Presentation 01
Introduction to Java Programming

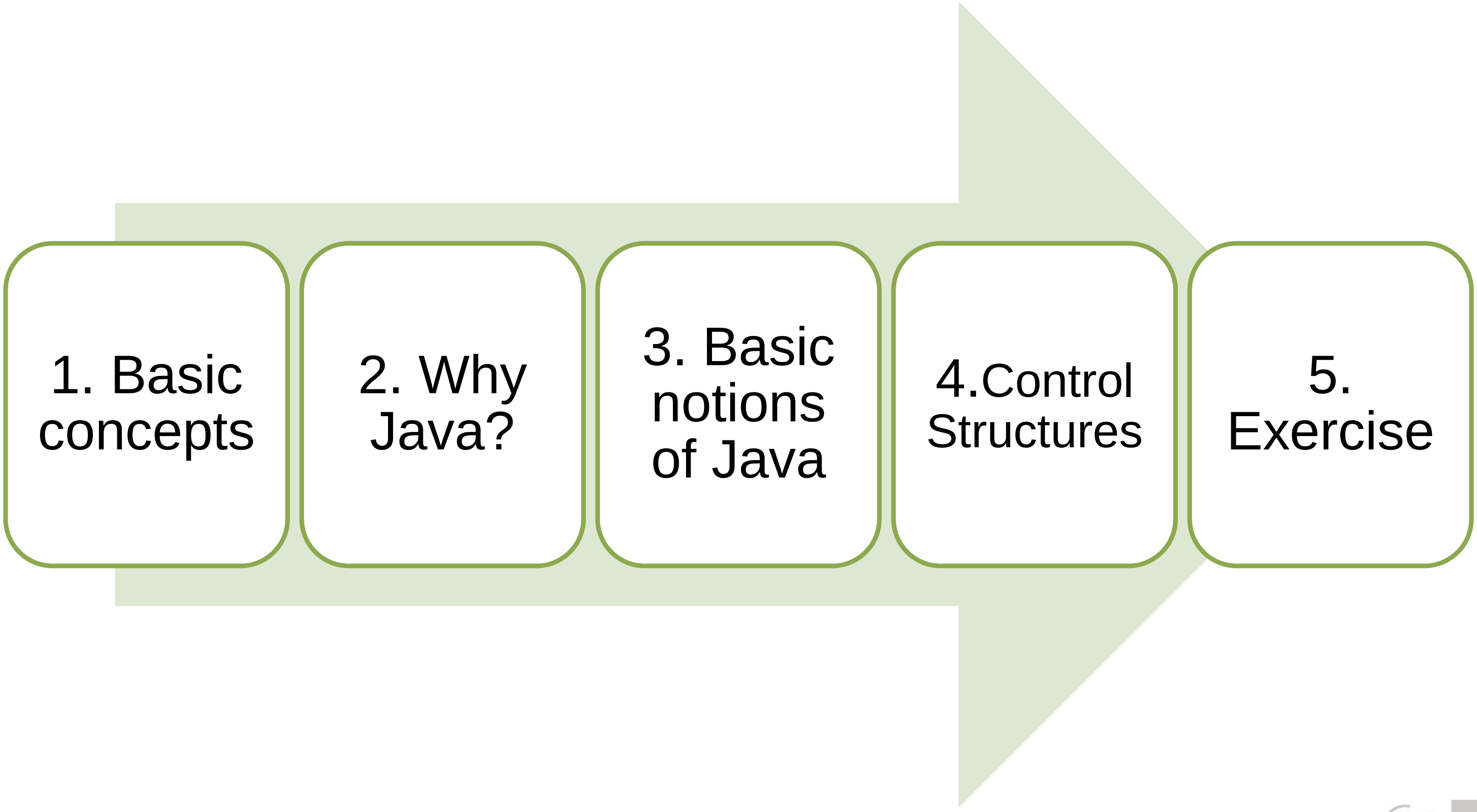
Christian Rodríguez Bustos

Edited by Juan Mendivelso

Object Oriented Programming

Basic programming review

Agenda



1. Basic concepts

2. Why Java?

3. Basic notions of Java

4.Control Structures

5. Exercise

1. Basic concepts

1.1 Algorithm

1.2 Pseudo code

1.3 Flow diagram

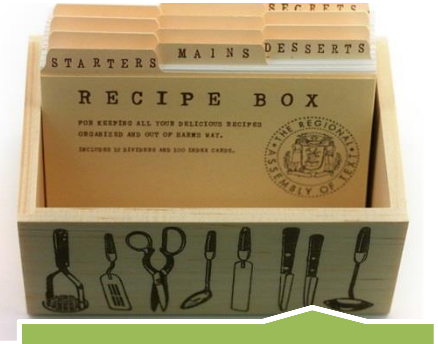
1.1 Algorithm

An **algorithm** is a step-by-step procedure

Any computing problem can be solved by executing a series of actions in a specific order.

Finite

Deterministic

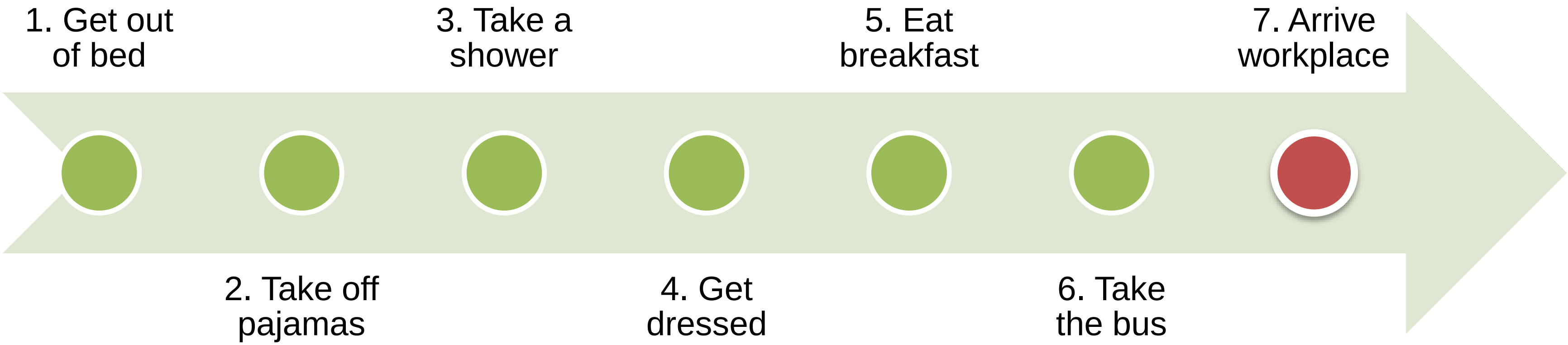


Precision

An **algorithm** is a step-by-step procedure

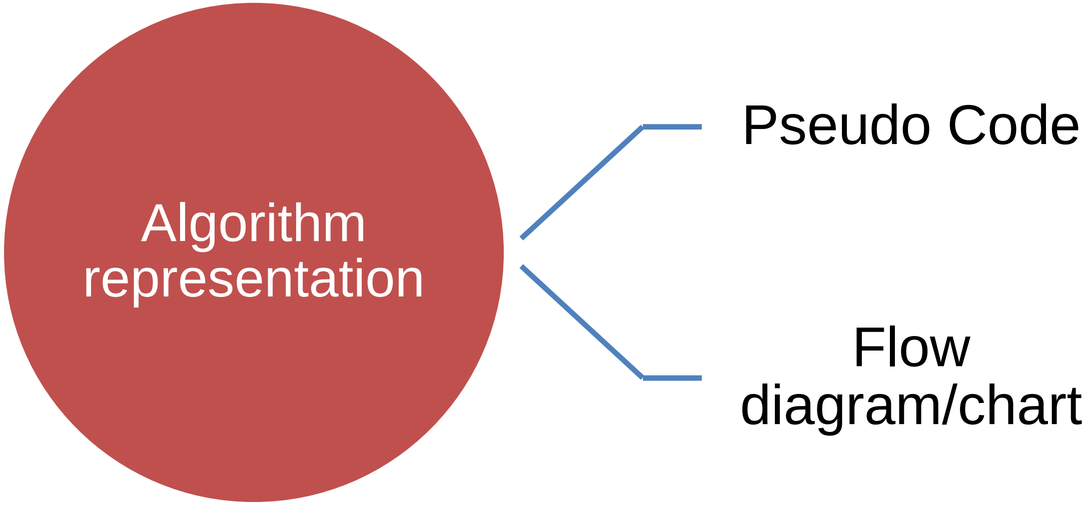
Problem that we want to solve

Going to work



1. Get out of bed
2. Take off pajamas
3. Take a shower
4. Get dressed
5. Eat breakfast
6. Take the bus
7. Arrive workplace

Algorithms can be represented in several ways



Algorithm representation

Pseudo Code

Flow diagram/chart

1.2 Pseudocode

Pseudocodes: Informal descriptions of algorithms

Informal descriptions or languages help programmers to develop algorithms **without** having to worry about the strict details of a programming language syntax.

All pseudo codes should be:

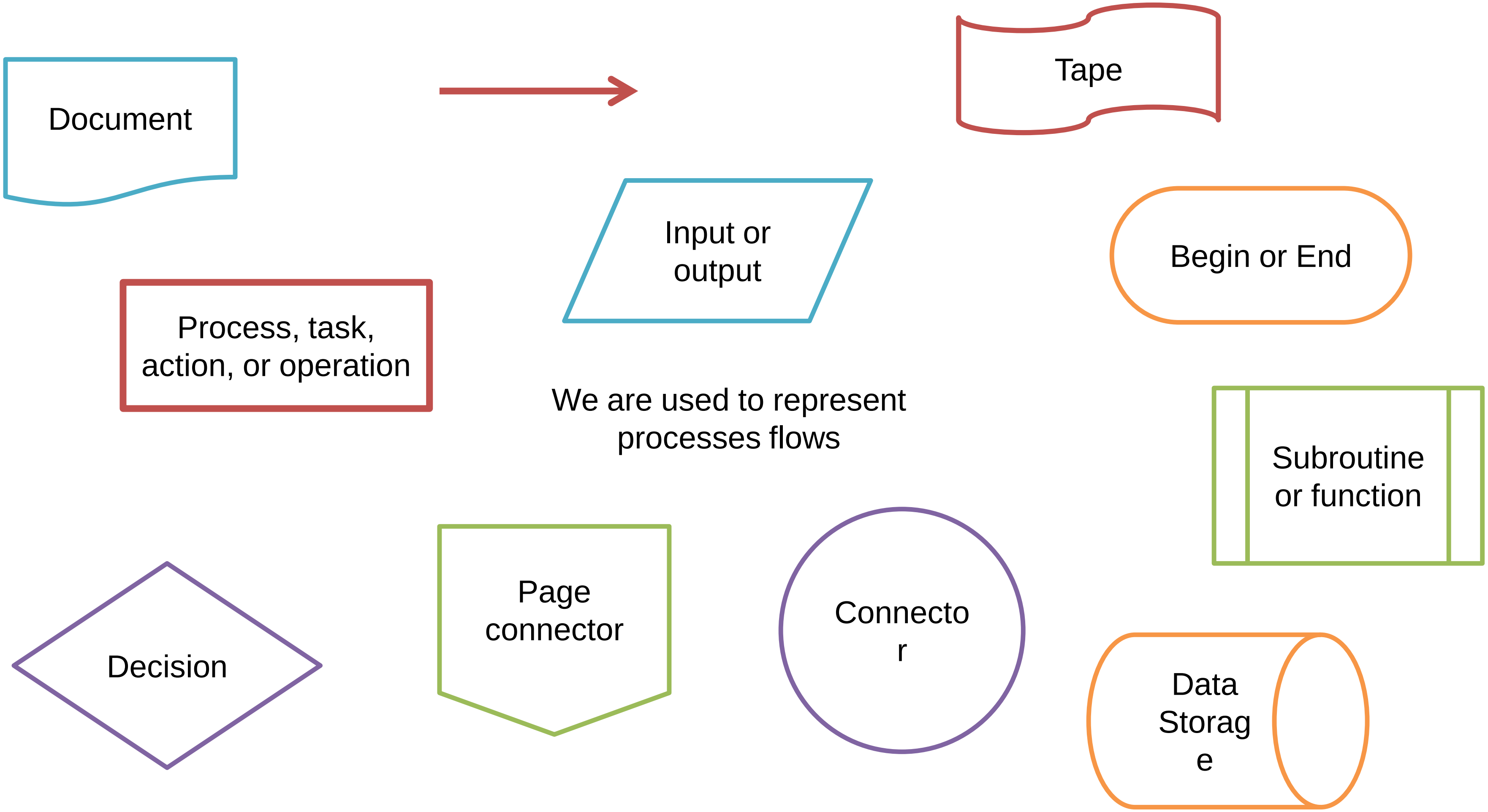
- Human readable
- Can easily be converted to any programming language

```
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

Set the class average to the total divided by ten

Print the class average.
```

I am a pseudo code

1.3 Flow Diagrams

Flow diagrams are used to represent algorithms



Document

Tape

Input or output

Begin or End

Process, task, action, or operation

We are used to represent processes flows

Subroutine or function

Decision

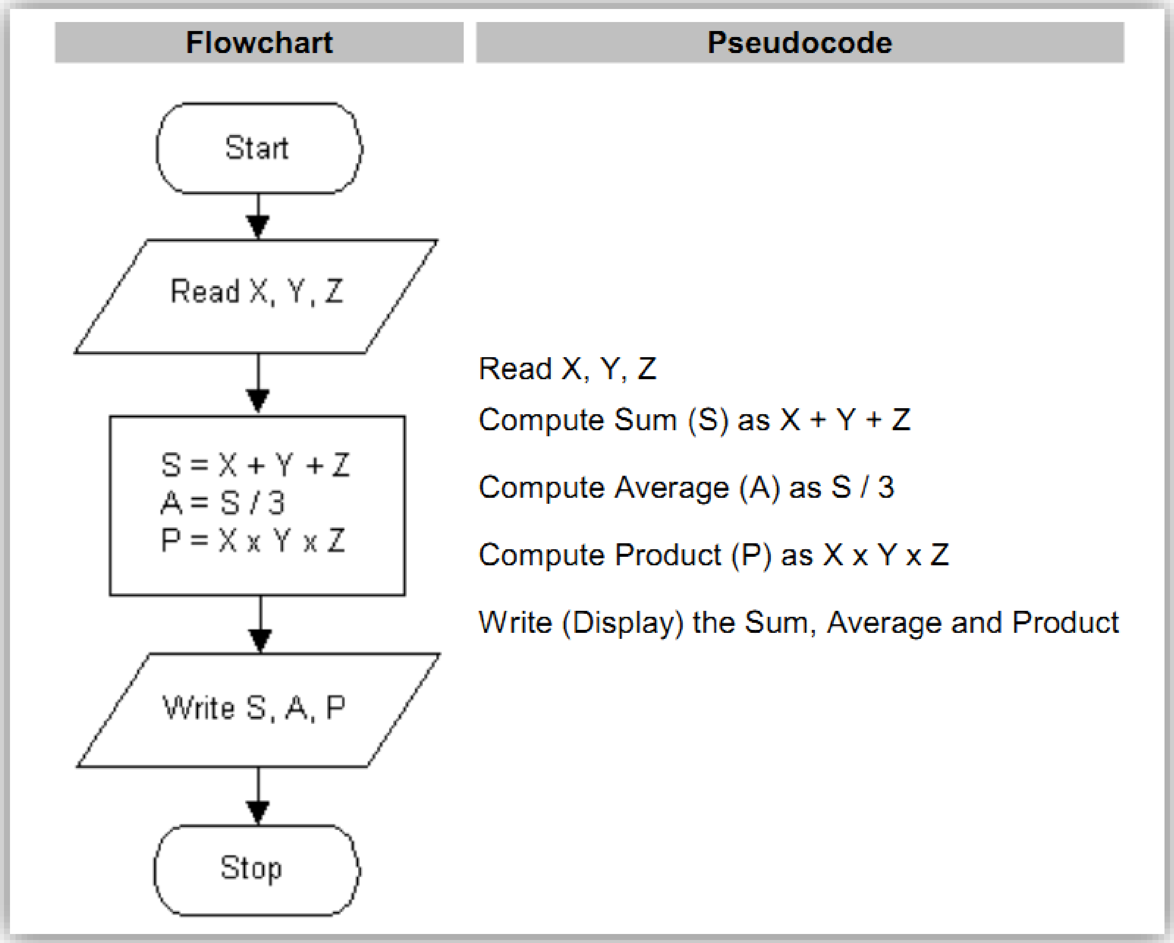
Page connector

Connecto r

Data Storag e

Resource: *What do the different flowchart shapes mean?*

Example



Pseudocode

```
Read X, Y, Z
Compute Sum (S) as X + Y + Z
Compute Average (A) as S / 3
Compute Product (P) as X x Y x Z
Write (Display) the Sum, Average and Product
```

2. Why Java?

1.1 Java Advantages

1.2 Java Acronyms

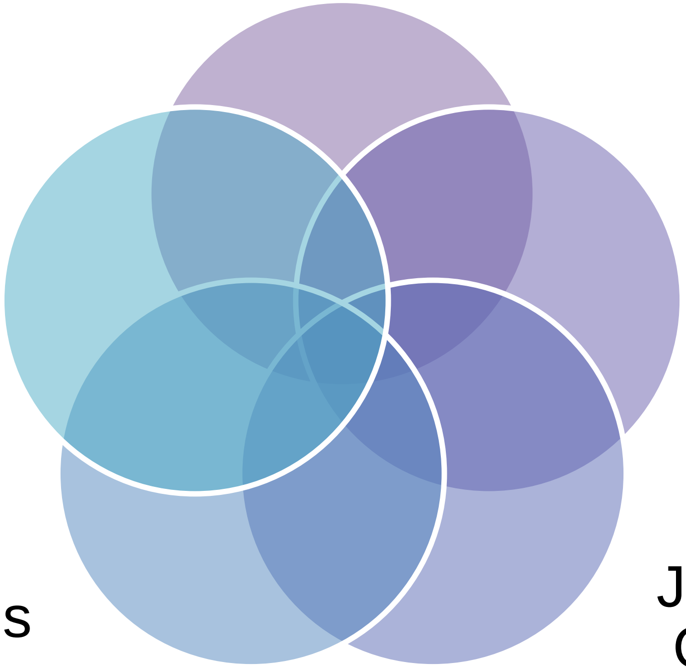
2.1 Java Advantages

Java advantages

Java Is Free!

Java Is Architecture Neutral

Java Provides “One-Stop Shopping”

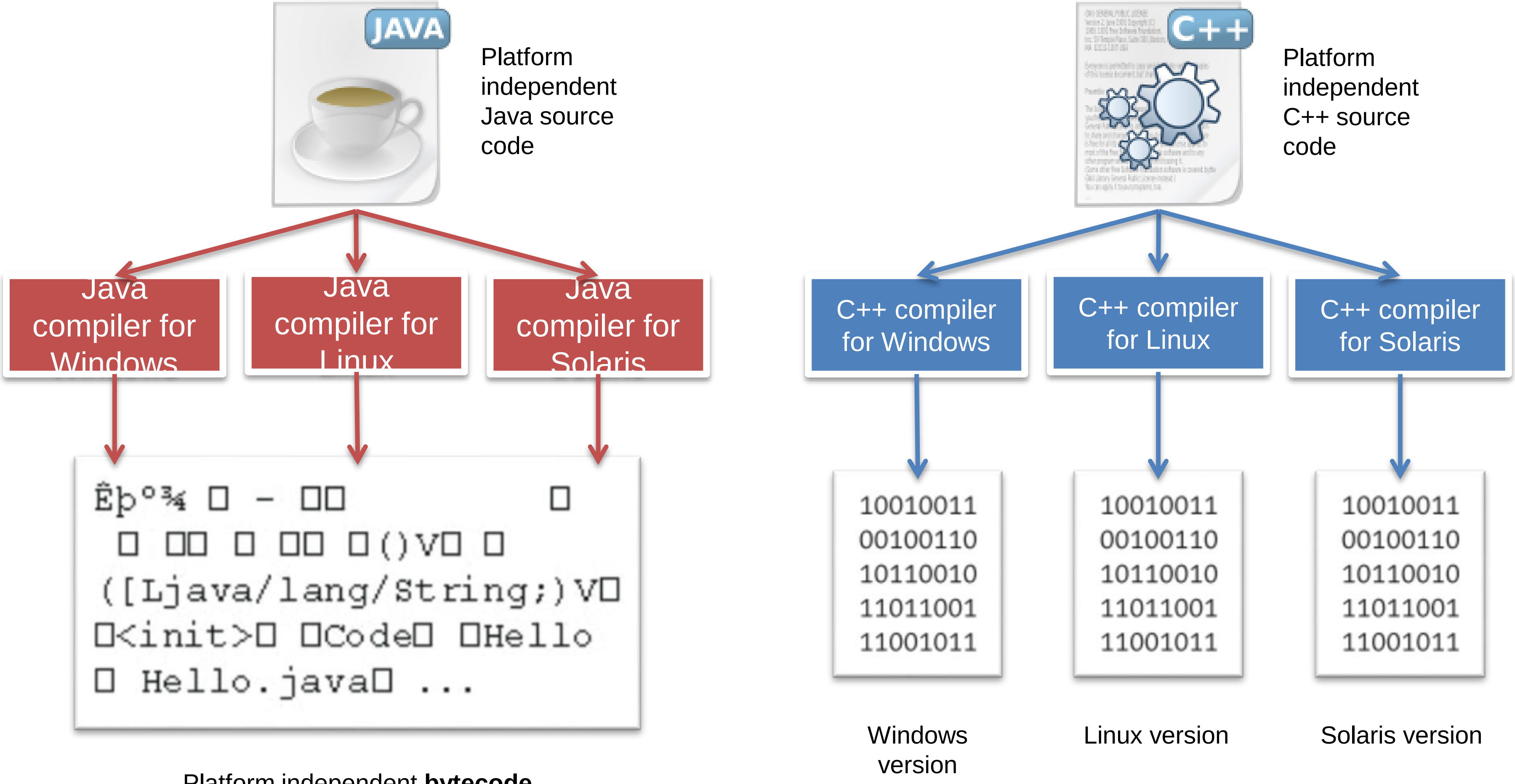


Practice Makes Perfect

Java Is Object-Oriented from the Ground Up

Java Is Free!

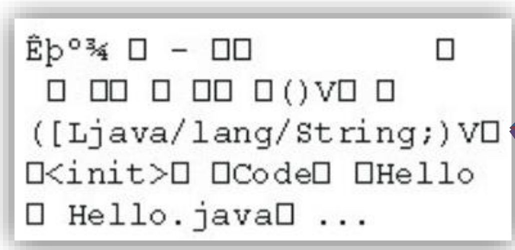
Java Is Architecture Neutral



Platform independent **bytecode**
(.class file)

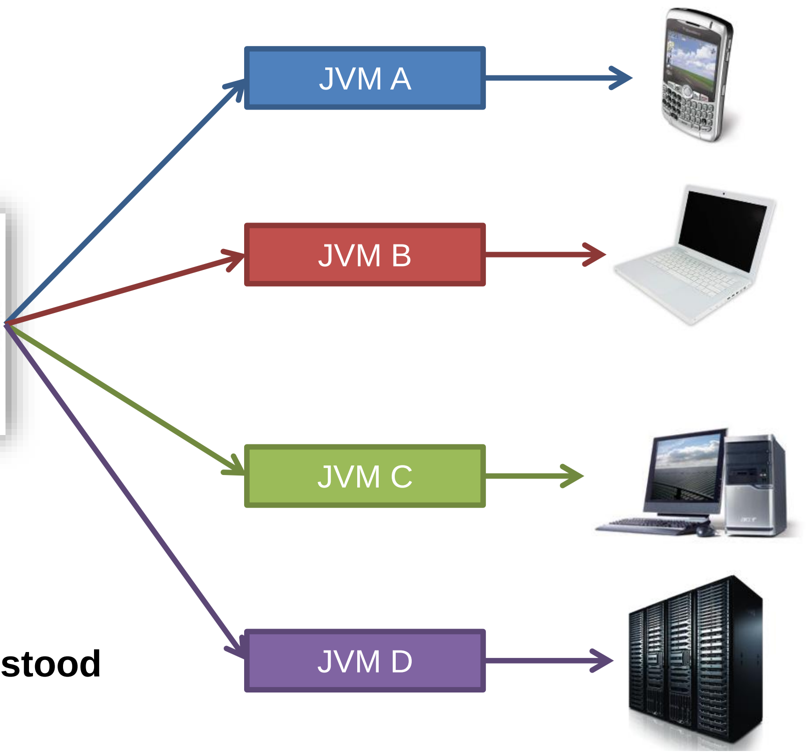
Java Is Architecture Neutral

*The **Java Virtual Machine** (JVM)*
converts the compiled Java byte code to machine code.



```
Êþº¾ □ - □□           □
 □ □□ □ □□ □()V□ □
([Ljava/lang/String;)V□
□<init>□ □Code□ □Hello
□ Hello.java□ ...
```

Platform independent **bytecode**
(.class file)

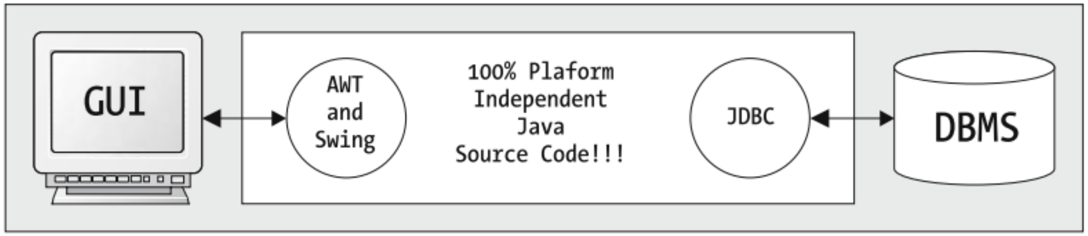


The same bytecode can be understood by different JVMs

In theory, bytecode is forward compatible with newer versions of the JVM

Java Provides “One-Stop Shopping”

Java language provides an extensive set of **application programming interfaces (APIs)**



java.io: Used for file system access

java.sql: The JDBC API, used for communicating with relational databases in a vendor-independent fashion

java.awt: The Abstract Windowing Toolkit, used for GUI development

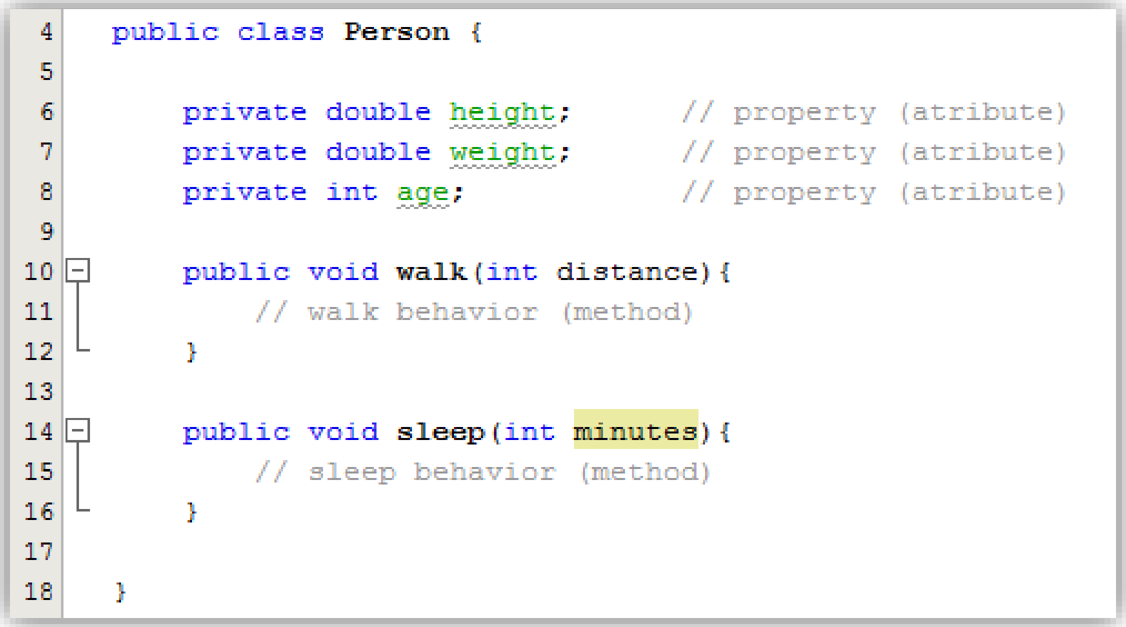
javax.swing: Swing components, also used for GUI development

And there are **many** more …

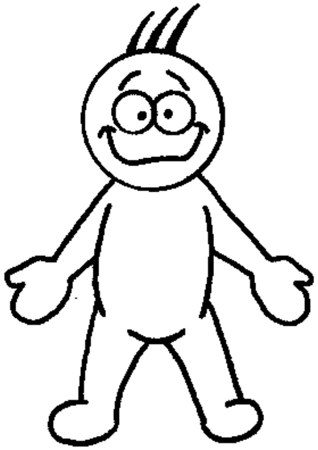
Java Is Object-Oriented from the Ground Up

Primitive or simple data types are still just single pieces of information

Object-oriented objects are complex types that have multiple pieces of information and specific **properties** (or attributes) and **behaviors** (methods).



```
public class Person {

    private double height;      // property (atribute)
    private double weight;      // property (atribute)
    private int age;            // property (atribute)

    public void walk(int distance){
        // walk behavior (method)
    }

    public void sleep(int minutes){
        // sleep behavior (method)
    }

}
```



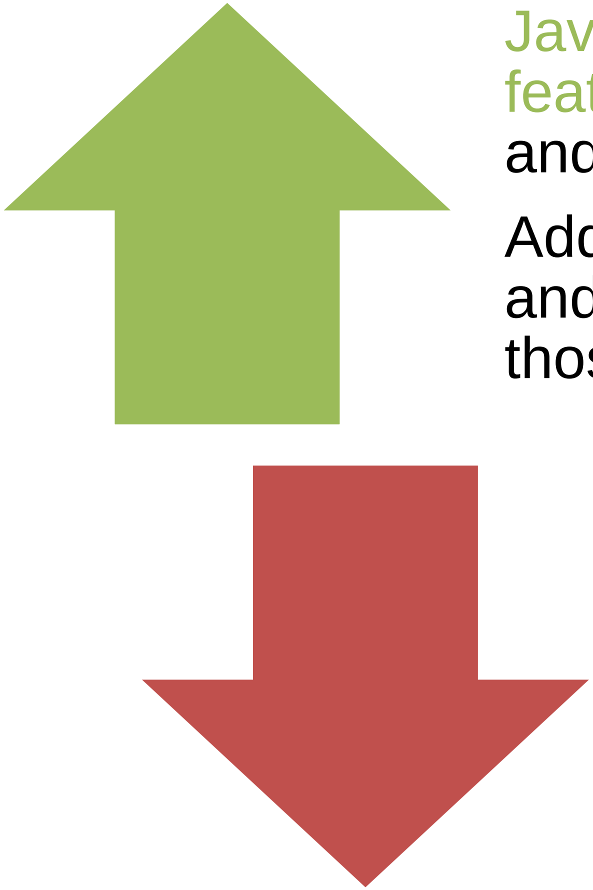
Java Is Object-Oriented from the Ground Up

All data, with the exception of a few primitive types are **objects**.

All of the GUI building blocks windows, buttons, text input fields, scroll bars, lists, menus, and so on **are objects**.

All functions are associated with objects and are known as methods there can be no “free-floating” functions as there are in C/C++.

Practice Makes Perfect

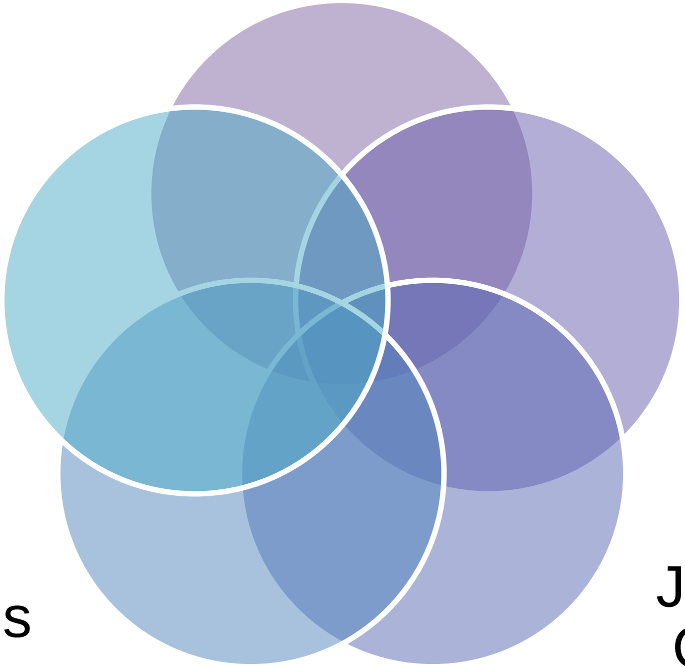


Java taken the best features of C++,Eiffel, Ada, and Smalltalk

Added some capabilities and features not found in those languages.

Features that had proven to be most troublesome in those earlier languages were eliminated.

Java advantages



Java Is Free!

Java Provides “One-Stop Shopping”

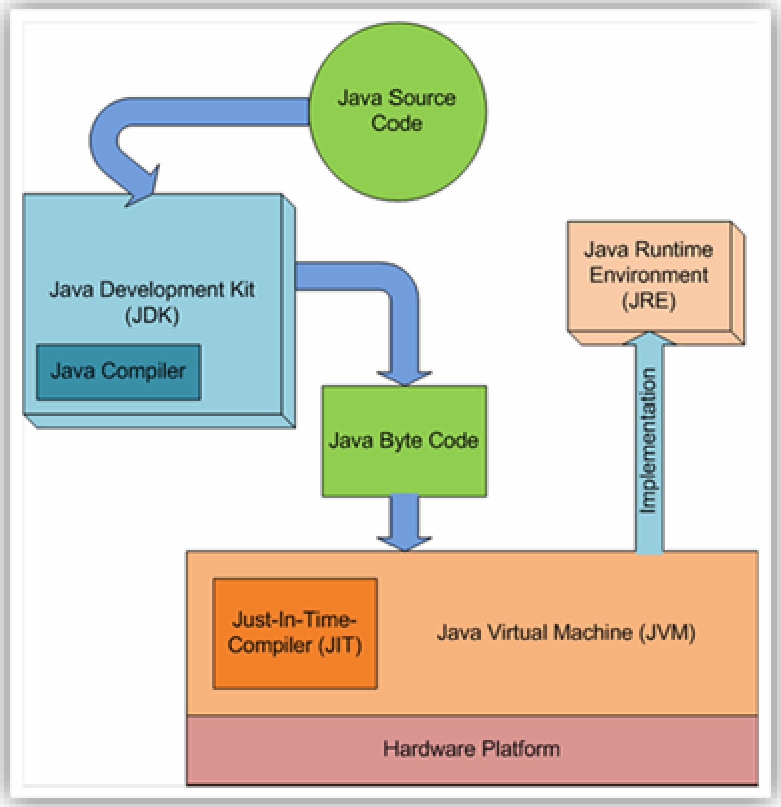
Java Is Object-Oriented from the Ground Up

Practice Makes Perfect

Java Is Architecture Neutral

2.2 Java Acronyms

Java acronyms

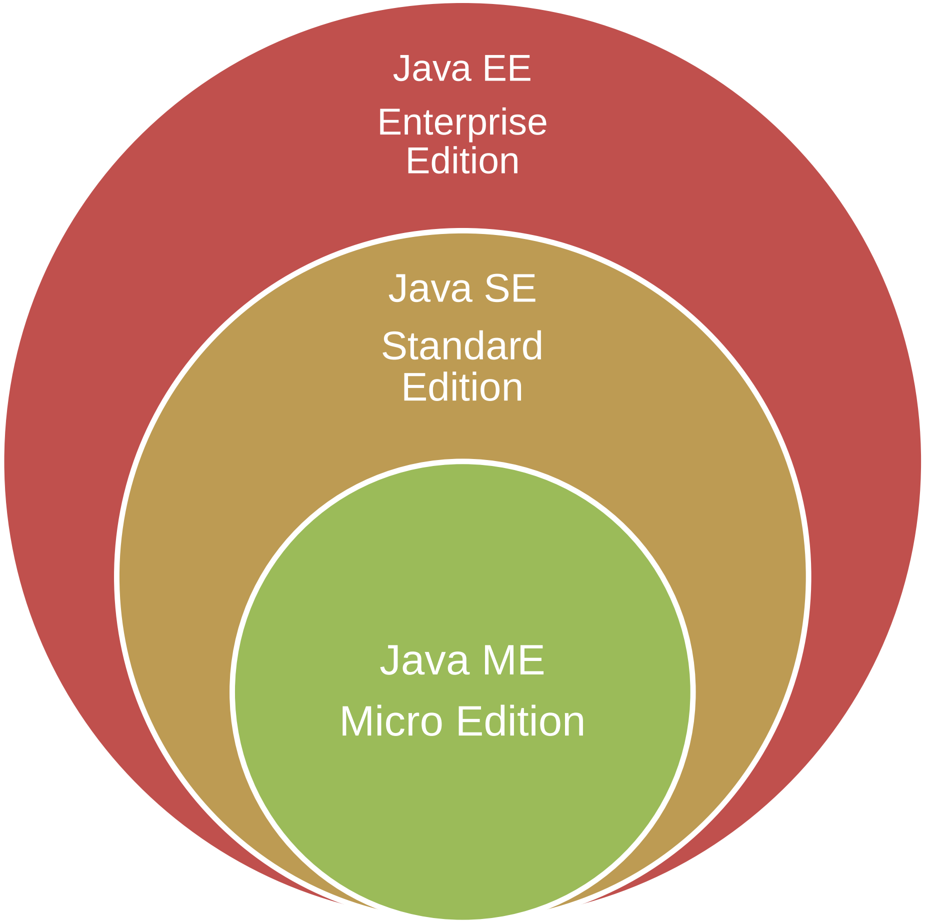

Java Source Code
Java Development Kit (JDK)
Java Compiler
Java Runtime Environment (JRE)
Java Byte Code
Implementation
Just-In-Time-Compiler (JIT)
Java Virtual Machine (JVM)
Hardware Platform

JDK: Java Developer Kit: Develop and execution

JRE: Java Runtime Environment: Execution

JVM: Java Virtual Machine

Java acronyms



3. Basic Notions of Java

3.1 File Structure

3.2 Classes

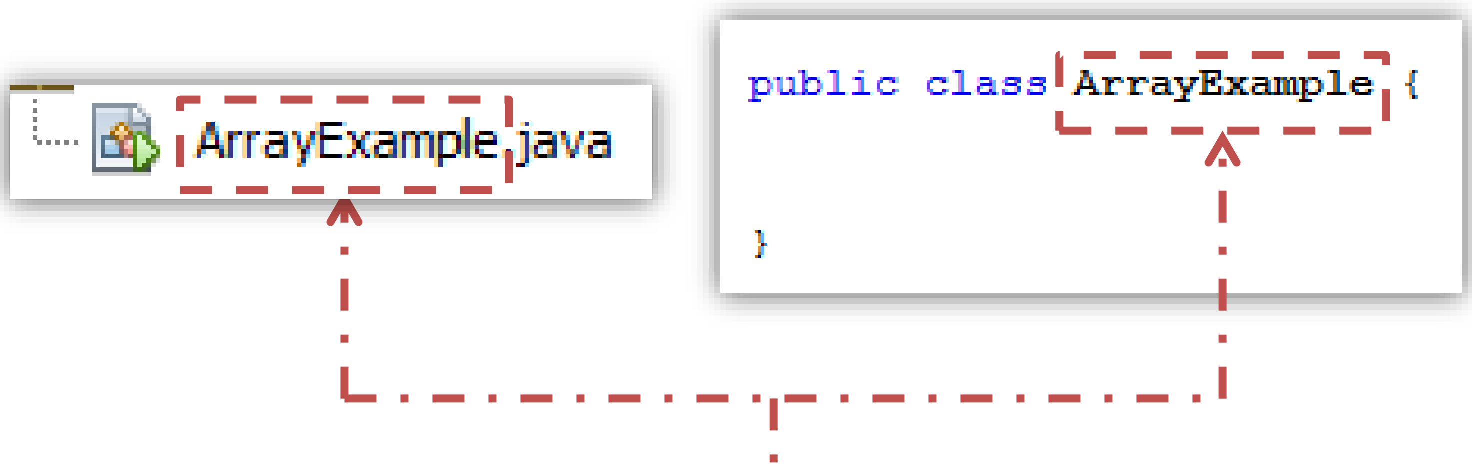
3.3 Methods

3.4 Comments

3.5 Operators

3.1 File Structure

A source code file holds **one** class definition



ArrayExample.java

```
public class ArrayExample {

}
```

Same name

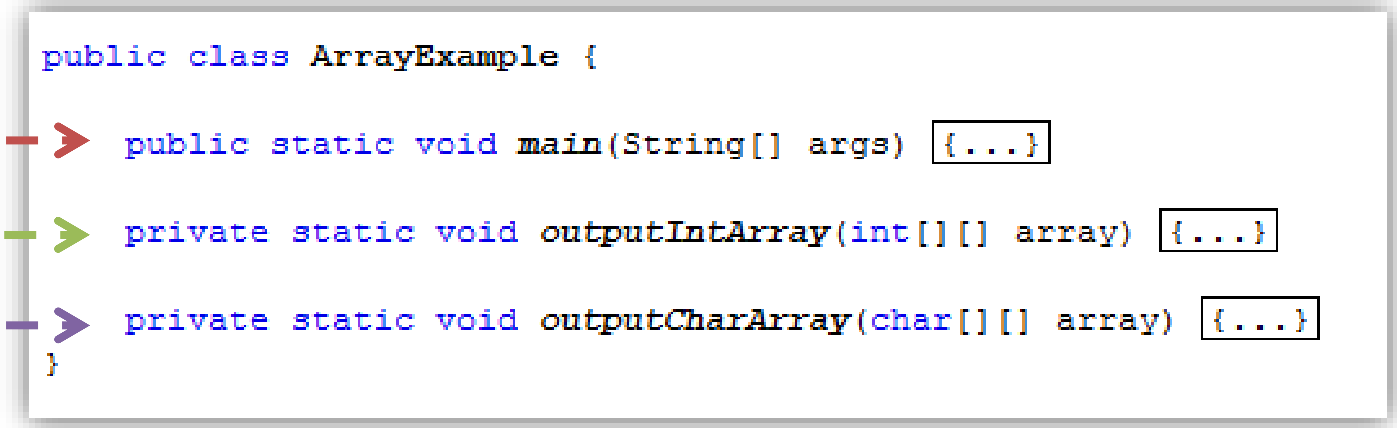
Put a class in a source file
!!!

A class holds **one or more** methods

Method 1

Method 2

Method 3



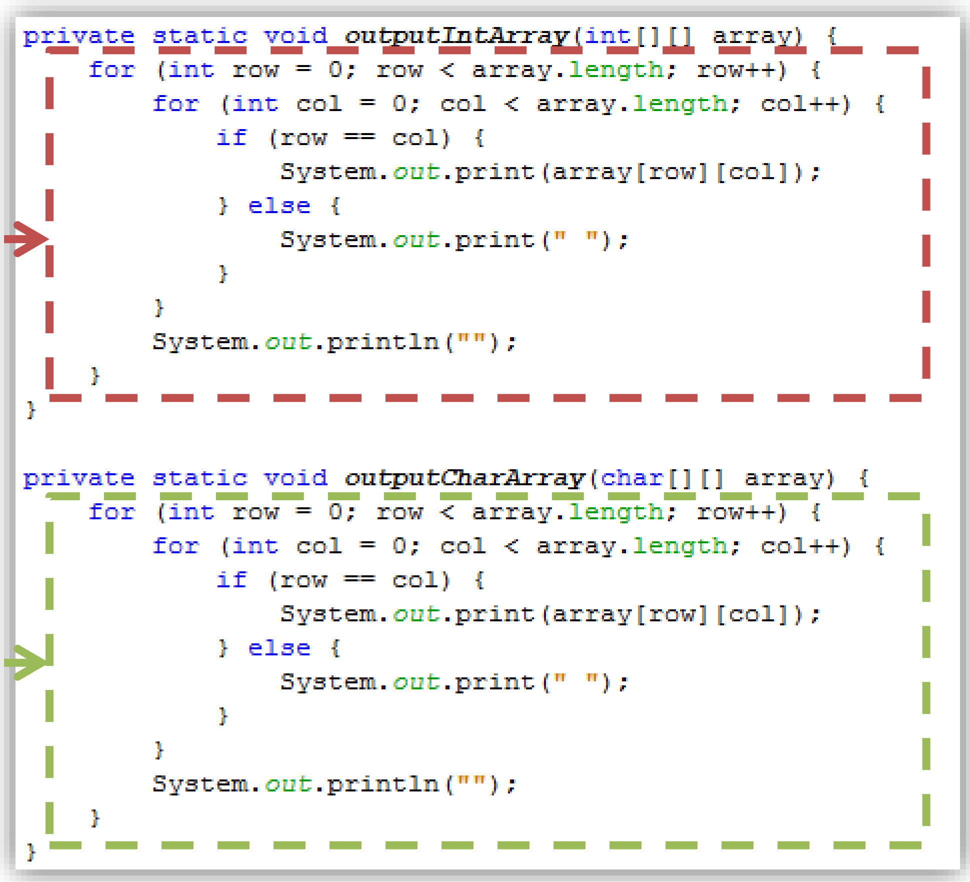
```
public class ArrayExample {

    public static void main(String[] args) {...}

    private static void outputIntArray(int[][] array) {...}

    private static void outputCharArray(char[][] array) {...}
}
```

Put methods in a class !!!

A method holds **statements**

Method 1 Statements

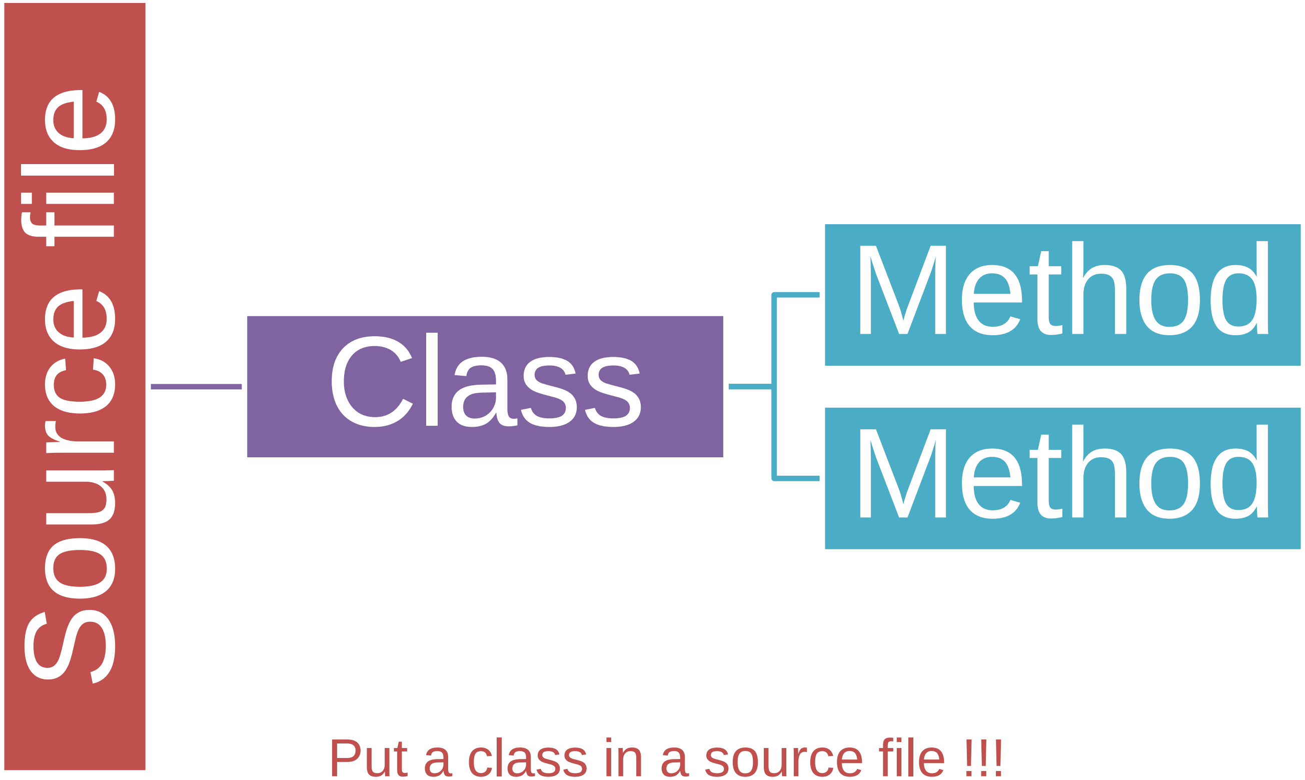
Method 2 Statements



```
private static void outputIntArray(int[][] array) {
    for (int row = 0; row < array.length; row++) {
        for (int col = 0; col < array.length; col++) {
            if (row == col) {
                System.out.print(array[row][col]);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}

private static void outputCharArray(char[][] array) {
    for (int row = 0; row < array.length; row++) {
        for (int col = 0; col < array.length; col++) {
            if (row == col) {
                System.out.print(array[row][col]);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}
```

Put statements in a method !!!

File structure



Put a class in a source file !!!

Put methods in a class !!!

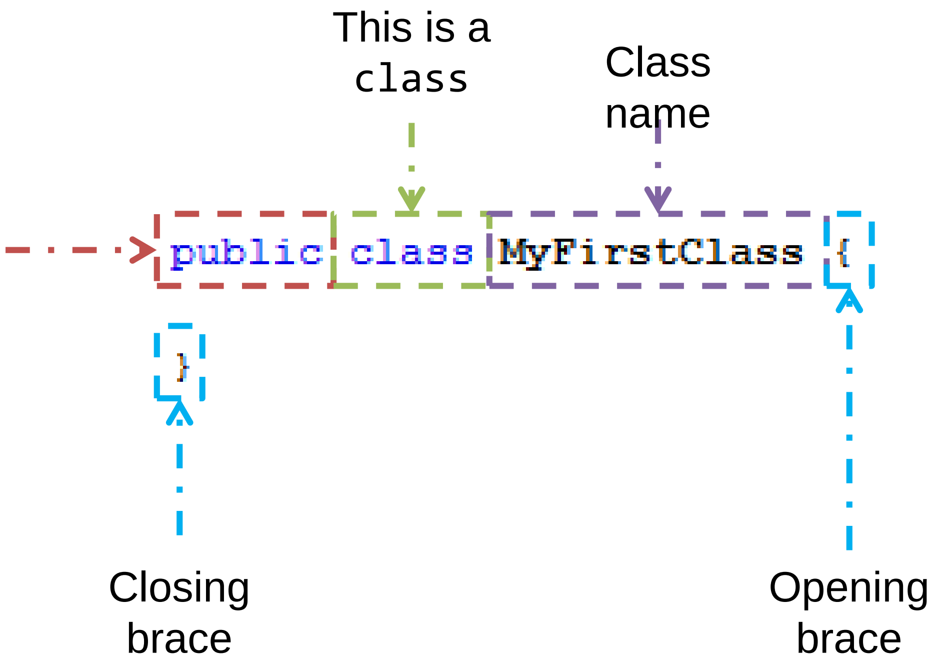
Put statements in a method !!!

3.2 Classes

Class definition

Define who can access the class:

`public` means everyone can access it



This is a `class`

Class name

```
public class MyFirstClass {
}
```

Closing brace

Opening brace

Class names Should be **nouns**

- Should be **nouns**, in mixed case with the first letter of each internal word capitalized.
- Try to keep your class names **simple and descriptive**.
- Use **whole words**, avoid **acronyms and abbreviations**.
- Java is case sensitive.

Good Examples:

- `class SoccerPlayer {…}`
- `class Person {…}`

Bad Examples

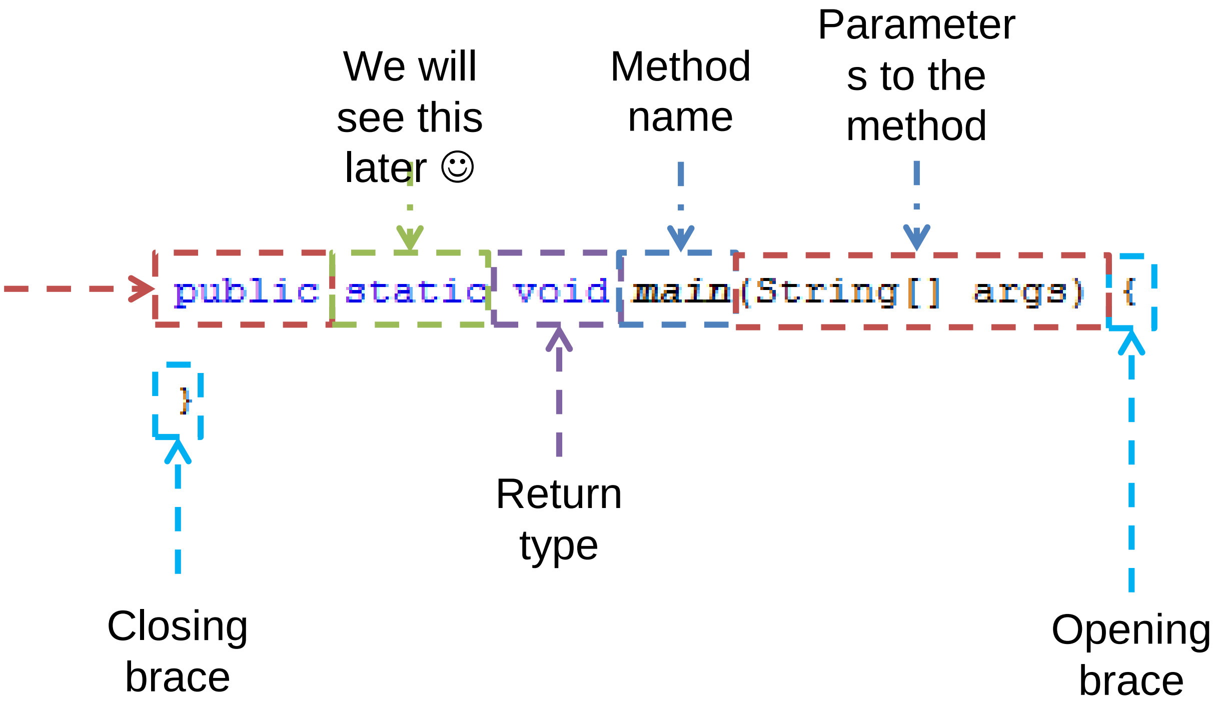
- `class XYZ {…}`
- `class PERSON {…}`
- `class soccerplayer {…}`

3.3 Methods

Methods definition

Define who can access the class:

`public` means everyone can access it



```
public static void main(String[] args) {
}
```

We will see this later ☺

Method name

Parameters to the method

Return type

Closing brace

Opening brace

Methods names Should be **verbs**

Good Examples:

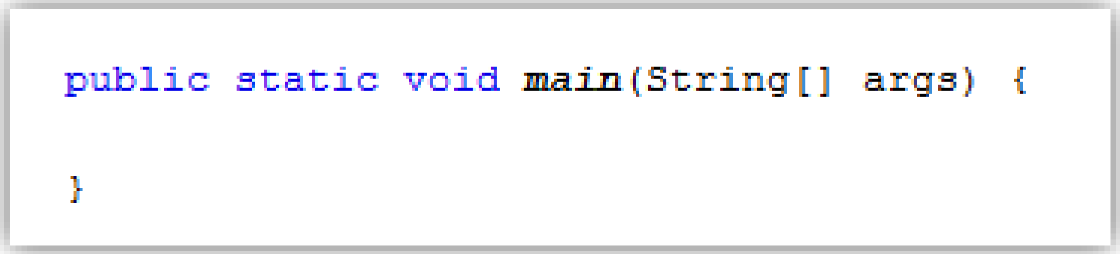
- `private static void` **`play`**`(int coinValue) {…}`
- `public static void` **`moveToRight`**`(int steps) {…}`
- `public static void` **`getDirection`**`() {…}`

Bad Examples

- `public static void` **`person`**`() {…}`
- `public static void` **`PLAY`**`() {…}`
- `public static void` **`soccerplayer`**`() {…}`

Should be **verbs** (behaviors), in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

The `main` method is where your program start to run



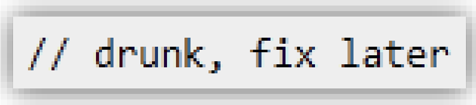
```
public static void main(String[] args) {

}
```

Is not necessary a main method in a class

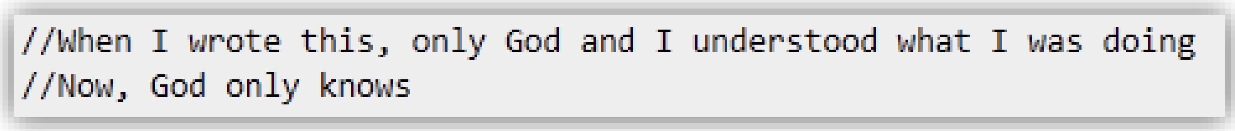
3.4 Comments

Comments improve readability of source code

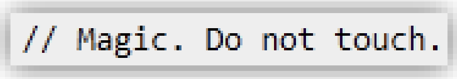
Most of the times ;)



```
// drunk, fix later
```

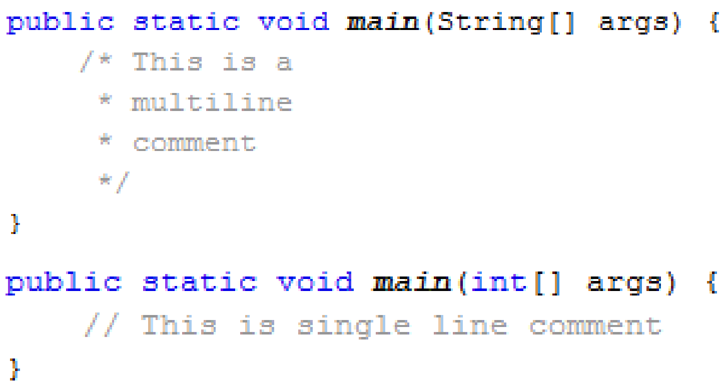


```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```



```
// Magic. Do not touch.
```

A good source code do not required comments

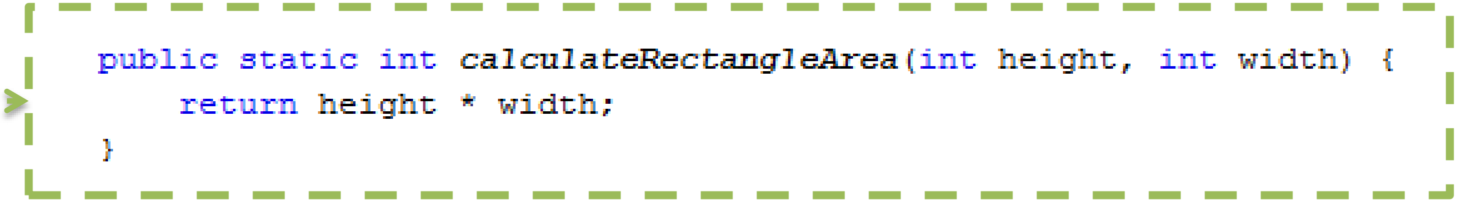


```
public static void main(String[] args) {
    /* This is a
     * multiline
     * comment
     */
}
public static void main(int[] args) {
    // This is single line comment
}
```

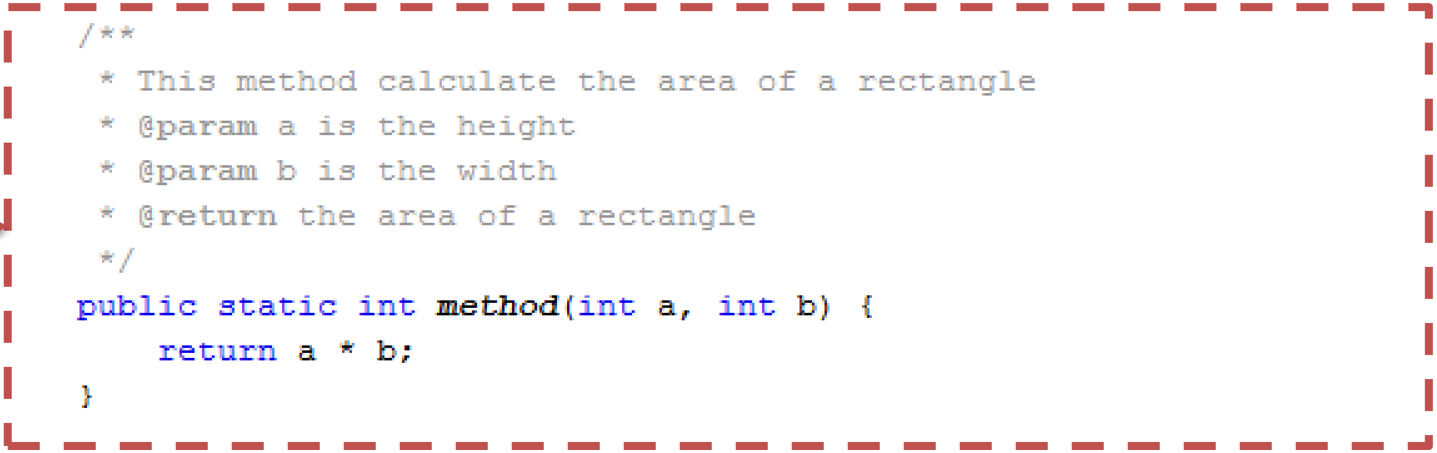
[What is the best comment in source code you have ever encountered]

Self explanatory code vs Commented code

Self explanatory



```
public static int calculateRectangleArea(int height, int width) {
    return height * width;
}
```



```
/**
 * This method calculate the area of a rectangle
 * @param a is the height
 * @param b is the width
 * @return the area of a rectangle
 */
public static int method(int a, int b) {
    return a * b;
}
```

Commented
code

3.5 Operators

3.5.1 Precedence of operators

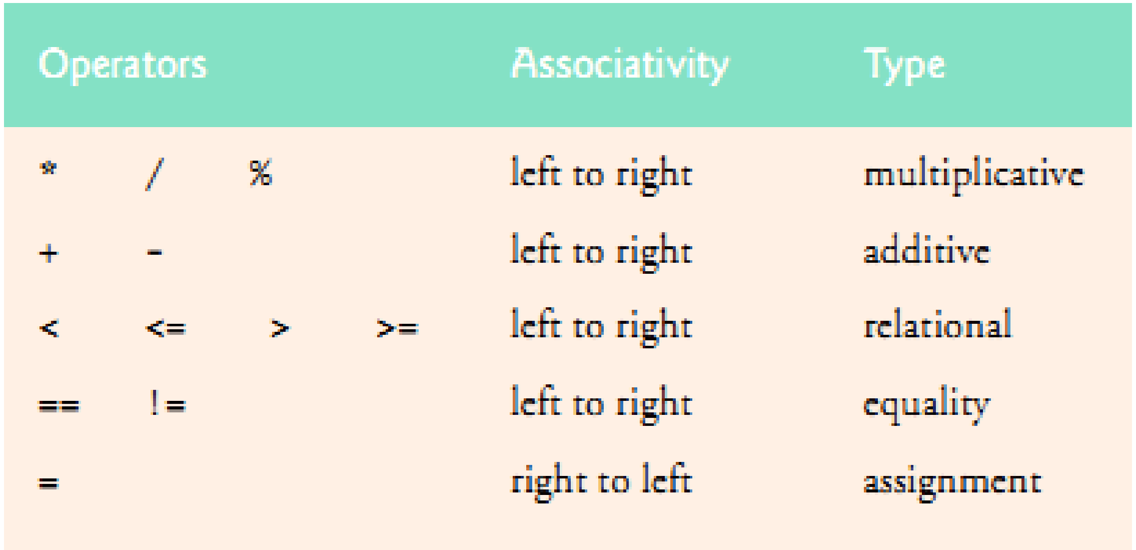
3.5.2 Assignment operators

3.5.3 Increment and Decrement Operators

3.5.4 Logical Operators

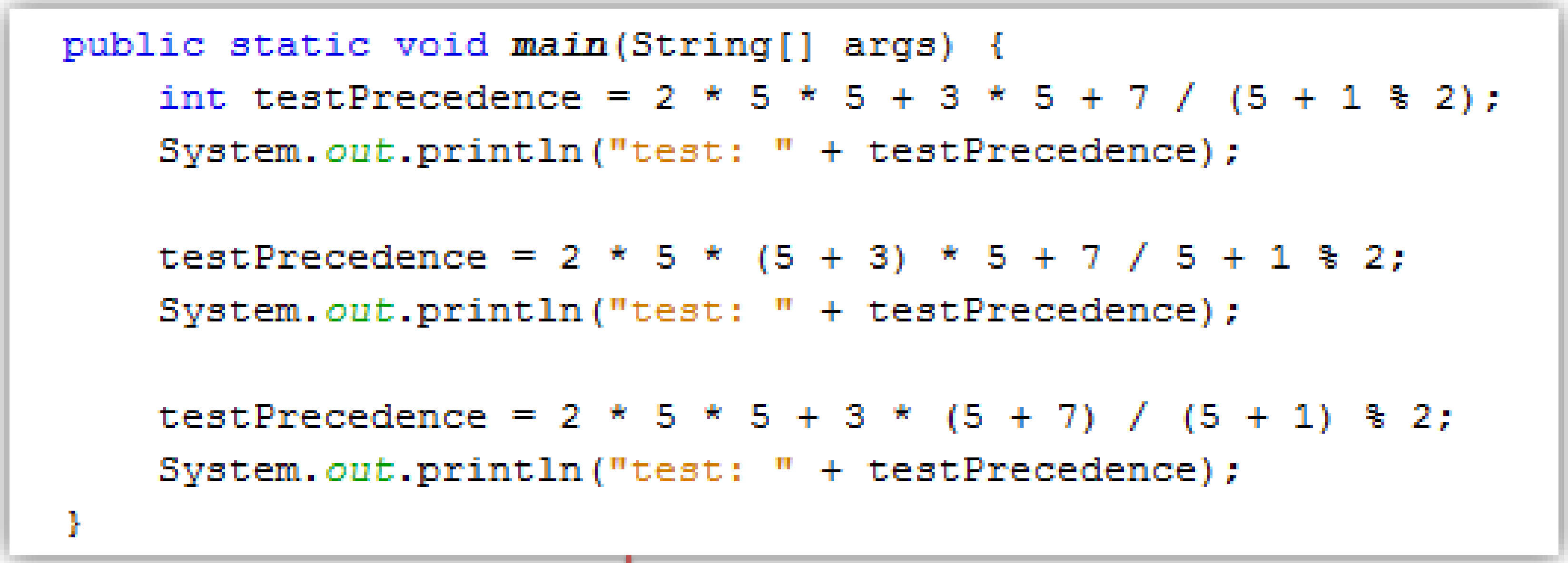
3.5.1 Precedence of Operators

Precedence of arithmetic operators



Operators	Associativity	Type
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

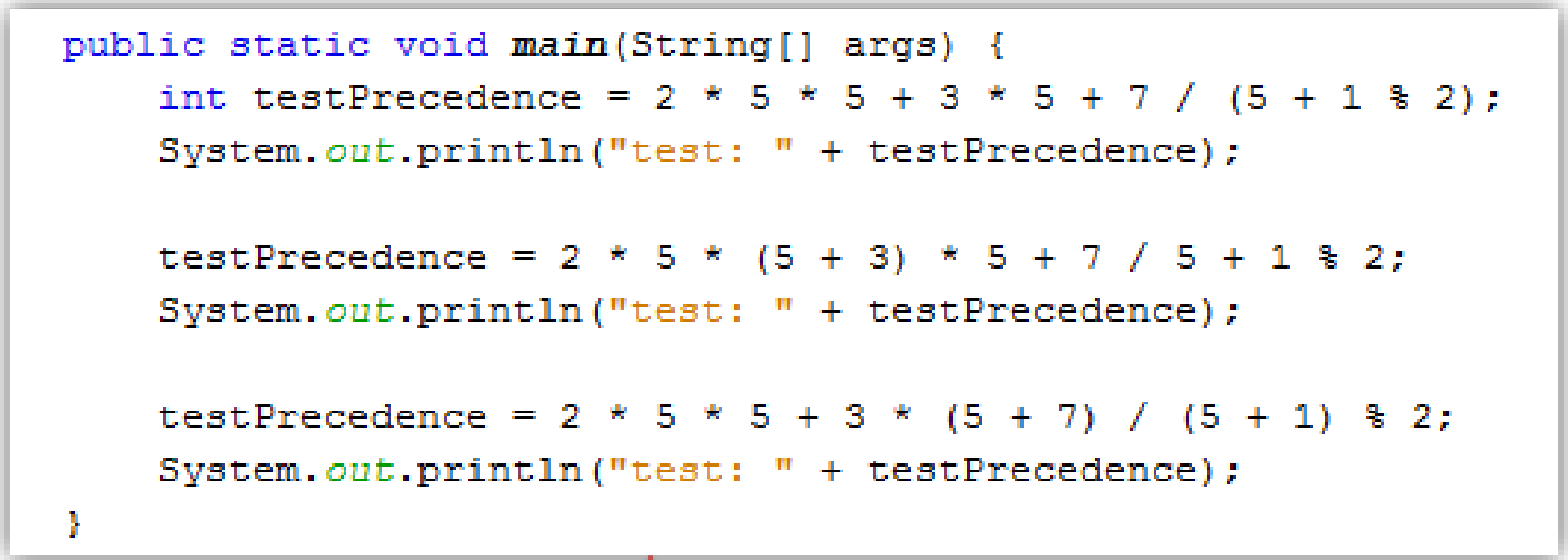
Precedence of arithmetic operators



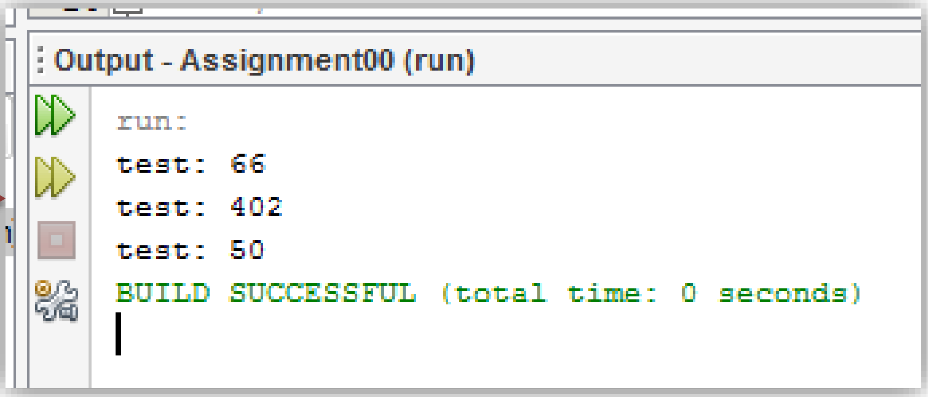
```
public static void main(String[] args) {
    int testPrecedence = 2 * 5 * 5 + 3 * 5 + 7 / (5 + 1 % 2);
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * (5 + 3) * 5 + 7 / 5 + 1 % 2;
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * 5 + 3 * (5 + 7) / (5 + 1) % 2;
    System.out.println("test: " + testPrecedence);
}
```

????
??

Precedence of arithmetic operators



```java
public static void main(String[] args) {
    int testPrecedence = 2 * 5 * 5 + 3 * 5 + 7 / (5 + 1 % 2);
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * (5 + 3) * 5 + 7 / 5 + 1 % 2;
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * 5 + 3 * (5 + 7) / (5 + 1) % 2;
    System.out.println("test: " + testPrecedence);
}
```

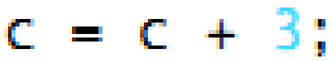


```
run:
test: 66
test: 402
test: 50
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.5.2 Assignment Operators

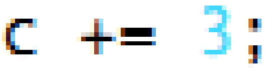
Assignment Operators

```
variable = variable operator expression;
```



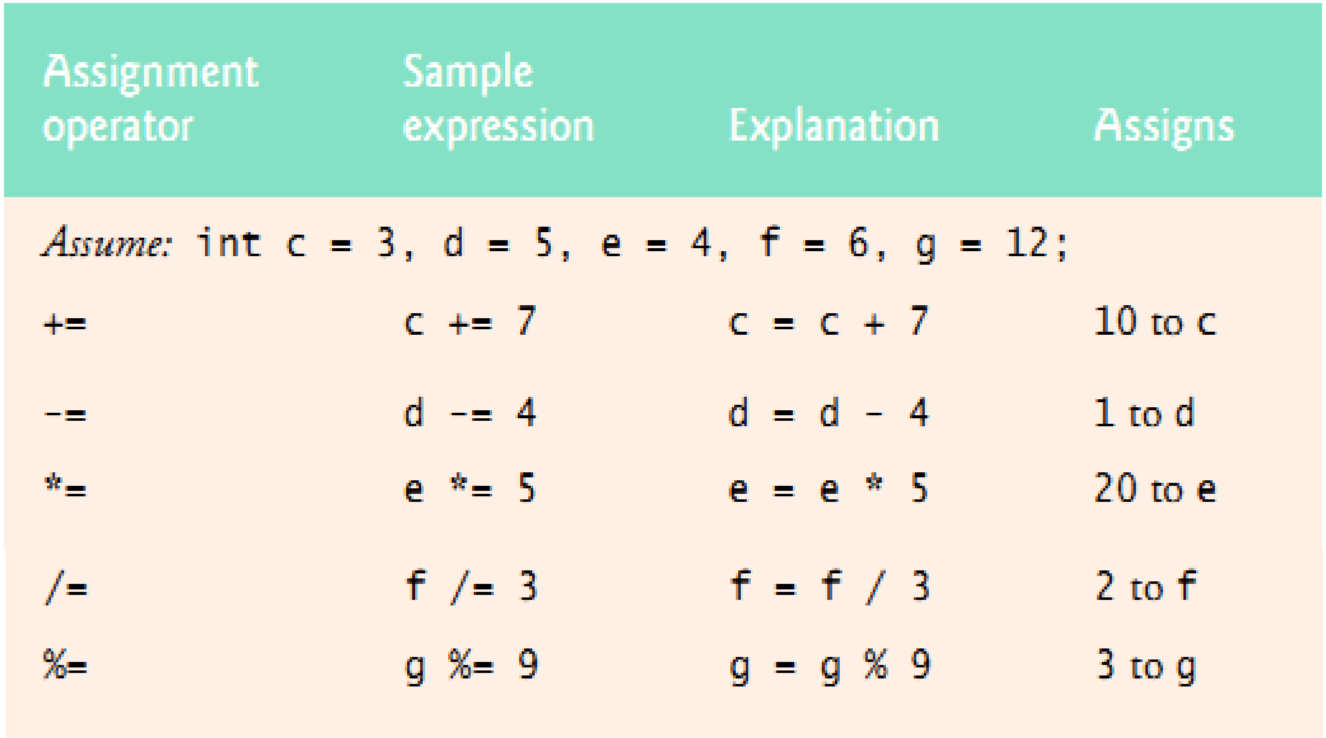
```
c = c + 3;
```

=



```
c += 3;
```

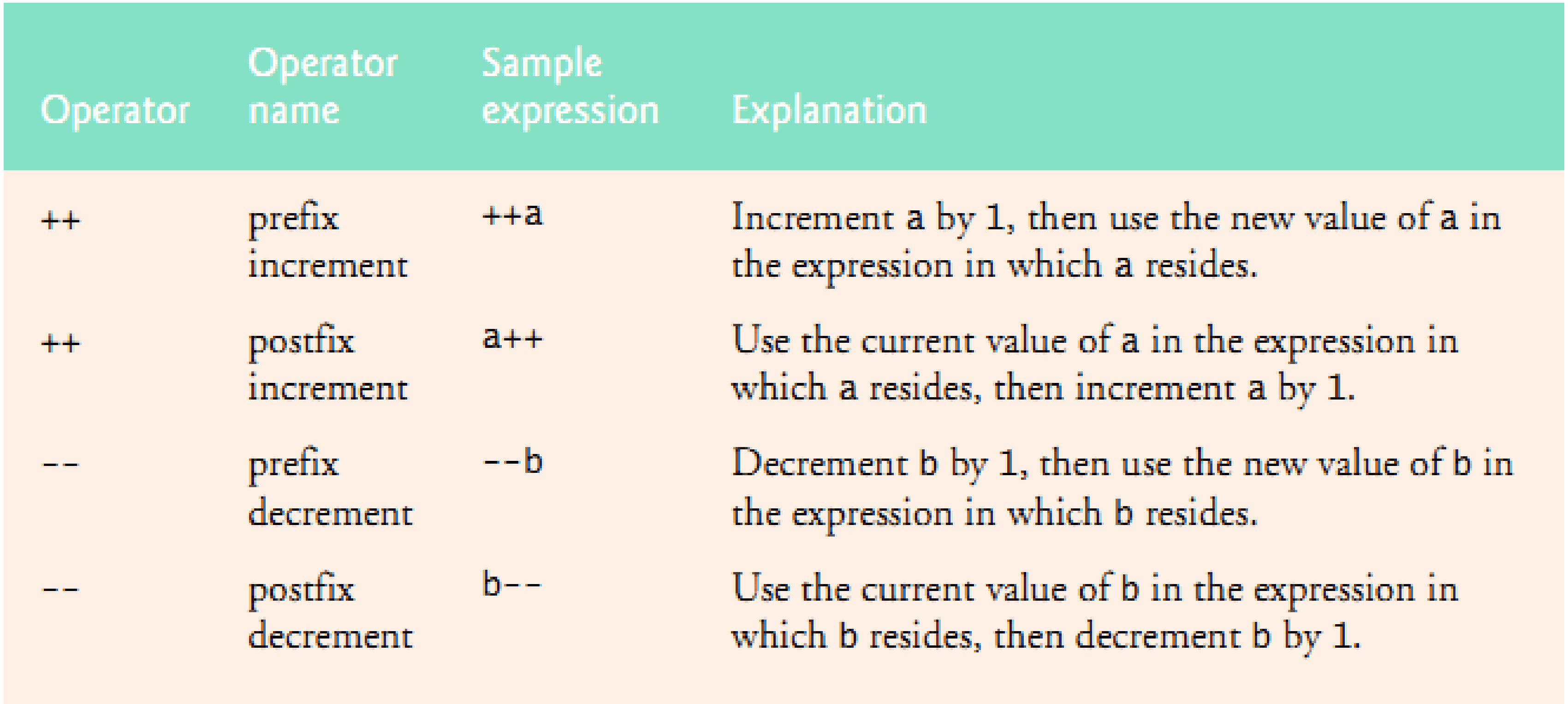
Compound Assignment Operators



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`

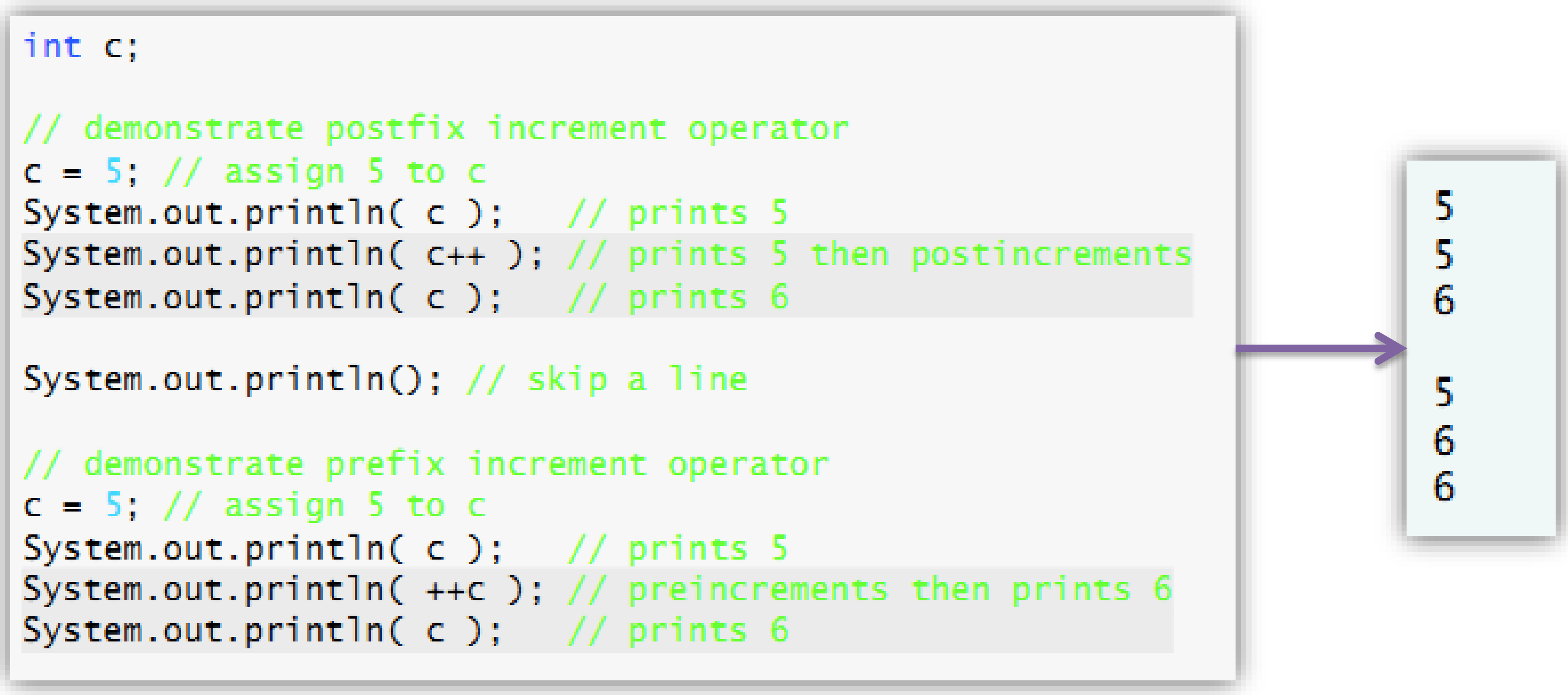
3.5.3 Increment and Decrement Operators

Increment and Decrement Operators



Operator	Operator name	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.

Prefix and Postfix Example



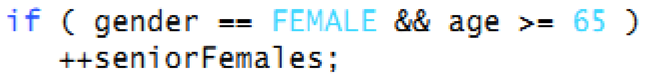
```
int c;

// demonstrate postfix increment operator
c = 5; // assign 5 to c
System.out.println( c );   // prints 5
System.out.println( c++ ); // prints 5 then postincrements
System.out.println( c );   // prints 6

System.out.println(); // skip a line

// demonstrate prefix increment operator
c = 5; // assign 5 to c
System.out.println( c );   // prints 5
System.out.println( ++c ); // preincrements then prints 6
System.out.println( c );   // prints 6
```

```
5
5
6

5
6
6
```

3.5.4 Logical Operators

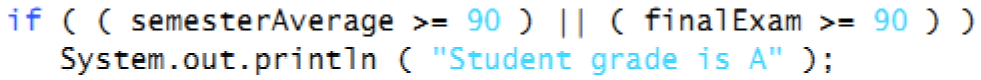
Logical Operators

Conditional AND (&&)



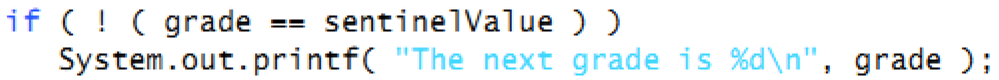
```
if ( gender == FEMALE && age >= 65 )
   ++seniorFemales;
```

Conditional OR (||)



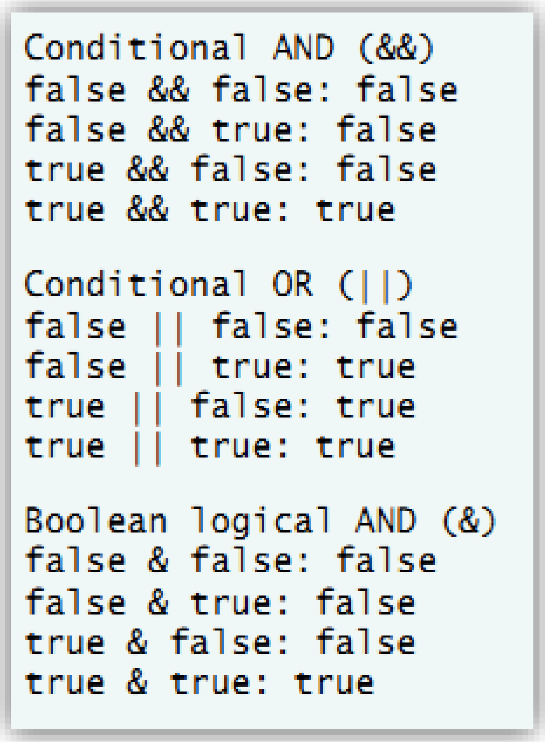
```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 ) )
   System.out.println ( "Student grade is A" );
```

Logical Negation (!)

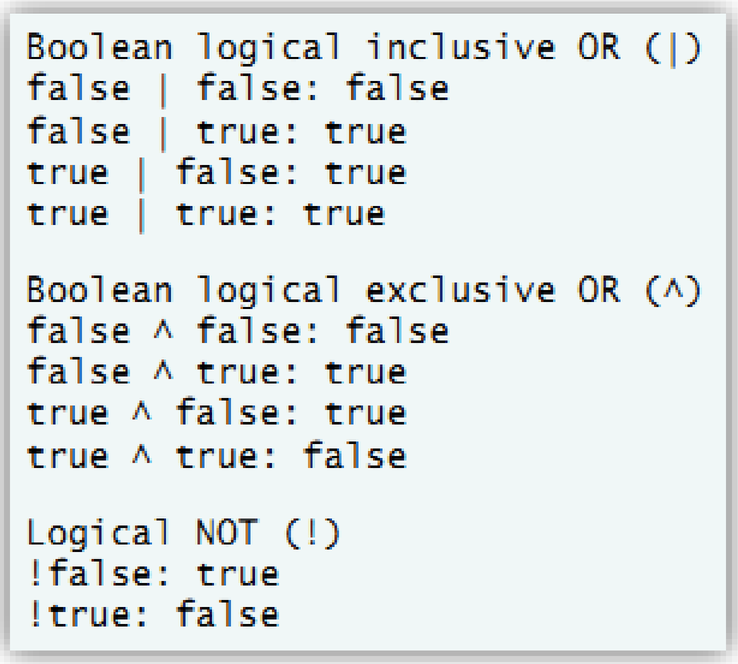


```
if ( ! ( grade == sentinelValue ) )
   System.out.printf( "The next grade is %d\n", grade );
```

Logical Operators - Truth tables



```
Conditional AND (&&)
false && false: false
false && true: false
true && false: false
true && true: true

Conditional OR (||)
false || false: false
false || true: true
true || false: true
true || true: true

Boolean logical AND (&)
false & false: false
false & true: false
true & false: false
true & true: true
```



```
Boolean logical inclusive OR (|)
false | false: false
false | true: true
true | false: true
true | true: true

Boolean logical exclusive OR (^)
false ^ false: false
false ^ true: true
true ^ false: true
true ^ true: false

Logical NOT (!)
!false: true
!true: false
```

4. Control Structures

4.1 Sequence Structure

4.2 Selection Statements

4.3 Repetition Statements

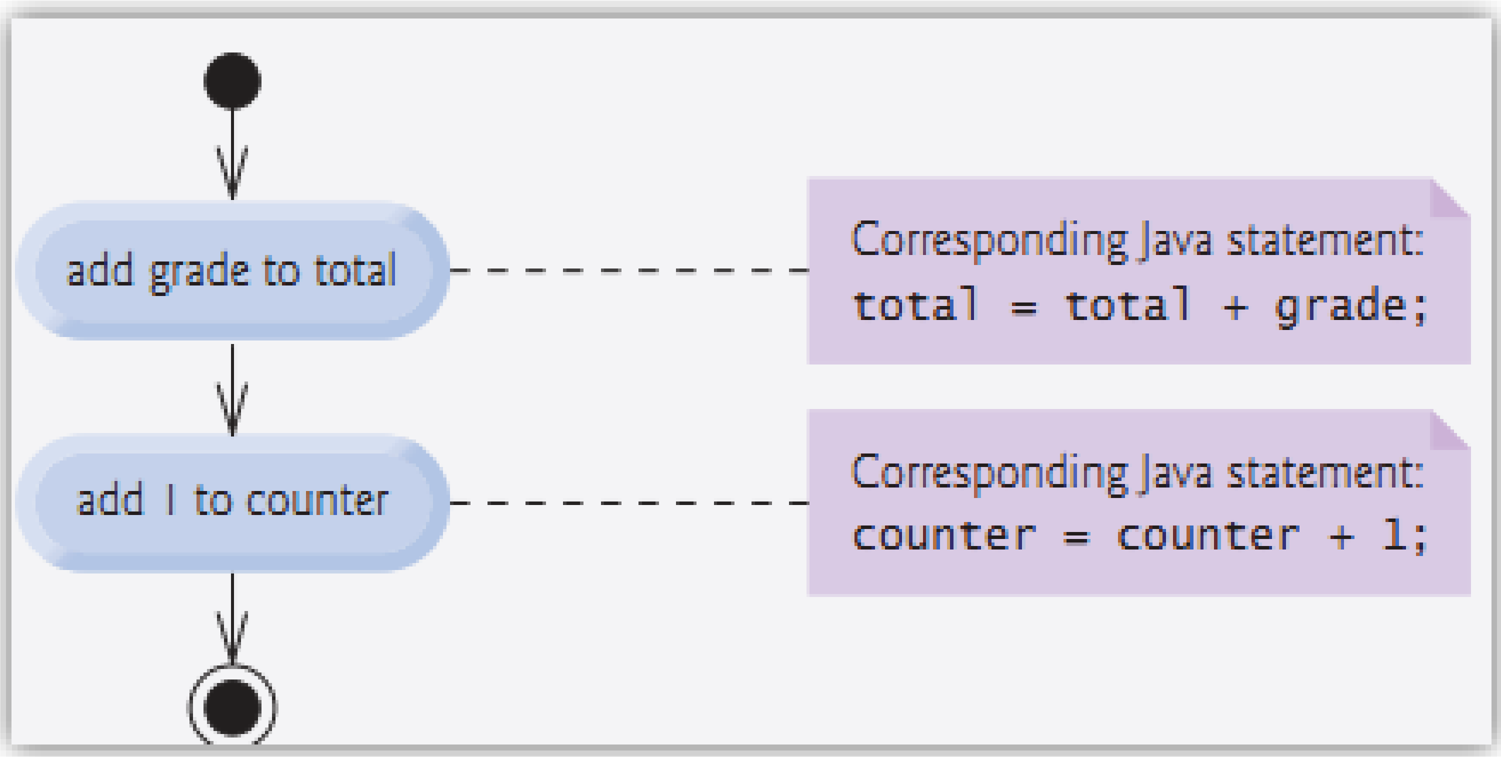
Control Structures

Programs are formed by combining as many **sequence**, **selection** and **repetition** statements.

selection	repetition
if	while
if…else	do…while
switch	for

4.1 Sequence Structure

Sequence structure



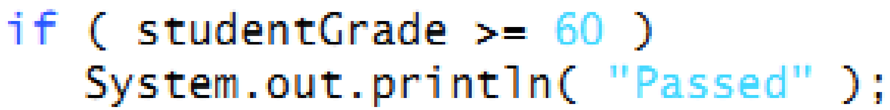
An ordered execution of two or more statements are called **sequence structure**

4.2 Selection Statements

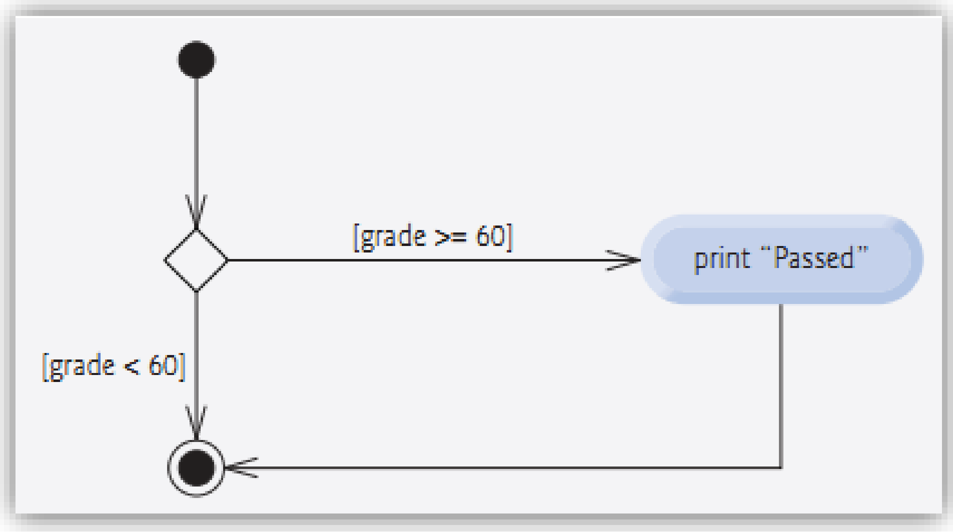
IF Selection Statement

If student’s grade is greater than or equal to 60

Print “**Passed**”



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



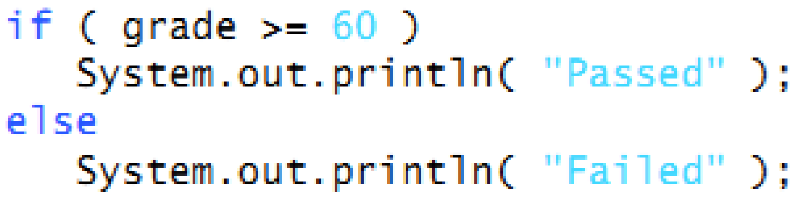
IF..ELSE Selection Statement

If student’s grade is greater than or equal to 60

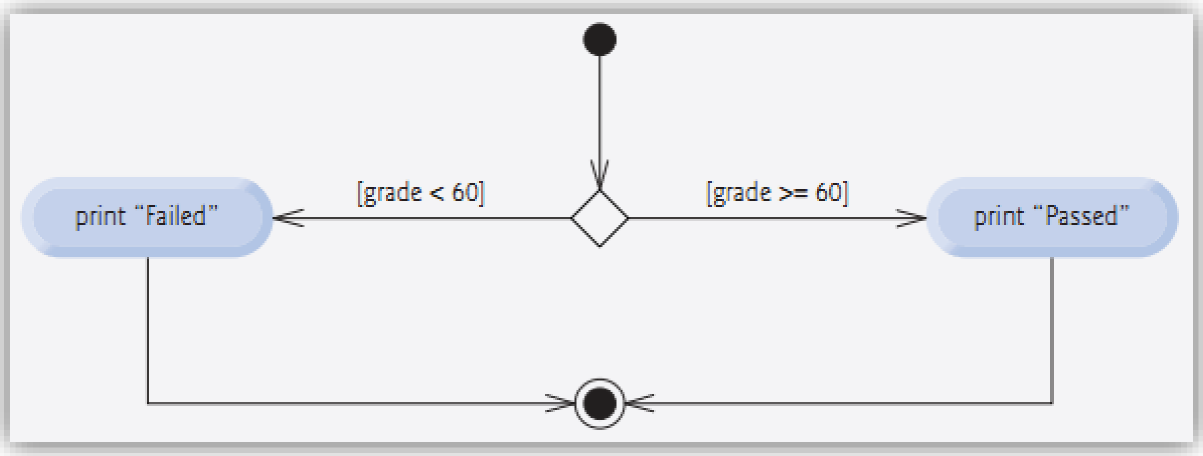
    Print “**Passed**”

Else

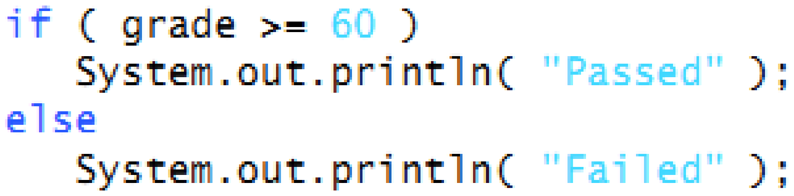
    Print “**Failed**”



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

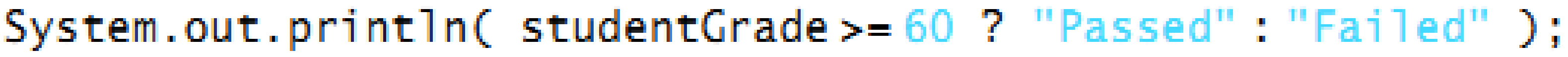


IF..ELSE Selection Statement abbreviated form



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

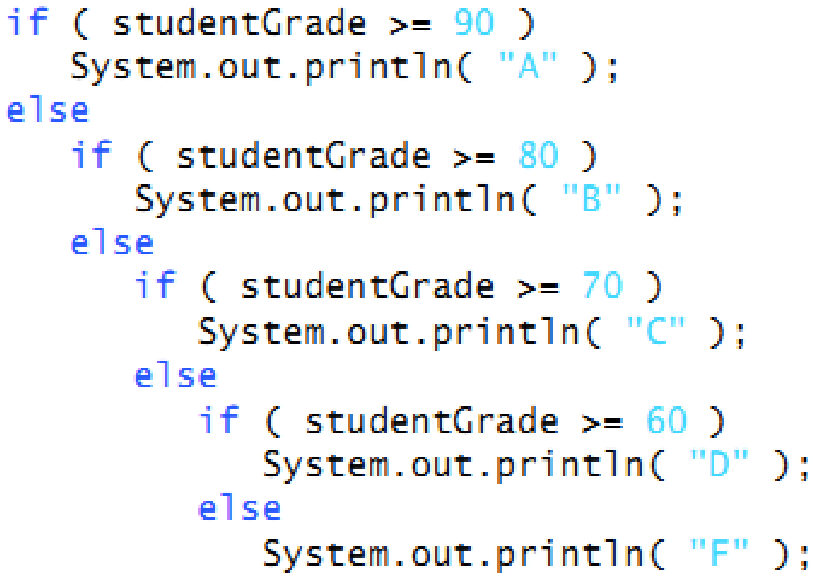
=



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

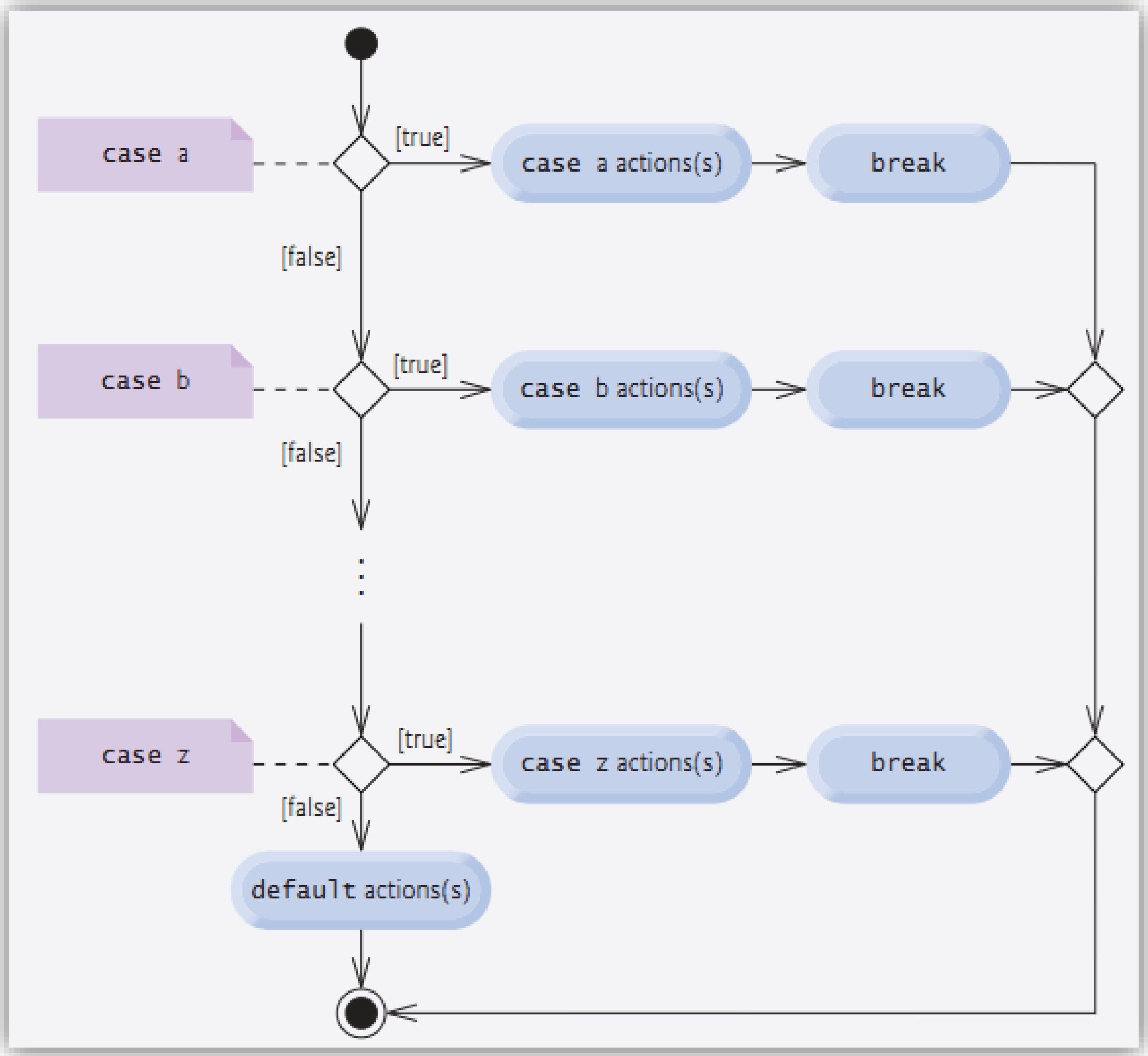
Nested IF..ELSE Selection Statement

```
If student’s grade is greater than or equal to 90
    Print “A”
else
    If student’s grade is greater than or equal to 80
        Print “B”
    else
        If student’s grade is greater than or equal to 70
            Print “C”
        else
            If student’s grade is greater than or equal to 60
                Print “D”
            else
                Print “F”
```

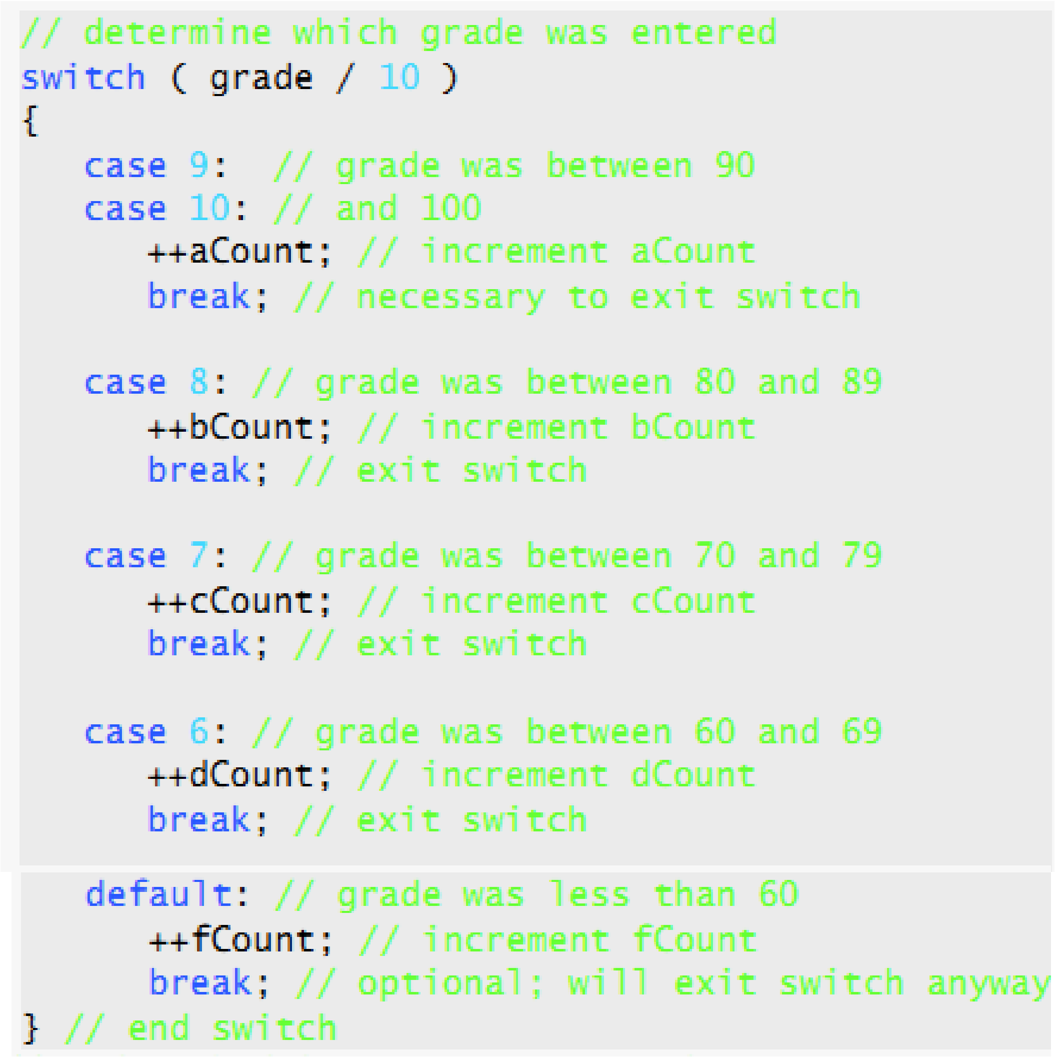


```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else
   if ( studentGrade >= 80 )
      System.out.println( "B" );
   else
      if ( studentGrade >= 70 )
         System.out.println( "C" );
      else
         if ( studentGrade >= 60 )
            System.out.println( "D" );
         else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

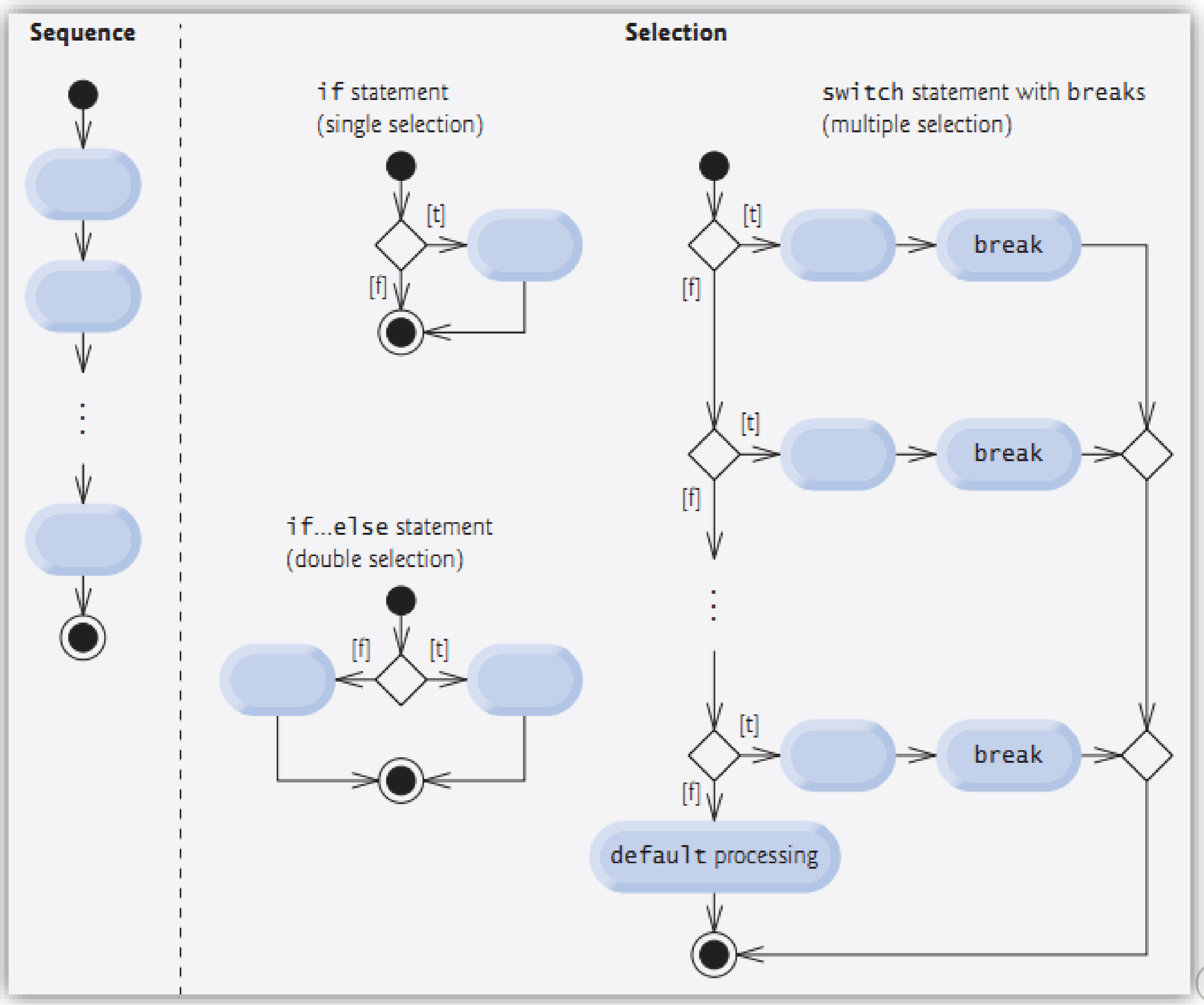
SWITCH Multiple-Selection Statement



SWITCH Multiple-Selection Statement

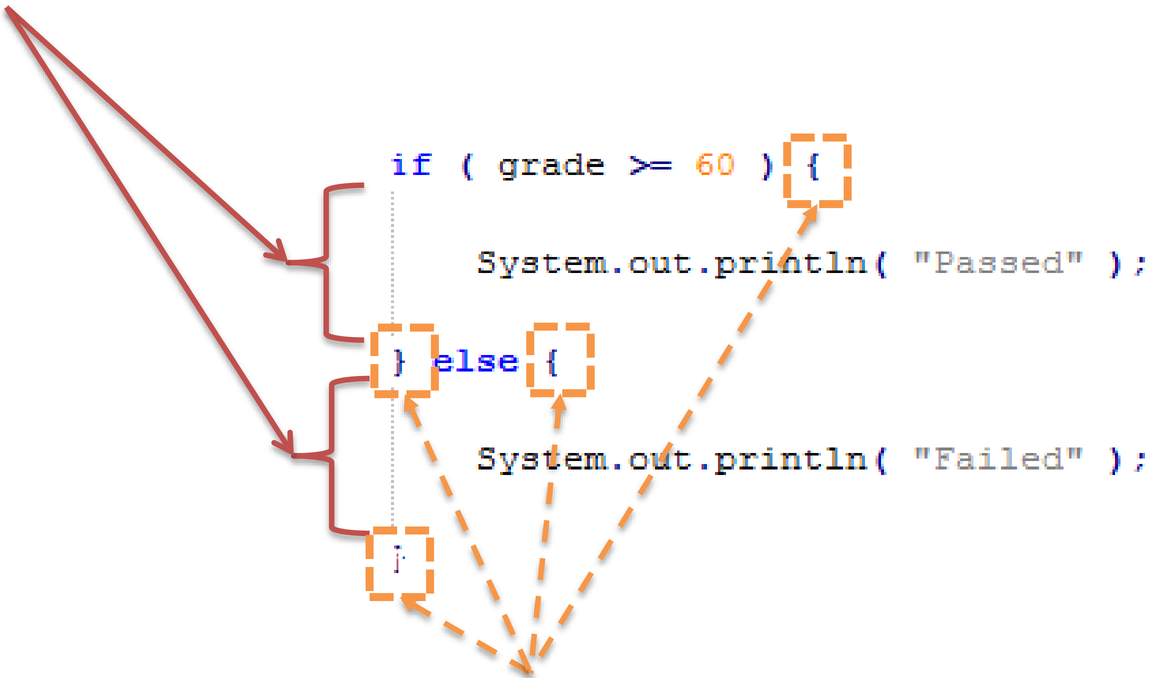


```
// determine which grade was entered
switch ( grade / 10 )
{
   case 9:  // grade was between 90
   case 10: // and 100
      ++aCount; // increment aCount
      break; // necessary to exit switch

   case 8: // grade was between 80 and 89
      ++bCount; // increment bCount
      break; // exit switch

   case 7: // grade was between 70 and 79
      ++cCount; // increment cCount
      break; // exit switch

   case 6: // grade was between 60 and 69
      ++dCount; // increment dCount
      break; // exit switch

   default: // grade was less than 60
      ++fCount; // increment fCount
      break; // optional; will exit switch anyway
} // end switch
```


Sequence
Selection
if statement
(single selection)
[t]
[f]
if...else statement
(double selection)
[f]
[t]
switch statement with breaks
(multiple selection)
[t]
[f]
break
[t]
[f]
break
[t]
[f]
break
default processing

Do not forget…

Indent both body statements of an if…else statement.

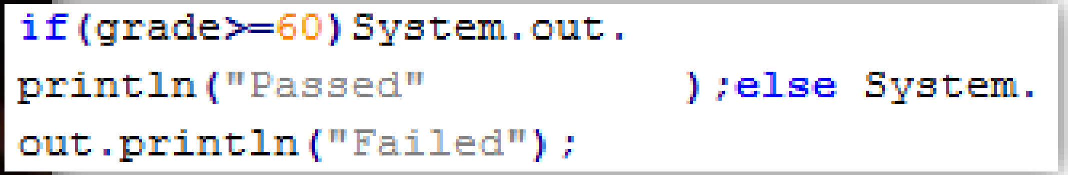


```
if ( grade >= 60 ) {
    System.out.println( "Passed" );
} else {
    System.out.println( "Failed" );
}
```

Always using braces in an if…else (or other) statement helps prevent their accidental omission

Do not be the beast

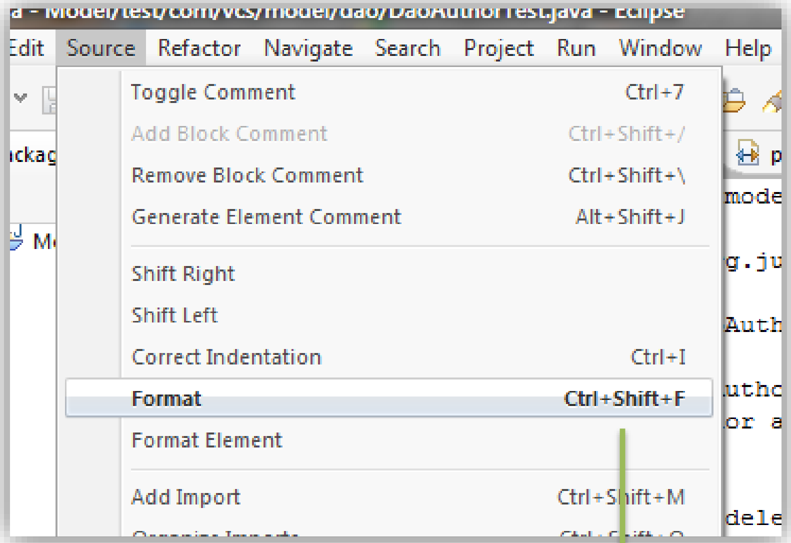
Ugly code is written by ugly people.

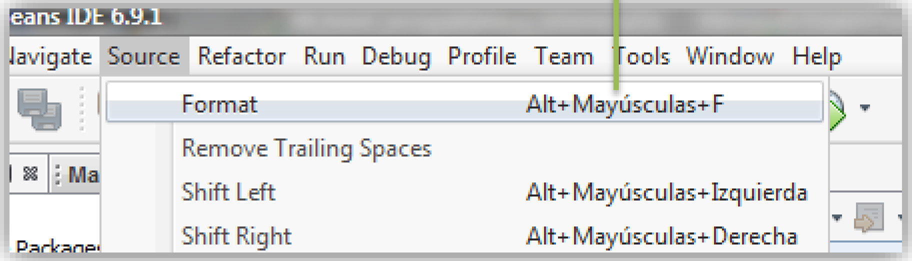


```
if(grade>=60)System.out.
println("Passed"          );else System.
out.println("Failed");
```

Do not worry, we can format the code automatically

Alt+Mayúsculas+F


a - Model/test/com/vcs/model/dao/DaoAuthorTest.java - Eclipse
Edit Source Refactor Navigate Search Project Run Window Help
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M


eans IDE 6.9.1
Navigate Source Refactor Run Debug Profile Team Tools Window Help
Format Alt+Mayúsculas+F
Remove Trailing Spaces
Shift Left Alt+Mayúsculas+Izquierda
Shift Right Alt+Mayúsculas+Derecha

Eclipse

NetBeans

Ctrl+Shift+F

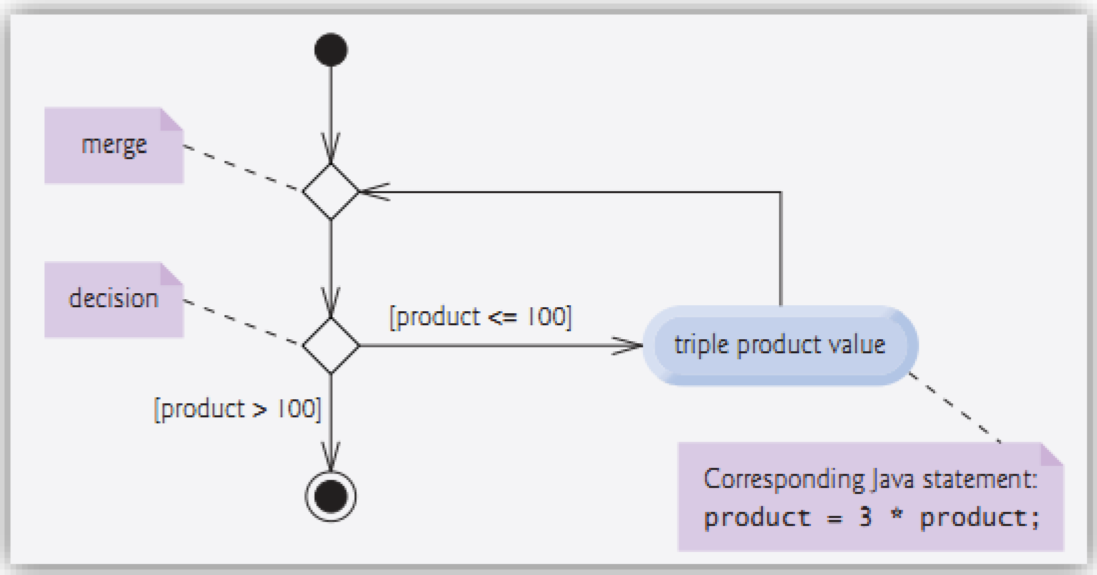
4.3 Repetition Statements

WHILE Repetition Statement

While product is less or equal than 100 products

Multiply by 3 the number of products

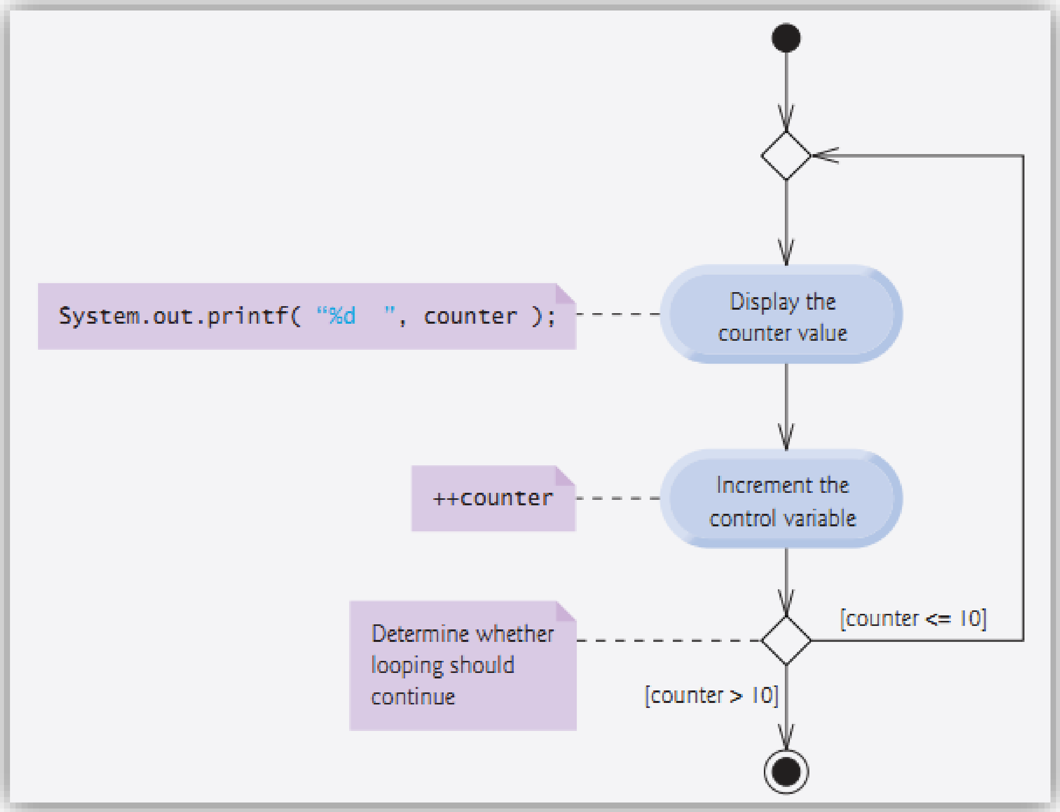
```
int product = 3;

while ( product <= 100 )
   product = 3 * product;
```

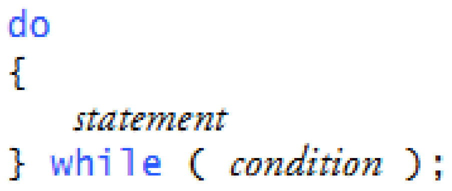


Corresponding Java statement:
`product = 3 * product;`

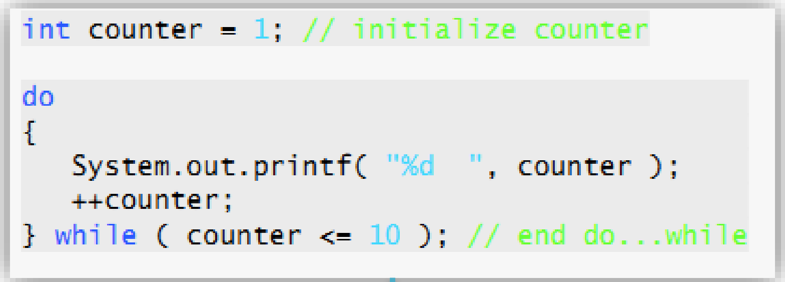
Be careful with infinite loops!!

DO…WHILE Repetition Statement

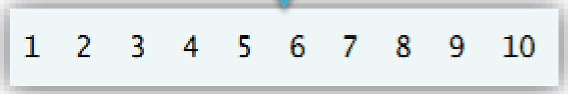

System.out.printf("%d ", counter);
Display the counter value
++counter
Increment the control variable
Determine whether looping should continue
[counter <= 10]
[counter > 10]



```
do
{
    statement
} while ( condition );
```



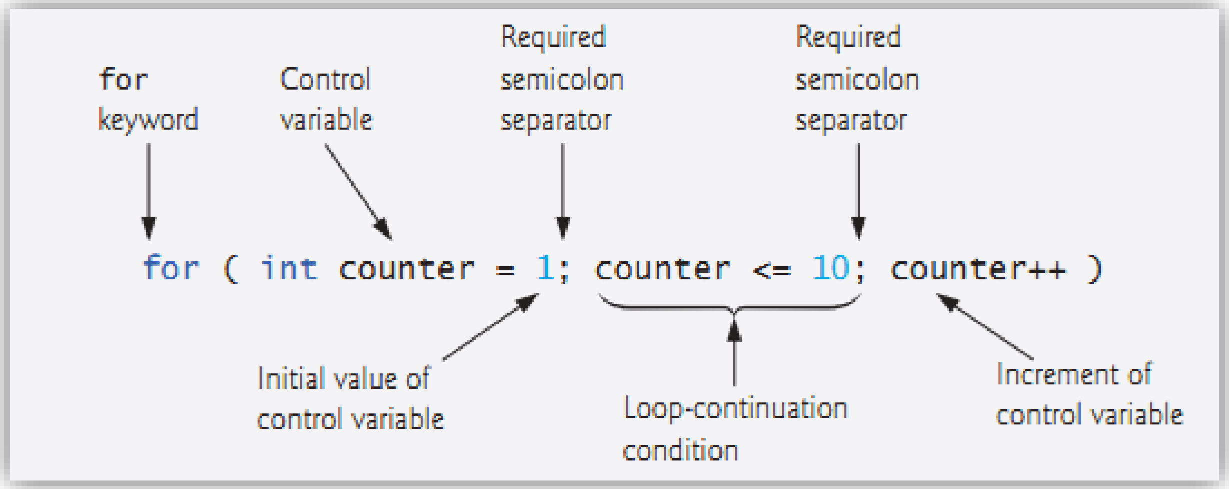
```
int counter = 1; // initialize counter

do
{
   System.out.printf( "%d  ", counter );
   ++counter;
} while ( counter <= 10 ); // end do...while
```



```
1  2  3  4  5  6  7  8  9  10
```

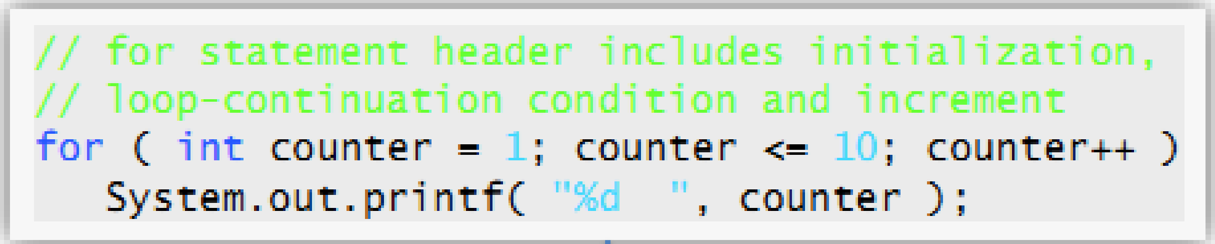
Remember always include braces !!!

FOR Repetition Statement



```
for ( int counter = 1; counter <= 10; counter++ )
```

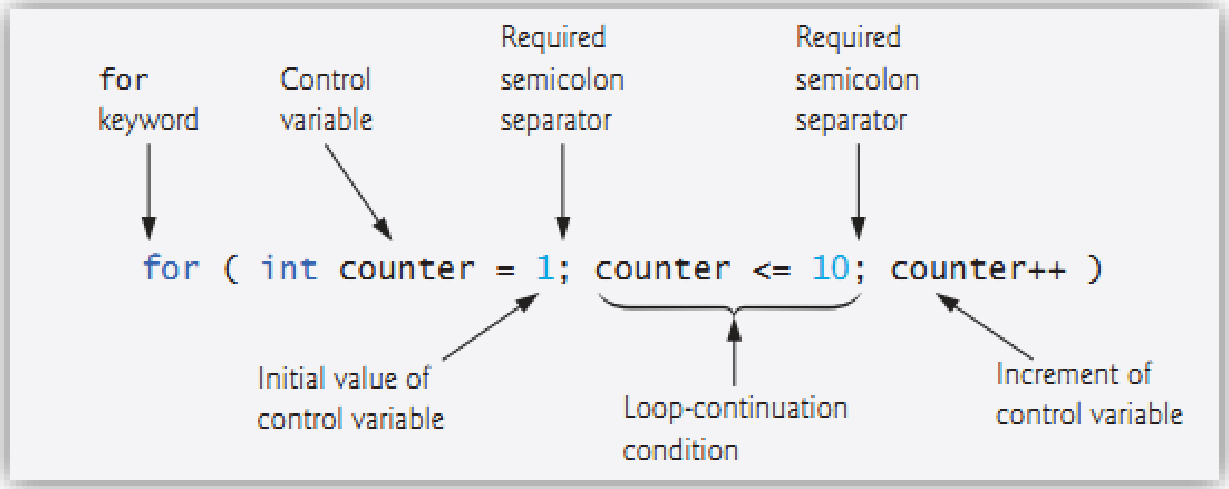
for keyword
Control variable
Required semicolon separator
Required semicolon separator
Initial value of control variable
Loop-continuation condition
Increment of control variable



```
// for statement header includes initialization,
// loop-continuation condition and increment
for ( int counter = 1; counter <= 10; counter++ )
   System.out.printf( "%d  ", counter );
```

???
??

FOR Repetition Statement



for keyword

Control variable

Required semicolon separator

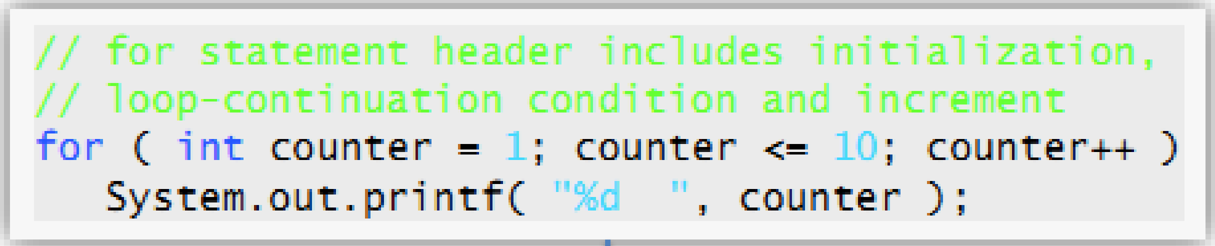
Required semicolon separator

```
for ( int counter = 1; counter <= 10; counter++ )
```

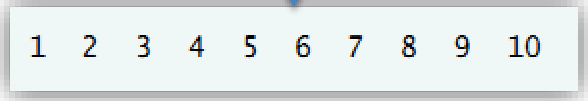
Initial value of control variable

Loop-continuation condition

Increment of control variable

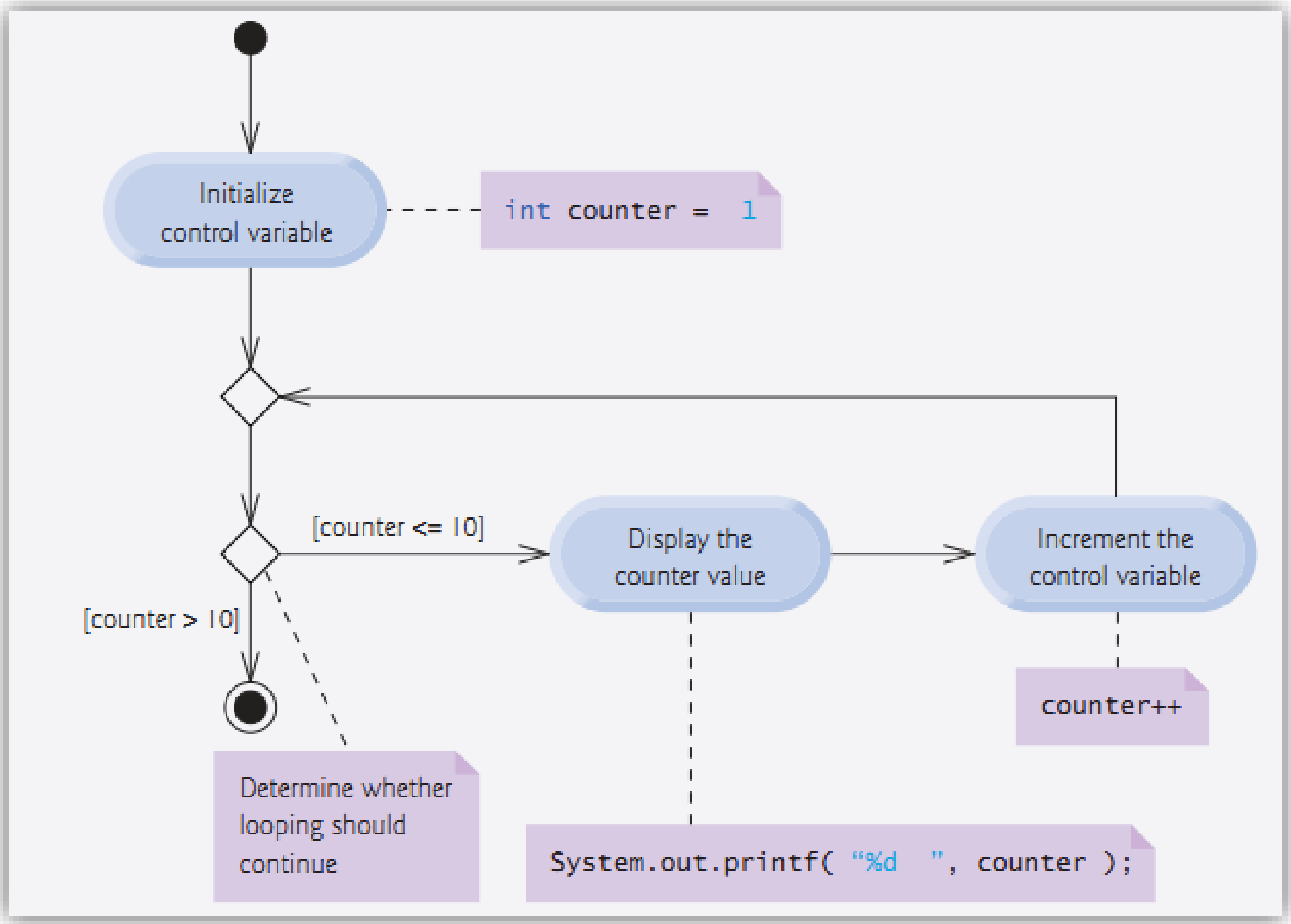


```
// for statement header includes initialization,
// loop-continuation condition and increment
for ( int counter = 1; counter <= 10; counter++ )
   System.out.printf( "%d  ", counter );
```



```
1  2  3  4  5  6  7  8  9  10
```

FOR Repetition Statement


Initialize control variable
int counter = 1
Display the counter value
Increment the control variable
[counter <= 10]
[counter > 10]
counter++
Determine whether looping should continue
System.out.printf("%d ", counter);

FOR Statements header examples

Vary the control variable from **1 to 100 in increments of 1**

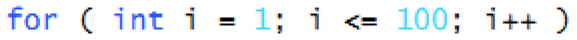
Vary the control variable from **100 to 1 in decrements of 1**

Vary the control variable from **7 to 77 in increments of 7**

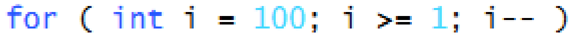
Vary the control variable from **20 to 2 in decrements of 2**

Vary the control variable over the following sequence of values: **?, ?, ?, ?, ?, ?, ?**

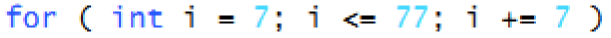
Vary the control variable over the following sequence of values: **?, ?, ?, ?, ?,?, ?, ?, ?, ?**



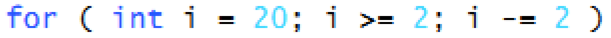
```
for ( int i = 1; i <= 100; i++ )
```



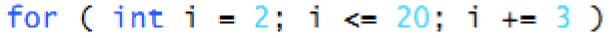
```
for ( int i = 100; i >= 1; i-- )
```



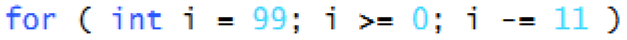
```
for ( int i = 7; i <= 77; i += 7 )
```



```
for ( int i = 20; i >= 2; i -= 2 )
```



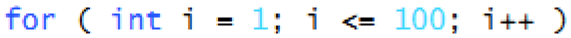
```
for ( int i = 2; i <= 20; i += 3 )
```



```
for ( int i = 99; i >= 0; i -= 11 )
```

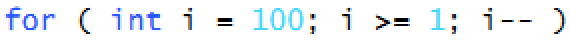
FOR Statements header examples

Vary the control variable from **1 to 100 in increments of 1**



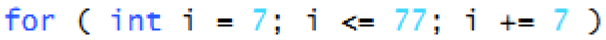
```
for ( int i = 1; i <= 100; i++ )
```

Vary the control variable from **100 to 1 in decrements of 1**



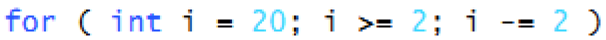
```
for ( int i = 100; i >= 1; i-- )
```

Vary the control variable from **7 to 77 in increments of 7**



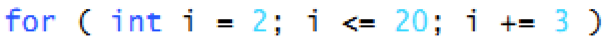
```
for ( int i = 7; i <= 77; i += 7 )
```

Vary the control variable from **20 to 2 in decrements of 2**



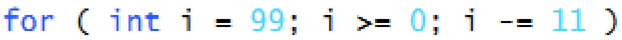
```
for ( int i = 20; i >= 2; i -= 2 )
```

Vary the control variable over the following sequence of values: **2, 5, 8, 11, 14, 17, 20**



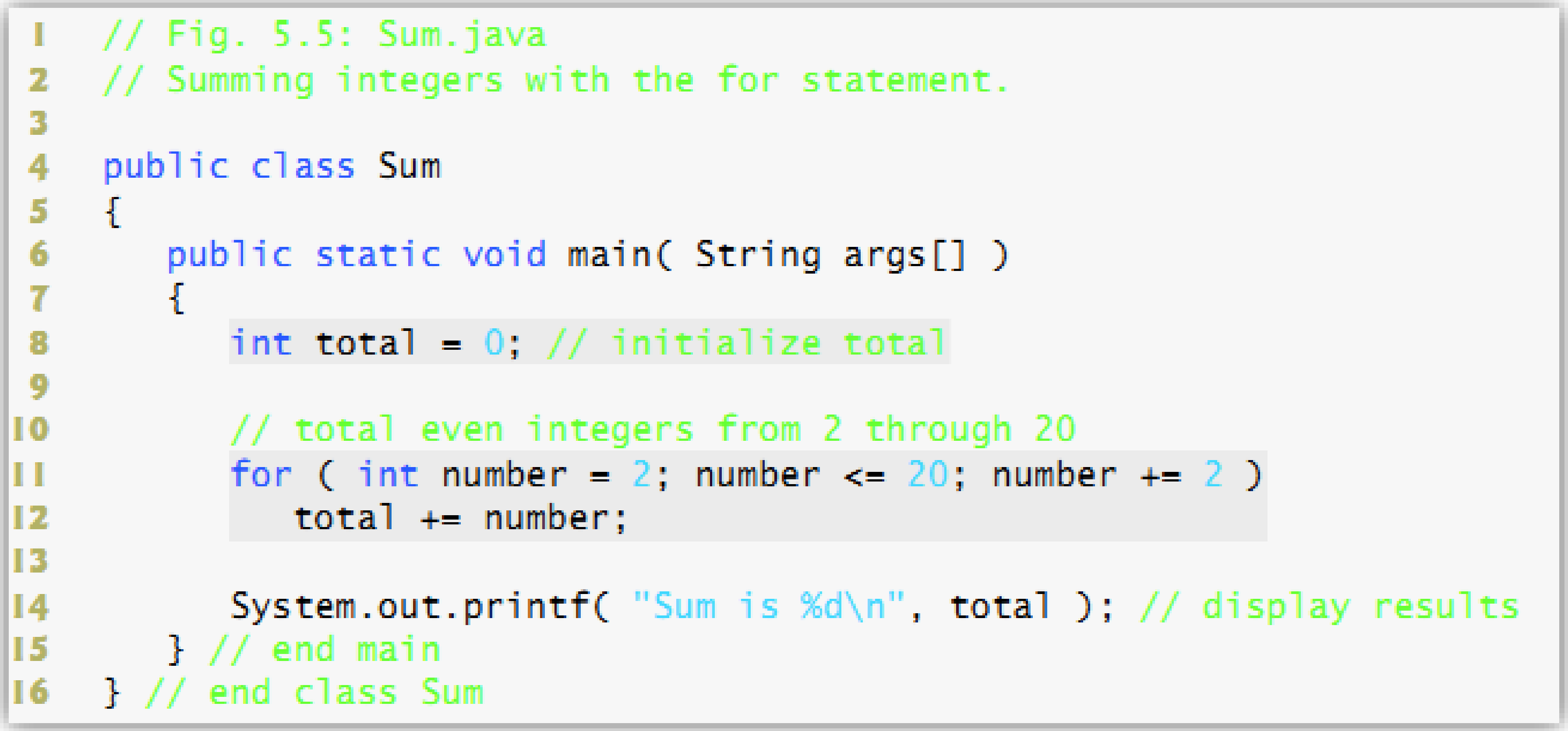
```
for ( int i = 2; i <= 20; i += 3 )
```

Vary the control variable over the following sequence of values: **99, 88, 77, 66, 55,44, 33, 22, 11, 0**



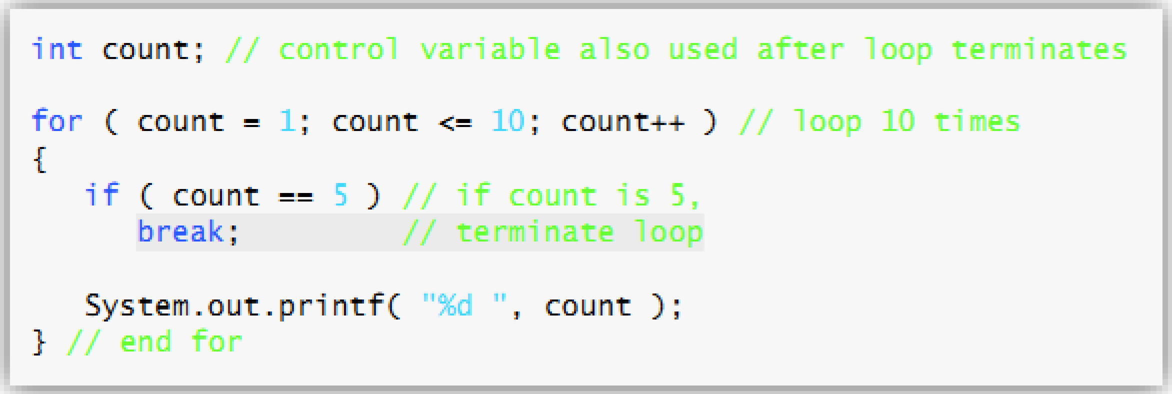
```
for ( int i = 99; i >= 0; i -= 11 )
```

FOR Statement example



```java
// Fig. 5.5: Sum.java
// Summing integers with the for statement.

public class Sum
{
   public static void main( String args[] )
   {
      int total = 0; // initialize total

      // total even integers from 2 through 20
      for ( int number = 2; number <= 20; number += 2 )
         total += number;

      System.out.printf( "Sum is %d\n", total ); // display results
   } // end main
} // end class Sum
```

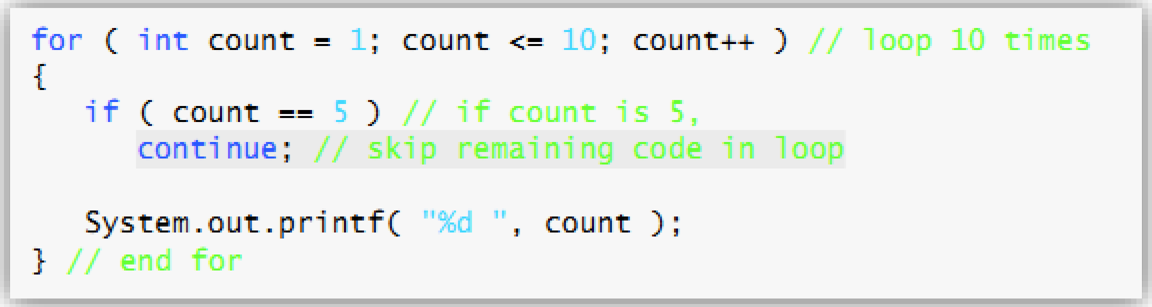
BREAK and CONTINUE Statements



```
int count; // control variable also used after loop terminates

for ( count = 1; count <= 10; count++ ) // loop 10 times
{
   if ( count == 5 ) // if count is 5,
      break;         // terminate loop

   System.out.printf( "%d ", count );
} // end for
```

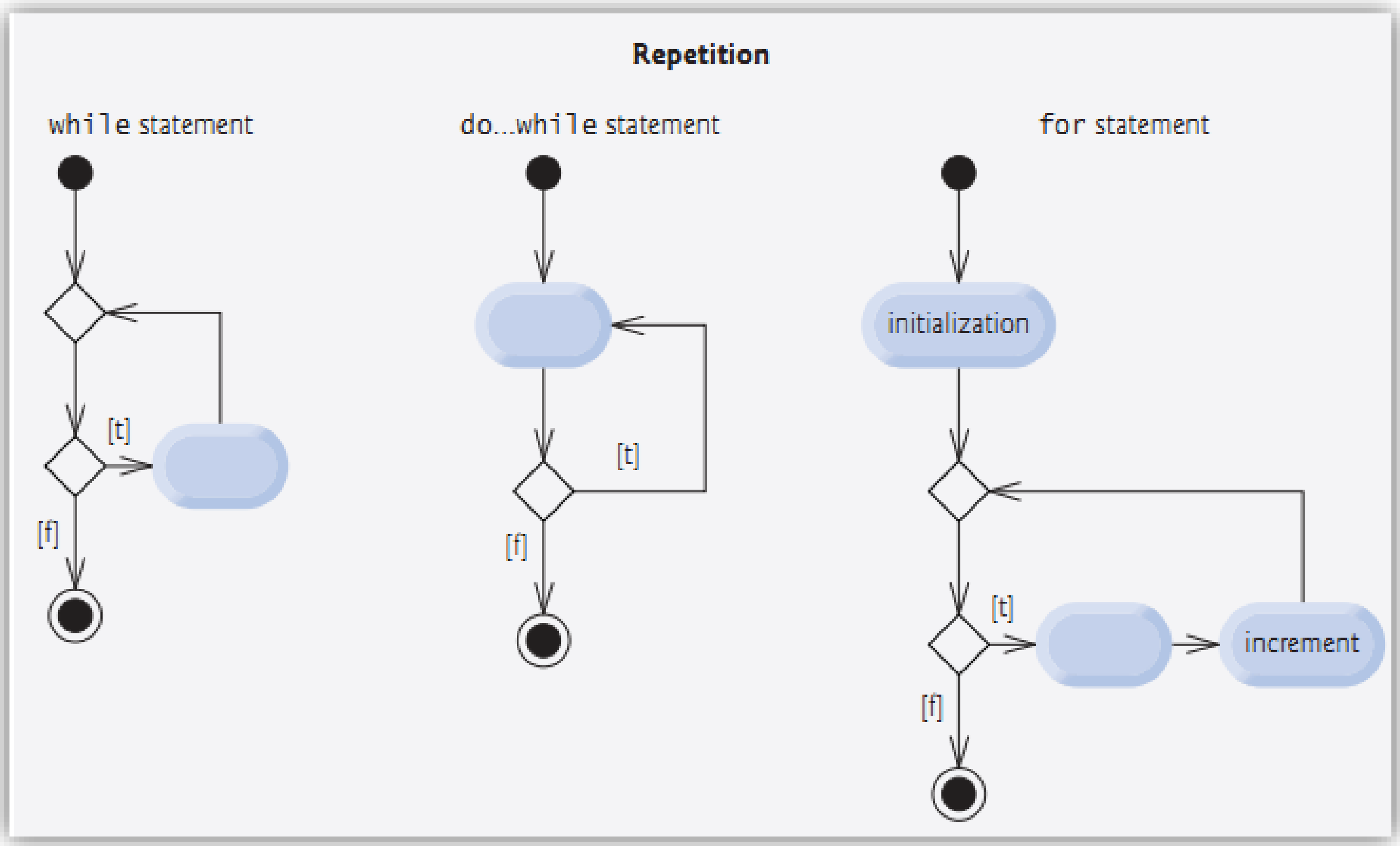
Break

Continue



```
for ( int count = 1; count <= 10; count++ ) // loop 10 times
{
   if ( count == 5 ) // if count is 5,
      continue; // skip remaining code in loop

   System.out.printf( "%d ", count );
} // end for
```

Summary


Repetition
while statement
[t]
[f]
do...while statement
[t]
[f]
for statement
initialization
[t]
increment
[f]

5. Exercise

Exercise

Write 4 programs. Specifically, they must…

1. determine if a number is prime or not.
2. print the prime numbers between 1 and 1000.
3. print the leap years between 2000 and 2100. A year is leap if it is divisible by 400 or if it is not divisible by 100 but it is divisible by 4.
4. print the minimum number of bills and coins (in the Colombian currency) that represent a given amount of money.

References

- [Deitel] H.M. Deitel and P.J. Deitel, *Java How to Program: Early Objects Version*, Prentice Hall, 2009.
- **Code Conventions for the Java Programming Language**, available at http://java.sun.com/docs/codeconv/CodeConventions.pdf
- **Oracle – Java Lesson: Language Basics**
  - http://download.oracle.com/javase/tutorial/java/nutsandbolts/index.html