4

Control Statements: Part 1

4.2 Algorithms

- **Algorithms**
  - **The actions to execute**
  - **The order in which these actions execute**
- **Program control**
  - **Specifies the order in which actions execute in a program**

4.3 Pseudocode

- **Pseudocode**
  - **An informal language similar to English**
  - **Helps programmers develop algorithms**
  - **Does not run on computers**
  - **Should contain input, output and calculation actions**
  - **Should not contain variable declarations**

4.4 Control Structures

- **Sequential execution**
  - **Statements are normally executed one after the other in the order in which they are written**
- **Transfer of control**
  - **Specifying the next statement to execute that is not necessarily the next one in order**
  - **Can be performed by the `goto` statement**
    - **Structured programming eliminated `goto` statements**

4.4 Control Structures – No GOTOs

- **Bohm and Jacopini's research**
  - **Demonstrated that `goto` statements were unnecessary**
  - **Demonstrated that all programs could be written with three control structures**
    - **The sequence structure,**
    - **The selection structure and**
    - **The repetition structure**

4.4 Control Structures (Cont.)

- **UML activity diagram (www.uml.org)**
  - **Models the workflow (or activity) of a part of a software system**
  - **Action-state symbols (rectangles with their sides replaced with outward-curving arcs)**
    - **represent action expressions specifying actions to perform**
  - **Diamonds**
    - **Decision symbols (explained in section 4.5)**
    - **Merge symbols (explained in section 4.7)**

4.4 Control Structures (Cont.)

- **Small circles**
  - **Solid circle represents the activity's initial state**
  - **Solid circle surrounded by a hollow circle represents the activity's final state**
- **Transition arrows**
  - **Indicate the order in which actions are performed**
- **Notes (rectangles with the upper-right corners folded over)**
  - **Explain the purposes of symbols (like comments in Java)**
  - **Are connected to the symbols they describe by dotted lines**

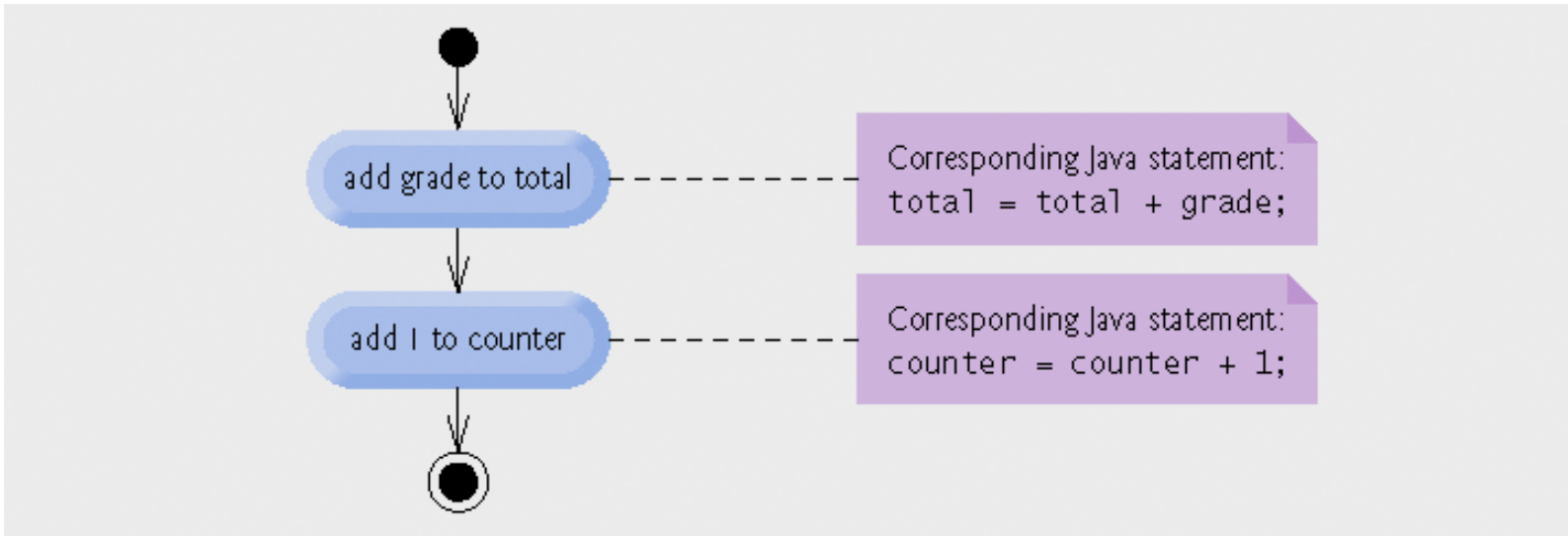


Fig. 4.1 | Sequence structure activity diagram.

4.4 Control Structures (Cont.)

- **Selection Statements**
  - **`if` statement**
    - **Single-selection statement**
  - **`if…else` statement**
    - **Double-selection statement**
  - **`switch` statement**
    - **Multiple-selection statement**

4.4 Control Structures (Cont.)

- **Repetition statements**
  - **Also known as looping statements**
  - **Repeatedly performs an action while its loop-continuation condition remains true**
  - **`while` statement**
    - **Performs the actions in its body zero or more times**
  - **`do…while` statement**
    - **Performs the actions in its body one or more times**
  - **`for` statement**
    - **Performs the actions in its body zero or more times**

4.4 Control Structures (Cont.)

- **Java has three kinds of control structures**
  - **Sequence statement,**
  - **Selection statements (three types) and**
  - **Repetition statements (three types)**
  - **All programs are composed of these control statements**
    - **Control-statement stacking**
      - **All control statements are single-entry/single-exit**
    - **Control-statement nesting**

4.5 `if` Single-Selection Statement

- **`if` statements**
  - **Execute an action if the specified condition is `true`**
  - **Can be represented by a decision symbol (diamond) in a UML activity diagram**
    - **Transition arrows out of a decision symbol have guard conditions**
      - **Workflow follows the transition arrow whose guard condition is true**

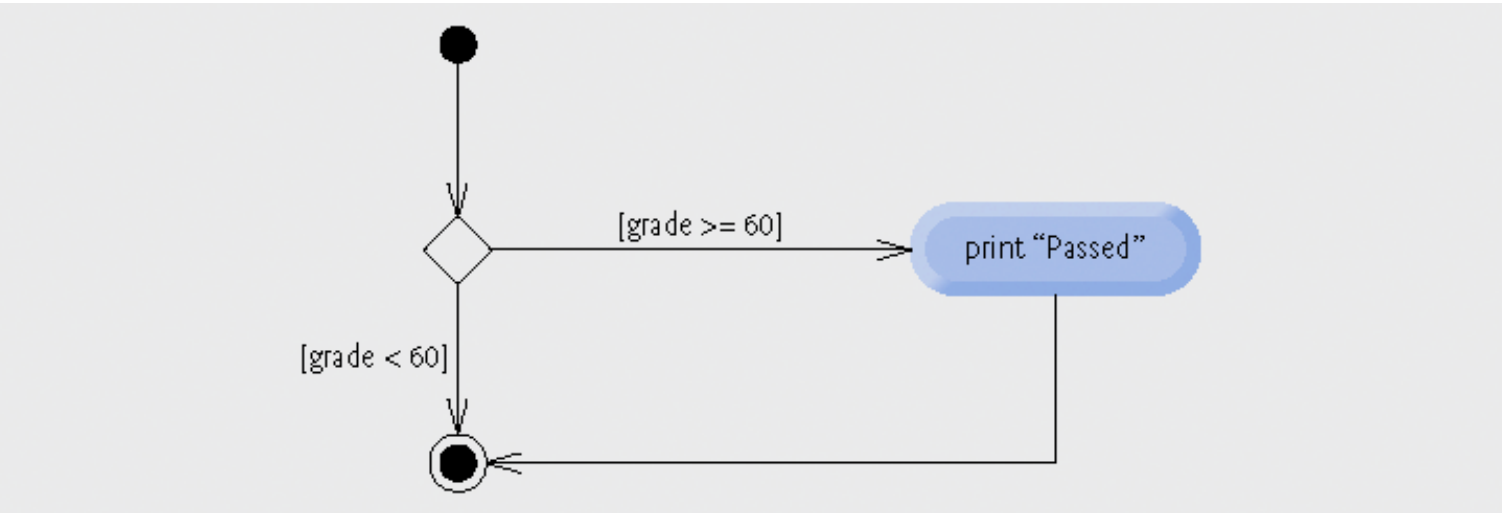


Fig. 4.2 | `if` single-selection statement UML activity diagram.

4.6 `if…else` Double-Selection Statement

- **`if…else` statement**
  - **Executes one action if the specified condition is `true` or a different action if the specified condition is `false`**
- **Conditional Operator (`?:`)**
  - **Java's only ternary operator (takes three operands)**
  - **`?:` and its three operands form a conditional expression**
    - **Entire conditional expression evaluates to the second operand if the first operand is `true`**
    - **Entire conditional expression evaluates to the third operand if the first operand is `false`**

4.6 `if…else` Double-Selection Statement

```
if (grade >= 60) {
  System.out.println(“Passed”);
} else {
  System.out.println(“Failed”);
}
```

same behavior as

```
System.out.println(grade>=60 ? “Passwd” : “Failed”)
```

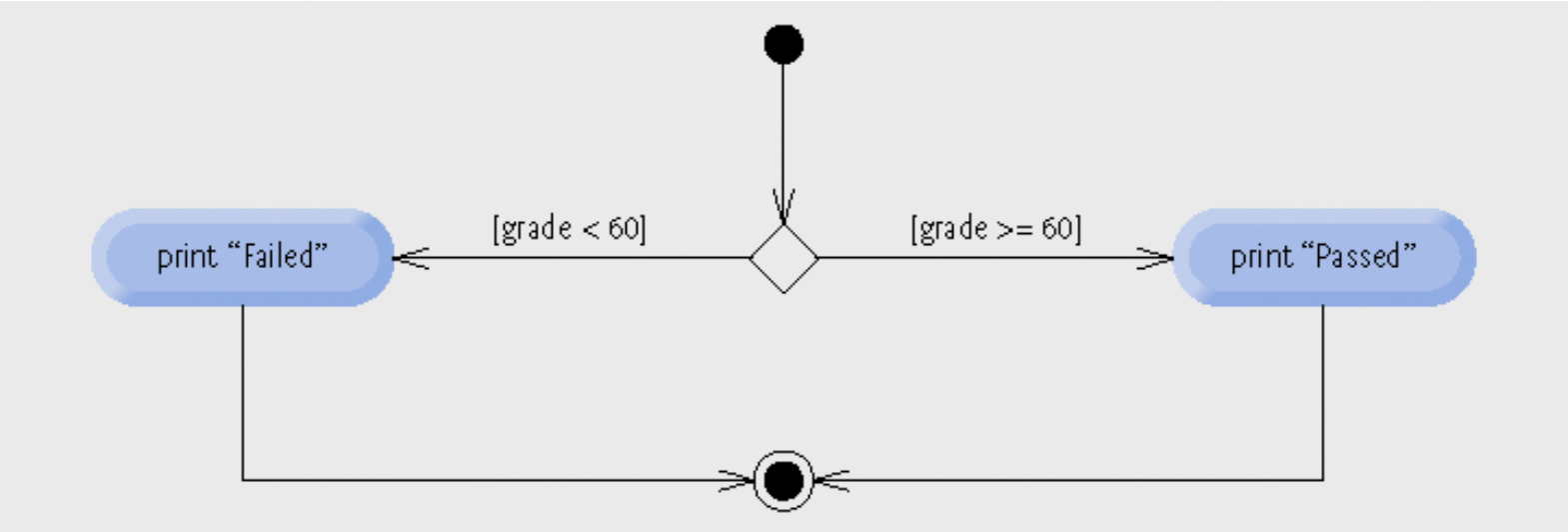


Fig. 4.3 | `if` else double-selection statement UML activity diagram.

4.7 `while` Repetition Statement

- **`while` statement**
  - **Repeats an action while its loop-continuation condition remains true**
  - **Uses a merge symbol in its UML activity diagram**
    - **Merges two or more workflows**
    - **Represented by a diamond (like decision symbols) but has:**
      - **Multiple incoming transition arrows,**
      - **Only one outgoing transition arrow and**
      - **No guard conditions on any transition arrows**

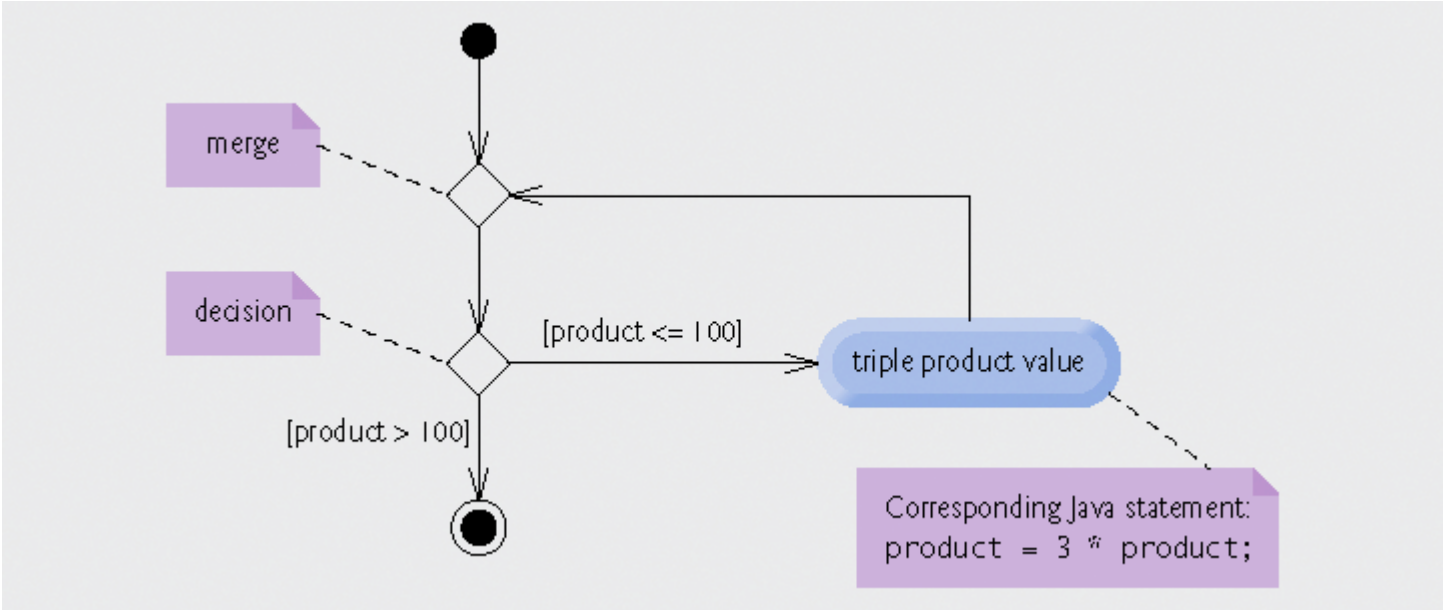


Fig. 4.4 | `while` repetition statement UML activity diagram.

4.8 Formulating Algorithms: Counter-Controlled Repetition

- **Counter-controlled repetition**
  - **Use a counter variable to count the number of times a loop is iterated**

4.9 Formulating Algorithms: Sentinel-Controlled Repetition

- **Sentinel-controlled repetition**
  - **Also known as indefinite repetition**
  - **Use a sentinel value (also known as a signal, dummy or flag value)**
    - **A sentinel value cannot also be a valid input value**

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- **Top-down, stepwise refinement**
  - **Top step: a single statement that conveys the overall function of the program**
  - **First refinement: multiple statements using only the sequence structure**
  - **Second refinement: commit to specific variables, use specific control structures**

4.10 Formulating Algorithms: Nested Control Statements

- **Control statements can be nested within one another**
  - **Place one control statement inside the body of the other**

```
1  Initialize passes to zero
2  Initialize failures to zero
3  Initialize student counter to one
4
5  While student counter is less than or equal to 10
6     Prompt the user to enter the next exam result
7     Input the next exam result
8
9     If the student passed
10       Add one to passes
11    Else
12       Add one to failures
13
14    Add one to student counter
15
16 Print the number of passes
17 Print the number of failures
18
19 If more than eight students passed
20    Print "Raise tuition"
```

Fig. 4.11 | Pseudocode for examination-results problem.

4.11 Compound Assignment Operators

- **Compound assignment operators**
  - **An assignment statement of the form:**
    ***variable* = *variable operator expression*;**
    where *operator* is +, -, *, / or % can be written as:
    ***variable operator*= *expression*;**
  - **example: `c = c + 3;` can be written as `c += 3;`**
    - **This statement adds `3` to the value in variable `c` and stores the result in variable `c`**

4.12 Increment and Decrement Operators

- **Unary increment and decrement operators**
  - **Unary increment operator (++) adds one to its operand**
  - **Unary decrement operator (- -) subtracts one from its operand**
  - **Prefix increment (and decrement) operator**
    - **Changes the value of its operand, then uses the new value of the operand in the expression in which the operation appears**
  - **Postfix increment (and decrement) operator**
    - **Uses the current value of its operand in the expression in which the operation appears, then changes the value of the operand**

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

Fig. 4.15 | Increment and decrement operators.

4.13 Primitive Types

- **Java is a strongly typed language**
  - **All variables have a type**
- **Primitive types in Java are portable across all platforms that support Java**

4.14 GUI and Graphics Case Study: Creating Simple Drawings

- **Java's coordinate system**
  - **Defined by x-coordinates and y-coordinates**
    - **Also known as horizontal and vertical coordinates**
    - **Are measured along the x-axis and y-axis**
  - **Coordinate units are measured in pixels**
- **`Graphics` class from the `java.awt` package**
  - **Provides methods for drawing text and shapes**
- **`JPanel` class from the `javax.swing` package**
  - **Provides an area on which to draw**

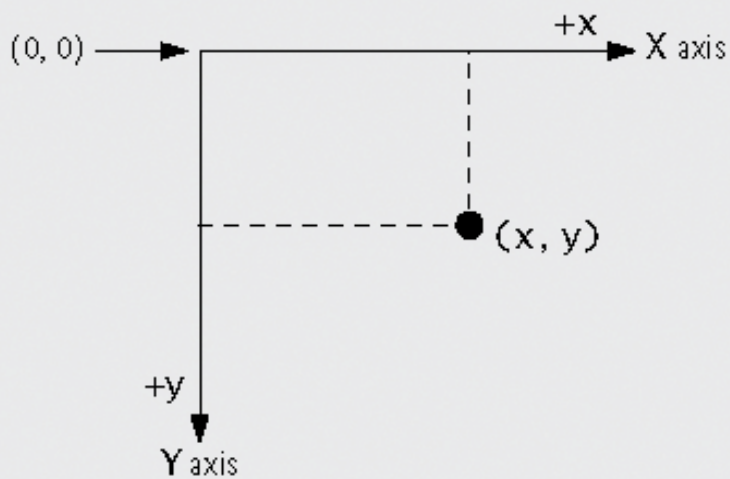


Fig. 4.18 | Java coordinate system. Units are measured in pixels.

4.14 GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- **Inheriting**
  - **`extends` keyword**
  - **The subclass inherits from the superclass**
    - **The subclass has the data and methods that the superclass has as well as any it defines for itself**

```
1  // Fig. 4.19: DrawPanel.java
2  // Draws two crossing lines on a panel.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class DrawPanel extends JPanel
7  {
8     // draws an X from the corners of the panel
9     public void paintComponent( Graphics g )
10    {
11       // call paintComponent to ensure the panel displays correctly
12       super.paintComponent( g );
13
14       int width = getWidth(); // total width
15       int height = getHeight(); // total height
16
17       // draw a line from the upper-left to the lower-right
18       g.drawLine( 0, 0, width, height );
19
20       // draw a line from the lower-left to the upper-right
21       g.drawLine( 0, height, width, 0 );
22    } // end method paintComponent
23 } // end class DrawPanel
```

Import the **`java.awt.Graphics`** and the **`javax.swing.JPanel`** classes

The **`DrawPanel`** class extends the **`JPanel`** class

•DrawPanel.java

Declare the **`paintComponent`** method

Retrieve the **`JPanel`**'s width and height

Draw the two lines

4.14 GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- The `JPanel` class
  - Every `JPanel` has a `paintComponent` method
    - `paintComponent` is called whenever the system needs to display the `JPanel`
  - `getWidth` and `getHeight` methods
    - Return the width and height of the `JPanel`, respectively
  - `drawLine` method
    - Draws a line from the coordinates defined by its first two arguments to the coordinates defined by its second two arguments

4.14 GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- **`JFrame` class from the `javax.swing` package**
  - **Allows the programmer to create a window**
  - **`setDefaultCloseOperation` method**
    - **Pass `JFrame.EXIT_ON_CLOSE` as its argument to set the application to terminate when the user closes the window**
  - **`add` method**
    - **Attaches a `JPanel` to the `JFrame`**
  - **`setSize` method**
    - **Sets the width (first argument) and height (second argument) of the `JFrame`**

```
1  // Fig. 4.20: DrawPanelTest.java
2  // Application to display a DrawPanel.
3  import javax.swing.JFrame;
4
5  public class DrawPanelTest
6  {
7     public static void main( String args[] )
8     {
9        // create a panel that contains our drawing
10       DrawPanel panel = new DrawPanel();
11
12       // create a new frame to hold the panel
13       JFrame application = new JFrame();
14
15       // set the frame to exit when it is closed
16       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18       application.add( panel );// add the panel to the frame
19       application.setSize( 250, 250 ); // set the size of the frame
20       application.setVisible( true );// make the frame visible
21    } // end main
22 } // end class DrawPanelTest
```

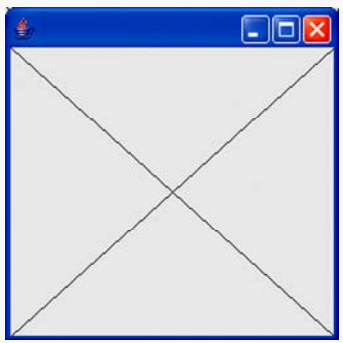
Import the **JFrame** class from the **javax.swing** class

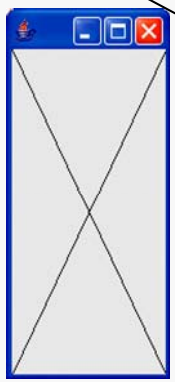
Create **DrawPanel** and **JFrame** objects

Set the application to terminate when the user closes the window

Add the **DrawPanel** to the **JFrame**

Set the size of and display the **JFrame**





•DrawPanel Test.java