JAVA INTRODUCTION:

A Program is a collection of set off instructions.

A Collection of programs that tell the computer how to work is called **Software**

C is the First programming language developed in early 1970 by Dennis M. Ritchie at Bell Laboratories.

C is a Structed/Procedural programming language. Its execution will take place step by step

The disadvantages of Procedural programming language includes unrelated execution of code, no code reuse functionality, no security to data

C++ is an objected oriented programming language by Bjarne Stroustrup at Bell Labs in the 1980s.

Any programming language is an object-oriented program language if it follows the principles of Abstraction, Encapsulation, Inheritance, Polymorphism.

C/C++ are System programming languages.

Java is an objected oriented programming language developed by James Gosling at Sun Microsystems, who is known as the father of Java, in 1995.

Java is an internet/Web programming language.

Java is a platform independent programming language. It means Java Programs can be run on any platform (operating System) like Windows, Macintosh, Solaris, Unix etc.

.Net is an object oriented programming language. But it is a platform dependent programming language. It means .Net programs can run only on windows operating system and not any other operating system. But the latest version of .Net allows to run the .Net applications in all operating systems

Differences between Java and .Net:

Java is platform independent technology whereas .Net is platform dependent technology

Java is a lightweight language whereas .Net is a heavy weight framework

.Net supports only one standard development IDE whereas java supports many IDEs

Java has support to open source platform while .Net has no direct support for Open source Platforms

History Of Java:

Java is developed by Sun microsystems in 1995

Java's first name is OAK

Java is used for developing Web applications, Desktop applications, Enterprise applications, Web and application servers, Mobile apps

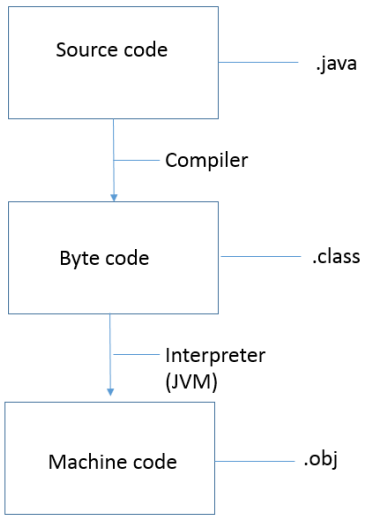
Java is mainly divided into 3 parts:

JAVA STANDARD EDITION

JAVA ENTERPRISE EDITION

JAVA MICRO EDITION

JAVA ARCHITECTURE:



Source code:

source code is any collection of instructions written using a human-readable programming language, usually as plain text.

Byte code:

Bytecode is program code that has been compiled from source code into low-level code designed for a software interpreter.

Machine code:

Machine code, consisting of machine language instructions, is a low-level programming language used to directly control a computer's central processing unit (CPU).

Compiler:

A *compiler* is a program that translates a source program written in some high-level programming language into low level language like Byte code

Interpreter(JVM):

An interpreter is a computer program that directly executes machine code instructions and runs java programme.

Java Features:

1.**Simple**:

Java is very easy to learn, and its syntax is simple, clean and easy to understand.

2. **Object-oriented:**

Java basically is an object oriented programing language and it follows the below four principles:

a.Abstraction b.Encapsulation c.Inheritance d.Polymorphism

3.**Secured**:

With Java's secure feature it enables to develop virus-free systems. Authentication techniques are based on public-key encryption.

4.**Platform independent:**

Java programes can run on any platform(operating system) like Macintosh,Linux,windows etc.

5.**Robust:**

Robust simply means strong. Java is robust because It uses strong memory management.

- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

6.**Portable:**

Java is portable because it allows to carry the Java bytecode to any platform.

7.**Architecture Neutral**:

Java is architecture neutral because the size of primitive types is fixed for both 32 bit and 64 bit operating systems

8.**Dynamic:**

Java is a dynamic language. It supports dynamic loading of classes from other programmes. It also supports functions from its native languages, i.e., C and C++.

9.**Interpreted:**

Java byte code is translated to machine code during run time and is not stored anywhere.

10. **High Performance**:

With the use of Just-In-Time compilers, and garbage collections ,Java enables high performance.

11. **Multithreaded**:

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously.

12.**Distributed:**

Java is designed for the distributed environment of the internet.

First Java Programme:

```
public class Demo
{
public static void main(String args[])
{
System. out. println("this is my first java programme");
}
}
```

Important Points:

JAVA is a case sensitive programming language, meaning capital letters and small letters are different in java programme.

Names used for classes, variables, and methods are called **identifiers**

Variables are reserved memory locations to store values

Data types specify the different sizes and values that can be stored in the variable

Data types are mainly of two types: 1. Primitive 2.Non Primitive

Primitive Data types:

Java supports the below 8 primitive data types:

Data type	Usage	Default Size	Values
boolean	Used for simple flags that track true/false conditions	1 bit	true false
char	used to store characters	2 byte	‘a’ , ‘b’ , ‘c’ ….etc
byte	Used to store integral values	1 byte	Minimum value is -128 Maximum value is 127
short	Used to store integral values	2 byte	Minimum value is -32,768 Maximum value is 32,767
int	Used to store integral values	4 byte	Minimum value is - 2,147,483,648 Maximum value is 2,147,483,647
long	used when you need a range of values	8 byte	Minimum value is -9,223,372,036,854,775,808 Maximum value is 9,223,372,036,854,775,807
float	Used to store decimal values	4 byte	stores 6 to 7 decimal digits
double	Used to store decimal values	8 byte	stores 15 decimal digits

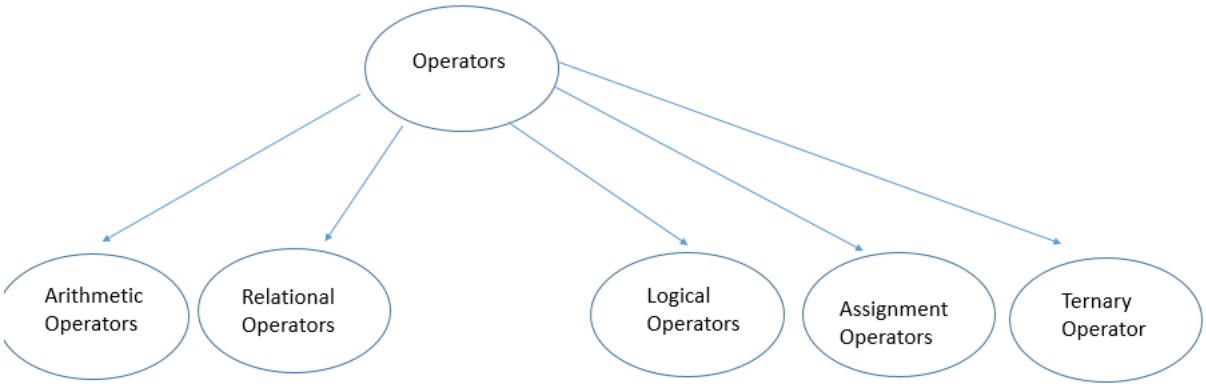
Example:

```
public class Hello
{
```

```
Public static void main(String args[])
{
 int val1=10;
char val2='c';
double val3=45.56d;
boolean val4=true;
System.out.println("the integer value is"+val1);
System.out.println("the character value is"+val2);
System.out.println("the double value is"+val3);
System.out.println("the boolean value is"+val4);
}
}
```

Operators

Java provides operators to manipulate variables



Arithmetic Operators: Arithmetic operators are used for mathematical expressions, they are :

+ - * / % ++ --

Example:

```
public class Hello
{
```

```
public static void main(String args[])
{
int val1=10;
int val2=20;
int val3;
val3=val1+val2;
System.out.println("value3 is"+val3);
        }
}
```

The Relational Operators: Relational operators are used for relational expressions, they are:

== != > < >= <=

Example:

```
public class Hello
{
public static void main(String args[])
{
int val1=10;
int val2=20;
System.out.println(val1>val2);

}
}
```

The Logical Operators: Logical operator works for logics , they are:

&& || !

Example:

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

```
{
boolean val1=true;
boolean val2=false;
System.out.println(val1&&val2);

}
}
```

The Assignment Operators: Assignment operators are used for assignation, they are:

= += -= *= /= %= &= !=

Example:

```
public class Hello
{
public static void main(String args[])
{
int val1=10;
int val2;
val2+=val1;
System.out.println("the value of val2 is"+val2);

}
}
```

The Ternary Operator: Ternary operator is used for evaluating conditions

Syntax:

Expression ? Value if true : value if false

Example:

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

```
{
int val1=10;
int val2;
val2=(val1==10)?30:40;
System.out.println("the value of val2 is"+val2);

}
}
```

Conditional Statements:

1. If statement

Syntax:

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

Example:

```
public class Hello
{
public static void main(String args[])
{
int val1=10;
int val2=10;
if(val1==val2)
{
System.out.println("inside if block");
 }
System.out.println("this is last statement");
}
}
```

2. If- else statement

Syntax:

```
if(condition)
{
```

```
}
else
{
}
```

Example:

```
public class Hello
{
public static void main(String args[])
{
int val1=10;
int val2=20;
if(val1==val2)
{
System.out.println("inside if block");
 }
else
{
System.out.println("inside else block");
}
System.out.println("this is last statement");
 }
}
```

3. else if statement

Syntax:

```
if(condition)
{
}
else if(condition)
{
}
else
{
```

```
}
Example:
public class Hello
{
public static void main(String args[])
{
int val1=10;
int val2=20;
if(val1==30)
{
System.out.println("inside if block");
 }
else if(val2==30)
{
System.out.println("inside else if block");
}
else
{
System.out.println("inside else statement");
}
System.out.println("this is last statement");
}
}
```

4.switch statement:

Syntax:

```
Switch(variable value)
Case variable value 1:
{
…
break;
}
….
```

```
default:
{
…
break;
}
```

Example:

```
public class Demo2
{
public static void main(String args[])
{
  int aa=2;
 switch(aa)
{
        case 1:
        {
        System.out.println("executing switch code1");
                break;
        }
        case 2:
        {
        System.out.println("executing switch code2");
                break;
        }
        default:
        {
        System.out.println("executing switch default code");
                break;
        }
}
}
}
```

Looping Statements:

A loop statement allows us to execute a statement or group of statements multiple times

1. While statement

Syntax:

while(condition)

{

}

Example:

```
public class Demo2
{
public static void main(String args[])
{
 int aa=0;
        while(aa<=2)
        {
                System.out.println("executing while loop");
                aa++;
        }
}
}
```

2.do while statement

Syntax:

```
do
{
}
While(condition)
```

Example:

```
public class Demo2
{
public static void main(String args[])
{
```

```
 int aa=0;
        do
        {
        System.out.println("executing while loop");
                aa++;
        }
        while(aa<=2);

}
}
```

3.for loop

Syntax:

```
for(initialization of operator; condition; increment/decrement operator )
{
}
```

Example:

```
public class Demo2
{
public static void main(String args[])
{
        for(int i=0;i<=5;i++)
        {
        System.out.println("executing while loop");
        }

}
}
```


Arrays:

An array is a data structure, which allows to store fixed-size data and similar kind of data

Syntax:

datatype array name[] = new datatype[array size];

Example:

```
public class Demo3
{
public static void main(String args[])
{
        int myarray[]=new int[5];
        myarray[0]=10;
        myarray[1]=20;
        myarray[2]=30;
        myarray[3]=40;
        myarray[4]=50;
        for(int i=0;i<=4;i++)
        {
                System.out.println(myarray[i]);
        }
 }
}
```

Command Line Arguments:

The command-line argument is an argument(value) that is passed at the time of running the java program.

Example:

```
public class Demo2
{
public static void main(String args[])
{
                System.out.println(args[0]);
 }
 }
```

Type Casting(Conversion):

Assigning the values of one data type to another data type is called type casting

Types of type casting:

1.Implicit Type casting

2.Explicit Type casting

1.Implicit Type casting:

This is done automatically,

a.Data types should be compatible

b.Destination data type size should be larger than source data type size

Example:

```
public class Hello
{
public static void main(String args[])
{
        int aa;
        double dd;
        aa=10;
        dd=aa;
System.out.println("the value of dd is "+dd);
} }
```

2.Explicit type casting:

This has to be done manually and can be done for datatypes of different sizes

```
public class Hello
{
public static void main(String args[])
{
        double dd;
        int aa;
        dd=10.4d;
        aa=(int)dd;
System.out.println("the value of aa is "+aa);  }  }
```

Object Oriented Programing System (OOPS):

JAVA supports the below 4 OOPS concepts:

1.Abstraction 2.Encapsulation 3.Inheritance 4.Polymorphism

1.Abstraction:
It is an act of representing the essential features hiding the background details

2.Encapsulation:
It is a mechanism of combining the data and code in to a single unit

Class:
Class is a collection of data and methods

Syntax:
Class classname
{
…….
}

Method:
Method is a set of instructions defined inside a class and performs some operations on the data

Synatax:
Method name
{
…….
}

Object:
Object is an instance of a class. Once object is created, memory is allocated to the data.

Synatax:
classname objectname=new classname();

Example:
```
class First
{
int aa=50;
public void demo()
{
System.out.println("executing demo method");
}
}

public class Hello
{
public static void main(String args[])
{
First ff=new First();
ff.demo();
System.out.println("the value of aa is "+ff.aa);
}
```

```
}
```

void keyword:
If a method does not return any value, we need to use void keyword for that method in method signature

return keyword:
If a method returns any values, we need to use return keyword in the method definition

Example:

```
 class One
{
public void show()
{
System.out.println("this is show method");
}
public int dummy()
{
int ff=34;
ff++;
return ff;
}
}
public class ThreadPrior
{
public static void main(String args[])
{
One oo=new One();
oo.show();
int val=oo.dummy();
System.out.println("the int value is"+val);
}
}
```

Call by value:
If we call(invoke) any method by passing a value we call it as call by value.

Example:

```
class One
{
public void dummy(int val)
{
val++;
System.out.println(val);}
}
public class ThreadPrior
{
public static void main(String args[])
{
One oo=new One();
oo.dummy(23);
}
}
```

3.Inheritance:
Inheritance can be defined as the process where one class acquires the properties (methods and
variables)

```
Class A
{
…….
}
Class B extends A
{
…….
}
```

Types of Inheritance:
1.Single Inheritance 2.Multi level Inheritance 3.Hierarchial Inheritance

1.Single Inheritance:
In this inertance there will be one parent class and one child class

Example:

```
class First
{
public void demo()
{
System.out.println("executing demo method");

}
}
class Second extends First
{
public void demo1()
{
System.out.println("executing demo1 method");
}
}
public class Hello
{
public static void main(String args[])
{
Second ss=new Second();
ss.demo();
ss.demo1();
}
}
```

2.Multi level Inheritance:
In this inheritance, there will be a minimum of three classes and child class again becomes the parent class

Example:

```
class First
{
public void demo()
```

```
{
System.out.println("executing demo method");
}
}

class Second extends First
{
public void demo1()
{
System.out.println("executing demo1 method");
}
}
class Third extends Second
{
public void demo2()
{
System.out.println("executing demo2 method");
}
}
public class Hello
{
public static void main(String args[])
{
Third tt=new Third();
tt.demo();
tt.demo1();
tt.demo2();
}
}
```

3.Hierarchial Inheritance:
In this inheritance, there will be one parent class and more than one child class

Example:

```
class First

{
public void demo()
{
System.out.println("executing demo method");
}
}
class Second extends First
{
public void demo1()
{
System.out.println("executing demo1 method");
}
}
class Third extends First
{
public void demo2()
{
System.out.println("executing demo2 method");
}
```

```
}
public class Hello
{
public static void main(String args[])
{
Second ss=new Second();
Third tt=new Third();
ss.demo();
ss.demo1();
tt.demo();

tt.demo2();
}
}
```

3.Polymorphism:
Polymorphism is the ability of an object to take on many forms.

Types of Polymorphism:
1.Static polymorphism 2.Dynamic polymorphism

1.Static polymorphism:
This is done at compile time through method overloading concept

2.Dynamic polymorphism:
This is done at run time through method overriding concept

Method overloading:
Two or more methods can have same name but they should differ in their parameters list in terms of number of parameters ,datatypes of parameters

Example:

```
class First
{
public void demo()
{
System.out.println("executing demo method");
}
public void demo(float ff)
{
System.out.println("executing demo method and value of ff is "+ff);
}
}
public class Hello
{

public static void main(String args[])
{
First ff=new First();
ff.demo();
ff.demo(34.5f);
} }
```

Method overriding:

Redefining the functionality of a method of superclass in subclass

Example:

```
class First
{
public void demo()
{
System.out.println("executing demo method of super class");
}
}
class Second extends First
{
public void demo()
{
System.out.println("executing demo method of sub class");
} }
public class Hello
{
public static void main(String args[])
{
Second ss=new Second(); ss.demo();
} }
```

Constructor:

Constructors are used to initialize the objects

1.Constructor name is same as that of class name
2.Constructor will not have any return type
3.Constructors will be called automatically once an object is created

Types of Constructors:
1.Default constructor 2.Parameterised constructor

1.Default constructor:
In this constructor there will not be any parameters

Example:

```
public class Hello
{
public static void main(String args[])
{
Hello1 one=new Hello1();
}
}
class Hello1
{
public Hello1()
{
System.out.println("executing constructor code");
}
}
```

2.Parameterised constructor:

In this constructor parameters are passed.

Example:

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

{
Hello1 one=new Hello1(56.2d);
}
}
class Hello1
{
public Hello1(double dd)
{
System.out.println("the value of dd inside constructor is "+dd);
}
}
```

Constructor overloading:

Two or more constructors can have same name but they should differ in their parameters in terms of number of parameters ,datatypes of parameters

Example:

```
public class Hello
{
public static void main(String args[])
{
Hello1 one=new Hello1(56.2d);
Hello1 two=new Hello1(67);
}
}
class Hello1
{
public Hello1(double dd)
{
System.out.println("the value of dd inside constructor is "+dd);

}
public Hello1(int kk)
{
System.out.println("the value of kk inside constructor is "+kk);
}
}
```

super:

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Example:

```
class One
{
public One()
{
System.out.println("executing constructor One code");
}
char cc='c';
public void demo1()
{
System.out.println("executing demo1 method code");
}
}
class Two extends One
{
public Two()
{
super();
System.out.println("executing constructor two code");
}
public void demo()
{
System.out.println("the value of cc is "+super.cc);
super.demo1();
}

}
public class Hello
{
public static void main(String args[])
{
Two oo=new Two();
oo.demo();
}
}
```

packages:
A javapackage is a group of similar types of classes, interfaces

Advantages:

package provides access protection.

Package is used to categorize the classes and interfaces so that they can be easily maintained

package removes naming collision.

Syntax:
package packagename;
import packagename.classname;

Types of packages:
1.pre defined packages 2.user defined packages

1.pre defined packages –These are inbuilt packages which contains predefined classes and predefined interfaces

2. user defined packages –These are our own packages created and contains the user defined classes and user defined interfaces

Example:

```
package javaexamples;

public class Hello

{

public void demo()

{

        System.out.println("executing demo code");

}

}
```

```
import javaexamples.*;

public class Hello1

{

public static void main(String args[])

{

Hello hh=new Hello();

hh.demo();

}

}
```

Sub package --A package inside another package is called sub package

String class:

String is a pre defined class and is a non primitive data type to store more than one character value

Important methods:

equals()

toUpperCase()

toLowerCase()

concat()

Example:

```
public class Hello
{
public static void main(String args[])
{
String ss="java";
String ss1=new String("JAVA");

if(ss.equals(ss1))
{
System.out.println("values are same");
}
else
{
System.out.println("values are not same");
}
System.out.println(ss.toUpperCase());
System.out.println(ss.toLowerCase());
System.out.println(ss.concat(ss1));
}
}
```

```
 public class Hello
{
public static void main(String args[])
{
//String aa="hyd";
//String bb="hyd";
String cc=new String("blr");
String dd=new String("blr");

if(aa.equals(bb))
{
System.out.println("values are equal");
}
else
{
System.out.println("values are not equal");
}
}
}
```

StringBuffer : StringBuffer is mutable in nature whereas String class is immutable.

Important methods:

append() --this method will joins two string values permanently
insert(index,string value) --- this method will insert string values in specified location
reverse() --- this method will reverse the string value

Example

```
public class Hello
{
public static void main(String args[]) {
//String aa="hyd";
```

```
String bb="hyd";
StringBuffer cc=new StringBuffer("blr");
StringBuffer dd=new StringBuffer("hyd");
dd.insert(2,"e");
System.out.println(cc.append(dd));
System.out.println(cc);
System.out.println(dd);
}
}
```

Modifiers:

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

There are four types of Java access modifiers:

1. **Private 2 Default 3 Protected 4 Public**

1.Private:

The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

2. Default : The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. Protected : The access level of a protected modifier is within the package and outside the package through child class.

4. Public : The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Example:

```
class One
{
private int age=10;
public void demo()
{
System.out.println("the value age is "+age);
}
}
class Two extends One
{
}
public class Hello
```

```
{
	public static void main(String args[])
	{
	Two oo=new Two();
	System.out.println("the value age is "+Two.age);
	}
}
```

Non Access Modifiers:

static , final , abstract

static : static keyword in is used for memory management mainly, We can apply static keyword with variables, methods,blocks of code

Syntax:

classname.method();

classname.variable name;

```
class One
{
	static int aa=45;
 static public void demo1()
	{
		System.out.println("class demo1 method executing...");
	}
}
public class Hello
{
	public static void main(String args[])
	{
	    System.out.println(One.aa);
		One.demo1();
	}
}
```

final: final keyword can be used for classes, methods, variables.

Example:

```
final class One
{
 final int kk=45;
public void demo()
        {
                kk=55;
        System.out.println("the value of KK is "+kk);
        }
}
class Two extends One
{
 public final void demo()
        {
                kk=65;
        System.out.println("the value of KK is "+kk);
        }
}
public class Hello
{
 public static void main(String args[])
        {
                Two tt=new Two();
                tt.demo();
        }}
```

abstract: abstract keyword is used for classes and methods only. If a method does not have any body ,it is called as abstract method. If a class contains at least one abstract method it is called as abstract class

Example:

```
abstract class One
{
int kk=45;
```

```
abstract public void demo();
}
class Two extends One
{
public void demo()
{
System.out.println("executing demo method code");
}
 }
public class Hello
{
 public static void main(String args[])
        {
                Two tt=new Two();
                tt.demo();
        }
}
```

Interface: Interface is similar to that of class but it will contain all the methods with no definitions

The keyword ‘implements’ will be used for a class to provide interface definitions

Example:

```
interface One
{
public void demo();
}
class Two implements One
{
public void demo()
{
System.out.println("executing demo method");
}
}
public class Hello
{
public static void main(String args[])
{
Two oo=new Two();
oo.demo();
}
}
```

Wrapper classes:

Wrapper classes are helpful for converting the primitive data types values into objects(autoboxing) and objects to primitive data type values(unboxing)

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Example:

```
 public class Hello
{
public static void main(String args[])
{
float val=30.4f;float ff1;String ss="12";
Float ff=new Float(val);
System.out.println(ff);
ff1=ff;
System.out.println(ff1);
float val5=Float.parseFloat(ss);
System.out.println(val5);
int val6=Integer.parseInt(ss);
System.out.println(val6);
}
}
```

Integer.parseInt(),Float.parseFloat() ---- these method are used to convert String objects into primitive data types

Exception Handling:
Exception Handling is a mechanism to handle runtime errors

Types of Exception: Checked Exception, Unchecked Exception

Checked Exception: Checked exceptions are checked at compile-time
Ex: IOException, SQLException etc

Un Checked Exception: Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
Ex: ArithmeticException,NullPointerException,ArrayIndexOutOfBoundsException etc

Advantages of Exception Handling:
1.Useful information to the user
2.To continue the normal flow of the application
3.To return the resources normally

Exception Handling mechanism:

try: In try block we need to place the code which might generate exception

catch: In catch block we need to place the code to handle exception

finally: finally block is executed irrespective of try catch blocks

throw: throw keyword is used to explicitly throw an exception(creating our own exception)

throws: throws keyword is used to handle multiple number of checked exceptions

one try many catch blocks are allowed

Exception class : This is the parent class for all the exception classes

Example1:

```
public class Hello
{
public static void main(String args[])
{
try
{
System.out.println("the passed argument is "+args[0]);
}
catch(Exception e)
{
System.out.println("please pass the arguments from command prompt");
}
finally
{
System.out.println("executing finally code");
}
}
}
```

Example2:

```
 Class One
{
Public void display()
```

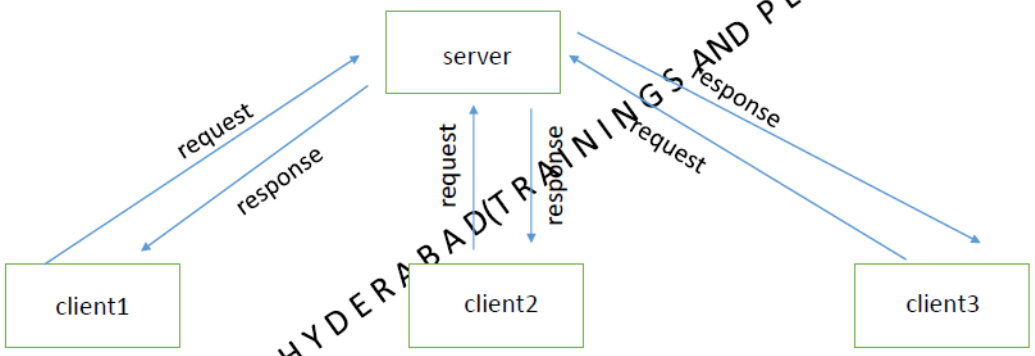
```
{
Int age=5;
If(age<21)
{
throw new ArithmeticException("not a valid age");
}
}
}
Public class Hello
{
Public static void main(String args[])
{
One obje=new One();
Obje.display();
}
}
```

Example3:

```
Import java.io.*;
Class One
{
Public void display() throws IOException
{
Int age=50;
If(age<21)
{
System.out.println(age);
}
}
}
```

```
Public class Hello
{
Public static void main(String args[])
{
Try{
One obje=new One();
Obje.display();
}
Catch(Exception ee)
{
System.out.println(ee);
}
}
}
```

Multi threading:



Thread is a separate path of the execution. Executing of several threads simultaneously is multithreading

Multithreading helps to execute threads as sequentially, synchronously and asynchronously

Multi Tasking:
Execution of several tasks at the same time is multi tasking

Difference between multi tasking and multi threading:

In multitasking CPU switches between programs(tasks) frequently,
In multithreading CPU switches between the threads frequently.

In Multitasking CPU will execute multiple tasks at the same time
In Multithreading CPU will execute multiple threads of a single task simultaneously.

In multitasking OS will allocate separate memory and resources to each program(task) that CPU is executing

In multithreading system will allocate memory to one process(task), multiple threads of that process(task) shares the same memory and resources allocated to the process.

Life cycle of thread:

1.New: Once the thread is created, it will be in New state

2.Runnable: In this state the thread instance is invoked with a start method and thread will start its operation

3.Running: When the thread starts executing, then the state is changed to "running" state

4.Waiting: In this state the thread will be waiting for synchronization.

5.Dead: once the processing is over , the thread will be in dead state

Thread Scheduler: Thread scheduler in java is the part of the JVM that decides which thread should run.

Important classes ,methods:

1.Thread class---This class is to create a thread
2.Runnable Interface –This interface is used to create a thread

Methods:
1.start() ---this method is used to start the thread

2.run() ---this method is used to perform some operation for thread

3.sleep(time in milliseconds) ---this method causes a thread to sleep for specified time

4.join() ---this method will make other threads to wait till the current thread completes its operation

5.currentThread() ---this method returns the reference of currently executing thread

6.setPriority() ----this method sets the thread priorityExample:

Example1:

```
class One extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
try{
System.out.println(i);
System.out.println(Thread.currentThread());
Thread.sleep(2000);
}
```

```
catch(Exception e)
{
System.out.println(e);
}
}
}
}
public class ThreadPriority
{
public static void main(String args[])
{
One oo=new One();
oo.setPriority(1);
oo.start();
One oo1=new One();
oo1.setPriority(10);
oo1.start(); }}
```

Example2:

```
class One implements Runnable
{
public void run()
{
for(int i=0;i<10;i++)
{
try{
System.out.println(i);
System.out.println(Thread.currentThread());
Thread.sleep(2000);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
}
public class ThreadPrior
{
public static void main(String args[])
{
One oo=new One();
Thread tt=new Thread(oo);
tt.start();
One oo1=new One();
Thread tt1=new Thread(oo1);
tt1.start(); }}
```

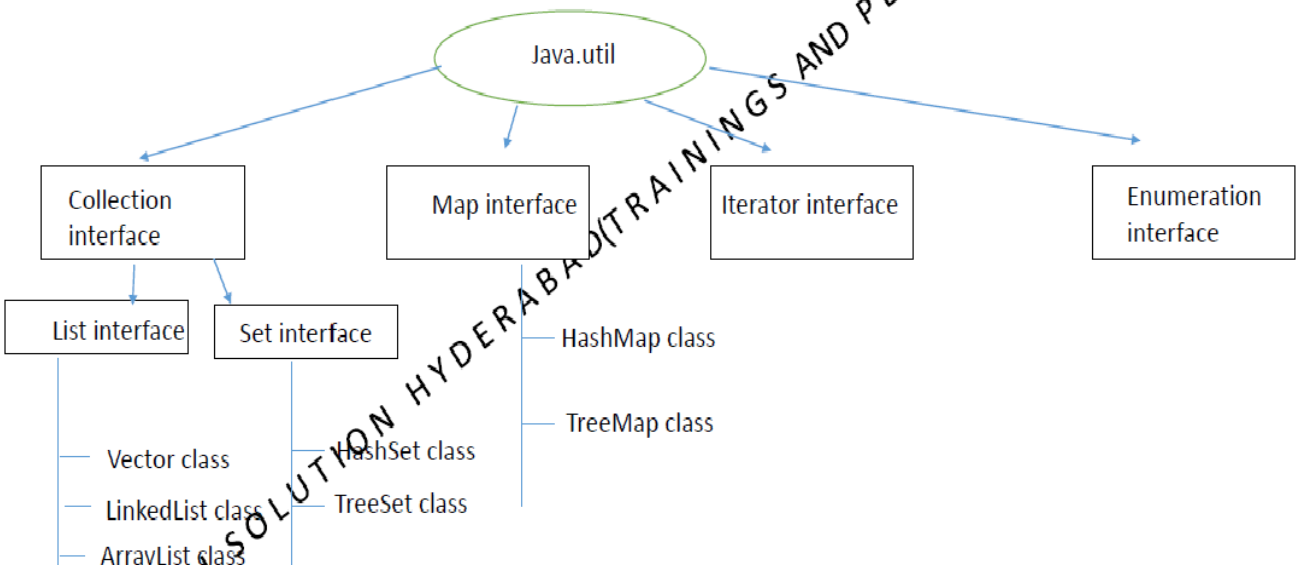
Example3:

```
class One implements Runnable
{
public void run()
{
```

```
for(int i=0;i<10;i++)
{
try{
System.out.println(i);
System.out.println(Thread.currentThread());
Thread.sleep(2000);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
}
public class ThreadPrior
{
public static void main(String args[])
{
One oo=new One();
Thread tt=new Thread(oo);
tt.start();
try{
tt.join();
}
catch(Exception ee)
{}
One oo1=new One();
Thread tt1=new Thread(oo1);
tt1.start();
}
}
```


Collections

Collection is a framework(predefined classes and interfaces) that provides an architecture to store and update a group of objects.



ArrayList:
It is a dynamic array to store different data types objects. It accepts duplicates. It is non-synchronized one.

Methods:
add(object);
remove(index)
get(index)

```
import java.util.*;

public class One

{

	public static void main(String aa[])

	{

		ArrayList l=new ArrayList();

		l.add("hyderabad");

		l.add("chennai");

		System.out.println(l);

		//l.remove(0);

		System.out.println(l);

		System.out.println(l.get(0));

	}}
```

Vector:
It is similar to ArrayList but Vector is by default synchronized one. It accepts duplicates.

Methods:
addElement(object);
removeElement(object);
get(index);

```
import java.util.*;

public class Two

{

	public static void main(String aa[])

	{

		Vector l=new Vector();

		l.addElement("hyderabad");

		l.addElement("chennai");

		l.addElement("mumbai");
```

```
                System.out.println(l);
                //l.removeElement("chennai");
                //System.out.println(l);
                System.out.println(l.get(0));
        }
}
```

LinkedList:

LinkedList uses a doubly linked list internally to store the elements.
It can store the duplicate elements. It is not synchronized.

In LinkedList, the updation is fast

Methods:
addFirst()
addLast():
getFirst()
getLast()
removeFirst()
removeLast()

```
import java.util.*;

public class Three extends LinkedList

{
        public static void main(String aa[])
        {
                Three  l=new Three();
                l.addFirst("hyderabad");
                l.addFirst("chennai");
                //System.out.println(l);
                l.addLast("mumbai");
                l.addLast("pune");
                //System.out.println(l);
                String ss=(String)l.getFirst();
                //System.out.println(ss);
                String sg=(String)l.getLast();
                //System.out.println(sg);
                l.removeFirst();
                System.out.println(l);
```

```
                l.removeLast();
                System.out.println(l);


        }
}
```

HashSet
In HashSet data is stored using hash table(key-value pair) . No duplicates are allowed
Objects are stored not in order

Methods:
add(object);
remove(object);

```
import java.util.*;

public class Four
{
        public static void main(String aa[])
        {
                Set l=new HashSet();
                l.add("rahul");
                l.add("priya");
                l.add("sumit");
                //l.add("sumit");
                System.out.println(l);
                l.remove("priya");
                System.out.println(l);
        }
}
```

TreeSet
In TreeSet objects are stored in the form of tree .No duplicates are allowed,
access and retrieval time of TreeSet is quite fast.
The objects in TreeSet stored is in ascending order.

Methods:
add(object);
remove(object);

HashMap
In HashMap data is stored using hash table(key-value pair) .In this, Keys also we need to provide.

No duplicates are allowed, Objects are stored not in order

Methods:
put(key,value);
get(key);

```
import java.util.*;

public class Five

{

	public static void main(String aa[])

	{

		Map l=new HashMap();

		l.put("1","java");

		l.put("2","dot net");

		String ss=(String)l.get("2");

		Integer a=new Integer(10);

		l.put("key",a);

		System.out.println(l);

	}

}
```

TreeMap

In TreeMap data is stored using hash table(key-value pair). In this Keys we need to provide. No duplicates are allowed, Objects are stored in order

Methods:
put(key,value);
get(key);

Iterator
Iterator allows to traverse a collection. It can be used for any collection framework

Methods:
iterator()---to get iterator object.
hasNext()-to check if elements are available

next()-to move to next objects

remove()-to remove objects

```
import java.util.*;

public class Six

{
```

```
	public static void main(String aa[])
	{
		//Iterator
		ArrayList l=new ArrayList();
		l.add("a");
		l.add("b");
		l.add("c");
		l.add("d");
		System.out.println(l);
		Iterator ii=l.iterator();
		int i=0;
		while(ii.hasNext())
		{
			Object o=ii.next();
			if(i==2)
			{
				ii.remove();
				System.out.println("removed");
			}i++;
		}
		System.out.println(l);
			}}
```

Enumeration
Enumeration is also used to traverse a collection, but does not contain remove().And it is used for Vector and HashTable.

Methods:
elements()
hasMoreElements()

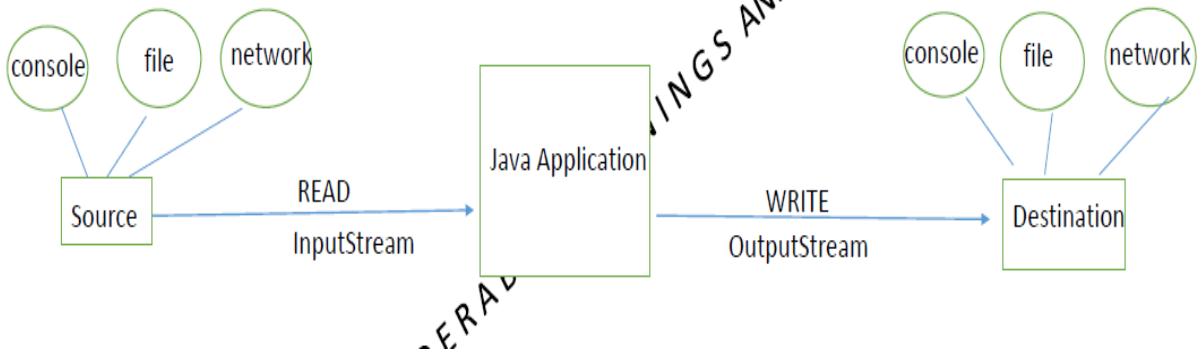
nextElement()

```
import java.util.*;
public class Seven extends Vector
{
	public static void main(String aa[])
	{
```

```
        //Enumerator
        Vector l=new Vector();
        l.addElement("a");
        l.addElement("b");
        l.addElement("c");
        l.addElement("d");
        Enumeration ii=l.elements();
        int i=0;
        while(ii.hasMoreElements())
        {
                Object o=ii.nextElement();
                System.out.println("No.is"+" "+o);
                        }
                        }
}
```


JAVA Input Output(I/O) Streams

Stream: Stream is flow of water, but in software it means flow of data



InputStream:

If java application gets the data from a source, it is InputStream

Read operation is done with InputStream

OutputStream:

If java application sends the data to a destination, it is OutputStream

Write operation is done with OutputStream

Stream Types:
1.Byte oriented streams 2.Character Oriented streams

1.Byte oriented streams: Byte streams perform the read write operation byte by byte(8-bit)

2.Character oriented streams: Character streams perform the read write operation character by character
(16-bit UNICODE)

Byte oriented streams:

Java.io.*;

InputStream classes
DataInputStream
FileInputStream
BufferedInputStream
SequenceInputStream
ObjectInputStream

OutputStream classes
DataOutputStream
FileOutputStream
ObjectOutputStream

InputStream classes

DataInputStream: This class is used to read the data from the keyboard

readLine() --this method is used for reading the bytes
close() ---this method is used for closing the stream object

```
import java.io.*;
public class One
{
public static void main(String aa[])throws IOException
{
DataInputStream di=new DataInputStream(System.in);
System.out.println("enter ur name");
String name=di.readLine();
System.out.println("hello "+name);
di.close();
}
}
```

FileInputStream: This class is used to read the data from a file

read() --this method is used for reading the bytes

close() --this method is used for closing the stream object

```
 public static void main(String aa[])throws Exception
{
FileInputStream fi=new FileInputStream("zero.txt");
int i=fi.read();
while(i!=-1)
{
System.out.print((char)i);
i=fi.read();
}
fi.close();
}
}
```

BufferedInputStream: This class will read the bytes fast and increases the performance of the application

read() --this method is used for reading the bytes
close() --this method is used for closing the stream object

```
import java.io.*;
public class Three
{
public static void main(String aa[])throws Exception
{
FileInputStream fi=new FileInputStream("Two.java");
BufferedInputStream br=new BufferedInputStream(fi);
long t=System.currentTimeMillis();
int i=br.read();
while(i!=-1)
{
System.out.print((char)i);
i=br.read();
}
long t1=System.currentTimeMillis();
System.out.println("time taken is"+(t1-t)+"milliseconds");
br.close();
fi.close();
}
}
```

SequenceInputStream: This class will read more from multiple streams. It read data sequentially

read() --this method is used for reading the bytes
close() --this method is used for closing the stream object

```
import java.io.*;
public class Four
```

```
{
public static void main(String aa[])throws Exception
{
FileInputStream fi=new FileInputStream("Two.java");
FileInputStream fa=new FileInputStream("Three.java");
```

```
SequenceInputStream si=new SequenceInputStream(fi,fa);
int i=si.read();
while(i!=-1)
{
System.out.print((char)i);
i=si.read();
}
si.close();
}
```

ObjectInputStream: This class will read the objects of a class
readObject();   --this method will read the bytes of an object

OutputStream classes:

DataOutputStream: This class is used to writing the java data types to a stream

write() --this method is used for writing the bytes
flush() --this method is used for flushing the data
close() --this method is used for closing the stream object

FileOutputStream: This class is used for writing the data to a file

write()
close()

```
import java.io.*;
public class Dataout
{
public static void main(String[] args) throws IOException
{
int aa=66610855;
```

```
FileOutputStream file = new FileOutputStream("testout.txt");
DataOutputStream data = new DataOutputStream(file);
data.write(aa);
data.flush();
data.close();
file.close();
}
}
```

ObjectOutputStream: This class will write the objects of a class

writeObject(); --this method will write the bytes of an object
close(); --this method will close the object stream

```
import java.io.*;
class Student implements Serializable
{
int id;
String name;
Student()
{
id=5;
name="manoj";
}
void display()
{
System.out.println("id is"+id);
System.out.println("name is"+name);
}
}
```

```
public class Five
{
public static void main(String aa[])throws Exception
{
Student s=new Student();
FileOutputStream fos=new FileOutputStream("Student.txt");
ObjectOutputStream os=new ObjectOutputStream(fos);
os.writeObject(s);
FileInputStream fi=new FileInputStream("Student.txt");
ObjectInputStream oi=new ObjectInputStream(fi);
Student ss=(Student)oi.readObject();
ss.display();
oi.close();
fi.close();
os.close();
}
}
```

2.Character oriented streams:

Reader
FileReader
BufferedReader
InputStremReader

Writer
FileWriter

RandomAccessFile

1.FileReader : This class is used for reading the data from the file in the form of characters(ASCII format)

Methods:
read()
close()

```
import java.io.*;
public class FR
{
public static void main(String aa[])throws Exception
{
FileReader fi=new FileReader("Two.java");
int i=fi.read();
while(i!=-1)
{
System.out.print((char)i);
i=fi.read();
}
fi.close();
}
}
```

2.BufferedReader : This class will read the data fast and increases the performance of the application

Methods:
close()
readLine()

3.InputSteamReader : This class is a bridge from byte streams to character streams. It reads bytes and converts them into characters

Methods:
close()
read()

```
import java.io.*;
public class eight
{
public static void main(String aa[])throws Exception
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("enter the name");
String ss=br.readLine();
System.out.println("ur name is"+ss);
br.close();
}
}
```

4.FileWriter : This class is used for writing the data to the file

Methods:
close()
read() ---to read the file

```
import java.io.*;
public class FW
{
public static void main(String aa[])throws Exception
{
String val="34";
FileWriter fos=new FileWriter("Student1.txt");
```

```
fos.write(val);
fos.close();
}
}
```

5. RandomAccessFile: This class is used for accessing a particular file randomly and perform read /write operations

Methods:
seek() ---placing the file pointer at the required location
writeUTF() --to write to the file
length() ---returns the size of the file
getFilePointer() --this will return the pointer location
close()

Example1:

```
import java.io.*;
public class Eighty
{
	public static void main(String aa[])throws Exception
		{
			RandomAccessFile ra=new RandomAccessFile("zero.txt","rw");
			ra.seek(ra.length()-1);
			String ss="java class data";
			ra.writeUTF(ss);
			System.out.println("written");
			ra.close();


		}
}
```

Example2:

```
import java.io.*;
public class Nine
{
	public static void main(String aa[])throws Exception
		{
			RandomAccessFile ra=new RandomAccessFile("zero.txt","rw");

			ra.seek(Integer.parseInt(aa[0]));
			int i=ra.read();
			while(i!=-1)
			{
				System.out.print((char)i);
				i=ra.read();
			}
			long ii=ra.getFilePointer();
			System.out.println("location is"+ ii);
			ra.close();


		}
}
```