2.CONTROL STATEMENTS

Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
```

example

```
import java.util.*;

class IfDemo {
        public static void main(String args[])
        {
                int i = 10;

                if (i < 15)
                        System.out.println("Inside If block");
        System.out.println("10 is less than 15");
                System.out.println("I am Not in if");
        }
}
```

Output

```
Inside If block
10 is less than 15
I am Not in if
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
4. else{
5. statement 2; //executes when condition is false
6. }
```

```
import java.util.*;
```

```
class IfElseDemo {
        public static void main(String args[])
        {
        int i = 10
                if (i < 15)
                        System.out.println("i is smaller than 15");
                else
                        System.out.println("i is greater than 15");
        }
}
```

Output

```
i is smaller than 15
```

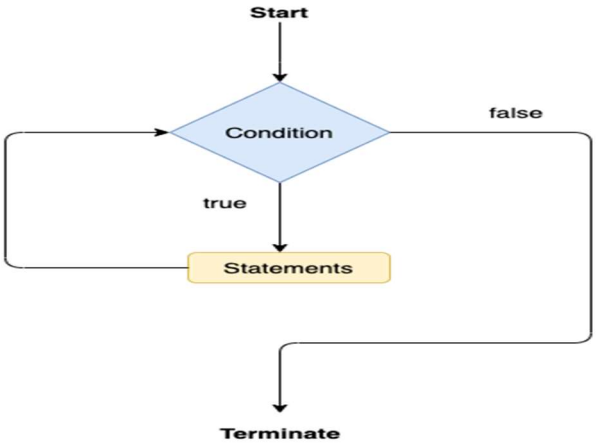
Java while loop:-

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
1. while(condition){
2. //looping statements
3. }
```

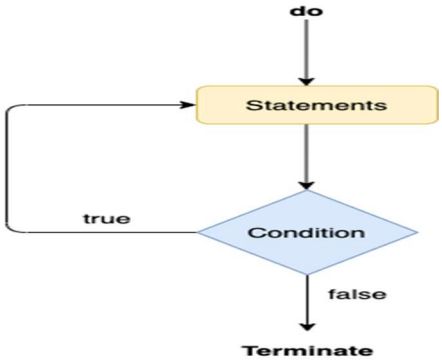


Java do-while loop:-

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
1. do
2. {
3. //statements
4. } while (condition);
```



Switch Statement:-

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:-

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

Syntax switch stmt:-

```
1. switch (expression){
2.     case value1:
3.      statement1;
4.      break;
5.     .
6.     .
7.     case valueN:
8.      statementN;
9.      break;
10.    default:
11.     default statement;
12. }
```

Java break statement:-

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Java continue statement:-

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Methods in java:-

The **method in Java** or Methods of Java is a collection of statements that perform some specific task and return the result to the caller. A Java method can perform some specific task without returning anything. Java Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class that is different from languages like C, C++, and Python.

1. A method is like a function i.e. used to expose the behavior of an object.
2. It is a set of codes that perform a particular task.

Syntax of Method:-

```
<access_modifier> <return_type> <method_name>( list_of_parameters)
{
    //body
}
```

Advantage of Method

- Code Reusability
- Code Optimization

Static Method:

Access the static data using class name. Declared inside class with **static** keyword.

The static keyword is used to construct methods that will exist regardless of whether or not any instances of the class are generated. Any method that uses the static keyword is referred to as a static method.

Features of static method:

- A static method in Java is a method that is part of a class rather than an instance of that class.
- Every instance of a class has access to the method.
- Static methods have access to class variables (static variables) without using the class's object (instance).
- Only static data may be accessed by a static method. It is unable to access data that is not static (instance variables).
- In both static and non-static methods, static methods can be accessed directly.

Syntax to declare the static method:

```
Access_modifier static void methodName()
{
      // Method body.
}
```

Why use Static Methods?

1. To access and change static variables and other non-object-based static methods.
2. Utility and assist classes frequently employ static methods.

Restrictions in Static Methods:

1. Non-static data members or non-static methods cannot be used by static methods, and static methods cannot call non-static methods directly.
2. In a static environment, this and super aren't allowed to be used.

Why is the main method in Java static?

It's because calling a static method isn't needed of the object. If it were a non-static function, JVM would first build an object before calling the main() method, resulting in an extra memory allocation difficulty.

Example of static method:-

```
import java.io.*;
public class GFG {
        // static variable
        static int a = 40;
        // instance variable
        int b = 50;
        void simpleDisplay()
        {
                System.out.println(a);
                System.out.println(b);
        }
// Declaration of a static method.
        static void staticDisplay()
        {
        System.out.println(a);
        }
        // main method
        public static void main(String[] args)
        {
                GFG obj = new GFG();
                obj.simpleDisplay();
                // Calling static method.
                staticDisplay();
        }
}
```

Output:-

```
40
50
40
```

Program module in java:;-

A Java module is a set of packages that declares which of them form an API accessible to other modules and which are internal and encapsulated — similar to how a class defines the visibility of its members. A module also declares what other modules it requires for its operation. A library author can choose to place it in a module, an application author can choose to place its components in modules, and, of course, the JDK itself is made of modules.

A Module is a group of closely related packages and resources along with a new module descriptor file.

Module Types

There are four types of modules in the new module system:

- **System Modules** – These are the modules listed when we run the *list-modules* command above. They include the Java SE and JDK modules.
- **Application Modules** – These modules are what we usually want to build when we decide to use Modules. They are named and defined in the compiled *module-info.class* file included in the assembled JAR.
- **Automatic Modules** – We can include unofficial modules by adding existing JAR files to the module path. The name of the module will be derived from the name of the JAR. Automatic modules will have full read access to every other module loaded by the path.
- **Unnamed Module** – When a class or JAR is loaded onto the classpath, but not the module path, it's automatically added to the unnamed module. It's a catch-all module to maintain backward compatibility with previously-written Java code.

Syntax

```
module myModuleName

{

// all directives are optional

 }
```

Example :-

```
package com.java4coding.Example;
public class Example {
        public static void sayHello() {
                System.out.println("Hello");
        }
}
```

Java Static Field:-

In the world of Java programming, static fields play a significant role in defining class-level variables that are shared across all instances of a class. These static fields are initialized only once, when the class is loaded into memory. Understanding how Java handles static field initialization is crucial for writing efficient and error-free code. In this article, we will explore the basics of Java static field initialization and delve into the various approaches available.

Static fields are declared using the static keyword and are associated with the class itself rather than with any specific instance of the class. They are commonly used to store data that is shared among all objects of a class, such as configuration settings, counters, or constant values. Since static fields are not tied to any specific instance, they can be accessed using the class name itself, without the need for object instantiation.

Syntax:-

```
public class MyClass {
    public static int myStaticField;
    static {
        // Complex initialization logic
        myStaticField = calculateInitialValue();
    }
}
```

Example:-

```
public class StaticFieldInitializationExample {
    public static int myStaticField = 10;
    static {
        System.out.println("Static initialization block");
        myStaticField = 20;
    }
    public static void main(String[] args) {
        System.out.println("Static field value: " + myStaticField);
    }
}
```

Output:-
```
Static initialization block
Static field value: 20
```

Java Math class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

Unlike some of the StrictMath class numeric methods, all implementations of the equivalent function of Math class can't define to return the bit-for-bit same results. This relaxation permits implementation with better-performance where strict reproducibility is not required.

If the size is int or long and the results overflow the range of value, the methods addExact(), subtractExact(), multiplyExact()and toIntExact() throw an ArithmeticException.

For other arithmetic operations like increment, decrement, divide, absolute value, and negation overflow occur only with a specific minimum or maximum value. It should be checked against the maximum and minimum value as appropriate.

Method	Description
Math.abs()	It will return the Absolute value of the given value.
Math.max()	It returns the Largest of two values.
Math.min()	It is used to return the Smallest of two values.
Math.round()	It is used to round of the decimal numbers to the nearest value.
Math.sqrt()	It is used to return the square root of a number.
Math.cbrt()	It is used to return the cube root of a number.
Math.pow()	It returns the value of first argument raised to the power to second argument.
Math.signum()	It is used to find the sign of a given value.

Math.ceil()	It is used to find the smallest integer value that is greater than or equal to the argument or mathematical integer.
Math.copySign()	It is used to find the Absolute value of first argument along with sign specified in second argument.
Math.nextAfter()	It is used to return the floating-point number adjacent to the first argument in the direction of the second argument.
Math.nextUp()	It returns the floating-point value adjacent to d in the direction of positive infinity.
Math.nextDown()	It returns the floating-point value adjacent to d in the direction of negative infinity.
Math.floor()	It is used to find the largest integer value which is less than or equal to the argument and is equal to the mathematical integer of a double value.
Math.floorDiv()	It is used to find the largest integer value that is less than or equal to the algebraic quotient.
Math.random()	It returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
Math.rint()	It returns the double value that is closest to the given argument and equal to mathematical integer.
Math.hypot()	It returns sqrt(x^2 +y^2) without intermediate overflow or underflow.
Math.ulp()	It returns the size of an ulp of the argument.
Math.getExponent()	It is used to return the unbiased exponent used in the representation of a value.
Math.IEEEremainder()	It is used to calculate the remainder operation on two arguments as prescribed by the IEEE 754 standard and returns value.
Math.addExact()	It is used to return the sum of its arguments, throwing an exception if the result overflows an int or long.

Math.subtractExact()	It returns the difference of the arguments, throwing an exception if the result overflows an int.
Math.multiplyExact()	It is used to return the product of the arguments, throwing an exception if the result overflows an int or long.
Math.incrementExact()	It returns the argument incremented by one, throwing an exception if the result overflows an int.
Math.decrementExact()	It is used to return the argument decremented by one, throwing an exception if the result overflows an int or long.
Math.negateExact()	It is used to return the negation of the argument, throwing an exception if the result overflows an int or long.
Math.toIntExact()	It returns the value of the long argument, throwing an exception if the value overflows an int.

Example:-

```
1.  public class JavaMathExample1
2.  {
3.      public static void main(String[] args)
4.      {
5.          double x = 28;
6.          double y = 4;
7.           System.out.println("Maximum number of x and y is: " +Math.max(x, y));
8.          System.out.println("Square root of y is: " + Math.sqrt(y));
9.          System.out.println("Power of x and y is: " + Math.pow(x, y));
10.         System.out.println("Logarithm of x is: " + Math.log(x));
11.         System.out.println("Logarithm of y is: " + Math.log(y));
12.         System.out.println("log10 of x is: " + Math.log10(x));
13.         System.out.println("log10 of y is: " + Math.log10(y));
14.         System.out.println("log1p of x is: " +Math.log1p(x));
15.         System.out.println("exp of a is: " +Math.exp(x));
16.         System.out.println("expm1 of a is: " +Math.expm1(x));
17.     }
18. }
```

19. **Output:**

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of a is: 1.446257064291475E12
expm1 of a is: 1.446257064290475E12
```

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

example

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

```
22
3
```

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

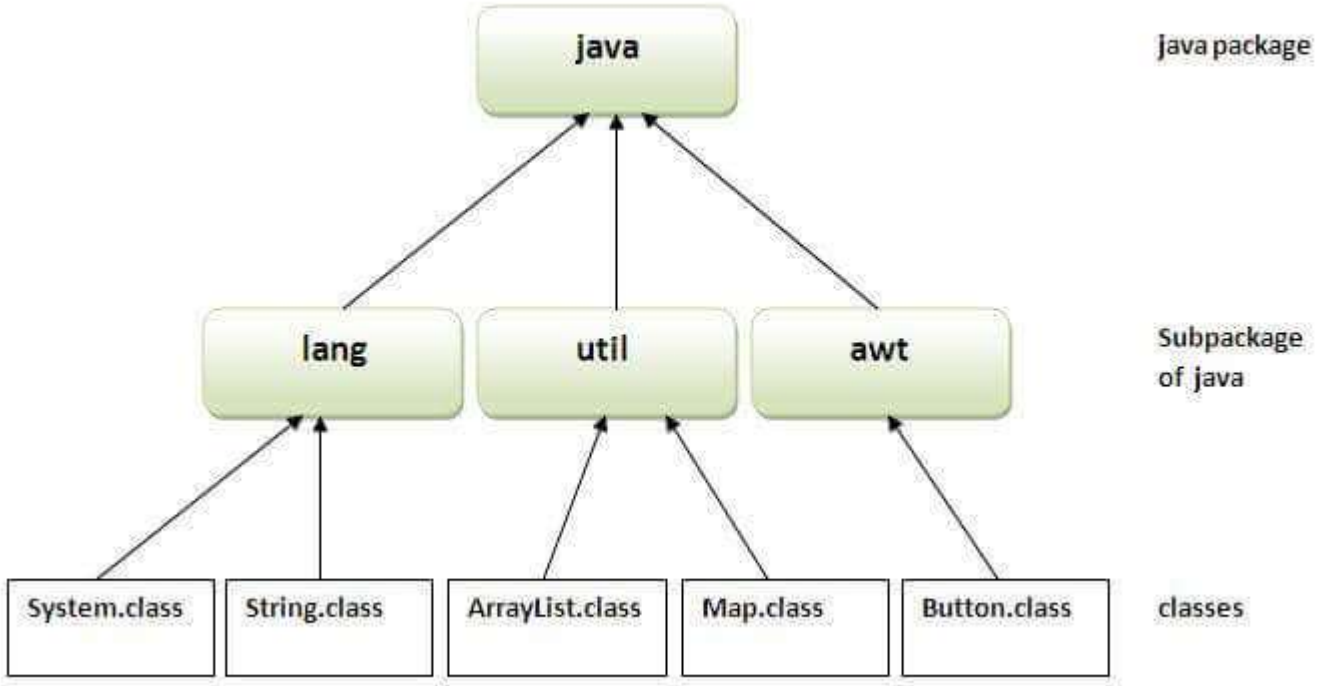
- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



Example of package that import the packagename.*

```
1. //save by A.java
2. package pack;
3. public class A{
4.   public void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. import pack.*;
4. class B{
5.   public static void main(String args[]){
6.    A obj = new A();
7.    obj.msg();
8.   }
9. }
```

```
Output:Hello
```