//////////core java....

//////what is a class?????????????????

Class is a template for multiple objects with similiar features and it is a blue print for objects. It defines a type of object according to the data.

multiple inheritance and operator overloading is not possible in java....

some misconceptions about java

java is an extension of html. html is a way to describe the structure of the web page, wherease java is a programming language and there are some html extensions for java applets.

javascript is a simpler version of java... javascript is a scripting language used inside web pages. it was first known as livescript made by netscape.

java makes two kinds of programs:

simple applications same as c and c++.

and java applets that are designed to transfer over web pages.

everything in a java program must be inside a class.

java classses are almost same as that of c++ but here we call class functions as methods and data members as fields.

c and c++ distinguishes between definition and declaration but java does not allow for the same.

int i=10; // is a definition

wherease extern int i; // is a declaration

in java no declaration are separate from definition.

there is no unsigned integer in java.

String e = "" /// empty string.

String e = "shivanshu";

String w= e.substring(0,3); // it will start from 0th position and go upto (3-1)th it will leave 3th character.

String e = "shivanshu";

String m= "umang";

String k= e + m ;

a java string is roughly analogous to a char* pointer.

char * greeting = "hello";

e.equals(t);

"hello".equalsIgnoreCase("hello");

"Hello".compareTo("hello");

Reading inputs>

Scanner in = new Scanner(System.in) // standard input stream....

int a = in.nextInt(); // it reads an integer from the keyboard.

String a= in.next(); // it reads a word from the keyboard.

String a = in.nextLine(); // it reads a line from the keyboard.

System is a class in which out method is statically defined that's why there is no need to have any object of that class this method is directly

called using class name System.

Scanner class is not suitable for reading a password from a console since the input is painly visible to anyone, java introduces a Console class specifically

for this purpose. to raad a password use the following code.

```
Console cons = System.console();
String username = cons.readLine("username : ");
char[] password = cons.readPassword("password :");
```

System.out.print(); // it simply prints the string given in the quatation marks.

System.out.printf(); // it prints the string according to the given format in the format specifier.

System.out.println(); // it prints the string with the new line character at the last of the string.

the java control flow are identical to those in c and c++, with few exceptions, as there is no goto statement in java, in place of goto statement there is labelled break statement .

```
program start....
....
..
denial:
System.out.print();
break denial;
```

continue is same as in c and c++.

for each loop

```
int array[10];
for(int e : array)          // e can be any arbitrary name.....
System.out.print(e);
```

int[] i = new int[100]; // this is the way to allocate memory for an array.

there is an even easier way to print all values of an array using the toString method of the arrays class.

System.out.println(Arrays.toString(a));

Array copying.....

int[] luckynum = {1,2,3,4};

int [] copyluckynum = Arrays.copyOf(luckynum, luckynum.length);

to sorting an array....

Arrays.sort(a); // a in an unsorted array.

any java program....

there is a class having name same as file name. It contains a main method which is defined as

public static void main(String[] args)

{

program body.....

}

here main method is public, as it can be called from the outside of this class.

it is static since we dont have any object of that class.

void because it is returning null.

class and object in java.....

```
public class EmployeeTest
{
        public static void main(String[] args)
        {
         Employee[] staff = new Employee[2];              // creating an array..
         staff[0]= new Employee("shivanshu",200);          // with calling every constructor
```

of any class new must be used there.

```
                staff[1]= new Employee("shivam",5454);
                for (Employee e: staff)                    // for each statement using arbitrary
name e....
                System.out.println("name>" + e.get_name());
               }
        }

        class Employee
        {
                public Employee(String n, int m)      // defining constructor there is no need to have
allfields(data members) here..
                {
                        name= n;
                        salary=m;
                }
                public String get_name()
                {
                        return names;
                }
                public int get_salary()
                {
                        return salary;
                }
                private String name;
                private int salary;
                }
        }
```

for static fields and methods .. there is no need to have any object of that class about which we are talking here... static methods and fields can be called directly by using class name with the dot operator(.). These fields and methods are common for the whole class....

final is very interesting word in java....

final class // the class that can not be extend any further . ie none of the class can be derived from that class.

final method // this method can not be overridden in the derived class again....

final field //this field can be initialized only once. it must be guaranteed that the field value has been set after the end of every constructor. afterwards it may not be modified again. System and String are final classes these classes can not be extended.....

Each of these statements has three parts (discussed in detail below):

Declaration: The code set in bold are all variable declarations that associate a variable name with an object type.

Instantiation: The new keyword is a Java operator that creates the object.

Initialization: The new operator is followed by a call to a constructor, which initializes the new object.

```
/* File name : Employee.java */

public class Employee

{

  private String name;

  private String address;

  private int number;

  public Employee(String name, String address, int number)
```

```
{
  System.out.println("Constructing an Employee");
  this.name = name;
  this.address = address;
  this.number = number;
}
public void mailCheck()
{
  System.out.println("Mailing a check to " + this.name
   + " " + this.address);
}
public String toString()
{
  return name + " " + address + " " + number;
}
public String getName()
{
  return name;
}
public String getAddress()
{
  return address;
}
public void setAddress(String newAddress)
{
  address = newAddress;
}
```

```
  public int getNumber()
  {
    return number;
  }
}
```

Now suppose we extend Employee class as follows:

```
/* File name : Salary.java */
public class Salary extends Employee
{
  private double salary; //Annual salary
  public Salary(String name, String address, int number, double salary)
  {
     super(name, address, number);
     setSalary(salary);
  }
  public void mailCheck()
  {
     System.out.println("Within mailCheck of Salary class ");
     System.out.println("Mailing check to " + getName()
     + " with salary " + salary);
  }
  public double getSalary()
  {
     return salary;
  }
  public void setSalary(double newSalary)
```

```
  {
     if(newSalary >= 0.0)
     {
        salary = newSalary;
     }
  }
  public double computePay()
  {
    System.out.println("Computing salary pay for " + getName());
    return salary/52;
  }
}
```

Now you study the following program carefully and try to determine its output:

```
/* File name : VirtualDemo.java */
public class VirtualDemo
{
  public static void main(String [] args)
  {
    Salary s = new Salary("Mohd Mohtashim", "Ambehta,  UP", 3, 3600.00);
    Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
    System.out.println("Call mailCheck using Salary reference --");
    s.mailCheck();
    System.out.println("\n Call mailCheck using Employee reference--");
    e.mailCheck();
   }
}
```

This would produce following result:

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

Mailing check to John Adams with salary 2400.0

Here we instantiate two Salary objects . one using a Salary reference s, and the other using an Employee reference e.

While invoking s.mailCheck() the compiler sees mailCheck() in the Salary class at compile time, and the JVM invokes mailCheck() in the Salary class at run time.

Invoking mailCheck() on e is quite different because e is an Employee reference. When the compiler sees e.mailCheck(), the compiler sees the mailCheck() method in the Employee class.

Here, at compile time, the compiler used mailCheck() in Employee to validate this statement. At run time, however, the JVM invokes mailCheck() in the Salary class.

This behavior is referred to as virtual method invocation, and the methods are referred to as virtual methods. All methods in Java behave in this manner,

whereby an overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

use of elipsis in java.....

```
import java.io.*;

class Employee
```

```
{
   public static void main(String[] args)
   {
      System.out.println("hey!");
      Manager e = new Manager();
      e.get_print(5.5,10.4,3.45,67);
   }
}
class Manager
{
   public Manager()
   {

   }
   public void get_print(double ... num)
   {
      for(double e: num)
      System.out.println("num >" + e);
   }
}
```

Example: declaring a Player object

Player player; // declares object

Declaring an object does not create an object. It simply sets up a named location in memory that stores a reference (address) to that object.

player = new Player(); // creates object

Declaration and creation can be done all in one step. But remember that there are multiple things going on:

Player player = new Player(); // declares and creates object

use of this in java ..... // in java this is not a pointer as it is in c++ .....

```
import java.io.*;

class Employee
{
   public static void main(String[] args)
   {
      System.out.println("hey!");
      Manager M= new Manager("shivanshu",1000);
      String n= M.get_name();
      System.out.println("name >" + n);
   }
}

class Manager
{
   public Manager(String n, int m)        // this is an error as this should be first statement in constructor....
   {
      name= n;                        /// this order is possible
      salary= m;                        this("shivam");  name =n;   salary= m ;
      this("shivam");
   }                                // super is same as this but it is used to call the super class
```

```
   public Manager(String n)              //          or base class.....
   {
      name= n;
   }
   private String name;
   private int salary;
   public String get_name()                    //   public String get_name()
   {                                      {
      return name;
   }
                              name="papa";

                                        this.name="mummy";

                                        return name;

                              }
}
public static void main(String args){
     this.toString{}; //compilation error: non static variable this can not be used in static context
 }


// enum in java.....
import java.io.*;


enum Day {
  MONDAY(1) {
    public Day next() { return TUESDAY; }   // each instance provides its implementation to abstract
method
```

```
},
TUESDAY(2) {
  public Day next() { return WEDNESDAY; }
},
WEDNESDAY(3) {
  public Day next() { return THURSDAY; }
},
THURSDAY(4) {
  public Day next() { return FRIDAY; }
},
FRIDAY(5) {
  public Day next() { return SATURDAY; }
},
SATURDAY(6) {
  public Day next() { return SUNDAY; }
},
SUNDAY(7) {
  public Day next() { return MONDAY; }
};

public abstract Day next();

private final int dayNumber;

Day(int dayNumber) {   // constructor
  this.dayNumber = dayNumber;
}
```

```
  int getDayNumber() {
    return dayNumber;
  }
}

class DayTest {
  public static void main(String[] args) {
    for (Day day : Day.values()) {
      System.out.printf("%s (%d), next is %s\n", day, day.getDayNumber(), day.next());
    }
  }
}
```

output:

```
MONDAY (1), next is TUESDAY
TUESDAY (2), next is WEDNESDAY
WEDNESDAY (3), next is THURSDAY
THURSDAY (4), next is FRIDAY
FRIDAY (5), next is SATURDAY
SATURDAY (6), next is SUNDAY
SUNDAY (7), next is MONDAY
```

1.You can specify values of enum constants at the creation time as shown in below example:

```
public enum Currency {PENNY(1), NICKLE(5), DIME(10), QUARTER(25)};
```

But for this to work you need to define a member variable and a constructor because PENNY (1) is actually calling a constructor which accepts int value ,

see below example.

```
public enum Currency {
```

```
    PENNY(1), NICKLE(5), DIME(10), QUARTER(25);

    private int value;


    private Currency(int value) {

        this.value = value;

    }

};
```

Constructor of enum in java must be private any other access modifier will result in compilation error. Now to get the value associated with each coin

you can define a public getValue() method inside java enum like any normal java class.Also semi colon in the first line is optional.

2. Enum constants are implicitly static and final and can not be changed once created. For example below code of java enum will result in compilation

error:

Currency.PENNY = Currency.DIME;

The final field EnumExamples.Currency.PENNY cannot be re assigned.

3. Enum in java can be used as an argument on switch statment and with "case:" like int or char primitive type. This feature of java enum makes them

very useful for switch operations. Let’s see an example of how to use java enum inside switch statement:

```
  Currency usCoin = Currency.DIME;

   switch (usCoin) {

        case PENNY:

            System.out.println("Penny coin");

            break;

        case NICKLE:

            System.out.println("Nickle coin");

            break;
```

```
        case DIME:
            System.out.println("Dime coin");
            break;
        case QUARTER:
            System.out.println("Quarter coin");
    }
```

////////////////java does not have abstract fields....

An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

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

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {
  // declare fields
  // declare non-abstract methods
  abstract void draw();
}
```

//////////////////abstract methods cannot have a body

An Abstract method does not have a method definition.

```
            abstract String bark();
```

Any class with an abstract method should also be declared abstract.

```
        abstract class Animal{
            abstract String bark();
        }
```

Classes that extend the abstract class should define the abstract method (or themselves be declared abstract).

```
        class Dog extends Animal{
                public String bark(){
                        return "BOW BOW";
                }
        }
```

I know that one class can implement multiple interfaces but can only extend one abstract class.

Is that only difference between an interface and an abstract class?

```
import java.util.*;


class Employee
{
   public static void main(String[] args)
   {
      Manager_stud e= new Manager_stud();
      e.get_name();
   }
}
abstract class Manager
{
   public Manager()
   {


   }
   abstract void get_name();
//   {
//      System.out.println("papa");
```

```
//    }
}
class Manager_stud extends Manager
{
   public Manager_stud()
   {
      super();
   }
//    void get_name()
//    {
//       System.out.println("shivanshu");
//    }
}
```

error: Manager_stud is not abstract and does not override abstract method get_name() in Manager class Manager_stud extends Manager.

A method which is not abstract i.e. if a methods definition is given in the same class its declared is called concrete.

a class which is not abstract called concrete class.....

a method which is abstract will be a method of a abstract class....

Defining abstract is a way of preventing someone from instantiating a class that is supposed to be extended first.

You can declare two kinds of classes: top-level classes and inner classes.

Top-level classes

You declare a top-level class at the top level as a member of a package.

Each top-level class corresponds to its own java file sporting the same name as the class name.

A top-level class is by definition already top-level, so there is no point in declaring it static; it is an error to do so.

The compiler will detect and report this error.

Creating an instance of a class is sometimes referred to as instantiating the class.

Give the list of Java Object class methods.

clone() - Creates and returns a copy of this object.

equals() - Indicates whether some other object is "equal to" this one.

finalize() - Called by the garbage collector on an object when garbage collection
determines that there are no more references to the object.

getClass() - Returns the runtime class of an object.

hashCode() - Returns a hash code value for the object.

notify() - Wakes up a single thread that is waiting on this object's monitor.

notifyAll() - Wakes up all threads that are waiting on this object's monitor.

toString() - Returns a string representation of the object.

wait() - Causes current thread to wait until another thread invokes the notify() method
or the notifyAll() method for this object.

//////////////////////default construcot in java

```
import java.util.*;


class Employee
{
   public static void main(String[] args)
   {
      System. out.println("shivanshu");
       Manager M= new Manager();
```

```
        Manager.print();
        M.get_name();
        //print();
    }
}
class Manager
{
    public static void print()
    {
        System.out.println("hey!! shivam");
    }
    public void get_name()
    {
        System.out.println("name >" + name);
    }
    private String name;
}
```

We cannot override static methods. Static methods are belogs to class, not belongs to object. Inheritance will not be applicable for class members

By specifying final keyword to the method you can avoid overriding in a subcalss. Similarlly one can use final at class level to prevent creating subclasses.

Default value of a boolean is false.

Here out is an instance of PrintStream. It is a static member variable in System class. This is called standard output stream, connected to console.

//////////////////////////////////java static import.....

////////////////Can interface be final????????

No. We can not instantiate interfaces, so in order to make interfaces useful we must create subclasses. The final keyword makes a class unable to be extended.

The local variables are not initialized to any default values. We should not use local variables with out initialization. Even the java compiler throws error.

Can we initialise uninitialized final variable????????

Yes. We can initialise blank final variable in constructor, only in construtor. The condition here is the final variable should be non-static.

//////////////////if a field is static and final both then//////////////////////

```
class Manager
{
   public Manager()
   {
      //salary=1000;
   }
   public static final int salary=900;                    /// field should be initialized at the same line.
   public static void print()
   {
      System.out.println("hey!! shivam");
   }
   public void get_name()
   {
      System.out.println("name >" + name + "\nsalary >" + salary);
   }
```

```
    private String name;
}
```

//very important//

Yes, we can declare main method as private. It compiles without any errors, but in runtime, it says main method is not public.

////////// use of @Override in a java program....

We use the @Override annotation as part of method declaration. The @Override annotation is used when we want to override methods and want to make sure have overriden the correct methods. As the annotation name we know that there should be the same method signature in the parent class to override. That means using this annotation let us know earlier when we are mistakenly override method that doesn't exist in the base class.

```
import java.util.*;

class Employee
{
    public static void main(String[] args)
    {
       System. out.println("shivanshu");
        Manager_stud M= new Manager_stud();
        M.get_name();
    }
}
class Manager
{
    public static void print()
    {
```

```
        System.out.println("hey!! shivam");
    }
//    public void get_name()
//    {
//        System.out.println("name >" + name);
//    }
    private String name;
}
class Manager_stud extends Manager
{
    @Override                    // method does not override or implement a method from a supertype.
    public void get_name()
    {
        System.out.println("mera beta naam karega!!");
    }
}
```

@Override - Checks that the method is an override. Causes a compile error if the method is not found in one of the parent classes or implemented interfaces.

@Deprecated - Marks the method as obsolete. Causes a compile warning if the method is used.

@SuppressWarnings - Instructs the compiler to suppress the compile time warnings specified in the annotation parameters

//////////////// JAVA INTERFACE CONCEPTS.....

An interface in the Java programming language is an abstract type that is used to specify an interface (in the generic sense of the term) that classes must implement.

Interfaces are declared using the interface keyword, and may only contain method signature and constant declarations (variable declarations that are declared to be both static and final). An interface may never contain method definitions.Interfaces cannot be instantiated, but rather are

implemented. A class that implements an interface must implement all of the methods described in the interface, or be an abstract class.

An interface is a collection of method declarations. ? An interface is a class-like concept.? An interface has no variable declarations or method bodies.

// implements instead of extends

public classname implements interfacename

{

}

// multiple inheritance

public classname implements interfacename1, interfacename2

{}

//inheritance and multiple interface implementation

public classname extends baseclass implements interfacename1,interfacename2

{}

Access modifiers ?

An interface can be public or “friendly” (the default).?

All methods in an interface are default abstract and public. ?

Static, final, private, and protected cannot be used. ?

All variables (“constants”) are public static final by default ?

Private, protected cannot be used.

Four often used Java interfaces ?

Iterator

Cloneable

Serializable

Comparable

byte is the smallest size data type for integers

if u have to convert an integer into byte than due to small range it takes the modulo of the integeral value to the number 256(which is the range of byte).

int a= 257;

byte b= (byte)a;

here b will be 1.

in right shift if the top most bit is 1 then all the top most bit will be replaced by 1 only in case of top mmost bit is 1.

////////// how to allocate space for 2 dimension array

int[][] array = new int[5][];

array[0]= new int[1];

array[1]= new int[2];

array[2]= new int[5];

array[3]= new int[2];

array[4]= new int[6];

//////// passing object as arguments (means u are passing reference to the object any change that u make in the object make the change in the sme object).

import java.util.*;

class Employee

{

public static void main(String[] args)

{

Manager s= new Manager();

Manager m= new Manager();

```
        s.objectchange(m,12,"shivam");

        m.get();

        s.get();

    }

}

class Manager

{

    public void objectchange(Manager m , int n, String s)

    {                                                                    /////////// here u are creating a
new object with new operator..

        m.name=s;       output will be: name > shivam          ////// m= new Manager();      output will be :
name > null

        m.salary= n;                salary>12             ///// m.name=s;                    salary>0

        m= new Manager();           name>null             ///// m.salary=n;                 name >null

    }                           salary>0             salary>0

    public void get()

    {

        System.out.println("name >" + name + "\nsalary >" + salary);

    }

    private int salary;

    private String name;

}
```

////////////////declaring object of a class.....

obtaining object of a class is a two step process. Declaring a variable of a class does not define the object. it is simply a object that can refer to an object.u must acquire a physical, actual copy of the object and assign it to that variable this can be done by using new operator. the new operator dynamically allocates memory for an object and returns a reference to it.

since objects are dynamically allocated by using new operator, you might be wondering how such objects are destroyed and their memory released for reallocation. In some languages, such as c++

dynamically allocated objects must be manually released by use of delete operator. java takes a different approach it handles deallocation for u automatically. the technique that accomplishes this is called garbage collection.it works like this: when no references to an object exist.

finalize() method.....

sometimes an object will need to perform some actions when it is destroyed. for example if an object is holding some non-java resource such as file handling or character font then u might want to make sure these resources are freed before an object is destroyed. for this finalize method is used.

.....BUT.....

it is not called when an object goes out-of-scope ie. u dont know when it will be executed.it must not rely on it for any release.

////////// DISTINCTION BETWEEN CLASS AND OBJECT.....

A class created a new type that can be used to create objects. a class creates a logical framework that defines the relationship between its members.when u declare an object it means u are creating an instance of the class, here class is a logical construct and object has physical reality.

//////////ASSIGNING OBJECT REFERENCE VARIABLES.....

classname object1= new classname();

classname object2= object1;

u might think that object2 is being assigned a reference to a copy of the object refered to by object1. ie u might thnk that object1 and object2 refer to seperate and distinct objects. this would be wrong. instead both refer to the same object. this assignment does not allocate any memory or copy any part of the original object. any changes in one oject make the sme change in second object. Although both refer to same object they are not linked in any other way for example a subsequent assignment to object1 will simply unhook object1 from the original object without affecting the object2, object1 is set to null but object2 is still points to the original object.

//// IN CASE OF OVERLOADING OF A FUNCTION.....

when no matching method is found. however, java can automatically convert an integer into a double this conversion is used to resolve the call.

when u pass an object to a method, the situation changes dramatically becuase object are passd vy what is effectively call-by-reference. Keep in mind that when u pass this reference to a method, the parameter that recieves it, will refer to the same as that refered to by the argument.changes to the object inside the method do affect the object used as an argument.

the object will exist as long as there is a reference to that object anywhere in the program.

//PREFACE IN JAVA.....

preface is both a naming and a visibility control mechanism. you can define classes inside a package that are not accessible by code outside that package.

u can import inbuilt packages to use the classes of those packages for example:

import java.util.*;

import java.io.*;

here java is top level package, util is subordinate package inside the outer

java package and last one is class to be used, * add all public classes of the package.

//////////////////////////u can define your own package.....

SYNTAX TO DEFINE PACKAGE.....

package aman; // aman is a package having public class shiv and default classManager

public class shiv { // u can use the public class in diefferent package and can extend any class to public class in different package.

```
 public void getdone()
 {
    System.out.println("u r in getdone method of class shiv");
 }
```

```
  protected void getprint()
  {
     System.out.println("u r in getprint method of class shiv");
  }
  private int a=10;
  int b=10;
}
class Manager                                   // u can not use Manager class in different package.
{
    public void doing()
    {
       System.out.println("amardeep");
    }
}


////////////////////// SECOND PACKAGE.....
package employeetest;
import aman.*;
public class shiv2
{
    protected void doit()
    {
       System.out.println("i will do it!!");
    }
    public static void main(String[] args)
    {
      shiv E= new shiv();
```

```
    //E.doit();              // the statements that can not be used are commented out here..
    E.getdone();
    //E.getprint();                          // this is protected member can not be used in shiv2 class.
    //E.b=10;                                // this is private member can not be used in shiv2 class.
    Employee k= new Employee();
    k.dooo();
    k.getdone();
    //Manager p= new Manager();                      // can not call Manager as Manager is default class.


  }
}
class Employee extends shiv
{
  public void dooo()
  {
     System.out.println("u r in dooo method..");
                getprint();                          //this can be used here as Employee extends shiv it can use protected member of shiv.
  }
}


//class Man extends Manager                    // this can not be used as Manager has default access in aman.


package employeetest;
import aman.*;
public class shiv2 extends shiv
```

```
{
   protected void doit()
   {
      System.out.println("i will do it!!");
   }
   public static void main(String[] args)
   {
      shiv E= new shiv();
     //E.doit();                                  //super class object can not use sub class method.
     E.getdone();
     //E.getprint();                               // here shiv2 object can call getprint method, but this is not possible
     getprint();                                  // it can not be used here because it will say that non static member can not be called in static manner
            //E.b=10;                               // it will show the same problem only in main method().
     Employee k= new Employee();
     k.dooo();
     k.getdone();
     new shiv2().getprint();                      // here it can be used because protected member of superclass can be called by subclass object.
     //Manager p= new Manager();

   }
}
class Employee extends shiv
{
   public void dooo()
   {
```

```
    System.out.println("u r in dooo method..");
  }
}
```

	Private	No Modifier	Protected	Public
Same class	yes	yes	yes	yes
Same Package subclass	no	yes	yes	yes
same package non subclass	no	yes	yes	yes
different pack subclass	no	no	yes	yes
different pack non subclass	no	no	no	yes

////////////////// EXCEPTION IN JAVA ..... TRY AND CATCH AND THROW AND THROWS AND FINALLY.....

an exception is an abnormal conditon that arises in a code sequence at run time. in other words exception is a run time error.

throwable is at the top of all Exceptions one kind of throwable is exception and other one is error. errors are errors which occured because of lack of memory.for example stack overflow we can overcome such errors in our code so we will not deal with such problems.

finally statement:

if a finally block is associated with a try, the finally block will be executed upon conclusion of try block and in case if catch statement is there finally will execute just after catch block.

```
import java.util.*;
public class trycatch {
  public static void checkerror()
  {
    try
    {
      int a;
```

```
        Scanner in= new Scanner(System.in);
        a= in.nextInt();
        int b = 400/a;
        System.out.println("there is no error in try statement.");
    }
    catch(Exception e)
    {
        System.out.println("error :" + e);
        throw e;
    }
    finally
    {
        System.out.println("you r in finally statement.");
    }
}
public static void main(String[] args)
{
    System.out.println("u r in main method.");
    try
    {
        checkerror();
    }
    catch(Exception e)
    {
        System.out.println("error in main method.");
    }
}
```

```
}
```

OUTPUT:

```
u r in main method.
0
error :java.lang.ArithmeticException: / by zero
you r in finally statement.
error in main method.
```

use of throw and throws.....

```
public class trycatch {
   public static void checkerror() throws NullPointerException
   {
      System.out.println("there is no error in try statement.");
      throw new NullPointerException("demo");
      //// if u change the order of these two statements it will show u and compile time error because println() method will not be of any use there.
   }
   public static void main(String[] args)
   {
      System.out.println("u r in main method.");
      try
      {
         checkerror();
      }
      catch(NullPointerException e)
      {
         System.out.println("error in main method.");
```

```
        }
        finally
        {
            System.out.println("u r in finally method.");
        }
    }
}
```

OUTPUT:

u r in main method.

there is no error in try statement.

error in main method.

u r in finally method.

//////////////////////USE OF MULTIPLE CATCH STATEMENTS.....

```
public class multiplecatches {
    public static void main(String[] args)
    {
        try
        {
            int a= arg.length;
            System.out.println("a = " + a);
            int b= 42/a;
            int c[] = {2};
            c[42]= 43;
        }
        catch (ArithmeticException e)
        {
```

```
        System.out.println("divide by zero :" + e);
      }
      catch (ArrayIndexOutOfBoundsException e)
      {
        System.out.println("Array index out of bound prob :" + e);
      }
      System.out.println("after try catch block.");
   }
}
```

if input is 0

output will be:

a = 0

divide by zero : ArithmeticException

after try catch block.

if input is 10

output will be:

a = 10

Array index out of bound prob :ArrayIndexOutOfBoundsException

after try catch block.

```
public class multiplecatches {
   public static void main(String[] args)
   {
      try
      {
        int a= arg.length;
```

```
        System.out.println("a = " + a);

        int b= 42/a;

        int c[] = {2};

        c[42]= 43;

      }

      catch (Exception e)                          // it will produce compie]le time error as second
catch statement is no longer of any use

      {                                          // first catch can catch all kinds of exceptions because all
other are subclaaes of exception class.

        System.out.println("divide by zero :" + e);

      }

      catch (ArrayIndexOutOfBoundsException e)

      {

        System.out.println("Array index out of bound prob :" + e);

      }

      System.out.println("after try catch block.");

   }

}


//////////////////////////////////////MULTITHREADED
PROGRAMMING.....................................................
```