Introduction to Software Testing

What is Software ?

- Software is a set of programs, procedures, functions, associated data and/or its available documentation.

What are the Types of Software?

1. System Software - e.g Operating System
2. Programming Software - e.g: compiler
3. Application Software - e.g: Web Application, Mobile Apps, Desktop Applications, etc.

What is Software Testing?
In order to know what software testing is, we should know why software testing first.

Why is Software Testing required?For E.g : Client/company -> Software company ->Develop -> Test -> Deliver -> Client/company.

- Software Testing is a part of the software development process.
- Software Testing is an activity to detect and identify the defects in the software.
- The object of testing is to release quality products to the client.

What is java and why java ?

- Java was developed by James Gosling at Sun Microsystems Inc, in the year 1991.

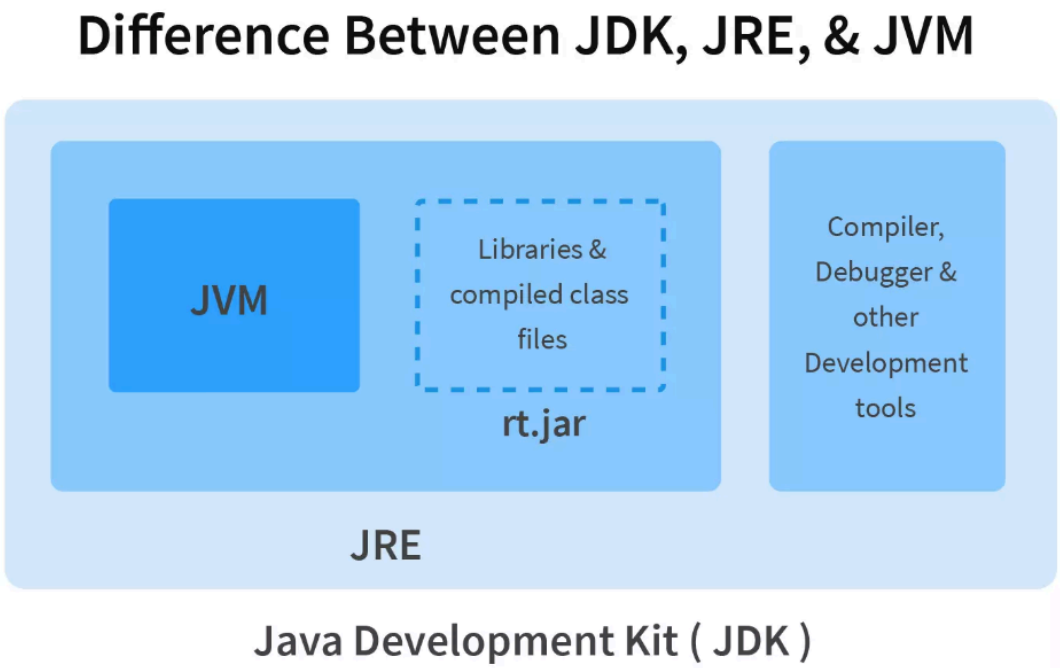
- Java is class-based, object-oriented programming language and is platform-independent and can run on a variety of platforms, such as Windows, MAC OS, Linux, etc. Java can be used to develop mobile apps, web apps, games and much more.

Here are some of the most important features of Java:

- Java is platform-independent and follows the principle of WORA (Write Once, Run Anywhere); it makes test scripts written in Java platform-independent, increasing test coverage.
- Many open-source automated testing frameworks based on Java, like JUnit and TestNG, have been created. They make automated testing easy by providing many inbuilt features and methods.
- Java's stack allocation system makes smooth integration with Selenium WebDriver, a commonly used tool for automation testing.
- Java supports multithreading and thus makes cross browser testing and parallel testing easier.

Which one do I need: JVM or a JRE or a JDK?

First let's understand each term and what does it stand for:



-

- **JVM**: JVM stands for Java Virtual Machine. As we know, Java codes are first compiled into bytecodes. The compilation is performed using the JAVAC compiler. It takes java code as input and converts it into bytecode. In the running phase, JVM then converts the bytecode into machine code. JVM is the one that actually calls the main method present in a java code. JVM loads the code, verifies and executes the code.
- So, the function of JVM is to convert bytecode into machine code. It is an essential component of both JDK and JRE. It is also platform-dependent and performs many functions, including memory management and security.

- **JDK**: JDK stands for Java Development Kit. It is the whole package containing compiler, Java Runtime Environment (JRE), java docs, debuggers etc. It contains development tools to provide an environment to develop java programs and JRE to execute java programs. It is thus essential to develop and run a java program on our computer.

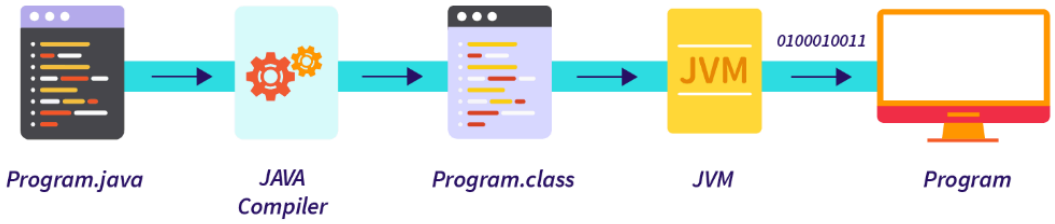
- **JRE**: JRE stands for Java Runtime Environment. It is included in the Java development kit. JRE allows us to run java programs on our computers using Java bytecodes. It has

class libraries and other resources which are needed to run a Java program. It combines the Java code created using JDK with the necessary libraries to run it on a JVM. Java Runtime Environment is what enables a Java program to run in any operating system without modification.

- So, we can deduce that JDK is the key to running and compiling our program. So, JDK entails JRE(consisting of JVM + Libraries)and development tools. JRE only helps to run the code but development tools like debugger, compiler, JavaDoc etc. aids in building applications.

How Does Java Programming Language Work?

It can be simplified in 3 simple steps :



- Here for the first step, we need to have a java source code otherwise we won't be able to run the program you need to save it with the fileName.java extension.

- Secondly, we need to use a compiler so that it compiles the source code which in turn gives out the java bytecode and that needs to have a fileName.class extension. The Java bytecode is a redesigned version of the Java source code, and this bytecode can be run anywhere irrespective of the machine on which it has been built.

- Later on, we put the java bytecode through the Java Virtual Machine which is an interpreter that reads all the statements thoroughly step by step from the java bytecode which will further convert it to the machine-level language so that the machine can execute the code. We get the output only after the conversion is through.

Java Syntax Basics

1. Java is a case sensitive language

Ex: NAME and name are not same as per Java Language
2. Java file name should be same as its Class name
3. Class name should start with upper case letter
4. Method name should start with lower case letter
5. Every statement should end with semicolon
6. Java program execution starts from main method which is mandatory in every program

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

}
```

Print in Java:
In Java, we use print to output required text directly to the console of IDE

Syntax:

Simple print statement:

```
System.out.print("Learning Java with Bishnu as GFG Mentor");
```

Simple print statement with new line:

```
System.out.println("Welcome to GFG SDET Session");
```

Difference in print & println

The basic difference between the print() and println() methods in Java are when using println(), the cursor in the output will be shown in the next line after printing the required output on the screen whereas when using print() method, the cursor will be shown in the same line after printing the required output on the screen.

Accepting user input

Sometimes, we may face a situation where we need to get the input from the user at runtime. We use the "Scanner" class to accept input from the user.

Syntax:

```
import java.util.Scanner;

Scanner userInput = new Scanner(System.in);
```

variable = userInput.next();
userInput.close();

Identifier:
A name in a java program is called identifier. It may be class name, method name, variable name and label name.

Rule 1: The only allowed characters in java identifiers are:
1) a to z
2) A to Z
3) 0 to 9
4) _
5) $

Rule 2: If we are using any other character we will get a compile time error.
Example:
1) total_number-------valid
2) Total#----------invalid

Rule 3: identifiers are not allowed to start with digits.
Example:
1) ABC123---------valid
2) 123ABC---------invalid

Rule 4: java identifiers are case sensitive up course java language itself treated as case sensitive language.
Example:

```
class Test{
        int number=10;
        int Number=20;
        int NUMBER=20; we can differentiate with case.
        int NuMbEr=30;
}
```

Rule 5: There is no length limit for java identifiers but it is not recommended to take more than
15 lengths.

Rule 6: We can't use reserved words as identifiers.
Example: int if=10; ----------invalid

Rule 7: All predefined java class names and interface names we use as identifiers.
Example 1:

```
class Test
{
public static void main(String[] args){
        int String=10;
        System.out.println(String);
}}
```

Code Output:
10

Types of Variables in details :

Based the type of value represented by the variable all variables are divided into 2 types. They are:

1) Primitive variables
2) Reference variables (Non primitive variables)

Primitive variables: Primitive variables can be used to represent primitive values.
Example: int x=10;

Reference variables: Reference variables can be used to refer objects.
Example: Student s=new Student();

Based on the purpose and position of declaration all variables are divided into the following 3 types.

1) Instance variables
2) Static variables
3) Local variables

Instance variables:

Rules for Instance Variable in Java

1. The instance variables cannot be marked as static. Refer to this Static variable in java.
2. The instance variables must be declared outside the class methods , and any kind of scope.
3. The instance variables can be marked by the access specifiers.

4. Instance variables cannot be marked by the abstract keyword(abstract keyword only used for the methods).

Accessing Instance Variables
We can access the instance variable using two ways.

1. Using "**this**" keyword.
2. Using an object of the class.

Let's understand one by using Object.
Example:

```
public class Runner {
   int num;
   public static void main(String[] args)
   {
      // System.out.println(i);  //C.E:non-static variable i cannot
//be referenced from a static context(invalid)
       //Creating an instance of the class
       Runner t=new Runner();
       //Inistaizing the value of the instance
       t.num = 10;
       //Calling the method of the class
       t.printMe();
   }
   //Method to print the name of the Person
   public void printMe()
   {
       System.out.println(this.num);  //10 (valid)
   }
}
```

Let's understand one by using this keyword.

```
public class Runner {



   int i;
   public Runner(int i) {
       //Accessing the instance variable using this keyword.
```

```
        this.i = i;
    }

    public static void main(String[] args)
    {
        Runner r = new Runner(20);
        r.printMe();
    }
    public void printMe()
    {
        System.out.println(this.i);
    }
}
```

For the instance variables it is not required to perform initialization JVM will always provide default values.

Static variable

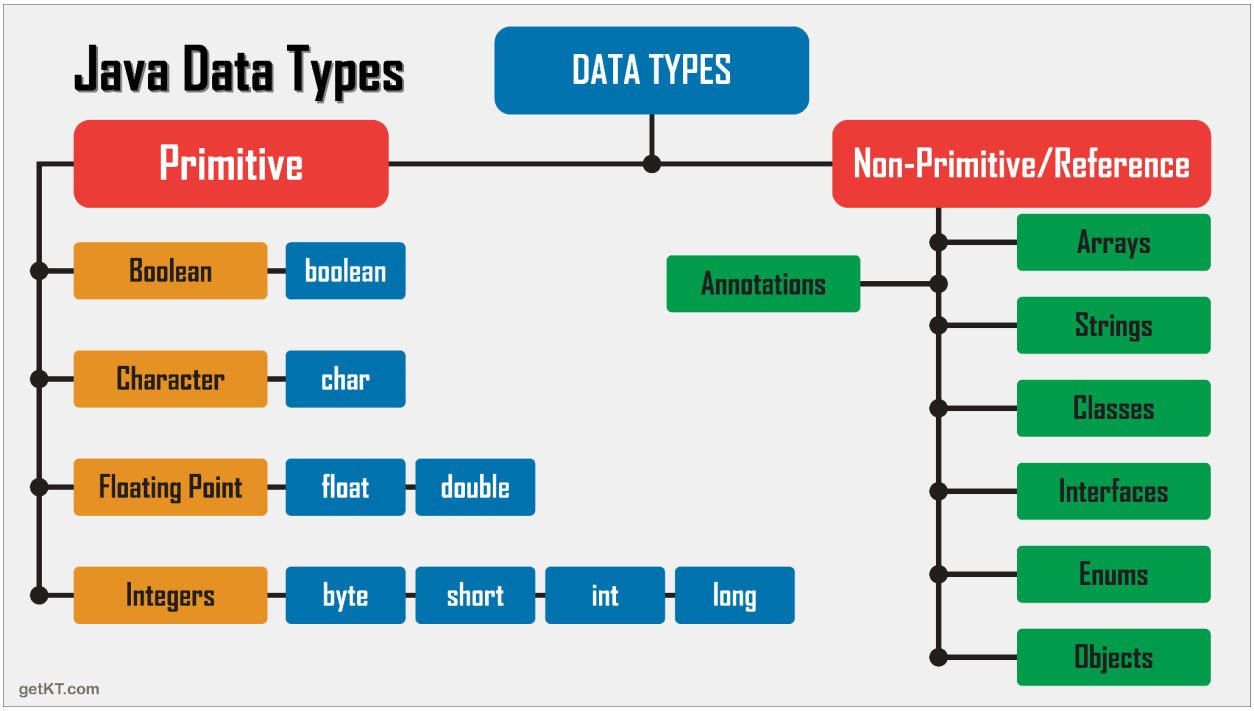
The main use of the static keyword in Java is memory management. Whenever we place a static keyword before initializing a particular class's methods or variables, these static methods and variables belong to the class instead of their instances or objects.

```
public class Runner {
    String project;
    static String empOrganization ;


    public static void main(String[] args)
    {
        Runner.empOrganization = "XYZ";

       Runner emp = new Runner();
       emp.project = "P1";
       emp.printEmpDetails();

       Runner emp2 = new Runner();
       emp.project = "P2";
       //emp.empOrganization = "xyz2";
       emp.printEmpDetails();
    }
    //Method to print the name of the Person
    public void printEmpDetails()
```

```
    {
        System.out.println("Emp project : " +project + ", from Org :
"+empOrganization);
    }
}
```

Reserved words for data types:

1) byte -
2) short
3) int
4) long
5) float
6) double
7) char
8) boolean



Byte data type **:**

- If we want to store very small numbers of very limited size. For e.g : the index of the notebook.
- The byte data values require 1 byte memory space in the computer's memory.
- The default value of the byte is 0.

Size : 1 byte
Range: -127 to 127

Example :
byte a=123;
System.out.print(a); //Printing 123
byte b=231; //Compiler will show type mismatch error

Short data type **:**

- The short data type is commonly used when we want to save memory in large arrays. For example, we can use a short data type for storing the admission number of the students of the school.
- The short data values require 2 bytes memory space in the computer's memory.
- The default value of the short data type is 0.

Size : 2 byte
Range: -32,768 to 32,767

Example:
short a=123;
System.out.println(a); //Printing the 123
short b= 32768; //Compiler will show error due to out of range value

int:

This is the most commonly used data type in java.
Size: 4 bytes
Range:-2147483648 to 2147483647 (-2^31 to 2^31-1)

Example:
int i=130;
int i=10.5;//C.E:possible loss of precision
int i=true;//C.E:incompatible types

long:

Whenever int is not enough to hold big values then we should go for a long data type.

Example:
To hold the no. Of characters present in a big file int may not enough hence the return type of length() method is long.

```
long num = Long.MAX_VALUE;
System.out.println(num);
```

Size: 8 bytes
Range:-2^63 to 2^63-1

Note: int & long can be used to represent whole numbers. If we want to represent real numbers then we should go for floating point data types.

Floating Point Data types:

float

1) If we want 6 to 7 decimal places of accuracy then we should go for a float.

2) Size:4 bytes.

3) Range:-3.4e38 to 3.4e38.

For example, 8.123456 this number can be stored in the float data type.

```
float num = 12.4f;
System.out.println(num);
```

The default value of the float data type is 0.0 and also we have to typecast the decimal values during initialization to float because by default the decimal is doubled in Java. So to prevent error we have to typecast the decimal values to float.

Double date type :
The double data type is most commonly used for storing large decimal values for example the value of pi i.e. 3.1473…

1) If we want 14 to 15 decimal places of accuracy then we should go for Double.

2) Size:8 bytes.

3) -1.7e308 to1.7e308.

```
double a=12312.23123;
System.out.print(a); //Printing the value of a
```

boolean data type:

- The boolean data type stores only two kinds of values i.e. false or true.
- The default value of the boolean data type is false.
- Memory allocation of the double data type is 1 byte.

```
boolean bl=false;
System.out.println(bl);//Printing the false boolean value
boolean ch="Sdfsd"; //Compiler will show compiler error
```

char data type**:**

- In java we are allowed to use any worldwide alphabets character and java is Unicode based to represent all these characters one byte is not enough compulsory we should go for 2 bytes.
- The default value of the char is ‘\u0000’.
- Memory allocation of the double data type is 2 bytes.
- The Range of char datatype is the ASCII value is -128 to 127.

Size: 2 bytes
Range: 0 to 65,535

Example:

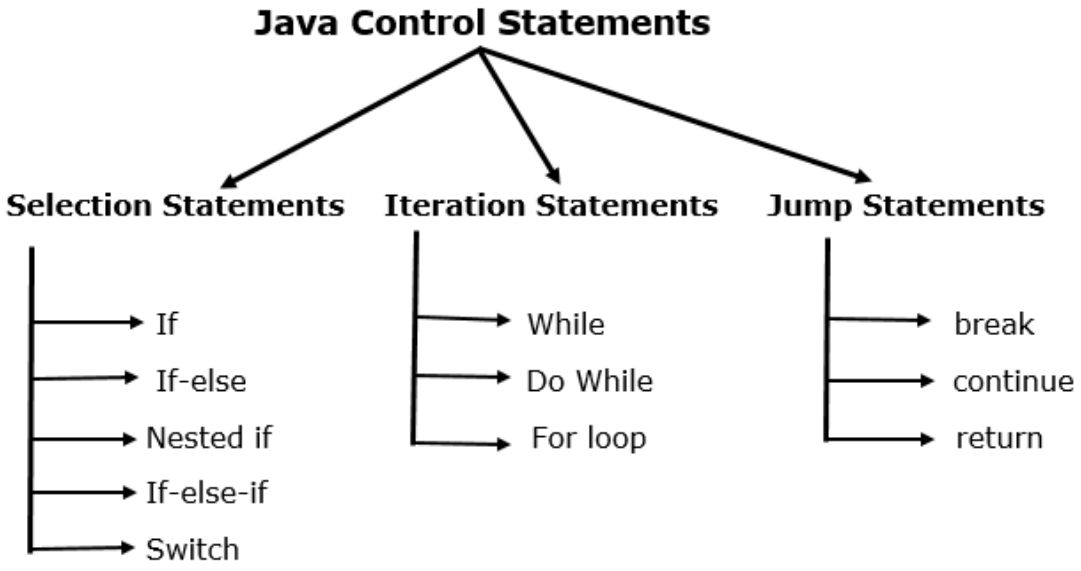
```
char ch1=97;
char ch2=65536;//C.E:possible loss of precision
char ch='a';
System.out.print(ch); //Printing the value of ch
```

Types of Operators in Java are

1. Arithmetic Operators (+, –, *, /, %)
2. Assignment Operators (=, +=, -=, *=, /=, %=)
3. Auto-increment Operator and Auto-decrement Operators (++, —)
4. Logical Operators (&&, ||, !)
5. Comparison (relational) Operators (==, !=, >, <, >=, <=)
6. Bitwise Operators (&, |, ^, ~, <<, >>)
7. Ternary Operator

Control Statements

Reserved words for flow control:
1) if
2) else
3) switch
4) case
5) default
6) for
7) do
8) while
9) break
10) continue
11) return



1. if-else

The argument to the if statement should be Boolean if we are providing any other type we will get “compile time error”.

Syntax:

```
if(b) // b is a boolean variable
{
        //action if ‘b’ is true
}

else
{
        // action if ‘b’ is false
}
```

EXAMPLE 1:

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

```
int x=0;
if(x)
{
System.out.println("hello");
}else
{
System.out.println("hi");
}}}
```

OUTPUT:
Compile time error

EXAMPLE 2:

```
public class ExampleIf
{
public static void main(String args[])
{
int x=10;
if(x==20)
{
System.out.println("hello");
}else
{
System.out.println("hi");
}}}
```

OUTPUT:
Hi

Notes:

1. Both else and curly braces are optional.
2. Without curly braces we can take only one statement under if, but it should not be declarative statement.

EXAMPLE:

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

```
{
boolean b=false;
if(b==true)
{
System.out.println("hello");
}
```

OUTPUT:

Nothing because value of b is false

2. Switch

The switch statement in Java is a multi branch statement. We use this in Java when we have multiple options to select. It executes a particular option based on the value of an expression.

Syntax:

```
switch(x)
{
case 1:
action1
case 2:
action2
.
.
.
default:
default action
}
```

1. Curly braces are mandatory.
2. Both case and default are optional.
3. Every statement inside the switch must be under some case (or) default. Independent statements are not allowed.

EXAMPLE 1:

```
public class ExampleSwitch{
public static void main(String args[]){
switch(x)
{
System.out.println("hello");
}}}
```

OUTPUT:
Compile time error

EXAMPLE 2:

```
public class ExampleSwitch{
public static void main(String args[]){
int x=10;
int y=20;
switch(x)
{
case 10:
System.out.println("10");
case y:
System.out.println("20");
}}}
```

OUTPUT:
Compile time error

Why?

case y:
If we declare y as final we won't get any compile time error.

EXAMPLE 3:

```
public class ExampleSwitch
{
public static void main(String args[])
{
int x=10;
final int y=20;
switch(x)
{
case 10:
```

```
System.out.println("10");
case y:
System.out.println("20");
}}}
```

OUTPUT:
10
20

Notes:
Switch argument and case label can be expressions also, but case should be constant Expression

FALL-THROUGH INSIDE THE SWITCH:

Within the switch statement if any case is matched from that case onwards all statements will be executed until the end of the switch (or) break. This is call “fall-through” inside the switch

DEFAULT CASE:

Within the switch we can take the default anywhere, but at most once it is convention to take default as last case

While loop:

if we don’t know the no of iterations in advance then best loop is while loop

EXAMPLE 1:

```
while(boolean)
{
----------
----------
----------
}
```

The argument to the while statement should be Boolean type. If we are using any other type we will get a compile time error.

EXAMPLE 1:

```
public class ExampleWhile{
public static void main(String args[]){
while(1)
{
System.out.println("hello");
}}}
```

OUTPUT:

Compile time error

EXAMPLE 2:

```
public class ExampleWhile{
public static void main(String args[]){
while(true)
System.out.println("hello");
}}
```

OUTPUT:

Hello (infinite times)

EXAMPLE 3:

```
public class ExampleWhile{
public static void main(String args[]){
while(true);
}}
```

OUTPUT:

No output

EXAMPLE 4:

```
public class ExampleWhile{
public static void main(String args[]){
final int a=10;
while(a<20)
{
System.out.println("hello");
}
System.out.println("hi");
}}
```

OUTPUT:

```
D:\Java>javac ExampleWhile.java
ExampleWhile.java:8: unreachable statement
System.out.println("hi");
```

Notes:

1. Every final variable will be replaced with the corresponding value by the compiler.
2. If any operation involves only constants then compiler is responsible to perform that operation.
3. If any operation involves at least one variable, the compiler won't perform that operation at runtime jvm is responsible for performing that operation.

Do-while

If we want to execute the loop body at least once then we should go for do-while.

Syntax:

```
do
{
} while(b); //semicolon is mandatory
```

1. Curly braces are optional
2. Without curly braces we can take only one statement between do and while and it should not be a declarative statement

Example 1:

```
public class ExampleDoWhile{
public static void main(String args[]){
do
System.out.println("hello");
while(true);
}}
```

Output:
Hello (infinite times)

Example 4:

```
public class ExampleDoWhile{
```

```
public static void main(String args[]){
do
{
int x=10;
}while(true);
}}
```

Output:
Compile successful

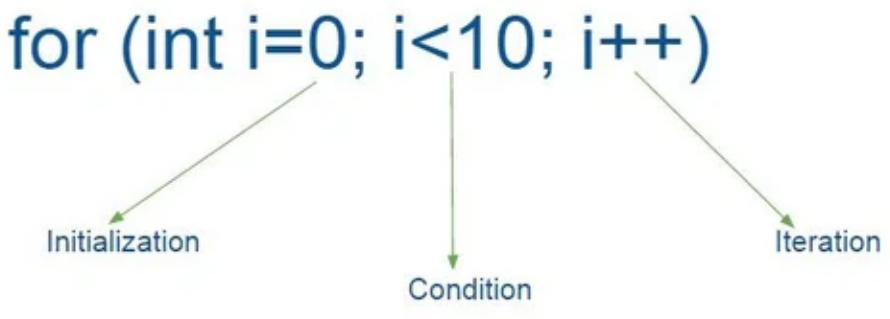
Example 8:

```
public class ExampleDoWhile{
public static void main(String args[]){
do
{
System.out.println("hello");
}
while(false);
System.out.println("hi");
}}
```

Output:
Hello
Hi

For Loop

This is the most commonly used loop and best suitable if we know the no of iterations in advance.



1) **Initialization section:**

- This section will be executed only once.
- Here usually we can declare loop variables and we will perform initialization.
- We can declare multiple variables but should be of the same type and we can't declare
- different types of variables.

Example:

int i=0,j=0; valid

2) **Conditional check:**

- We can take any java expression but should be of the type Boolean.
- Conditional expression is optional and if we are not taking any expression compiler will place true.

3) **Increment and decrement section:**

- Here we can take any java statement including s.o.p also.

Example:

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(i);
    }
  }
}
```

Output:

0
1
2
3
4

Example:

```
public class ExampleFo
r{
public static void main(String args[]
){
for(;;)
{
System.out.println("hello");
}}}
```

Output:
Hello (infinite times)

Notes:

Curly braces are optional and without curly braces we can take exactly one statement and it should not be a declarative statement

Example :

```
public class ExampleFor
{
public static void main(String args[])
{
for(int i=0;;i++)
{
System.out.println("hello");
}
System.out.println("hi");
}}
```

Output:

Compile time error

Enhanced For Loop

For each:

Best suitable to retrieve the elements of arrays and collections. In for-each loop traversal technique, you can directly initialize a variable with the same type as the base type of the array. The variable can be used to access the collection elements without any indexing. It changes to successive elements in the Collection after each iteration.

Syntax:

```
// datatype is same as that of the elements of the iterable
for(datatype element : iterable) {
    // Code
}
```

Example:

```
public class Main {

  public static void main(String args[]) {
    int arr[] = { 1, 2, 3, 4, 5, 6 };

    for (int elem : arr) {
      System.out.print(elem + ", ");
    }
  }
}
```

Output:

1, 2, 3, 4, 5, 6,

Jump Statements

Break

We can use break statement in the following cases:-

1) Inside switch to stop fall-through.
2) Inside loops to break the loop based on some condition.
3) Inside label blocks to break block execution based on some condition.

Example:

```
class HelloWorld {
   public static void main(String[] args) {
      int x = 10;

      System.out.println("Hello");

     for(int i=10; i<=x; i++)
      {
         System.out.println("World");
         break;
      }

      System.out.println("End");
   }
}
```

Output:

```
Hello
World
End
```

Note: If we try to use break statement without a loop, we will get a compile time error

Example:

```
class Test{
public static void main(String args[]){
int x=10;
if(x==10)
break;
System.out.println("hello");
}}
```

Output:
Compile time error

Continue

We can use continue statement to skip current iteration and continue for the next Iteration.

We can use continue only inside loops if we are using anywhere else we will get compile time error saying “continue outside of loop”.

Example:

```
class Test
{
public static void main(String args[]){
int x=10;
if(x==10);
continue;
System.out.println("hello");
}
}
```

Output:
Compile time error

Example:

```
public class ContinueExample {

public static void main(String[] args) {

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {
continue;
}
```

```
System.out.println(j);
}
}
}
}
```

Output:

```
0
1
2
3
5
1
2
3
5
2
3
5
```

Return

The return statement is used for returning a value when the execution of the block is completed. The return statement inside a loop will cause the loop to break and further statements will be ignored by the compiler

A return keyword is used for returning the resulting value.

The void return type doesn't require any return statement. If we try to return a value from a void method, the compiler shows an error.

Syntax:
The syntax of a return statement is the return keyword is followed by the value to be returned.

```
return returnvalue;
```

Example:

```
public class Test
```

```
{
    /* Method with an integer return type and arguments */
    public int CompareNum(int x, int y)
    {
        System.out.println("x = " + x + "\ny = " + y);
        if(x>y)
            return x;
        else
            return y;
    }



    /* Driver Code */
    public static void main(String ar[])
    {
        SampleReturn2 obj = new SampleReturn2();
        int result = obj.CompareNum(15,24);
        System.out.println("The greater number among x and y is: " + result);
    }
}
```

Output:

```
x = 15
y = 24
The greater number among x and y is: 24
```

Array in JAVA [1D]

- An array is an indexed collection of a fixed number of homogeneous data elements.
- The main advantage of arrays is we can represent multiple values with the same name so that readability of the code will be improved

But the main disadvantage of arrays is:

- Fixed in size that is once we created an array there is no chance of increasing or decreasing the size based on our requirement that is to use arrays concept compulsory
- We should know the size in advance, which may not always be possible

- We can resolve this problem by using collections

Array Declaration Example:

Single Dimension

int[] a;//recommended to use because name is clearly separated from the type
int []a;
int a[];

Note: At the time of declaration we can't specify the size otherwise we will get compile time Error.

Example:

int[] a;//valid

int[5] a;//invalid

Two Dimensional Array Declaration

Example:

int[][] a;
int [][]a;
int a[][];
int[] []a;
int[] a[];
int []a[];

Array construction: Every array in java is an object hence we can create by using new operator.

Example:
int[] a=new int[3];

Array initialization:

Whenever we are creating an array every element is initialized with default

value automatically.

Example 1:

```
int[] a=new int[3];
System.out.println(a); //[I@3e25a5
System.out.println(a[0]); //0
```

Array Out Of Bound Exception:

Example:

```
int[] a=new int[4];
a[0]=10;
a[1]=20;
a[2]=30;
a[3]=40;
a[4]=50; //R.E:ArrayIndexOutOfBoundsException: 4
a[-4]=60; //R.E:ArrayIndexOutOfBoundsException: -4
```

Note: if we are trying to access array element with out of range index we will get Runtime Exception saying ArrayIndexOutOfBoundsException.

Declaration of Array in Single Line:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Example:

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Arrays class in JAVA

Methods in Java Array Class
The static methods of the Java Array class could be used to perform operations like:

- Filling the elements
- Sorting the elements
- Searching for the elements
- Converting the array elements to String

```
int Arr[] = { 10, 20, 11, 21, 31 };
 int Arr1[] = { 10, 11, 21 };

                    Arrays.sort(Arr);

 System.out.println("Integer Array is: " + Arrays.toString(Arr));
 System.out.println("Integer Array is: "+ Arrays.toString(Arrays.copyOfRange(Arr, 2, 4)));
System.out.println("Integer Arrays on comparison are : " + Arrays.equals(Arr, Arr1));


 int Arr[] = { 10, 20, 11, 21, 31 };
       int Key = 23;
       Arrays.fill(Arr, Key);

       System.out.println("Integer Array on filling is: "
                  + Arrays.toString(Arr));
```

2D Array in JAVA

An array is a homogeneous data structure that stores data of the same type in contiguous memory locations. Multidimensional Arrays can be thought of as an array inside the array i.e. elements inside a multidimensional array are arrays themselves. They can hold more than one row and column in tabular form.

Multidimensional arrays, like a 2D array, are a bunch of 1D arrays put together in an array.

Print the given array as output :

```
public class Main {
   public static void main(String[] args){
      int[][] twoD_array = {
                           {1, 3, 4},
                           {9, 8, 6},
                           {2, 0, 5}
                         };
      for(int[] temp : twoD_array){
         for(int val : temp){
            System.out.print(val+" ");
         }
         System.out.println();
      }
   }
}
```

```
1 3 4
9 8 6
2 0 5
```

Another way of assigning value to a 2D array :

```
import java.util.*;

public class Main {
   public static void main(String[] args) {
      // 2D array to store characters
      char[][] CharacterArray = new char[2][4];
      // First row stores JAVA in capital letters
      CharacterArray[0][0] = 'J';
      CharacterArray[0][1] = 'A';
      CharacterArray[0][2] = 'V';
      CharacterArray[0][3] = 'A';
      //second-row stores java in small letters
      CharacterArray[1][0] = 'j';
      CharacterArray[1][1] = 'a';
      CharacterArray[1][2] = 'v';
      CharacterArray[1][3] = 'a';
      System.out.println(
            "Printing 2D array using Arrays.deepToString() method: ");
      // deepToString() method converts the 2D array to string as printed in the
```

```
        // output
        System.out.println(Arrays.deepToString(CharacterArray));
    }
}
```

o/p :
Printing 2D array using Arrays.deepToString() method:
[[J, A, V, A], [j, a, v, a]]

Functions or Methods :

Method/function describes the behavior of an Object.

A method consists of a collection of statements which performs an action.

Methods are also known as procedures or functions

Let's see an example of a method declaration.

```
public int sum(int a, int b, int c){

// method body

}
```

```
public void sum(int a, int b, int c){

// method body

}
```

The signature of the above declared method is :

Int sum(int a, int b, int c)

Every method declaration must have return type of the method, a pair of parenthesis, and a body between braces

Let's see how to call methods using an object:-

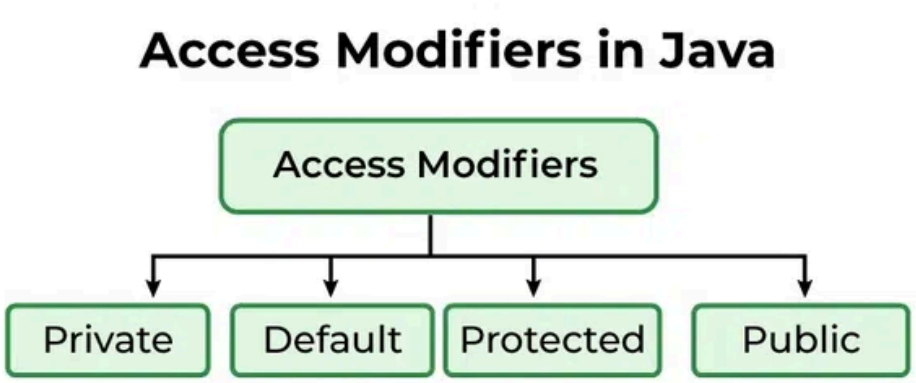
```
class Computer{

// method

void turnOn(){

        System.out.println("Started Successfully");

}

public static void main (String [] args){

          // Created an object

          Computer laptop = new Computer();

          //Method called

          laptop.turnOn();

}

}
```

Methods are of two types

1. **Built-in methods or Pre-defined methods**: Methods such as String methods, Date & Time methods, etc.,

2. **User Defined methods**: It contains 'method with returning value' and 'method without returning any value'

Access Modifiers: Access modifiers are subdivided into four types such as Default, Public, Private, Protected



default: The scope of default access modifier is limited to the package only. If we do not mention any access modifier, then it acts like a default access modifier.

private: The scope of private access modifier is only within the classes.

Note: Class or Interface cannot be declared as private

protected: The scope of protected access modifier is within a package and also outside the package through inheritance only.

Note: Class cannot be declared as protected

public: The scope of public access modifiers is everywhere. It has no restrictions. Data members, methods and classes that are declared public can be accessed from anywhere.

Quick Look, about how access levels work:

	ACCESS LEVELS			
MODIFIER	Class	Package	Subclass	Everywhere
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

Examples ;

```
class PrivateClassOne {
    private int x = 100;
    int y = 200;
    private int myMethod(int a){
        return a;
    }
    int myMethodOne(int a){
        return a;
    }
}

public class ProtectedClassFour {
        protected int myMethod(int a){
                return a;
        }
```

```
}


public class PublicClassSix {
        public int myMethod(int x){
                return x;
        }
}
```

Classes and Objects in Java

Class:

A class is a blueprint or prototype from which objects are created. A class contains **variables** (data types) and **methods** (functions) to describe the behavior of an object.

- A Java class doesn't take up memory.
- It acts as a template for organizing and defining the behaviour of objects in your program.

Syntax of Java Classes

One can declare a public, private, or default class as:

```
public class PublicClassName {
   // declaration section for
   // methods and attributes
}

private class PrivateClassName {
   // declaration section for
   // methods and attributes
}

class DefaultClassName {
   // declaration section for
   // methods and attributes
}
```

Example:

```
class Class_Name{
        //member variables
        //methods
}
```

Another Example of Java Class

```
class Car{
   // declaration of private attributes
   private String modelName;
   private String owner;
   private int regNumber;

   // declaration of public constructor
   public Car(String modelName, String owner, int regNumber){
      this.modelName = modelName;
      this.owner = owner;
      this.regNumber = regNumber;
   }

   // declaration of public methods
   public void startEngine(){
      System.out.println("Engine is starting ....");
   }

   public void accelerate(){
      System.out.println("Car is accelerting ...");
   }

   public void stop(){
      System.out.println("Car is stopping ...");
   }
   // prints car attributes
  public void showCarInformation(){
              System.out.println("The car is owned by: " + this.owner);
              System.out.println("Car Model: " + this.modelName);
              System.out.println("Registration Number: " + String.valueOf(this.regNumber));
```

```
   }
}
```

Explanation : The Car class contains private attributes hidden from the outside world. That means calling out the modelName attribute outside the scope of the Car class will raise a compiler error.

Object:

Object is a software bundle of related state and behavior. Objects have two characteristics namely state and behavior.

We can also say, Object is an entity that has state and behavior.

State: It represents value (data types/variables) of an object
Behavior: It represents the functionality (methods) of an object

Object is an instance of a class.

```
class Computer{
String Maker;
int Model;
String Color;
void turnOn{
        //statement(s)
}
void turnoff{
          //statement(s)
}
}
```

State: Maker, Model, Color etc.,
Behavior: Turn on, Turn off etc.,

To understand a class and object in detail, let me give you a basic example related to a computer. Computer with Model and Price.

Assume, you have two computers, Apple and Lenovo. Now say the model of Apple is MacBook Pro and the model of Lenovo is Yoga. The price of Apple is $299 and the price of Lenovo is $99.

Computer is a class which has two attributes namely Model and Price. Apple and Lenovo are the objects of the class Computer.

Let's see how to create an object:

Computer laptop = new Computer();

Class: Computer
Reference: laptop
Keyword: new
Constructor: Computer()
Object: new Computer()

Computer is a class name followed by the name of the reference laptop. Then there is a "new" keyword which is used to allocate memory. Finally, there is a call to the constructor "Computer()". This call initializes the new object "new Computer()".

We create an Object by invoking the constructor of a class with the new keyword.

Constructors:

A constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

Types of Java constructors

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default Constructor

A constructor is called a "Default Constructor" when it doesn't have any parameters.

Syntax of default constructor:
<class_name>(){}

Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example:

```
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Output:
111 Karan
222 Aryan

Static KeyWord:

We can apply static keywords with variables, methods, blocks and nested classes. The static keyword belongs to the class rather than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Nested class

Java static variable

If you declare any variable as static, it is known as a static variable.

Notes:

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Example:

```
//Java Program to demonstrate the use of static variable
class Student{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   //constructor
   Student(int r, String n){
   rollno = r;
   name = n;
   }
   //method to display the values
   void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
```

```
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

Output:

```
111 Karan ITS
222 Aryan ITS
```

Java static method

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access a static data member and can change the value of it.

Example:

```
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
```

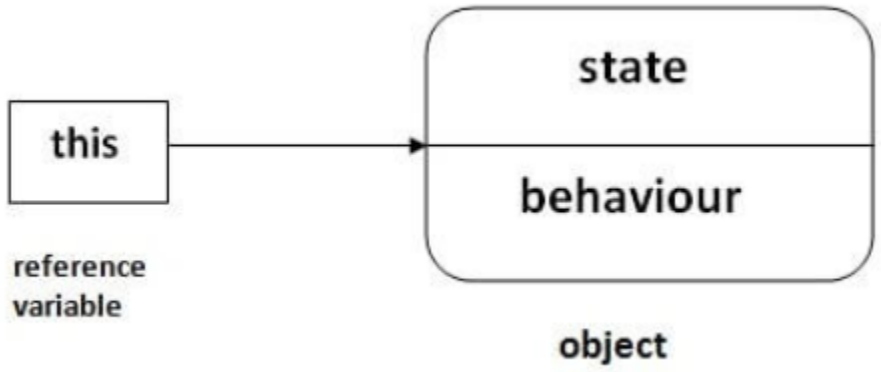
```
    s1.display();
    s2.display();
    s3.display();
    }
}
```

Output:

```
111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT
```

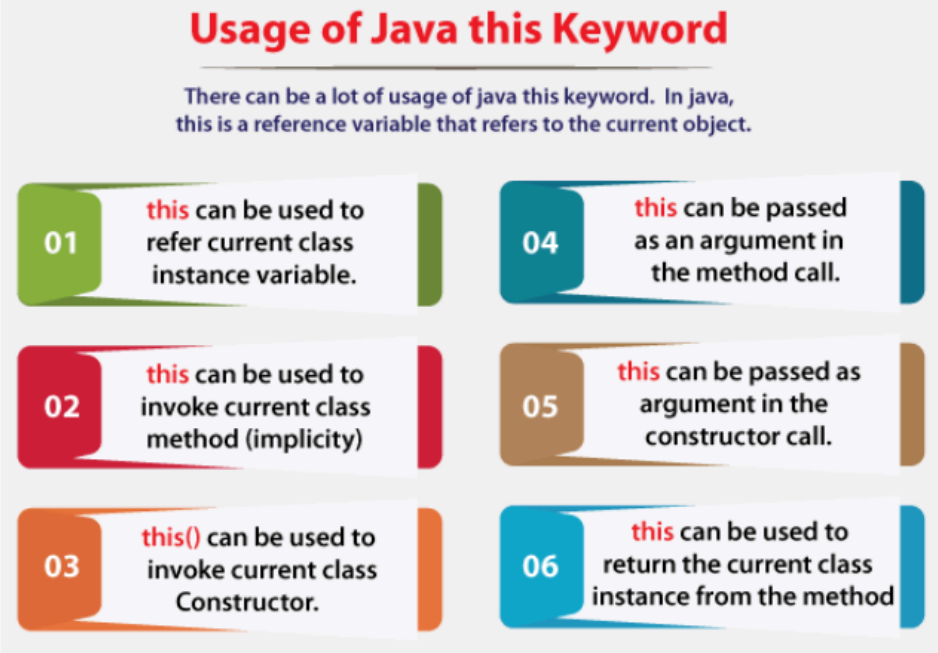
Notes:

The static method can not use non-static data members or call non-static methods directly. this and super cannot be used in a static context.



This is a reference variable that refers to the current object.

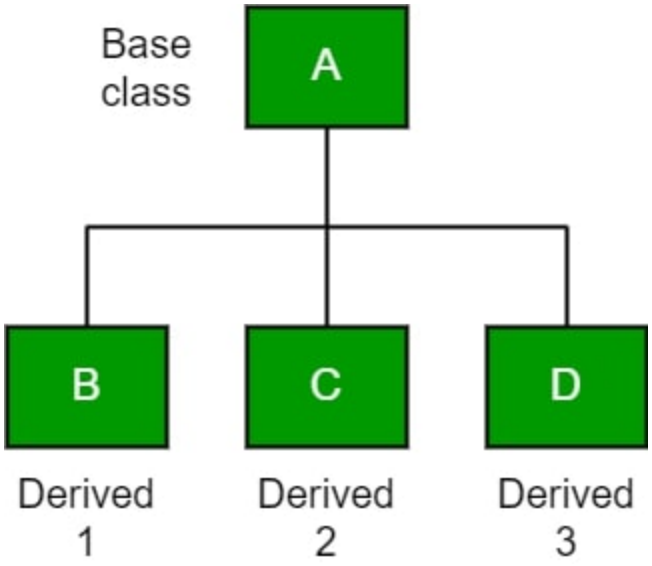
Uses of This KeyWord



Core OOPS:

Inheritance:

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).



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.

Polymorphism:

Polymorphism allows us to perform a task in multiple ways. Let's break the word Polymorphism and see it, 'Poly' means 'Many' and 'Morphos' means 'Shapes'.

There are two types of Polymorphism in Java
1. Compile time polymorphism (Static binding) – Method overloading
2. Runtime polymorphism (Dynamic binding) – Method overriding

Method Overloading:

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

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

Advantage of method overloading
Method overloading increases the readability of the program.

Different ways to overload the method
There are two ways to overload the method in java

By changing number of arguments
By changing the data type

Method Overriding:

Declaring a method in child class which is already present in the parent class is called Method Overriding.

In simple words, overriding means to override the functionality of an existing method.

In this case, if we call the method with a child class object, then the child class method is called. To call the parent class method we have to use the super keyword.

```
//Parent Class
public class MethodOverridingParentClass {

	public void myMethod(){
		System.out.println("I am a method from Parent Class");
	}

}

//Child Class
public class MethodOverridingChildClass extends MethodOverridingParentClass{

	public void myMethod(){
		System.out.println("I am a method from Child Class");
	}
```

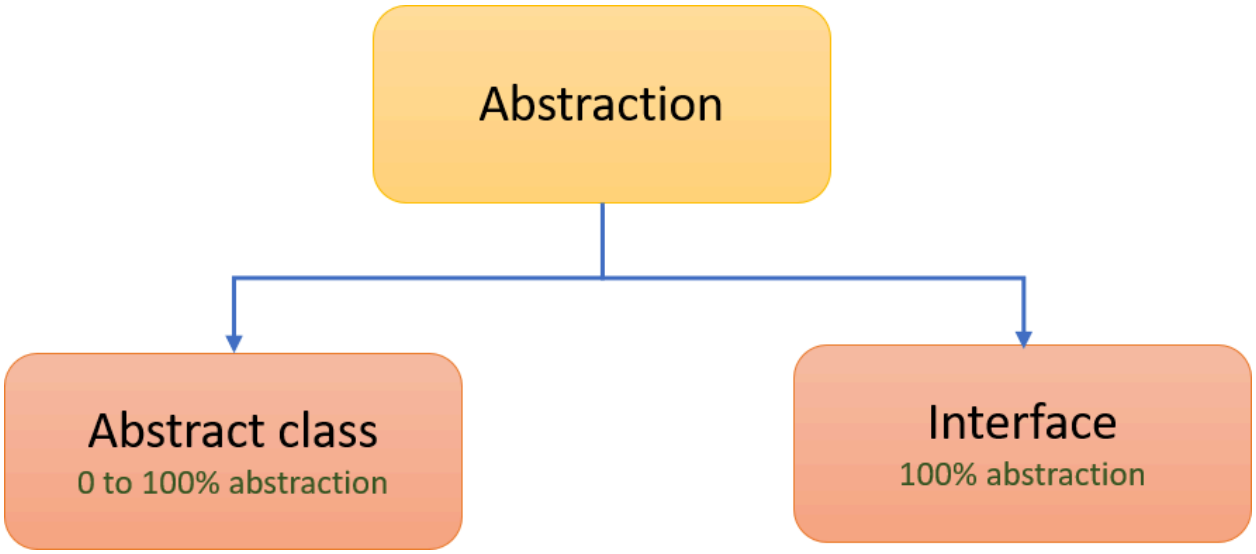
```
        public static void main(String [] args){

                //creating object
                MethodOverridingChildClass obj = new MethodOverridingChildClass();
                // It calls the child class method myMethod()
                obj.myMethod();

        }
}
```

Output:

I am a method from Child Class

Abstraction:

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.



Abstract Class:

We can easily identify whether a class is an abstract class or not. A class which contains abstract keywords in its declaration then it is an Abstract Class.

Syntax:

abstract class <class-name>{}

Points to remember:

1. Abstract classes may or may not include abstract methods
2. If a class is declared abstract then it cannot be instantiated.
3. If a class has abstract method then we have to declare the class as abstract class
4. When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Abstract Method:

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

abstract void myMethod();
In order to use an abstract method, you need to override that method in subclass.

Let's see an example:

To create an abstract class, just use the abstract keyword before the class keyword, in the class declaration.

```
// Here class is abstract
public abstract class AbstractSuperClass {

        // myMethod() is an abstract method
        abstract void myMethod();

}
```

Interface

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in the interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface.

Points to remember:

1. Java interface represents IS-A relationship similar to Inheritance
2. Interface cannot be instantiated same like abstract class
3. Java compiler adds public and abstract keywords before the interface methods
4. Java compiler adds public, static and final keywords before data members
5. Interface extends another interface just like a Class extends another Class but a class implements an interface.
6. The class that implements the interface must implement all the methods of that interface.
7. Java allows you to implement more than one interface in a Class

```
public interface MyInterface {

        // Compiler treats it as public abstract void myMethodOne();
        // Below method has no body
        public void myMethodOne();

}
```

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Encapsulation:

Encapsulation is a mechanism of binding code and data together in a single unit. Let's take an example of Capsule. Different powdered or liquid medicines are encapsulated inside a capsule. Likewise in encapsulation, all the methods and variables are wrapped together in a single class.

```
public class EncapsulationClassOne {

    // Variables declared as private
    // These private variables can only be accessed by public methods of class
```

```
        private int age;
        private String name;

    // getter method to access private variable
        public int getAge(){
                return age;
        }

        public String getName(){
                return name;
        }

    // setter method to access private variable
        public void setAge(int inputAge){
                age = inputAge;
        }

        public void setName(String inputName){
                name = inputName;
        }

}


public class EncapsulationClassTwo {

        public static void main(String [] args){

                EncapsulationClassOne obj = new EncapsulationClassOne();
            // Setting values of the variables
                obj.setAge(25);
                obj.setName("Bishnu");

                System.out.println("My name is "+ obj.getName());
                System.out.println("My age is "+ obj.getAge());

        }

}
```

Output:

```
My name is Bishnu
My age is 25
```

Exception Handling :

Keywords for exception handling:
1) try
2) catch
3) finally
4) throw
5) throws

Exceptional handling in Java is a very powerful mechanism as it helps to identify exceptional conditions and maintain the flow of the program as expected, by handling/avoiding the errors if occurred. In some cases, it is used to make the program user-friendly.

What is an Exception in java?
An exception is an unwanted or unexpected event that occurs during the execution of the program, that disrupts the flow of the program.

Examples :

```
class SampleCode {
  public static void main(String args[]) {
    Sysytem.out.println("Hello World!");
    int a = 10;
    int b = 0;
    System.out.println(a / b);
    System.out.println("Welcome to java programming.");
    System.out.println("Bye.")
  }
}
```

```
Hello World!
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at SampleException.main(SampleException.java:8)
```

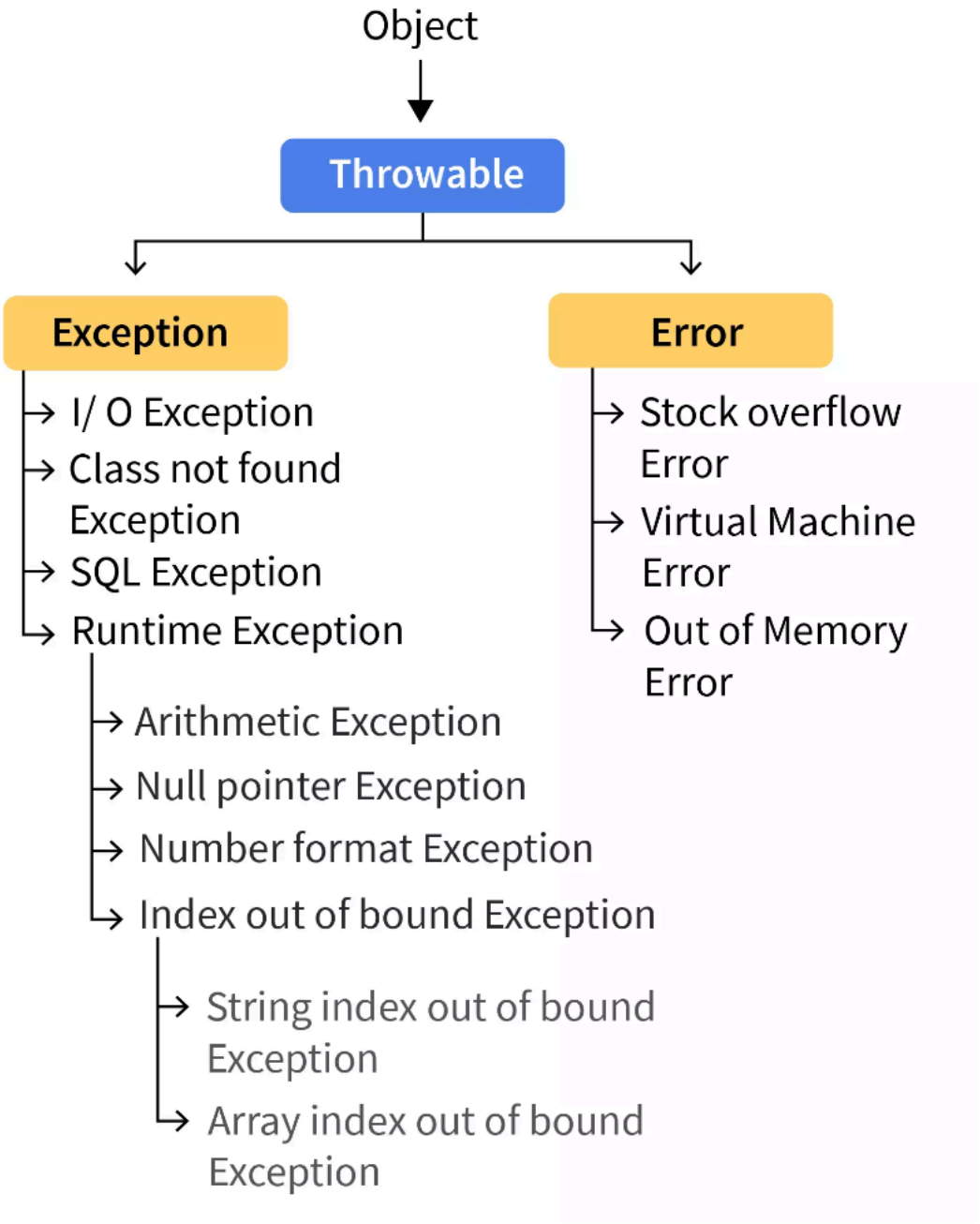
In the above code, the first three lines in the main method are executed properly. At the 4th line, an integer is divided by 0, which is not possible and an exception is raised by JVM(Java Virtual Machine). In this case, the exception is not handled by the programmer which will halt the program in between by throwing the exception, and the rest of the lines of code won't be executed.

The parent Exception class in java is Throwable class.

How java handles Exceptions in java :-
Whenever an exception has occurred inside a method, the method creates an exception object and hands it over to JVM.

This object contains the name and description of the exception and the current state of the program where the exception has occurred. This is called the process of object creation and handing it over to JVM is known as throwing an Exception. This object is an exception that is further handled by JVM.

Hierarchy of Java Exception Classes



Types of Exceptions Handling in Java

1. Checked Exceptions

- Checked exceptions are those exceptions that are checked at compile time by the compiler.
- The program will not compile if they are not handled.
- These exceptions are child classes of the Exception class.

- IOException, ClassNotFoundException, SQL Exception are a few of the checked exceptions in Java.

2. Unchecked Exceptions

- Unchecked exceptions are those exceptions that are checked at run time by JVM, as the compiler cannot check unchecked exceptions,
- These are child classes of Runtime Exception Class.
- ArithmeticException, NullPointerException, NumberFormatException, IndexOutOfBoundException are a few of the unchecked exceptions in Java.

How does a Programmer Handles an Exception?

Customized exception handling in java is achieved using five keywords: try, catch, throw, throws, and finally. Here is how these keywords work in short.

- Try block contains the program statements that may raise an exception.
- Catch block catches the raised exception and handles it.
- Throw keyword is used to explicitly throw an exception.
- Throws keyword is used to declare an exception.
- Finally block contains statements that must be executed after the try block.

Example :

```
class ExceptionExample {
  public static void main(String args[]) {
    try {
      // Code that can raise exception
      int div = 509 / 0;
    } catch (ArithmeticException e) {
      System.out.println(e);
    }
    System.out.println("End of code");
  }
}
```

```
java.lang.ArithmeticException: / by zero
End of code
```

Another example :

```
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {10, 1, 2, 3, 5, 11};
```

```
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

Something went wrong.

Multiple catch block :

```
public class MultipleCatchBlock1 {

  public static void main(String[] args) {

    try {
      int a[] = new int[5];
      a[5] = 30 / 0;
    } catch (ArithmeticException e) {
      System.out.println("Arithmetic Exception occurs");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("ArrayIndexOutOfBounds Exception occurs");
    } catch (Exception e) {
      System.out.println("Parent Exception occurs");
    }
    System.out.println("End of the code");
  }
}
```

o/p : Arithmetic Exception occurs
End of the code

Multiple catch blocks

```
public class MultipleCatchBlock2 {

  public static void main(String[] args) {

    try {
      int a[] = new int[5];

      System.out.println(a[10]);
    } catch (ArithmeticException e) {
```

```
      System.out.println("Arithmetic Exception occurs");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("ArrayIndexOutOfBounds Exception occurs");
    } catch (Exception e) {
      System.out.println("Parent Exception occurs");
    }
    System.out.println("rest of the code");
  }
}
```

finally block

- finally block is associated with a try, catch block.
- It is executed every time irrespective of exception is thrown or not.
- finally block is used to execute important statements such as closing statement, release the resources, and release memory also.
- finally block can be used with try block with or without catch block.

Example :

```
public class Main {
  public static void main(String[] args) {
    try {
      int data = 100/0;
      System.out.println(data);
    } catch (Exception e) {
      System.out.println("Can't divide integer by 0!");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

Interview Questions :

1. Why is the Java main method static?

   It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then calls main() method that will lead to the problem of extra memory allocation.

2. Can we execute a program without the main() method?

   Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main method.

Collection Framework:

What is java collection ?

- A collection - can be called as container - is simply an object that groups multiple elements into single unit.
- Example : List, Set and Map

Benefits of the Java Collections Framework

The Java Collections Framework provides the following benefits:

1. Reduces programming effort by providing data structures and algorithms so you don't have to write them yourself.
2. Increases performance by providing high-performance implementations of data structures and algorithms.

Commonly used implementations

The Java Collections Framework provides several general-purpose implementations of the core interfaces:

- For the **Set interface**, HashSet is the most commonly used implementation.
- For the **List interface**, ArrayList is the most commonly used implementation.
- For the **Map interface**, HashMap is the most commonly used implementation.

ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.

Example 1: Creating an ArrayList and Adding New Elements

The below example demonstrates how to create an ArrayList using the ArrayList() constructor and add new elements to an ArrayList using the **add()** method.

```
import java.util.ArrayList;
import java.util.List;

// How to create an ArrayList using the ArrayList() constructor.
// Add new elements to an ArrayList using the add() method.
public class CreateArrayListExample {

    public static void main(String[] args) {
        // Creating an ArrayList of String using
     List<String> fruits = new ArrayList<>();
        // Adding new elements to the ArrayList
     fruits.add("Banana");
     fruits.add("Apple");
     fruits.add("mango");
     fruits.add("orange");
        System.out.println(fruits);
    }
}
```

Output:

[Banana, Apple, mango, orange]

Set & HashSet

The Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.
HashSet implementation class provides no ordering guarantees. The ***LinkedHashSet implementation class maintains the insertion order.***

Methods in HashSet

add()

contains()

remove()

clear()

size()

isEmpty()

Set Interface with Its HashSet Implementation Class

```
// Creating a HashSet
HashSet<String> daysOfWeek = new HashSet<>();

// Adding new elements to the HashSet
daysOfWeek.add("Monday");
daysOfWeek.add("Tuesday");
daysOfWeek.add("Wednesday");
daysOfWeek.add("Thursday");
daysOfWeek.add("Friday");
daysOfWeek.add("Saturday");
daysOfWeek.add("Sunday");

// Adding duplicate elements will be ignored
daysOfWeek.add("Monday");

System.out.println(daysOfWeek);
```

Output:
[Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday]

HashMap & Maps

A map is a key-value mapping, which means that every key is mapped to exactly one value and that we can use the key to retrieve the corresponding value from a map.

Uses and advantages of HashMaps

1. A hashmap basically specifies a unique key for those values that can be recovered at any point.
2. Java HashMap is a class that is used to perform tasks such as putting, removing and detecting elements in a map. We create a map, where we pass two types of values that are ‘key’ and ‘value’.

Creation of HashMap:

HashMap hashmap = new HashMap();

Add any element in HashMap you need to provide 2 thing, key and value.

Key : key with which specified value will be associated. null is allowed.
Value : value to be associated with specified key.

Example:

```
import java.util.Map;

class Hashmap_Java {

  public static void main(String args[]) {

  HashMap<Integer, String> map = new HashMap<Integer, String>();

      map.put(100, "Mumbai");

      map.put(101, "Delhi");

      map.put(102, "Pune");
```

```
        // Add Element
        map.put(103, "Tata");

        // Size of map
        System.out.println(map.size());

        // clears hashmap , removes all element
        map.clear();

        // Remove element from hashmap
        map.remove(100);

        // Checking if HashMap is empty
        System.out.println("Is HashMap is empty: " + map.isEmpty());

        for (Map.Entry m : map.entrySet()) {

        System.out.println(m.getKey() + " " + m.getValue());

        }

    }

}
```

Output:

```
4
Is HashMap is empty: true
```

Methods in HashMaps:

put(), get(), isEmpty(), size() and many more