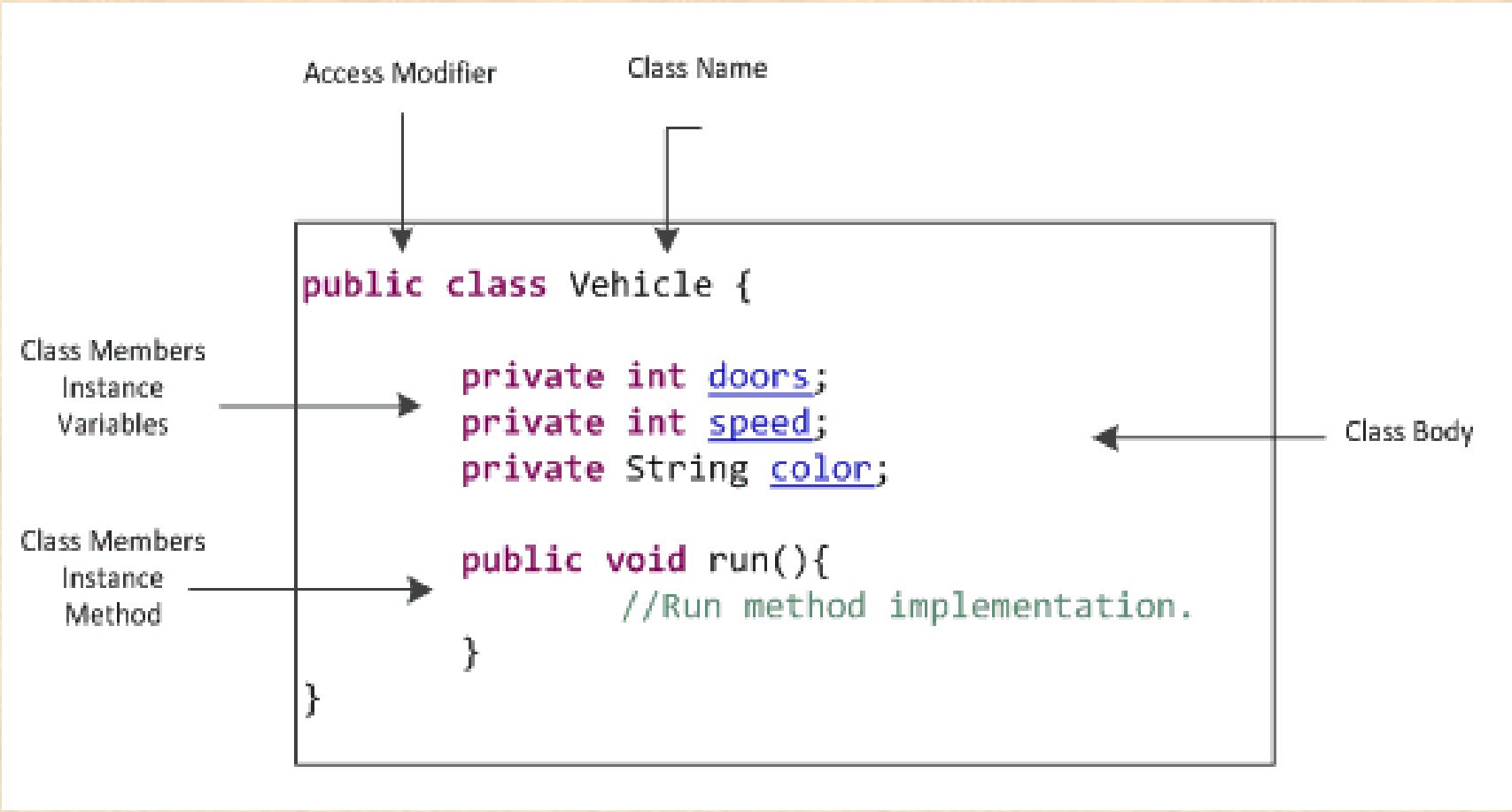
Classes

- A class is a **user defined data type** and it's **declared** by the use of **class keyword**.
- It is a template or blueprint from which objects are created.
- The class body is enclosed between curly braces { and }.
- The data or variables, defined within a class and outside method are called **instance variables**.
- The methods and variables defined within a class are called **members of the class**.
- Method which is **defined inside the class** is **instance method.**

Synatax:

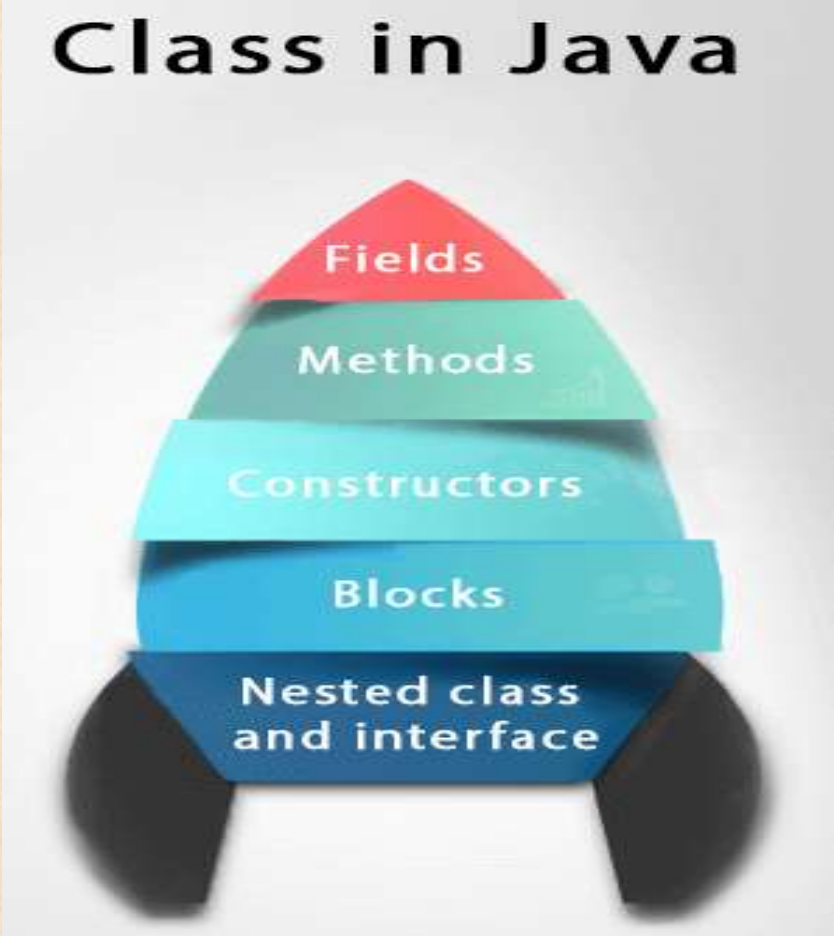
```
class ClassName
{
        //class code
}
```

Class Example



Class

- Fields
- Methods
- Constructors
- Blocks
- Nested class and
- interface



Method

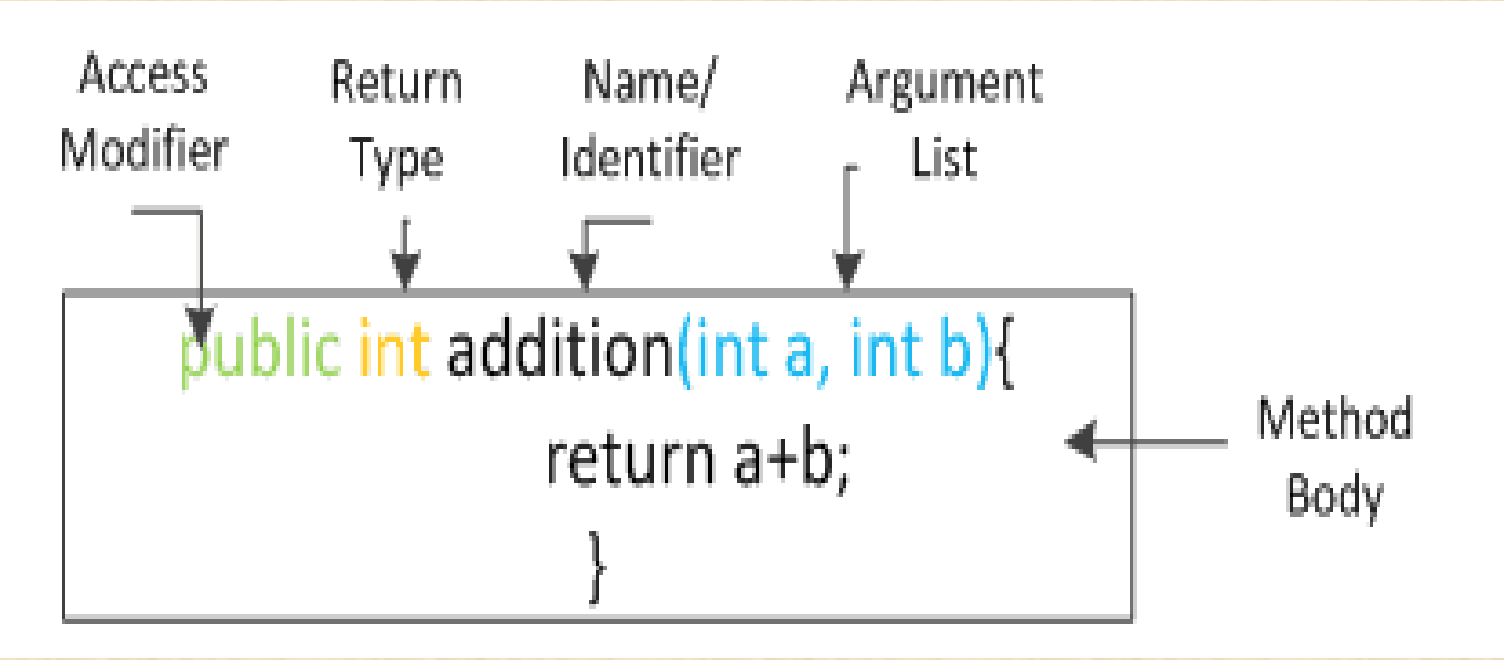
- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used **to perform certain actions**, and they are also known as **functions**.
- Why use methods? To reuse code: define the code once, and use it many times.

Method Example

- **Sysntax:**

```
AccessModifier returntype functionname (arguments)
{
    // code of function to perform specific task
}
```

Example:



Example program – class and function

```
class Student{
 int rollno;
 String name;
 void insert(int r, String n)
{
  rollno=r;
  name=n;
 }
 void display()
{
   System.out.println(rollno+" "+name);
}
```

Example program – class and function (Contd..)

```
class Test{
 public static void main(String args[])
{
  Student s1=new Student();
  Student s2=new Student();
  s1.insert(111,“Arun");
  s2.insert(222,“Babu");
  s1.display();
  s2.display();
 }
}
```

Conditional statements

Conditional statements

- **Conditional statements in a computer program support decisions based on a certain condition.**
- **If the condition is met, or "true," a certain piece of code is executed.**

Conditional statement

It is used to test the condition

- If
- If else
- Ladder
- Nested if else
- Switch

Looping

- for
- while
- do while

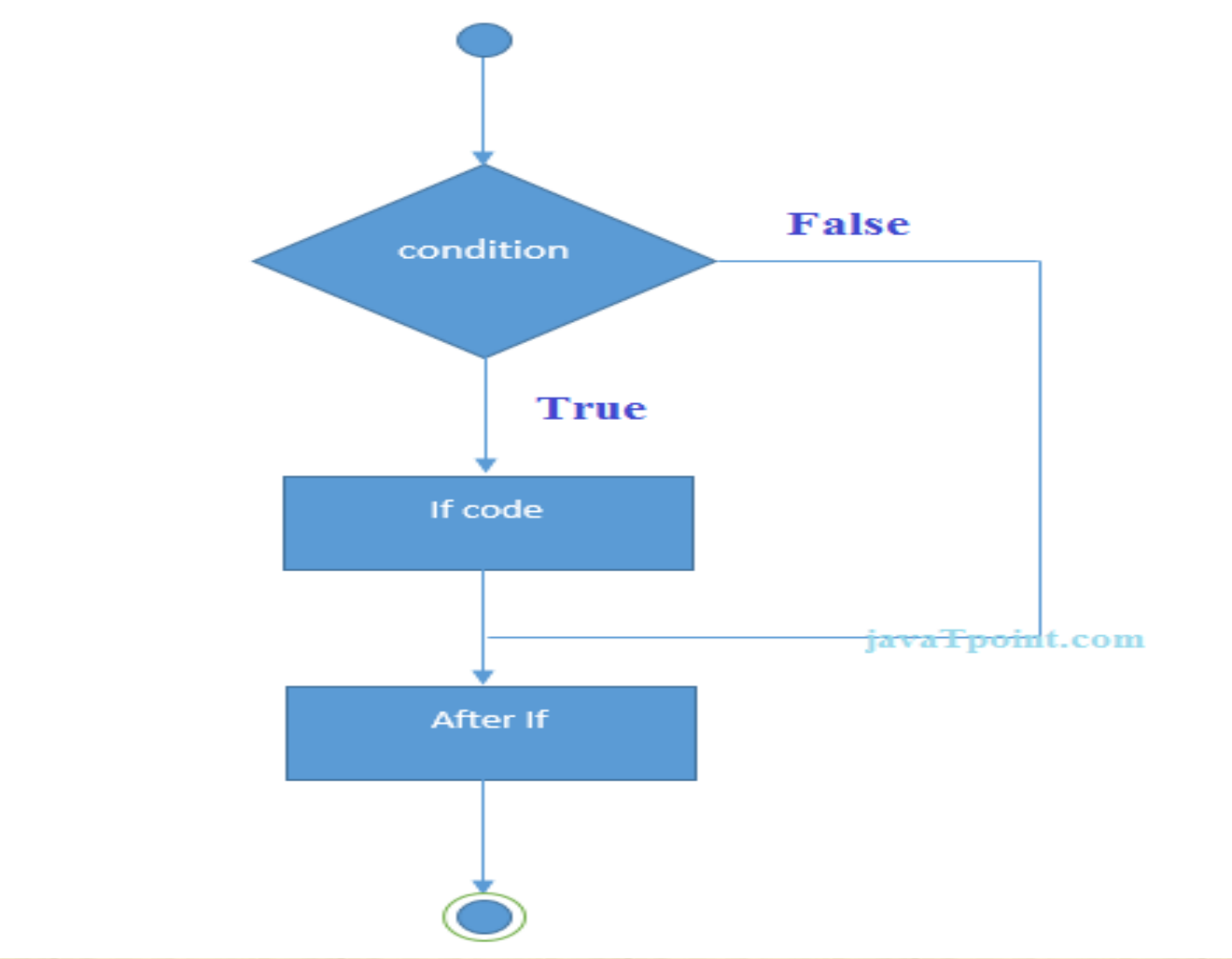
if

- The Java if statement tests the condition.
- If the if condition is true the body of the if block will be executed **(It executes the if block if condition is true**.)

Syntax:

```
if(condition)
{
        //code to be executed  (true statements)
}
```

Flow Structure



Example

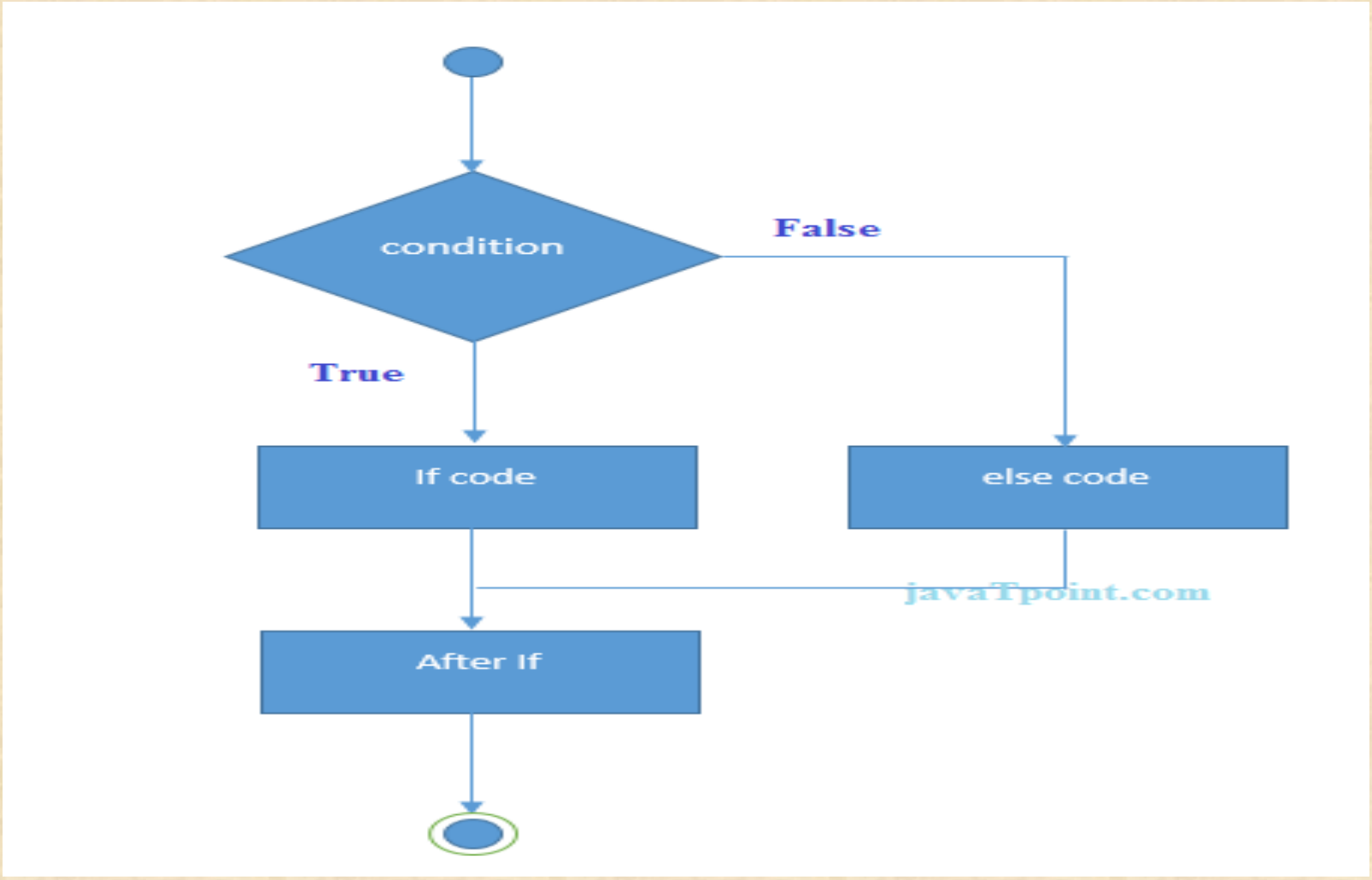
```
public class IfExample
{
public static void main(String[] args) {
  int mark=60;
  if(mark>=50)
  {
        System.out.print(“Pass Mark");   }
  }
}
```

if - else

- If the if condition is true the if block will be executed otherwise the else block will be executed.
- **It executes the if block if condition is true otherwise else block is executed.**

Syntax:

```
if(condition)
{
        //code  (if condition is true )
}
else
{
        //code  (if condition is false )
}
```


condition
False
True
If code
else code
javaTpoint.com
After If

Example

```
public class IfelseExample
{
public static void main(String[] args) {
  int mark=60;
  if(mark>=50)
  {
        System.out.print(“Pass Mark");   }
  }
  else
 {
        System.out.print(“Fail Mark");
  }
}
}
```

Ladder if else

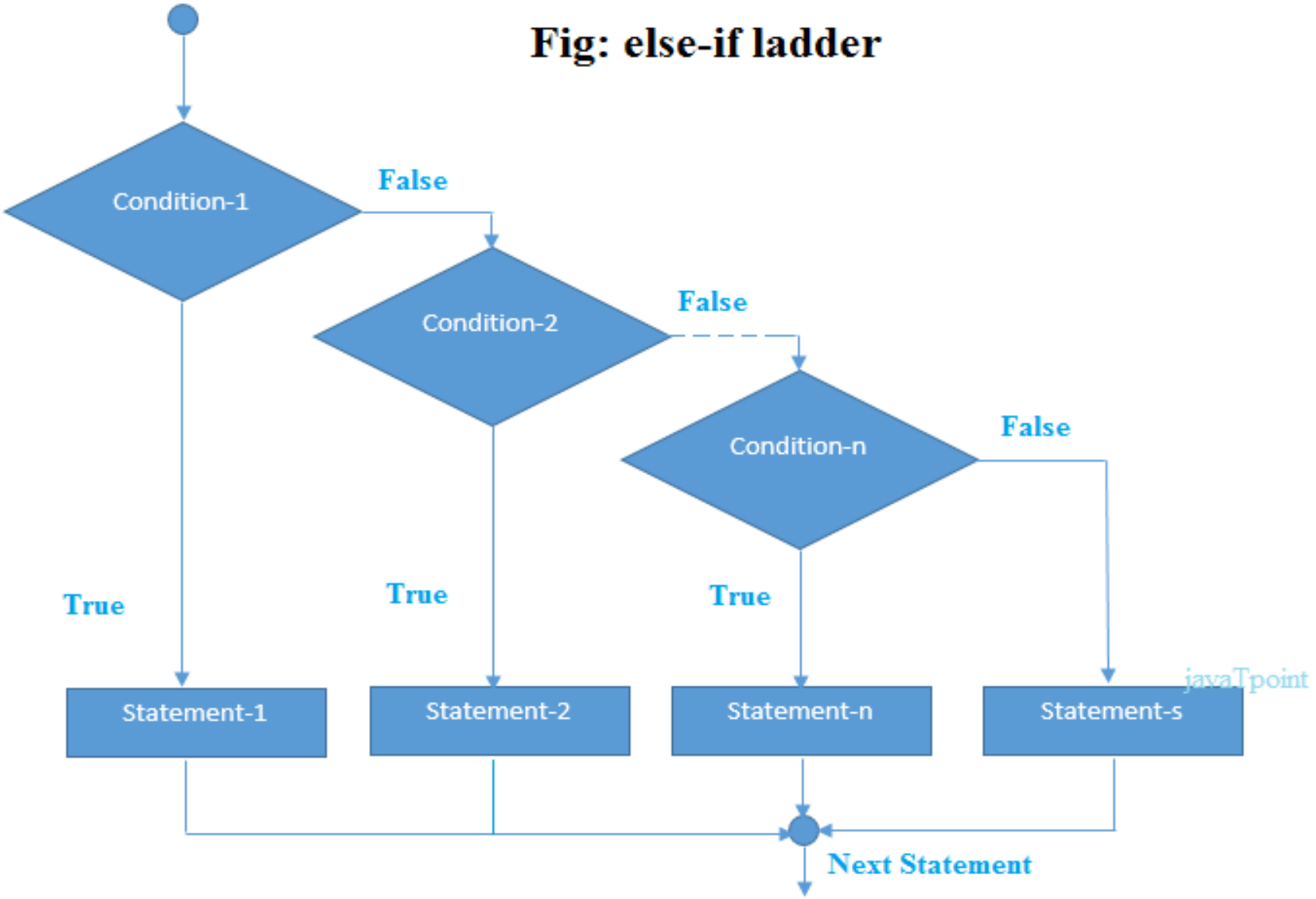
- First if the first condition is true that true block will be executed. and if it false it will go and check next immediate else if block until condition is getting true.

syntax

```
if(condition1){
        //code to be executed if condition1 is true
}
else if(condition2){
        //code to be executed if condition2 is true
}
else if(condition3){
        //code to be executed if condition3 is true
}
 else{
        //code to be executed if all the conditions are false

}
```

Fig: else-if ladder



Example

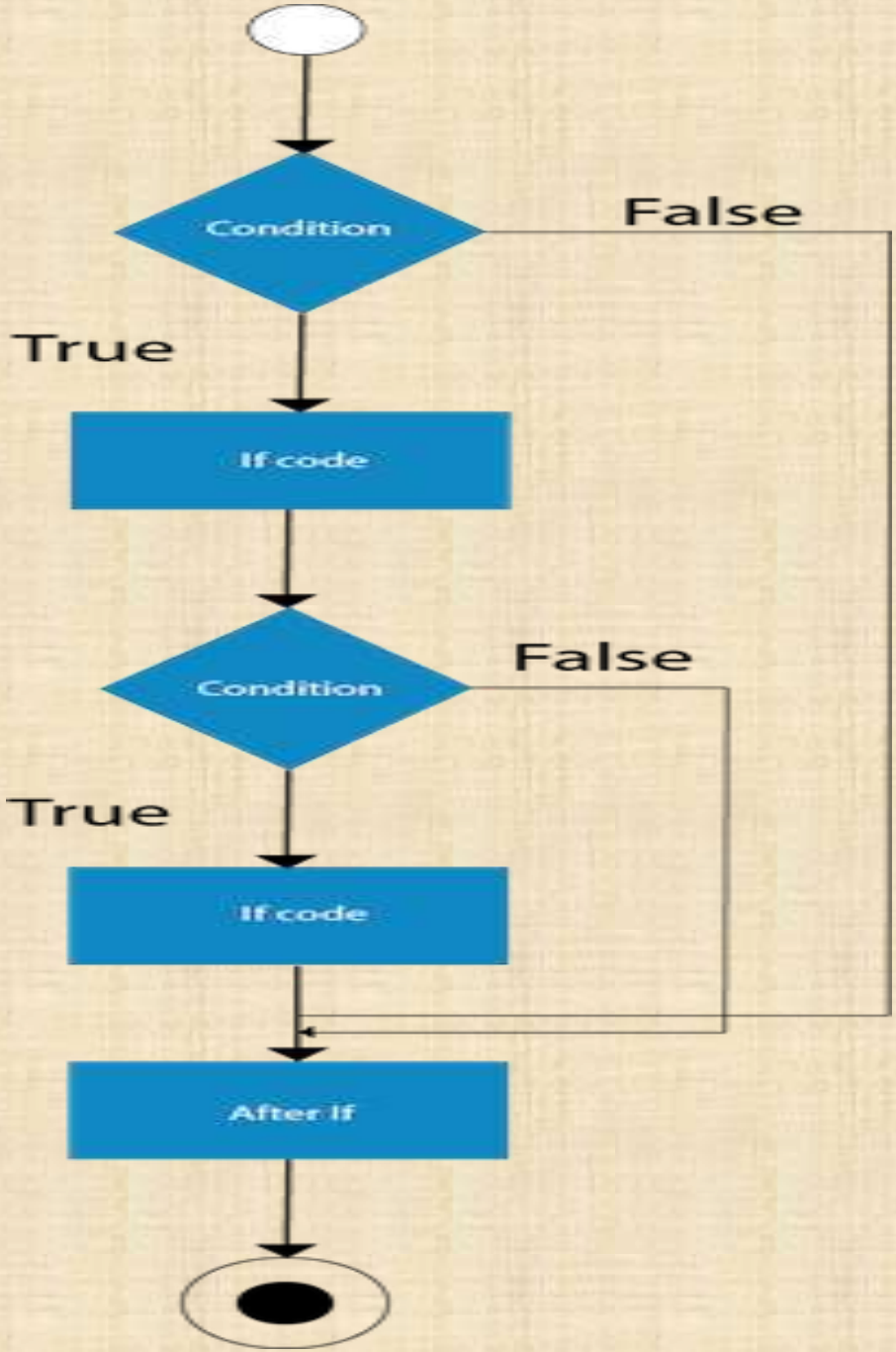
```
public class IfelseladderExample
{
public static void main(String[] args) {
  int mark=60;
  if(mark<50)
  {
            System.out.print(“fail Mark");   }
  }
else if(mark>90&& mark<=100)
{
             System.out.print(“S grade");
}
else if(mark>80 && mark<=90)
{
            System.out.print(“A grade");
}
} }
```

Nested if (or) if else statement

- The nested if statement represents the *if block within another if block*.
- Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition)
{
    //code to be executed
        if(condition)
        {
                //code to be executed
         }
}
```


Condition
False
True
If code
Condition
False
True
If code
After If

Example

```
public class nestedifExample
{
public static void main(String[] args) {
  int mark=85;
  if(mark>=50)
  {         System.out.print(“Pass Mark");

             if(mark>90&& mark<=100)     // Nested if statement 1
             {
                        System.out.print(“S grade");
             }
             else if(mark>80 && mark<=90)    // Nested if statement 2
             {
                        System.out.print(“A grade");
             }
  }
  }
}
```

Example

```
public class JavaNestedIfExample {
public static void main(String[] args) {
    int age=20;
   int weight=80;
      if(age>=18)
         {
              if(weight>50){
                 System.out.println("You are eligible to d
                                                  onate blood");
                              }
         }
}}
```

Switch

- The Java switch expression must be of byte, short, int, long (with its Wrapper type), enums and string.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.

```
switch(expression)
{
case value1:
        //code to be executed;
        break;  //optional
case value2:
        //code to be executed;
        break;  //optional
......

default:
        code to be executed if all cases are not matched;
}
```

Example

```
public class SwitchExample {
public static void main(String[] args) {
 int number=3;
        switch(number){
       case 1:
        System.out.println(“Monday");
        break;
        case 2: System.out.println(“Tuesday");
        break;
```

```
case 3: System.out.println(“Wednesday");
break;
case 4: System.out.println(“Thursday”);
break;
case 5: System.out.println(“Friday”);
break;
case 6: System.out.println(“Saturday”);
break;
Case 7:System.out.println(“Sunday”);
break;
  default:System.out.println(“enter the number between 1 and 7");
    }
}
}
```

Looping

- In programming languages, loops are used to execute a set of instructions/functions **repeatedly when some conditions become true.** There are three types of loops in java.

Types:

- for loop
- while loop
- do-while loop

for loop

- The Java *for loop* is used to iterate a part of the program several times.

- If the number of iteration is fixed, it is recommended to use for loop.

for loop Syntax

```
for(initialization; test condition; incr/decr)
{
        // code to be executed
}
```

Syntax for infinitive loop:

```
for(;;)
{
        //code to be executed
}
```

Example

```
public class ForEx
 {
   public static void main(String[] args)
   {
      for(int i=1;i<=10;i++)
          {
                  System.out.println(i);
          }
   }
}
```

While loop

- The Java *while loop* is used to iterate a part of the program several times

- If the number of iteration is not fixed, it is recommended to use while loop.

Syntax

```
// Initialization
while(condition)
{
        //code to be executed
        // increment or decrement
}
```

Example program

```
public class WhileEx
{
   public static void main(String[] args) {
          int i=1;
         while(i<=10) //
         {
                  System.out.println(i);
                   i++;
           }
}
}
```

do-while

- If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop

Syntax:

```
do
{
        //code to be executed
        // increment or decrement

}while(condition);
```

Example

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

break

- The Java *break* is used to **break loop or switch statement**. **It breaks the current flow of the program** at specified condition.
- In case of **inner loop, it breaks only inner loop**.

continue

- The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
- In case of an inner loop, it continues the inner loop only.