Chapter 5

Control Statements

Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

Introduction

- Before writing a program
  - Have a thorough understanding of problem
  - Carefully planned approach for solving it
- While writing a program
  - Know what “building blocks” are available
  - Use good programming principles

Algorithms

- Computing problems
  - All can be solved by executing a series of actions in a specific order
- Algorithm
  - Procedure in terms of
    - *actions* to be executed
    - *order* in which these actions are to be executed
- Program control
  - Specify order in which statements are to be executed

Pseudocode

- Pseudocode
  - Artificial, informal language
    - Helps develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - “Think out” a program before writing it in a programming language
    - Easy to convert into a corresponding Java program
    - Consists only of executable statements
      - Declarations are not executable statements
      - Actions: input, output, calculation

Control Structures

- Sequential execution
  - Statements executed one after the other in the order written
- Transfer of control
  - Next statement executed is not the next one in sequence.
  - Overuse of `goto` in 1960's led to many problems
    - Java does not have `goto`
- Bohm and Jacopini demonstrated that all programs written in terms of 3 control structures, known as the structured programming constructs.

1. Sequence: This is the simplest control structure and represents a sequence of statements executed one after the other. It's denoted by writing statements one below the other. For example:

Statement 1

Statement 2

Statement 3

Control Structures

2. Selection (or Conditional): This structure involves making decisions in a program. It allows you to execute one of two or more blocks of code based on a condition. In most programming languages, this is typically implemented using if, else if, and else statements. For example: Three types: **`if`**, **`if/else`**, and **`switch`**

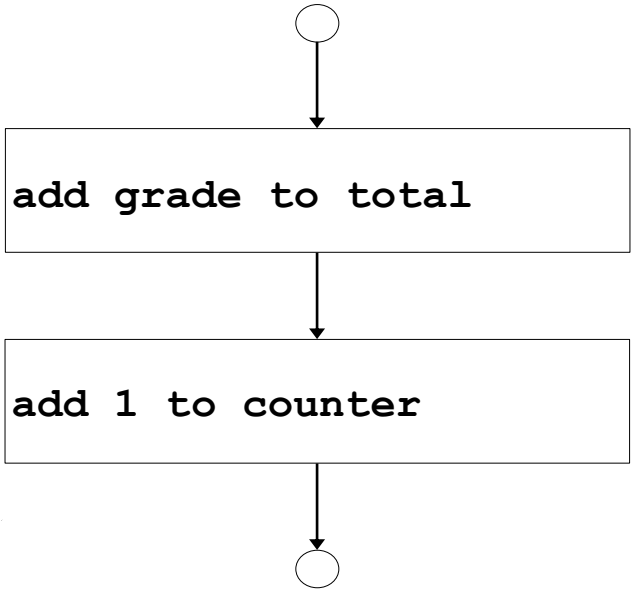
3. **Iteration (or Loop):** This structure allows you to repeat a block of code multiple times as long as a certain condition is met. Common loop constructs include for and while loops.

Control Structures

- Flowchart
  - Graphical representation of an algorithm
  - Drawn using symbols connected by arrows called *flowlines*
    - Rectangle symbol (action symbol)
      - Indicates any type of action.
    - Oval symbol:
      - Indicates beginning or end of a program, or a section of code (circles)
  - When flowcharting a complete algorithm
    - Oval containing "Begin" is first symbol
    - Oval containing "End" is last symbol
    - When drawing a portion, small circles used
  - Useful for algorithms, but psuedocode generally preferred

Control Structures

- Flowchart of sequence structure
  - Two actions performed in order




```
total = total + grade;
```

```
counter = counter + 1;
```

Control Structures

- Single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (*control-structure stacking*)
  - Makes programs easy to build
  - Control structure nesting
    - Only other way to connect control structures
- Algorithms in Java
  - Constructed from seven types of control structures
  - Only two ways to combine them

The if Selection Structure

- Selection structure
  - Used to choose among alternative courses of action
    - Pseudocode: *If student's grade is greater than or equal to 60*
      Print "Passed"
  - If condition **`true`**
    - Print statement executed and program goes on to next statement
  - If condition **`false`**
    - Print statement is ignored and the program goes onto the next statement
  - Indenting makes programs easier to read
    - Java ignores whitespace characters

The if Selection Structure

- Selection structure
  - Psuedocode statement

    If student's grade is greater than or equal to 60
    Print "Passed"
  - Statement in Java

```
if ( studentGrade >= 60 )
   System.out.println( "Passed" );
```

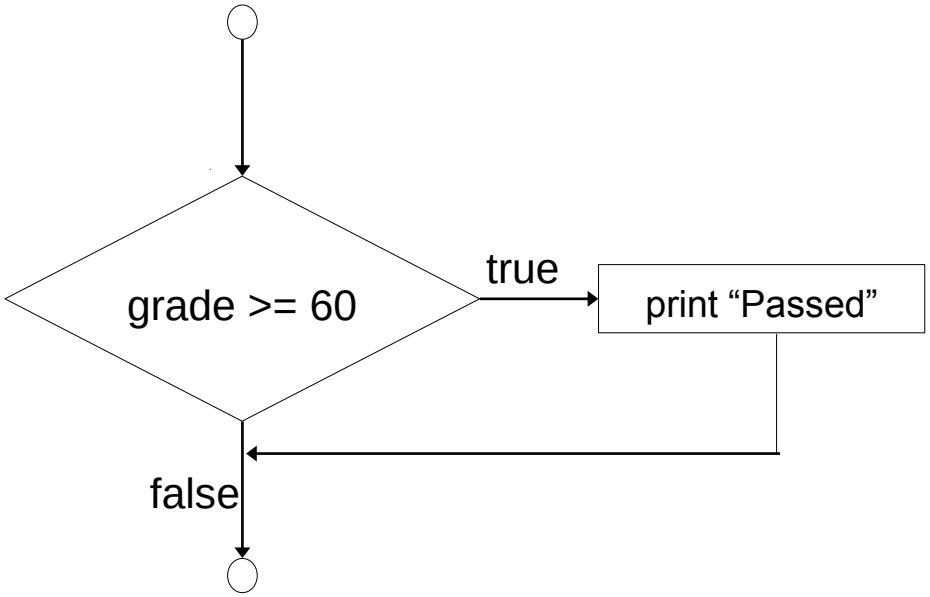
    - Corresponds closely to pseudocode
- Flowcharts
  - Diamond symbol - decision symbol
    - Important - indicates an decision is to be made
    - Contains expression that can be **`true`** or **`false`**

The if Selection Structure

- Flowcharts
  - Diamond symbol has two paths
    - Indicates what to do if condition **`true`** or **`false`**
  - Decision can be made on anything that evaluates to a value of data type **`boolean`**
    - **`true`** or **`false`**

Flowchart example of the **`if`** selection structure

`if` is a single-entry/single-exit structure



The if/else Selection Structure

- Selection structures
  - **`if`**
    - Only performs an action if condition **`true`**
  - **`if/else`**
    - Performs different action when condition **`true`** than when condition **`false`**
- Psuedocode

If student's grade is greater than or equal to 60
Print "Passed"
else
Print "Failed"

  - Java code:

```
if ( studentGrade >= 60 )
   System.out.println( "Passed" );
else
   System.out.println( "Failed" );
```

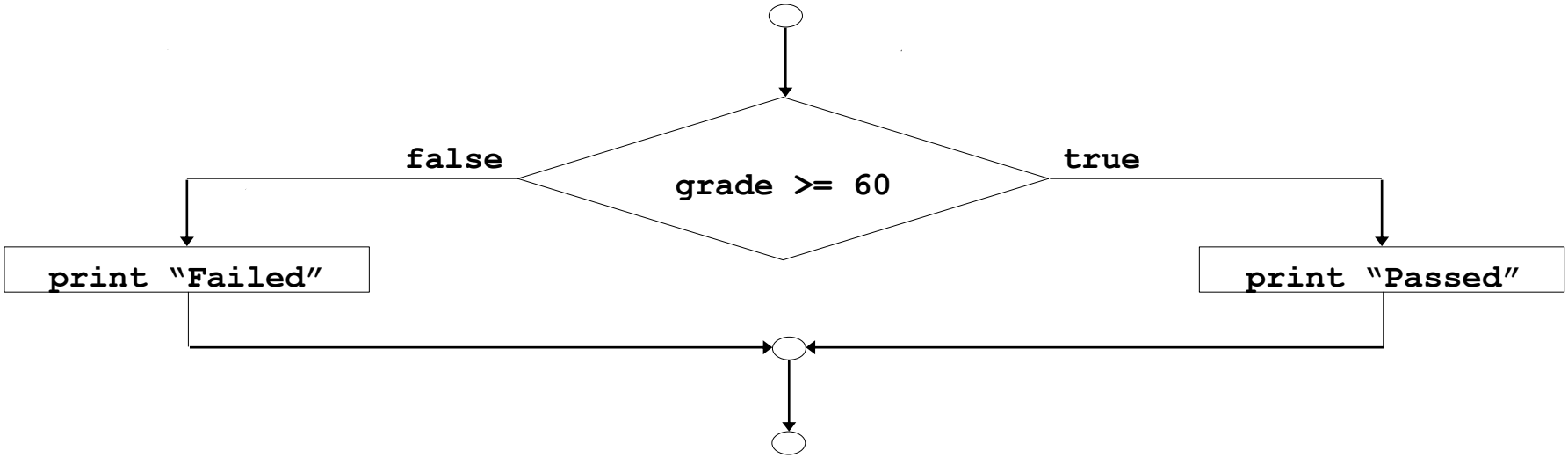
Note spacing/indentation conventions

The if/else Selection Structure

```
int a, b;
// ...
if(a < b) a = 0;
else b = 0;
```

The if/else Selection Structure

- Flowchart of **`if/else`** structure
  - Note that only symbols (besides circles and arrows) are
    - Rectangles: actions
    - Diamonds: decisions
  - Action/decision model model of computing
    - Programmer's job to assemble structures by stacking and nesting, then the define actions and decisions



The if/else Selection Structure

- Ternary conditional operator (**?:**)
  - Takes three arguments
    - Condition **?** value if **true :** value if **false**
  - Our pseudocode could be written:

```
System.out.println(
    studentGrade >= 60 ? "Passed" : "Failed" );
```

- Nested **if/else** structures
  - Test for multiple cases
  - Place **if**/**else** structures inside **if**/**else** structures
  - If first condition met, other statements skipped

The if/else Selection Structure

- Nested structures

```
if ( studentGrade >= 90 )
    System.out.println( "A" );
 else
    if ( studentGrade >= 80 )
       System.out.println( "B" );
    else
       if ( studentGrade >= 70 )
          System.out.println( "C" );
       else
          if ( studentGrade >= 60 )
             System.out.println( "D" );
          else
             System.out.println( "F" );
```

`else` only executes when the **if** condition fails.

Once a condition is met, the rest of the statements are skipped.

The if/else Selection Structure

- Alternate form of nested structures
  - Avoids deep indentation
  - Generally preferred to previous format

```
if ( grade >= 90 )
   System.out.println( "A" );
else if ( grade >= 80 )
   System.out.println( "B" );
else if ( grade >= 70 )
   System.out.println( "C" );
else if ( grade >= 60 )
   System.out.println( "D" );
else
   System.out.println( "F" );
```

As before, **`else`** only executes when the **`if`** condition fails.

The if/else Selection Structure

```
// Demonstrate if-else-if statements.
class IfElse {
  public static void main(String args[]) {
    int month = 4; // April
    String season;
    if(month == 12 || month == 1 || month == 2)
      season = "Winter";
    else if(month == 3 || month == 4 || month == 5)
      season = "Spring";
    else if(month == 6 || month == 7 || month == 8)
      season = "Summer";
    else if(month == 9 || month == 10 || month == 11)
      season = "Autumn";
    else
      season = "Bogus Month";
    System.out.println("April is in the " + season + ".");
  }}
```

The if/else Selection Structure

- Important note
  - Java always associates **`else`** with previous **`if`** unless braces (**`{}`**) present
    - Dangling-else problem

```
if ( x > 5 )
   if ( y > 5 )
       System.out.println( "x and y are > 5" );
else
   System.out.println( "x is <= 5" );
```

  - Does not execute as it appears, executes as

```
if ( x > 5 )
   if ( y > 5 )
       System.out.println( "x and y are > 5" );
   else
       System.out.println( "x is <= 5" );
```

If **`x<= 5`**, nothing is output.

The if/else Selection Structure

- Important note
  - Must force structure to execute as intended
    - Use braces to indicate that second **`if`** in body of first

```
if ( x > 5 ) {
  if ( y > 5 )
      System.out.println( "x and y are > 5" );
}
else
   System.out.println( "x is <= 5" );
```

- Compound statements
  - Compound statement - set of statements within braces
    - Can be used wherever a single statement can
  - **`if`** expects one statement in its body
  - To enclose multiple statements, enclose them in braces

The if/else Selection Structure

- Compound statements
  - Example:

```
if (grade >= 60)
    System.out.println( "Passed" );
else {
    System.out.println( "Failed" );
    System.out.println( "You must take this course again." );
}
```

    - Without braces, second **println** always executes
  - Compound statements may contain declarations
    - Example: body of **main**

The if/else Selection Structure

- Errors
  - Syntax errors
    - Caught by compiler
    - Example: forgetting a brace in a compound statement
  - Logic errors
    - Have effect at execution time
      - Non-fatal: program runs, but has incorrect output
      - Fatal: program exits prematurely

The while Repetition Structure

- **`while`** repetition structure
  - Repeat an action while some condition remains **`true`**
  - Psuedocode:

    While product is less than or equal to 1000
    double its value
  - **`while`** loop repeated until condition becomes **`false`**
    - First statement after repetition structure executed
  - Body may be a single or compound statement
  - If condition initially false then body never executed

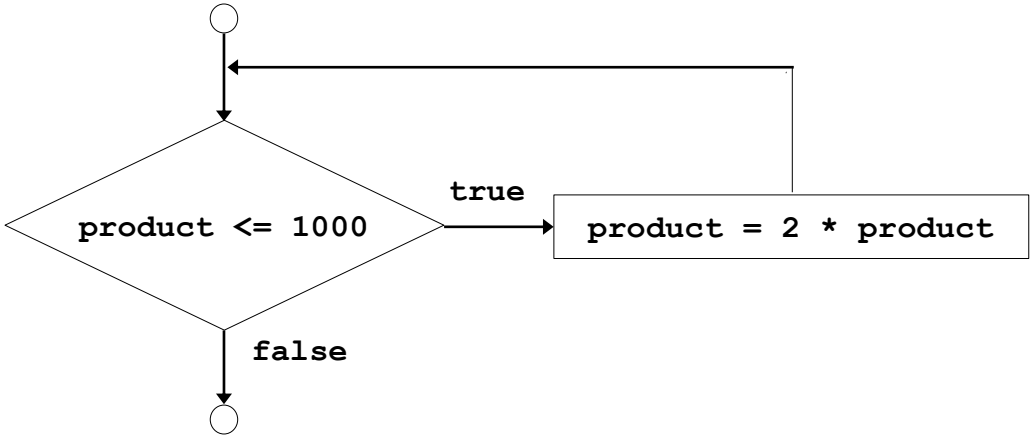
The while Repetition Structure

- Example

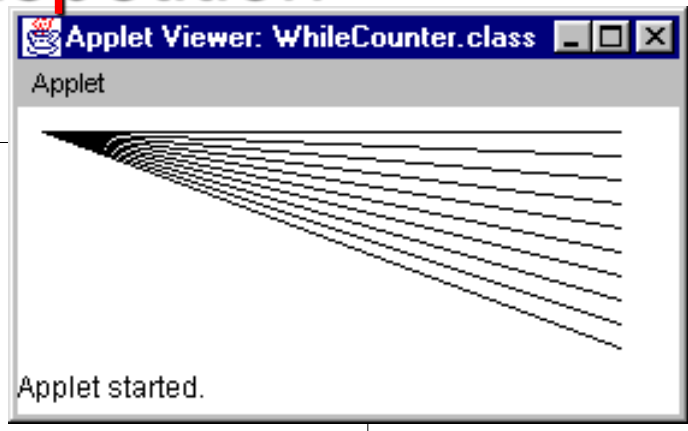
```
int product = 2;

while ( product <= 1000 )
   product = 2 * product;
```

- Flowchart



Counter-Controlled Repetition




```
1// WhileCounter.java
 2// Counter-controlled repetition
 3import java.awt.Graphics;
 4import javax.swing.JApplet;
 5
6public class WhileCounter extends JApplet {
 7   public void paint( Graphics g )
 8   {
 9      int counter = 1;                  // initialization
10
11    while ( counter <= 10 ) { // repetition condition
12       g.drawLine( 10, 10, 250, counter * 10 );
13       ++counter;                        // increment
14     }
15   }
16}
```

Essentials of Counter-Controlled Repetition

```
9       int counter = 1;                    // initialization
```

- Name and initialize the control variable
  - Declarations alone are not executable statements
  - Assignment statements are executable statements

```
11    while ( counter <= 10 ) { // repetition condition
```

- Condition to test for final value (`11`)

```
12          g.drawLine( 10, 10, 250, counter * 10 );
```

- Method `drawLine( x1, y1, x2, y2 )`
  - Called using reference to Graphics object
  - Draws line from (`x1`, `y1`) to (`x2`, `y2`)

```
13          ++counter;                          // increment
```

- Increment

Essentials of Counter-Controlled Repetition

- Loop can be shortened
  - Initialize **`counter`** to zero
- Change loop to:

```
while ( ++counter <= 10 )  //repetition condition
   g.drawLine( 10, 10, 250, counter *10 );
```

  - Increment done inside **`while`**

Counter-Controlled Repetition

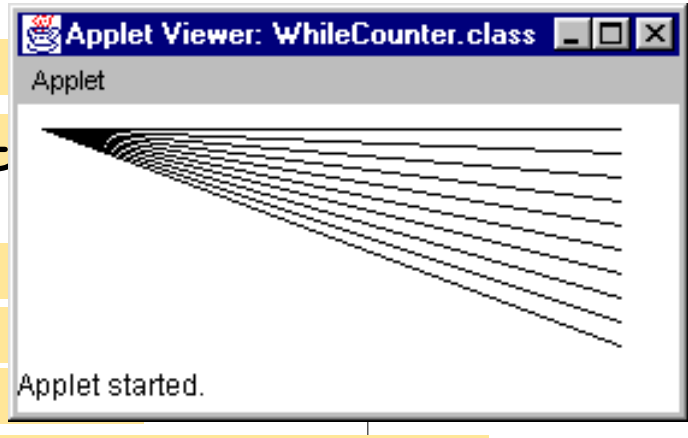
```
// Demonstrate the while loop.
class While {
  public static void main(String args[]) {
    int n = 10;

    while(n > 0) {
      System.out.println("tick " + n);
      n--;
    }
  }
}
```

The `for` Repetition Structure

- Redo previous example
  - Use **`for`** structure

The `for` Repetition Structure




```
1// ForCounter.java
2// Counter-controlled repetition wit
the for structure
3import java.awt.Graphics;
4import javax.swing.JApplet;
5
6public class ForCounter extends JApplet {
7   public void paint( Graphics g )
8   {
9// Initialization, repetition condition and incrementing


10       // are all included in the for structure header.
11       for ( int counter = 1; counter <= 10; counter++ )
12          g.drawLine( 10, 10, 250, counter * 10 );
13   }
14}
```

The `for` Repetition Structure

```
11        for ( int counter = 1; counter <= 10; counter++ )
12           g.drawLine( 10, 10, 250, counter * 10 );
```

- Immediate observations
  - **`for`** "does it all" : initialization, condition, increment
- General format

  `for (` *initialization* **`;`** *loopContinuationTest* **`;`** *increment* **`)`**
  statement

  - If multiple statements needed, enclose in braces
  - Control variable only exists in body of **`for`** structure
  - If *loopContinuationTest* is initially **`false`**, body not executed

The `for` Repetition Structure

- May use arithmetic expressions in **`for`** loops
  - Let **`x =2`**, **`y=10`**

```
for ( int j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

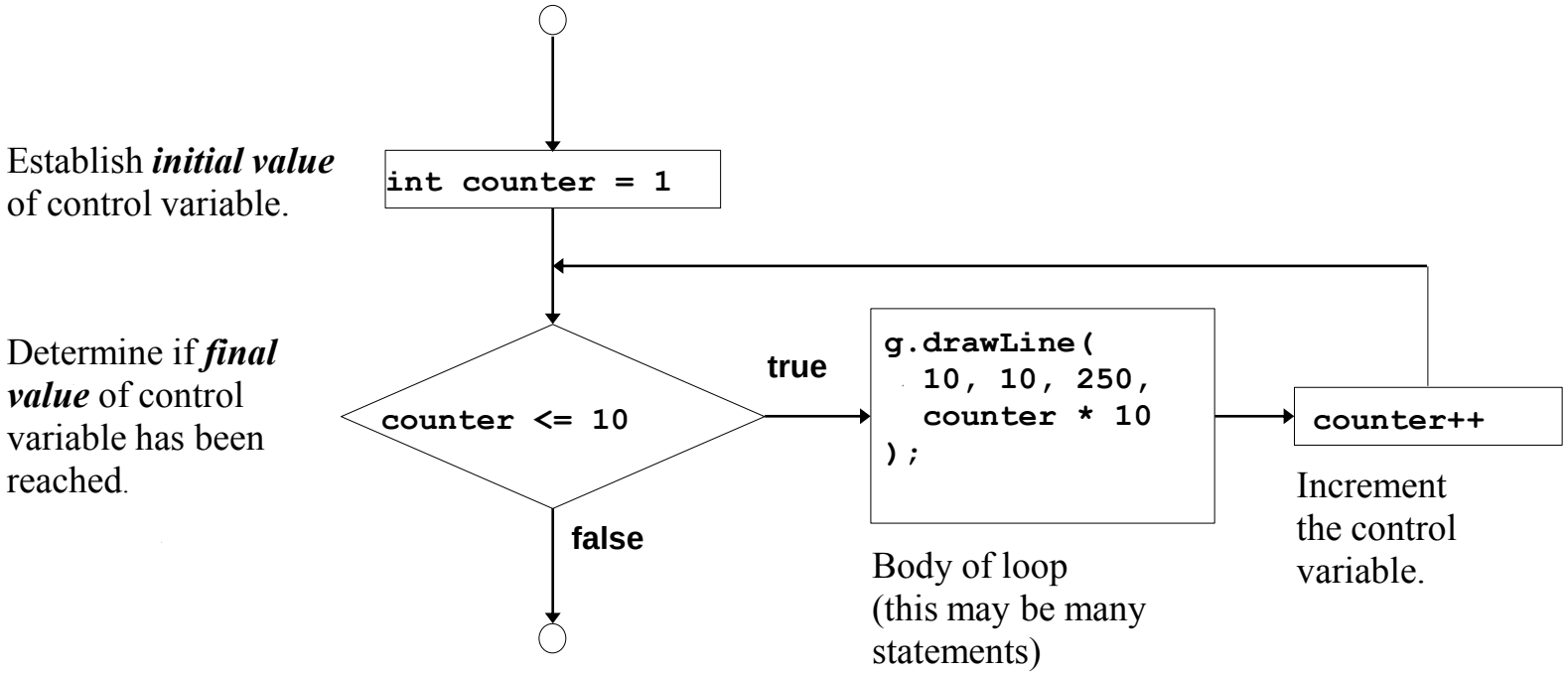
```
for ( int j = 2; j <= 80; j += 5 )
```

- **`for`** can usually be written as a **`while`** loop:

```
initialization;
while ( loopContinuationTest ) {
   statement
   increment;
}
```

The `for` Repetition Structure

```
11        for ( int counter = 1; counter <= 10; counter++ )
12           g.drawLine( 10, 10, 250, counter * 10 );
```



Examples Using the `for` Structure

- Problem
  - Calculate the value each year of a $1000 deposit, yielding 5% annually
    - Calculate the value for 10 years
  - Use $a = p(1 + r)^n$
    - p - principal
    - r - interest rate
    - n - number of years
    - a - amount on deposit after *n*th year

Examples Using the `for` Structure

```
// Using the comma.
class Comma {
  public static void main(String args[]) {
    int a, b;
    for(a=1, b=4; a<b; a++, b--) {
      System.out.println("a = " + a);
      System.out.println("b = " + b);
    }
  }
}
```

In C/C++ comma is an operator that can be used in any valid expression. In Java, the comma is a separator that applies only to the for loop.

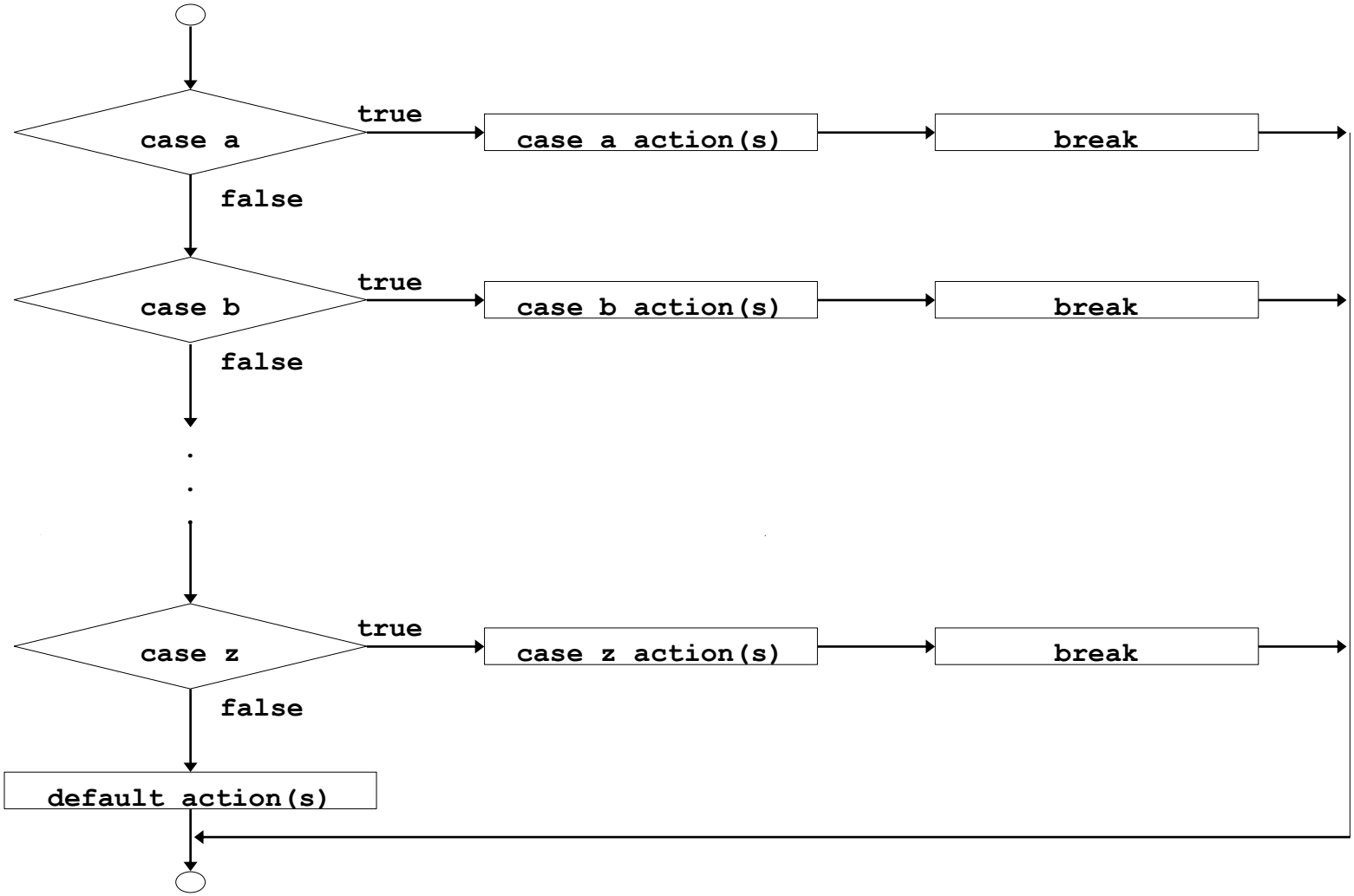
The `switch` Multiple-Selection Structure

- **`switch`** statements
  - Useful to test a variable for different values
    - Different action taken
- Format
  - Series of **`case`** labels and an optional **`default`** case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
    }
```

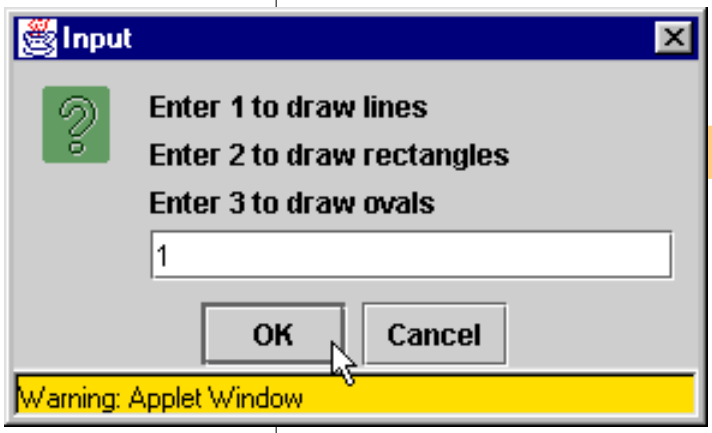
  - **`break;`** causes exit from structure

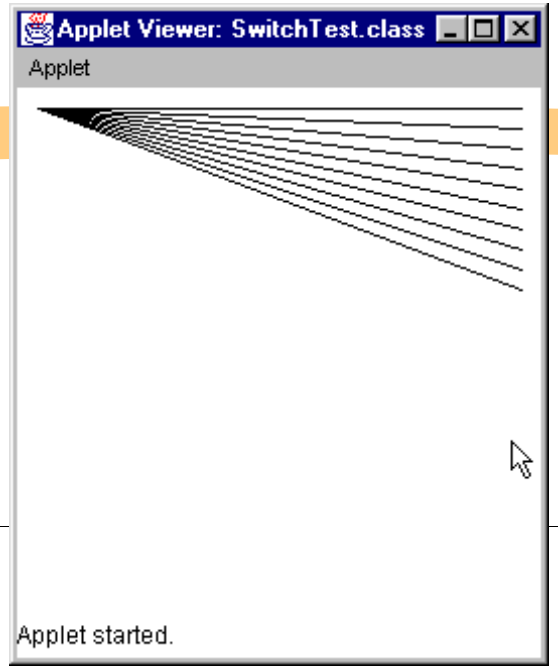
The `switch` Multiple-Selection Structure

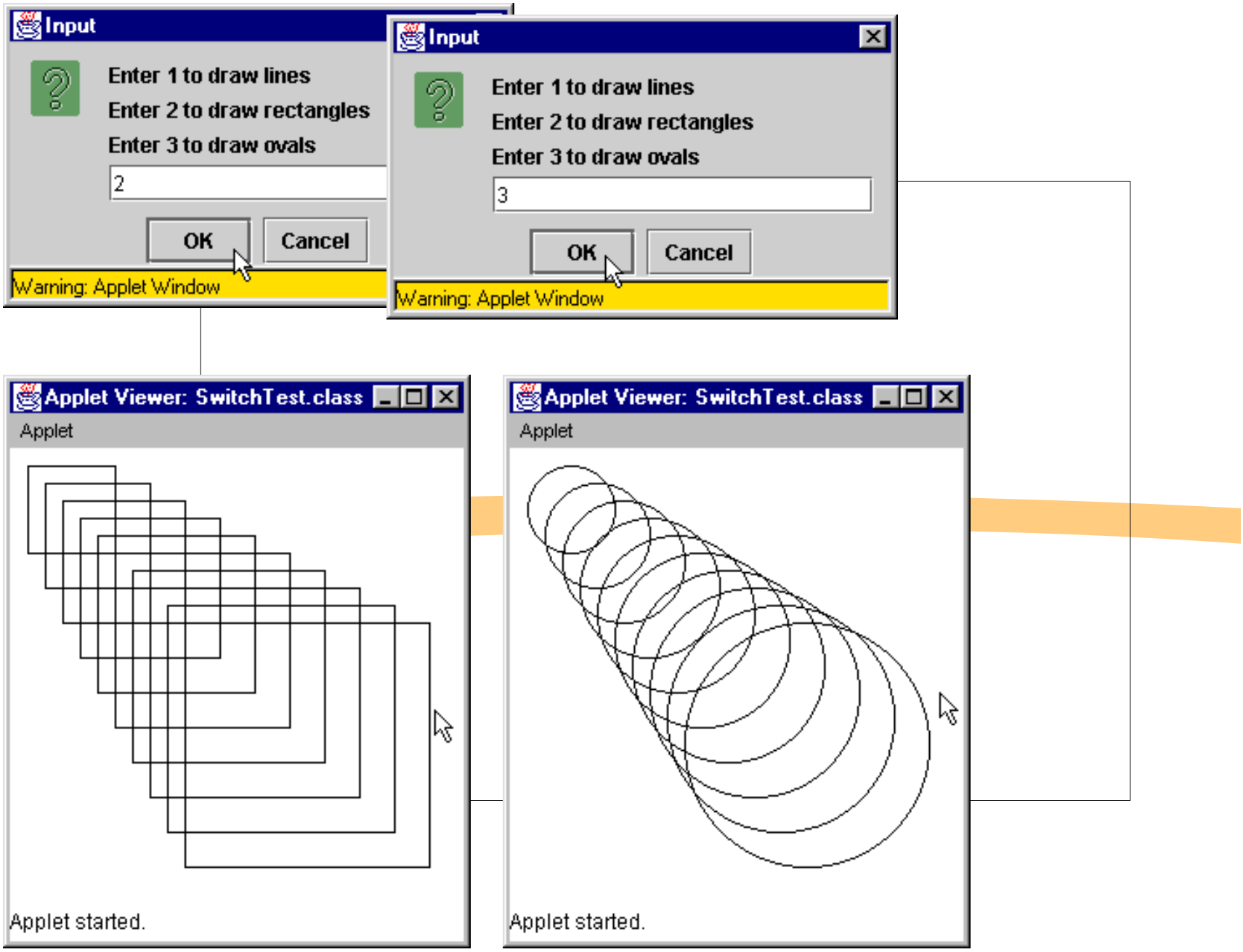


```
1// SwitchTest.java
2// Counting letter grades
3import java.awt.Graphics;
4import javax.swing.*;
5
6public class SwitchTest extends JApplet {
7   int choice;
8
9   public void init()
10   {
11      String input;
12
13      input = JOptionPane.showInputDialog(
14                  "Enter 1 to draw lines\n" +
15                  "Enter 2 to draw rectangles\n" +
16                  "Enter 3 to draw ovals\n" );
17
18      choice = Integer.parseInt( input );
19   }
20
21   public void paint( Graphics g )
22   {
23      for ( int i = 0; i < 10; i++ ) {
24         switch( choice ) {
25            case 1:
26               g.drawLine( 10, 10, 250, 10 + i * 10 );
27               break;
28            case 2:
29               g.drawRect( 10 + i * 10, 10 + i * 10,
30                           50 + i * 10, 50 + i * 10 );
31               break;
```

```
32            case 3:
33               g.drawOval( 10 + i * 10, 10 + i * 10,
34                           50 + i * 10, 50 + i * 10 );
35               break;
36            default:
37               JOptionPane.showMessageDialog(
38                  null, "Invalid value entered" );
39         } // end switch
40      } // end for
41   } // end paint()
42} // end class SwitchTest
```






Input
Enter 1 to draw lines
Enter 2 to draw rectangles
Enter 3 to draw ovals
2
OK
Cancel
Warning: Applet Window
Input
Enter 1 to draw lines
Enter 2 to draw rectangles
Enter 3 to draw ovals
3
OK
Cancel
Warning: Applet Window
Applet Viewer: SwitchTest.class
Applet
Applet started.
Applet Viewer: SwitchTest.class
Applet
Applet started.

The `switch` Multiple-Selection Structure

```
7   int choice;
9   public void init()
10   {
11      String input;
12
13      input = JOptionPane.showInputDialog(
14                  "Enter 1 to draw lines\n" +
15                  "Enter 2 to draw rectangles\n" +
16                  "Enter 3 to draw ovals\n" );
17
18      choice = Integer.parseInt( input );
19   }
```

The `switch` Multiple-Selection Structure

```
24          switch( choice ) {
25             case 1:
26                g.drawLine( 10, 10, 250, 10 + i * 10 );
27                break;
```

- **`switch`** structure - compare **`choice`** to **`case`**s
  - **`case`** labels - can be constant integral values of type **`byte`**, **`short`**, **`int`**, **`long`**, and **`char`**
    - Use single quotes to represent characters: **`'A'`**
    - Can have multiple actions per **`case`**
  - **`break`** - exits **`switch`** structure

```
36             default:
37                JOptionPane.showMessageDialog(
38                   null, "Invalid value entered" );
```

  - **`default`** label - optional, actions to take if no cases met

```
1// SwitchTest.java
2// Counting letter grades
3import java.awt.Graphics;
4import javax.swing.*;
5
6public class SwitchTest extends JApplet {
7   int choice;
8
9   public void init()
10   {
11      String input;
12
13      input = JOptionPane.showInputDialog(
14                  "Enter 1 to draw lines\n" +
15                  "Enter 2 to draw rectangles\n" +
16                  "Enter 3 to draw ovals\n" );
17
18      choice = Integer.parseInt( input );
19   }
20
21   public void paint( Graphics g )
22   {
23      for ( int i = 0; i < 10; i++ ) {
24         switch( choice ) {
25            case 1:
26               g.drawLine( 10, 10, 250, 10 + i * 10 );
27               break;
28            case 2:
29               g.drawRect( 10 + i * 10, 10 + i * 10,
30                           50 + i * 10, 50 + i * 10 );
31               break;
```

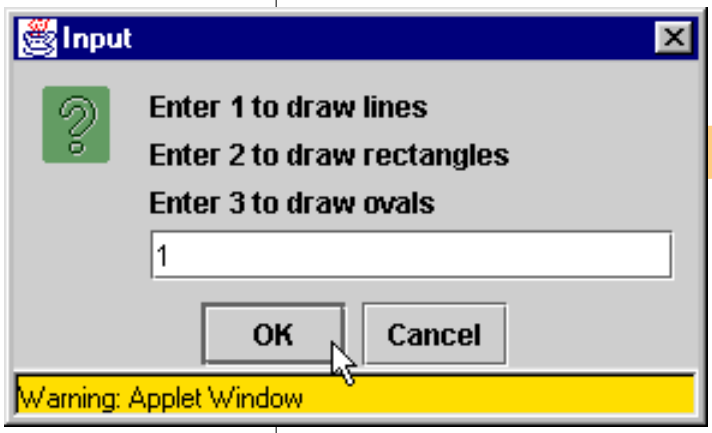
Place the value to compare inside the **`switch`** statement.

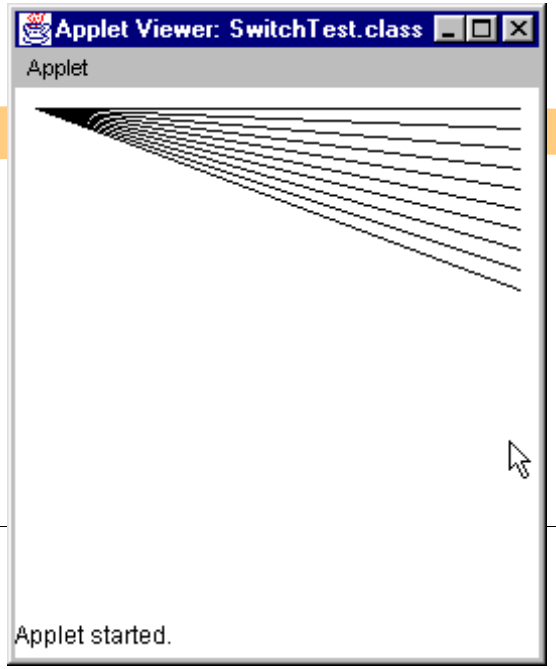
Notice how **`case`** labels are used to test for the integer entered.

`break` exits the switch structure.

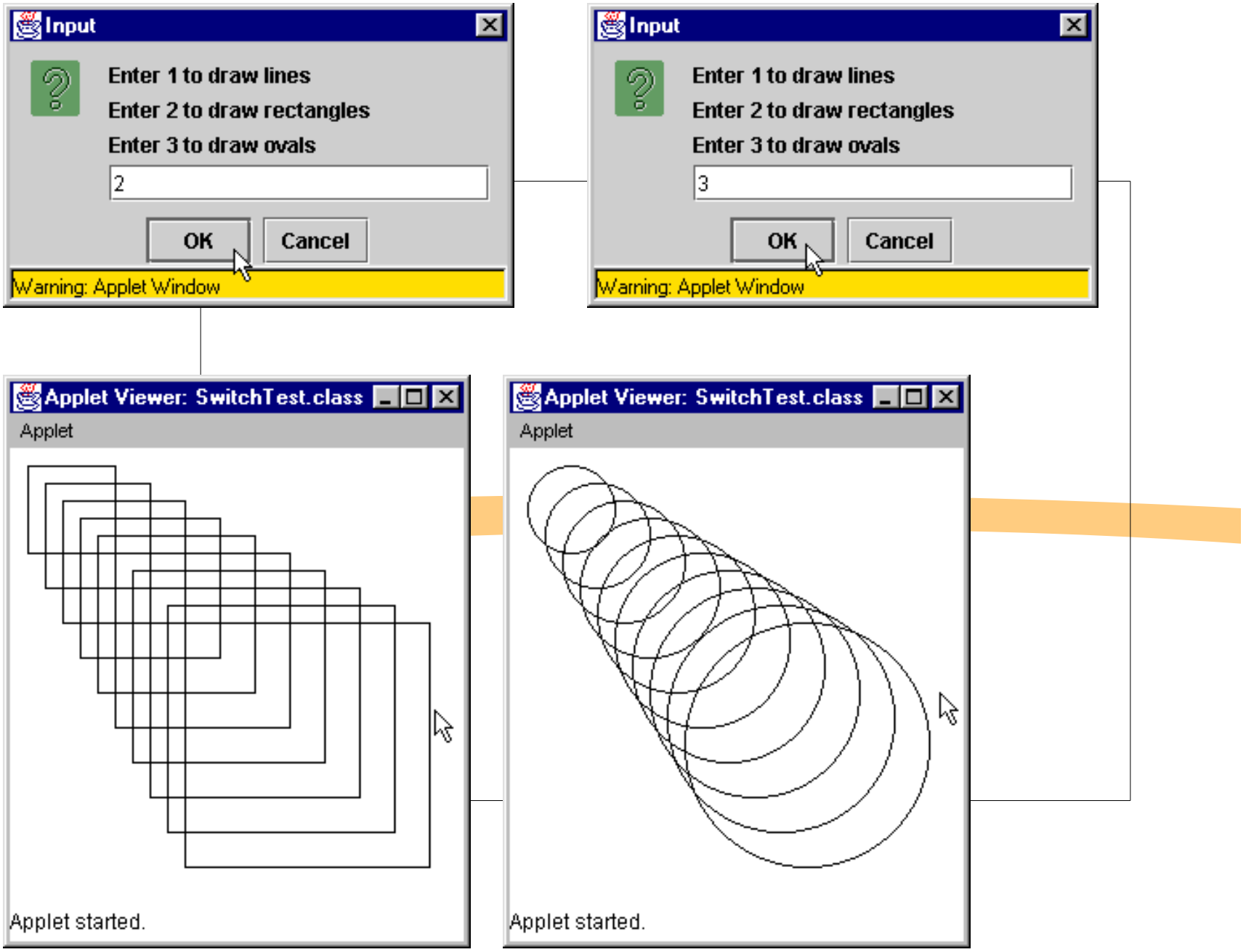
```
32            case 3:
33               g.drawOval( 10 + i * 10, 10 + i * 10,
34                           50 + i * 10, 50 + i * 10 );
35               break;
36            default:
37               JOptionPane.showMessageDialog(
38                  null, "Invalid value entered" );
39         } // end switch
40      } // end for
41   } // end paint()
42} // end class SwitchTest
```

default case





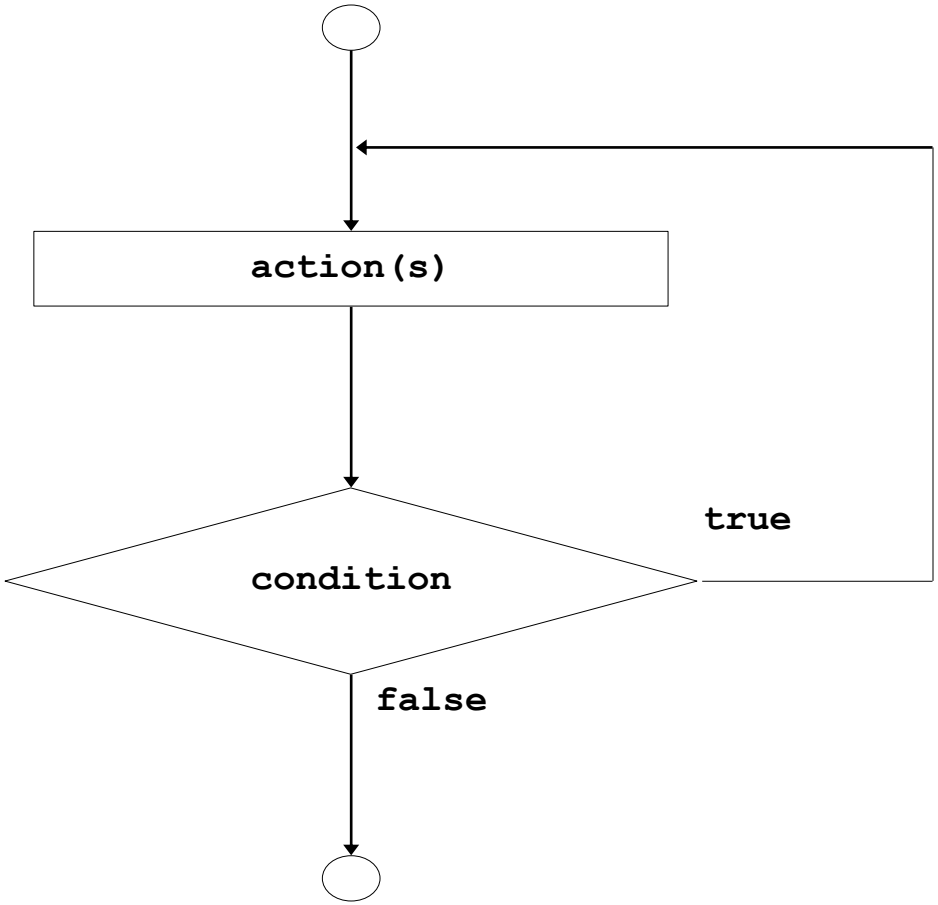
Program Output


Input
Enter 1 to draw lines
Enter 2 to draw rectangles
Enter 3 to draw ovals
2
OK
Cancel
Warning: Applet Window
Input
Enter 1 to draw lines
Enter 2 to draw rectangles
Enter 3 to draw ovals
3
OK
Cancel
Warning: Applet Window
Applet Viewer: SwitchTest.class
Applet
Applet started.
Applet Viewer: SwitchTest.class
Applet
Applet started.

The `do/while` Repetition Structure

- The **`do/while`** repetition structure
  - Similar to the **`while`** structure
  - Condition for repetition tested *after* the body of the loop is performed
  - Actions are performed at least once
- Format
  - **`do {`**
    statement
    `} while (` *condition* **`);`**
  - Good practice to put brackets in, even if not required

The do/while Repetition Structure

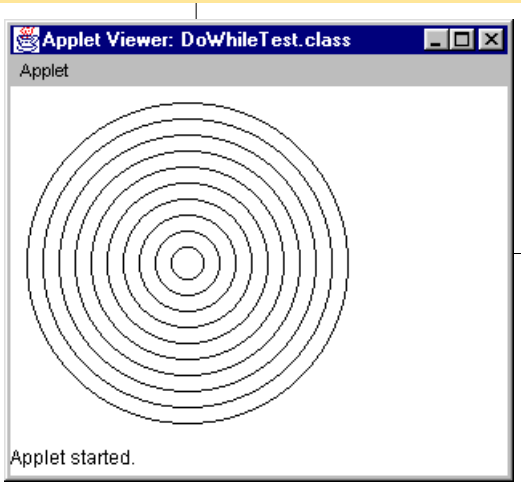


```
1// DoWhileTest.java
2// Using the do/while repetition structure
3import java.awt.Graphics;
4import javax.swing.JApplet;
5
6public class DoWhileTest extends JApplet {
7   public void paint( Graphics g )
8   {
9      int counter = 1;
10
11      do {
12         g.drawOval( 110 - counter * 10, 110 - counter * 10,
13                     counter * 20, counter * 20 );
14         ++counter;
15      } while ( counter <= 10 );
16   }
17}
```

Notice format of **do/while** loop.

Method **drawOval(x1, y1, width, height)**

Same arguments as **drawRect**, but the rectangle defines the oval's bounding box.



The `break` and `continue` Statements

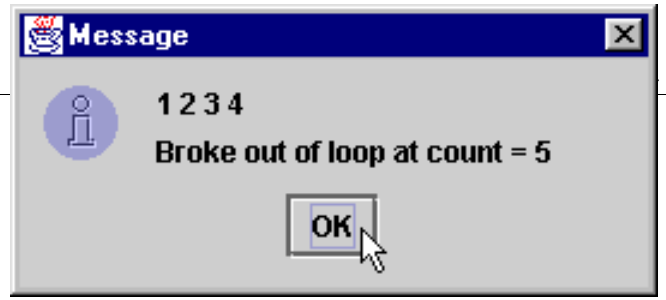
- **`break`**
  - Immediate exit from **`while`**, **`for`**, **`do`**/**`while`** or **`switch`**
  - Program continues with the first statement after the structure
  - Common uses of the **`break`** statement
    - Escape early from a loop
    - Skip the remainder of a **`switch`** structure

The `break` and `continue` Statements

- **`continue`**
  - Skips the remaining statements in body of **`while`**, **`for`** or **`do`**/**`while`**
    - Proceeds with the next iteration of the loop
  - **`while`** and **`do`**/**`while`**
    - Loop-continuation test is evaluated immediately after **`continue`**
  - **`for`** structure
    - Increment expression is executed, then the loop-continuation test is evaluated

```
1// BreakTest.java
2// Using the break statement in a for structure
3import javax.swing.JOptionPane;
4
5public class BreakTest {
6   public static void main( String args[] )
7   {
8      String output = "";
9      int count;
10
11      for ( count = 1; count <= 10; count++ ) {
12         if ( count == 5 )
13            break; // break loop only if count == 5
14
15         output += count + " ";
16      }
17
18      output += "\nBroke out of loop at count = " + count;
19      JOptionPane.showMessageDialog( null, output );
20      System.exit( 0 );
21   }
22}
```

break causes an immediate exit from the loop.



```
1// ContinueTest.java
2// Using the continue statement in a for structure
3import javax.swing.JOptionPane;
4
5public class ContinueTest {
6   public static void main( String args[] )
7   {
8      String output = "";
9
10      for ( int count = 1; count <= 10; count++ ) {
11         if ( count == 5 )
12            continue;  // skip remaining code in loop
13                       // only if count == 5
14
15         output += count + " ";
16      }
17
18      output += "\nUsed continue to skip printing 5";
19      JOptionPane.showMessageDialog( null, output );
20      System.exit( 0 );
21   }
```

continue skips the rest of the body and goes to the next iteration.

The Labeled `break` and `continue` Statements

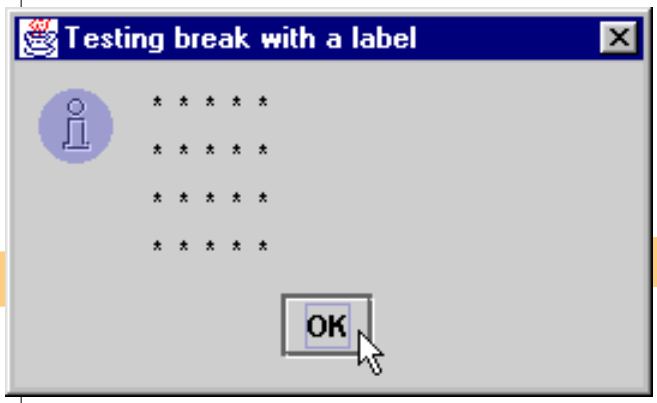
- Nested set of structures
  - **`break`** statement
    - Can only break out of immediately enclosing structure
  - Use labeled **`break`** statement
    - Label - identifier followed by colon, i.e. **`myLabel:`**
    - Breaks out of enclosing statement and any number of repetition structures
    - Program resumes after enclosing *labeled compound statement*
  - Labeled **`continue`** statement
    - Skips statements in enclosing structure
    - Continues with next iteration of enclosing *labeled repetition structure*
      - Repetition structure preceded by a label

```
1// BreakLabelTest.java
2// Using the break statement with a label
3import javax.swing.JOptionPane;
4
5public class BreakLabelTest {
6   public static void main( String args[] )
7   {
8      String output = "";
9
10      stop: {   // labeled compound statement
11         for ( int row = 1; row <= 10; row++ ) {
12            for ( int column = 1; column <= 5 ; column++ ) {
13
14               if ( row == 5 )
15                  break stop; // jump to end of stop block
16
17               output += "*  ";
18            }
19
20            output += "\n";
21         }
22
23         // the following line is skipped
24         output += "\nLoops terminated normally";
25      }
26
```

Begins labeled compound statement **`stop:`**

Labeled **`break`** statement to exit **`stop`** block.

```
27       JOptionPane.showMessageDialog(
28          null, output,"Testing break with a label",
29          JOptionPane.INFORMATION_MESSAGE );
30       System.exit( 0 );
31    }
32}
```

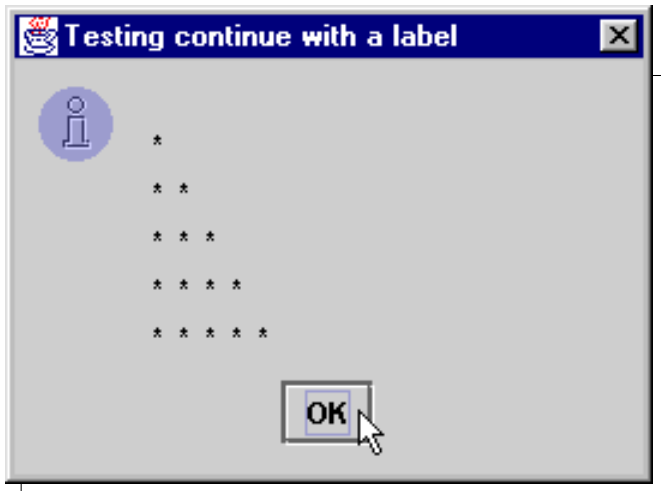


Program Output

```
1// ContinueLabelTest.java
2// Using the continue statement with a label
3import javax.swing.JOptionPane;
4
5public class ContinueLabelTest
6   public static void main( St
7   {
8      String output = "";
9
10      nextRow:   // target label of continue statement
11         for ( int row = 1; row <= 5; row++ ) {
12            output += "\n";
13
14            for ( int column = 1; column <= 10; column++ ) {
15
16               if ( column > row )
17                  continue nextRow; // next iteration of
18                                    // labeled loop
19
20               output += "*  ";
21            }
22         }
23
24      JOptionPane.showMessageDialog(
25         null, output,"Testing continue with a label",
26         JOptionPane.INFORMATION_MESSAGE );
27      System.exit( 0 );
28   }
29}
```

This label applies to the following **`for`** loop (labeled repetition structure).

Labeled **`continue`** statement skips remaining statements, goes to next iteration of labeled repetition structure.



Program Output

Return

```
// Demonstrate return.
class Return {
  public static void main(String args[]) {
    boolean t = true;
    System.out.println("Before the return.");
    if(t) return; // return to caller
    System.out.println("This won't execute.");
  }
}
```

The if(t) statement is necessary. Without if Java compiler would flag an “unreachable code” error, because the compiler would never be executed. To prevent this error, the if statement is used here to trick the compiler for the sake of this demonstration.

End of Chapter 5

Questions?