returns an `int` value named either `YES_OPTION` or `NO_OPTION`, but you do not need to think of them as `int`s.
- The second argument (`"End program?"` in our example) appears in the window.
- The third argument (`"Want to end?"` in our example) is displayed as the title of the window.

A `JOptionPane` Yes/No Window, cont.

- The last argument (`JOptionPane.YES_NO_OPTION` in our example) requests a window with `yes` and `no` buttons.
- The first argument (`null` in our example) affects the placement of the window on the screen.
  - Simply use `null` for now.