# 4 - 5

# Control Statements: Overview

# Exam 1 – in about two weeks

- in-class, open book
- 5% of the total grade
- practice exams will be on the class website
- covers Chap. 1 – 6
- some theory, emphasis on coding practice

# Last time

- OOP and UML diagrams
- constructors
- garbage collection (GC)
- packages

# Objectives

- overview of control statements in Java
- if and if-else statements
- while, do-while, and for loop
- GUI programming: simple graphics drawing

Outline



Outline



# Algorithms and Pseudocode

- **Algorithms**
  - **The actions to execute**
  - **The order in which these actions execute**

- **Pseudocode**
  - **An informal language similar to English**
  - **Helps programmers develop algorithms**
  - **Should contain input, output and calculation actions**

# 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**

# Control Structures (Cont.)

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

# 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**

# `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**

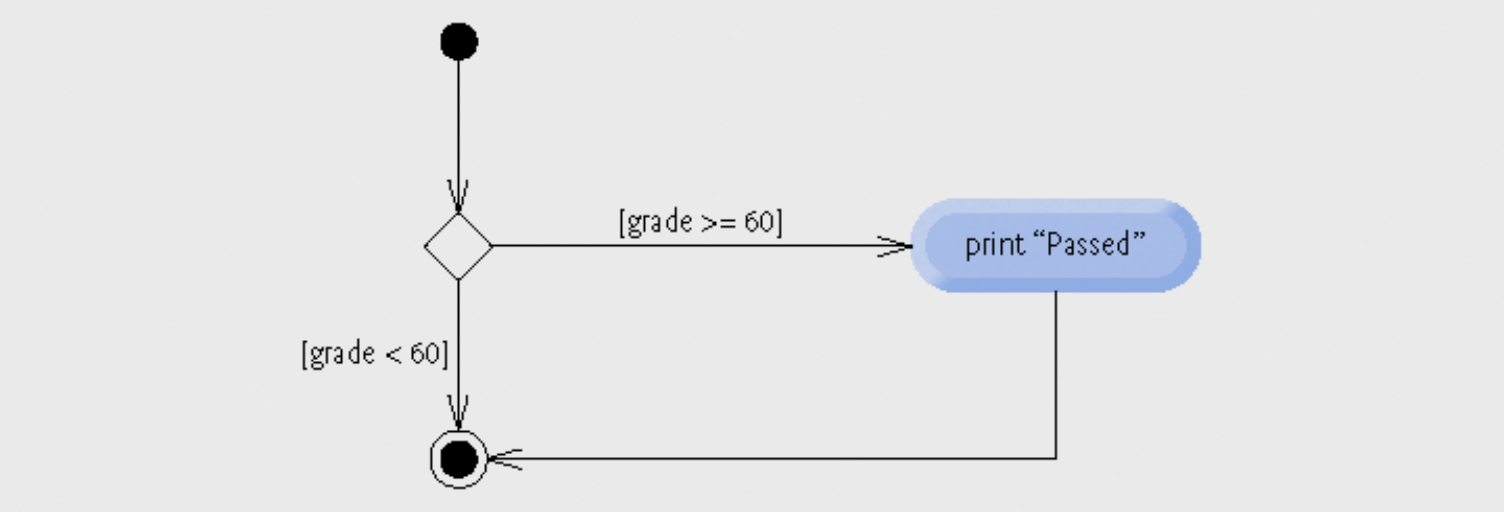



**Fig. 4.2 | `if` single-selection statement UML activity diagram.**

# `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`**

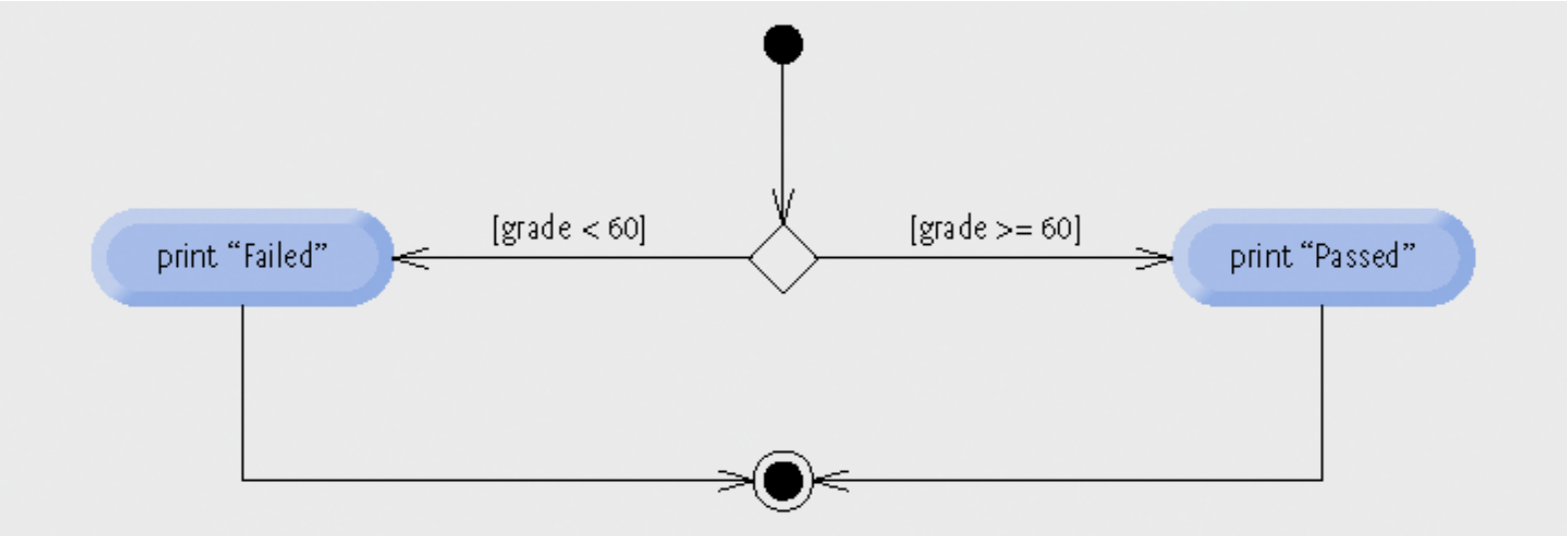



**Fig. 4.3 | `if`…`else` double-selection statement UML activity diagram.**

# `if…else` Double-Selection Statement

- Nested `if…else` statements
  - `if…else` statements can be put inside other `if…else` statements
- Dangling-`else` problem
  - `else`s are always associated with the immediately preceding `if` unless otherwise specified by braces `{ }`
- Blocks
  - Braces `{ }` associate statements into blocks
  - Blocks can replace individual statements as an `if` body

# 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)**

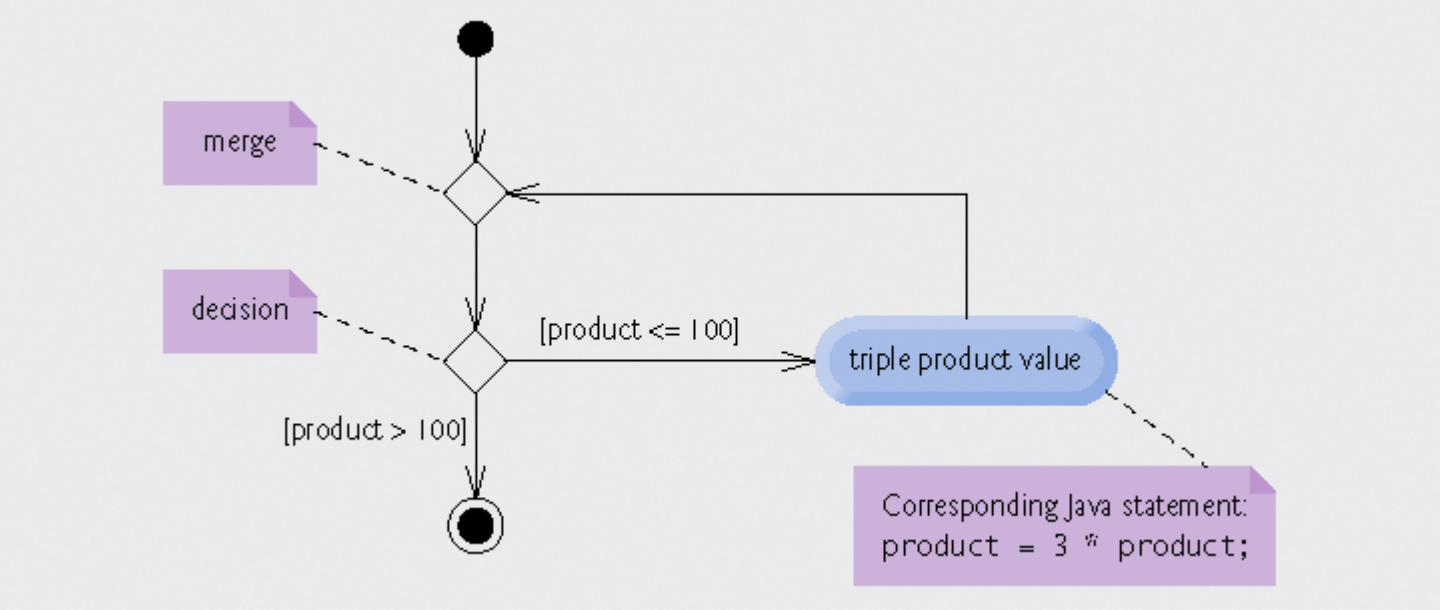



**Fig. 4.4 | `while` repetition statement UML activity diagram.**

# Program: Count digits

# do…while Repetition Statement

- do…while statement
  - Similar to while statement
  - Tests loop-continuation after performing body of loop
    - i.e., loop body always executes at least once

# for Repetition Statement

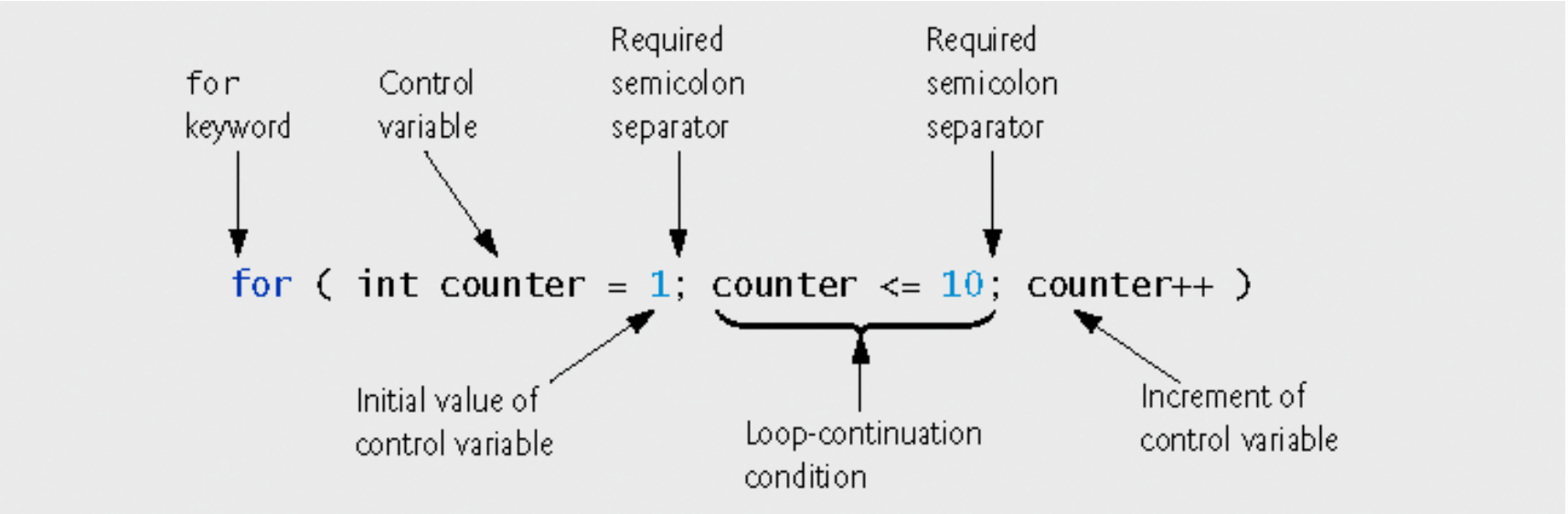



**Fig. 5.3 | for statement header components.**

# `for` Repetition Statement (Cont.)

```
for ( initialization; loopContinuationCondition; increment )
    statement;
```

can usually be rewritten as:

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

# Examples Using the for Statement

- **Varying control variable in `for` statement**
  - **Vary control variable from 1 to 100 in increments of 1**
    - `for ( int i = 1; i <= 100; i++ )`
  - **Vary control variable from 100 to 1 in increments of –1**
    - `for ( int i = 100; i >= 1; i-- )`
  - **Vary control variable from 7 to 77 in increments of 7**
    - `for ( int i = 7; i <= 77; i += 7 )`
  - **Vary control variable from 20 to 2 in decrements of 2**
    - `for ( int i = 20; i >= 2; i -= 2 )`

# Program: Factorial using for loop

# Logical Operators

- **Logical operators**
  - **Allows for forming more complex conditions**
  - **Combines simple conditions**
- **Java logical operators**
  - **&&  (conditional AND)**
  - **||  (conditional OR)**
  - **&  (boolean logical AND)**
  - **|  (boolean logical inclusive OR)**
  - **^  (boolean logical exclusive OR)**
  - **!  (logical NOT)**

# Logical Operators (Cont.)

- **Boolean Logical Exclusive OR (^)**
  - **One of its operands is `true` and the other is `false`**
    - **Evaluates to `true`**
  - **Both operands are `true` or both are `false`**
    - **Evaluates to `false`**
- **Logical Negation (!) Operator**
  - **Unary operator**

| 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. 4.14 | Arithmetic compound assignment operators.**

# 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.**

```
1  // Fig. 4.16: Increment.java
2  // Prefix increment and postfix increment operators.
3
4  public class Increment
5  {
6     public static void main( String args[] )
7     {
8        int c;
9
10       // demonstrate postfix increment operator
11       c = 5; // assign 5 to c
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       // demonstrate prefix increment operator
19       c = 5; // assign 5 to c
20       System.out.println( c );   // print 5
21       System.out.println( ++c ); // preincrement then print 6
22       System.out.println( c );   // print 6
23
24    } // end main
25
26 } // end class Increment
```

Postincrementing the `c` variable

Preincrementing the `c` variable

```
5
5
6

5
6
6
```

| Operators | | | | | | Associativity | Type |
|---|---|---|---|---|---|---|---|
| `++` | `--` | | | | | right to left | unary postfix |
| `++` | `--` | `+` | `-` | `(` *type* `)` | | right to left | unary prefix |
| `*` | `/` | `%` | | | | left to right | Multiplicative |
| `+` | `-` | | | | | left to right | Additive |
| `<` | `<=` | `>` | `>=` | | | left to right | Relational |
| `==` | `!=` | | | | | left to right | Equality |
| `?:` | | | | | | right to left | Conditional |
| `=` | `+=` | `-=` | `*=` | `/=` | `%=` | right to left | assignment |

**Fig. 4.17 | Precedence and associativity of the operators discussed so far.**

# 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

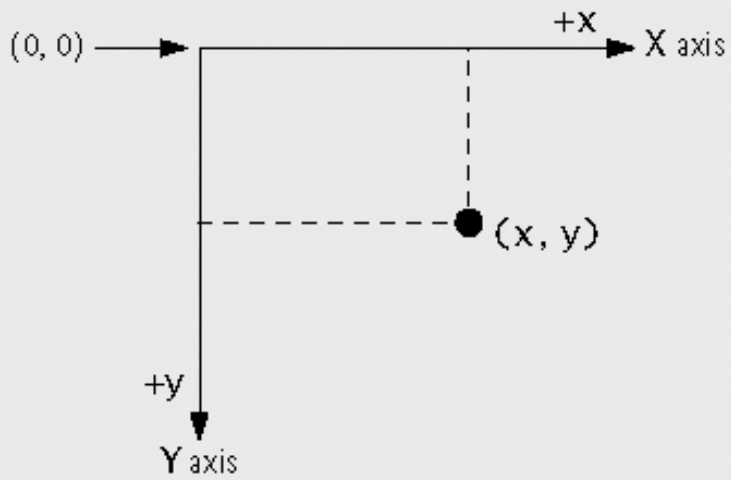



**Fig. 4.18 | Java coordinate system. Units are measured in pixels.**

# 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

# 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

# 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
```

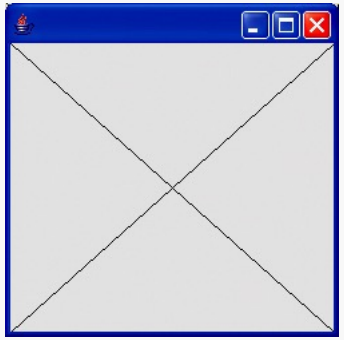
**DrawPanelTest.java**

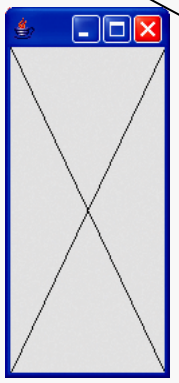
Import the **JFrame** class from the **javax.swing** package

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**