UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica etsinf

Unit 3. Introduction to control structures and subprogramming

Introduction to Computer Science and Computer Programming
Introducción a la Informática y la Programación (IIP)

Year 2017/2018

Departamento de Sistemas Informáticos y Computación

Contents

Contents

Introduction and motivation

- Until this moment, we considered the code as simple sequences of instructions
- Most real problems require to take decisions depending on the situation, and choose among different subsequences
- Control structures allow to change this simple flow of instructions in order to obtain these features
- The two main control structures are decision and iteration
- Some complex operations are repeated in different parts of the code and they can be reused by using subprograms
- Code reuse improves programs (legibility, mainteinance, . . .)
- More in-depth description in Units 4, 5 and 6

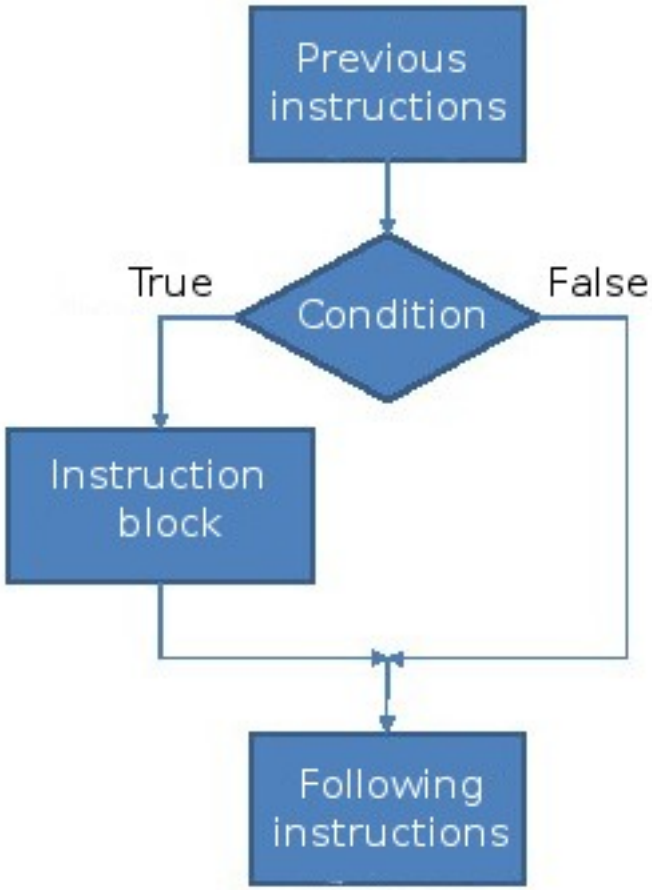
Contents

Simple decision: `if`

The most simple conditional in Java has the following structure:

```
if (B) S
```

- B is a condition
- S is any instruction or instruction block



Execution:

1. `B` is evaluated
2. When `B` is true, execute S
3. Continue with the instruction that follows the conditional

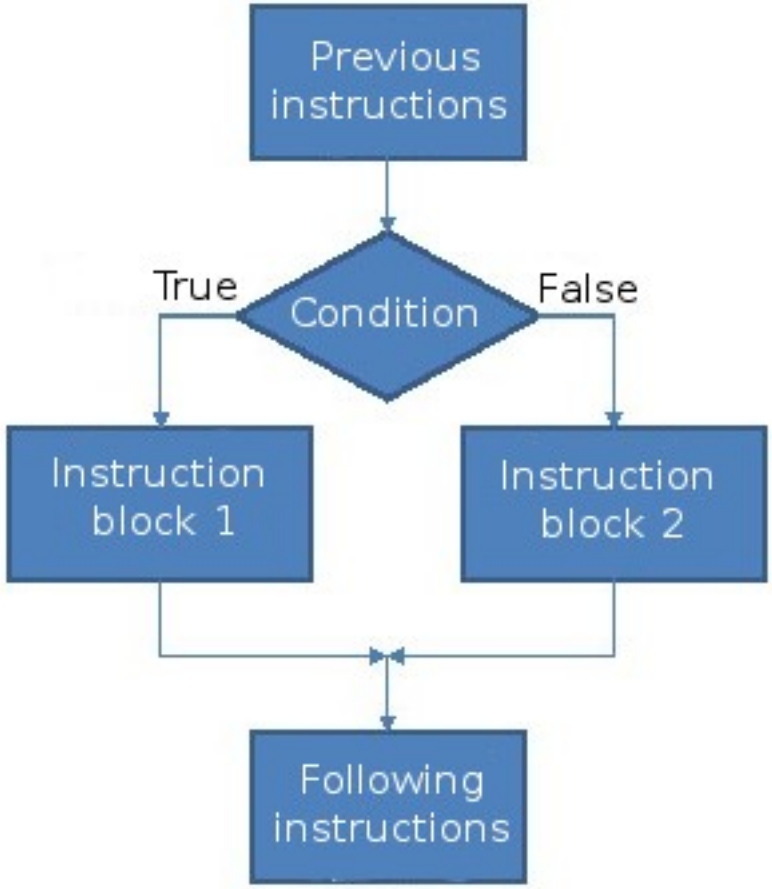
Contents

Double decision: `if-else`

Its general form is

```
if (B) S1 else S2
```

- B is a condition
- S_1 and S_2 are any instruction or instruction block



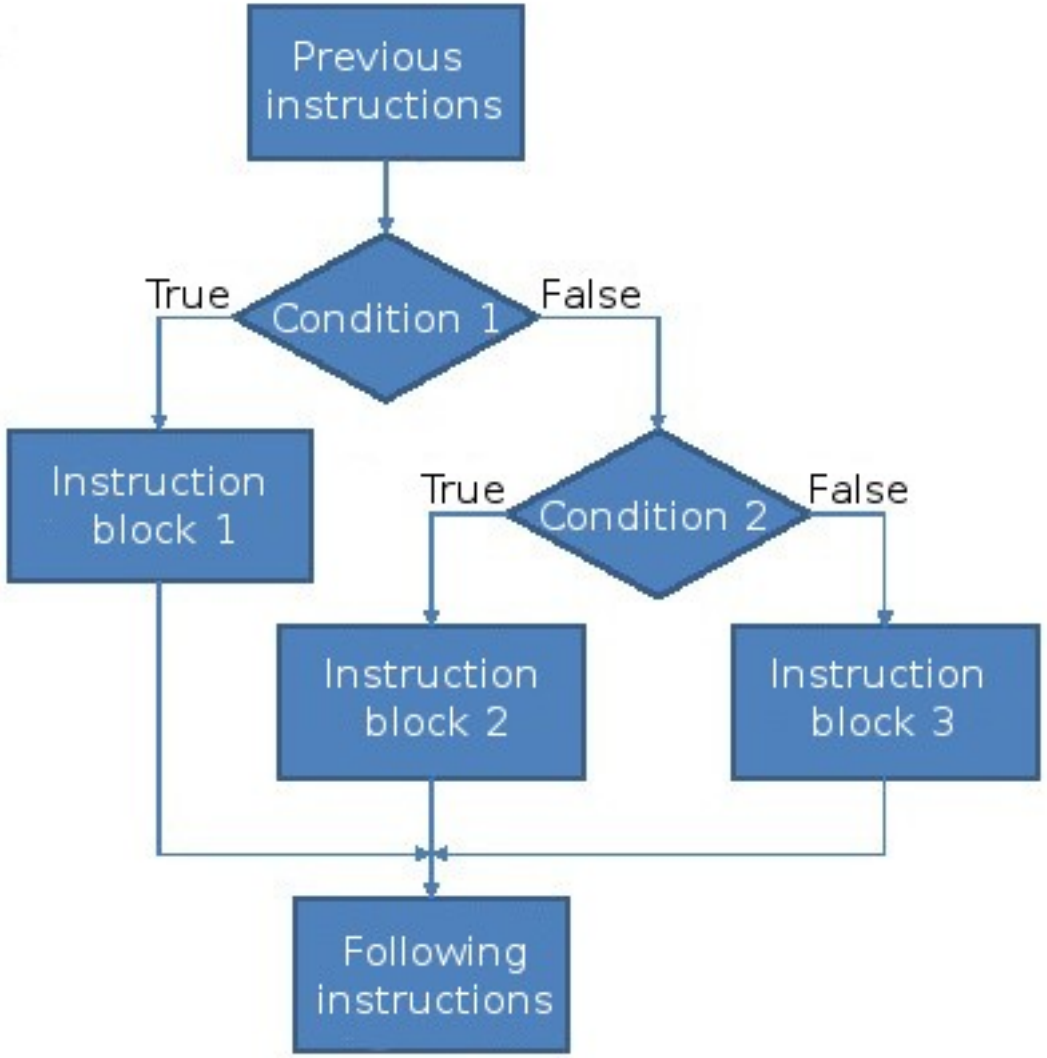
Execution:

1. `B` is evaluated
2. When `B` is true, execute S_1
3. When `B` is false, execute S_2
4. Continue with the instruction that follows the conditional

Double decision: `if-else`

`if` and `if-else` instructions can be ***nested*** in order to execute only one of several instructions or instruction blocks (this structure is called ***multiple*** `if-else`)

```
if (B1) S1
else if (B2) S2
else if (B3) S3
...
else if (Bn) Sn
else Sn+1
```



Double decision: `if-else`

In any of the forms of the conditional instruction `if ...else ...`, the instruction blocks can contain other conditional instructions: ***nested*** conditional instructions

Some examples:

```
if (B1)
    if (B2) S1;
    else S2;
```

```
if (B1)
    if (B2) S1;
else S2;
```

```
if (B1)
    if (B2) S1;
    else S2;
else
    if (B3) S3;
    else S4;
```

Contents

Conditional iteration: `while`

The simplest forms are:

```
while (condition)
  instruction;
```

```
while (condition) {
  instruction_1;
  instruction_2;
  ...
  instruction_n;
}
```

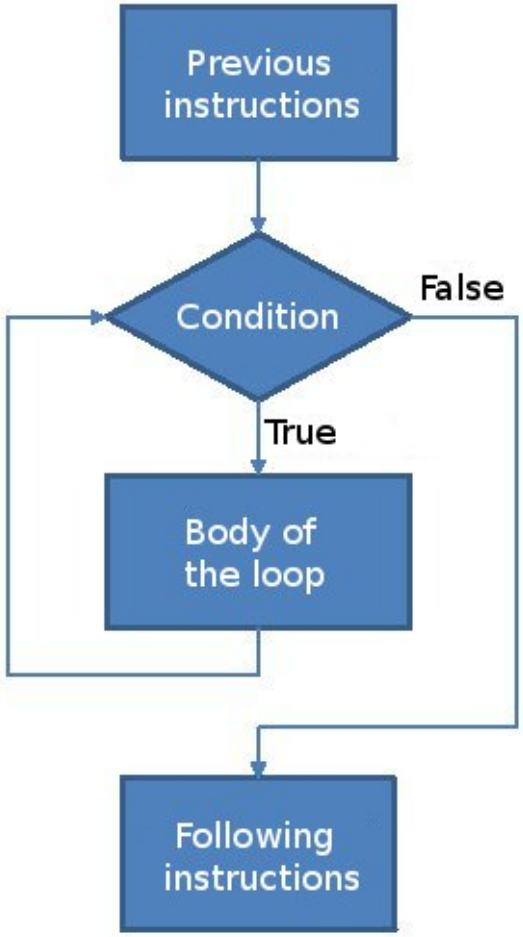
where `condition` is any boolean condition and the body of the loop is an instruction or a block of instructions

Conditional iteration: `while`

Execution

1. Evaluate `condition`
2. When it is `true`, execute the instructions of the body of the loop
3. Otherwise, the execution finishes
4. After stopping, the execution continues with the instruction that follows the loop

It is possible that the instructions of the body of the loop never get executed (0 iterations)



Contents

Subprogramming: `static` methods

Suppose the following program class:

```
public class ProgramTriangle {
  public static void main(String args[]) {
    double p1x = 2.5, p1y = 3;      // 1st vertex coordenates
    double p2x = 2.5, p2y = -1.2;   // 2nd vertex coordenates
    double p3x = -1.5, p3y = 1.4;   // 3rd vertex coordenates
    double dx, dy, side12, side23, side13, perimeter;

    System.out.println("Triangle with vertexes:\n(" + p1x + "," + p1y + ")");
    System.out.println("(" + p2x + "," + p2y + ")");
    System.out.println("(" + p3x + "," + p3y + ")");
    dx=p1x-p2x; dy=p1y-p2y; side12 = Math.sqrt(dx*dx + dy*dy);
    dx=p2x-p3x; dy=p2y-p3y; side23 = Math.sqrt(dx*dx + dy*dy);
    dx=p1x-p3x; dy=p1y-p3y; side13 = Math.sqrt(dx*dx + dy*dy);
    perimeter=side12+side23+side13;
    System.out.println("Perimeter: "+perimeter);
  }
}
```

The instructions in red are repeated for different data (p1 and p2, p2 and p3, p1 and p3)

Subprogramming: `static` methods

With ***subprogram***: `static` method

```
public class ProgramTriangle {
  public static void main(String args[]) {
    double p1x = 2.5, p1y = 3;      // 1st vertex coordenates
    double p2x = 2.5, p2y = -1.2;   // 2nd vertex coordenates
    double p3x = -1.5, p3y = 1.4;   // 3rd vertex coordenates
    double side12, side23, side13, perimeter;

    System.out.println("Triangle with vertexes:\n(" + p1x + "," + p1y + ")");
    System.out.println("(" + p2x + "," + p2y + ")");
    System.out.println("(" + p3x + "," + p3y + ")");
    side12 = dist(p1x,p1y,p2x,p2y); side23 = dist(p2x,p2y,p3x,p3y); side13 = dist(p1x,p1y,p3x,p3y);
    perimeter=side12+side23+side13; System.out.println("Perimeter: "+perimeter);
  }

  /** Calculate distance between two points */
  public static double dist(double px, double py, double qx, double qy) {
    double dx=px-qx; double dy=py-qy;
    return Math.sqrt(dx*dx + dy*dy);
  }
}
```

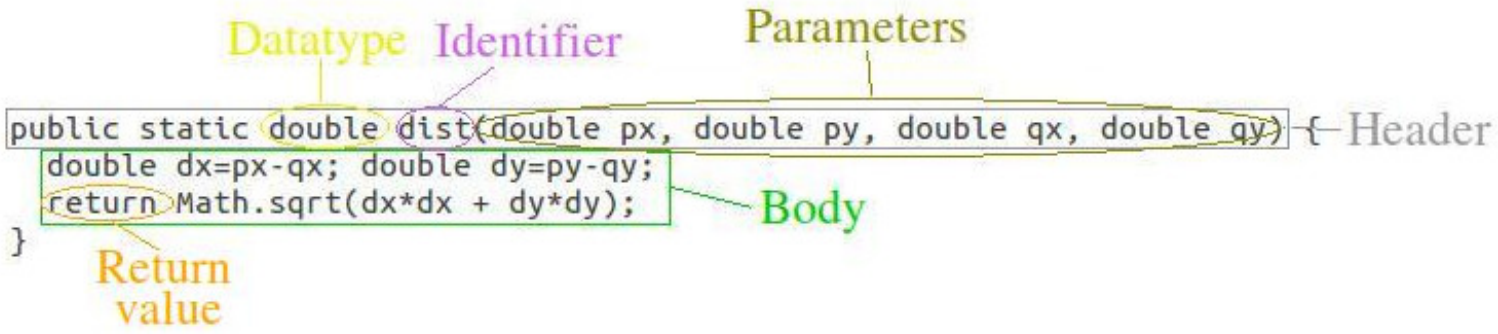
Improves legibility and security, eases maintenance

Subprogramming: `static` methods

Method definition

Elements of a method:

- ***Header***: name, input/output data
- ***Body***: instructions to be executed



Subprogramming: `static` methods

Method definition

- ***Datatype***: that of the result of the method
  - `void`: no result
  - Primitive or reference datatype (`int`, `double`, `String`, ...)
- ***Parameters***: input data
  - Value given when method is used
  - Be careful with syntax!

```
(int a, b, double x)        (int a, int b, double x)
```

WRONG CORRECT

- ***Body***: instruction to be executed
  - Any type of instructions: var declaration, assignment, control structures
  - Only parameters and locally declared variables can be used
  - `return` to provide final result if necessary

Subprogramming: `static` methods

Method use

Methods do not get executed by themselves, they have to be ***called***

Method call: `methodName(par1, par2, ..., parn)`

- `methodName`: identifier
- `pari`: input parameters
  - Expressions with same datatype that corresponding parameter in header
  - They get evaluated and header parameters get these values

```
public class ProgramTriangle {
  public static void main(String args[]) {
    ...
    side12 = dist(p1x,p1y,p2x,p2y);  // Call to method
    ...
  }
  public static double dist(double px, double py, double qx, double qy) {
    ...
  }
}
```

Subprogramming: `static` methods

Pass of parameters

Java uses only ***pass by value***: original values are not modified

```
public static void increment(int a) {
  a = a + 1;
}
public static void main(String [] args) {
  int a = 5;
  increment(a);
  System.out.println(a);   // Output will be 5
}
```

Warning!: When passing references, internal modifications are visible

```
public static void incrPoint(Point p) {
  p = new Point(p.x + 1, p.y);
}
public static void main(String [] args) {
  Point p = new Point(2, 3);
  incrPoint(p);
  System.out.println(p.x);
  // Output will be 2
}
```

```
public static void incrPoint(Point p) {
  p.x = p.x + 1;
}
public static void main(String [] args) {
   Point p = new Point(2, 3);
   incrPoint(p);
   System.out.println(p.x);
   // Output will be 3
}
```