Q. What is the OOPs?
Ans: How to stablish a link or interaction betⁿ two or more different objects.
Or
Intraction betⁿ new object and existing object.

OOPs: Any purely object oriented language should follow all the 6 features of object oriented Programming System (OOPs).They are

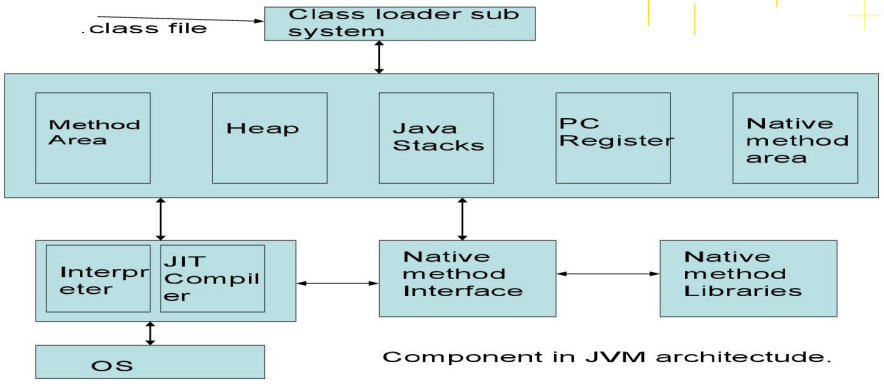
1. Abstraction.
2. Class & Object.
3. Encapsulation.
4. Inheritance.
5. Polymorphism.
6. Message Passing.

Q. What are the advantages of Object Oriented Programming Languages (OOPL)?
A : The Object Oriented Programming Languages directly represent the real life objects like *Car*, *Jeep*, *Account*, *Customer* etc. The features of the OO programming languages like ***polymorphism, inheritance*** *and* ***encapsulation*** make it powerful.

Q. What is byte-code?
Ans: The output of compiler java source code is called byte-code. A byte-code is a program that contains instruction that must be interpreted. A java interpreter interprets the bytecode to give the desired output.

The Java Development Kit (JDK): This bundle includes three tools — a Java compiler, a Java virtual machine, and the Application Programming Interface. With the JDK, you can create and run your own Java programs. Another name for the JDK is the *Java SDK* — the *Java Software Development Kit.* Some people still use the SDK acronym, even though the folks at Sun Microsystems don't use it anymore. (Actually, the original name was the JDK. Later Sun changed it to the SDK. A few years after that, Sun changed back to the name JDK. As an author, this constant naming and renaming drives me crazy.)

Q JVM: First, The .java program converted into a .class file consisting of byte code instructions by the java compiler. Remember, this java compiler is outside the JVM. Now this .class file is given to the JVM. In JVM
i) First of all, it loads the .class file into memory.
ii) Then it verifies whether all byte code instruction are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.
iii) If the byte instruction are proper, then it allocation necessary memory to execute the program. This memory divided into 5 parts.
A) Method area: Store the class code, code of the variable & code of the method in the java program.
B) Heap: This is area where objects are created.
C) Java Stacks: It needs some more memory to store the data and results.
D) PC (Program Counter) register: Its contain memory address of the instructions of the method. If there are 3 methods, 3 PC registers will be used to track the instruction of the method.
E) Native method stacks: To execute the native methods, generally native method libraries are required. These header files are located and connected to JVM by a program.



Component in JVM architectude.

In java interpreter & JIT (Just in Time) Compiler both are needed. Most of the JVM implementations use both the interpreter & JIT (Just in Time) Compiler simultaneously to convert the byte code .This technique is also called "adaptive optimize".
JVM is a program ,which is written in C Language.
Q. Garbage collector: JVM is also capable of deallocating the memory when it is not used by garbage collector. It is automatically invoked when the program is being run. It is call by gc() method of Runtime class or System class in java.
Q. what is object?
Ans: A real world instance which have some attributes & behaviors.
Q. What is java?
Ans: java is cross-platform, object-oriented, network-based, reliable programming language.

Q. History of java.
Ans: Gosling thought C and C++ could be used to develop the project. But the problem he faced with them is that was system dependent languages and hence could not be used on various processors.
Java was developed by a team led by ***James Gosling*** at sun Microsystems.
Java was originally called Oak in 1991.It was redesigned for developing internet applications and renamed java in 1995.

- Java has divided into 3 parts i) Java SE: Java Standard Edition ii) Java EE : Java Enterprise Edition iiii) Java ME : Java Micro Edition.

Q. Characteristics of java. (Java is Strongly Language)
Ans:

1. java is Simple.
2. java is Object Oriented
3. java is Distributed.
4. java is Portable.
5. java is Secure.
6. java is High Performance.
7. java is Multithreaded.
8. java is Dynamic,
9. java is robust(reliable).
10. java is Interpreted.
11. java is Architecture Neutral.

Q. What is the difference between C++ and Java?

1	C++ is not a purely object-oriented programming language,Since it is possible to write C++ programming without using a class or an object.	Java is purely an object-oriented programming language,since it is not possible to write a java program without using atleast one class.
	Pointers are available in c++	We cannot create and use Pointers in java.
2	Multiple Inheritance feature is available in c++.	No Multiple Inheritance in java but there are means to achieve it.
3	Operator overloading is available in c++.	It is not available in java.
4	There are 3 access specifiers in c++. : private, public, protected.	There are 4 access specifiers in c++. : private, public, protected and default.
5	Destructors in c++.	Destructors not support in java.
6	C++ has goto statement.	Java does not have goto statement.
7	#define,typedef ,header file and unions are available in c++.	#define,typedef ,header file and unions are not available in java.
8	Automatic casting is available in c++.	In some cases,implicit casting is available.But it is advisable that the programmer should use casting wherever required.
9	Allocation & deallocation of memory is the responsibility of the programmer.	Allocation & deallocation of memory is the responsibility of the programmer.
10	C++ is system dependent languages.	Java is system independent languages.

Q. What is the difference between C++ and Java?
Ans: Both C++ and Java use similar syntax and are Object Oriented, but:
i) Java does not support pointers. Pointers are inherently tricky to use and troublesome.
ii) Java does not support multiple inheritances because it causes more problems than it solves. Instead Java supports **multiple interface inheritance**, which allows an object to inherit many method signatures from different interfaces with the condition that the inheriting object must implement those inherited methods. The multiple interface inheritance also allows an object to behave **polymorphically** on those methods.

iii) Java does not support destructors but rather adds a finalize() method. Finalize methods are invoked by the garbage collector prior to reclaiming the memory occupied by the object, which has the finalize() method. This means you do not know when the objects are going to be finalized. **Avoid using finalize() method to release non-memory resources** like file handles, sockets, database connections etc because Java has only a finite number of these resources and you do not know when the garbage collection is going to kick in to release these resources through the finalize() method.

iv) Java does not include structures or unions because the traditional data structures are implemented as an object oriented framework

v) All the code in Java program is encapsulated within classes therefore Java does not have global variables or functions.

vi) C++ requires explicit memory management, while Java includes automatic garbage collection.

Java is not a purely OOPL. Points are		Java is a purely OOPL. Points are
1	Purely object oriented means it should contain only classes and objects.It should not contain primitive datatypes like int,char,float etc ,since they are neither classes not objects.	Even if java has primitive datatypes, these types are used inside a class and never outside of it. So, they are part of a class. See the API specification on the class: 'Classs'.java specification says that all the arrays and the primitive java types and the keyword void are also represented as objects of the class 'Class'.
2	In pure object oriented languages, we should access every thing by message passing (through object).But, java contains static variables and methods which can be accessed directly without using object.	Even static variables and static method are written inside a class. When accessing them from outside, we should use classname.
3	Java does not contain multiple inheritance. It mean an important feature of object oriented design is lacking. so how can we say it is purely object oriented?	Any purely object oriented language should follow all the 5 features of OOPs 1. class & object 2.Encapsulation 3.Abstraction 4.Inheritance 5.polymorphism. Java does not contain multiple inheritance, we should not say it is not purely OOL. Multiple inheritance is not the main feature of OOPs.

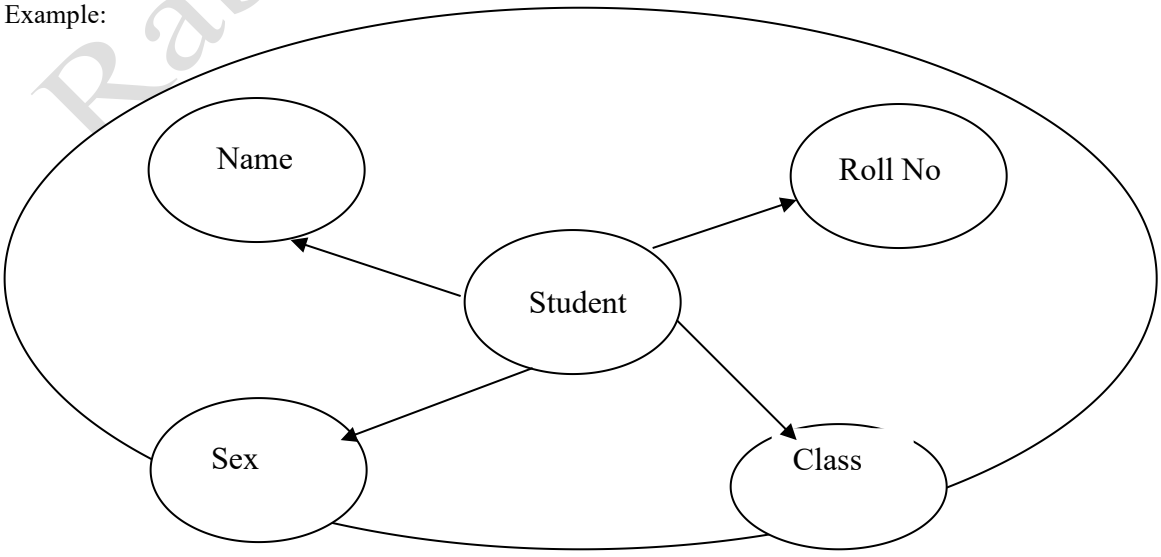
Q. Difference between object Oriented programming & **Procedural/structural programming?**

00P		PSP	
1	Oops follow bottom to up approach.	1	PSP follow top to bottom approach.
2	Communication is done through the object.	2	Communication is done through the procedural.
3	Abstraction is done through the object.	3	Abstraction is done through the procedural.
4.	Program is divide in parts is called object.	4.	Program is divide in parts is called function.
5.	Oops very easy to understand by the program.	5	It has some complex in the procedural.
6	Ex:- Java,VB.Net etc	7	Ex:- Pascal, FORTAN etc.

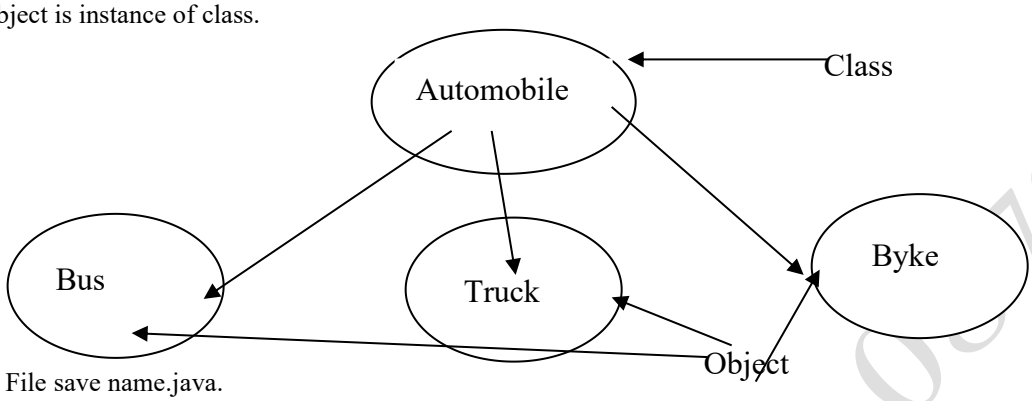
Q. Abstraction: It shows only those parts of information, which is very necessary for users. Or it hides the unnecessary information.

Q. Encapsulation: Wrapping of Properties & methods into a single unit is called Encapsulation. In C++ or java wrap is a Class.

Example:



Q. Class: It is collection of Properties & methods. Or
Class: It is defined as a space where all Properties & methods of an object are declared & defined.
Object: Object is instance of class.



Q. Step 1: File save name.java.
Step2:For compile => javac filename.java
Step3: For Run => java classname

Q.

```
Line 1      class abc
Line 2       {
Line 3                    public static void main(String args[])
Line 4                     {
Line 5                    System.out.print("Statement");
Line 6               }
Line 7    }
```

public : main() is accessible outside the class.

static : main() is accessible outside the class without any object creation.
Since we want to call main() method without any object. Then, how is the main() method called and executed ? The answer is by using the classname.method().

void : main() does return any value to the compiler.

main() : it is indicates a execution of the program.
String args[] : The code compilers but JVM cannot run it, as it cannot see the main() method with String args[].

For line 5: print the member of out class and out is static member of the system class.

Class PrintStream
{
Print()
{
----- 	
}
Println()
{
----- 	
------ 	
}
---------- 	
}
	Class System
	{
	Static PrintStream out;
	
	}

Q. Java is called case sensitive program because we have written the class System starting with a capital letter. You cannot write this class name as system or SYSTEM.

Q. What are the kinds of variables in java?
Ans: Three types 1. instance variable 2. local variable 3.class variables.

Q. java primitive (Simple) Data Type:

1. Integer. 2. Decimal. 3. Char. 4. Boolean.

Q Integer:

	Name:	Size	Range	**Default value**
1	Byte	1 bytes (8 bits)	$-(2)^7$ to $(2)^7$ -1=-128 to 127	0
2	Short	2 bytes (16 bits)	$-(2)^{15}$ to $(2)^{15}$ -1=-32,768 to 32,767	0
3	int	4 bytes (32 bits)	$-(2)^{31}$ to $(2)^{31}$ -1	0
4	Long	8 bytes (64 bits)	$-(2)^{63}$ to $(2)^{63}$ -1	0

Q Decimal :

	Name	Size	Range	**Default value**
1	Float	4 bytes (32 bits)	-3.4e38 to 3.4e38	0.0
2	Double	8 bytes (64 bits)	-1.8e308 to 1.8e308	0.0

Q. char :

	Name	Size	Range	**Default value**
1	Char	2 bytes (16 bits)	0 to 65535	An Unicode character

Q. Boolean: True or False. Default value is false.

Q. java program:

```
class Test
{
   public static void main(String a[])
   {
      int a;
      byte b;
      System.out.println("a:::"+a);
      System.out.println("b:::"+b);
   }
}
```

```
class Swap
{
    public static void main(String a[])
   {
      int a,b;
      a=a+b;      b=a-b;      a=a-b;
      System.out.println("a:::"+a);
      System.out.println("b:::"+b);
   }
}
```

```
Class TestChar
{   public static void main(String a[])
   {     char  ch='A';
     System.out.println("ch:;;"+ch);
     ch++;
     System.out.println("ch:::"+ch);
     ch=88;
     System.out.println("ch:::"+ch);
   }
}
```

Type Casting:

Q. Type Casting: one data type convert into an other Data Type is called Type Casting.

There are 2 types

1. Implicit.(Widening conversion) or (Up casting)
2. Explicit.(Narrowing conversion) or (down casting)

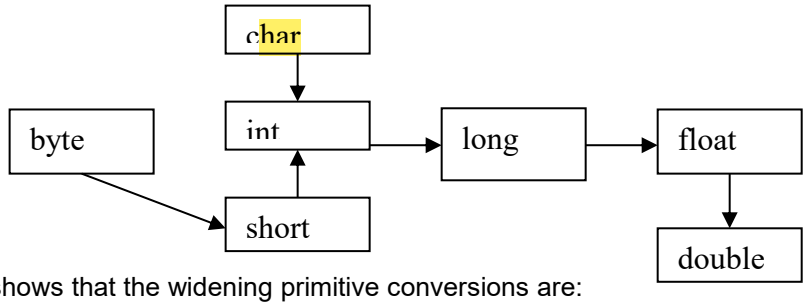
Implicit casting: Automatic casting done by the java compiler internally is called implicit casting. Implicit casting is done to convert a lower data type into a higher data type.

Explicit casting: The casting done by the programmer is called explicit casting. Explicit casting is compulsory while converting from a higher data type to a lower data type.

```
Class implicit
{
   Public static void main(String args[] )
    {
     int a;
     byte b=10;
     a=b;
     System.out.println(a);
     System.out.println(b);
    }
}
```

```
Class explicit
{
   Public static void main(String args[] )
    {
     int a=10;
     byte b;
     b=(byte) a;
     System.out.println(a);
     System.out.println(b);
    }
}
```

The following figure shows the path of valid conversions in java:



This code shows that the widening primitive conversions are:

- **byte** to **short**, **int**, **long**, **float**, or **double**
- **short** to **int**, **long**, **float** , or **double**
- **char** to **int**, **long**, **float** , or **double**
- **int** to **long**, **float**, or **double**
- **long** to **float** or **double**
  - **float** to **double**

Q. **Access Specifiers:**

i) private: 'private' members of a class are not accessible any where outside the class. They are accessible only within the class by the methods of that class.

ii) public: 'public' members of a class are accessible every where outside the class. So any other program can read them and use them.

iii) protected: 'protected' members of a class are accessible outside the class, but generally, within the same directory.

iv) default: if no access specifier is written by the programmer, then the java compiler uses a 'default' access specifier. It accessible outside the class but within the same directory.

```
Class ParentClass
{
        int a;
        private int b;
        protected int c;
        public int d;
        ParentClass()
        {
        a=10;
        b=20;
        c=30;
        d=40;
        }

        void DefaultMethod()
        {
        System.out.println("Default Method");
        }
        public void PublicMethod()
        {
        System.out.println("Public Method");
        }
        private void PrivateMethod()
        {
        System.out.println("Private Method");
        }
        protected void ProtectedMethod()
        {
        System.out.println("ProtectedMethod");
        }
}
```

```
class ChildClass  extends ParentClass
{
        void show()
        {
System.out.println("Method Access in Child Class");
        DefaultMethod();
        //PrivateMethod();
        PublicMethod();
                ProtectedMethod();
System.out.println("Value Access in child class");
        System.out.println("Default="+a);
        //System.out.println("private="+b);
        System.out.println("Public="+c);
        System.out.println("Protected="+d);
        }
}

class abc
{
    public static void main(String args[])
        {
        ChildClass obj=new ChildClass();
          obj.show();
    System.out.println("Value Access in Main
method");
System.out.println("Default="+obj.a);
//System.out.println("private="+obj.b);
System.out.println("Public="+obj.c);
System.out.println("Protected="+obj.d);
        }
}
```

Constructor

Q. Constructor: it is behave like a method where class name and method name must be same.

Q. Characteristics of Constructor:

1. **Whenever an object of class being created, constructor will call & execute automatically.**
2. **No need object to call constructor.**
3. **No return allowed with the name of constructor, not even void.**

Q. Uses of Constructor:

1. **Default value Setting.**
2. **One time execution program may be written inside the constructor.**

Q. Types of Constructor:

1. **Default Constructor.**
2. **Parameterized Constructor.**
3. **Copy Constructor.**

Q. Default Constructor: A Constructor have no parameter is called Default constructor.

```
class Test
{
   Test()
    {
     System.out.println("Default constructor");
     }
}

class a
{
 public static void main(String args[])
  {
    Test obj=new Test();
  }
}
```

Note: new=> **The "new" operator dynamically allocates memory for an object.**

Test()=>Constructor.

Note: Without construction we cannot create any object of any classes.

Q. Write a code add 2 +ve number.

```
class add
{
   int a,b,c,d;
    add()
     {
       a=10;        b=20;
      }
     void input (int p, int q)
      {
       c=p;        d=q;
      }
     void check()
     {
       if(p<0)          p=a;
       if(q<0)          q=b;
     }
    void sum()
    {
     System.out.println(p+q);
    }
}
class abc
{
 public static void main(String args[])
  {
    add obj=new add();      obj.input(100,200);
    obj.check();             obj.sum();
  }
}
```

Q. Parameterized Constructor: A constructor has one or more parameter is called parameterized constructor.

```
class add
{
        int a,b,c;
        add()
        {
                a=0;            b=0;
                System.out.println(a+b);
        }
        add(int x,int y)
        {
                a=x;            b=y;
                System.out.println(a+b);
        }
        add(int x,int y, int z)
        {
                a=x;    b=y;    c=z;
                System.out.println(a+b+c);
        }
}
 class test2
{
   public static void main(String args[])
        {
                add obj=new add();
                add obj1=new add(2,4);
                add obj2=new add(2,3,4);
        }
}
```

Q. Copy Constructor: It is same as parameterized Constructor but in parameter we will pass object of class.

```
class add
{
        int a,b,c;
        add()
        {
                a=0;
                b=0;
                System.out.println(a+b);
        }
        add(int x,int y)
        {
                a=x;
                b=y;
                System.out.println(a+b);
        }
        add(int x,int y, int z)
        {
                a=x;
                b=y;
                c=z;
                System.out.println(a+b+c);
        }
```

```
  add( add obj)
        {
                a=obj.a;
                b=obj.b;
                c=obj.c;
                System.out.println(a+b+c);
        }

}
 class test
{
   public static void main(String args[])
        {
                add obj=new add();
                add obj1=new add(2,4);
                add obj2=new add(2,3,4);
                add obj3=new add(obj1);
        }
}
```

Q When a constructor is declare as private then calling process is :

```
class A
{
  private A()
        {
        System.out.print("Private Constructor");
        }

        public static void main(String a[])
        {
                A ob=new A();
        }
}
```

Q.How can you call a constructor from another constructor? Or "this " Keyword use as method.

Ans: this() must be call the first line in constructor.

```
class B{
int x;
B()
{
this(1);
System.out.print("a[i]");
}
B(int x)
{this.x=x;
System.out.print(x);
}
}
class A{
public static void main(String args[])
{
B ob=new B();
}
}
```

```
class B{
int x;
B()
{
System.out.print("a[i]");
}
B(int a)
{
this();
x=a;
System.out.print(x);
}
}
class A{
public static void main(String args[])
{
B ob=new B(5);
}
}
```

this **is a keyword in Java. It can be used inside the Method or constructor of Class. It(this) works as a reference to the current Object whose Method or constructor is being invoked. The this keyword can be used to refer to any member of the current object from within an instance Method or a constructor.**

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

Example::

```
class Student10{
   int id;
   String name;
   Student10(int id,String name){
   id = id;
   name = name;
   }
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   Student10 s1 = new Student10(111,"Karan");
   Student10 s2 = new Student10(321,"Aryan");
   s1.display();
   s2.display();
   }
}
```

Output:

```
0 null
0 null
```

In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

Solution of the above problem by this keyword:::::

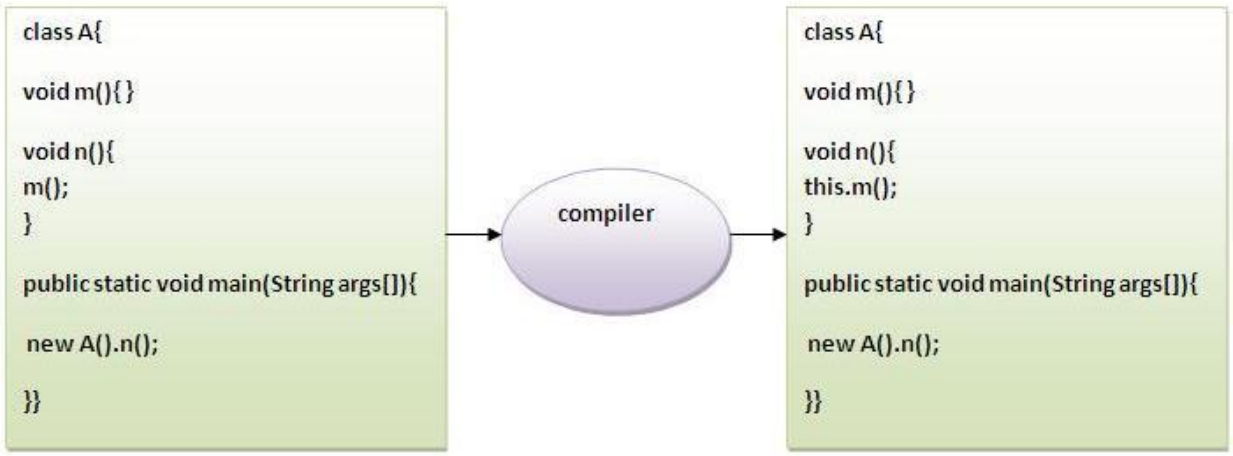
```
class Student11{
   int id;
   String name;

   Student11(int id,String name){
   this.id = id;
   this.name = name;
   }
   void display(){System.out.println(id+" "+name);}
   public static void main(String args[]){
   Student11 s1 = new Student11(111,"Karan");
   Student11 s2 = new Student11(222,"Aryan");
   s1.display();
   s2.display();
}
}
```

3)The this keyword can be used to invoke current class method (implicitly).

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the

example



4) this keyword can be passed as an argument in the method.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
class S2{
  void m(S2 obj){
 System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }

  public static void main(String args[]){
  S2 s1 = new S2();
  s1.p();
  }
}
```

Q. Defining and Testing a Simple Method: Write a method called triangleType. The triangleType method is sent 3 integers A, B, and C which are assumed to represent the sides of a triangle. The sides can be sent in any order, but the method must put the sides in ascending order in order to determine the type of triangle. The method returns a string indicating if the triangle is EQUILATERAL, ISOSCELES, SCALENE, or INVALID TRIANGLE.

Logic: If A, B, and C, represent the sides in ascending order, the rules for determining the triangle type are as follows: if A + B <= C, then the sides do not represent a valid triangle. if A = C (all the sides must be the same length) then the triangle is EQUILATERAL. if A = B or B = C, then the triangle is ISOSCELES; otherwise the triangle is SCALENE. After you have defined the method, you must: Write a main method to do the following: Repeatedly have the user enter 3 integers. Call the triangleType method and display the return type. Be sure you don't have an infinite loop (allow the user to quit).

```
import java.io.*;
class TriangleType
{
   int A,B,C;
   //Input Method
   int [] getInput()throws IOException {
      System.out.print("Enter three Inputs: \n");
      DataInputStream in = new
DataInputStream(System.in);

      int arr[] = new int[3];
      int i=0,j=0;
      for ( i = 0; i<3; i++) {
         System.out.print("Enter Value #" + (i + 1) +
":");
         arr[i] =Integer.parseInt(in.readLine());
      }
      return arr;
   }
```

```
//Bubble Short.
     int [] ascendingOrder(int arr[])throws
IOException {
      System.out.print("Numbers in Ascending
Order:" );
      int i=0,j=0;
for (i = 0; i < arr.length - 1; i++)
      {
         for ( j = 0; j < arr.length - i - 1; j++)
         {
            if (arr[j] > arr[j + 1])
            {
               int temp = arr[j];
               arr[j] = arr[j + 1];
               arr[j + 1] = temp;
            }
         }
      }
      for (i = 0; i < arr.length; i++)
```

```
  System.out.print("  " +arr[i]);
      System.out.println("\n");
      return arr;
      }

//Triangle validations.
   TriangleType()throws IOException
   {
     int [] input= getInput(); // For input call this
metiod.
      int[] input1 = ascendingOrder(input);
       A=input1[0];B=input1[1];C=input1[2];
       if((A + B) <= C)
          System.out.println("INVALID TRIANGLE");
       else if (A == C)
         System.out.println("Triangle is
EQUILATERAL");
       else if (A == B || B ==C)
         System.out.println("Triangle is ISOSCELES");
       else
         System.out.println("Triangle is SCALENE");
   }
}
```

```
 class Main {
   public static void main(String[] args)throws
IOException {
       char ch='n',buffer_value='a';
       do
       {
           TriangleType  tt = new TriangleType();
//Object creation
           System.out.println("Do you wish to continue?
answer y or n.\n");
          DataInputStream in = new
DataInputStream(System.in);
          ch=(char)in.read();
          buffer_value=(char)in.read();
         // Second Solution: in.skip(2);
       }while(ch=='y'|| ch=='Y');
   }
}
```

Destructor

Destructor is method for DeAllocation the memory.
Destructor Execution End of the Program.
NOTE: Destruction is not support by java.
Because in Java Garbage collector Execution End of the Program Automatically.

User input

Q. Write a program add two integer number using user input.

```
import  java.io.*;
class add
{
        public static void main(String a[])  throws IOException
        {
                DataInputStream  ds=new DatainputStream(System.in);
                int a,b;
                a=Integer.parseInt(ds.readLine());
                b=Integer.parseInt(ds.readLine());
                System.out.println((a+b));
        }
}
```

Q. Write a program add two Floating number using user input.

```
import  java.io.*;
class add
{
        public static void main(String a[]) throws IOException
        {
                DataInputStream  ds=new DatainputStream(System.in);
                float  a,b;
                a=Float.parseFloat(ds.readLine());
                b=Float.parseFloat(ds.readLine());
                System.out.println((a+b));
        }
}
```

Q. Write a program Find out sqrt root using user input.

```
class Root
{
        public static void main(String[] arguments)
        {
            int number = 225;
          System.out.println("The square root of " + number+ " is "+ Math.sqrt(number) );
            System.out.println("Square"+Math.pow(10,2));

        }
}
```

Q Why do we need import statement in java programs?

Ans: The classes and functions that are core to the Java language fundamental are packaged in java.lang. We don't need to import this as this is implicity import in all java programs. The classes which are not fundamental but important to java programming are package in various packages like java.io,java.util, java.sql. If we use methods of the classes inside these packages then we need to explicitly import that particular class by the import statement. The import statement tells the compiler that the current file is using the specified classes, or classes from the spacified package, and allows us to refer to those classes with abbreviated names. Java uses wile character(*) to import classes from a package. But this is not a good practice because during class loading time all these classes are unnecessarily referenced. This can have a native impact on the class loading time. So it is wise to import those classes explicitly in a program, which are required only.

Array

Arrays are created on Dynamic memory by JVM. There is no question of static memory in java; every thing (variable, array, object etc) is created on dynamic memory **only.**

One Dimensional Array

```
import java.io.*;
class Oned
{
        public static void main(String arg[]) throws
IOException
        {
                int[] a=new int[100];
                int n;
                BufferedReader br=new
BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the value of n");
                n=Integer.parseInt(br.readLine());
                for(int i=0;i<n;i++)
                {
          System.out.print("Enter the value:");
                    a[i]=Integer.parseInt(br.readLine());
                }
        System.out.println(" the values are:");
                for(int i=0;i<n;i++)
                {
                   System.out.println(a[i]);
                }
        }
}
```

```
import java.io.*;
class Twod
{
public static void main(String arg[]) throws
IOException
        {
                int[][] a=new int[100][100];
                int n,m;
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
System.out.println("Enter the value of n");
        n=Integer.parseInt(br.readLine());
System.out.println("Enter the value of m");
        m=Integer.parseInt(br.readLine());
                for(int i=0;i<n;i++)
                {
                  for(int j=0;j<m;j++)
                  {
            System.out.print("Enter the value:");
 a[i][j]=Integer.parseInt(br.readLine());
                  }
                }
        System.out.println(" the values are:");
                for(int i=0;i<n;i++)
                {
                  for(int j=0;j<m;j++)
                  {
              System.out.print(a[i][j]+"  ");
                  }
                System.out.print("\n");
                }
        }
}
```

Jagged Arrays:

```
 import java.io.*;
class Oned
{
        public static void main(String arg[]) throws IOException
        {
                int a[][]=new int[2][];
                a[0]=new int[2];
                a[1]=new int[3];
                a[0][0]=10;
                a[0][1]=20;

                a[1][0]=30;
                a[1][1]=40;
                a[1][2]=50;


                System.out.println(" the values are:");
                for(int i=0;i<2;i++)
                {
                  System.out.println(a[0][i]);
                }

                System.out.println(" the values are:");
                for(int i=0;i<3;i++)
                {
                  System.out.println(a[1][i]);
                }
        }
}
```

Command Line Argument

Q. Write a program add two integer number using command line argument.

```
    class command
{
        public static void main(String ar[])
        {
                int a,b;
                a=Integer.parseInt(ar[0]);
                b=Integer.parseInt(ar[1]);
                System.out.println((a+b));
        }
}
```

Q. Write a program add two Floating number using command line argument.

```
class command
{
        public static void main(String ar[])
        {
                float a,b;
                a=Float.parseFloat(ar[0]);
                b=Float.parseFloat(ar[1]);
                System.out.println((a+b));
        }
}
```

Q. Write a program add even number using command line argument

```
class command
{
        public static void main(String ar[])
        {
                int a=0,b;
                for(int x=0;x<ar.length;x++)
                {
                 b=Integer.parseInt(ar[x]);
                        if(b%2==0)
                        a=a+b;
                }
                System.out.println(a);
        }
}
```

Method in java

Q. What is a method?
Ans: A method is a function that is written in a class .This means whenever a function is written in java it should be written inside the class only. But in C++, we can write the function inside as well as outside the class, they are called member function & not methods.

Q What is instance variables?
Ans: An instance variable is a variable whose separate copy is available to each object & it is created in the objects on heap memory.

Q What is instance methods?
Ans: Instance methods are the methods which act on the instance variable of the class. To call the instance methods, we should use the form: objectname.methodname ().

Static

Q **Static Method**: No require any object, at the time of static method call. Static Method well calls with the Name of class. And static methods are the methods which do not act upon the instance variables of a class.

Q. What are restrictions for static method?

1. Static method can access static variable. Static variable are also declared as "static" keyword.
2. They can not refer to this or super keyword.

Q. **class variables**: class variable is static variable. Class variable is a variable whose single copy in memory is shared by all objects. And it is stored on method area.

Example:

Q. Why instance variable are not available to static methods?
Ans: After executing static methods, JVM creates the objects. So the instance variables of the objects are not available to static method.

NOTE: JVM execution a static block on a highest priority basis. This means JVM first goto to static block even before it looks for the main() method in the program.

```
class Test
{
        static
        {
        System.out.println("Static block");
        }
                public static void main(String ar[])
        {
                System.out.println("Main method");
        }
}
```

Output:
Static block
Main method

Q. Is it possible to compile and run a java program without writing main() method?
Ans: Yes, it is possible by using a static block in the java program.

```
class Test
{
        static
        {
System.out.println("Static block");
        System.exit(0);
        }
}
```

```
class Test
{
    int x=10;
    static void show ()
    {
            System.out.println ("Static Method");
        System.out.println("x:::"+x);
    }
}
class A
{
        public static void main(String a[] )
        {
                Test.show();
        }
}
```

Output:
Error

```
class Test
{
        static  int x=10;
           static void show ()
           {
                    System.out.println ("Static Method");
               System.out.println("x:::"+x);
           }
}
class Atext
{
           public static void main(String a[] )
           {
                    Test.show();
           }
}
```

Output:
Static Method
x:::10

1. They can only call other static methods.
2. They can only access static data
3. They can not refer to this or super keyword.

NESTING OF METHODS:

We know that a method of a class can be called only by an object of that class (or class itself, in the case of static method) using the dot operator. There is an exception to this ,A method can be called by using only its name by another method of the same class. This is known as Nesting of methods.

```
class A
{
  void display ()
           {
           System.out.print("1st method");
           }
void show()
           {
           System.out.print("2nd  method ");
        display()
        }
}
```

```
class test
{
 public static void main(String arg[])
  {
    A obj=new A();
    obj.show();
  }
}
```

Polymorphism

Q Polymorphism: Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behaviors in different instructions. In java, polymorphism is supported through

i) Overloading: - often referred to as a compiler-time or static polymorphism or Early Binding, or Static binding.

ii) Overriding: - often referred to as a run-time or dynamic polymorphism ,or lade binding or dynamic binding.

Q **Method overloading**: In Method overloading, Methods name is must be same, but have different Signature ,different return type as wall as different behaviors in same class.

Overloading Methods:

```
class abc
{
            int a,b;
            double c,d;
          void add()
          {
      System.out.println("No parameter");
          }
           void add(int r,int s)
           {
           a=r;   b=s;
            int n=a+b;
           System.out.println("int + int="+n);
           }
           void add(double r,double s)
           {
                      c=r;                    d=s;
                      double n=c+d;
System.out.println("double+double="+n);
           }
           double add(int r,int s,double t)
           {
                      a=r;
                      b=s;
                      c=t;
           System.out.println(a+b+c);
                      return (a+b+c);
           }
}
class overloding1
{
  public static void main(String args[])
           {
                      double v;
                      abc obj=new abc();
                    obj.add();
                      obj.add(10,12);
                      obj.add(10.2,10.3);
                      v=obj.add(1,2,3.3);
                      System.out.println(v);
           }
}
```

Q. Write a program find a square for
1) one input.
2) two input.
3) Three input.

Overloading Constructor Methods:

```
class a
{
          int a,b;
          double c,d;
          a(int r,int s)
          {
                    a=r;
                    b=s;
                    int n=a+b;
System.out.println("int + int"+n);
          }
          a(double r,double s)
          {
                    c=r;
                    d=s;
                    double n=c+d;
System.out.println("double+double"+n);
          }

a(int r,int s,double t)
          {
                    a=r;
                    b=s;
                    c=t;
                    System.out.println(a+b+c);

          }
}
class overloding
{
  public static void main(String args[])
          {
                    double v;
                    a obj=new a(10,12);
                    a obj1=new a(10.1,11.2);
                    //v=obj.add(5,4,12.3);
                    a obj2=new a(5,4,12.3);
          }
}
```

Q. Write a program find a square for
1) one input.
2) two input.
3) Three input.

Method overriding

Writing two or more method in super and sub classes such that the methods have same name and same signature but different behavior is called method overriding in different class.

Advance of inheritance: It is a technique in which child classes member function will execute. If member function name & signature of parent & child classes are same.

NOTE: super keyword calls any where in method overriding. BUT
super keyword call first line in sub classes method body in construct method.

```
//Example of Dynamic polymorphism
class A
{
        int c,d;
        void input()
        {
        c=20;   d=20;
        }
   void show()
     {
      System.out.println(c+d);
      System.out.println("Parent Class");
     }
}
class B extends A
{
        int a,b;
        void input()
        {
        a=10;    b=20;
        }
    void show()
        {
      System.out.println(c+d);
        System.out.println(a+b);
        System.out.println("Child class");
        }
}
class overridding1
{
   public static void main(String args[])
        {
        B obj=new B();   obj.input();  obj.show();
        }
}
Output:   0
          30
          Child class
According to above program, we can say the sub class method is replacing the super class method.
```

```
//Example of Dynamic polymorphism using super()
Class A
{
        int c,d;
        void input()
        {
        c=20;    d=20;
        }
     void show()
     {
     System.out.println(c+d);
     System.out.println("Parent Class");
     }
}
class B extends A
{
        int a,b;
        void input()
        {
        super.input();
        a=10;    b=20;
        }
   void show()
        {
        System.out.println(c+d);
        System.out.println(a+b);
      System.out.println("Chil d class");
        }
}
class overridding2
{
   public static void main(String args[])
        {
        B obj=new B();  obj.input();   obj.show();
        }
}
Output:   40
          30
          Child class
```

```
//Example of Static polymorphism
class A
{
static void show() {
      System.out.println("Parent Class");   }
}

class B extends A
{
        static void show()  {
       System.out.println("Chil d class");      }
}
```

```
class overridding4
{
   public static void main(String args[])
        {
 //Super class reference refers to sub class object
                A obj=new B();
                obj.show();
        }
}


Output:      Parent class
```

Note: Private methods are not available in the sub classes, so they cannot be overridden. So, private method overriding using a public method.

Method overloading	**Method** overriding
Writing two or more methods with the same name but with different signatures is called method overloading.	Writing two or more methods with the same name but with same signatures is called method overriding.
Method overloading is done in the same class.	Method overriding is done in the super and sub classes.
In method overloading, method return type can be same or different.	In method overriding, method return type should also be same.
JVM decides which method is called depending on the difference in the method signatures.	JVM decides which method is called depending on the data type (class) of the object used to call the mehod.
Method overloading is done when the programmer wants to extend the already available fueature.	Method overloading is done when the programmer wants to provide a different implementation (body) for the same featue.
Method overloading is code refinement. Same method is refined to perform a different task.	

Inheritance

Q. **Inheritance**: It is a mechanism, where one class inherits the all properties & methods (essential features) of an other class. A class from which inheritance, tack place is known as, Base class or super-class or parent class. A class that inherits the all the properties & methods of an other class is known as Derived class or sub class or child class. inherits by keyword "extends" .

Q Only private Data member or methods are not inherited by their child class.
Q **Type of inheritance**: i) Single Inheritance ii) Multilevel Inheritance iii) Multiple Inheritance.

```
//Example of  Single Inheritance
class A
{
     public int  a=10;
     void show()         {
          System.out.println(a);          }
}

class B extends A
{
      void display()       {
      System.out.println(a);        }
}
class Test
{
  public static void main(String arg[])
   {
     B obj=new B();
     obj.show();
      obj.display();
   }
}
```

```
//Example of  Multilevel Inheritance
class A
{
    int  a=10;
     void show()          {
          System.out.println(a);          }
}

Class B extends A
{
   int b=20;
   void display1()       {
     System.out.printf("MultiLavel");        }
}
class C extends  B
{
      void display()       {
       System.out.println(b+a);
      }
}
class Test
{
  public static void main(String arg[])
   {
     C obj=new C();
      obj.show();
      obj.display();
      obj.display1();
   }
}
```

Method overloading using Inheritance:

```
class A
{
        int a,b;
        double c,d;
        void add(int r,int s)   {
                a=r;       b=s;
                int n=a+b;
        System.out.println("int + int="+n);
        }
        void add(double r,double s)
        {
                c=r;  d=s;
                double n=c+d;
System.out.println("double+double="+n);
        }
        double add(int r,int s,double t)        {
                a=r;       b=s;       c=t;
                System.out.println(a+b+c);
                return (a+b+c);
        }
}
```

```
class B extends A
{
            void add( int r,int s,int t,int u)  {
        System.out.println("b"+(r+s+t+u));
            }
}

class overloding3
{
  public static void main(String args[])
        {
                double v;
                B obj=new B();
                obj.add(10,12);
                obj.add(12,2,3,4);
                obj.add(10.2,10.3);
                v=obj.add(1,2,3.3);
                System.out.println(v);
        }
}
```

Super keyword

In Java the super keyword is used to refer the constructor or a method of a parent class in an inheritance relationship. The super keyword cannot be used within in static context. For example, We cannot use super in main method. There are three forms of super:

1. super(): Use of this calls the default constructor of the parent class. It should appear as the first line in a constructor of a child class.
2. super.method_name(): Use of this calls a method of the parent class from a child class,
3. super.variable_name: Use of this type refers to a variable of the parent class.

Use of "super" keyword:

1. call superclass constructor. (Ex-1)
2. to access superclass members (either data or method member) (Ex-2)

call superclass constructor(Ex-1):

```
class sqr   {
        sqr()       {
          System.out.println("Defaul call");          }
       sqr(int a)  {
          System.out.println(a*a);             }
}
class cub extends sqr
{
          cub( int r,int s)   {
         // super();
 // default construct call by default.
          super(r);
          System.out.println((r+s)*(r+s));   }
}
class overloding3    {
  public static void main(String args[])     {
                    cub obj=new cub(10,12);  }
}
```

Overloading with super method(Ex-2):

```
class A  {
      int a=10;
       void sqr(int r)       {
          System.out.println((r*r));          }
}
class B extends A   {
           int a=20;
          void sqr( int r)   {
                    super.sqr(r);
          System.out.println(a);
          System.out.println(super.a);
          System.out.println((r+r));          }
}
class overloding3   {
  public static void main(String args[])     {
                    B obj=new B();
                    obj.sqr(9);                }
}
```

Note: When you assign one object reference variable to another object reference variable, you are not creating a copy of the object. You are only making a copy of the reference.

Using Objects as Parameters

```
class Box
{
  double width;
  double height;
  double depth;

  Box(Box obj)
    {
      width=obj.width;
      height=obj.height;
     depth=obj.depth;
}
boolean equals(Box obj)
{
  if(obj.width==width && obj.height==hight &&
obj.depth= =depth)
return true;
else
retyrn false;
}
}
```

```
class Test
{
    public static void main(String arg[])
     {
        Box obj1=new Box(100,200);
        Box obj2=new Box(100,200);
        Box obj3=new Box(1,2);

System.out.println("obj1==obj2:"+obj1.equals(obj2));
System.out.println("obj1==obj2:"+obj1.equals(obj3));
      }
}
```

NOTE: when a simple type is passed to a method it is done by use of call-by-value. Objects are passed by use of call-by-reference.

Final Keyword

The word final in Java means, "State of an entity cannot be changed". In Java the final modifier can be associated with class, reference variables, primitive variables and method.

"final " keyword is used in three ways:

i) It is used to declare constants, as final int a=10;

ii) It is used to before a method; this method can not be override. Private method and Final method are same.

```
Class A
{
   final void show()
        {
                System.out.println("not override a method");
        }
}
Class B
{
        void show()  //error
        {
                System.out.println("Error in this method");
        }
```

iii) It is used to prevent inheritance, as

```
final class A
{
        -------
        -------
}

Class B extend A  //Error
{
……
```

……..
}

iv) The reference variables declared as final cannot be re-instantiated with the new operator. That is, only once we can use the new operator.

For Variables	For Method	**For Class**

class variable_Final
{
p.s.v.m(String arg[])
{
final int a=10;
a=20; // Error value never change;
SOP("a:"+a);
}
}
Output: Error
	class A
	{
	final void display()
	{
	SOP("Not overriding");
	}
	}
	Class B extends A
	{
	Void display() //Error---
	{
	---------- 	
	}
	}
		final class A
		{
		void display()
		{
		SOP("Not overriding");
		}
		}
		Class B extends A //Error---
		{
		Void display()
		{
		---------- 	
		---------- 	
		}
		}

"finalize()" :- This method is called by the garbage collection when an object is removed from memory. We can garbage collector of JVM to delete any unused variables and unreferenced objects from memory using gc() method. This gc() method appears n both Runtime and System classes of java.lang package For example, we can call it as System.gc(); or Runtime.getRuntime().gc();

Abstract class

Q. **Abstract class: It is a collection of abstract methods as well as Concrete or Simple Method.**

If you want to make a abstract class. Just put "abstract" keyword before the class keyword.
E.g abstract class A{ }

Q. Abstract method: an abstract method without method body .An abstract method is written when the same method has to perform different tasks depending on the object calling.

Q. Characteristics of abstract class:

1. **abstract method are only declared in abstract class.**
2. **Defination is given for abstract method by their child class.**
3. **abstract method can not overriding by their child class.**
4. **You can not make constructor, static, private methods abstract.**
5. **We cannot use abstract classes to instantiate object directly.**

```
Example:
abstract class A
{
    public void show()
    {
    System.out.println("Wel come");
    }
   abstract void display();
   abstract void message();
}
class B extends A
{
    void display()
    {
System.out.println("1st abstract class");
    }

  void message()
{
System.out.println("2nd Abstract class");
}
}

class Test
{
    public static void main(String a[])
    {
        B obj=new B();
        obj.show();
        obj.display();
        obj.message();
    }
}
```

```
Abstract class A
{
  abstract  void calculate(int x);
}
class B extends A
{
  Void calculate(int x)
   {
System.out.println("Square="+Math.pow(x,2));
   }
}
class C extends A
{
  Void calculate(int x)
   {
System.out.println("Squ Root="+Math.Sqrt(x,2));
   }
}
class D extends A
{
  Void calculate(int x)
   {
System.out.println("Cube="+(x*x*x));
   }
}

class Test
{
    public static void main(String a[])
    {
        B obj1=new B();
        C obj2=new C();
        D obj3=new D();
        Obj1.calculate();
        Obj2.calculate();
        Obj3.calculate();
    }
}
```

Q. How can you force your programmers to implement only the features of your class?
Ans: By writing an abstract class and an interface.

Q.Can you declare a class as abstract and final also?
Ans : No, abstract class need sub class. But "final" key represents a sub classes which can not created.

Interfaces

Q. What is Interfaces?

Ans: It is a collection of only abstract method as well as constant variables or data member.

Characteristics of Interfaces:

1. All Methods of an Interfaces are by default an abstract & public.
2. No need to make or mention abstract before the method.
3. A class implements to an interface not extends.

```
e.g:  interface  Ravi                    class Rabi implements Ravi
                 {                       {
                 void show();              void display();
               }                         }
```

4. Interface creates by using interface keywords.

```
e.g:  interface  Ravi
      {
        void show();
      }
```

5. An interface cannot implements another interface.
6. An interface can extend another interface.

```
e.g:  interface  Ravi                    interface Rabi extends Ravi
                 {                       {
                 void show();              void display();
               }                         }
```

7. We cannot create an object to an interface, but we can create a reference of interface type.
8. It is possible to write a class within a interface.
9. A class can implement multiple interfaces.

```
e.g:  interface  Ravi                    interface Rabi
                 {                       {
                 void show();              void display();
               }                         }

  Class A
        {
        ……….
        ……….
        }
Class B extends A implements Rabi , Ravi
{
……..
……..
}
```

So no need of Multiple Inheritance in java.

10. An interface can have variables which are public static and final by default. This means all the variables of the interface are constants.

Q. Difference between an abstract class and an interface:

abstract class	Interface
1. An abstract class is written when there are some common features shared by all the object.	1. An interface is written when all the features are implemented different object.
2. When an abstract class is written, it is the duty of the programmer to provide sub classes to it.	2. An interface is written when the programmer wants to leave the implementation to the third party vendors.
3. An abstract class contains some abstract method and also concrete method.	3. An interface class contains only abstract method.
4. An abstract class can contain instance variables also.	4. An interface class can not contain instance variables. It contains only constants.
5. Abstract class is declared by using the keyword “abstract”.	5. Interface class is declared by using the keyword “interface”.

Q. Why multiple inheritance not Support?
Ans: In Multiple inheritances, sub classes are derived from multiple super classes. If two super classes have same names for their members (variables & methods) then which member is inherited into the sub class is the main confusion in multiple inheritance. This is the reason; java does not support the concept of multiple inheritances.

This confusion is reduced by using multiple interfaces.

Multiple inheritances properties solve by i) using interface & ii) using MultiLevel:

Example:

```
interface A
{
      void show();
      void display();
}
class B implements A
{
      public void show()
      {
      System.out.println("Interface");
      }
      public void display()
      {
       System.out.println("AIEMD");
      }
}

class test
{
      public static void main(String a[])
      {
            B obj=new B();
            obj.show();    obj.display();
      }
}
```

```
Interface A
{
      void show();
}
interface B
{
       void display();
}
class C implements A,B
{
      public void show()
      {
      System.out.println("Interface");
      }
      public void display()
      {
       System.out.println("AIEMD");
      }
}

class test
{
      public static void main(String a[])
      {
            C obj=new C();
            obj.show();
            obj.display();
      }
}
```

```
class A
{
    public int  a=10;
    void show()       {
        System.out.println(a);
     }
}

interface B
{
  final int b=20; //finale ,public & static
   void display();
}
```

```
class C extends A implements B
{
    public void display()
    {
     System.out.println(B.b);
    }
}

class Test
{
 public static void main(String arg[])
  {
    C obj=new C();
     obj.show();
     obj.display();
  }
}
```

Exception Handling

Q. Type of Error?
Ans: There are 3 types of errors

- Compiler-time error: All syntax errors will be detected and display by the java compiler and therefore these errors are known as Compiler-time error.
- Run-time error: These error represent inefficiency of the computer system to execute a particular statement. Run time errors are not detected by the Java compiler. The are detected by the JVM ,only at runtime.
- Logical error.

Q. Exception Handling:

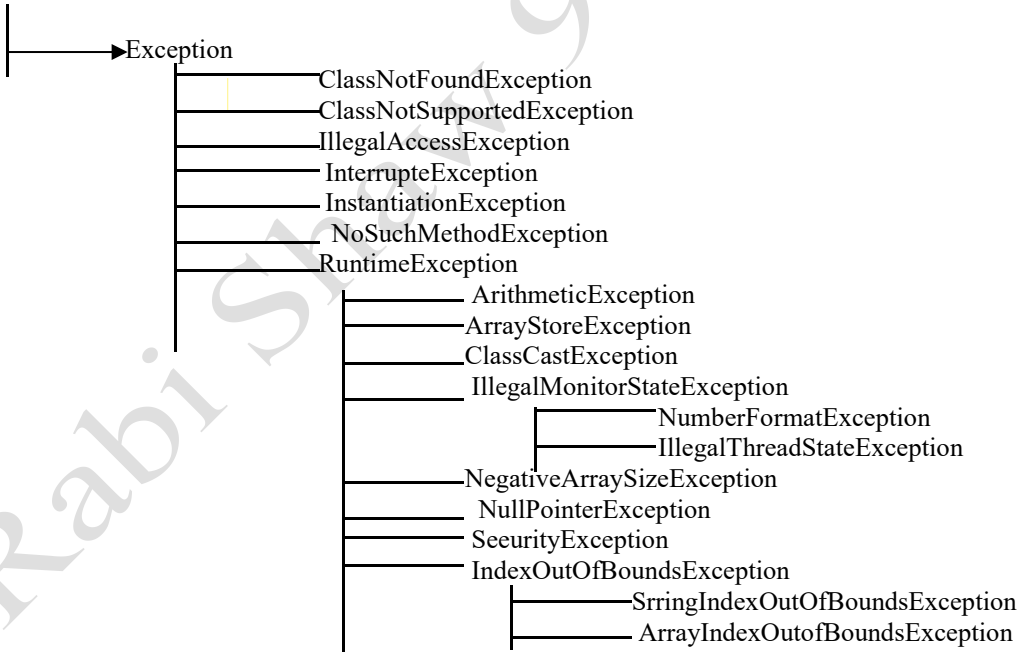
Q. why Exception handling?
Ans : After error part Excution.
Q. Type of Exception?
Ans: Exception there are 2 types

- Checked Exception: The Exception that is checked at compilation time by the Java compiler is called Checked Exception.

1. The executable class is not found in the classpath when we try to load a class using Class,forName. This is handled by ClassNotFoundException
2. There can be hardware faults during file reading and printing which is handled by the class IOException.
3. There can be some network failures to connect to a remote database or it may be a wrong SQL statement. All thes scenarios are presumed and will handled by the class SQLException.

- Unchecked Exception: The Exception that is checked by the JVM is called Unchecked Exception.

Q. Throwable



ClassNotFoundException	This exception is throws when the referenced class is not found in the classpath or no definition for the class with the specified name could be found.
CloneNotSupportedException	This Exception is thrown when the clone method in class Object has been called to clone an object, but that the object's class does not implement the Cloneable interface. Applecations that override the clone method can also throw this exception to indicate that an object could not or should not be cloned.

Q. What is the difference between an exception and an error?
An- exception is an error which can be handled. It means when an exception. -happens, programmer can do something to avoid any harm. But an error is an error which cannot- be handleel; happens and the programmer cannot do any thing.

Q. Five keywords are used for Handled Exception:
1. try 2. catch 3. throw 4. throws 5. finaly

Q Example: for try catch

```
class A
{
    public static void main(String args[])
     {
       int  a,b;
       a=10;
       b=0;
       System.out.println(a/b);
       System.out.println("Rabi");
     }
}

Compile =>javac filename.java
Run => java A
java.lang.ArithmeticException: / by zero
at A.main() method.
```

```
Class A
{
    public static void main(String args[])
     {
       int  a,b;        a=10;        b=0;
       try         {
        System.out.println(a/b);     }
        Catch(ArithmeticException ae)      {
         System.out.println(ae.getMessage);
         }
        System.out.println("Rabi");
      }
}
Compile =>javac filename.java
Run => java A
Division by zero ,illegal output
Rabi
```

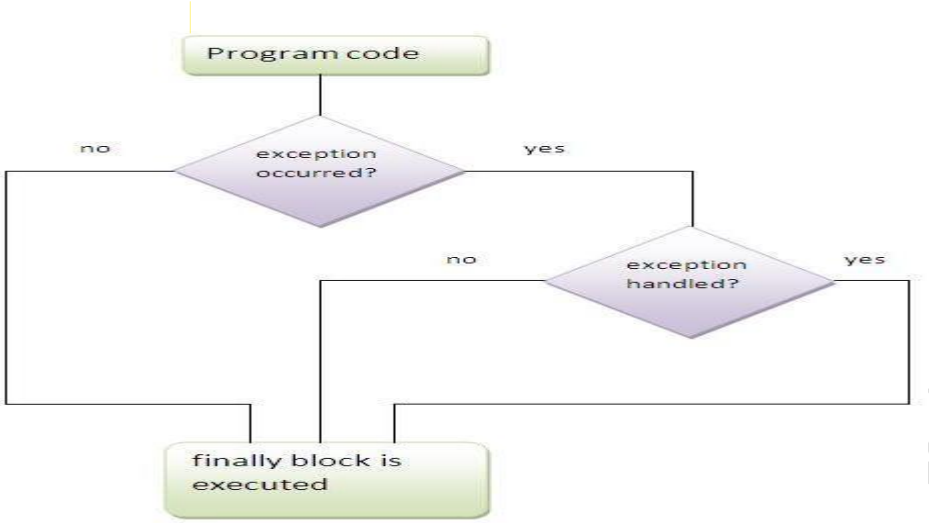
Example Multiple try-catch:

```
class  Multiplecatch
{
  public static void main(String arg[])
    {
          int a=arg.length;
          System.out.println(a);
          int c[]={1};
          try
Compiler-time erro            {
                    int b=42/a;
                    c[42]=99;
              }
           catch(ArithmeticException e)
           {
    System.out.println("Divide by 0:"+e);
        }
catch(ArrayIndexOutOfBoundsException e)
        {
   System.out.println("Array index oob:"+e);
        }
          System.out.print("Close Program");
    }
}
```

Nested try Statement:

```
class NestTry
{
 public static void main(String arg[])
  {
   try        {
       int a =arg.length;
       int b=42/a;
       System.out.println("a::"+a);
       try         {
                   if(a==1)
                   a=a/(a-a);
                  if(a==2)
                       {
                        int c[]={1};
                        c[42]=99;
                       }
                  }
   catch( ArrayIndexOutOfBoundsException e)
      {
    System.out.println("Arrayindex out-of-
bounds:"+e);
       }
catch(ArithmeticException e)   {
    System.out.println("Divide by 0:"+e);
       }
}
}
}
```

Q. Finally block: The finally block is a block that is always executed. It is mainly used to perform some important tasks such as closing connection.Before terminating the program , JVM executes finally block



Case1::

```
class Simple{
  public static void main(String args[])
{
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e)
{System.out.println(e);}
    finally{
System.out.println("finally block is always executed"); }
  System.out.println("rest of the code...");
  }
}
```

Output: Output:**5**
finally block is always executed
rest of the code...

Case 2::

```
class Simple{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e)
{System.out.println(e);}

  finally{
System.out.println("finally block is always executed");}
    System.out.println("rest of the code...");
  }
}
```

Output:finally block is always executed
Exception in thread main java.lang.ArithmeticException:/ by zero

Case3::

```
class Simple{
 public static void main(String args[]){
 try{
  int data=25/0;
  System.out.println(data);
 }
 catch(ArithmeticException e)
{System.out.println(e);}
    finally{
System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output: Exception in thread main
java.lang.ArithmeticException:/ by zero
finally block is always executed
rest of the code...

Rule: For each try block there can be zero or more catch blocks, but only one finally block.

Note: The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Q. Example: For finally block. finally block is executed even if Exception occur or not.

```
Import java.io.*;
class ex1
{
      public static void main(String args[]) throws IOException
      {
            int a,b;
            DataInputStream br=new DataInputStream(System.in);
            a=Integer.parseInt(br.readLine());
            b=Integer.parseInt(br.readLine());
            try
            {
            System.out.println(a/b);
            }
           catch(Exception e)              {
            System.out.println(e.getMessage());
            }
            finally
             {
             System.out.println("Work necessary");
             }
      }
}
```

```
Compile =>
javac filename.java
Run => java intab
10
0
/ by zero
Work necessary


Compile =>
javac filename.java
Run => java intab
10
5
2
Work necessary
```

Q. Example: for throws and throw

```
 public class Number
{
  static String str="A";
  public static void main(String arg[]) throws Exception
          {
           try
              {
            int i=Integer.parseInt(str);
           }catch(Exception e)
           {
            SOP(e.getmessage());
            trow new Exception("Not a valid number");
            }
      }
}
```

```
import java.io.*;
class Myclass extends Exception
{
   Myclass(String s)
     {
        System.out.print(s);
     }
}

class Main
{
public static void main(String ar[])throws IOException
{
DataInputStream di=new DataInputStream(System.in);
float b;
System.out.print("enter the value of b");
try
{
b=Float.parseFloat(di.readLine());
if(b!=3.14f)
throw new Myclass("NotEqualsException");
else
System.out.print(b);
}
catch(Exception e)
{
System.out.print(e.getMessage());
}
}
}
```

Q. Difference between throws and throw

1	throws use for method.	throw use for statement.
2	throws exception not handle by try-catch block.	throw exception handle by try-catch block.
3	throws exception create by user.	throw exception create by programmer.

Import java.io.*;
class intab
{
public static void main(String args[])
{
int a,b;
DataInputStream br=new DataInputStream(System.in);
a=Integer.parseInt(br.readLine());
b=Integer.parseInt(br.readLine());
System.out.println("a::"+a+"b::"+b);
}
}
	Compile =>
	javac filename.java
	
	IO Exception

Import java.io.*;
class intab
{
public static void main(String args[])throws IOException
{
int a,b;
DataInputStream br=new DataInputStream(System.in);
a=Integer.parseInt(br.readLine());
b=Integer.parseInt(br.readLine());
System.out.println("a::"+a+"b::"+b);
}
}
	Compile => javac filename.java
	Run => java intab
	10
	15
	a::10 b::15

Threads

Q. What is Thread?

Ans: Thread is small unit of program.

There are two type 1) single tasking. 2) Multi tasking.

Q. Single tasking: Only one task is given to the Processor.

Q. Disadvantage: In single tasking we are wasting a lot of processor time & microprocessor has sit idle without any job for a long time.

Q. Example for single tasking:

```
class Test extends Thread
{
        public void run()
        {
                for(int i=0;i<=5;i++)
                {
                        System.out.println(i);
                }
        }
}
class Test1
{
        public static void main(String a[])
        {
                Test obj=new Test();
                Thread t=new Thread(obj);
                t.start();
        }
}
```

```
class Test extends Thread
{
    public void run()
    {
      void task1();
      void task2();
      void task3();
    }
    void task1()
    {
     System.out.println("Task1");
    }
    void task2()
    {
     System.out.println("Task2");
    }
    void task3()
    {
     System.out.println("Task3");
    }
}
class Test1
{
        public static void main(String a[])
        {
                Test obj=new Test();
                Thread t=new Thread(obj);
                t.start();
        }
}
```

Q. Multi tasking: More then one task simultaneously performs.

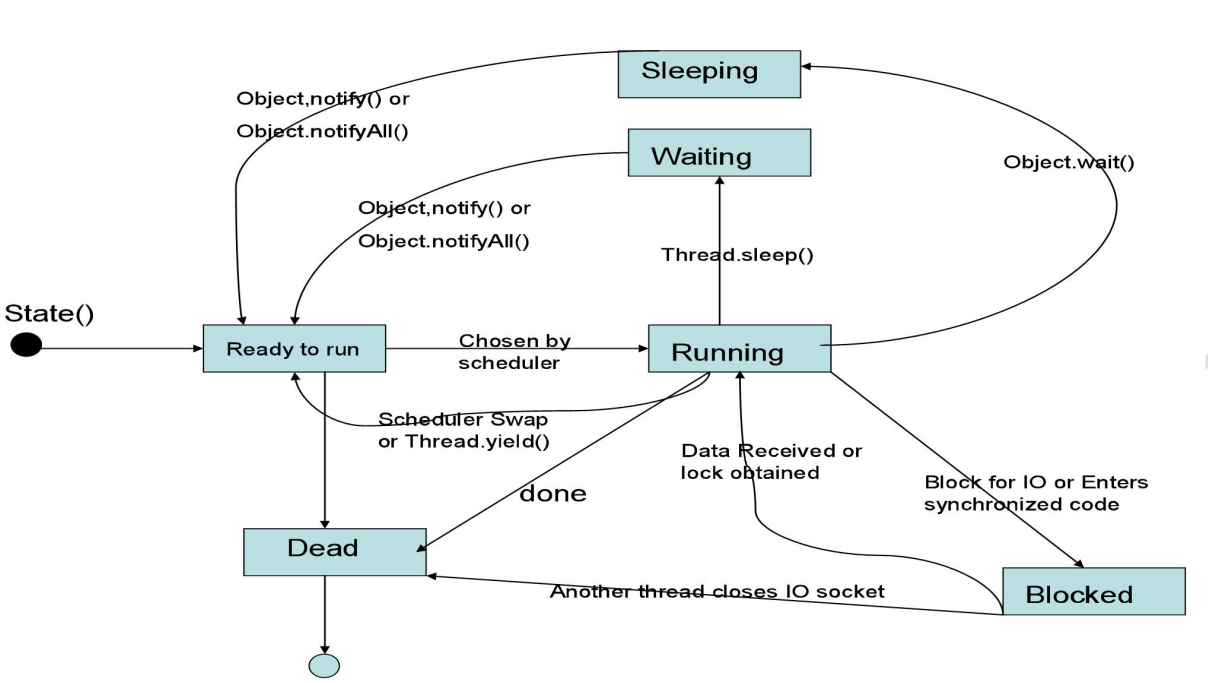
Q. Multi tasking is better then single tasking: Multi tasking is to use the processor time and it is not sitting idle. In this way, we can complete several tasking at a time.

Q. Example for single tasking:

```
class Test1 extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
        System.out.println("Test1"+i);
        }
    }
}
class Test2 extends Thread
{
   public void run()
    {
        for(int i=0;i<=5;i++)
        {
        System.out.println("Test2"+i);
        }
     }
}
```

```
class Test
{
        public static void main(String a[])
        {
           Test1 obj1=new Test1();
           Test2 obj2=new Test2();
           Thread t1=new Thread(obj1);
           Thread t2=new Thread(obj2);
           t1.start();
           t2.start();
        }
}
```

Q. **Thread Life Cycle:**



Q. **In Thread class some methods:**

Name: Meaning

1) Thread.start() ----------→ calls the run() method
2) Thread.stop() ----------→ Stops the thread

This is a property of thread class, used for stop the execution of current threads.
It takes Boolean value either true or false.

Syntax: Threadobject.stop=Boolean value

e.g t.stop=true;

3) Thread.suspend() ----------→ Suspends a thread execution
4) Thread.resume()----------→ resumes a suspends the thread
5) Thread.sleep(100)----------→ Sleeps for 100 milli seconds

Sleep methods used for hold execution of program for few seconds. This method automatically resumes.
Sleep methods al generate; an exception i.e InterraptedException.
So write sleep methods in try-catch Block.

Syntax : try

```
{
  sleep(1000);
}
catch (InterraptedException e)
{}
```

6) Thread.yeild() ----------→ yields the thread control to another thread
7) Thread.setPriority()----------→ Set thread priority of the current thread
8) Thread.getPriority()--------→Gets thread priority of the current thread.
9) Thread.currentThread()-----→gets the thread in which the method is running.
10) Thread.getName()----------→gets the name of the thread in which the method is running.

How can we improve the efficiency of communication betⁿ threads?

11) Thread.wait()----------→ Waits for something to happen till it receive a notification from a notify() or notifyAll() methods.

12) Obj.notify() : it is sends a notification to a waiting thread .
13) Obj.notifyAll() : it is sends a notification to all waiting thread .

Qu. What is the difference between yielding and sleeping?

When a task invokes its yield() method, it returns to the ready state. When a task invokes its sleep() method, it returns to the waiting state.

Q. Example:

Example for start method

```
class Test1 extends Thread   {
    public void run()       {
            for(int i=0;i<=5;i++)
            {
            System.out.println("Test1"+i);
            }
    }
}

class Test2 extends Thread
{
   public void run()
     {
            for(int i=0;i<=5;i++)
            {
            System.out.println("Test2"+i);
            } } }
class Test
{
            public static void main(String a[])
            {
               Test1 obj1=new Test1();
               Test2 obj2=new Test2();
               Thread t1=new Thread(obj1);
               Thread t2=new Thread(obj2);
               t1.start();
               t2.start();
            }
}
```

Example for stop method

```
import java.io.*;
class Test extends Thread
{
            boolean stop=false;
             public void run()
             {
                        for(int i=0;i<=50;i++)
                        {
                        System.out.println(i);
                        if(stop)
                        return;
                        }
            }
}

class Test1
{
   public static void main(String a[])
   throws IOException
            {
                  Test obj=new Test();
                Thread t=new Thread(obj);
                   t.start();
                   System.in.read();
                   obj.stop=true;
            }
}
```

Example for sleep method

```
class Test1 extends Thread
{
  public void run()
  {
     try
     {
            for(int i=0;i<=5;i++)
            {
            System.out.println("Test1"+i);
            sleep(1000);
            }
     }
     catch(Exception e)
     {
            System.out.println(e.getMessage());
      }
    }
}
```

```
Class Test1 extends Thread
{
  public void run()
   {
     try
            {
             for(int i=0;i<=5;i++)
               {
            System.out.println("Test1"+i);
                sleep(1000);
             }
             }
            catch(Exception e)
            {
            System.out.println("Test1");
            }
    }
}
```

```
class Test
{
        public static void main(String a[])
        {
            Test1 obj1=new Test1();
            Thread t1=new Thread(obj1);
            t1.start();
        }
}
```

```
class Test2 extends Thread
{
 public void run()
 {
  try
   {
    for(int i=0;i<=5;i++)
        {
        System.out.println("Test2"+i);
        sleep(100);
        }
    }
    catch(Exception e)
    {
        System.out.println("Test2");
     }
  }
}
class Test
{
    public static void main(String a[])
        {
            Test1 obj1=new Test1();
            Test2 obj2=new Test2();
          Thread t1=new Thread(obj1);
            Thread t2=new Thread(obj2);
            t1.start();
            t2.start();
        }
}
```

Q. **Thread Scheduling:**

```
import java.io.*;
class Test extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
        System.out.println(Thread.currentThread().getName());
        }
    }
}
```

Output for non pre-emptive scheduling

```
boys
boys
boys
boys
.
.

.
```

class Test1
{
public static void main(String a[]) throws IOException
{
Test obj=new Test();
new Thread(obj,"boys").start();
new Thread(obj,"girls").start();
}
}
	Output for pre-emptive scheduling
	
	boys
	boys
	boys
	boys
	boys
	girls
	girls
	girls
	.
	.
	.
	.
	.
	.

This scheduling problem is solve by yiled() method.

```
import java.io.*;
class Test extends Thread
{
    public void run()
    {
        for(int i=0;i<35;i++)
        {
            System.out.println(Thread.currentThread().getName());
            yield();

        }
    }
}

class Test1
{
    public static void main(String a[]) throws IOException
    {
        Test obj=new Test();
        new Thread(obj,"Boys").start();
        new Thread(obj,"Girls").start();
    }
}
```

Q. **Thread Priorities:**

Threads have priorities that can be set and changed. A higher priority thread executes ahead of a low priority thread. Priorities run from 1 to 10.

The minimum priority (like this Thread.MIN_PRIORITY) is 1.
The normal priority (like this Thread.NORM_PRIORITY) is 5.
The maximum priority (like this Thread.MAX_PRIORITY) is 10.
When a thread is created, by default its priority is 5.

Example with implements Runnable	**Example with Extends Threads**

class priorityes implements Runnable
{
public void run()
{
while(true)
{
System.out.println(Thread.currentThread().getName());
}
	class priorityes extends Thread
	{
	public void run()
	{
	while(true)
	{
	System.out.println(Thread.currentThread().getName());
	}

```
        }
}

class test
{
        public static void main(String a[])
        {
          priorityes obj=new priorityes();
          Thread t1=new Thread(obj,"Boys");
          Thread t2=new Thread(obj,"Girls");
           t2.setPriority(t1.getPriority()+5);
           t1.start();
           t2.start();
        }
}
```

```
        }
}

class test
{
public static void main(String a[])
{
        priorityes obj=new priorityes();
        Thread t1=new Thread(obj,"Boys");
        Thread t2=new Thread(obj,"Girls");
        t2.setPriority(t1.getPriority()+5);
        t1.start();
        t2.start();
        }
}
```

Q. **Synchronization:** When a thread is already action on an object, Preventing any other thread from action on the same object is called "Thread Synchronization" or "Thread safe".
Synchronization there are two way

2) Synchronization block:
Syntex: Synchronization(object)
{
Statement
}

3) Synchronization keyword :
Syntex: Synchronization void displat()
{
Statement
}

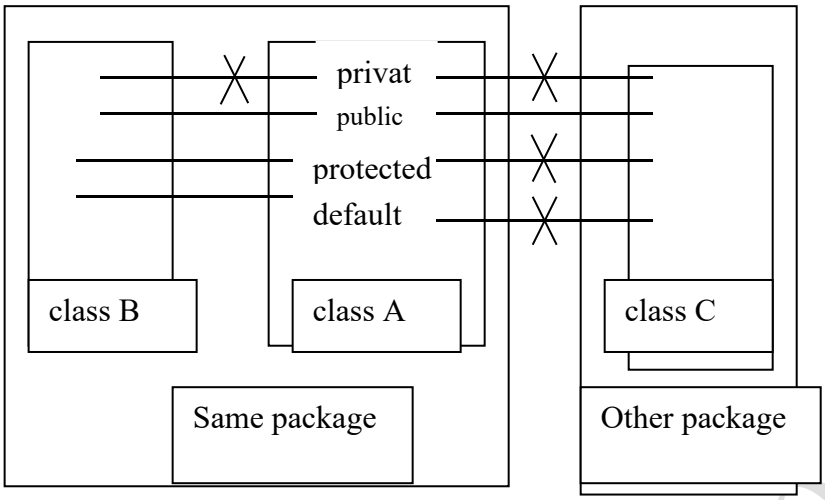
Q What is synchronization and why is it important?
With respect to multithreading, synchronization is the capability to control the access of multiple threads to Shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors.

Q. **What is Thread deadlock?**
Ans: When a thread has locked an object and waiting for another object to be released by another thread, and the other thread is also waiting for the first thread to release the first object, both the threads will continue waiting forever. This is called 'Thread deadlock'.

Packages

Q. What is Package?
Ans: It is a collection of related group of class and interfaces.
Q. Access Specifiers in java:

1. Private: private Scope is limited only for class scope.
2. Public: public Scope is global scope i.e. outside the class & outside the packages.
3. Protected: protected Scope is outside the class but not outside the packages.
   i.e. accessible specifier acts as public with respect to sub classes.
4. Default: It is access same package, not other package i.e. Scope is package scope.



For same package.	For Different package.

Step1: Create a folder in bin folder.
Step2:
public class apple
{
void show()
{
System.out.println("Very Sweet");
}
}
Step3: save apple.java **in new folder**.
Step4: compile => javac –d . apple.java
Step5:
class test
{
public static void main(String ar[])
{
apple obj=new apple();
obj.show();
}
}
Step6: save filename.java in new folder.
Step6: path set
c:\ProgramFiles\Java\jdk1.6.0_02\bin
\FRUITS set path="c\ Program Files\
Java\jdk1.6.0_02\bin"
Step7: compile => javac filename
	Step1:
	package cricket;
	public class sachin
	{
	public void run()
	{
	System.out.println("20000");
	}
	}
	Step2: save sachin.java **in bin folder.**
	Step3: compile =>javac sachin.java
	Step4:
	import cricket.packg;
	class packRabi
	{
	public static void main(String a[])
	{
	packg obj=new packg();
	obj.run();
	}
	}
	Step5: save filename.java in new folder.
	Step6: compile => javac filename

Class member Access:

	Privare	No Modifier	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Sub class	No	Yes	Yes	Yes
Same Package Non-Sub class	No	Yes	Yes	Yes
Different Package Sub class	No	No	Yes	Yes
Different Package Non-Sub class	No	No	No	Yes

Create Environment path:

1. My_Computer ->Right click->properties->Advanced->Environment->path

CLASSPATH: The CLASSPATH is an environment variable that tells the java compiler where to look for class files to import. CLASSPATH is generally set to a directory or a JAR (java Archive) file.

CLASS

Wrapper Class: A Wrapper class is a class whose object wraps or contains a primitive data type. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data type. In other words, we can wrap a primitive value into a wrapper class object.

Q. **Wrapper class:** A wrapper class is a class whose object wraps or contains a primitive data type.

Q. **Why do we need wrapper classes?** :-

1. They convert primitive data types into objects and this is needed on Internet to communicate between two applications.
2. The classes in java.util package handle only objects and hence wrapper classes help in this case also.

Q. **Boxing: Converting a primitive datatype into an object is called "boxing"**

Q. unboxing: Converting an object into corresponding primitive datatype is called unboxing

Example of AutoBox

```
class AutoBox
{
static int m(Integer v)
{
return v;
}
public static void main(String args[])
{
Integer iob=m(100);
System.out.println(iob);
}
}
```

Example of Unboxing:

```
public class UnBoxing
	{
public static void main(String args[])
	{Integer iob,iob2;
	int i;
	iob=100;
System.out.println("Original value of iob:
"+iob);
	++iob;
System.out.println("After ++iob: "+iob);
	iob2=iob+(iob/3);
System.out.println("iob2 after expression:
"+iob2);
	i=iob+(iob/3);
System.out.println("i after expression:"+i);
	}
	}
```


Nexted class or inner class

Outer variable access inner class but inner variable not access outside the outer class.

Class Outer
{
int outer_v=20;
class Inner
{
void display()
{
System.out.println(“Display outer Variable”+outer_v);
}
}
 	
void show()
{
Inner inner=new Inner();
inner.display();
}
}
 	
class Abc
{
public static void main(String arg[])
{
Outer outer=new Outer();
outer.show();
}
}
	Class Outer
	{
	int outer_v=20;
	class Inner
	{
	int inner_v=10;
	void display()
	{
	System.out.println("Display inner Variable" +outer_v);
	}
	}
	void show()
	{
	Inner inner=new Inner();
	inner.display();
	System.out.println("Display inner Variable" +inner_v);
	}
	.}
	class Abc
	{
	public static void main(String arg[])
	{
	Outer outer=new Outer();
	outer.show();
	}
	}
		Class Outer
		{
		int outer_v=20;
		class Inner
		{
		void display()
		{
		System.out.println(“Display outer Variable”+outer_v);
		}
		}
		
		void show()
		{
		Sop(outer_v);
		}
		}
		
		class Abc
		{
		public static void main(String arg[])
		{
		Outer outer=new Outer();
		Outer.Inner inner=outer.new Inner();
		outer.show();
		inner.display();
		}
		}

	Error: System.out.println("Display inner Variable" +inner_v);	

There are two types of nested class, they can access the members of the enclosing class through an object while non-static nested class can access the members of the enclosing class directly. The non-static nested class is known as inner class.

```
public class Test
{
   static class InnerClass
   {
      public static void InnerMethod()
      { System.out.println("Static Inner Class!"); }
   }
   public static void main(String args[])
   {
     Test.InnerClass.InnerMethod();
   }
}
```


Applet

Applet: An applet is a GUI program written in Java. Applets are embedded in a web page for eg. A HTML page. When the html page is viewed in a browser, the applet byte codes are downloaded in the client machine and is executed by the JVM of the web browser.

Q. Life Cycle of Applet:

1. init():- It is method, is the first method to be called. This is where you should initialize variables. This method is called only once during the run time of your applet.

2. start():- if call after init(). It is called to start or restart an applet. In some cases it may be called more than once.

3. paint():- The paint() method is called each time your applet's output must be redrawn. This situation can occur for several reasons.

4. stop():- The stop() method is called at least once ,when the browser window is stopped loading in which the applet is embedded. The applets start() method will be called if at some leader point the browser returns to the page containing the applet. In the cases the stop() method may be called multiple times in an applet's life cycle.

5. destroy():- The destroy() method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform clean up of resources held by the applet.

Q. **What are local applet and remote applet?**

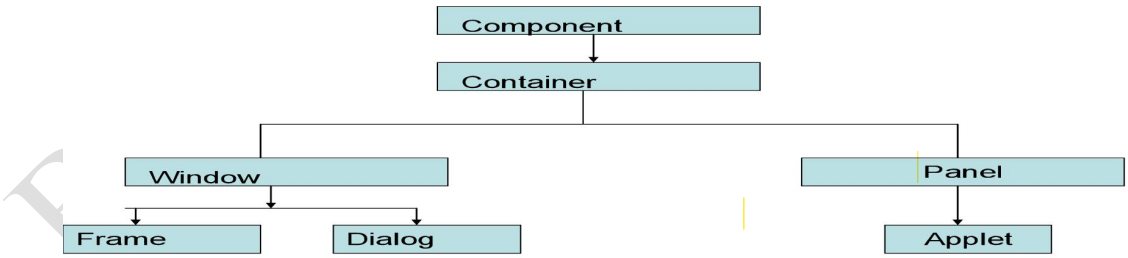
Ans: A local is the one which is stored locally in a computer system. When browser try to access the applet, it is not necessary for a computer to be connected to any network.

A remote applet is the one which is not stored in a computer system. When a browser tries to access such an applet it gets downloaded from the remote place.

Q. The class Color defines the constants show here that can be used to specify colors:

1. Color.black ;
2.Color.magenta;
3.Color.biue
4. Color.orange
5. Color.cuan
6. Color.pink
7. Color.darkGray
8. Color.red
9. Color.gray
10. Color.white
11. Color.green
12. Color.yellow
13. Color.lightGray

Q. AWT Package: The "Abstract Window Toolkit" (AWT) package in java enble the programmers to create GUI-based applications. It contains a number o classes that help to implement common Windows-based tasks, such as manipulating windows, adding scrollbars , buttons, list items, text boxes, etc. All the classes are contained in the "java.awt" package.



Some method of a Graphics class : Graphics class od java.awt package has the following method which help to draw various shapes.

1. **drawstring("String",x,y);**

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleLine  extends Applet
{
        public void paint(Graphics obj)
        {
                obj. drawString("Rabi Shaw",100,200);
        }
}
```

2.drawLine(int x1, int y1, int x2, int y2);
Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleLine  extends Applet
{
        public void paint(Graphics obj)
                {
                        obj.drawLine(100,100,200,100);
                }
}
```

3. **drawRect(int x, int y, inr w,int h):**This method draws outline of a rectangle. The left top corner of the rectanglestarts at (x,y), the width is w, and height is h.
Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
        public void paint(Graphics obj)
                {
                        obj.drawRect(100,100,200,100);
                }
}
```

4. drawRoundRect(int x, int y, inr w,int h, int arcw, int arch);
Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
        public void paint(Graphics obj)
                {
                        obj.drawRoundRect(100,100,200,100,20,20);
                }
}
```

5. drawOval(int x, int y, inr w,int h);
Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
```

```
	public void paint(Graphics obj)
		{
			obj.drawOval(100,100,200,100);
		}
}
```

6 .drawPolygon(int x[], int y[], int n);

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
int  x[]={20,50,100};
int y[]={50,80,150};
	public void paint(Graphics obj)
		{
			obj.drawPolygon(x,y,3);
		}
}
```

7. drawArc(int x, int y, inr w,int h, int start_angle , int difference_angle);

g.drawArc(x,y,w,h,90,180);	g.drawArc(x,y,w,h,180,-180);	g.drawArc(x,y,w,h,180,180);	g.drawArc(x,y,w,h,90,-180);

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
	public void paint(Graphics obj)
		{
			obj.drawArc(100,100,200,100,0,180);
		}
}
```

Example of Color class:

```
import java.applet.*;
import java.awt.*;

public class app_color extends Applet
{
	public void paint(Graphics g)
	{
		setBackground(Color.green);
		setForeground(Color.red);   //   g.setColor(Color.red); // Both are same function
		g.drawString("Rabi Shaw",100,100);
	}
}
/*
```

```
<html>
<applet code=app_color height=500 width=500>
</applet>
</html>
*/
```

Filling with Colors:

1. **fillLine(int x1, int y1, int x2, int y2);**

```
Example:
import java.awt.*;
import java.applet.*;
public class ExampleLine  extends Applet
{
        public void paint(Graphics obj)
                {
                        obj.setColor(Color.red);
                        obj.fillLine(100,100,200,100);
                }
}
```

2. **fillRect(int x, int y, inr w,int h):This method draws outline of a rectangle. The left top corner of the rectanglestarts at (x,y), the width is w, and height is h.**

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
        public void paint(Graphics obj)
        {
                obj.setColor(Color.red);
                obj.fillRect(100,100,200,100);
        }
}
```

3. **fillRoundRect(int x, int y, inr w,int h, int arcw, int arch);**

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
        public void paint(Graphics obj)
                {
                        obj.setColor(Color.red);
                        obj.fillRoundRect(100,100,200,100,20,20);
                }
}
```

4. **fillOval(int x, int y, inr w,int h);**

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
        public void paint(Graphics obj)
                {
                        obj.setColor(Color.red);
```

```
                    obj.fillOval(100,100,200,100);
                }
}
```

5. **fillPolygon(int x[], int y[], int n);**

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
int  x[]={20,50,100};
int y[]={50,80,150};
        public void paint(Graphics obj)
                {
                        obj.setColor(Color.red);
                        obj.fillPolygon(x,y,3);
                }
}
```

6. **fillArc(int x, int y, inr w,int h, int start_angle , int difference_angle);**

Example:

```
import java.awt.*;
import java.applet.*;
public class ExampleRectangle  extends Applet
{
        public void paint(Graphics obj)
        {
                obj.fillArc(100,100,200,100,0,180);
         }
   }
```

Q. What is the Difference between a window and frame?

Ans: A window is a frame without any borders and title, whereas a frame contains borders and title.

Method of Frame class:

1. f.setSize(w,h); // set the size of the Frame // in pixel;
2. f.setVisible(true);//Display the Frame
3. f.setTitle("xyz"); // set a Title for the Frame

Example:

```
import java.awt.*;
import java.applet.*;
public class MyFrame  extends Frame
{
        public static void main(String ar[])
                {
                        MyFrame f=new MyFrame();
                        f.setTitle("My Fream");
                        f.setSize(500,400);
                        f.setVisible(trye);
                }
}
```

Component	Listener	Listener methods

Button	**ActionListener**	**Public void actionPerformed (ActionEvent e)**
CheckBox	**ItemListener**	**Public void itemStateChanged (ItemEvent e)**
CheckBoxGroup	**ItemListener**	**Public void itemStateChanged (ItemEvent e)**
TextField	**ActionListener**	**Public void actionPerformed (ActionEvent e)**
	FocusListener	**Public void focusGained(FocusEvent e)** **Public void focusLost(FocusEvent e)**
TextArea	**ActionListener**	**Public void actionPerformed (ActionEvent e)**
	FocusListener	**Public void focusGained(FocusEvent e)** **Public void focusLost(FocusEvent e)**
Choice	**ActionListener** **ItemListener**	**Public void actionPerformed (ActionEvent e)** **Public void itemStateChanged (ItemEvent e)**
List	**ActionListener** **ItemListener**	**Public void actionPerformed (ActionEvent e)** **Public void itemStateChanged (ItemEvent e)**
Keyboard	**keyListener**	**Public void keyPressed(KeyEvent e)** **Public void keyReleased(KeyEvent e)** **Public void keyTyped(KeyEvent e)**
Label	**No listener is needed**	

1. **Example of Button:**

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class app_button extends Applet
{
 Button b;
 public void init()
          {
          b=new Button("Ok");
         add(b);
          }
}

/*
<html>
<applet code=app_button height=500 width=500>
</applet>
</html>
*/
```

2. **Example of Label:**

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class app_label extends Applet
{
 Label b;
 public void init()
          {
          b=new Label("Name");
         add(b);
          }
}
```

```
/*
<html>
<applet code=app_label height=500 width=500>
</applet>
</html>
*/
```

2. Example of TextField:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class app_textfeild extends Applet
{
 TextField t;
 public void init()
          {
          t=new TextField(20);
         add(t);
          }
}

/*
<html>
<applet code=app_textfeild height=500 width=500>
</applet>
</html>
*/
```

4. Example of TextArea:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class app_textarea extends Applet
{
 TextArea t;

 public void init()
          {
          t=new TextArea(4,6);
          add(t);
          }
}

/*
<html>
<applet code=app_textarea height=500 width=500>
</applet>
</html>
*/
```

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class sita2 extends Applet implements ActionListener
{
Button b1,b2;
public void init()
{
b1=new Button("red");
b2=new Button("green");
add(b1);
add(b2);
b1.addActionListener(this);
b2.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
String s=ae.getActionCommand();
if(s.equals("red"))
setBackground(Color.red);
else if(s.equals("green"))
setBackground(Color.green);
}
}
/*<HTML>
<Applet code=sita2 height=1000 width=1000>
</Applet>
</HTML>*/
```

```
import java.awt.*;
import java.applet.*;

public class TestApplet2 extends Applet
{
        public void paint(Graphics obj)
        {
        obj.drawLine(100,100,200,100);
        obj.drawLine(100,100,100,200);
        obj.drawLine(100,200,200,200);
        obj.drawLine(200,100,200,200);
        }
}
```

```
import java.awt.*;
import java.applet.*;

public class TestApplet3 extends Applet
{
        public void paint(Graphics obj)
        {
        obj.drawLine(100,100,200,100);
        obj.drawLine(100,100,100,200);
        obj.drawLine(100,200,200,200);
        obj.drawLine(200,100,200,200);
        obj.drawString("Ravi Sir",125 ,151);
          }
}
```

```
Compile=>
javac TestApplet3.java
Run=>
appletviewer TestApplet3.java
```

```
import java.awt.*;
import java.applet.*;

public class TestApplet5 extends Applet
{
        String msg;
        int x[]={100,200,250,50,100};
        int y[]={50,50,200,200,50};

        public void init()
        {
                setBackground(Color.green);
                setForeground(Color.red);
                msg="Inside init....";
        }
```

```
import java.awt.*;
import java.applet.*;

public class TestApplet4 extends Applet
{
        String msg;
        public void init()
        {
                setBackground(Color.green);
                setForeground(Color.red);
                msg="Inside init....";
        }
        public void start()
        {
                msg +="Inside start....";
```

```
        public void start()
        {
                msg +="Inside start....";
        }
        public void paint(Graphics obj)
        {
                obj.drawLine(100,100,200,100);
        }

}
```

```
        }
        public void paint(Graphics obj)
        {
        obj.drawLine(100,100,200,100);
        obj.drawLine(100,100,100,200);
        obj.drawLine(100,200,200,200);
        obj.drawLine(200,100,200,200);
        obj.drawString("Ravi Sir",125 ,151);
        }
}
```

```
/*      <html>
        <applet code=mouse5 height=200 width=300>
        </applet>
        </html>
*/

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class mouse5 extends Applet implements
ActionListener
{
        String msg="";
        Button yes,no,maybe,yes1;
        public  void init()
        {
                yes=new Button("Yes");

                no=new Button("No");
                maybe=new Button("Undecided");

                yes1=new Button(msg);

                add(yes);
                add(no);
                add(maybe);

                add(yes1);

                yes.addActionListener(this);
                no.addActionListener(this);
                maybe.addActionListener(this);

        }

        public void actionPerformed(ActionEvent ae)
        {
       String str =ae.getActionCommand();
                if(str.equals("No"))
                msg="You Pressed No";
               else if(str.equals("Yes"))
                msg="You Pressed Yes";
                else
                msg="You Pressed Undecided";

        }
        public void paint(Graphics g)
        {
                g.drawString(msg,6,100);
```

Q Add 2 no's……

```
/*      <html>
        <applet code=mouse7 height=200 width=200>
        </applet>
        </html>
*/

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class mouse7 extends Applet implements
ActionListener
{
        Label no1,no2;
        Button sum;
        int a,b,c;
        String msg;
        TextField tno1,tno2,tsum;
        public void init()
        {
                no1=new Label("No1");
                no2=new Label("No2");
                tno1=new TextField();
                tno2=new TextField();
                sum=new Button("sum");
                tsum=new TextField();

                add(no1);
                add(no2);
                add(tno1);
                add(tno2);
                add(sum);
                add(tsum);

                sum.addActionListener(this);
        }

public void actionPerformed(ActionEvent ae)
        {
        String str =ae.getActionCommand();
                if(str.equals("sum"))
                {
        a=Integer.parseInt(tno1.getText());
        b=Integer.parseInt(tno2.getText());
        c=(a+ b);
        tsum.setText(String.valueOf(c));
        //repaint();
        }
}
```

g.drawLine(100,200,300,400);
}
}
	public void paint(Graphics g)
	{
	//repaint();
	}
	}

Q c to f:::::::::::::::::::::::::::::::: **import java.applet.*;** **import java.awt.*;** **import java.awt.event.*;** **public class A extends Applet implements ActionListener** **{** **TextField t1,t2;** **Label l1,l2;** **Button b1,b2;** **public void init()** **{** **t1=new TextField(12);** **t2=new TextField(12);** **l1=new Label("enter the value");** **l2=new Label("output");** **b1=new Button("farren");** **b2=new Button("calcius");** **add(l1); add(t1); add(l2); add(t2);** **add(b1); ad**	**b1.addActionListener(this);** **b2.addActionListener(this);** **}** **public void actionPerformed(ActionEvent ae)** **{** **String str= ae.getActionCommand();** **if(str.equals("farren"))** **{** **double a=Double.parseDouble(t1.getText());** **double b=9*a/5+32;** **t2.setText(String.valueOf(b));** **}** **if(str.equals("calcius"))** **{** **double a=Double.parseDouble(t1.getText());** **double b=(5*a-160)/9;** **t2.setText(String.valueOf(b));** **}** **}** **}**

Project:
EMPLOYEE MANAGEMENT SYSTEM:
SOURCE CODING :

REGISTRATION PAGE

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

public class registraction extends JFrame implements ActionListener      {
        private static final long serialVersionUID = 1L;
        JLabel a1,a2,a3,a4,a5,a6,a7,a8;
        JTextField t1,t2,t5,t6,t7;
        JPasswordField t4,t3;
        JButton b1,b2,b3;
        JRadioButton r1, r2;
        ButtonGroup bg;
        public  registraction() {
```

```
a1 = new JLabel(" Enter  Employee Name ");
t1 = new JTextField(10);
a2=new JLabel(" Employee Id ");
t2 = new JTextField(10);
a3=new JLabel(" Enter password ");
t3 = new JPasswordField(10);
a4=new JLabel(" Re-Enter password ");
t4 = new JPasswordField(10);
a5=new JLabel(" Date ");
t5 = new JTextField(10);
a8=new JLabel(" DD-JAN-YYYY ");
a6=new JLabel(" Phone No ");
t6 = new JTextField(10);
a7=new JLabel(" Nationality ");
t7 = new JTextField(10);
b1=new JButton(" Submit ");       b1.addActionListener(this);
b2=new JButton(" Reset ");         b2.addActionListener(this);
b3=new JButton(" Back ");           b3.addActionListener(this);
r1=new JRadioButton(" Male ",true);
r1.addActionListener(this);
r2=new JRadioButton(" Female ",false);
r2.addActionListener(this);
 bg=new ButtonGroup();
bg.add(r1);     bg.add(r2);
setLayout(new GridLayout(7,1));
JPanel p;
p = new JPanel(); p.add(a1); p.add(t1); add(p);
p = new JPanel(); p.add(a2); p.add(t2); add(p);
p = new JPanel(); p.add(a3); p.add(t3); add(p);
p = new JPanel(); p.add(a4); p.add(t4); add(p);
p = new JPanel(); p.add(a5); p.add(t5);p.add(a8); add(p);
p = new JPanel(); p.add(a7); p.add(t7); add(p);
p = new JPanel(); p.add(a6); p.add(t6); add(p);
p = new JPanel(); p.add(b1); p.add(b2); p.add(b3); add(p);
p = new JPanel(); p.add(r1); p.add(r2); add(p);
setSize(600,600);
setTitle("My window");
setResizable(false);
setResizable(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public void actionPerformed(ActionEvent ae) {
String s=ae.getActionCommand();
if(s.equals(" Reset "))     {
t1.setText(""); t2.setText(""); t3.setText(""); t4.setText(""); t5.setText(""); t6.setText("");
t7.setText("");
}
if(s.equals(" Back "))      {
          dispose();
          new LogIn();     }
else if(s.equals(" Submit "))     {
String a = t1.getText();
String b = t2.getText();
String c =t3.getText();
String d=t4.getText();
```

```
                String e = t5.getText();
                int f =Integer.parseInt(t6.getText());
                String g = t7.getText();
                String j="";
                if(r1.getModel().isSelected()) j=r1.getLabel();
                if(r2.getModel().isSelected()) j=r2.getLabel();
                if(d.equals(c))     {
                try   {
                        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                        Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system","123");
                        Statement statement = connection.createStatement();
                    statement.executeUpdate("insert into reg
values('"+a+"','"+b+"','"+c+"','"+e+"','"+f+"','"+j+"','"+g+"')");
                        JOptionPane.showMessageDialog(this," U Are Register Sucessfully ");
                        statement.close();
                        connection.close();          }
                catch (Exception ex)      {
                        System.out.println("The exception raised is:" + ex);
                                        }                       }
                else    {
                JOptionPane.showMessageDialog(this," Your password & Re-password not match ");
                          }
                  }
                 }
        }
```

LOGIN PAGE

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class LogIn  extends JFrame implements ActionListener    {
        private static final long serialVersionUID = 1L;
        ImageIcon i;
        JPanel p;
        JLabel a1,a2,a3,l2;
        JButton b1,b2,b3,b4;
        JTextField t1,t2;
        Boolean p1=false;
        public LogIn() {
                i=new ImageIcon("shubh.jpg");
```

```
                l2=new JLabel(i);
        a1=new JLabel(" User LogIn ");
        a2=new JLabel(" USER  EMPLOYEE  ID ");
        a3=new JLabel(" ENTER  PASSWORD ");
        b1=new JButton(" OK ");                          b1.addActionListener(this);
        b1.setToolTipText("press to take Action");
        b2=new JButton(" RESET ");                       b2.addActionListener(this);
        b2.setToolTipText("press to take Action");
        b3=new JButton(" New User ");                    b3.addActionListener(this);
        b3.setToolTipText("press to take Action");
        b4=new JButton(" EXIT ");                        b4.addActionListener(this);
        b4.setToolTipText("press to take Action");
        t1=new JTextField(20);
        t2=new JPasswordField(20);
        l2.setBounds(0,0,300,270);
        setLayout(new GridLayout(5,1));
        p=new JPanel(); p.add(a1); add(p);
        p=new JPanel(); p.add(a2); p.add(t1); add(p);
        p=new JPanel(); p.add(a3); p.add(t2); add(p);
        p=new JPanel(); p.add(b1); p.add(b2); p.add(b3); p.add(b4); add(p);
        pack();
        setSize(600,600);
        setTitle("My Window");
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);        }
public void actionPerformed(ActionEvent ae)      {
                String s=ae.getActionCommand();
                if(ae.getSource()==b2)      {
                t1.setText("");            t2.setText("");
                                        }
        else if(s.equals(" New User "))      {
                setVisible(false);
                 registraction r2=new registraction();
                        r2.setVisible(true);              }
        else if(ae.getSource()==b4)      {
                        System.exit(0);     }
        else if(ae.getSource()==b1)    {
                String a = t1.getText();
                String b = t2.getText();
                Try    {
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
System.out.println("Connecting to the database111...");
                Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system","123");
                Statement statement = connection.createStatement();
ResultSet res=statement.executeQuery("select 1 from reg where emp_id ='"+a+"' and password ='"+b+"'");
      while(res.next())  {
                        a=res.getString(1);
                        System.out.println(a);
                        if(a.equals("1"))
                        p1 = true;          }
                statement.close();
                 connection.close();
        }
        catch (Exception ex)
```

```
        {
                System.out.println("The exception raised is:" + ex);
        }

if(p1)
{
         setVisible(false);
         EmployeeScreen r1=new EmployeeScreen();
        r1.setVisible(true);
}
 else
  {
         JOptionPane.showMessageDialog(this,"Invaild  User name or password ");

        }
                }
}
        public static void main(String ar[])
        {
        new LogIn();
        }
        }
```

EMPLOYEE SEARCH PAGE

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;
public class MyFrame2 extends JFrame implements ActionListener
{
private static final long serialVersionUID = 666673306509800602L;
String s1,s2,s3,s4,s5,s6,s8;
int s7;
JLabel  a1,a2;
JTextField t1;
JButton b1,b2,b3;
boolean p1=false;

public MyFrame2() {
a1 = new JLabel("Employee  Search Screen  ");
a2=new JLabel("Enter Employee No");
t1 = new JTextField(20);
b1 = new JButton("Search");
b1.addActionListener(this);
b2 = new JButton("Reset");
b2.addActionListener(this);
b3 = new JButton("BACK");
b3.addActionListener(this);
setLayout(new GridLayout(3,1));
JPanel p;
p = new JPanel(); p.add(a1); add(p);
p = new JPanel(); p.add(a2); p.add(t1); add(p);
p = new JPanel(); p.add(b1); p.add(b2); p.add(b3);add(p);
setSize(500,500);
setTitle("My window");
```

```
setResizable(false);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public void actionPerformed(ActionEvent ae)
{
        String s=ae.getActionCommand();
        System.out.println(s);
        if(s.equals("Reset"))
        {
                t1.setText("");
        }
        else if(s.equals("BACK")){
                dispose();
                MyFrame3 b4=new MyFrame3();
                        b4.setVisible(true);        }
        else if(s.equals("Search"))
        {
        String a=t1.getText();
        System.out.println(a);

        try {
                DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                System.out.println("Connecting to the database111...");
                Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system","123");
                Statement statement = connection.createStatement();
                ResultSet res=statement.executeQuery("select * from emp_add where emp_no ='"+a+"'");
                System.out.println("Connecting to the database.222....");

                while(res.next()){
                        s1=res.getString(1);
                        System.out.println(s1);
                        if(s1.equals(a))
                        {
                            p1=true;
                        s2=res.getString(2);
                        System.out.println(s2);
                        s3=res.getString(3);
                        s4=res.getString(4);
                        s5=res.getString(5);
                        s6=res.getString(6);
                        s7=res.getInt(7);
                        s8=res.getString(8);

                        }
                }
                //System.out.println(s1+s2+s3+s4+s5+s6+s7+s8);
        statement.close();
        connection.close();
        }
        catch (Exception ex)
        {
                System.out.println("The exception raised is:" + ex);
        }
```

```
            if(p1)
            {
                    setVisible(false);
                    new SearchScreen(s1,s2,s3,s4,s5,s6,s7,s8);
            }
            else
              {
                    JOptionPane.showMessageDialog(this,"Invaild Emp No ");
            }
            p1=false;
            }
}
}
```

DELETE EMPLOYEE PAGE

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;

class MyFrame4 extends JFrame implements ActionListener
{
private static final long serialVersionUID = -9041081572646088834L;
JLabel  a1,a2;
JTextField t1,t2,t3;
JButton b1,b2,b3;
boolean p1=false;
String s1,s2,s3,s4,s5,s6;
int s7,s8;
public MyFrame4() {
a1 = new JLabel("Employee Delete Screen  ");
a2=new JLabel("Enter Employee No");
t1 = new JTextField(10);
b1 = new JButton(" Delete ");          b1.addActionListener(this);
b2 = new JButton("Reset");            b2.addActionListener(this);
b3 = new JButton("Back");              b3.addActionListener(this);
setLayout(new GridLayout(3,1));
JPanel p;
p = new JPanel(); p.add(a1); add(p);
p = new JPanel(); p.add(a2); p.add(t1); add(p);
p = new JPanel(); p.add(b1); p.add(b2); p.add(b3); add(p);
pack();
setSize(500,500);
setTitle("My window");
setResizable(false);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);        }
public void actionPerformed(ActionEvent ae)       {
        String s=ae.getActionCommand();
if(s.equals("Reset"))     {
t1.setText("");            }
else if(s.equals("Back"))     {
        dispose();
```

```
  new   MyFrame3();         }
else if(s.equals(" Delete "))     {
          s1=t1.getText();
          try {
                        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                        Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system","123");
Statement statement = connection.createStatement();
ResultSet res1=statement.executeQuery("select * from emp_add where emp_no ='"+s1+"'");
System.out.println("Connecting to the database.222....");
        while(res1.next()){
                              s2=res1.getString(1);
                              if(s1.equals(s2))      {
                              p1=true;                  }
                        }
                        System.out.println(s1+s2+s3+s4+s5+s6+s7+s8);
                statement.close();
                connection.close();
                }
                catch (Exception ex)        {
                        System.out.println("The exception raised is:" + ex);
                                        }
        try {
                 DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                System.out.println("Connecting to the database111...");
Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
"system","123");
Statement statement = connection.createStatement();
           if(p1) {
                JOptionPane.showMessageDialog(this,"  Deleted  Sucessfully  ");
                        }
        else {
                 JOptionPane.showMessageDialog(this,"Invaild Emp No ");          }
                        p1=false;
                        statement.close();
                        connection.close();      }
                catch (Exception ex)        {
                          System.out.println("The exception raised is:" + ex);
                                        }
                                }
                        }
        }
```

INSERT EMPLOYEE PAGE

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import javax.swing.*;
public class MyFrame1 extends JFrame implements ActionListener  {
JLabel a1,a2,a3,a4,a5,a6,a7,a8,a9;
JButton b1,b2,b3;
JTextField t1,t2,t3,t4,t5,t6,t7,t8;
```

```
public MyFrame1()
{
a1=new JLabel(" EMPLOYEE INSERT ");
a2=new JLabel(" ENTER EMPLOYEE NO ");
a3=new JLabel(" ENTER EMP.NAME ");
a4=new JLabel(" DOB ");
a5=new JLabel(" Martial Status ");
a6=new JLabel(" Nationality ");
a7=new JLabel(" Designation ");
a8=new JLabel(" Salary ");
a9=new JLabel(" Phone No");
b1=new JButton("ADD");                          b1.addActionListener(this);
b1.setToolTipText("press to take Action");
b2=new JButton(" RESET ");                      b2.addActionListener(this);
b2.setToolTipText("press to take Action");
b3=new JButton(" BACK ");                       b3.addActionListener(this);
b3.setToolTipText("press to take Action");
t1=new JTextField(20);  t2=new JTextField(20);  t3=new JTextField(20);t4=new JTextField(20);
t5=new JTextField(20);t6=new JTextField(20);t7=new JTextField(20); t8=new JTextField(20);
setLayout(new GridLayout(9,1));
JPanel p;
p=new JPanel(); p.add(a1); add(p);
p=new JPanel(); p.add(a2); p.add(t1); add(p);
p=new JPanel(); p.add(a3); p.add(t2); add(p);
p=new JPanel(); p.add(a4); p.add(t3); add(p);
p=new JPanel(); p.add(a5); p.add(t4); add(p);
p=new JPanel(); p.add(a6); p.add(t5); add(p);
p=new JPanel(); p.add(a7); p.add(t6); add(p);
p=new JPanel(); p.add(a8); p.add(t7); add(p);
p=new JPanel(); p.add(a9); p.add(t8); add(p);
p=new JPanel(); p.add(b1); p.add(b2); p.add(b3); add(p);
pack();
setSize(800,400);
setTitle("My Window");
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent ae)      {
        String s,s1,s2,s3,s4,s5,s6,b;
        int  a;
s=ae.getActionCommand();
System.out.println(s);
if(s.equals(" RESET "))     {
t1.setText(""); t2.setText(""); t3.setText(""); t4.setText(""); t5.setText("");
t6.setText(""); t7.setText(""); t8.setText("");
 }
else if(s.equals("ADD"))    {
s1=t1.getText();  s2=t2.getText(); s3=t3.getText();  s4=t4.getText();  s5=t5.getText();
s6=t6.getText();
a=Integer.parseInt(t7.getText());
b=t8.getText();
try {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
"system", "123");          java.sql.Statement statement = connection.createStatement();
```

```
        statement.executeUpdate("insert into emp_add
values('"+s1+"','"+s2+"','"+s3+"','"+s4+"','"+s5+"','"+s6+"',"+a+","+b+")");
        JOptionPane.showMessageDialog(this," Add Sucessfully ");
        statement.close();
        connection.close();
}
catch (Exception ex)    {
        System.out.println("The exception raised is:" + ex);          }
}
else if(s.equals(" BACK "))   {
 dispose();
  new MyFrame3();    }
    }
  }
```

EMPLOYEE SCREEN PAGE

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
class  EmployeeScreen  extends JFrame  implements ActionListener   {
        private static final long   serialVersionUID = 1L;
        JButton b1,b2,b3,b4;
public EmployeeScreen()  {
        b1 = new JButton(" EMPLOYEE ");                b1.addActionListener(this);
        b1.setToolTipText("press to   take Action");
        b2 = new JButton(" PAYMENT  ");          b2.addActionListener(this);
        b2.setToolTipText("press to   take Action");
        b3 = new JButton(" CONTACT ");              b3.addActionListener(this);
        b3.setToolTipText("press to  take Action");
        b4 = new JButton(" LOGOUT ");                          b4.addActionListener(this);
        b4.setToolTipText("press to  take Action");
        setLayout(new GridLayout(4,1));
        JPanel p;
        p=new JPanel();   p.add(b1);   add(p);
        p=new JPanel(); p.add(b2);    add(p);
        p=new JPanel(); p.add(b3);    add(p);
       p=new JPanel(); p.add(b4);    add(p);
       setSize(600,600);
       setTitle("My window");
       setVisible(true);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
public void actionPerformed(ActionEvent ae)
{
        String p1=ae.getActionCommand();
        if(p1.equals(" EMPLOYEE "))
        {
        dispose();
        new MyFrame3();
        }
```

```
        else if(p1.equals(" PAYMENT  ")) {    dispose();      new ReportScreen();         }
        else if(p1.equals(" CONTACT ")) {
                dispose();
JOptionPane.showMessageDialog(this," for more detail call on no 7699487844  Or email at
shubhamraj844@gmail.com");
                new EmployeeScreen();
        }
        else if(p1.equals(" LOGOUT "))     {
                dispose();
        LogIn r2=new LogIn();
                r2.setVisible(true);        }
        }
public static void main(String[]  arg)
{               new EmployeeScreen();
  }
   }
```

EMPLOYEE MANAGEMENT PAGE

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class MyFrame3 extends JFrame implements  ActionListener   {
        JButton b1,b2,b3,b4;
public MyFrame3()   {
b1 = new JButton(" Insert  Employee ");
b1.addActionListener(this);
b1.setToolTipText("press to take Action");
b2 = new JButton(" Search  Employee ");
b2.addActionListener(this);
b2.setToolTipText("press to take Action");
b3 = new JButton(" Delete  Employee ");
b3.addActionListener(this);
b3.setToolTipText("press to take Action");
b4 = new JButton("   Back   ");
b4.addActionListener(this);
b4.setToolTipText("press to take Action");
setLayout(new GridLayout(5,1));
JPanel p;
p=new JPanel(); p.add(b1); add(p);
p=new JPanel(); p.add(b2); add(p);
p=new JPanel(); p.add(b3); add(p);
p=new JPanel(); p.add(b4); add(p);
setSize(600,600);
setTitle("My window");
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public void actionPerformed(ActionEvent ae)
{
        String p=ae.getActionCommand();
        System.out.println(p);
if(p.equals(" Insert  Employee "))
{
setVisible(false);
```

```
new MyFrame1();
}
else if(p.equals(" Search  Employee ")) { dispose(); new MyFrame2(); }
else if(p.equals(" Delete  Employee ")) { dispose(); new MyFrame4(); }
else if(p.equals("   Back   "))
{ setVisible(false);
EmployeeScreen r2=new EmployeeScreen();
        r2.setVisible(true);
                }
}
}
```

PAYMENT PAGE

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class ReportScreen extends JFrame implements ActionListener   {
        private static final long serialVersionUID = 1L;
        JLabel a1,l1,l2;
        JButton b1,b2,b3;
   JTextField t1,t2;
   String s1;
   int s7;
   boolean p1=false;
        public ReportScreen()
        {
                l1=new JLabel("Enter emp_no");
                l2=new JLabel("Payment");
                b1=new JButton("Search");
                b1.addActionListener(this);
                b2=new JButton("Reset");
                b2.addActionListener(this);
                b3=new JButton("Back");
                b3.addActionListener(this);
                t1=new JTextField(20);
                t2=new JTextField(20);
                ImageIcon ii=new ImageIcon("1.JPG");
                setLayout(new GridLayout(5,1));
                JPanel p;
                p=new JPanel(); p.add(l1); p.add(t1); add(p);
                p=new JPanel(); p.add(b1); add(p);
```

```
                p=new JPanel(); p.add(b2); add(p);
                p=new JPanel(); p.add(b3); add(p);
                p=new JPanel(); p.add(l2); p.add(t2); add(p);
                setSize(600,600);
                setTitle("My window");
                setBackground(Color.RED);
                setVisible(true);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         }
public void actionPerformed(ActionEvent ae)
        {
                String s=ae.getActionCommand();
                if(s.equals("Reset"))       {
                        t1.setText("");
                        t2.setText("");       }
                else if(s.equals("Back")){
                        dispose();
                        new EmployeeScreen();
                                }
                else if(s.equals("Search"))     {
                String a=t1.getText();
                System.out.println(a);
                try {
                        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
System.out.println("Connecting to the database111...");
                        Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system","123");

        Statement statement = connection.createStatement();
        ResultSet res=statement.executeQuery("select * from emp_add where emp_no ='"+a+"'");
                        System.out.println("Connecting to the database.222....");

                        while(res.next()){
                                s1=res.getString(1);
                                if(s1.equals(a))      {
                                        p1=true;
                                s7=res.getInt(7);
                                }
                        }
                statement.close();
                connection.close();
                }
                catch (Exception ex)
                {
                        System.out.println("The exception raised is:" + ex);
                }

                 if(p1)
                 {
                         t2.setText(String.valueOf(s7));


                }
                 else
                   {
                         JOptionPane.showMessageDialog(this,"  Invalid emp_no  ");
```

```
                }
            }
        }
}
```

SEARCH SCREEN PAGE

```
package com.ibm;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public   class  SearchScreen  extends JFrame implements ActionListener
        {
                private static final long serialVersionUID = 1L;
        JLabel  a1,a2,a3,a4,a5,a6,a7,a8,a9;
        JTextField t1,t2,t3,t4,t5,t6,t7,t8;
        JButton b1;
         Boolean r1;
public SearchScreen (String s1,String s2 , String s3, String s4, String s5,String s6,int s7,Strings8)    {
        a1 = new JLabel("Employee  Detail Screen :");
        a2=new JLabel(" Employee No ");
        a3=new JLabel("  Employee Name ");
        a4=new JLabel("  DOB  ");
        a5=new JLabel("  Martial Status ");
        a6=new JLabel("  Nationality ");
        a7=new JLabel("  Designation  ");
        a8=new JLabel(" Salary ");
        a9=new JLabel(" Phone No");
        t1 = new JTextField(20);
        t2 = new JTextField(20);
        t3 = new JTextField(20);
        t4 = new JTextField(20);
        t5 = new JTextField(20);
        t6 = new JTextField(20);
        t7 = new JTextField(20);
        t8 = new JTextField(20);
        b1 = new JButton("Back");
        b1.addActionListener(this);
        setLayout(new GridLayout(10,1));
        JPanel p;
        p = new JPanel(); p.add(a1); add(p);
        p = new JPanel(); p.add(a2); p.add(t1); add(p);
        p = new JPanel(); p.add(a3); p.add(t2); add(p);
        p = new JPanel(); p.add(a4); p.add(t3); add(p);
        p = new JPanel(); p.add(a5); p.add(t4); add(p);
        p = new JPanel(); p.add(a6); p.add(t5); add(p);
        p = new JPanel(); p.add(a7); p.add(t6); add(p);
```

```
p = new JPanel(); p.add(a8); p.add(t7); add(p);
p = new JPanel(); p.add(a9); p.add(t8); add(p);
p = new JPanel(); p.add(b1); add(p);
System.out.println(s1+s2+s3+s4+s5+s6+s7+s8);
t1.setText(s1);
t2.setText(s2);
t3.setText(s3);
t4.setText(s4);
t5.setText(s5);
t6.setText(s6);
t7.setText(String.valueOf(s7));
t8.setText(s8);
setTitle("My window");
setSize(800,800);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent ae)
{
if(ae.getSource()==b1)
{
dispose();
new MyFrame2();
 }
 }
}
```

String

The String is class, which is supports several constructors. This is defined in java.lang.

Method	Function
1. String()	String with no char
2. String(char chars[])	String with char
3. s.length()	Count the no of characters in a string
4. compareTo()	
Value	
i) Less than zero	The left hand side string is less than Right hand side
ii) Greater than zero	The left hand side string is greater than Right hand side
iii) zero	The left hand side string and Right hand side string are equal.

5. compareToIgnoreCase() without case.
6. i) equals() : it return Boolean value true/false (with case)
   ii) equalsIgnoreCase() : it return Boolean value true/false (without case)

Q. What is different between = = and equals()?
Ans: = = An operator compares the references of the string objects. It does not compare the contents of the objects. In equals() method compares the contents.
e.g String s1="Hello"; String s2=new String("Hello"); in = =operators if(s1= =s2) -→ False
String s1="Hello"; String s2=new String("Hello"); in equals() method if(s1.equal(s2)) -→True
String s1="Hello"; String s2="Hello" in = =operators if(s1= =s2) -→ True

When an object is created by JVM, it returns the memory address of the object as a hexadecimal number, which is called **object reference.** When another object created, a new reference number is allocated to it. Every object will have a unique reference.

7. Searching String
   i) indexOf() Searching for the first occurrence of a character or Sub-string.
   e.g int indexOf(int ch) , int indexOf(String str)
   ii) lastIndexOf() Searching for the last occurrence of a character or Sub-string.

e.g int lastindexOf(int ch) ,int lastindexOf(String str)

8. Modifying a string:
i) substring(int startInded) : Here startIndex specifies the index at which the substring will begin. This form returns a copy of the substring that being at startInndex and runs to the end of the invoking string.
ii) substring(int startInded ,int endIndex) : startIndex specifies the beginning index , and endIndex specifies the stopping point. The string returned contains all the characters from the beginning index, up to, but not including, the ending index.

iii) concat(): add two string. E.g: String s1="Rabi"; String s2=s1.concat("Shaw");
other form String s1="Rabi"; String s2= s1 + "Shaw";
iv) replace (char c1, char c2) : This method replaces all occurrences of one character in the invoking string with another character. E.g String s="Rabi".replace('b','v'); output=> Ravi ;
E.g String s="Rabbi".replace('b','v'); output=> Ravvi ;
v) toUpperCase() & toLowerCase() : covert Upper case & lower case.
E.g String s="Rabi"; String s1=s.toUpperCase(); output=> RABI
E.g String s="Rabi"; String s1=s.toLowerCase(); ouput=>rabi
9.
i) startsWith(Strinh s) : check String starts with sub string or not, and returns a Boolean value.
ii) endWith(Strinh s) : check String end with sub string or not, and returns a Boolean value.

10. String.trim() : Remove free space Starts and End position. e.g: " Hello Baby "
Remove space before Hello and after Baby output: Hello Baby

Q Different between "= =" operator and equals method in the context of string object:
Ans: "= = "operator compares the references of the string object. It does not compare the contents of the objects. "equals()" method compare the contents. While comparing the string, equals() method should be used as it yields the correct result.

Q. Difference between String and StringBuffer classes?
Ans:1. String class objects are immutable and hence their contents cannot be modified.
StringBuffer class objects are mutable and hence their contents can be modified.
2. The methods that directly manipulate data of the object are not available in String class.
Such method are available in StringBuffer

Example:

```
class String_Demo
{
        public static void main(String args[])
        {
                String s1="A book on java";
                String s2="A book on java";
                String s3=new String("A book on java");
                String s4=new String("I like it");
                char ch[]={'R','A','B','I',' ','S','H','A','W'};
                String s5=new String(ch);
                String s6="     Ravi   Sir    ";


                //Display

                System.out.println(".....Display.....");
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
                System.out.println(s4);
                System.out.println(s5);
```

```
//Find length

System.out.println(".....Find length.....");
System.out.println("Length of string s1::"+s1.length());

//Concatenate two string

System.out.println(".....Concatenate.....");
System.out.println(s1+" "+s2);
System.out.println(s1.concat(s2));

//equals or not
System.out.println(".....Equals cheack.....");
if(s1.equals(s2))
System.out.println("==");
else
System.out.println(" !=");

if(s1.equals(s3))
System.out.println("==");
else
System.out.println(" !=");

if(s1==s2)
System.out.println("==");
else
System.out.println(" !=");

if(s1==s3)
System.out.println("==");
else
System.out.println(" !=");


System.out.println(".....Sub-String.....");
String p=s4.substring(0,7);// 0th to 6th place
String q=s5.substring(5,9);
System.out.println(p+q);


System.out.println(".....Upper_case.....");
System.out.println("upper s1="+s1.toUpperCase());

System.out.println(".....Lower_case.....");
System.out.println("Lower s1="+s1.toLowerCase());

System.out.println(".....check Starts position.....");

boolean x=s1.startsWith("A");
if(x)
System.out.println("YES");
else
System.out.println("No");
boolean y=s5.startsWith("RABI");
if(y)
System.out.println("YES");
else
System.out.println("No");
System.out.println(".....Remove Starts and End Free space.....");
System.out.println(s6);
```

```
                System.out.println(s6.trim());
        }
}
```

Example:

```
class StringReplace
{
        public static void main(String args[])
        {
                String org="This is a test.This is, too.";
                String search ="is";
                String sub="was";
                String result="";
                int i;
                do
                {
                        System.out.println(org);
                        i=org.indexOf(search);
                        if(i!=0)
                        {
                                result=org.substring(0,i);
                                result= result + sub;
                                result= result+ org.substring(i +search.length());
                                org= result;
                        }
                }while(i!=0);
        }
}
```

System. in: This is represents InputStream object, which by default represents standard input device, i.e keyboard.

System. out: This is represents PrintStream object, which by default represents standard input device, i.e monitor.

i) Connect the keyboard(System.in) to an input stream object, using **InputStreamReader**.
ii) Connect the **InputStreamReader** to **BufferedReader** , which is another input tupe of stream. Using BufferedReader .Syntex:

```
BufferedReader  br=new  BufferedReader(new InputStreamReader(System.in));
```

1) Accepting a Single character from keybord:
   I. Create a BufferedReader object (br).
   II. Read a character from the keyboard using **read()** method.
   III. char ch=(char) br.read();
   (char) because br.read() return a integer value(ASCII) for character. int data type convert char using type casting (char);
   Note: This statement create a problem, which is solve by char ch=br.readLine().charAt(0);

2) Accepting a String from keybord:
String s=br.readLine(); this method accepts a string and return a string. casting is not needed since readLine() method.
But give a runtime error: throws IOException.

3) Accepting integer value from keybord:
   I. First, we should accept the integer number from the keyboard as a string. Using readLine()
   II. Second string convert into integer.
   Syntex:
   int n=Integer.parseInt(br.readLine());

4) Accepting Float ,Double , byte,short ,long and Boolean value from keybord:

Similar: Float n=Float.parseFloat(br.readLine());
double n=Double.parseDouble(br.readLine());
byte n= byte.parsebyte(br.readLine());
short n= Short.parseShort(br.readLine());
long n=Long.parseLong(br.readLine());
boolean n=Boolean.parseBoolean(br.readLine());
double n=Double.parse Double(br.readLine());

5) **Input accepting in a line:**

```
String str;
StringTokenize st=new StringTokenize(str, ",");
        Or
StringTokenize st=new StringTokenize(str, " ");

String token = st.nextToken();
```

6) StringBuffer: it a class. And it's method is:

I. **sb.append(x)**
II. **sb.insert(int i, x)**
III. **sb.reverse()**

x= int ,char, float, double, String etc.

```
import java.util.*;
class StringTokenizerDemo
{
public static void main(String ar[])
{
String str="she is good girl";
StringTokenizer st=new StringTokenizer(str," ");
System.out.println("the tokens are:");
while(st.hasMoreTokens())
{
String one=st.nextToken();
System.out.println(one);
}
}
}
```

LAYOUT MANAGERS

Q. What is Layout Managers?
Ans: A layout managers is a class that is useful to arrange components in a particular manner in a frame or container.

Q. Type of Layout managers:
1. FlowLayout 2.BorderLayout 3.CardLayout 4.GridLayout 5.GridBagLayout 6.BoxLayout

Q ***FlowLayout*** : it is useful to arrange the components in a line one after the other. When a line is filled with components, they are automatically placed in the next line. This is the default Layout in applets and panels.

To create FlowLayout, Following way:

1. FlowLayout obj=new FlowLayout();
   By default, the gap between components will be 5 pixels and the components are centered in the first line.

2. FlowLayout obj=new FlowLayout(int alignment);
   The alignment of components can be specified. We can use FlowLayout.LEFT. , FlowLayout.RIGHT FlowLayout.CENTER

3. FlowLayout obj=new FlowLayout(int alignment, int hgap , int vgap);

The hgap and vgap space between components, hgap represent horizontal and vgap represent vertical gap in pixel.

Using JFrame:

```
import java.awt.*;
import javax.swing.*;
 class FlowLayout_demo extends JFrame
{
        Button b1;
        Label l1,l2;
        TextField t1,t2,t3;

        FlowLayout_demo()
        {

                Container c= getContentPane();
                FlowLayout obj=new FlowLayout();
                c.setLayout(obj);

                l1=new Label("Enter value for a");
                t1=new TextField();
                l2=new Label("Enter value for b");
                t2=new TextField();
                b1=new Button("Button");
                t3=new TextField();

                c.add(l1);
                c.add(t1);
                c.add(l2);
                c.add(t2);
                c.add(b1);
                c.add(t3);
        }

public static void main(String args[])
{
FlowLayout_demo m=new FlowLayout_demo();
m.setSize(500,400);
m.setVisible(true);
m.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Using Applet:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<html>
<applet code="FlowLayout_demoA" width=300
height=200>
</applet>
</html>
*/
public class FlowLayout_demoA extends Applet
implements ItemListener
{
        Button b1;
        Label l1,l2;
        TextField t1,t2,t3;

        public void init()
        {
                setLayout(new FlowLayout());
l1=new Label("Enter value for a");
                t1=new TextField();
l2=new Label("Enter value for b");
                t2=new TextField();
                b1=new Button("Button");
                t3=new TextField();
                add(l1);
                add(t1);
                add(l2);
                add(t2);
                add(b1);
                add(t3);
        }
public void itemStateChanged(ItemEvent ie)
        {
                repaint();
        }
}
```

```
import java.awt.*;
import javax.swing.*;
class BorderLayout_demo extends JFrame
{
    JButton b1;
    JLabel l1,l2;
    JTextField t1,t2,t3;
    BorderLayout_demo()
    {

            Container c= getContentPane();
            BorderLayout obj=new BorderLayout();
            c.setLayout(obj);


            l2=new JLabel("Enter value for b");
            t2=new JTextField(2);
            b1=new JButton("Button");
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class CardLayout_demo extends JFrame implements
ActionListener
{
    JButton b1;

    JTextField t1,t2,t3;
    Container c;
    CardLayout obj;
    CardLayout_demo()
    {
            c= getContentPane();
            obj=new CardLayout(40,10);
            c.setLayout(obj);

            t1=new JTextField(5);
```

```
                t3=new JTextField(2);

                c.add("East",l2);
                c.add("South",t2);
                c.add("Center",b1);
                c.add("West",t3);
        }

        public static void main(String args[])
        {
                BorderLayout_demo m=new
BorderLayout_demo();
                m.setSize(500,400);
                m.setVisible(true);

        m.setDefaultCloseOperation(JFrame.EXIT_ON_
CLOSE);
        }
}
```

```
                t2=new JTextField(5);
                b1=new JButton("Button");
                t3=new JTextField(5);

                c.add("First Card",t1);
                c.add("Second Card",t2);
                c.add("Third Card",b1);
                c.add("Fourth Card",t3);

                t1.addActionListener(this);

                t2.addActionListener(this);
                b1.addActionListener(this);
                t3.addActionListener(this);
        }

        public void actionPerformed(ActionEvent ae)
        {

        //when a clicked show the next card
        obj.next(c);
        /* To show a particular card
        card.next(c,"First Card"); */
        }
public static void main(String args[])
        {
        CardLayout_demo m=new CardLayout_demo();
                m.setSize(500,400);
                m.setVisible(true);

        m.setDefaultCloseOperation(JFrame.EXIT_ON_
CLOSE);
        }
}
```

```
import java.awt.*;
import javax.swing.*;
 class GridLayout_demo extends JFrame
{
        Button b1;
        Label l1,l2;
        TextField t1,t2,t3;

        GridLayout_demo()
        {

                Container c= getContentPane();

                GridLayout grid=new GridLayout(3,2,20,20);
                c.setLayout(grid);

                l1=new Label("Enter value for a");
                t1=new TextField();
                l2=new Label("Enter value for b");
                t2=new TextField();
                b1=new Button("Button");
                t3=new TextField();
```

```
                c.add(l1);
                c.add(t1);
                c.add(l2);
                c.add(t2);
                c.add(b1);
                c.add(t3);
        }

        public static void main(String args[])
        {
                GridLayout_demo m=new GridLayout_demo();
                m.setSize(500,400);
                m.setTitle("Flow Layout");
                m.setVisible(true);
                m.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

FILES

Q. Write a program write and read a files.

```
import java.io.*;
class createFile
{
public static void main(String args[])throws IOException
{
DataInputStream dis=new DataInputStream(System.in);
FileOutputStream fout=new FileOutputStream(args[0], true);
BufferedOutputStream bout=new BufferedOutputStream(fout,1024);
System.out.println("Enter the text(# at the end):");
char ch;
while((ch=(char) dis.read())!='#')
bout.write(ch);
bout.close();
FileInputStream fin=new FileInputStream(args[0]);
System.out.println("File contents::::::::::::::::");
int ch1;
while( (ch1=fin.read() )!=-1)
System.out.print((char)ch1);
fin.close();
}
}
```

Q. Write a program write and read a files

```
import java.io.*;
class createFile
{
public static void main(String args[])throws IOException
{
DataInputStream dis=new DataInputStream(System.in);
FileOutputStream fout=new FileOutputStream("File_name.txt", true);
BufferedOutputStream bout=new BufferedOutputStream(fout,1024);
System.out.println("Enter the text(# at the end):");
char ch;

while((ch=(char) dis.read())!='#')
bout.write(ch);
bout.close();
FileInputStream fin=new FileInputStream("File_name.txt");
System.out.println("File contents::::::::::::::::");
```

```
int ch1;
while( (ch1=fin.read() )!=-1)
System.out.print((char)ch1);
fin.close();
}
}
```

Or

```
import java.io.*;
class FileRead
{
public static void main(String args[])throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the file name");
String name=br.readLine();

FileInputStream fin=null;
try
{
fin=new FileInputStream(name);
}
catch(FileNotFoundException fe)
{
System.out.println("File not found.");
return;
}
BufferedInputStream bin=new BufferedInputStream(fin);
System.out.println("File contents::::::::::::::::");
int ch1;
while( (ch1=bin.read() )!=-1)
System.out.print((char)ch1);
bin.close();
}
}
```

Q. Write a program copy a file from another file.

C & Java

1. In Java comma operator use only in for loop.
2. In java if(Boolean value) use. i.e if(any number) is error because any number is constant number.
3. java does not support goto statement.
4. java.awt package Abstract window Toolkit(awt)

5. Discuss the difference between shift right assignment (>>) and shift right zero fill operator (>>>) with proper example.

Ans: Java supports another shift operator >>> known as zero-fill-right-shift operator. When dealing with positive numbers , there is no difference this operator and the right-shift operator . They both shift zeros into the upper bits of a number . The difference arises when dealing with negative numbers.

Note that negative numbers have the high-order bit set to 1. The right-shift operator preserves the high-order bit as 1. The zero-fill-right-shift operator shifts zeros into all the upper bits, including the high-order bit, thus making a negative number into positive.

Example

```
class bit
{
  public static void main(String ar[] )
            {
              int a=8,b=-8;
              System.out.print("a>>1"+(a>>1));
```

```
                System.out.print("a>>>1"+(a>>>1));

               System.out.print("b>>1"+(b>>1));
               System.out.print("b>>>1"+(b>>>1));
               }
        }
```

Output: a>>1: 4 a>>>1: 4 b>>1: - 4 b>>>1: 2147483644

8. For modulo division: a%b= a- (a/b) *b where a and b both integer.
   Example: - 14%3= - 2
   - 14% -3= - 2
   14% -3= 2

9.

In C	

int a[]={10,20},b[2];
b=a; // This statement not suport
	int a[]={10,20},b[];
	b=a;// This statement support
	// output is same value of a[].

10.

```
class HelloWorld {
   public static void main(String[] args) {                    //    Output:
      System.out.println(Math.sqrt(4));                         //    2
      System.out.println(Math.sqrt(-9));                        //    Nan
      System.out.println(Math.pow(4,3));                        //    64
      System.out.println(Math.pow(3,4));                        //    91
      System.out.println(Math.ceil(2.4));                       //    3
      System.out.println(Math.floor(2.9));                      //    2
      System.out.println(Math.floor(-2.9));                     //   -3
      System.out.println (Math.ceil(-3.7));                     //   -3
      System.out.println(Math.floor(-3.7));                     //   -4
      System.out.println (Math.round(2.4));                     //2
      System.out.println (Math.round(2.9));                     //3
      System.out.println (Math.round(-3.7));                    //-4
      System.out.println(Math.rint(2.4));                       //2
      System.out.println(Math.rint(2.9));                       //3
      System.out.println(Math.round(3.5));                      //4
      System.out.println(Math.round(-3.5));                     //-3
      System.out.println(Math.rint(3.5));                       //4
      System.out.println(Math.rint(-3.5));                      //-4
   }
}
```