Lecture 3: Control Statements (Part 1)
Object Oriented Concepts and Programming (CSC244)

By
Dr. Tabbasum Naz
tabbasum.naz@ciitlahore.edu.pk

Displaying Text in a Dialog Box

- Display
  - Most Java applications use windows or a dialog box
  - Class **`JOptionPane`** allows us to use dialog boxes
- Packages
  - Set of predefined classes for us to use
  - Groups of related classes called *packages*
    - Group of all packages known as Java class library or Java applications programming interface (Java API)
  - **`JOptionPane`** is in the **`javax.swing`** package
    - Package has classes for using Graphical User Interfaces (GUIs)

Displaying Text in a Dialog Box (contd)

- Upcoming program
  - Application that uses dialog boxes
  - Explanation will come afterwards

```java
// Fig. 2.6: Welcome4.java
// Printing multiple lines in a dialog box

// Java extension packages
import javax.swing.JOptionPane;  // import class JOptionPane

public class Welcome4 {

   // main method begins execution of Java application
   public static void main( String args[] )
   {
      JOptionPane.showMessageDialog(
         null, "Welcome\nto\nJava\nProgramming!" );

      System.exit( 0 );  // terminate application

   }  // end method main

}  // end class Welcome4
```

Outline

`Welcome4.java`

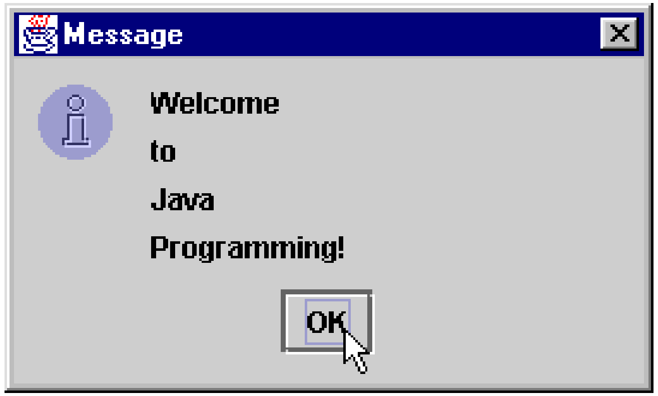
1. `import` statement

2. Class `Welcome4`

2.1 `main`

2.2 `showMessageDialog`

2.3 `System.exit`



```
Welcome
to
Java
Programming!
```

Program Output

Displaying Text in a Dialog Box (contd)

- Lines 1-2: comments as before

```
4     // Java extension packages
```

- Two groups of packages in Java API
- Core packages
  - Begin with **`java`**
  - Included with Java 2 Software Development Kit
- Extension packages
  - Begin with **`javax`**
  - New Java packages

```
5     import javax.swing.JOptionPane;
```

- **`import`** statements
  - Used by compiler to identify and locate classes used in Java programs
  - Tells compiler to load class **`JOptionPane`** from **`javax.swing`** package

Displaying Text in a Dialog Box (contd)

- Lines 6-11: Blank line, begin class **`Welcome4`** and **`main`**

```
12          JOptionPane.showMessageDialog(
13             null, "Welcome\nto\nJava\nProgramming!" );
```

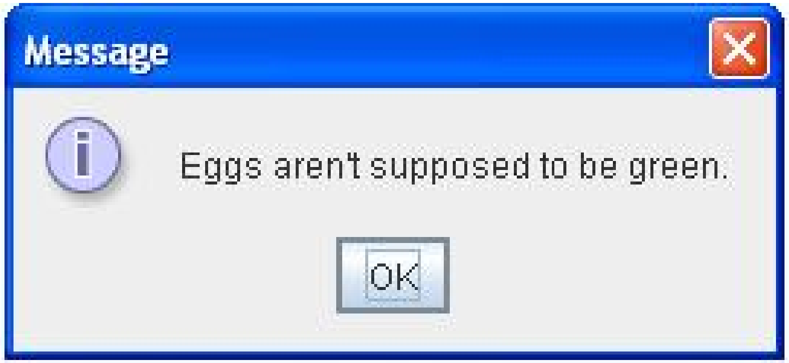
- Call method **`showMessageDialog`** of class **`JOptionPane`**
  - Requires two arguments
  - Multiple arguments separated by commas (**`,`**)
  - For now, first argument always **`null`**
  - Second argument is string to display
- **`showMessageDialog`** is a **`static`** method of class **`JOptionPane`**
  - **`static`** methods called using class name, dot (**`.`**) then method name

Displaying Text in a Dialog Box (contd)

```
15          System.exit( 0 );
```

- Calls **`static`** method **`exit`** of class **`System`**
  - Terminates application
    - Use with any application displaying a GUI
  - Because method is **`static`**, needs class name and dot (**`.`**)
  - Identifiers starting with capital letters usually class names
- Argument of **`0`** means application ended successfully
  - Non-zero usually means an error occurred
- Class **`System`** part of package **`java.lang`**
  - No **`import`** statement needed
  - **`java.lang`** automatically imported in every Java program
- Lines 17-19: Braces to end **`Welcome4`** and **`main`**

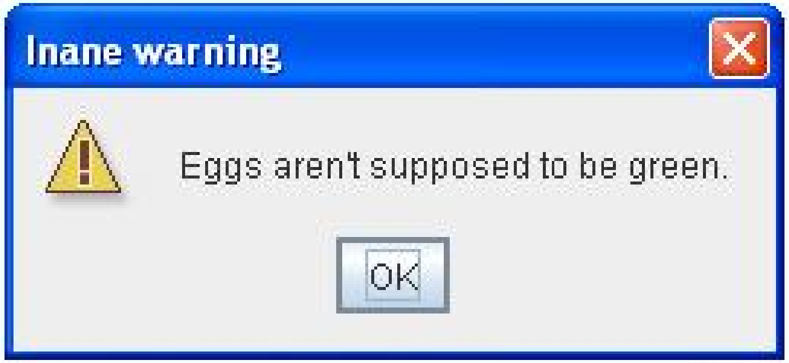
Displaying Text in a Dialog Box (contd)



Message

Eggs aren't supposed to be green.

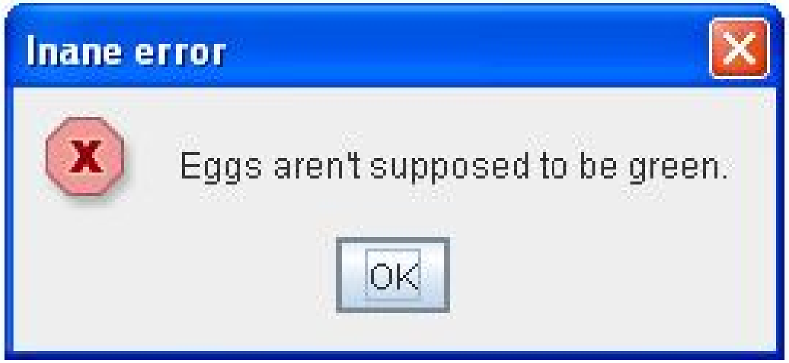
```
//default title and icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.");
```



Inane warning

Eggs aren't supposed to be green.

```
//custom title, warning icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);
```

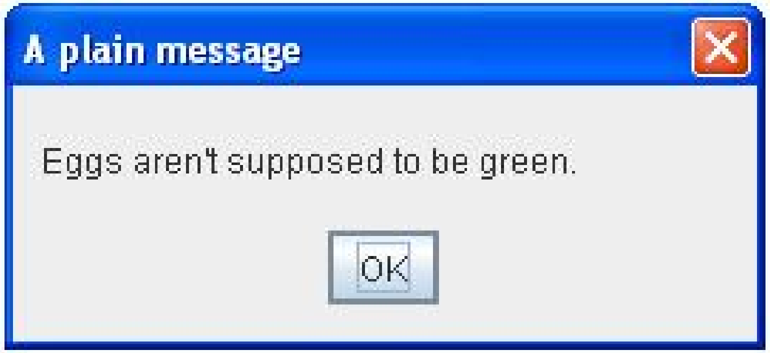


Inane error

Eggs aren't supposed to be green.

```
//custom title, error icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```

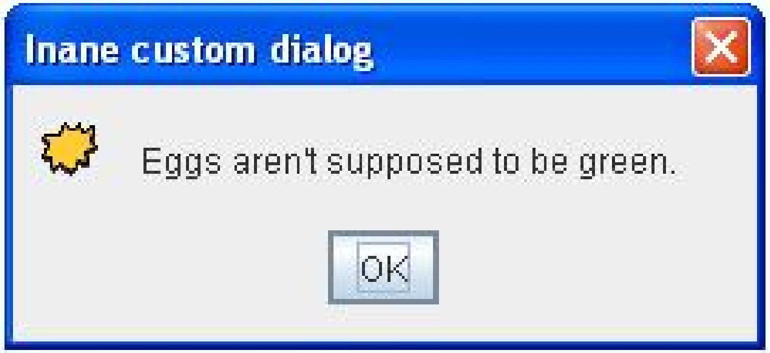
Displaying Text in a Dialog Box (contd)



A plain message

Eggs aren't supposed to be green.

```
//custom title, no icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "A plain message",
    JOptionPane.PLAIN_MESSAGE);
```



Inane custom dialog

Eggs aren't supposed to be green.

```
//custom title, custom icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane custom dialog",
    JOptionPane.INFORMATION_MESSAGE,
    icon);
```

Introduction to Control Structures

- We learn about Control Structures
  - Structured-programming principle
  - Control structures help build and manipulate objects

Algorithms

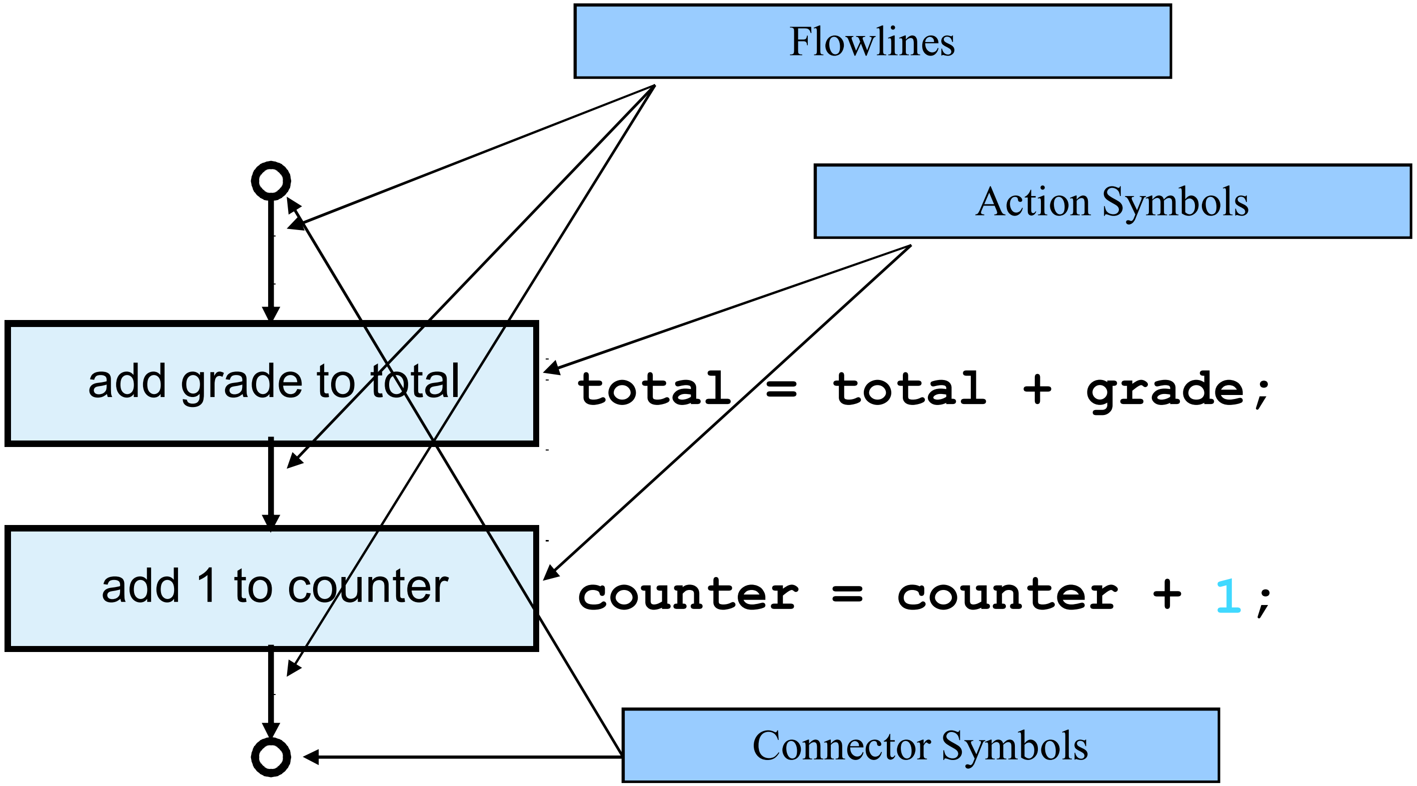
- Algorithm
  - Series of actions in specific order
    - The actions executed
    - The order in which actions execute
- Program control
  - Specifying the order in which actions execute
    - Control structures help specify this order

Pseudocode

- Pseudocode
  - Informal language for developing algorithms
  - Not executed on computers
  - Helps developers “think out” algorithms

Control Structures

- Sequential execution
  - Program statements execute one after the other
- Transfer of control
  - Three control statements can specify order of statements
    - Sequence structure
    - Selection structure
    - Repetition structure
- Flowchart
  - Graphical representation of algorithm
    - Flowlines indicate order in which actions execute



```
total = total + grade;

counter = counter + 1;
```

Fig 4.1 Flowcharting Java's sequence structure.

Selection Structures

- Java has a sequence structure “built-in”
- Java provides three selection structures
  - `if`
  - `if/else`
  - `switch`
- Java provides three repetition structures
  - `while`
  - `do/while`
  - `do`
- Each of these words is a Java keyword

The `if` Selection Structure

- Single-entry/single-exit structure
- Perform action only when condition is **`true`**
- Action/decision programming model

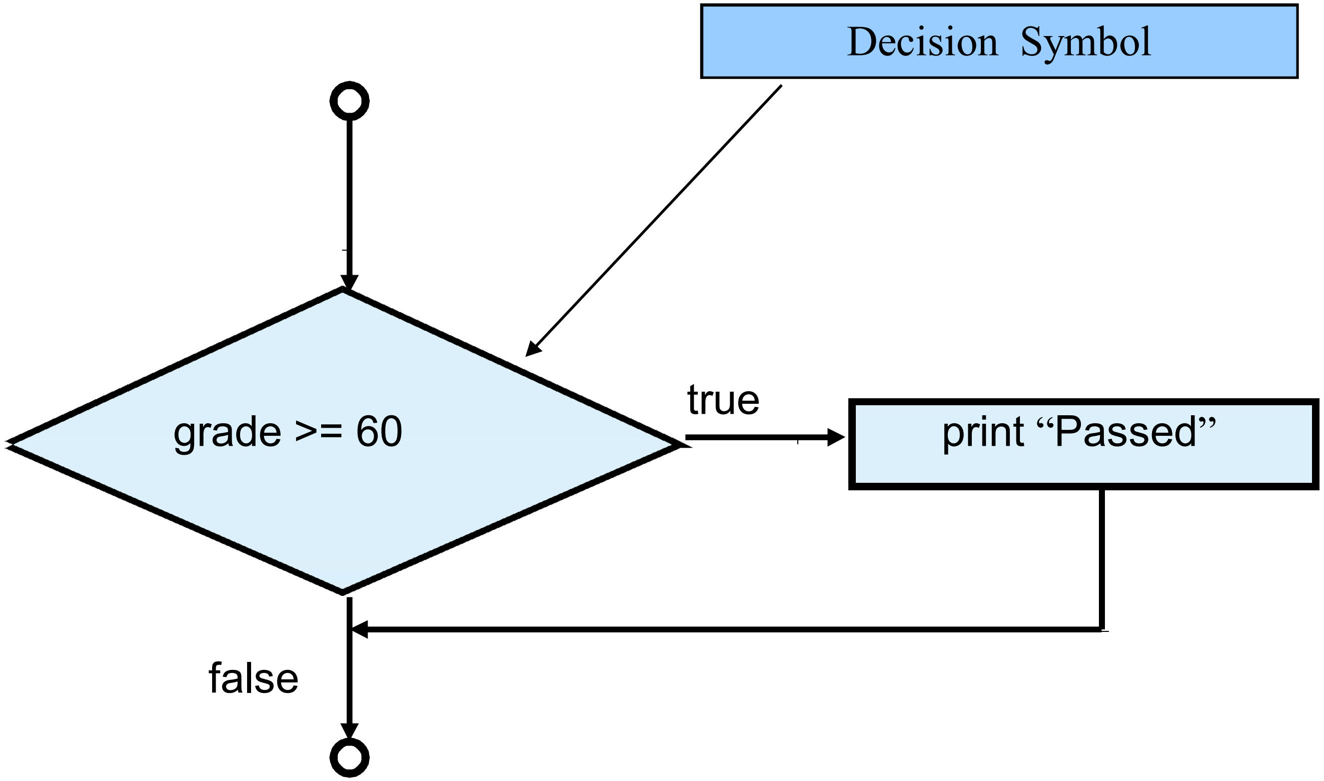
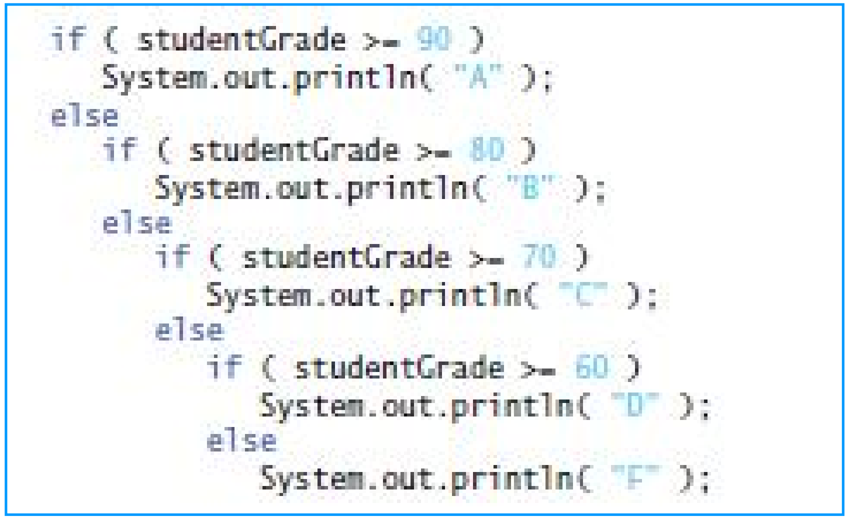


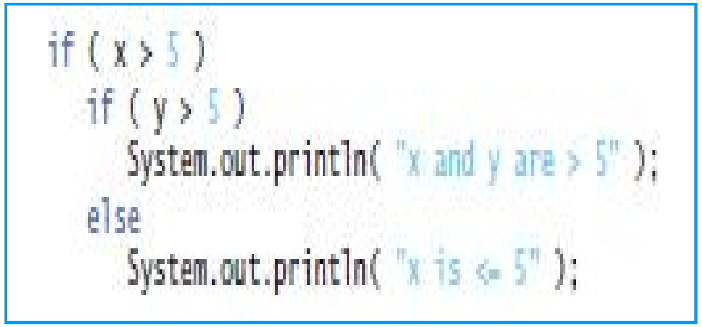
Fig 4.3 Flowcharting the single-selection `if` structure.

The if/else Selection Structure

- Perform action only when condition is **`true`**
- Perform different specified action when condition is **`false`**
- Nested **`if/else`** selection 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" );
```



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

- Conditional operator (**`?:`**)
  - ***`System.out.println(marks>50?″Pass″:″Fail″);`***

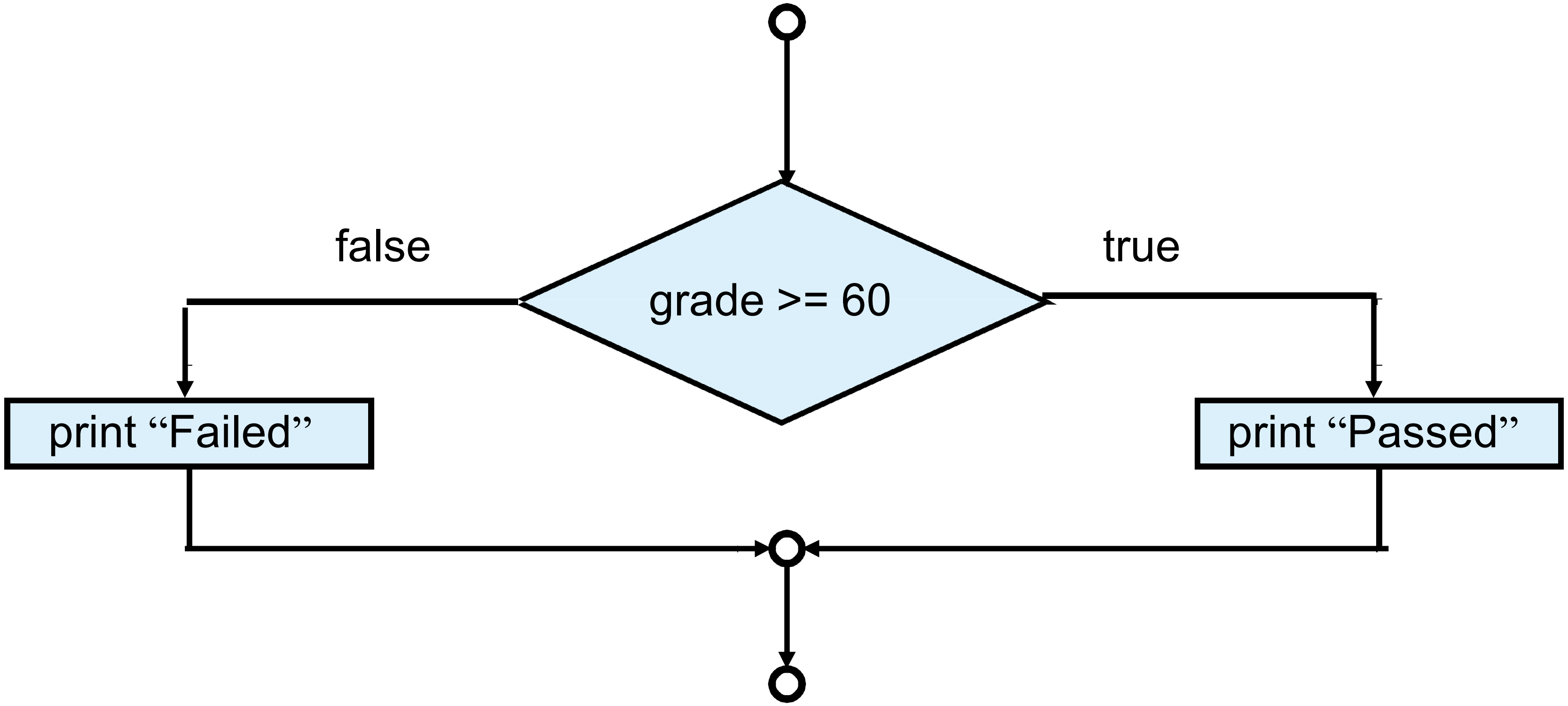


Fig 4.4 Flowcharting the double-selection `if/else` structure.

The while Repetition Structure

- Repeat action while condition remains **`true`**

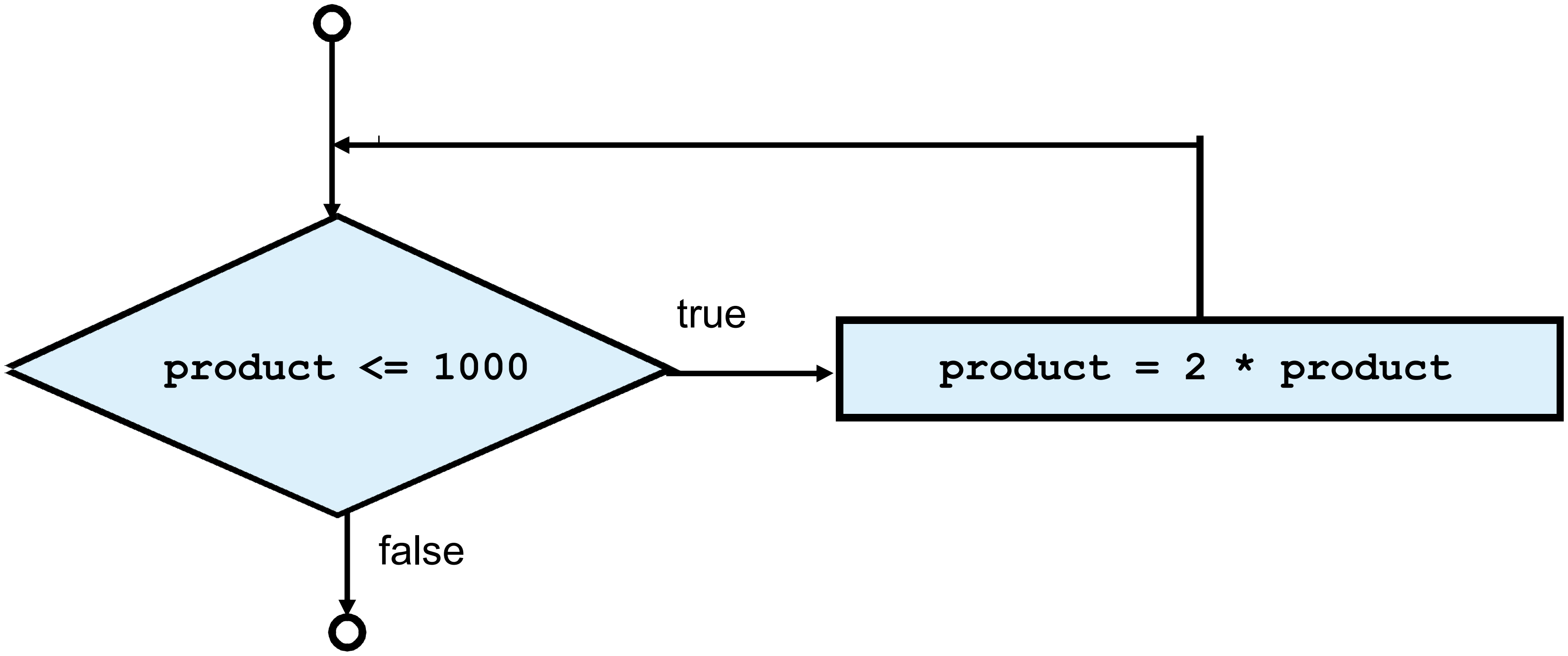


Fig 4.5 Flowcharting the `while` repetition structure.

Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)

- Counter
  - Variable that controls number of times set of statements executes
- **`Average1.java`** calculates grade averages
  - uses counters to control repetition

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
    Add one to the grade counter

Set the class average to the total divided by ten
Print the class average
```

Fig. 4.6 Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

```
1  // Fig. 4.7: Average1.java
2  // Class average program with counter-controlled repetition.
3
4  // Java extension packages
5  import javax.swing.JOptionPane;
6
7  public class Average1 {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int total,           // sum of grades input by user
13           gradeCounter,    // number o
14           gradeValue,      // grade va
15           average;         // average
16       String grade;        // grade ty
17
18       // Initialization Phase
19       total = 0;           // clear total
20       gradeCounter = 1;    // prepare to loop
21
22       // Processing Phase
23       while ( gradeCounter <= 10 ) {  // loop 10 times
24
25          // prompt for input and read grade from user
26          grade = JOptionPane.showInputDialog(
27             "Enter integer grade: " );
28
29          // convert grade from a String to an integer
30          gradeValue = Integer.parseInt( grade );
31
32          // add gradeValue to total
33          total = total + gradeValue;
34
```

Declare variables;
gradeCounter is the counter

Continue looping as long as **gradeCounter** is less than or equal to **10**

Average1.java

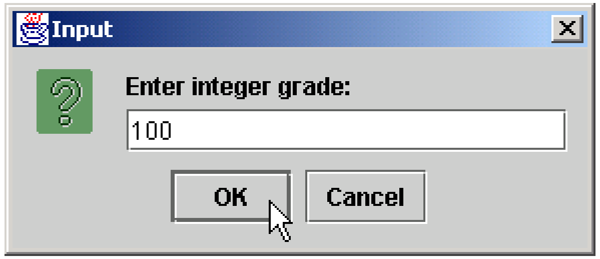
gradeCounter

Line 23

Outline

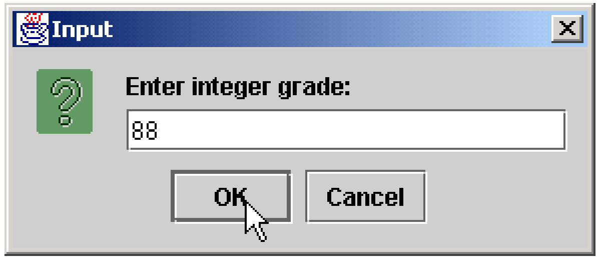
Average1.java

```
         // add 1 to gradeCounter
         gradeCounter = gradeCounter + 1;

      }  // end while structure

      // Termination Phase
      average = total / 10;  // perform integer division

      // display average of exam grades
      JOptionPane.showMessageDialog( null,
         "Class average is " + average, "Class Average",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );      // terminate the program

   }  // end method main

}  // end class Average1
```


Input

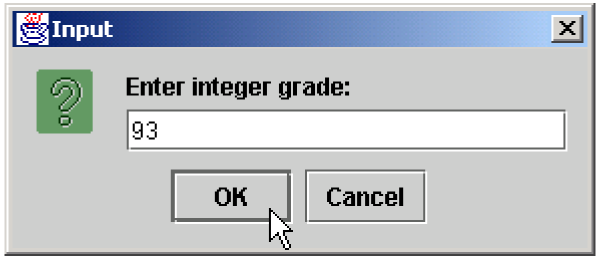
Enter integer grade:

100


Input

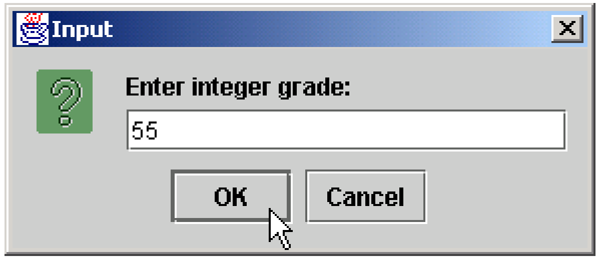
Enter integer grade:

88


Input

Enter integer grade:

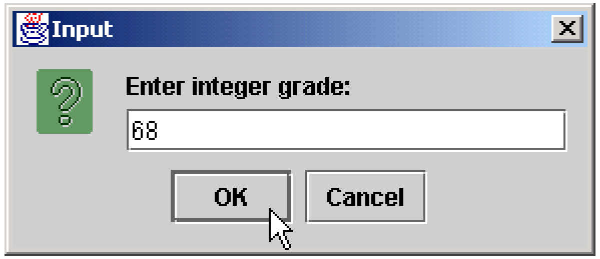
93


Input

Enter integer grade:

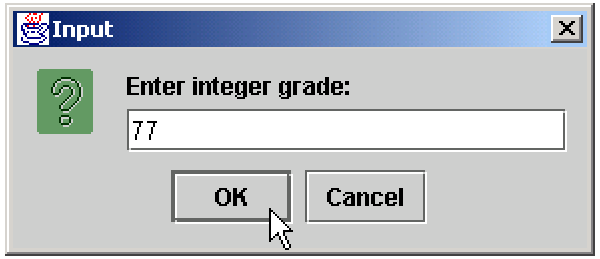
55

Average1.java



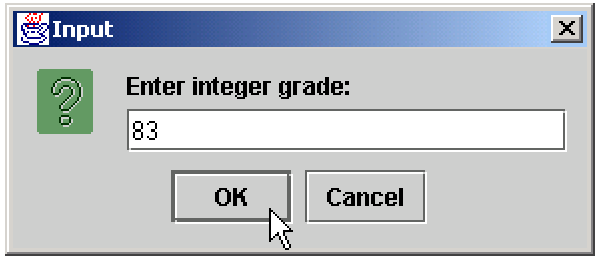
Enter integer grade:

68



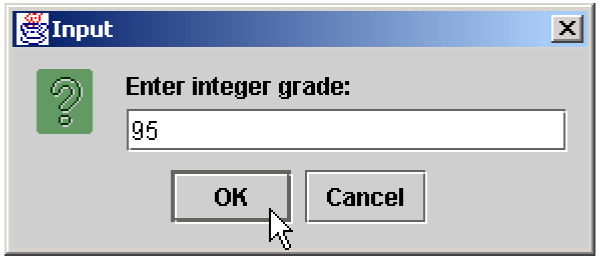
Enter integer grade:

77



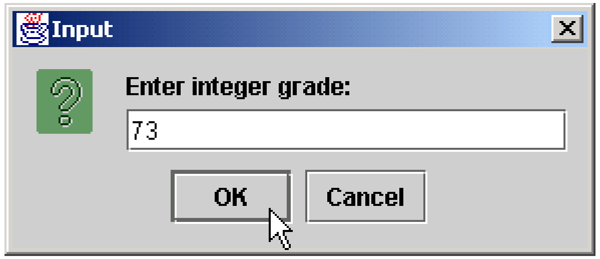
Enter integer grade:

83



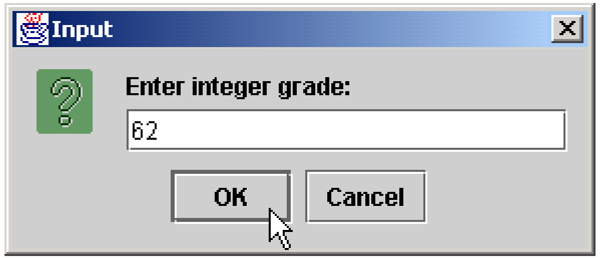
Enter integer grade:

95



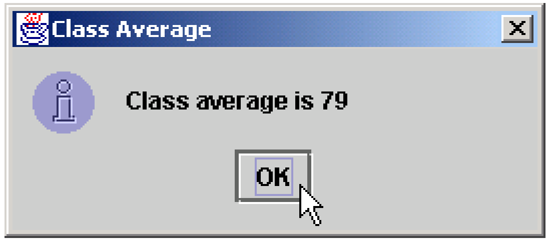
Enter integer grade:

73



Enter integer grade:

62



Class average is 79

Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study (Nested Control Structures)

- Nested control structures

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
    Input the next exam result

    If the student passed
        Add one to passes

    else
        Add one to failures

    Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
    Print “Raise tuition”
```

Fig 4.10 Pseudocode for examination-results problem.

Outline

Analysis.java

Line 21

Line 31

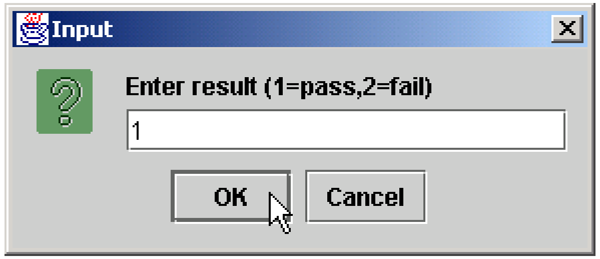
```
1  // Fig. 4.11: Analysis.java
2  // Analysis of examination results.
3
4  // Java extension packages
5  import javax.swing.JOptionPane;
6
7  public class Analysis {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       // initializing variables in declarations
13       int passes = 0,               // number of passes
14           failures = 0,             // number of failures
15           student = 1,              // student counter
16           result;                   // one exam result
17       String input,                 // user-entered value
18              output;                // output string
19
20       // process 10 students; counter-controlled loop
21       while ( student <= 10 ) {
22
23          // obtain result from user
24          input = JOptionPane.showInputDialog(
25             "Enter result (1=pass,2=fail)" );
26
27          // convert result to int
28          result = Integer.parseInt( input );
29
30          // process result
31          if ( result == 1 )
32             passes = passes + 1;
33          else
34             failures = failures + 1;
```

Loop until **student** counter is greater than **10**

Nested control structure

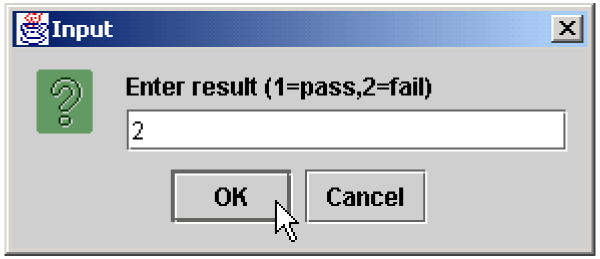
Analysis.java

```java

         student = student + 1;
      }

      // termination phase
      output = "Passed: " + passes +
         "\nFailed: " + failures;

      if ( passes > 8 )
         output = output + "\nRaise Tuition";

      JOptionPane.showMessageDialog( null, output,
         "Analysis of Examination Results",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );  // terminate application

   }  // end method main

}  // end class Analysis
```



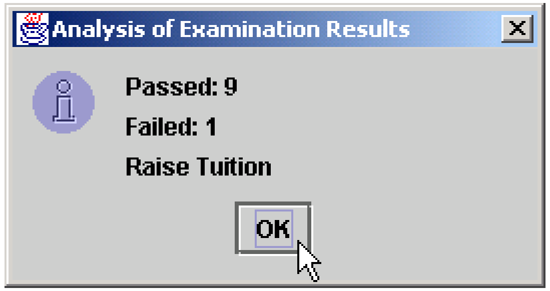
Enter result (1=pass,2=fail)

1



Enter result (1=pass,2=fail)

2



Passed: 9
Failed: 1
Raise Tuition

Assignment Operators

- Assignment Operators
  - Any statement of form
    - *variable* **=** *variable operator expression* **;**
  - Can be written as
    - *variable operator***=** *expression* **;**
  - e.g., addition assignment operator **`+=`**
    - **`c = c + 3`**
  - can be written as
    - **`c += 3`**

Assignment operator	Sample expression	Explanation	Assigns

Assume:
`int c = 3,`
`d = 5, e = 4,`
`f = 6, g = 12;`			

`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 4.12 Arithmetic assignment operators.

Increment and Decrement Operators

- Unary increment operator (**`++`**)
  - Increment variable’s value by **`1`**
- Unary decrement operator (**`--`**)
  - Decrement variable’s value by **`1`**
- Preincrement / predecrement operator
- Post-increment / post-decrement operator

Operator	Called	Sample expression	Explanation
++	preincrement	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postincrement	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	predecrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postdecrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 4.13 The increment and decrement operators.

Increment.java

Line 13 postincrement

Line 20 preincrement

```
1  // Fig. 4.14: Increment.java
2  // Preincrementing and postincrementing
3
4  public class Increment {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        int c;
10
11        c = 5;
12        System.out.println( c );    // print 5
13        System.out.println( c++ ); // print 5 then postincrement
14        System.out.println( c );    // print 6
15
16        System.out.println();       // skip a line
17
18        c = 5;
19        System.out.println( c );    // print 5
20        System.out.println( ++c ); // preincrement then print 6
21        System.out.println( c );    // print 6
22
23     }  // end method main
24
25 }  // end class Increment
```

Line 13 postincrements **c**

Line 20 preincrements **c**

```
5
5
6

5
6
6
```

THANK YOU