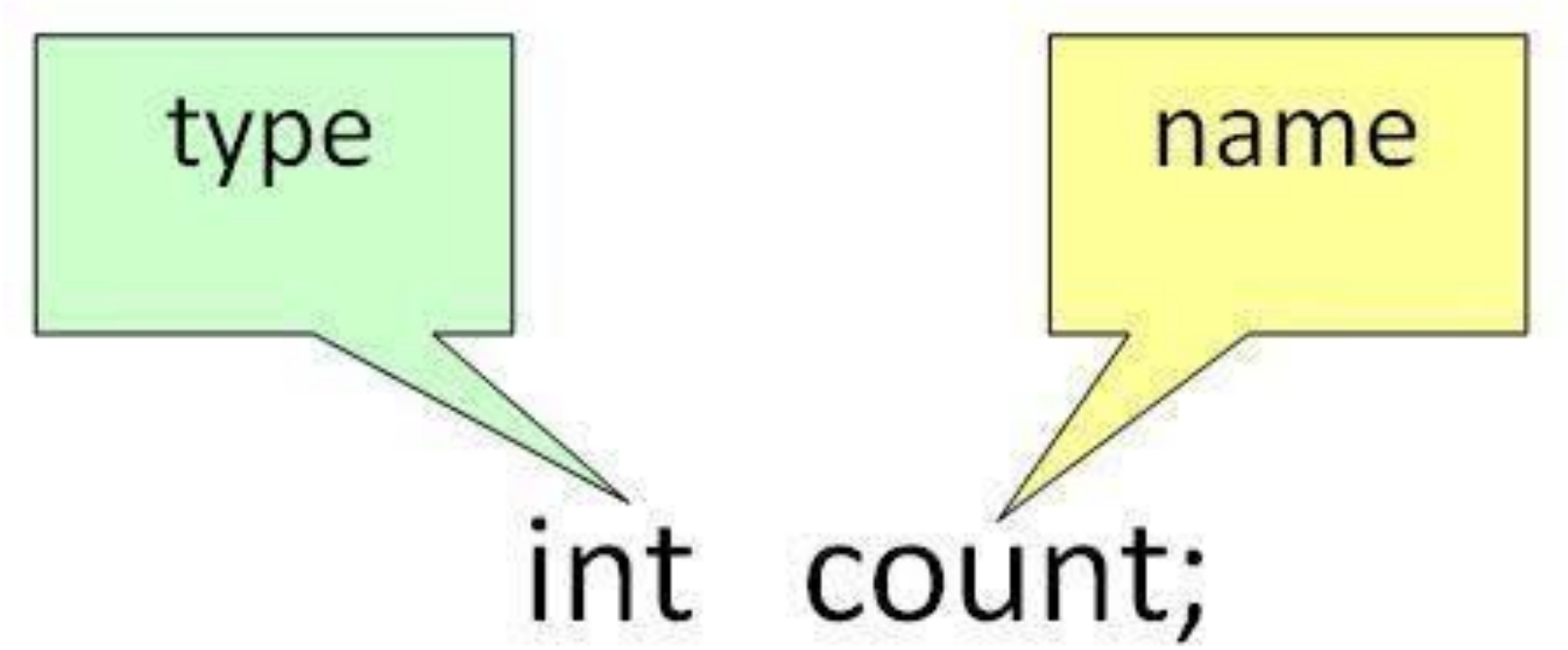
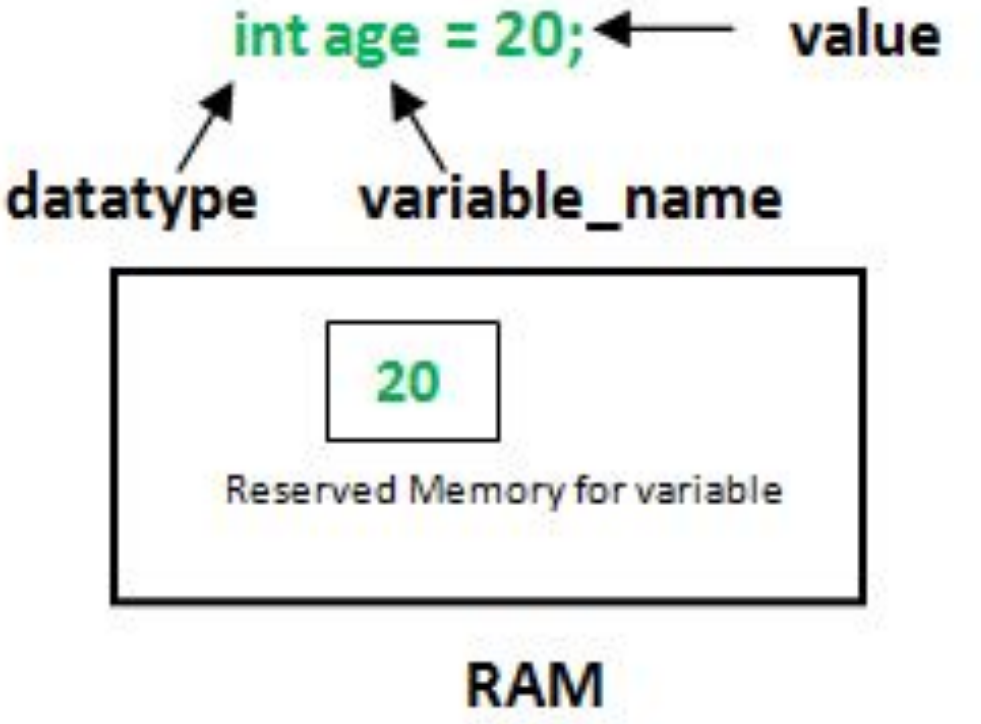
Variables in Java

Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. A variable is a name given to a memory location.

How to declare variables?



How to initialize variables?

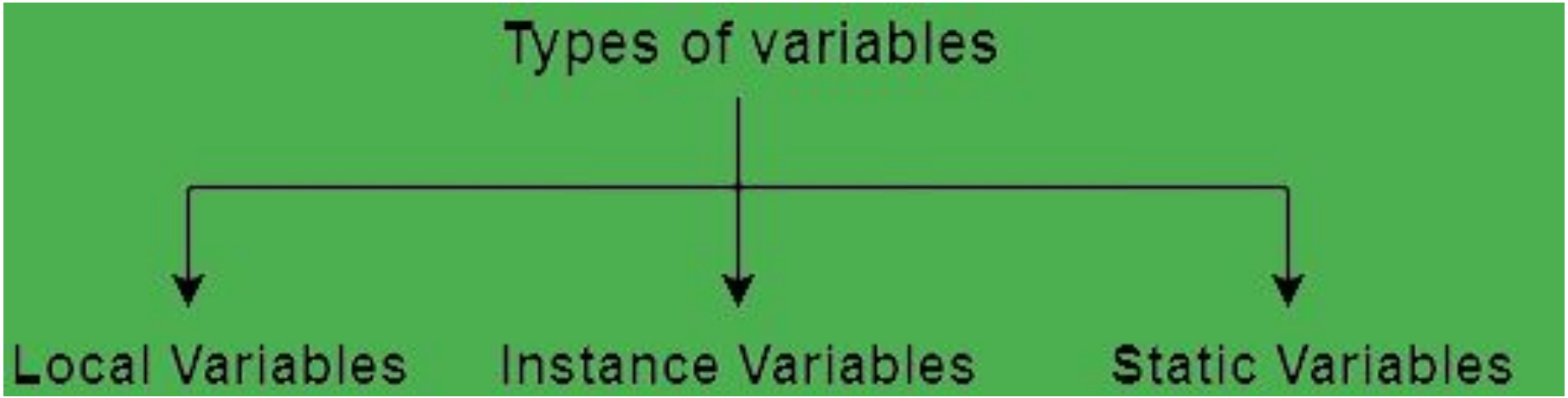


```
float simpleInterest;
// Declaring float variable
```

```
int time = 10, speed = 20;
// Declaring and Initializing integer variable
```

```
char var = 'h';
// Declaring and Initializing character variable
```

Types of Variables in Java



Identifiers

- All the Java components —classes, variables, and methods— need names.
- In Java these names are called *identifiers*, and, there are **rules for what constitutes a** legal Java identifier.
- Beyond what's *legal, Java programmers (and* Sun) have created ***conventions for naming*** methods, variables, and classes.

Rules for naming Identifiers

1. Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_). Identifiers cannot start with a number!
2. After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
3. Identifier can be any length. In practice, there is no limit to the number of characters an identifier can contain.
4. You can't use a Java keyword as an identifier. (See "Java Keywords," for a list of reserved words).
5. Identifiers in Java are case-sensitive; **amoud and Amoud are two** different identifiers.
6. Identifier cannot be true, false or null.

Keywords or Reserved words are the words in a language that are used for some internal process or represent some predefined actions. These words are therefore not allowed to use as variable names or objects.

Java keywords

abstract	else	long	synchronized	boolean	
break	byte	case	extends	final	
finally	native	new	package	private	
this	throw	throws	transient	catch	
char	class	float	for	Goto	
if	protected		public	return	
try	void	volatile	const	continue	
default	do	implements	import	instanceof	
int	Short	static	strictfp	super	
while	double		Interface	switch	

- Which of the following are valid Java identifiers? If invalid, give reason:

i. Double

ii. 2CS

iii. Big_N

iv. $67

v. Boolean

vi. _number

- Which of the following are valid Java identifiers? If invalid, give reason:

i. Int

ii. public

iii. private

iv. AMOUD

v. 2*K

vi. var one

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.

```java
public class Main {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

// I just got executed!
// I just got executed!
// I just got executed!
```

myMethod() is the name of the method

static means that the method belongs to the Main class and not an object of the Main class.

void means that this method does not have a return value.

```
public class Main
{    static void sum1(int a,int b){
    int c = a + b;
    System.out.println(c);
}
    public static void main(String[] args)
    {
        sum1(23,25);
    }
}
```

Let's learn about objects and how to access methods through objects .
(**OR** creating mathod without static keywords)

```java
public class Main
{     void sum1(int a,int b){
    int c = a + b;
    System.out.println(c);
}
    public static void main(String[] args)
    { Main obj = new Main();
        obj.sum1(23,25);
    }
}
```

Java Scanner Class

Java Scanner class allows the user to take input from the console. It belongs to java.util package. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.

Syntax -

```
Scanner obj-name =new Scanner(System.in);
```

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

```java
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
        Scanner sc =new Scanner(System.in);
        System.out.println("Enter first number");
        int a = sc.nextInt();
        System.out.println("Enter Second number");
        int b= sc.nextInt();
        int c = a+b;

        System.out.println("sum is " +c);
    }
}
```

Primitive Data Types

Data Type	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter or ASCII values

```
String myString = "Hey World";
```

Boolean Values

A boolean type is declared with the boolean keyword and can only take the values true or false.

```
boolean isJavaFun = true;
boolean isLifeEazy = false;
System.out.println(isJavaFun);
System.out.println(isLifeEazy);
```

O/P :- true
false

Boolean Expression

A Boolean expression is a Java expression that returns a Boolean value: true or false.

You can use a comparison operator, such as the greater than (>) operator to find out if an expression (or a variable) is true:

```
int x = 10;
int y = 9;
System.out.println(x > y);
```

O/P :- true

Java Operators

Operators are symbols that perform operations on variables and values.

For example, + is an operator used for addition, while * is also an operator used for multiplication.

1. **Java Arithmetic Operators**

Arithmetic operators are used to perform arithmetic operations on variables and data.

```
a + b;
```

Here, the `+` operator is used to add two variables `a` and `b`. Similarly, there are various other arithmetic operators in Java.

Operator	Operation
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulo Operation (Remainder after division)

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

    // declare variables
    int a = 12, b = 5;

    // addition operator
    System.out.println("a + b = " + (a + b));

    // subtraction operator
    System.out.println("a - b = " + (a - b));

    // multiplication operator
    System.out.println("a * b = " + (a * b));

    // division operator
    System.out.println("a / b = " + (a / b));

    // modulo operator
    System.out.println("a % b = " + (a % b));
  }
}
```

Output

a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2

Java Relational Operators

Relational operators are used to check the relationship between two operands.

For example,

// check if **a** is less than **b**

a < b;

Here, < operator is the relational operator. It checks if a is less than b or not.

It returns either true or false.

Operator	Description	Example
`==`	Is Equal To	`3 == 5` returns **false**
`!=`	Not Equal To	`3 != 5` returns **true**
`>`	Greater Than	`3 > 5` returns **false**
`<`	Less Than	`3 < 5` returns **true**
`>=`	Greater Than or Equal To	`3 >= 5` returns **false**
`<=`	Less Than or Equal To	`3 <= 5` returns **true**

```
public static void main(String[] args) {
  int a = 7, b = 11;

  // value of a and b
  System.out.println("a is " + a + " and b is " + b);

  // == operator
  System.out.println(a == b);  // false

  // != operator
  System.out.println(a != b);  // true

  // > operator
  System.out.println(a > b);  // false

  // < operator
  System.out.println(a < b);  // true

  // >= operator
  System.out.println(a >= b);  // false

  // <= operator
  System.out.println(a <= b);  // true
}
```

Java Logical Operators

Logical operators are used to check whether an expression is true or false. They are used in decision making.

Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa

Example Logical Operators

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

    // && operator
    System.out.println((5 > 3) && (8 > 5));  // true
    System.out.println((5 > 3) && (8 < 5));  // false

    // || operator
    System.out.println((5 < 3) || (8 > 5));  // true
    System.out.println((5 > 3) || (8 < 5));  // true
    System.out.println((5 < 3) || (8 < 5));  // false

    // ! operator
    System.out.println(!(5 == 3));  // true
    System.out.println(!(5 > 3));  // false
  }
}
```

Java Unary Operators

Unary operators are used with only one operand. For example, ++ is a unary operator that increases the value of a variable by 1. That is, ++5 will return 6.

Operator	Meaning
`+`	**Unary plus:** not necessary to use since numbers are positive without using it
`-`	**Unary minus:** inverts the sign of an expression
`++`	**Increment operator:** increments value by 1
`--`	**Decrement operator:** decrements value by 1

Example: Increment and Decrement Operators

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

    // declare variables
    int a = 12, b = 12;
    int result1, result2;

    // original value
    System.out.println("Value of a: " + a);

    // increment operator
    result1 = ++a;
    System.out.println("After increment: " + result1);

    System.out.println("Value of b: " + b);

    // decrement operator
    result2 = --b;
    System.out.println("After decrement: " + result2);
  }
}
```

Output

```
Value of a: 12
After increment: 13
Value of b: 12
After decrement: 11
```

Java Ternary Operator

The ternary operator (conditional operator) is shorthand for the if-then-else statement.

For example,

variable = Expression ? expression1 : expression2 ;
Here's how it works.

If the Expression is true, expression1 is assigned to the variable.
If the Expression is false, expression2 is assigned to the variable.

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

    int februaryDays = 29;
    String result;

    // ternary operator
    result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
    System.out.println(result);
  }
}
```

Output-

```
Leap year
```

Java Assignment Operators

Assignment operators are used in Java to assign values to variables.

For example,
int age;
age = 5;

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

Example 2: Assignment Operators

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

    // create variables
    int a = 4;
    int var;

    // assign value using =
    var = a;
    System.out.println("var using =: " + var);

    // assign value using =+
    var += a;
    System.out.println("var using +=: " + var);

    // assign value using =* and the latest value of var is 8
    var *= a;
    System.out.println("var using *=: " + var);
  } }
```

Output

```
var using =: 4
var using +=: 8
var using *=: 32
```

Constructor

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. In Java, a constructor is a block of codes similar to the method.

```
Class Main {

public static void main ( String [ ]
args ){
 Test obj = new Test(Ram,100);



}

}
```

```
Class Test  {
Static Test (String s, int p){
System.out println(“Name is”+s);
System.out.println(“Marks = “+ p);
}
}
```

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

Terms used in Inheritance

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

The syntax of Java Inheritance

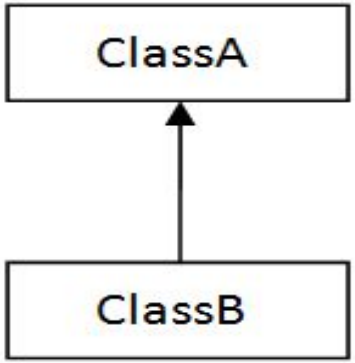
```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

The **extends** keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality

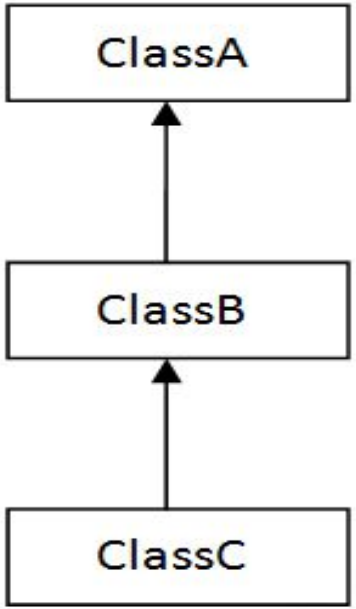
```
class Employee{
 float salary=40000;
}

class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer
is:"+p.bonus);  } }
```

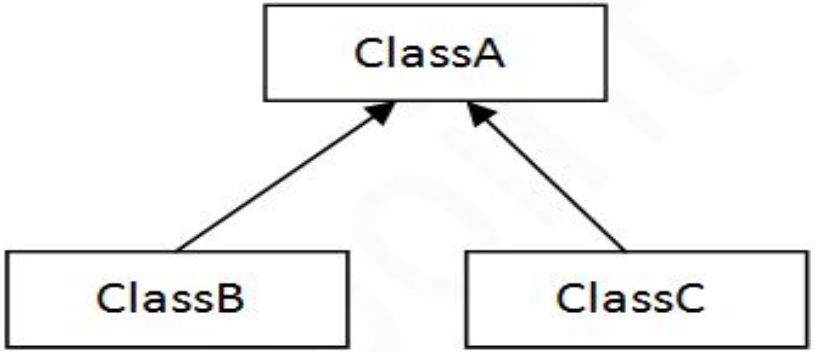
Types of inheritance in java



1) Single



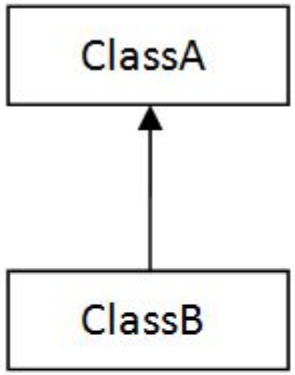
2) Multilevel



3) Hierarchical

Single Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();  } }
```



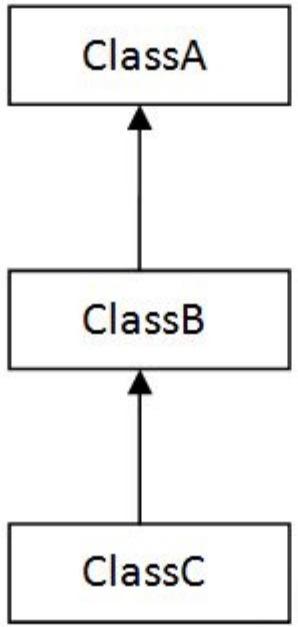
1) Single

Output:

barking...
eating…

Multilevel Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();  } }
```

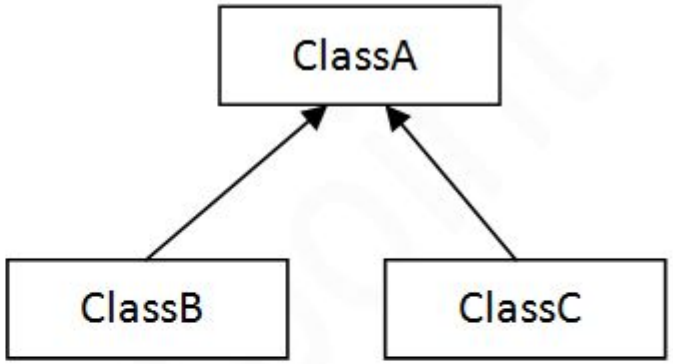


2) Multilevel

Output :
weeping...
barking...
eating…

Hierarchical Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```



3) Hierarchical

Output:

meowing...
eating…

Java Math methods

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

Package - java.lang.Math;

Math.sqrt(x)

The Math.sqrt(x) method returns the square root of x

Example

```
Math.sqrt(64); // output 8
```

Math.abs(x)

The Math.abs(x) method returns the absolute (positive) value of x

Example

```
Math.abs(-4.7) // output 4.7
```

Random Numbers

Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive)

Example

```
Math.random();
```

To get more control over the random number, e.g. you only want a random number between 0 and 100, you can use the following formula:

Example

```
int randomNum = (int)(Math.random() * 101);  // 0 to 100
```

Math.cbrt(x)

The Math.cbrt(x) method returns the square root of x

Example

```
Math.cbrt(125); // output 5
```

Math.pow() method The pow() method returns the result of the first argument raised to the power of the second argument.

That is, `pow(a, b) = a^b`

Example

```
Math.pow(5, 3); // Output 125
```

Math.min(x,y)

The Math.min(x,y) method can be used to find the lowest value of x and y:

Example

```
Math.min(5, 10); //output 5
```

Math.max(x,y)

The Math.max(x,y) method can be used to find the highest value from x and y

Example

```
Math.max(5, 10); // output 10
```

Math.round()

The round() method rounds the specified value to the closest int or long value and returns it. That is, 3.87 is rounded to 4 and 3.24 is rounded to 3.

Example

```
double d = Math.round(3.87)//output 4
double e = Math.round(3.24)//output 3
```

Math.ceil()

The ceil() method rounds the specified double value upward and returns it. The rounded value will be equal to the mathematical integer. That is, the value 3.24 will be rounded to 4.0 which is equal to integer 4.

Example

```
double d= Math.ceil(3.24) //output 4
double d= Math.ceil(3.78) //output 4
```

Math.floor()

The floor() method rounds the specified double value downward and returns it. The rounded value will be equal to a mathematical integer. That is, the value 3.8 will be rounded to 3.0 which is equal to integer 3.

Example

```
double d= Math.floor(3.8) // output 3
double d= Math.floor(3.2) // output 3
```

Java String Methods

The String class has a set of built-in methods that you can use on strings.

concat()

Appends a string to the end of another string

Example

String S1 = “hello”;
String S2 = “world”
S1.concat(S2) // output helloworld

Another way to add - S1+S2

equals()

Compares two strings. Returns true if the strings are equal, and false if not

Example

```
String myStr1 = "Hello";
String myStr2 = "Hello";
String myStr3 = "Another String";
System.out.println(myStr1.equals(myStr2)); // Returns true because they are equal
System.out.println(myStr1.equals(myStr3)); // false
```

startsWith()

Checks whether a string starts with specified characters

Example

```
String str = "Hello";
str.startsWith("He"); //This will return true because string str starts with "He”
```

endsWith()

Checks whether a string ends with the specified character(s)

Example-

```
String myStr = "Hello";
System.out.println(myStr.endsWith("Hel"));   // false
System.out.println(myStr.endsWith("llo"));   // true
System.out.println(myStr.endsWith("o"));     // true
```

isEmpty()

Java.lang.String.isEmpty() String method checks whether a String is empty or not. This method returns true if the given string is empty, else it returns false.

Example

```
String str1 = "Hello_Gfg";   // non-empty string
String str2 = "";    // empty string
System.out.println(str1.isEmpty());    // prints false
System.out.println(str2.isEmpty());  // prints true
```

length()

Returns the length of a specified string

Example-

```
String s1="javatpoint";
String s2="python";
System.out.println("string length is:
"+s1.length());//10 is the length of javatpoint string
System.out.println("string length is:
"+s2.length());//6 is the length of python string
```

split()

Splits a string into an array of substrings

Example

```
String text = "Java is a fun programming language";
String[ ] result = text.split(" "); // Split string from space
System.out.println(result[0]) ;     // Java
System.out.println(result [1]) ;    //is
```

substring()

Returns a new string which is the substring of a specified string. We pass beginIndex and endIndex number position in the Java substring method where beginIndex is inclusive, and endIndex is exclusive.

Example

```
String str1 = "java is fun"; // extract substring from index 0 to 3.
System.out.println(str1.substring(0, 4)); // java
```

0	1	2	3	4	5	6	7	8	9	10
j	a	v	a		i	s		f	u	n

toLowerCase()

Converts a string to lower case letters.

Example

```
String txt = "Hello World";
System.out.println(txt.toLowerCase()); // hello world
```

toUpperCase()

Converts a string to upper case letters

Example

```
String txt = "Hello World";
System.out.println(txt.toUpperCase()); //HELLO WORLD
```

trim()

Removes whitespace from both ends of a string.

Example

```
String myStr = "       Hello World!       ";
System.out.println(myStr);
System.out.println(myStr.trim());
```

Output

```
       Hello World!
Hello World!
```

Java Array (Single Dimensional Array)

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Example

```
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
System.out.println(a[1]); //output 20
System.out.println(a[2]); //output 70
System.out.println(a[0]); //output 10
```

Another way

```
int a[ ]={33,3,4,5};
System.out.println(a[0]); //output 33
System.out.println(a[1]); //output 3
System.out.println(a[2]); //output 4
System.out.println(a[3]); //output 5
```

Multidimensional Array in Java.

Multidimensional Arrays can be defined in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form (in row major order).

Example to initialize Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

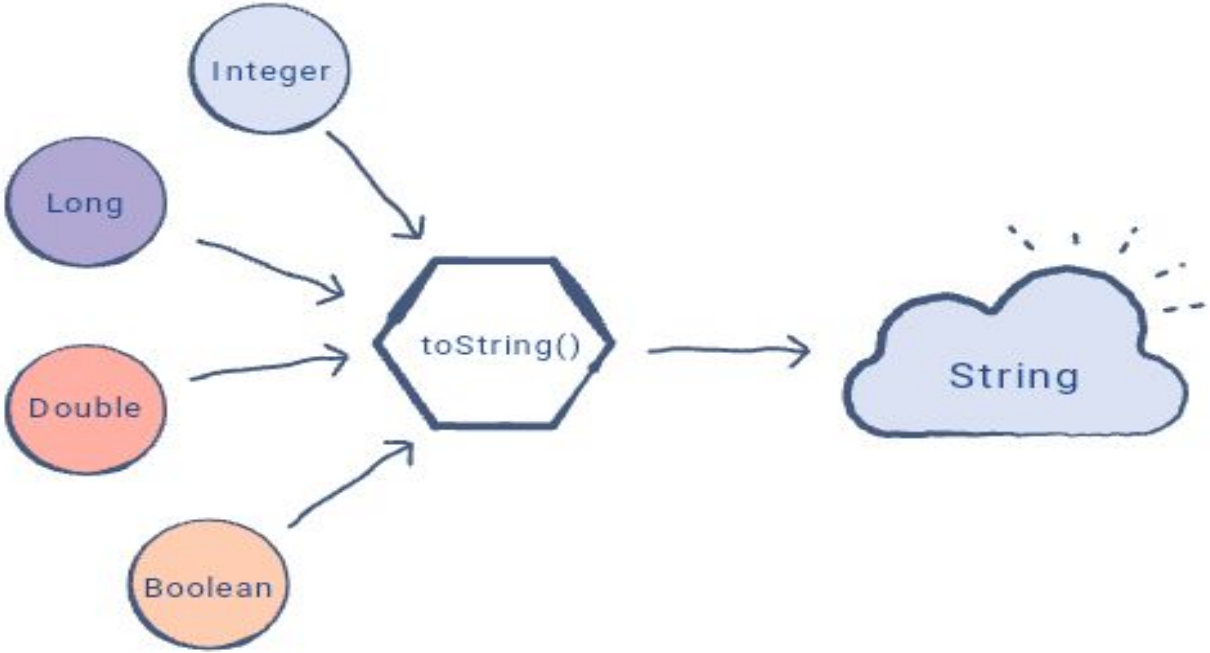
Example

```
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

[Type Casting , toString () ,parseInt()]

toString()

A toString() is an in-built method in Java that returns the value given to it in string format. Hence, any object that this method is applied on, will then be returned as a string object


Integer
Long
Double
Boolean
toString()
String

Example

```
int a = 123;
String s= Integer.toString(a);
System.out.println(s);

System.out.println(Boolean.toString(true));
System.out.println(Integer.toString(457));
System.out.println(Double.toString(26.578));
System.out.println(Long.toString(25625567));
```

parseInt()

The method generally used to convert String to Integer in Java is parseInt().

Example

```
String s = “466”;
int i = Integer.parseInt(s);// String will convert into integer
```

Similarly

```
String s=”22.3”;
double d= Double.parseDouble(s);// String to Double
float f= Float.parseFloat(s);// String to Float
short s= Short.parseShort(s);// String to Short
byte b= Byte.parseByte(s);// String to Byte
long l= Long.parseLong(s);// String to Long
```

Type conversion/Type casting in Java

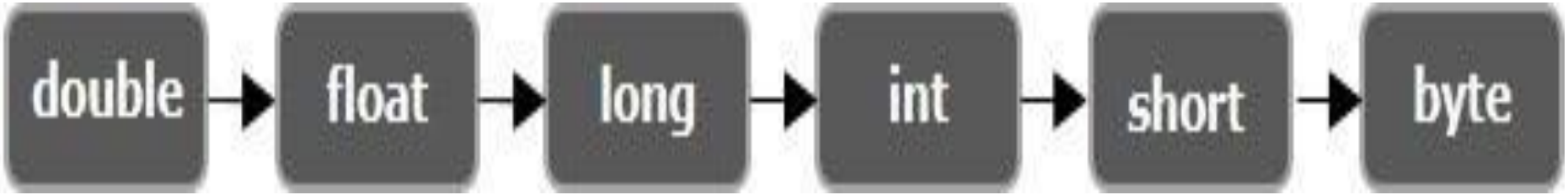
Type casting is when you assign a value of one primitive data type to another type.

Widening − Converting a lower datatype to a higher datatype is known as widening. In this case the casting/conversion is done automatically.

Example

```
byte b= 23;
short c= b;     // byte to short
int d= c;       // short to int
long e= d;      // int to long
float f = e;    // long to float
double g = f;  // float to double
```

Narrowing − Converting a higher datatype to a lower datatype is known as narrowing. In this case the casting/conversion is not done automatically.



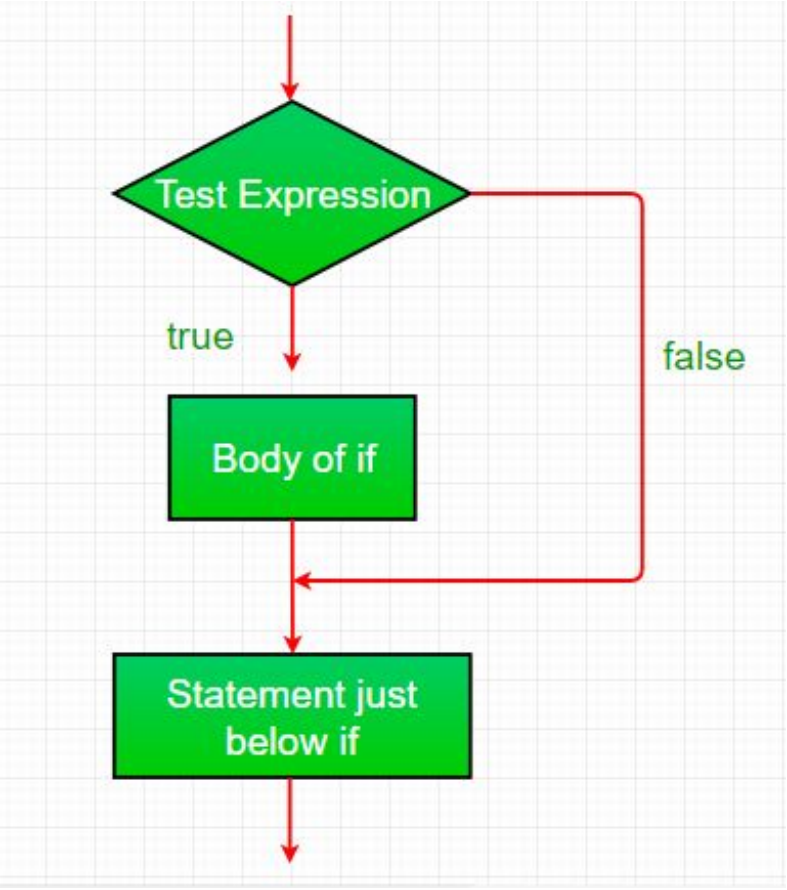
Example

```
double d = 23.44;
float e = (float) d ;   // double to float
long f = (long) e ;    // float to long
int g = (int) f;   //long to int
short h= (short) g;  // int to short
byte i = (byte) h ;   // short to byte
```

Decision-making statements / Conditional statements

1. Java if statement

The Java if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.



If statement

3.b) If false

3.a) If true

1. 2.

```
if ( condition )
{
    // body of the if
    // statements to be executed
}
```

4.

5.

```
// statements outside the if
```

Example

```
int x = 20;
int y = 18;
if (x > y) {
  System.out.println("x is greater than y");
}
```

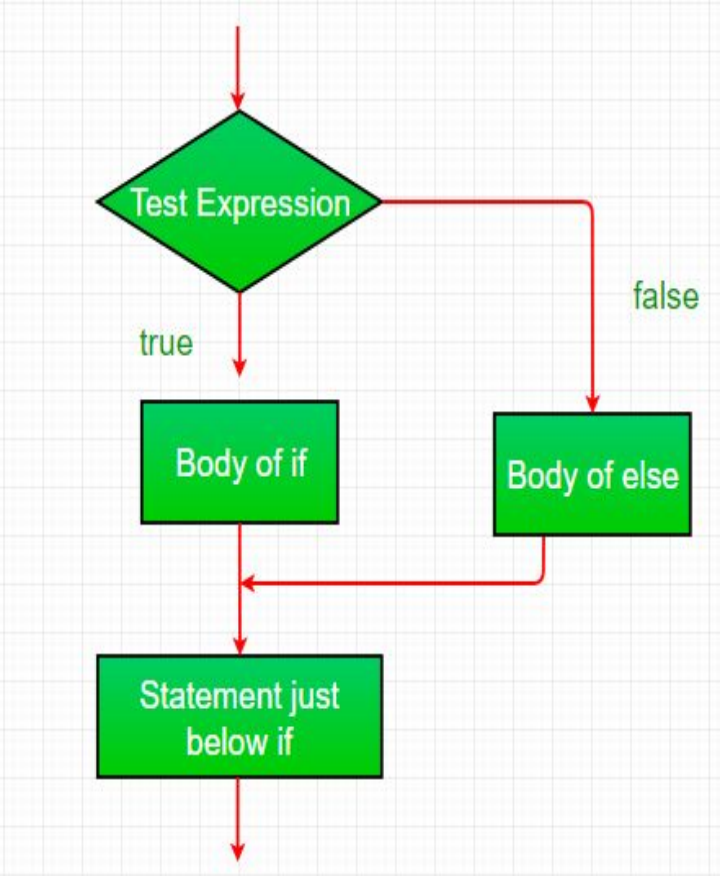
2. Java if-else Statement

The Java if-else statement also tests the condition. 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
}
```

If - else statement



3.b) If false

3.a) If true

1. 2.

```
if ( condition )
{
    // body of the if
    // statements to be executed
}
else
{
    // body of the else
    // statements to be executed
}
// statements outside the if-else
```

4. 5. 6.

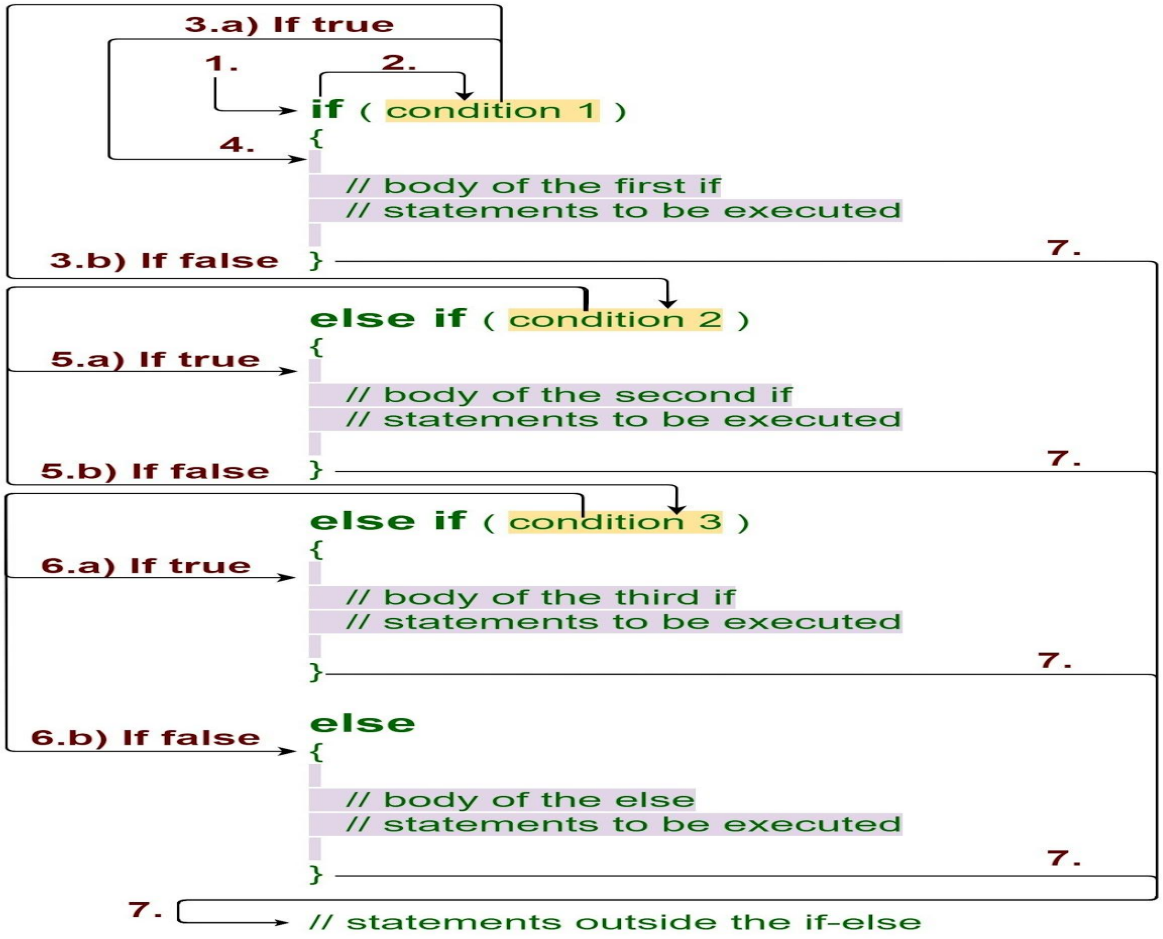
Example

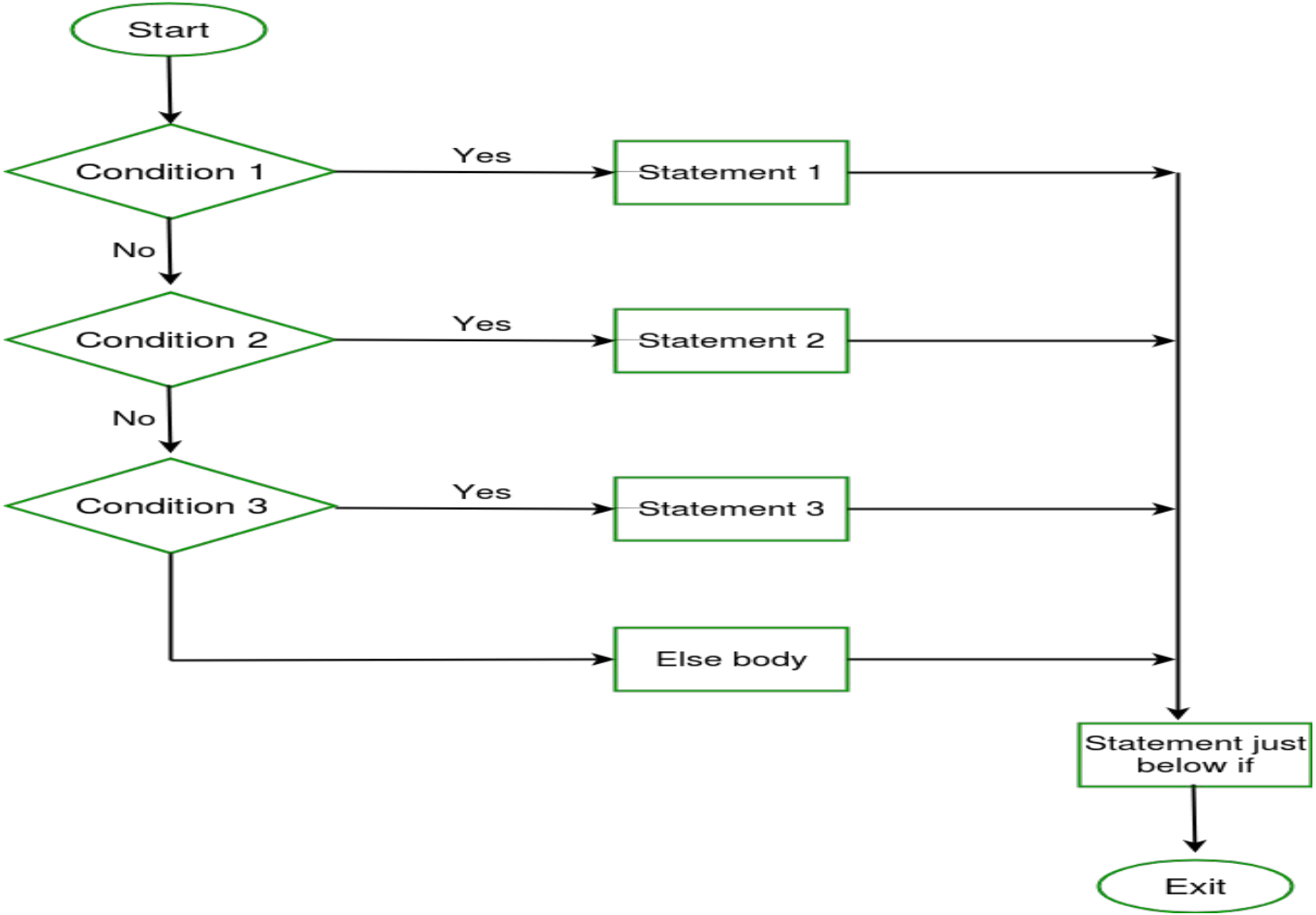
```
public static void main(String[] args) {
    //defining a variable
    int number=13;
    //Check if the number is divisible by 2 or not
    if(number%2==0){
        System.out.println("even number");
    }
else{
        System.out.println("odd number");
    }
}
```

Java if-else-if ladder

The if statement and if else statements are used to check the single condition but if you have multiple conditions dependent on each other means only one condition can be true at a time then you should use else if ladder in Java.

If - else if ladder



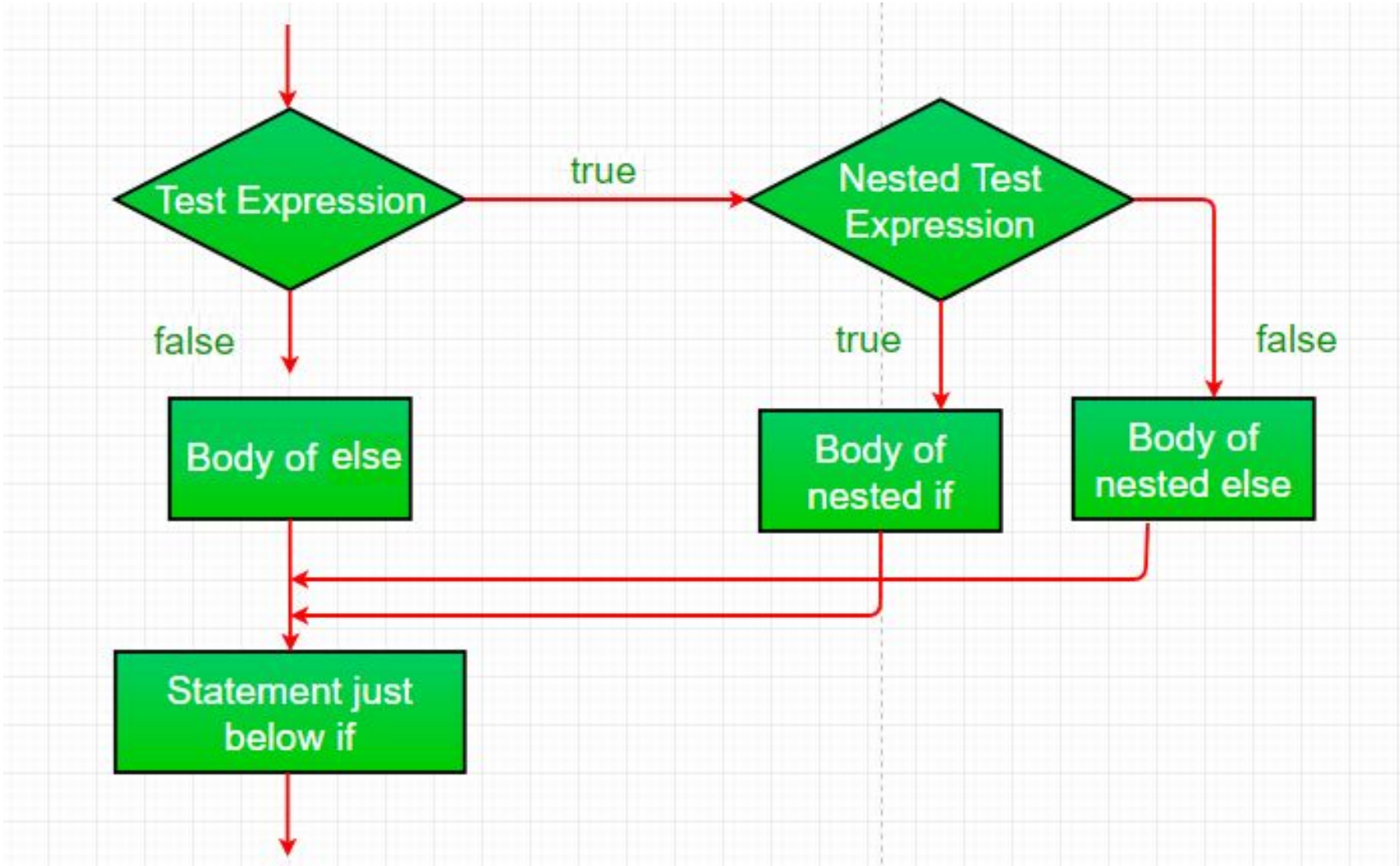

Start
Condition 1
Yes
Statement 1
No
Condition 2
Yes
Statement 2
No
Condition 3
Yes
Statement 3
Else body
Statement just below if
Exit

Example

```
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter a percentage");
float per = sc.nextFloat();
if(per>=70)
{
System.out.println("Distinction");
}else if(per>=60)
{
System.out.println("First Class");
}else if(per>=50)
{
System.out.println("Second Class");
}else if(per>=40)
{
System.out.println("Pass");  }
else
{
System.out.println("Failed"); } }
```

Nested If Else in Java

The nested if else statement means using if else statement inside if body or else body.


Test Expression
true
Nested Test Expression
false
true
false
Body of else
Body of nested if
Body of nested else
Statement just below if

Syntax :

```
if( condition ){
        if( condition ){
                if( condition ){
                        ......
                }
        }
}
```

Example

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

    {
        int a=10;
        int b=20;
        if(a==10){
                    if(b!=20){
                        System.out.println("TechBuzz”);

                            }

                    else{

                            System.out.println("coding is fun");
  }
}
```

Java Switch Statements

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

The break Keyword

When Java reaches a break keyword, it breaks out of the switch block.

Example

```
int day = 7;
switch (day) {
  case 6:
    System.out.println("Today is Saturday");
    break;
  case 7:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
// Outputs -Today is Sunday
```

Example The default Keyword

The default keyword specifies some code to run if there is no case match

```
int day = 4;
switch (day) {
  case 6:
    System.out.println("Today is Saturday");
    break;
  case 7:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
// Outputs "Looking forward to the Weekend”
```

for Loops in Java

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.

Syntax

```
for (initialExpression; testExpression; updateExpression)
 {
    // body of the loop
}
```

Example

```
public static void main(String[] args) {
     int n = 5;
  // for loop
    for (int i = 1; i <= n; ++i) {
      System.out.println("Java is fun");
    }
  }
```

Output

```
Java is fun
Java is fun
Java is fun
Java is fun
Java is fun
```

Java for-each Loop

The Java for loop has an alternative syntax that makes it easy to iterate through arrays and collections.

for-each Loop Sytnax

```
for(dataType item : array) {
    ...
}
```

example,

```
public static void main(String[] args) {
    // create an array
    int[] numbers = {3, 7, 5, -5};

    // iterating through the array
    for (int number: numbers) {
        System.out.println(number);
    }
  }
```

Output

```
3
9
5
-5
```

Example 2: Sum of Array Elements

```
// Calculate the sum of all elements of an array
 public static void main(String[] args) {

   // an array of numbers
   int[] numbers = {3, 4, 5, -5, 0, 12};
   int sum = 0;

   // iterating through each element of the array
   for (int number: numbers) {
     sum += number;
   }
   System.out.println("Sum = " + sum);
 }
```

Output:

```
Sum = 19
```

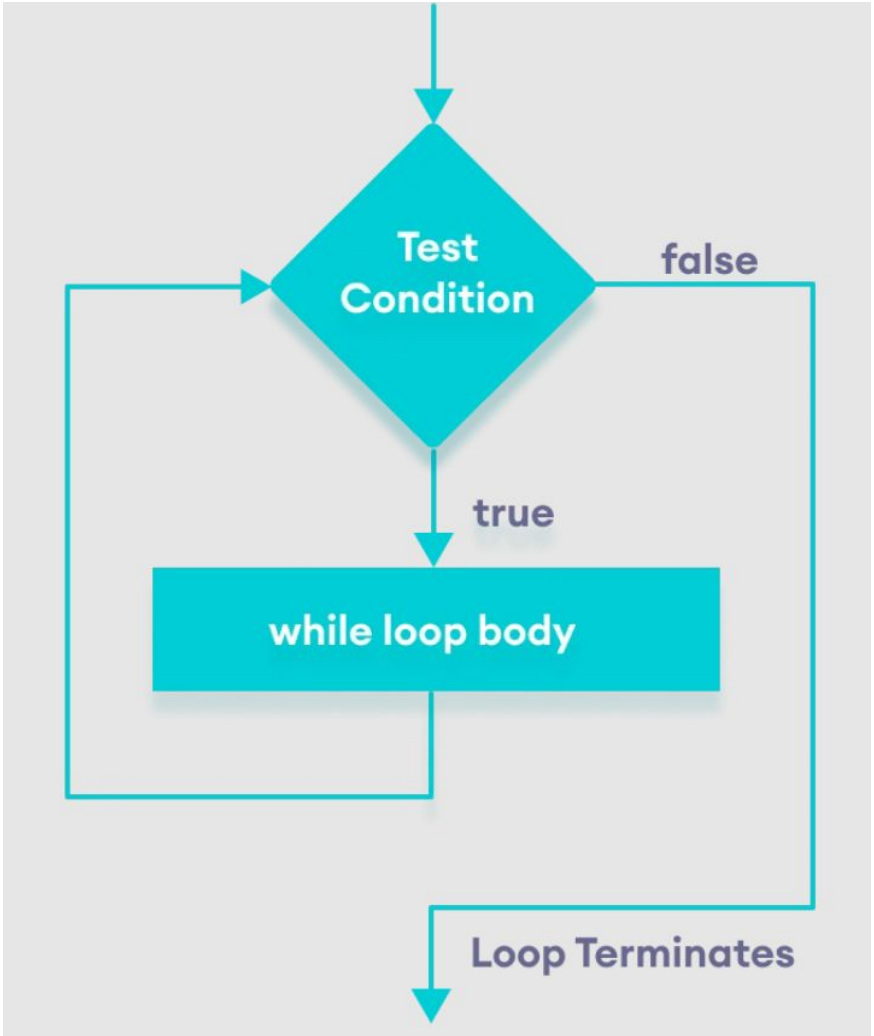
While loop and do while loop

Java while loop

Java while loop is used to run a specific code until a certain condition is met.

Syntax

```
while (testExpression) {
    // body of loop
}
```



Example

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

  // declare variables
  int i = 1, n = 5;

  // while loop from 1 to 5
  while(i <= n) {
    System.out.println(i);
    i++;
  }
}
```

Output

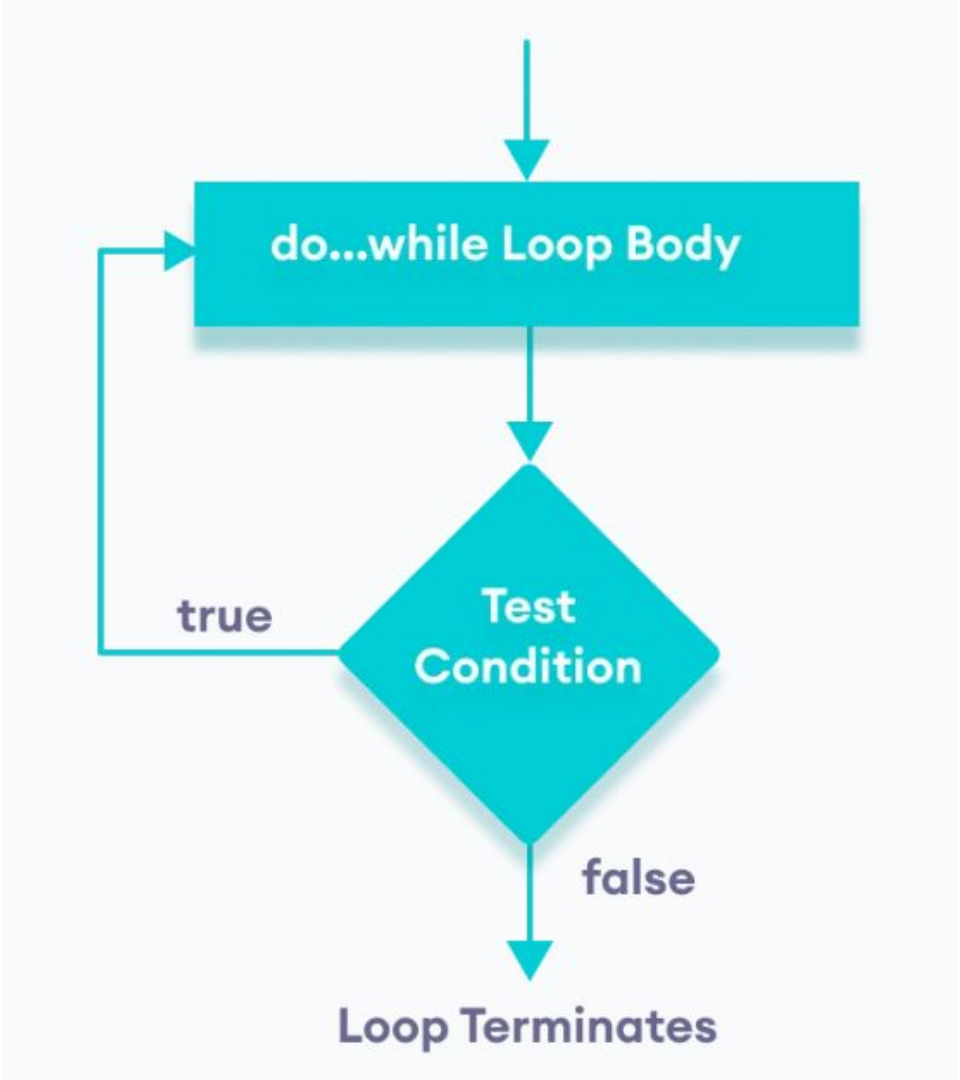
```
1
2
3
4
5
```

Java do...while loop

The do...while loop is similar to while loop. However, the body of do...while loop is executed once before the test expression is checked.

Syntax

```
do {
    // body of loop
} while(textExpression);
```


do...while Loop Body
Test Condition
true
false
Loop Terminates

Example

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

    int i = 1, n = 5;

    // do...while loop from 1 to 5
    do {
      System.out.println(i);
      i++;
    } while(i <= n);
```

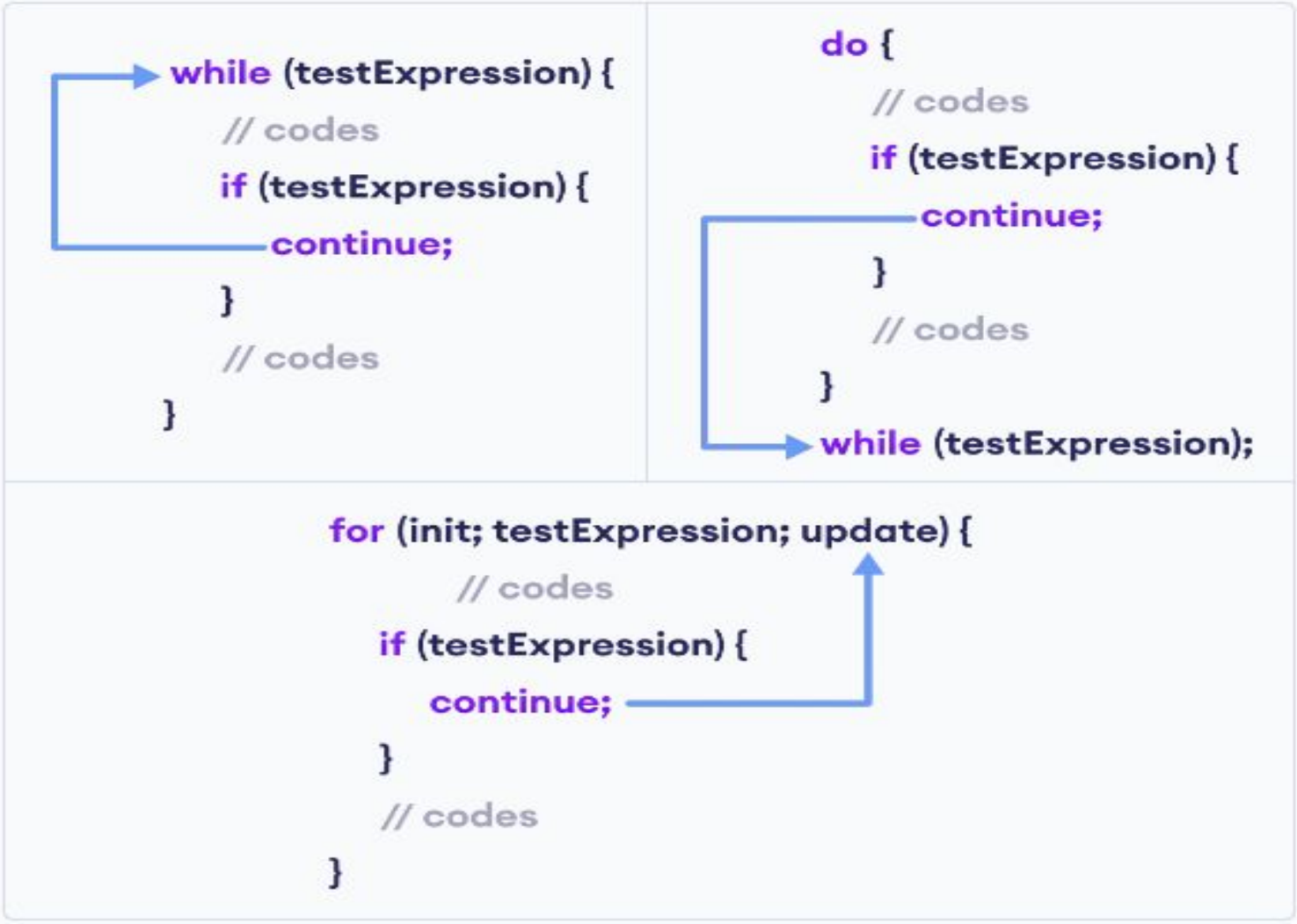
Output

```
1
2
3
4
5
```

Java continue

The continue statement skips the current iteration of a loop (for, while, do...while, etc).

After the continue statement, the program moves to the end of the loop. And, test expression is evaluated


while (testExpression) {
// codes
if (testExpression) {
continue;
}
// codes
}
do {
// codes
if (testExpression) {
continue;
}
// codes
}
while (testExpression);
for (init; testExpression; update) {
// codes
if (testExpression) {
continue;
}
// codes
}

Example

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

    // for loop
    for (int i = 1; i <= 10; ++i) {

      // if value of i is between 4 and 9
      // continue is executed
      if (i > 4 && i < 9) {
        continue;
      }
      System.out.println(i);
    }
```

Output:

```
1
2
3
4
9
10
```

Java break Statement

The break statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

Example

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

    // for loop
    for (int i = 1; i <= 10; ++i) {

        // if the value of i is 5 the loop terminates
        if (i == 5) {
            break;
        }
        System.out.println(i);
    }
}
```

Output:

```
1
2
3
4
```