Ques 1

INHERITANCE :- Reusability is yet another aspect of oop paradigm. It is always nice if we could reuse something that already exists rather than creating the same all over again. Java supports this concept. Java classes can be reused in several ways. This is basically done by creating new classes, reusing the properties of existing ones.

The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or super class or parent class and the new one is called the subclass or derived class or child class.

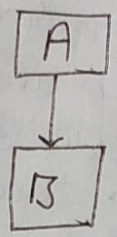
The inheritance allows subclasses to inherit all the variables and methods of their parent classes.

Inheritance may take different forms.

01 Single inheritance (only one superclass)
02 multiple inheritance (several super classes)
03 Hierarchical inheritance (one superclass, many subclasses)
04 multilevel inheritance (derived from a derived class)

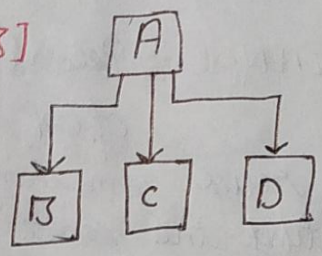
Java does not directly implement multiple inheritance however, this concept is implemented using a secondary inheritance path in the form of interfaces.

[A]



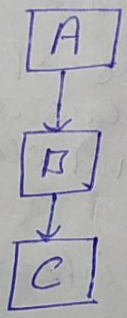
(a) Single inheritance

[B]

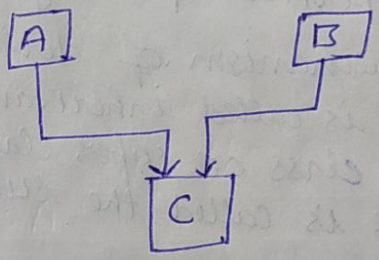


(b) Hierarchical inheritance

[C]



(c) multilevel inheritance



(d) multiple inheritance

01→ Multilevel Inheritance→

A common requirement in object-oriented programming is the use of a derived class as a super class. Java supports this concept and uses it extensively in building its class library. This concept allows us to build a chain of classes as shown in fig→

The class A serves as a base class for the derived class B which is turn serves as a base class for the derived class C. The chain ABC is known as inheritance path.

A derived class with ~~multiple~~ multilevel base classes is declared as follows.

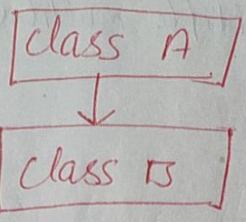
```
class A
{
  - - -
  - -
}
class B extends A // first level
{
  - - - -
  - - - -
}
class C extends B // second level
{
  - - -
  - - -
}
```

This process may be extended to any number of levels.
The class C can inherit the members of both A and B

~~02~~

02:- Single Inheritance :- Single inheritance is the simple inheritance of all.

when a class extends another class (only one class) then we call it as single inheritance
The below diagram represent the single inheritance in Java where class B extends only one class Class A. Here class B will be the sub class and class A will be one and only super class.

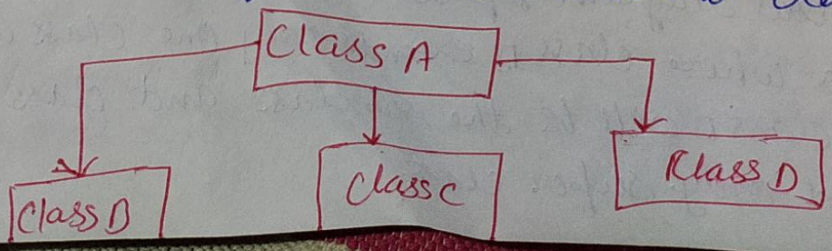


Syntax

```
Class A
  {
    - - - - -
      - - - - -
  }
Class B extends Class A
  {
    - - - - - -
      - - - - -
  }
```

03) Hierarchical Inheritance :

In Hierarchical Inheritance one parent class will be inherited by many sub classes. As per the below example. Class A will be inherited by Class B, Class C, and Class D. Class A will be acting as a parent class for Class B, Class C and Class D.

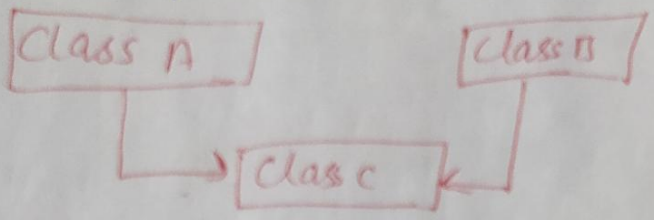


Syntax :-

```
Class A
{
  ------
  ------
}
Class B extends class A
{
  ------
  ------
}
Class C extends Class A
{
  ------
  ------
}
class D extends class A
{
  ------
  ------
}
```

04- Multiple Inheritance :- multiple inheritance not like but one class extending more than one class. multiple inheritance is basically not supported by many Object Oriented programming languages such as Java, Smalltalk, C#. But we

Achieve multiple inheritance in Java using interface



02:- POLYMORPHISM :- Polymorphism is the ability to take more than one form. Polymorphism is one of the most important concept in oops (object oriented programming concepts) Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class.

In Java, there are two ways by which we can achieve polymorphism behavior.

01- method overloading (compile time Polymorphism)
02- method overriding (Runtime Polymorphism)

Method overloading :- method overloading implies we have more than one method with the same name within the same class but the condition here is that the parameter which is passed should be different.

Example of overloading :-

```
class overloading
{
  public void disp()
  {
    System.out.println("Inside first disp method");
  }
```

```
Public void disp (String val)
{
System.out.Println ("Inside second disp method, value is: "+val);
}
Public void disp (String val1, ~~val2~~ String val2)
{
System.out.Println ("Inside third disp method, values
are: ~~"val1"~~, "+val1+"~~,~~"+val2);
}
}
Public class methodoverloading-Example
{
Public ~~void~~ static void main (String arg[])
{
overloading obj = new overloading ();
obj.disp (); // Calls the first disp method
obj.disp (" Java"); // Call the second disp method
obj.disp ("Java", "OOC"); // Call the third disp method
}
}
```

The output will be

Inside first disp method
Inside second disp method, value is :- Java
~~Inside~~ third disp method, values are :- Java, OOC

Method overriding :-

In any object-oriented programming language, overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type (or sub type) as a method in its super-class, then the method in the sub class is said to override the method in the super class.

Example of method overriding :-

```
Class vehicle
 {
 void run()
 {System.out.Println("vehicle is running");
 }
}
Class BIKE2 extends vehicle
 {
 void run()
 {System.out.Println("BIKE is running safely");
 }
PUBLIC static void main(String args[])
```

REDMI NOTE 8
DEEPANSHU NEGI

```
Bike2 obj = new Bike2();
obj.run();
  }
}
```

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

Encapsulation :-

Encapsulation is the concept of hiding the implementation details of a class and allowing access to the class through a public interface. For this, we need to declare the instance variables of the class as private or protected.

The wrapping up of data and methods into a single unit is known as encapsulation. Data Encapsulation is the most striking features of a class. The data is not accessible to the outside world and only those methods which are wrapped in the class can access it.

Encapsulation makes it possible for objects to be treated like black boxes, each performing a specific task without any concern for internal implementations.

Java Features :-

Various features of Java are explained as follows :-

01 Familiar, Simple and Small :- Java is a simple and small language. Programs are easy to write and debug because Java does not use pointers eg., Java does not use preprocessor header files, go to statements or others. It also eliminates operator overloading and multiple inheritance.

02. Interpreted :- Java is an interpreted language such as Java programs run directly from the source code. The interpreter program reads the source code and translates it on the fly into computation. Java as an interpreted language depends on an interpreter.

03 object oriented :- Java is a true object oriented language. Almost everything in Java is an object. All program code and data reside with in object and classes. Java comes with an extensive set of classes, arranged in packages that we can use in our programs by inheritance.

04 Dynamic and Extensible:- While executing the Java program the user can get the required files dynamically from a local computer or drive thousands of miles away from the user just by connecting with the internet.

05 High performance:- Java performance is impressive for an interpreted language mainly due to the use of intermediate byte code. Java architecture is also designed to reduce overheads during runtime.

06- Platform Independent and Portable:- The Java is more portable language. Java program can be easily moved from one computer system to another anywhere and any time. Changes and upgrades in operating systems, Processors and system resources will not force any changes in Java program. Java ensures portability in two ways. First Java compiler generates bytecodes instructions that can be implemented on any machine. Secondly the size of the primitive data types are machine independent.

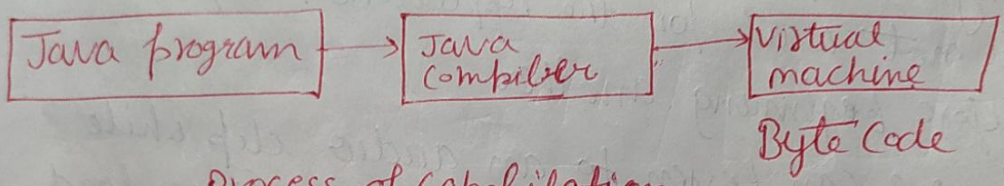
07 Multithreaded :- Java is also a multithreaded programming language. Multi threading means a single program having different thread executing independently at the same time. Multiple threads execute instructions according to the program code in a process or a program. Multithreading works the similar way as multiple processes run on one computer. Multithreaded means handling multiple tasks simultaneously. This means that we need not wait for the application to finish one task before beginning another.

Example :- we can listen to an audio clip while scrolling a page and at the same time download an applet from a distance computer.

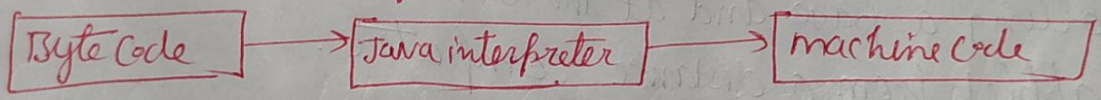
08 Distributed :- Java is designed as a distributed language for creating application on networks. It has the ability to share both the data and programs. Java application can open and access remote objects on internet as easily and they can do in local system.

Java virtual machine :-

JVM stands for Java virtual machine. This virtual machines runs a special set of "instructions" called bytecodes. JVM accepts a form of computer intermediate language commonly referred to as Java byte codes. Java virtual machines operates on Java byte codes which is normally generated from Java source code.



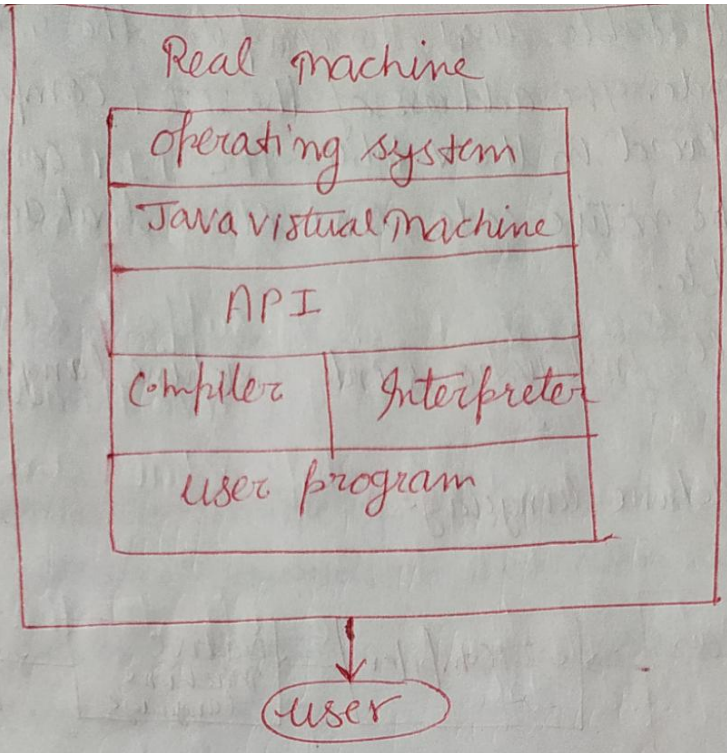
Process of compilation



Virtual machine

Process of converting byte code into machine code.

The virtual machine is not machine specific. The machine specific code is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine



Layer of Java program.

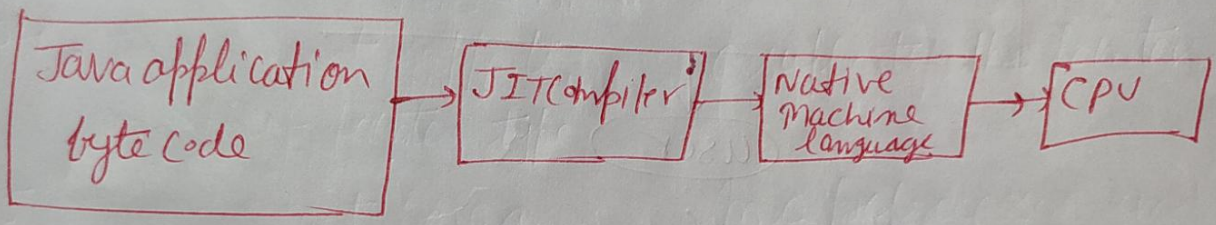
Just in Time compilers :-

The Job of the JIT compiler is to convert this byte-code to the machine code or native code that is run by the machine. A Just in time compiler is a program that turns Java bytecode (a program that contains instructions must be interpreted) into instruction that can be directly sent to the processor. After writing a Java program, the source language statements are compiled by the Java compiler into bytecode rather than into code that contains instructions that match a particular hardwares processors. The JIT compiler uses a pointer to the methods in the class.

This internal table used to compile the methods to native code. The address of the JIT compiler itself is placed in the V table. The JIT compiler executes the native code and stores that address to the V table.

01 converts all instructions into machine language as needed.

(II). Run time machine language



Comparision between C++ and Java

S.No	Java	C++
01	"Java" extension is based on compiler / interpreter	".cpp" extension is based on compiler.
02.	It supports 40 keywords	It supports 60 keywords.
03	Java does not supports operator overloading	C++ supports operator overloading or function overloading.
04	Java does not use pointers	C++ uses ~~pointer~~ pointers.
05	There are no header file in Java	Header files are used in C++ e.g #include <iostream.h>

06 Java does not support multiple inheritance of classes. This is ~~accomp~~ accomplished using a new feature called Interface	C++ supports all types of inheritance like multiple, single or others.
07 classes declaration ends without semicolon ;	classes declaration in C++ with semicolon.
08 overwriting is possible by using "super" keywords.	In C++ overwriting is possible by using scope resolution (::)
09 Java does not supports global variables every variable and method is a part of the classes.	C++ supports global variables.
10 supports labels with loops and statement blocks.	supports the goto statement.
11 Automatic garbage collection does not have the concept of destructior.	Explicit memory management through third party frame works exist to provide garbag collection supports destructors

Difference between C and Java

S.NO	Java	C
01	Its extension is .java	Its extension .c.
02	It supports 32 keyword.	It supports 60 keywords.
03	Java does not contain the data types struct and union.	In C uses the all data types struct and union.
04	Java does not supports an explicit pointer type	C supports the pointers.
05	Java does not have a preprocessor and therefore we cannot use #define, #include and ~~#self~~ #ifdef statements	In C preprocessor and #define, #include can be used.
06	It supports to "import" keyword is used to call package.	It supports to Header files/libraries.
07	Java does not include the C, unique statements keywords size of, typedef	C used these statements sizeof, typedef.

Java Application Types :-

There are three types of Java Applications

01 Java console Applications :-

Java console applications can only display textual data. console applications resemble Dos based applications in that all the interaction with the program through keyboard and text output. The mouse and any use of multiple windows is not supported. Background applications are almost always written as console applications. A background application is a program that runs often for a long period of time, without user interaction.

When to use a console application :-

(i) Limited user interaction

(ii) Applications that run in the background.

(iii) Quick test applications to try out techniques in Java.

Limitations of a console applications :-

- No Mouse supports.
- No graphical / window support.
- No additional windows.

Java Application Types :

There are three types of Java Applications

01 Java Console Applications :

Java console applications can only display textual data. console applications resemble Dos based applications in that all the interaction with the program through keyboard and text output. The mouse and any use of multiple windows is not supported. Background applications are almost always written as console applications. A background application is a program that runs often for a long period of time, without user interaction.

When to use a console application :

(i) Limited user interaction

(ii) Applications that run in the background.

(iii) Quick test applications to try out techniques in Java.

Limitations of a console applications :

- No Mouse supports.
- No graphical / window support.
- No additional windows.

02. Graphical user interface :- Most applications used with windows are regular Graphical user interface or GUI applications. If we are going to create an application that the user works directly with, we will most likely want to create a GUI application.

When to use a GUI application :-

- Working directly with the user.
- Application that must display graphical information.

Limitations of a GUI applications.

- More complex to setup than console application.
- Less convenient to run in the background.

03 Java Applet :- Java applet allow you to imbed a program directly into a browser. The user simply has to visit your website to view the applet. Applets can be used to display graphics animation and produce sound music. Applet can be very useful for displaying advertisements on websites or providing a greater deal of interactively than an HTML page alone can provide.

When

01 when

02 when

03 Appli
quick

Limi

01 Appl

02 Ap
tec

When to use a Java Applets :-

01 When our applications should run directly with a website.

02 When your applications enhances the use of a website.

03 Application that display animation that should be quickly accessed form a website.

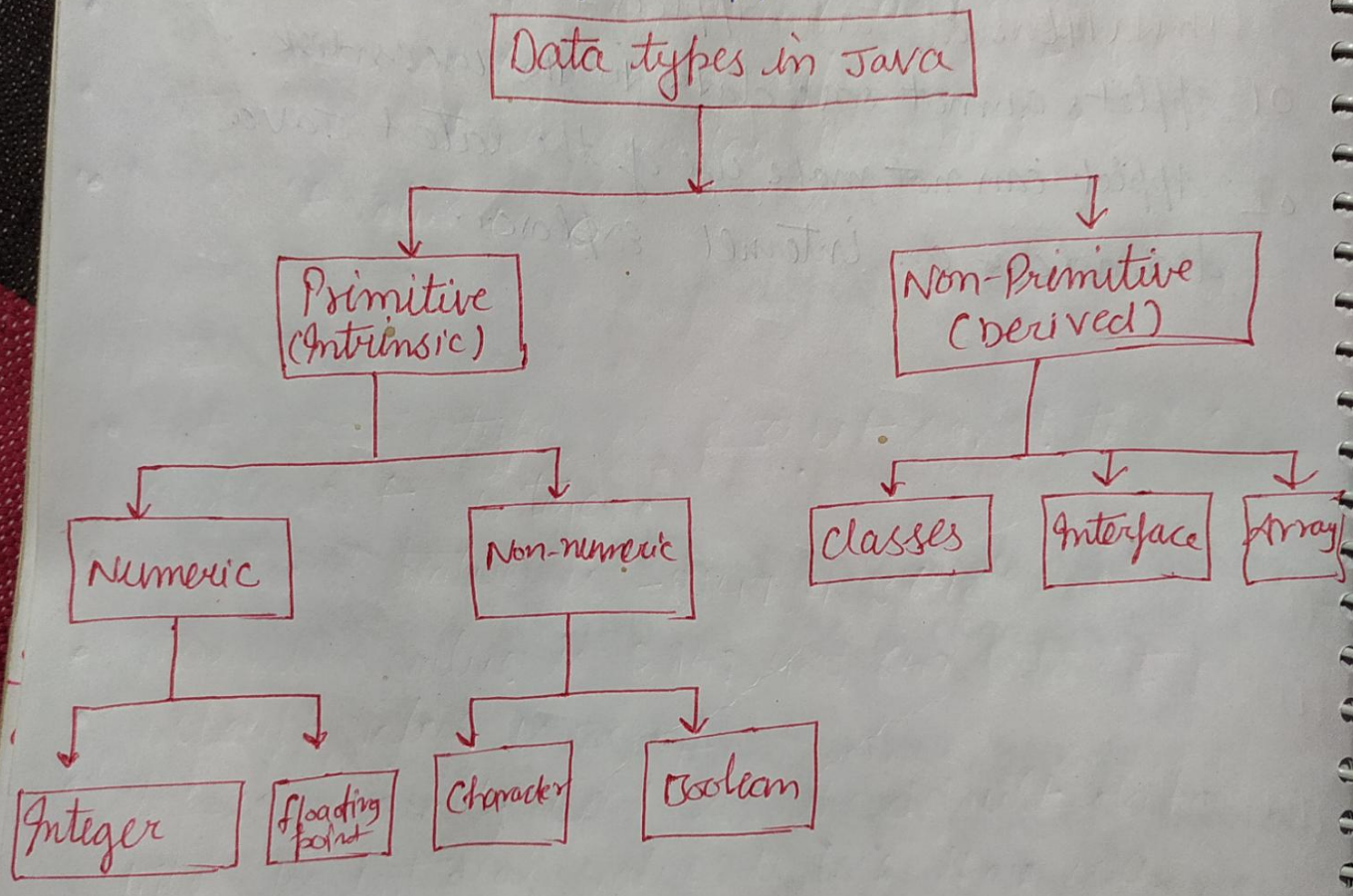
Limitations of Java Applets :-

01 Applets cannot save data to the user's disk.

02 Applet can not make use of the latest Java technology on internet explorer.

Data Types :-

Data types specify the size and type of values that can be stored. Java language is rich in its data types. The variety of data types available allows the programmer to select the type appropriate to the needs of the application.



Primitive data type :- Primitive data types are predefined by the language and named by a key word. Primitive data type is also known as Intrinsic data type.

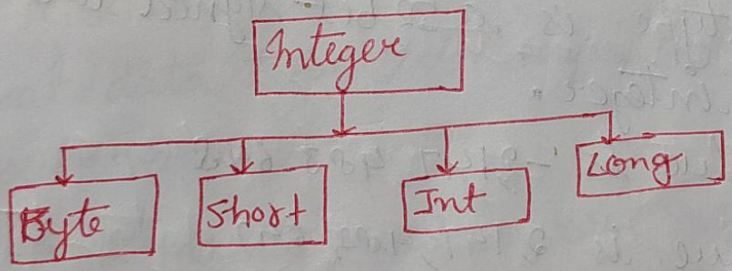
Primitive data types are two type

Numeric :- Numeric data types are also two types

(I) Integer type

(II) Floating point type

(I) Integer :- Integer data types are used to represent signed integer numbers. The size of the values that can be stored depend on the integer data type we choose. Java supports four types of integers. ~~As Amoon~~ They are byte, short, int, and long it does not support the concept of unsigned types.



(i) Byte :

- Byte data type 8 bit signed two's complement integer.
- Minimum value is -128
- Maximum value is 127
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.

(ii) Short :-

- Short data type is a 16 bit signed two's complement integer.
- Minimum ~~size~~ value is -32,768
- Maximum value is 32,767
- Short data type can also be used to save memory as byte data type. A short data type is 2 times smaller than an int.
- Default value is 0

(iii) INT :-

- Int data type is a 32 bit signed two's complement integer.
- Minimum value is -2,147,483,648
- Maximum value is 2,147,483,647
- Default value is zero
- Int is generally used as the default data type for integral values unless there is a concern about memory.

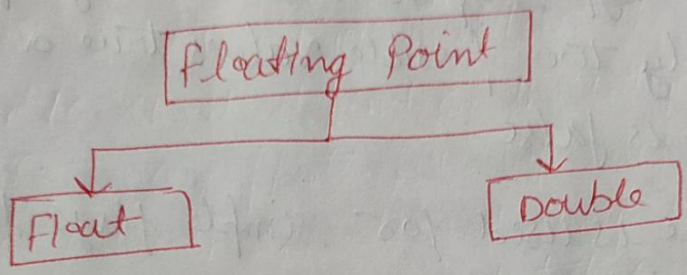
(iv) LONG :-

- Long data types is 64 bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808
- Maximum value is 9,223,372,036,854,775,807

• Default value is 0L.

(ii) Floating type :-

Integer types can hold only whole numbers and therefore we use another type known as floating point type to hold numbers containing fractional part such as 27.59 and -1.375.



(a) Float :-

(i) Float data type is a single precision 32 bit

(ii) Default value is 0.0F

(iii) Float is mainly used to save memory in large arrays of floating point numbers.

(b) Double :

(a) Double data type is a double precision 64 bit

(b) default value is 0.0d.

(c) This data type is generally used as the default data type for decimal values.

Boolean type :- Boolean data type is used when we want to test a particular condition during the execution of the program. There are only two values that a boolean type can take true or false.

- Boolean data type represents one bit of information
- There are only two possible values true or false.
- default value is false
- The data type is used for simple flags that tracks true/false conditions.

Example Boolean one = true.

Character data type → Java provides a character data type called char. The char type assumes a size of 2 bytes but basically it can hold only a single character.

- Char data type is a single 16 bit unicode character.

Minimum value is "\u0000" (or 0)

Maximum value is "\uffff" (or 65,535 inclusive)

Character data type is used to store any character.

Constructors in Java :-

Constructors are used to initialize the object's state. Like methods, a constructor also contains collection of statements (i.e instruction) that are executed at time object creation.

Types of constructor :-

There are two types of constructor in Java.

01 :- No-argument constructor :-

A constructor that has no parameter is known as default constructor. If we dont define a constructor in a class then compiler creates default constructor (with no arguments). for the class and if we write the constructor with arguments or no arguments then the compiler does not create a default constructor.

default constructor provides the default values of the object like, 0, null etc, depending on the type

Example :-

```
import Java.IO.*;
Class DEMO
{
  int num;
  String name;
  DEMO()
  { System.out.Println("constructor called");
```

```
class aa
{
  Public static void main (String [] args)
  {
    DEMO obj1 = new DEMO();
    System.out.Println (obj1.name);
    System.out.Println (obj1.num);
  }
}
```

output:

```
constructor called
null
0
```

(11) Parameterized constructor → A constructor that has parameter is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

Example:

```
import java.io.*;
class DEMO
{
String name;
int id;
DEMO (String name, int id)
  {
    this.name = name;
    this.Id = id;
  }
}
class Test
{
Public static void main (String [] args)
  {
   DEMO obj1 = new DEMO ("AKshay", 1);
System.out.Println (" name : "+ obj.name + "Id"+
                              obj1.id);
  }
}
```

output

name : Akshay Id 1

* Interface and packages

Interface :- An interface is a collection of abstract methods. A class implements an interface thereby inheriting the abstract methods of the interface. A interface is basically a kind of class. Like classes, interface contain methods and variables but with a major difference. The difference is that interfaces define only abstract methods and final fields. This means that interface do not specify any code to implement these methods and data fields contain only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

The syntax for defining an interface is very similar to that for defining a class. The general form of an interface defination is :-

```
interface Interfacename
{
  variables declaration;
  methods declaration;
}
```

Properties of interface :-

Interface have the following properties :-

(i) An interface is implicitly abstract. we do not need to use the abstract keyword when declaring an interface.

(ii) Each method in an interface is also implicitly abstract, so the abstract keyword is not needed. method in an interface are implicitly public.

```
interface animal
{
  public void eat();
  public void travel();
}
```

Advantages of interface :-

① Interface are mainly used to provide polymorphic behaviour.

② Interface function to breakup the complex designs and clear the dependencies between object.

Synchronization :- one thread may try to read a record from a file while another is still writing to the same file. Depending on the situation, we may get strange results. Java enables us to overcome this problem using a technique known as synchronization. The keyword synchronized helps to solve such problems by keeping a watch on such locations. example the method that will read information from a file and the method that will update the same file may be declared as ~~synchon~~ synchronized. Example

```
Synchronized void update()
{
    ..... // code here is synchronized
    .....
    .....
}
```

When we declare a method synchronized Java creates a monitor and hands it over to the thread that calls the method first time. As long as the threads holds the monitor no other thread can enter the synchronized section of code. A monitor is like a key and the thread that holds the key can only open the lock. It is also possible to make a block of code as synchronized as shown below.

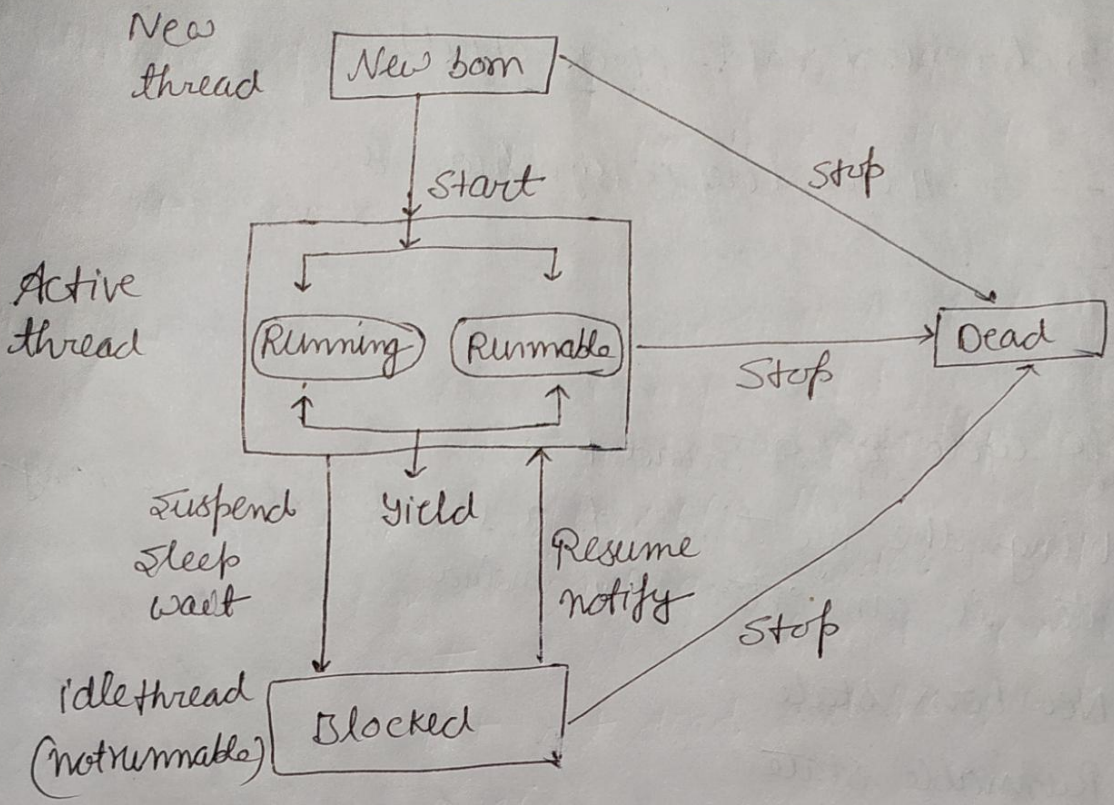
```
Synchronized ~~void~~ (lock-object)
{
  ------ // Code here is synchronized
  ---
}
```

Life cycle of A Thread :-

During the life time of a thread, there are many states it can enter. They include

(I) New born state
(II) Runnable state
(III) Running state
(IV) Blocked state
(V) Dead state

A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in figure.



State transaition diagram of a thread

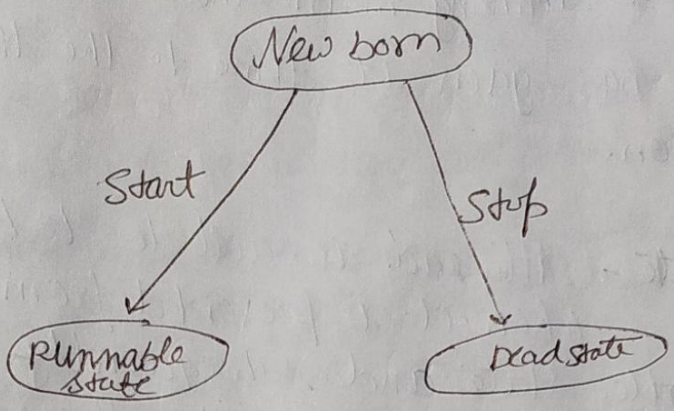
01. New born state :- when we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state we can do only one of the following things with it.

(i) Schedule it for running using start() method.

(ii) Kill it using stop() method.

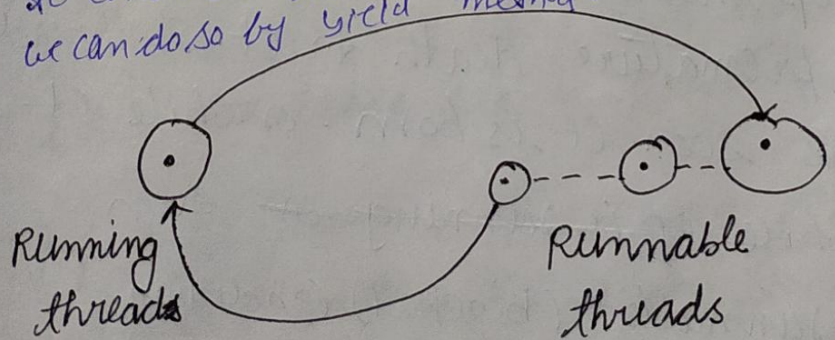
If scheduled, it moves to the runnable state. If attempt to use any other method at this stage

an exception will be thrown.



Scheduling a new born thread.

02 Runnable state :- The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion i.e first come first serve manner. If we want a thread to relinquish control to another thread to equal priority before its turn comes we can do so by yield method.



Relinquishing control using yield() method

03- Running state → Running means that the processor has given its time to the thread for its execution.

04 → Blocked - state → A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

05 → Dead state → Every thread has a life cycle. A running thread ends its life when it has completed executing its run() method. It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. A thread can be killed as soon it is born. or while it is running, or ~~while it is running, or~~ even when it is "not runnable" (blocked) condition.

The thread Control methods →

There are methods that can move thread one state to another. These are yield(), sleep(), and stop() method. which are define as follow →

yielding → when a threads has nothing to do it call the yield() method of its thread object. This method tells the scheduler to run a different thread. The value of calling yield() depends largely on whether the scheduling mechanism for the platform or which the program is running is pre-emptive or non-preemptive.

Sleeping → The thread class contains a static method named sleep() that causes the currently running thread to pause its execution and transit to the sleeping state. The method does not relinquish any lock that the thread might have. The thread will sleep for at least the time specified in its argument, before entering the runnable state where it's taken its turn to run again.

Stop → we can use this to kill the thread instantly without waiting for run() to exit.

Thread → A thread in the context of Java, is the path followed when executing a program. All Java programs have at least one thread, known as the main thread, which is created by the Java Virtual machine (JVM) at the program's start, when the main() method is invoked with the main thread. In Java, creating a thread is accomplished by implementing an interface and extending a class. Every Java thread is created and controlled by the java.lang.Thread class.

Multithread → Multithreading is a powerful programming tool that makes Java distinctly & different from its fellow programming languages. Multithreading is useful in a number of ways. It enables programmers to do multiple things at one time. They can divide a long program (containing operations that are conceptually concurrent) into threads and execute them in paralle

Creating threads:- Creating threads in Java is simple. Threads are implemented in the form of objects that contain a method called run(). The run() method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the threads behaviour can be implemented. A typical run() would appear as follows.

```
Public void run()
{
  ---
  ---
  ---
}
```

(statement for implementing thread)

The run() method should be invoked by an object of the concerned thread. This can be achieved by creating ~~thread~~ the thread and initiating it with the help of another thread method called start(). A new thread can be created in two ways.

① By creating a thread class:- Define a class that extends thread class and override its run() method with the code required by the thread.

(11)

By Converting a class to a thread→ Define a class that implements runnable interface. The runnable interface has only one method run() that is to be defined in the method ~~which~~ which the code to be executed by the thread.

Implementing the Runnable interface→

The Runnable interface signature.

```
Public interface Runnable
{
 void Run ();
}
```

One way to create a thread in Java is to implement the runnable interface and then instantiate an object of the class. We need to override the run() method into our class which is only method that ~~needed~~ needs to be implemented. The run() method contains the logic of the threads.

The procedure for creating threads based on the runnable interface is as follows:

(i) A class implements the runnable interface providing the run() method that will be executed by the thread. An object of this class is runnable object.

(ii) An objects of

Exceptions:- An exceptions is a condition that is caused by a run time error in the program. When the Java ~~program~~ interpreter encounters an error such as dividing an integer by zero. It creates an exception object and throws it. (i.e, Informs us that an error has occurred.

Common Java exception are:-

01- Arithmetic exception: Caused by mat error such as divisi by zero.

02- Array Index out of bounds exception:- Caused by bad array indexes.

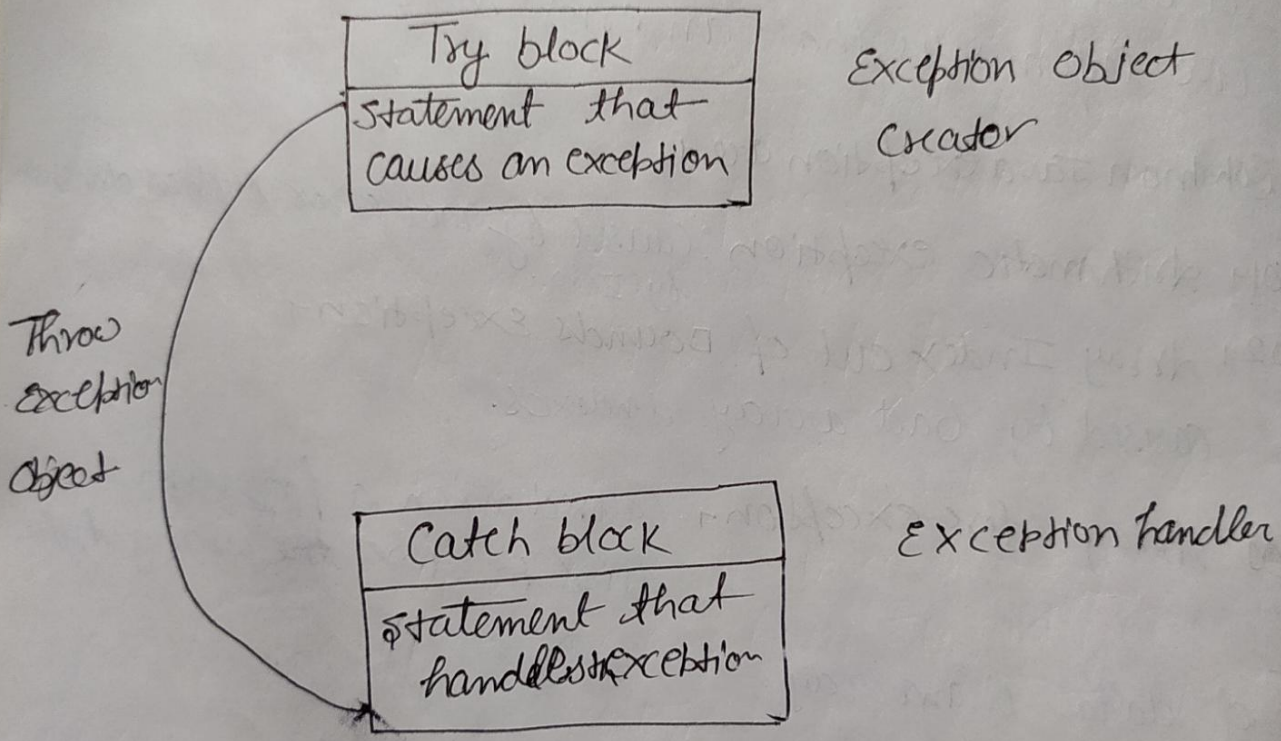
03 Array Store exception:- Caused when a program tries to store the wrong typ of data in an array.

04- File not found exception:- Caused by an attempt to access a non existent file.

05- IO Exception:- Caused by general I/O failures such as inability to read from a file.

Syntax for exception handling code:

The basic concept of exception handling are throwing an exception and catching it. This is show in the figure.



Java uses a keyword try to preface a block of code that is likely to cause of error condition and "throw" an exception. A catch block defined by the keyword catch "e catches" the exception thrown by the try block. and ~~handel~~ handles it appropriately. The catch block immediately after try block.

Example-

```
try
{
 Statement; // generates an exception
}
Catch (Exception type e)
{
 Statements; // Processes the exception
}
```

The try block can have more ~~than~~ or one statement that could generate an exception. If any one statement generates an ~~exc~~ exception the remaining statement in the block are skipped and execution Jumps to the catch block that is placed next to the try block.

The catch block ~~to~~ can have one or more statements that are necessary to process the exception.
remember every try statement should be followed by atleast one catch statement. otherwise compiler error will occur.

Methods for exception handling

(1) Try()

(2) catch()

finally → The finally block always executes when the try block exits. When an exception is raised, the statement in the try block is ignored. Sometimes it is necessary to process certain statements irrespective of whether an exception is raised or not the finally block is used for this purpose.

Throw → The throw class is used to call exceptions explicitly. we want to throw an exception when the user enters a wrong login ID and password we can use the throw statement to do so. The throw statement takes an single argument. which is an object of exception class.
If the object does not belong to a valid exception class. the compiler given error.

Throws → The throws statement ~~Specify~~ Specify the list of exception that has thrown by a method. If a method is capable of raising an

exception that it ~~doesn~~ does not handle. It must specify the exception has to be handle by the calling method this is done by using the throw statement.

Command Line arguments :

There may be occasions when we may like our program to act in a particular ways depending one the input provided at the time of execution. This is achieved in Java programs by using what are known as command line arguments.

Command line arguments are parameters that are supplied to the application programs at the time of invoking it for execution

We can write Java programs that can receive and use the arguments provided in the command line. Recall the signature of main method.

Public static void main (String args[])

args declared as an array of strings (known as String object). Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements.

Example :

```
Class Comeline Test
{
  Public Static Void main (String args[])
  {
    int count, i=0;
    String string;
    count = args.length;
```

```
System.out.Println("Number of arguments =" + Count);
  While (i<Count)
  {
    String = args[i];
     i = i+1;
  System.out.Println ( i + ":" + "Java is" + String + "!");
    }
  }
}
class Command
{
Public static void main (String args[])
{
int i, j, sum;
i = Integer.parseInt (ar[0]);
J = Integer.ParseInt (ar[1]);
sum = i+J;
System.out.Println ("sum is" +sum);
  }
 }
```

CASTING:-

In Java we can force one data type to another by an operation called casting.

Converting from one type to another is necessary because sometimes a function F accept type A, and our expression e has type B and we want to do "F(e)" so we need to convert e to type A.

In Java there are primitives types like short, int, double, Java data types are of two kinds: primitive and reference. The reference types are class interface and array.

The general from of a cast is:

(typename) expression

where type-name is one of the standard data types. The expression may be a constant, variable or an expression. some example of casts and their actions:-

Example	casts
(int)7.5	7.5 is converted to integer
(int)21/(int)4.5	evaluated as 21/4 and the result would be 5.

casting can be used to round off a given value to an integer.