

# Announcements

- Project 2 has been posted
  - Due Feb 1st at 10:00pm
  - Work ALONE!
- Help hours
  - Monday – Thursday, 7-10pm
  - LWSN B146
- Quiz solutions will be on newsgroup
- Sign up for newsgroup!
- We expect you to be reading book. These slides should be a review

# Flow of Control

## Chapter 3

# Outline

- Branching Statements
- Java Loop Statements
- Programming with Loops
- The Type `boolean`
- (optional) Graphics Supplement

# Compound Statements

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

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

# 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 to 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.

# 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

# The `switch` Statement

- The `switch` statement is a multiway 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("Unbelievable.");
                System.out.println(numberOfBabies + " babies");
                break;
            default:
                System.out.println("I don't believe you.");
                break;
        }
    }
}
```

*controlling expression* (points to `numberOfBabies`)

*case label* (points to `case 1:`)

*break statement* (points to `break;`)

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 `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");
```

# 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);
}
```



# Programming with Loops: Outline

- The Loop Body
- Initializing Statements
- Ending a Loop
- Loop Bugs
- Tracing Variables

# The Loop Body

- To design the loop body, write out the actions the code must accomplish.
- Then look for a repeated pattern.
  - The pattern need not start with the first action.
  - The repeated pattern will form the body of the loop.
  - Some actions may need to be done after the pattern stops repeating.

# Initializing Statements

- Some variables need to have a value before the loop begins.
  - Sometimes this is determined by what is supposed to happen after one loop iteration.
  - Often variables have an initial value of zero or one, but not always.
- Other variables get values only while the loop is iterating.

# Ending a Loop

- If the number of iterations is known before the loop starts, the loop is called a *count-controlled loop*.
  - use a `for` loop.
- Asking the user before each iteration if it is time to end the loop is called the *ask-before-iterating technique*.
  - appropriate for a small number of iterations
  - Use a `while` loop or a `do-while` loop.

# 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.



# Declaring Variables Outside Loop Bodies

- The declaration of variables **inside** a loop body is repeated with each execution of the loop body.
  - This can be inefficient, depending on the compiler.
- If the declaration of variables can be moved outside the loop body, generally it is appropriate to do so.

# 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.

# Subtle Infinite Loops

- Verify that the monthly payment exceeds the penalty, for example, before entering a loop to determine the number of payments needed to get out of debt.

```
if (payment <= penalty)
    System.out.println("payment is too
        small");
else
{
    ...
}
```

# Off-by-One Errors

- The loop body is repeated one too many times or one too few times.
- examples
  - $<$  is used when  $\leq$  should be used or  $\leq$  is used when  $<$  should be used
  - using the index of the last character of a string instead of the length of the string (or vice versa)
- easy to overlook

# Testing Equality of Floating-point Numbers

- `==` works satisfactorily for integers and characters.
- `==` is not reliable for floating-point numbers (which are approximate quantities).
  - Use `<=` or `>=` rather than `==` or `!=`.

# Tracing Variables

- *Tracing variables* means watching the variables change while the program is running.
  - Simply insert temporary output statements in your program to print of the values of variables of interest
  - or, learn to use the debugging facility that may be provided by your system.

# Tracing Variables, cont.

```
int time;  
for (time = 1; time <= 4; time++)  
    System.out.println("One more time.");
```

```
int result = 1;  
int count;  
for (count = 1; count <= 5; count++)  
    result = 2*result;
```

0  
0  
1  
0  
1  
2

# 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`.

# Truth Tables

## *&& (and)*

Value of <i>A</i>	Value of <i>B</i>	Resulting Value of <i>A &amp;&amp; B</i>
true	true	true
true	false	false
false	true	false
false	false	false

## *|| (or)*

Value of <i>A</i>	Value of <i>B</i>	Resulting Value of <i>A    B</i>
true	true	true
true	false	true
false	true	true
false	false	false

## *! (not)*

Value of <i>A</i>	Resulting Value of <i>!(A)</i>
true	false
false	true

Display 3.15  
Truth Tables for Boolean Operators

# Precedence Rules

- Parentheses should be used to indicate the order of operations.
- When parentheses are omitted, the order of operation is determined by *precedence rules*.

# Precedence Rules, cont.

- Operations with *higher precedence* are performed before operations with *lower precedence*.
- Operations with *equal precedence* are done left-to-right (except for unary operations which are done right-to-left).

# Precedence Rules, cont.

## *Highest Precedence*

First: the unary operators `+`, `-`, `++`, `--`, and `!`

Second: the binary arithmetic operators `*`, `/`, `%`

Third: the binary arithmetic operators `+`, `-`

Fourth: the boolean operators `<`, `>`, `<=`, `>=`

Fifth: the boolean operators `==`, `!=`

Sixth: the boolean operator `&`

Seventh: the boolean operator `|`

Eighth: the boolean operator `&&`

Ninth: the boolean operator `||`

## *Lowest Precedence*

Display 3.16

Precedence Rules

# Precedence Rules, cont.

- In what order are the operations performed?

```
score < min/2 - 10 || score > 90
```

```
score < (min/2) - 10 || score > 90
```

```
score < ((min/2) - 10) || score > 90
```

```
(score < ((min/2) - 10)) || score > 90
```

```
(score < ((min/2) - 10)) || (score > 90)
```



# 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
}
```

# Using a Boolean Variable to End a Loop, cont

- `class BooleanDemo`

```
import java.util.*;

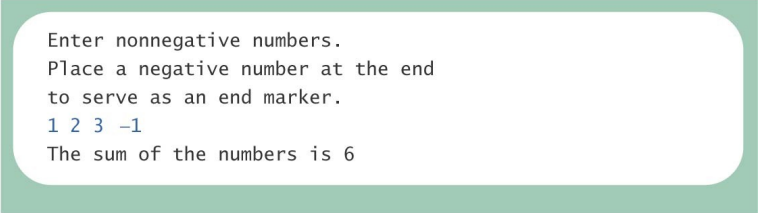
/**
 * Illustrates the use of a boolean variable to control loop ending.
 */
public class BooleanDemo
{
    public static void main(String[] args)
    {
        System.out.println("Enter nonnegative numbers.");
        System.out.println("Place a negative number at the end");
        System.out.println("to serve as an end marker.");

        int next, sum = 0;
        boolean numbersLeft = true;
        Scanner keyboard = new Scanner(System.in);
        while (numbersLeft)
        {
```

```
            next = keyboard.nextInt();
            if (next < 0)
                numbersLeft = false;
            else
                sum = sum + next;
        }

        System.out.println("The sum of the numbers is " + sum);
    }
}
```

Sample Screen Dialog

A screenshot of a Java application window titled "Sample Screen Dialog". The window has a light green border and a white background. It contains the following text: "Enter nonnegative numbers.", "Place a negative number at the end", "to serve as an end marker.", "1 2 3 -1", and "The sum of the numbers is 6".

```
Enter nonnegative numbers.
Place a negative number at the end
to serve as an end marker.
1 2 3 -1
The sum of the numbers is 6
```

Display 3.17

Use of a Boolean Variable to End a Loop

# (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;

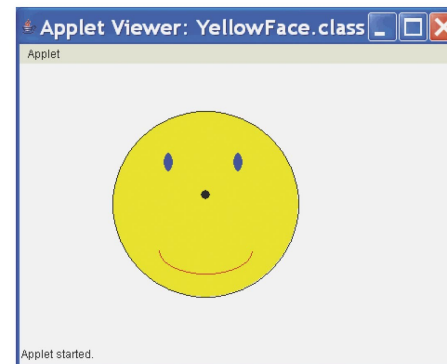
    public static final int NOSE_DIAMETER = 10;
    public static final int X_NOSE = 195; // Center of nose will be at 200
    public static final int Y_NOSE = 135;

    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.

Resulting Applet



Display 3.18  
Adding Color

# Programming Example

- class MultipleFaces

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

public class MultipleFaces extends JApplet
{
    public static final int FACE_DIAMETER = 50;
    public static final int X_FACE0 = 10;
    public static final int Y_FACE0 = 5;
    public static final int EYE_WIDTH = 5;
    public static final int EYE_HEIGHT = 10;
    public static final int X_RIGHT_EYE0 = 20;
    public static final int Y_RIGHT_EYE0 = 15;
    public static final int X_LEFT_EYE0 = 45;
    public static final int Y_LEFT_EYE0 = Y_RIGHT_EYE0;
    public static final int NOSE_DIAMETER = 5;
    public static final int X_NOSE0 = 32;
    public static final int Y_NOSE0 = 25;
    public static final int MOUTH_WIDTH = 30;
    public static final int MOUTH_HEIGHT0 = 0;
    public static final int X_MOUTH0 = 20;
    public static final int Y_MOUTH0 = 35;
    public static final int MOUTH_START_ANGLE = 180;
    public static final int MOUTH_DEGREES_SHOWN = 180;

    public void paint(Graphics canvas)
    {
```

```
        int i;
        for (i = 0; i < 5; i++)
        { //Draw one face:
            //Draw face circle:
            if (i%2 == 0) //if i is even
            { //Make face yellow
                canvas.setColor(Color.YELLOW);
                canvas.fillOval(X_FACE0 + 50*i, Y_FACE0 + 30*i,
                               FACE_DIAMETER, FACE_DIAMETER);
            }
            canvas.setColor(Color.BLACK);
            canvas.drawOval(X_FACE0 + 50*i, Y_FACE0 + 30*i,
                           FACE_DIAMETER, FACE_DIAMETER);

            //Draw eyes:
            canvas.setColor(Color.BLUE);
            canvas.fillOval(X_RIGHT_EYE0 + 50*i, Y_RIGHT_EYE0 + 30*i,
                           EYE_WIDTH, EYE_HEIGHT);
            canvas.fillOval(X_LEFT_EYE0 + 50*i, Y_LEFT_EYE0 + 30*i,
                           EYE_WIDTH, EYE_HEIGHT);

            //Draw nose:
            canvas.setColor(Color.BLACK);
            canvas.fillOval(X_NOSE0 + 50*i, Y_NOSE0 + 30*i,
                           NOSE_DIAMETER, NOSE_DIAMETER);

            //Draw mouth:
            canvas.setColor(Color.RED);
            canvas.drawArc(X_MOUTH0 + 50*i, Y_MOUTH0 + 30*i,
                          MOUTH_WIDTH, MOUTH_HEIGHT0 + 3*i,
                          MOUTH_START_ANGLE, MOUTH_DEGREES_SHOWN);
        }
    }
```

Display 3.20

An Applet that Uses Looping and Branching

# Programming Example, cont.

- class MultipleFaces, contd.

```
//i == 5 ← After the last iteration of the loop body, the
//Draw kissing face:
//Draw face circle:
canvas.setColor(Color.BLACK);
canvas.drawOval(X_FACE0 + 50*i, Y_FACE0 + 30*i,
               FACE_DIAMETER, FACE_DIAMETER);

//Draw eyes:
canvas.setColor(Color.BLUE);
canvas.fillOval(X_RIGHT_EYE0 + 50*i, Y_RIGHT_EYE0 + 30*i,
               EYE_WIDTH, EYE_HEIGHT);
canvas.fillOval(X_LEFT_EYE0 + 50*i, Y_LEFT_EYE0 + 30*i,
               EYE_WIDTH, EYE_HEIGHT);

//Draw nose:
canvas.setColor(Color.BLACK);
canvas.fillOval(X_NOSE0 + 50*i, Y_NOSE0 + 30*i,
               NOSE_DIAMETER, NOSE_DIAMETER);

//Draw mouth in shape of a kiss:
canvas.setColor(Color.RED);
canvas.fillOval(X_MOUTH0 + 50*i + 10, Y_MOUTH0 + 30*i,
               MOUTH_WIDTH 20, MOUTH_WIDTH 20);

//Add text:
canvas.drawString("Kiss, Kiss.",
                 X_FACE0 + 50*i + FACE_DIAMETER, Y_FACE0 + 30*i);
```

```
//Draw blushing face:
i++;
//Draw face circle:
canvas.setColor(Color.PINK);
canvas.fillOval(X_FACE0 + 50*i, Y_FACE0 + 30*i,
               FACE_DIAMETER, FACE_DIAMETER);
canvas.setColor(Color.BLACK);
canvas.drawOval(X_FACE0 + 50*i, Y_FACE0 + 30*i,
               FACE_DIAMETER, FACE_DIAMETER);

//Draw eyes:
canvas.setColor(Color.BLUE);
canvas.fillOval(X_RIGHT_EYE0 + 50*i, Y_RIGHT_EYE0 + 30*i,
               EYE_WIDTH, EYE_HEIGHT);
canvas.fillOval(X_LEFT_EYE0 + 50*i, Y_LEFT_EYE0 + 30*i,
               EYE_WIDTH, EYE_HEIGHT);

//Draw nose:
canvas.setColor(Color.BLACK);
canvas.fillOval(X_NOSE0 + 50*i, Y_NOSE0 + 30*i,
               NOSE_DIAMETER, NOSE_DIAMETER);

//Draw mouth:
canvas.setColor(Color.RED);
canvas.drawArc(X_MOUTH0 + 50*i, Y_MOUTH0 + 30*i, MOUTH_WIDTH,
               MOUTH_HEIGHT0 + 3*4, //i == 4 is the smile
               MOUTH_START_ANGLE, MOUTH_DEGREES_SHOWN);

//Add text:
canvas.drawString("Tee Hee.",
                 X_FACE0 + 50*i + FACE_DIAMETER, Y_FACE0 + 30*i);
```

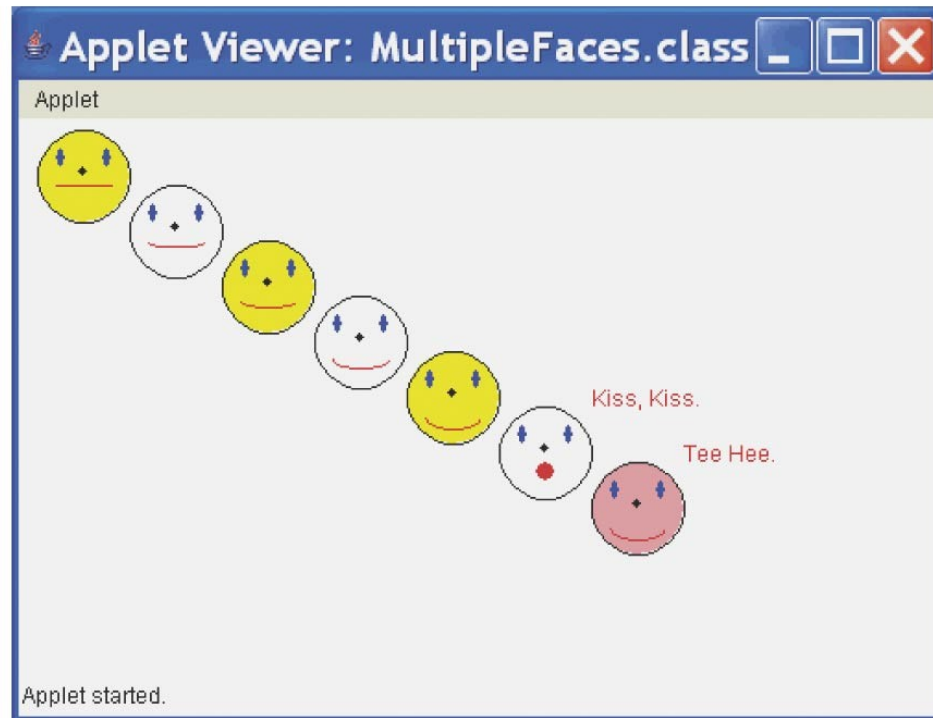
```
}
```

Display 3.20

An Applet that Uses Looping and Branching

# Programming Example, cont.

Resulting Applet



Display 3.20  
Class MultipleFaces

# 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`.

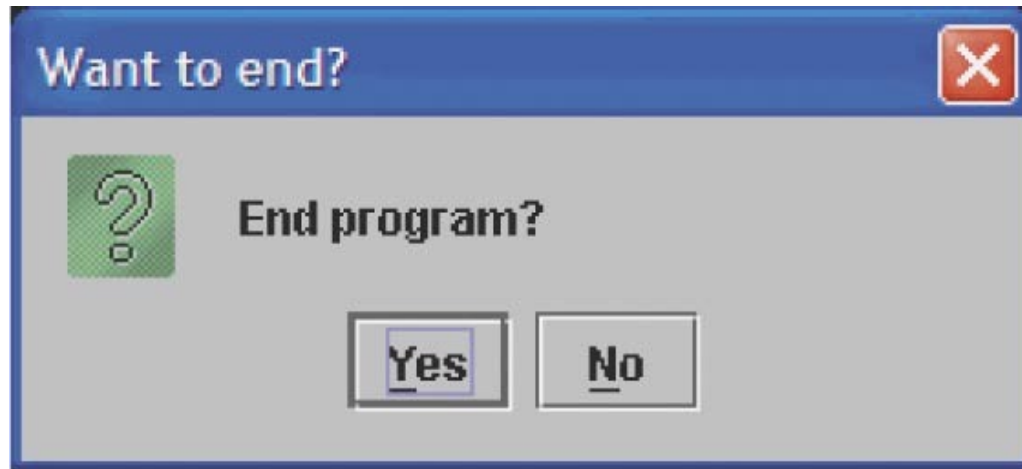
# A JOptionPane Yes/No Window, cont.

- 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 `ints`.
- 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.

# Summary

- You have learned about Java branching statements.
- You have learned about loops.
- You have learned about the type `boolean`.
- (optional) You have learned to use color and the `JOptionPane` yes/no window.