Java notes copy

Java Notes

Rules:

- Smile
- Regular
- notes- Class Notes, programming notes, Assignment notes

History of Java

James gosling in the year 1991 invented Java. Birth-name of java was green-talk with extension .gt

Later it was named Oak and this brought a law suit for name clash. Later and finally it was named as Java with extension .java

Language

way of communication. Programming lang helps us to communicate between machines and humans.

Programming language examples are java, C#,C++ etc.

Arch of java-

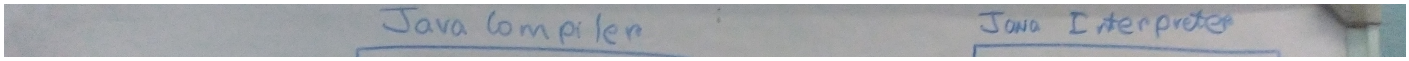
In English language

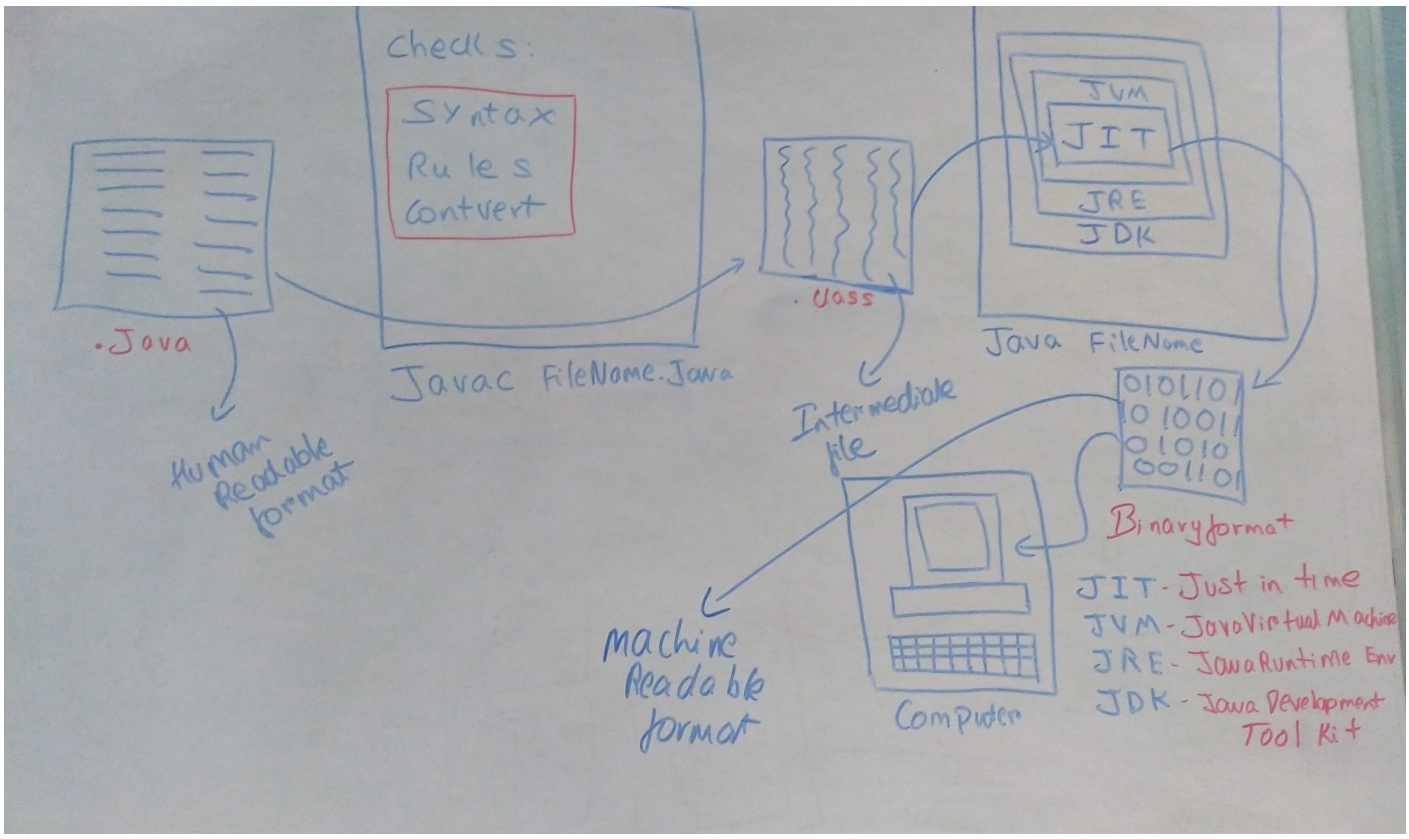
Alphabets -> words -> sentences -> paragraph -> story.

In programming language

Alphabets, Numbers, special char -> tokens -> statements -> program -> s/w.

Source code written in .java file -> this is human readable form of the program.





This is converted into .class file by javac which is a java complier which can invoked from cmd. This complier checks syntax, Rules, Translate. After the program is complied and this will be in an intermediate language which can't be understood neither by human nor machine.

Then the class file is taken by the interpreter to convert it into a machine format ie binary format. This is done by the use of the command called java. The conversation of .class file is done by **JIT**(just in time)- it is sole responsible for conversion for .class file to binary format. Which is inside the **JVM**- It is virtual machine which doesn't exists physically and it is whole responsible to execute the java program. which is inside the **JRE**- environmental setup provides to run the java program. JRE is available on each and every devices. Which is inside the **JDK**-It is kit consists of all the library files and utilities to develop the software.

Here the program is read line by line and executed by the interpreter. Then translate. The java program will be executed from right to left and top to bottom.

why java is platform independent?

Once the code is compiled the .class file can be executed on any devices because each and every electronic device has JRE, hence java is platform independent.

Tokens

Tokens are the smallest unit of a program. In Tokens we have

- Identifiers
- Keywords
- Literals
- Operators
- Separators
- Comments

Identifiers

They are the name given for the java program. They can’t be keywords.

Keywords

They are the per-defined words which has its own meaning. In keywords we have 50 words as follows

abstract assert boolean break byte case catch char class constant continue default do double else enum extends final finally float for goto if implements import instance_of int interface long native new package private protected public return short static strictfp super switch synchronised this throws throw transient try void volatile while.

Literals

They are the value which is used in Java programming language to perform some operation.

In the literals we have

- String literals - "hello","ab"....
- Char literals - 'a','9','d'....
- Numeric literals -
  - Interger - 1,3,4,5....
  - Decimals - 2.4,3.5,1.9....
- Boolean literals - True or False.

Operators

They are the symbols which is used to perform some operations on the operands.

example: 10 + 20

10 & 20 are the operands and + is a operator

10 & 20 are the operands and + is a operator.

Seperators

They are used to separate the give code.

we have the following separators.

- Barces -{}
- Brackets -[]
- Paranthesis -()
- semicolon -;
- comma -,

Comments

They are used to provide additional information. In comments we have

- Single line - // this is a single line comment
- Block line - /* this is block line comment */

Write java program to print hello world

```
class hello{
        public static void main(String [] args){
                System.out.println(“hello world”);
        }
}
```

Output hello world

Should save this file as **hello.java.Filename** and **classname** should be same.

Java types

In java types we have

- class type
- interface
- Enum
- annotation

Class :

Class is a blue print or a template to create Object.

Syntax for compilation : javac filename.java Syntax for Interpretation : java filename

Assignment- Write a java program to print hello Qspiders 3 times.

```
class hello{//class declration
        public static void main(String [] args)
//main method decleration
//main method body{
                System.out.println(“hello Qspider”);
                System.out.println(“hello Qspider”);
                System.out.println(“hello Qspider”);
        }
}
```

Output hello Qspider hello Qspider hello Qspider

Write a program to print integer, decimal, char, Boolean, and string value.

```
class hello{
        public static void main(String [] args){
                System.out.println(20);
                System.out.println(2.56);
                System.out.println('A');
                System.out.println(true);
                System.out.println("hello");
}
```

Output: 20 2.56 A true hello

Write a program to add two number 20+20 in print statement

```
class hello{
        public static void main(String [] args){
                System.out.println(20+20);
                System.out.println(20+" is the value");
                System.out.println("the value is "+(20+20));
                System.out.println("the value is "(1/2));
                System.out.println("the value is "(1/2.0));
        }
}
```

Output 40 20 is the value the value is 2020 the value is 0 the value is 0.5

Conditional statement

if statement

If we need to check for any logical condition then we will go for if condition.

Syntax

if(condition){
statements
....
}

Example

```
class hello{
        public static void main(String [] args){
                System.out.println("****** start *****");
                if(5>4){
                        System.out.println("5 is greater");
                }
                System.out.println("****** stop ****");
        }
}
```

Output

- start* 5 is greater stop *

if-else statement

If true then one block of statement will be executed or another block of statement will be executed.

syntax

if(condition){
true block
}else
{
false block
}

Example

```
class hello{
        public static void main(String [] args){
                System.out.println("****** start *****");
                if(5>4){
                        System.out.println("5 is greater");
                }else
                        System.out.println("5 is not greater");
                }
                System.out.println("****** stop ****");
        }
}
```

Output

- start* 5 is greater stop *

else-if

syntax

if(condition){
  block 1
}else if(condition){
  block 2
}
else{
 block 3
}

Example

```
class hello{
        public static void main(String [] args){
                System.out.println("****** start *****");
                if(5>4){
                        System.out.println("5 is greater");
                }else if(5<1){
                        System.out.println("5 is not greater");
                }else{
                        System.out.println("print nothing");
                }
                System.out.println("****** stop ****");
        }
}
```

Output

- start* 5 is greater stop *

- start > 5 is greater stop

Nested if-else

Syntax

```
if(condition1){
        if(condition2){
                statement
        }else{
                statement
        }else{
                statement
        }
}
```

Components of class

In a class we have 3 members

- Variables or data members - used to store some data.
- Methods or function members - used to perform some operations. It can be static or non-static.
- Constructors - used to initialise some variables. It is always not static.

Variables

Variables is a named memory location which stores some value or a data and it can change n number of times during execution.

In variables we have two types - primitive and non-primitive / Reference type. Primitive - byte, short, int, float, - Integer type Double, long - decimal type char, boolean. Non-primitive - array, String or any class type. Variable type is also called as data type.

In variables we have 3 stages, they are explained below.

Variable declaration.

Syntax

```
        datatype variableName
```

example

```
        int a;
```

Variable Initialisation

Syntax

variableName = Value;

example

a = 10;

Variable utilisation

Example

System.out.println(a);

Variable reinitialisation

Example

```
a = 40;
System.out.println(a);
```

Coping var form one var to another

Example

```
int a = 10;
int b;
b = a;
System.out.println(a+" "+b);
```

Variables can be classified into

- local variable
- global variable

local variable

- Any variable which is declared with in the method is called as local variables .
- The scope of local variable is form the beginning of the method till the end of the method or within the method.
- Local variables can not be classified into static and non-static.
- Local var will not have default values.
- Local var should be initialised before utilisation

Example

```
class Sample{
        public static void main(String [] args){
                int a; // var declaration
                a = 10; // var initilisation
                System.out.println(a) // var utilisation
        }
```

```
}
```

Global Variable

- Any var which is declared outside the method and inside the class is called as global var
- The scope of the global var is from the beginning of the class till the end of the class.
- It can be classified into Static and non-Static
- It will have default values.
- once the global var is declared immediately in the next line we can not initialise or reinitialise.

Example

```
class Tester{
        static int a = 40;
        public staic void main(String [] args){
                x = 80;
                System.out.println(x);
        }
}
```

Output 80

Default values Data type Default values byte 0 short 0 int 0 float 0.0f Double 0.0d long 0 char '\U00000' boolean false

---

Primitives data types

The difference between datatypes size is as followed.

Methods

example:
dinga & dingi honeymoon story for methods

Method is a block of statements which will get executed when ever it is called or invoked.

- without arguments and without return type.
- with arguments and without return type.
- with arguments and with return type.

Method syntax

AccessSpecifier modifier returnType MethodName(arguments)

AccessSpecifiers are of 4 types

- Public
- private
- protected
- default/Package

Modifiers of two types

- static
- non-static

ReturnType are datatypes returned by the method

- void - returns nothing
- int - returns integer

className - returns object of the class

```
class Square{
    static void add(){
        int a = 10;
        int b = 10;
        int c = a + b;
        System.out.println("sum of two numbers is"+c);
 }
 public static void main(String []args){
    System.out.println("---------Method start---------");
    add();
    System.out.println("---------Method ends---------");
  }
}
```

output

Method start

20

Method ends

Program to find multiplication of tow numbers

```
class Square{
        static void multiply(){
```

```
                int a = 10;
                int b = 10;
                int c = 2;
                int mul = a * b * c;
                System.out.println("multiplication of three numbers is"+mul);
        }
        public static void main(String []args){
                System.out.println("---------Method start---------");
                multiply();
                System.out.println("---------Method ends---------");
        }
}
```

```
class Area{
        static void circle(){
                int r = 10;
                final double pi = 3.142;
                double res = pi * r * r;
                System.out.println("Area of circle is "+res);
        }
        public static void main(String []args){
                System.out.println("---------Method start---------");
                circle();
                System.out.println("---------Method ends---------");
        }
}
```

Formula

Triangle, ellipse, sector, circle, Trapezoid, square, Parallelogram, rectangle

assigement

write 8 programs to calculate areas without entering any values without arguments.

Post-increment and pre-increment

post increment means using and then increment.

example

```
i = 0;
i = i++ + i++ + i++ + i++;
```

output - 6

pre increment means increment and then use it.

example

```
i = 0;
i = ++i + ++i + ++i + i + ++i;
```

output - 11

Methods with parameters

example to add two numbers

```
class Demo{
        static void add(int a, int b, int c){//parameters
                int addAns = a + b + c;
                System.out.println("addition of three numbers is"+addAns);
        }
        public static void main(String []args){
                add(2,5,7);//arguments
}
```

example to multiply three numbers by passing arguments

```
class Demo{
        static void Multiply(int a, int b, int c){
                int mulAns = a * b * c;
                System.out.println("Multiplication of three numbers is"+mulAns);
        }
        public static void main(String []args){
                Multiply(2,5,7);
}
program to calculate a area of circle by passing parameters
class Demo{
        static void areaCircle(double r){
                final double pi = 3.14;
                double area = pi * r * r;
                System.out.println("area of circle"+area);
        }
        public static void main(String []args){
                areaCircle(5.6);
}
```

Method with return type specifier

When ever we want to perform some operation and that result is required for some other operation then we should go for method with return type.

Real time example for this type of method

Sanvi joined IBM with the salary 10,000. She worked for one year and she was the best employe for the year and manager gave 20K as bonus. write a program to calculate sanvi's total salary earned for this year.

```
class IBM{
        static int accounts(){
                int msal = 10000;
                int nYears = 12;
                return msal * nYears;
        }
        public static void main(String []args){
                int x = accounts();
                int bonus = 20000;
                int total = x + bonus;
                System.out.println("Total salary is"+total);
        }
}
```

Write a program to calculate area of circle using method with return type

```
class Area{
        static float circle(){
                int r = 5;
                return 3.142 * r * r;
        }
        public static void main(String []args){
                float result = circle();
                System.out.println("Area of circle is"+result);
        }
}
```

Explanation of the above program

JVM starts executing from the main method and the first statement when its starts executing, equal operator will work from right to left the area method will be called now the control goes to area method executes all the statements and return the results

The value which is returned by the method will be stored in x and then print the value.

For loop

when ever we want to perform any operation repeatedly for n number of times then we should go for loop.

syntax

for(inital condition; terminating condition; increment/decrement statement){
    looping statements;
}

Write a program to print hello java for 5 times

```
class Sample{
        public static void main(String [] args){
                        for( int i =0; i <=4;i++){
                System.out.println("hello java");
                        }
        }
}
```

write a program to print from 1 to 10

```
class Sample{
        public static void main(String [] args){
                        for( int i =1; i <=10;i++){
                System.out.println(i);
                        }
        }
}
```

Write a program to print from 10 to 1

```
class Sample{
        public static void main(String [] args){
                        for( int i =10; i >=1;i--){
                System.out.println(i);
                        }
        }
}
```

Static

Any member of the class declared with the keyword static is called static member of the class.

- Static is always associated with the class.
- static is one copy.

- all static members will be stored in static pool area.
- when ever we want to access static member from the class, Then we should use classname.variableName or className.methodName.

Non-static

Any member of the class declare without the keyword static is called non-static member of the class.

- non-static is always associated with the object.
- non-static is multiple copy.
- all non-static members will be stored in heap memory.
- when ever we want to access non-static member of the class, then we should use new keyword to create a new object. refrenceVariable.variableName, refrenceVariable.methodName.

Reference variable

Reference variable is a special type of variable which is used to store object address.

This can hold only two values i.e object address or null.

Syntax for reference variable declaration

//declaration
calssname reference_variable;
//Initialzation
reference_variable = new classname();

```
//example
Smaple s1;
s1 = new Sample();




//program
class Sample{
        void area(){
                final double pi = 3.14;
                int r = 5;
                double res = pi*r*r;
                System.out.println(res);
        }
        public static void main(String []args){
                Sample s1 = new Sample();
                s1.area();
                s1.area();
        }
}
```

Assignment

- write 24 programs to calculate area from non static to static using reference variable.

Write a program to calculate area of circle from non-static to static with method with return type.

```
class Sample{
        double area(){
                final double pi = 3.14;
                int r = 5;
                double res = pi*r*r;
                return res;
        }
        public static void main(String []args){
                Sample s1 = new Sample();
                System.out.println(s1.area());
        }
}


class Sample{
        double area(){
                final double pi = 3.14;
                int r = 5;
                double res = pi*r*r;
                return res;
        }
        public static void main(String []args){
                Sample s1 = new Sample();
                System.out.println(s1);
        }
    }
```

output:
address contaied in s1

how reference variable points to object both points to same class but different instance.

Write a program to calculate area of circle from non static to static through reference variable

```
class Area{
        void circle(){
                int r = 10;
                double a = 3.14*r*r;
                System.out.println(a);
        }
        public static void main(String []args){
        Area cir = new Area();
        cir.circle();
```

```
        }
}
```

Write a program to calculate area of circle with method with return type between class

```
class Area{
        double circle(){
                int r =3;
                return 3.14*r*r;
        }
}
class Caller{
        public static void main(String []args){
                Area kal = new Area();
                System.out.println(kal.circle());
        }
}
```

Write a program to calculate to area of circle from non-static to static between different classes with method with parameters.

```
class Area{
        double circle(int r){
                return 3.14*r*r;
        }
}
class Caller{
        public static void main(String []args){
                Area kal = new Area();
                System.out.println(kal.circle(2));
        }
}
```

Assigment

Write 24 program to calculate the areas from non static to static between classes with return type(8), with parameters(8) and without returning value, with out parameters.

Why we use static and non-static

Real time example

Rethu and rohan joined Qspiders on one fine Tuesday, in first java mock Rethu got the mock rating as 1 and rohan got the mock rating as 2.Rohan was very much fastructed he took the remock on wednesday and he scored the mock rating as 1, Write the program to print the java mock rating of both rethu and rohan.

```
class Qspider{
        int javaMock;
        public static void main(String []args){
                Qspider std1 = new Qspider();
                std1.javaMock = 1;
                System.out.println(std1.javaMock);
                Qspider std2 = new Qspider();
                std2.javaMock = 2;
                std2.javaMock = 1;
                System.out.println(std2.javaMock);
        }
}
```

Variable name: find out static or non-static

- mobileCost
- mobileModelName
- MobileColor
- carCost
- carModelName
- carType(petrol)
- schoolName
- schoolGrade
- schoolStrength
- tvCost
- tvBrand
- tvType
- bikeCost
- bikeBrand
- bikeColor
- courseName
- unvirName
- examCost
- laptopName
- laptopCost
- laptopBrand
- homeName
- homeCost
- homeColor
- empID
- empSal
- empGrade
- clothColor
- clothCost
- clothSize(L,M,XL)

Assignment

Assignment

Red 10 programs to decide when to go for static and non-static

Write a program to initialise employee id and salary for two employees where they are joing for a company called IBM

```
class Compamy{
        static String cName = "IBM";
        int empid;
        int empsalary;
        public static void main(String []args){
                Company e1 = new Company();
                e1.empid = 987787;
                e1.empsalary = 245000.00;
                System.out.println(e1.empid+"  "+e1.empsalary);
                Company e2 = new Company();
                e2.empid = 897888;
                e2.empsalary = 45688.00;
                System.out.println(e2.empid+"  "+e2.empsalary);
        }
}
```

Inheritance

All movie are sucessful because of java.

Bahubali

Apthamitra

Magadeera

Inheriting the properties from one class to another class is called as Inheritance.

It is also called as is a relationship.

In this we have 5 types

- Single level Inheritance
- Multi levelInheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Class Diagram

It is a pictorial representation to reprsent the members of the class.

Single level Inheritance

A sub class inheriting the properties from only one super class is called as Single level Inheritance.

Diagram:page 1

Example for single level Inheritance

```
class Demo{
        int a = 10;
}
class Subclass extends Demo{
        void disp(){
                System.out.println("hello");
        }
}
class Sample{
        public static void main(String []args){
                Subclass e1 = new Subclass();
                System.out.println("Starts");
                e1.disp();
                System.out.println(e1.a);
                System.out.println("Ends");




        }
}
```

Multilevel Inheritance

A sub class inheriting the properties from its super class inturn super class inheriting the properties from its super class is called as multilevel inheritance.

Example:

```
class Tester{
        int y = 10;
}
class Demo extends Tester{
        void add(){
                System.out.println("hey");
        }
```

```
        }
}
class Subclass extends Demo{
        void disp(){
                System.out.println("hello");
        }
}
class Sample{
        public static void main(String []args){
                Demo deo = new Demo();
                Subclass e1 = new Subclass();
                System.out.println("Starts");
                deo.add();
                e1.disp();
                System.out.println(deo.y);
                System.out.println("Ends");




        }
}
```

Hierarchical Inheritance

A multiple sub classes inheriting the propeties from only one super class or common superclass is called as Hierarchical Inheritance

Example:

```
class Tester{
        int y = 10;
}
class Demo extends Tester{
        void add(){
                System.out.println("hey");
        }
}
class Subclass extends Tester{
        void disp(){
                System.out.println("hello");
        }
}
class Sample{
        public static void main(String []args){
                Demo deo = new Demo();
                Subclass e1 = new Subclass();
                System.out.println("Starts");
                deo.add();
                e1.disp();
                System.out.println(deo.y);
                System.out.println("Ends");
```

```
        }
}
```

hybrid Inheritance

It is a combination of single level, multilevel and hirecrical inheritnace.

Multiple inheritance is not possible in java through classes but we can achieve through interface.

This keyword

This keyword is used to point to the correct object, when ever the global varibale and the local variable names are same to differentiate between them we use this keyword

This keyword should to only in the non-static context.

This keyword is also called default reference variable.

example:

```
class Demo{
        int a = 10;
        void disp(){
                int a =20;
                System.out.println(a);
                System.out.println(this.a);
        }
}
```

Output:
20
10

Example:

```
class Student{
        int studID;
        String studName;
        Student(int studID, String studName){
                this.studID = studName;
                this.studName = studName;
        }
        public static void main(String []args){
                Student ram = new Student(0987,"Ram");
                System.out.println("Name "+ram.studName+"\\nID "+ram.studID);
```

```
        }
}
```

Composition

A class having an object of another class is called as composition.

It is also called as a relationship.

Example:

```
class Sample{
        void add(){
                System.out.println("hi");
        }
}
class Demo{
        public static void main(String []args){
                Sample s1 = new Sample();
                s1.add();
        }
}
```

Global and local reference variable

```
class Sam{
        int x = 10;
        Sam s1 = new Sam();//global reference variable
        public static void main(String []args){
                int y = 20;
                Sam s2 = new Sam();//local reference variable
        }
}
```

Pass by value

calling or invoking the method by passing primitive type of data is called as pass by value or call by value.

```
class demo{
        void add( int a, int b){
                System.out.println(a+b);
        }
        public static void main(String []args){
                int y = 10;
                int x = 10;
                add(y,x);
        }
```

```
}
```

Pass by reference

Calling or invoking the method by passing reference variable is called as pass by reference.

Example-1:

```
class Demo{
        int iceCream = 10;
        void add(Demo d1){
                System.out.println(d1.iceCream);
        }
        public static void main(String []args){
                Demo s1 = new Demo();
                System.out.println(s1.iceCream);
                add(s1);
        }
}
```

Example-2:

```
class MainHr{
        int totalEmp = 5000;
        public static void main(String []args){
                System.out.println("******main starts******");
                MainHr hr = new MainHr();
                PuneHr.needcandi(hr);
                BlgHr.needcandi(hr);
                System.out.println("********main ends*******");
        }
}
class PuneHr{
        static void needcandi(MainHr pune){
                pune.totalEmp = 3000;
        }
}
class BlgHr{
        static void needcandi(MainHr blg){
                blg.totalEmp = 1500;
        }
}
```

Method Overloading

Developing multiple methods in the same name but variation in arguments list is called as method overloading.

variation in argument list means

- Variation in datatype
- variation in the length of argument
- variation in order of occurence of the argument

Rules

- The method name should be same
- There should be varitation in arguments list
- There is no ristriction on access specifier, modifer, return type.

Assignment: write 5 programs on method overloading.

Example:

```
class Whatsapp{
  void send(int num){
    System.out.println("Send number "+num);
  }
  void send(String str){
    System.out.println("Send String "+str);
  }
  void send(int num,String str){
    System.out.println("Send number "+num+" and String "+str);
  }
  void send(String str,int num){
    System.out.println("send String "+str+" and number "+num);
  }


  public static void main(String [] dp){
    System.out.println("*******main starts*******");
    Whatsapp chat = new Whatsapp();
    chat.send(10);
    chat.send("Hello");
    chat.send(10,"hello");
    chat.send("hello",10);
    System.out.println("*******main ends*******");


  }
}
```

Example:

```
class Swiggy{
  void order(int qty){
    System.out.println("Order by Quantity "+qty);
```

```
  }
  void order(String name){
    System.out.println("Order by Food name "+name);
  }
  void order(int qty,String name){
    System.out.println("Order by Quantity "+qty+ " and food name "+name);
  }
  void order(String name,int qty){
    System.out.println("Order by name "+name+" and Quantity "+qty);
  }
  public static void main(String []args){
    System.out.println("*****main starts******");
    Swiggy user = new Swiggy();
    user.order(5);
    user.order(5,"pizza");
    user.order("pizza");
    user.order("pizza",5);
    System.out.println("******main ends*****");
  }
}
```

Method overrideing

Developing a method in the sub class in the same name and the signature as in the super class but with the differnt implemention in the subclass is called as method overriding

Rules

- There should be is a relationship (inheritance).
- The method should be non-static.
- The complete signature should be same in the sub-class as in the super class
- when ever we want to provide new implementation then we should go for method overriding.

Example

```
class Kitkat{
        void display(){
                System.out.println("4mp camera");
        }
}
class Lopipop extends Kitkat{
        void display(){
                System.out.println("8mp camera");
        }
}
class Mainclass{
        public static void main(String [] dp){
                Lopipop funcall =  new Lopipop();
                funcall.display();
        }
}
```

Super keyword is used in the case of method overriding along with the subclass implementation if we need super class implementation then we should go for super key word that is super.methodName

```
class WhatsappV1{
	void chat(){
		System.out.println("chat with text");
	}
}
class WhatsappV2 extends WhatsappV1{
	void chat(){
		System.out.println("chat with simley and emoji");
		super.chat();
	}
}
class mainClass{
	public static void main(String [] dp){
		WhatsappV2 funcall = new WhatsappV2();
		funcall.chat();
	}
}
```

Type casting

converting from one class object to another class type is called as class type casting or derived type casrting

UpCasting

converting from sub class object to any of its super class type is called as upcasting.

upcasting can be done both

- Implicitly
- explicitly

DownCasting

converting from super class object to sub class type is called as downcasting.

downcasting can be done only explicitly

Story

Grand childern playing there grandparents role on stage for ethnic competation.

Explains how upcasting and downcasting works

Primitive Type casting

Converting from one primitive data type to another primitive data type is called as Primitive type casting.

In primitive typecasting we have

- Widening

  Converting from smaller primitive data type to any of its bigger primitive datatype is called as Widening.

  Widening can be done both
  - Implicitly
  - Expecitly

Widening(Implicitly) double x = 20;//int to double double y = 16.7;//float to double float z = 40;//int to float

Widening(Expectitly) double x = (double)20;//int to double double y = (double)16.7;//float to double float z = (float)40;//int to float

Narrowing

converting from bigger primitive data type to any of its smaller primitive data type is called as Narrowing.

This should be done expectitly.

```
Narrowing(Expectitly only)
int y = (int) 29.36d;
int z = (int) 59.99f;
byte a = (byte) 31.36f;
long c = (long) 31.37d;
```

Blocks

SIB- Static Initialisation Block

IIB- Instance Initialisation Block

SIB-

```
class Tester{
	static{
		System.out.println("SIB1");
	}
	static{
		System.out.println("SIB2");
	}
	public static void main(String []dp){
		System.out.println("Main starts");
	}
}
```

IIB- Any block which is decleared without the keyword static is called as instance initialisation block.

IIB will get executed when ever an object is created

when we have both constructor and IIB, IIB will get executed first then the constructor.

```
class Demo{
	static{
		System.out.println("SIB1");
	}
	{
		System.out.println("IIB2");
	}
	public static void main(String []dp){
		System.out.println("Main starts");
		new Demo();
	}
}
```

Array

It is a linear data structure which is used to store homogenous type of data.

Array declaration

syntax:

```
datatype[] array_name;

//example
int[] Array;
//or
int Array[];
```

Array Size initilasation

syntax:

```
array_name = new datatype[size];

//example
Array = new int[4]
int[] Array2 = new int[4]
```

Store the value into an array

syntax:

```
array_Name[index] = value;

//example
Array[0] = 10;
Array[1] = 20;
....
```

Array utilization

```
System.out.println(Array[0]);
.....
```

Copying the value from one array to another array

```
int[] abb = arr;
```

To find out length of the value

syntax

```
System.out.println(Array.length);
```

Array declaration and store the value directly

syntax:

```
datatype[] arrayName = {v1,v2,v3};

//example
int[] Array = {20,30,40,50};
```

Assignment

Write 9+9 programs with array with new operator and without new operator.

Tomorrow questions

- [ ] Inheritance
- [ ] Method overloading
- [ ] Method overriding
- [ ] Type casting
- [ ] Pass by refrence

Switch case

It is used for pattern matching based on the char sequence matching,perticular case gets executed.

syntax:

```
switch(charSeq){
      case charseq1 : statement;
                                                                  break;
      case charseq2 : statement;
                                                                  break;
      default : statement;
  }
```

example:

Poloymorphism

An object showing differnt behaviuour at its different stages of its life cycle is called as polymorphism. Poly means many morphism is forms.

In this we have 2 types

- Run time polymorphism

  The method declaration getting binding to its definition at the run time by the JVM based on the object created is called as Run time polymorphism. Since the method declaration getting binded to its definition it can be re-binded hence it is called as dynamic binding. Since the method declaration getting binded to its definition at the run time hence it is called as late binding. Method overriding is an example for run time polymorphism.
- compile time polymorphism

  The method declaration getting binded to its definition at the compile time by the compiler based on the arguments passed is called as compile time polymorphism.

  The method declaration gets binded to its definition at the compile time it self is called as early binding. once the method declaration gets binded to its definition, it can not be re-bibded hence it is called as static binding.

  The case of method overriding even though it is up casted we will get over ridden implementation.

  to achieve run time polymorphism we must have inheritance, upcasting, overriding.

  Through run time polymorphism we can achieve generalisation loose coupling,if any internal enhancement is done

  single point of contact

Example for run time polymorphism

```
class Sample{
        void add(){
                System.out.println("heyyy you");
        }
}
class Demo extends Sample{
        void disp(){
                System.out.println("Demo:disp");
        void add(){
                System.out.println("Demo:add");
        }
}
class MainClass{
        public static void main(String[] dp){
                Sample s1 = new Demo()
                s1.add();
```

```
                s1.disp();
        }
}
```

Polymorphism JVM memory allocation diagram

Example for compile time polymorphism

```
class Whatsapp(){
        void send(int number){
                System.out.println("Send number");
        }
        void send(String msg){
                System.out.println("Send message");
        }
}
class mainClass{
        public static void main(String[] dp){
                Whatsapp s1 = new Whatsapp();
                s1.send(1);
                s1.send("hey");
        }
}
```

Abstract class

Concrete method

Any method which has both declaration and definition is called as concrete method

Example:

void disp(){

}

Concrete class

Any class which has only concrete method is called as concrete class.

Example

class Demo{
    void disp(){

```
    ------
    ------
    -----
    }
}
```

Abstract method

Any method which is declared with keyword abstract is called as abstract method.

Example:

abstract disp();

Abstract class

Any class which is declared with the keyword with abstract is called as abstract class.

```
abstract class Demo{
    abstract disp();
}
```

If a class have an abstract method then the class should be declared as abstract class vice versa is not true.

```
abstract class Sample{
        abstract void test();
}
```

An abstract class can have both abstract method and concrete method.

```
abstract class Sample{
        abstract void test();
        void disp(){
                System.out.println("This is me talking");
}
```

We can also develop only abstract methods in the abstract class.

We can not create object for abstract class and interface.

We can not declare an abstract method as

- static : abstract class can't have object so they can't be called or invoked.
- final: anything which is declared as final it can't be altered

- final: anything which is declared as final it can't be altered.
- private : because private methods can not be inherited.

Abstract class will have constructor.

The class which provides the implementation for the abstract method that sub class is also called as implementation class.

If any of the abstract method is not over ridden in the sub class then the sub class should be declared as abstract class.

Example 1:

```
//abstract class with only abstract method
abstract class Tester{
        abstract void disp();
        abstract void text();
}


//
class Demo extends Tester{
        void disp(){
                System.out.println("Hey");
        }
        void text(){
                System.out.println("Hello");
        }
}


//
class Mainclass{
        public static void main(String[] dp){
                Demo d1 = new Demo();
                d1.disp();
                d1.text();
        }
}
```

Example 2:

```
//abstract class with only abstract method
abstract class Tester{
        abstract void disp();
        abstract void text();
}


//
abstract class Demo extends Tester{
        void disp(){
```

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


//
class Tester2 extends Demo{
        void text(){
                System.out.println("Hello");
        }
}


//
class Mainclass{
        public static void main(String[] dp){
                Demo d1 = new Demo();
                d1.disp();
                d1.text();
        }
}
```

Interface

Interface is a java type. Interface is by default abstract.

Interface is a pure abstract body.

In Interface we have two members

- Variable - by default static and final
- Methods - methods are by default public and abstract.

Java provides a separate keyword called implements to inherit the properties from interface to class

Interface doesn't support constructor.

We can not create object for interface and abstract class.

The class which provides the implementation for the abstract method,it is also called as implementation class.

To inherit the properties from interface to interface we use a keyword called extends.

In class type each and every class extends Object class.

In Interface, it is the super most type, it doesn't extends any class.

Through Interface we can achieve multiple inheritance.

Through Interface we can achieve 100% abstraction.

Examaple 1:

```
//synatx
interface interfaceName{


}


//example
interface Sample{
        int x = 10;
        void disp();
}
Example 2:
interface Tester{
        void add();
        void text();
}
class Sample implements Tester{
        public void add(){
                System.out.println("HIII");
        }
        public void text(){
                System.out.println("Hello");
        }
}
class Mainclass{
        public static void main(String [] dp){
                Sample s1 = new Sample();
                s1.add();
                s1.text();
        }
}
```

Example 3:

```
interface Demo{
        void cool();
        void hot();
}


class Tester implements Demo{
        public void cool(){
                System.out.println("I am cool");
        }
}
```

```
class Bingo extends Tester{
        public void hot(){
                System.out.println("I am hot");
        }
        public static void main(String[] dp){
                Bingo b3 = new Bingo();
                b3.cool();
                b3.hot();
        }
}
```

If we know partial implementation then we should go for abstract class.

If we don't know 100% implementation then we should go for interface.

Example 1:

```
interface Audi{
        void wheels();
        void engine();
        void carBreak();
}
class AudiA4 implements Audi{
        void engine(){
                System.out.println("Engine sound");
        }
        void wheels(){
                System.out.println("Wheels");
        }
        void carBreak(){
                System.out.println("car break");
        }
}
```

Example 2:

```
abstract class Audi{
        abstract void wheels();
        abstract void engine();
        abstract void carBreak();
        void color(){
                System.out.println("Black");
        }
}
class AudiA6 extends Audi{
        public void wheels(){
                System.out.println("wheels");
        }
        public void engine(){
                System.out.println("Engine sound");
        }
        public void carBreak(){
                System.out.println("carBreak");
        }
```

```
}
```

Abstraction

Hiding the complexity of the system and exposing only the required functionality to the end user is called as abstraction.

To achieve abstraction declare all the essential properties in the interface and provide the implementation in the sub class

Declare the reference variable of the interface type and init that reference variable with the implementation class object

This is how we achieve abstraction.

Assignment

Write 10 programs on ppt on abstraction concept

```
interface Animal{
        void noise();
}
class cat implements Animal{
        public void noise(){
                System.out.println("Meow");
        }
}
class dog implements Animal{
        public void noise(){
                System.out.println("Bow Bow");
        }
}
class Snake implements Animal{
        public void noise(){
                System.out.println("Buss tuss");
        }
}
class Stimulator{
        static void anisim(Animal a){
                a.noise();
        }
}
class mainClass{
        public static void main(String[] dp){
                cat c = new cat();
                dog d = new dog();
                Sanke s = new Snake();
                Stimulator.anisim(c);
                Stimulator.anisim(d);
```

```
                Stimulator.anisim(s);
        }
}
```

Access specifier

Access specifier is used to restrict access from one class to another class.

They are 4 types of access specifier

- public
- protected
- default/package level
- private

public:

Any member of a class declared as public is called public member of the class or public access specifier.

It can be accessed

- within the class
- outside the package
- within the package
- anywhere

protected

Any member of the class declared with the keyword protected is called as protected member of the class or protected access specifier.

It can be accessed

- with in the class
- with in the package
- outside the class with is a relationship

default/package level:

There is no specific keyword called default i.e if any member declared without any access specifier it is called as default access specifier.

Private:

Any member of the class declared with the keyword private is called as private member of the class and it can be accessed

- only within the class

  public/default class mainClass{ varibale; method; constructor; } //all three can be any of the 4 access specifier

  public/default class mainClass{ varibale; method; constructor; } //all three can be any of the 4 access specifier

Package

package is the folder structure which is used to store similar kinds of files.

Package statement should be the first statement in any java file.

Java provides a keyword a called import to import the files from one folder to another folder.

Import statement should be the second statement in the java file.

We can have n number of import statements in a single file.

The imported files will be virtually present in the package.

When ever we create the package it should be in the reverse order of the URL.

Example:

www.google.gmail.com

then package name should be

com → gmail → google → www

package com.gmail.google.www

www.kar.nic.gov

gov → nic → kar → www

package gov.nic.kar.www

The class member can have all 4 access specifier

The class can have only 2 access specifier public and default.

```
package com.family.myfamily;


public class father{
	private void atm(){
		System.out.println("ATM of dad's");
	}
	void car(){
		System.out.println("Car of dad's");
	}
	protected void bike(){
		System.out.println("bike of dad's");
	}
	public void cycle(){
		System.out.println("cycle of dad's");
	}
}
```

```
package com.family.myfamily;
public class Son{
	public static void main(String[] dp){
		Father Ramesh = new Father();
		Ramesh.car();
		Ramesh.bike();
		Ramesh.cycle();
	}
}
```

```
package com.family.unclefamily;


import com.family.myfamily.Father;


public class Anuty extends Father{
	public static void main(String[] dp){
		Aunty Latha = new Aunty();
		Latha.bike();
		Latha.cycle();
	}
}
```

Encapsulation

Declare the data member as private and restrict the direct access and provide indirect access through

public services called getter and setters.

Java is by default encapsulated.

We can't declare outside the class.

We can not have print statement outside the method.

Encapsulation is one the OOP's principle in java.

There is no rule that the method name should be get and set, it is an industrial convection to use the method names as get and set.

Get is used to get the value, set is used to set the value.

What is java bean class?

Declare the data member as private and restrict the direct access outside the class and provide the indirect access through public services called as getters and setters is called as java bean class.

Example:

```
class Demo{
        private int a = 10;
        public int getA(){
                return a;
        }
        public void setA(int val){
                a =  val;
        }
}


class MianClass{
        public static void main(String[] dp){
                Demo d1 = new Demo();
                int value = d1.getA();
                d1.set(120);
                S
                System.out.println("Value of A is");
        }
}
```

Assignment

Write 5 programs on real time examples on encapsulation.

Interview

Note:

any class which is declared with the keyword final is called final class.

Example:

```
final class Demo(){
}
```

Final class can not be inherited but we can create the object for the final class.

final method:

any method which is declared with the keyword final is called as final method.

we can inherit final method.

we can overload final method, but we can not override final method.

we can declare constructor as final and it can have all 4 access specifiers.

String class

It is a final class which belongs to java.lang package.

String class is imported to each and every package.

String class is immutable

immutable means it will not change the state of the object

Example:

String s1 = "hey";
s1 = "this";

//now the value is not change from hey to this the object s1 is dereferenced from hey object and this object
//is careated which is referenced by s1.

why string class is immutable

When multiple reference var is pointing to the same object if one the reference var is reinitialised it will deference with the current object and refer to the new object where it will not affect other reference variable this is why it is immutable.

String objects can be created in two ways

String object with new operator.

String object without new operator.

All the string objects will be stored in string pool area under heap memory.

String pool area is further divided into

non-constant pool

constant pool

Any string object which is created with new operator will be stored in non-constant pool area

Any string object which is created without new operator will be stored in constant pool area

String objects with new operator will allow duplicates.

String objects without new operator will not allow duplicates.

```
class Demo{
        public static void main(String[] dp){
                String s1 = "hello";
                String s2 = "hello";
                System.out.println(s1==s2);
                System.out.println(s1.equals(s2));
                System.out.println("**************");
                String s3 = new String("cool");
                String s4 = new String("cool");
                System.out.println(s3==s4);
                System.out.println(s3.equals(s4));
        }
}
```

Object class

object class is the super most class in java

Object class is the super most class in java

Object class belongs to java.lang package.

Each and every package imports java.lang package.

Each and every class extends object class.

Object class is the super most class in java where each and every class inherits object class properties.

In object class we have the following method

- int hashCode()
- boolean equals()
- String toString()
- void notify()
- void notifyAll()
- void wait()
- void finalize()

Java library

java library are the in-built classes and interfaces and we have the following packages.

toString()

It is a non-static non-final method of object class

toString method will be inherited to each and every class

When ever we print a reference variable toString method will be invoked where it returns fully qualified path in the form of string

Fully qualified path means packageName.className@hex-decimal-Number

The signature of toString method is public String toString.

```
class Demo{
        public static void main(String[] dp){
                Demo d1 = new Demo();
                System.out.println(d1);
        }
}
```

output:

PackageName.className@Hex-decimal-number

```
class Demo{
        public String toString(){
                return "The is over ridden implementation";
        }
        public static void main(String[] dp){
                Demo d1 = new Demo();
                System.out.println(d1);
        }
}
```

output:

The is over ridden implementation

HashCode()

It is non-static non-final method of object class.

Each and every class inherits hashCode method

This method is available in each and every class

When ever we invoke the hashCode method it will generate unique int number called hash number based on object address

Signature of hashCode method is public int hashCode()

Example

```
class Demo{
        public static void main(String[] dp){
                Demo d1 = new Demo();
                Demo d2 = new Demo();
                System.out.println(d1.hashCode());
                System.out.println(d2.hashCode());
        }
}
```

//output will be two different integer values which is created base on object address

Equals()

This is non-static non-final method of object class

Equals method will be inherited to each and class

When ever we invoke the equals method it will compare object address

If the both the object address are same it will return true else it will return false

The signature of equals method is public Boolean equals()

Example:

```
class Demo extends Object{
                public static void main(String[] dp){
                        Demo d1 = new Demo();
                        Demo d2 = d1;
                        System.out.println(d1.equals(d2));
                }
}
```

output:

False

Since the address the two reference variable is different

Class and Object

class is a blue print of a template to create Object.

Object is a real time entity which has state and behaviour.

State defines non-static data members, behaviour defines non-static function members.

State defines what data it can hold and behaviour defines the way it can behave.

When ever we create an object we will have both state and behaviour.

all the object address should be stored in reference variable.

multiple objects can be pointed by multiple reference variable.

multiple reference variable can point to single object

multiple reference variable can point to single object

Exception handling

It is an event triggered during the execution of the program which interrupt the execution and stop it abruptly and handling this event using try and catch or throws is called exception handling.

It can be handling in two way

- try and catch
- throws

syntax

```
try{
}
catch(Exceptionclass ref_variable){
}


try{

}
catch(Exceptionclass ref_variable){
}catch(){
}


try{
   try{

      }
       catch(){

      }
}
catch(Exceptionclass ref_variable){
}


try{
   try{
    }
   catch(){

    }
}
finally(){
}


try{
```

```
}
finally{
}
```

- Throwable (java.lang.Throwable)
  - Error(java.lang.Error)
    - OutOfMemoryError
    - StackOverFlowError
  - Exception(java.lang.Exception)
    - RuntimeException(java.lang.Runtime.Exception)
      - ArithmeticException
      - NullPointerException
      - IndexOutOfBoundsException
      - IllegalArgumentException
    - IOException
    - SQLException
    - FileNotFoundException
    - ClassNotFoundException

Example

```
class Sample{
        public static void main(String[] dp){
                System.out.println("Main Starts");
                try{
                        int i = 1/0;
                }
                catch(ArithmeaicException e){
                        System.out.println("handled");
                }
                System.out.println("MainEnds");
        }
}
```

output:

Main starts
handled
Main ends

Example 2

```
class Sample{
        public static void main(String[] dp){
                System.out.println("Main Starts");
                Sample s1 = null
                try{
                        System.out.println(s1.hashCode());
                }
                catch(NullPointerException e){
```

```
                catch(NullPointerException e){
                        System.out.println("handled");
                }
                System.out.println("MainEnds");
        }
}
```

output:

Main starts
handled
Main ends

Example 3

```
class mainClass{
        public static void main(String[] dp){
                System.out.println("*****main starts******");
                int[] arry = {10,20,30};
                try{
                        System.out.println(arry[8]);
                }
                catch(ArrayIndexOutOfBoundException e){
                        System.out.println("Handled");
                }
                System.out.println("*********main ends********");
        }
}
```

For one try block we can have multiple catch blocks.

```
class mainClass{
        public static void main(String[] dp){
                System.out.println("*****main starts******");
                int[] arry = {10,20,30};
                try{
                        System.out.println(arry[8]);// throw new
ArrayIndexOutOfBoundsException(something); this is written internally
                }
                catch(ArithmeticException e){
                        System.out.println("Arithmetic exception");
                }
                catch(ArrayIndexOutOfBoundsException e){
                        System.out.println("Index is large");
                System.out.println("*********main ends********");
        }
}
```

What is Exception handling?

It is an event trigged during the execution of the program which interprets the execution and stops

the execution abruptly or suddenly handling this event by using try and catch or throws is called as exception handling

Exception can be handled in two ways

try and catch

throws

all the abnormal statements should be developed in the try block and addressed them in the catch block

We can not develop any statements between try and catch

for one try block there should be min one catch block

for one try block we can develop multiple catch block but only one catch block will get executed which can address the exception

Once the exception occurs further statements of the try block will not get executed.

Once the exception is addressed if we try to re-address it will throw compile time error.

We can develop try finally block or try catch finally also.

All the exception class belongs to java.lang package.

Exception are of two types:

compile time exception

Run time exception

Compile time Exception: Any exception which is caught at the compile time by the compiler is called as compile time Exception.

It can be addressed in two ways

try and catch

throws

All the compile time exception should be addressed at the compile time itself

Run time Exception: Any exception which is not caught by the compiler and found at the run time is called as run time Exception

All the run time exception should be addressed at the run time itself

Run time exception can be handled in two ways

try and catch

throws

finally block

finally block is the block in exception handling which will get executed mandatorly even though the exception is addressed or not addressed.

Example: Database connection closing statements should be developed in finally method

printStackTrace method

It is a non-static method of throwable class which will be inherited to each and every exception class which will print the complete back trace of System.err

Once the exception occurs further statements of try block will not get executed.

```
class Tester{
    public static void main(String[] dp){
        try{
            int i = 1/0;
            System.out.println(“hii”);
            }
        catch(ArithmeticException e){
            System.out.println(“handled”);
        }
        System.out.println(“Main ends”);
    }
```

output:
handled

Once the exception is addressed we can not readdress.

```
class Tester{
    public static void main(String[] dp){
        try{
            int i = 1/0;
            System.out.println("hii");
            }
```

```
            }
        catch(ArithmeticException e){
            System.out.println("handled");
        }
        catch(ArithmeticException e){
        System.out.println("handled")
        }
        System.out.println(“Main ends”);
    }
```

output
compile time error

Once the exception is handled we can not readdress

```
class Tester{
    public static void main(String[] dp){
        System.out.println("Main starts ");
        try{
            int i = 1/0;
        }
        catch(Exception e){
        System.out.println("handled");
        }
    }
}
```

Finally block

```
    class sample{
        public static void main(String[] dp){
            System.out.println("Main starts");
            try{
                int i = 1/0;
            }catch(Exception e){
                System.out.println("handled");
            }
            finally{
                System.out.println("I am finally block ");
            }
            System.out.println("main ends")
        }
    }
```

Stack unwinding

if an exception occurs in any one of the method the exception will get propogated to the called method and

if the exception is addressed no where then the stack will get destroyed hence it is called as stack unwinding

Example:

```
public class Sample{
        static void disp4(){
            int i = 1/0;
        }
        static void disp3(){
            disp4();
        }
        static void disp2(){
            disp3();
        }
        static void disp1(){
            disp2();
        }


        public static void main(String[] dp){
            System.out.println("main starts");


            try{
                disp1();
            }
            catch(ArithmeticException e){
                e.printStackTrace();
            }
            System.out.println("main ends");
        }
    }
```

output:
Error

How to create own exception

```
package playground;


class Start{

     void call(int n)throws IdontLikeException {
         if(n == 7) {
             System.out.println("good");
         }else{
             throw new IdontLikeException("I don't like "+n+" number Exception");
         }

     }

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

```
        Start obj = new Start();
        try {
            obj.call(77);
        }catch(IdontLikeException e) {
            e.getError();
        }

    }

}


class IdontLikeException extends Exception{
    String err;
    IdontLikeException(String val){
        err = val;
    }
    void getError() {
        System.out.println("Error is "+err);
    }

}
```

File handling

File is nothing but collection of similar kinds of data and performing operations like writing, reading, and updating the file is called file handling.

File is a class which belongs to java.io package which should be imported explicitly.

```
package classProgram;
import java.io.File;

public class FileHandling {
    public static void main(String[] dp) {
        File first = new File("/Users/deepakpatel/Desktop/file");
        if(first.mkdir()) {
            System.out.println("Flie created");
        }else {
            System.out.println("File not created");
        }

        if(first.exists()) {
            System.out.println("File exists");
        }else {
            System.out.println("File does not exists");
        }

        if(first.delete()) {
            System.out.println("file deleted");
        }else {
```

```
            System.out.println("not able to delete the file");
        }

    }
}
```

FileWriter and FileReader
These are the built-in functions which are present in java.io package. Which helps us to read and write the files
In FileWriter class we have a method called write method which helps to write the data into the file
In FileReader class we have read method which helps to read the data from the file
CreateNewFile is non-static method of file class which helps to create the file in the specified directory.

Program to create a directory and create a file write data into the file and read the data from the file

```
package classProgram;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;

public class FileHandling {
     public static void main(String[] dp) throws Exception {
         File f1 = new File("/Users/deepakpatel/Desktop/file");
         if(f1.mkdir()) {
             System.out.println("Floder cerated ");
         }
         File first = new File("/Users/deepakpatel/Desktop/file/info.txt");

         if(first.createNewFile()) {
             System.out.println("File created");
         }else {
             System.out.println("File not created");
         }
         FileWriter fw = new FileWriter(first);
         fw.write("This is Deepak patel");
         fw.flush();

         char[] fileRead = new char[(int)first.length()];
         FileReader fr = new FileReader(first);
         fr.read(fileRead);

         String out = new String(fileRead);

         System.out.println(out);



     }
}
```

Collection

Collection

It is an unified arch which consists of classes and interfaces. In order to overcome the drawback of array we go for collection

It's size is dynamic and can store heterogenous type of data.

All the collection classes belongs to java.util package. Collection has implemented growable data structure which will increase in size by 50%. The default capacity will be 10 and it grows its size with the below formula

Final capacity = current capacity *(3/2)+1

```
package classProgram;
import java.util.ArrayList;


public class Collections {
     public static void main(String[] dp) {
         ArrayList list = new ArrayList();
         list.add(10);
         list.add("String");
         list.add(5.36);
         list.add('a');
         System.out.println(list);
     }


}
```

Output:

```
[10, String, 5.36, a]
```


This is arrayList class methods

add(object e); - adds object to arrayList.

add(int pos, object e)- adds element to the particular location on the arrayList, it doesn't replaces the element it

Pushes the object and adds new object.

addAll(Collection)- adds all the object in the argument passed to the called arrayList at the end of the arrayList.

Example:

```
package classProgram;
import java.util.ArrayList;
public class Collections {
     public static void main(String[] dp) {
         ArrayList list = new ArrayList();
         list.add(10);
         list.add("String");
         list.add(5.36);
         list.add('a');
```

```
        list.add("a");
        System.out.println("list one");
        System.out.println(list);

        ArrayList list2 = new ArrayList();
        list2.add(100);
        list2.add(200);
        list2.add(300);
        System.out.println("list two");

        System.out.println("added list of both ");
        list.addAll(list2);
        System.out.println(list);

        System.out.println("adding in between the");
        list.add(2,500);
        System.out.println(list);
    }
}
```


Output:

```
list one
[10, String, 5.36, a]
list two
added list of both
[10, String, 5.36, a, 100, 200, 300]
adding in between the
[10, String, 500, 5.36, a, 100, 200, 300]
```

addAll(int index,Collection)- adds all the object from collection to called arrayList from the index position.
remove()- this method works with both index and value, this only works for String object.
contains()- this method returns Boolean value by checking if the passed object if present in collection.

Example:

```
package classProgram;
import java.util.ArrayList;

public class Collections {
    public static void main(String[] dp) {
        ArrayList list = new ArrayList();
        list.add("Banglore");
        list.add("Goa");
        list.add("Delhi");
        list.add("Manglore");
        System.out.println("list one");
        System.out.println(list);
        if(list.contains("Banglore")) {
            System.out.println("Banglore is present");
        }
        list.remove(2);
        System.out.println(list);
```

```
        list.remove("Manglore");
        System.out.println(list);

    }

}
```

Output:

```
list one
[Banglore, Goa, Delhi, Manglore]
Banglore is present
[Banglore, Goa, Manglore]
[Banglore, Goa]
```

Note: removeAll() will remove all the duplicates in one ArrayList with respective to another ArrayList.
retainAll() will retain all the duplicates in one ArrayList with respective to another ArrayList.

Example for retainAll()

```
package classProgram;
import java.util.ArrayList;

public class Collections {
    public static void main(String[] dp) {
        ArrayList list = new ArrayList();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        System.out.print("list one->");
        System.out.println(list);

        ArrayList list2 = new ArrayList();
        list2.add(30);
        list2.add(40);
        list2.add(50);
        list2.add(60);
        System.out.println(list);

        list.retainAll(list2);
        System.out.println("list->"+list);

    }

}
```

Output:

```
list one->[10, 20, 30, 40]
[10, 20, 30, 40]
list->[30, 40]
```

Example for removeAll();

```
package classProgram;
import java.util.ArrayList;

public class Collections {
     public static void main(String[] dp) {
         ArrayList list = new ArrayList();
         list.add(10);
         list.add(20);
         list.add(30);
         list.add(40);
         System.out.print("list one->");
         System.out.println(list);

         ArrayList list2 = new ArrayList();
         list2.add(30);
         list2.add(40);
         list2.add(50);
         list2.add(60);
         System.out.println(list);

         list.removeAll(list2);
         System.out.println("list->"+list);

     }

}
```

Output:

```
list one->[10, 20, 30, 40]
[10, 20, 30, 40]
list->[10, 20]
```

Write the program to get one by one element in arrayList

```
package classProgram;
import java.util.ArrayList;

public class Collections {
     public static void main(String[] dp) {
         ArrayList list = new ArrayList();
         list.add(10);
         list.add(20);
         list.add(30);
         list.add(40);
         System.out.println("list one->");
         for(int i = 0;i < list.size();i++) {
             System.out.println(list.get(i));
         }
     }
```

```
}
```

Output:

```
list one->
10
20
30
40
```

To find length
In String- var.length()
In array- var.length
In Collection- var.size()

ArrayList class have 3 constructors which are overloaded i.e
Constructor with no parameters - ArrayList()- it will inti default capacity as 10.
Constructor with int parameter- ArrayList(int initialCapacity)- it will help to inti size with the initial capacity as per the user request.
Constructor with Collection parameter- ArrayList(Collection e)- which helps to copy from one collection object to another collection object.

To sort the objects in collection we have a class called Collections and inside that we have static sort method

```
package classProgram;
import java.util.ArrayList;

public class Collections {
     public static void main(String[] dp) {
         ArrayList list = new ArrayList();
         list.add(30);
         list.add(20);
         list.add(50);
         list.add(40);
         System.out.println("list one->");
         java.util.Collections.sort(list);
         System.out.println(list);
     }

}
```

Output:

```
list one->
[20, 30, 40, 50]
```

Write a program to copy from one collection object to another collection object

```
package classProgram;
```

```
import java.util.ArrayList;

public class Collections {
     public static void main(String[] dp) {
         ArrayList list = new ArrayList();
         list.add(30);
         list.add(20);
         list.add(50);
         list.add(40);
         System.out.println("old ArrayList");
         System.out.println(list);
         ArrayList list2 = new ArrayList(list);
         System.out.println("new ArrayList");
         System.out.println(list2);
     }

}
```

Output:

```
old ArrayList
[30, 20, 50, 40]
new ArrayList
[30, 20, 50, 40]
```

Vector Program Example:

```
package classProgram;
import java.util.Vector;
public class Collections {
     public static void main(String[] dp) {
         Vector vic = new Vector();
         vic.add(30);
         vic.add(20);
         vic.add(50);
         vic.add(40);
         System.out.println("Vector");
         System.out.println(vic);

     }
}
```


Output:

```
Vector
[30, 20, 50, 40]
```


PriorityQueue

```
package classProgram;
```

```
import java.util.PriorityQueue;

public class Collections {
     public static void main(String[] dp) {
         PriorityQueue Q = new PriorityQueue();
         Q.add(30);
         Q.add(20);
         Q.add(50);
         Q.add(40);
         System.out.println("Priority Queue");
         System.out.println(Q);
         System.out.println("Peek function-"+Q.peek());
         System.out.println("Poll function-"+Q.poll());
         System.out.println(Q);
     }

}
```

Output:

```
Priority Queue
[20, 30, 50, 40]
Peek function-20
Poll function-20
[30, 40, 50]
```

LinkedList

```
package classProgram;
import java.util.LinkedList;

public class Collections {
     public static void main(String[] dp) {
         LinkedList Q = new LinkedList();
         Q.add(30);
         Q.add(20);
         Q.add(50);
         Q.add(40);
         System.out.println("Priority Queue");
         System.out.println(Q);
         System.out.println("Peek function-"+Q.peek());
         System.out.println("Poll function-"+Q.poll());
         System.out.println(Q);
     }

}
```

Output:

```
Priority Queue
[30, 20, 50, 40]
Peek function-30
Poll function-30
[20, 50, 40]
```

What is Collections?

What is Collections ?

It is an unified arch which consist of classes and interfaces.

In order to over come the draw back of array we can go for collections.

All the collection class and interface belongs to java.util package which should be imported explicitly.

Collection have implemented a growable data structure with the formula

Capacity = current capacity *(3/2)+1

The default capacity for collection classes is 10.

In the collections all the elements will be stored in the form of the object.

Collection is an interface which had 3 sub interfaces

1. List
    1. List is an interface which extends collection interface.
       When we should go for list type of collection?
       When ever we want to store the elements upon index and allow duplicates then we should go for list type of collections.
    2. Features : size is dynamic
    - It can store heterogenous type of data
    - It is index type of collections
    - It allows duplicate.
    - It allows null.
    - It follows order of insertion.
    - Since it is stored upon index we can get the element in random

List has 3 classes:

- ArrayList
    - It is a class which implements list interface
    - Features
        - It can store heterogenous type of data
        - It is index type of collections
        - It allows duplicate
        - It allows null
        - It follows order of incersion
        - Since it is stored upon index we can get the element in random
        - It increase its size by 50%
        - It is not synchronised.
        - Since it is not synchronised the performance is fast
- Vector
    - It is a class which implements List interface
    - Features
        - It can store heterogenous type of data
        - It is index type of collections
        - It allows duplicate
        - It allows null
        - It follows order of incersion
        - Since it is stored upon index we can get the element in random

        - It increases its by 100%
        - It is synchronised
        - Since it is synchronised the performance is slow
- Linked list
    - It is a class which implements List and Queue interface which has dual property
    - When even we want proper order of insertion then we should go for Linked list
    - Features
        - It can store heterogenous type of data
        - It is index type of collections
        - It allows duplicate
        - It allows null
        - It follows order of incersion
        - Since it is stored upon index we can get the element in random
        - It inherits the properties from both list and queue

2. Queue
    - It is an interface which extents collection interface
    - When will we go for Queue type of collections ?
        - When ever we want to maintain general queue then we will go for queue type
    - Queue interface implements two sub classes
        - Linked List
        - Propriety class
            - It is an class which implements Queue interface, it is pure Queue type
            - features
                - size is dynamic
                - we can store heterogenous type of data
                - It is not indexed type of collections
                - It allows duplicates
                - It doesn't allow null
                - It is auto sorted
                - Since it is not a indexed type of collection we can not fetch the elements upon index and hence we use poll() and peek()
                - poll()- it is a non-static method of propriety queue class which will fetch the top most element of the queue and reduce the size of the queue by 1
                - peek()- it is a non-static method of propriety queue class it will fetch the top most element of the queue and it will not reduce size of the queue by one

3.set

- It is an interface which extends collection interface
- In set interface we have 3 sub classes
    - HashSet
    - LinkedHashSet
    - TressSet
- When we should go for set type of collections?
    - When ever we want to store the elements but avoiding duplicates and without index then we should go for set type of collection

we should go for set type of collection
    - Features of set
        - size is dynamic
        - heterogenous type of data
        - It is not indexed type of collection
        - It doesn't allow duplicates
        - Since it is not indexed type of collection we can not fetch the elements upon index.
- HashSet
    - It is the class which implements set interface
    - Features of HashSet
        - It will not follow order of insertion
        - size is dynamic
        - heterogenous type of data
        - It is not indexed type of collection
        - It doesn't allow duplicates
        - Since it is not indexed type of collection we can not fetch the elements upon index.
- Linked HashSet
    - It is a class which extends hashSet
    - Features
        - It will follow proper order of insertion
        - size is dynamic
        - heterogenous type of data
        - It is not indexed type of collection
        - It doesn't allow duplicates
        - Since it is not indexed type of collection we can not fetch the elements upon index.
- TreeSet
    - It is a class which implements set interface
    - features
        - size is dynamic
        - heterogenous type of data
        - It is not indexed type of collection
        - It doesn't allow duplicates
        - Since it is not indexed type of collection we can not fetch the elements upon index.
        - It is completely auto sorted

For-each

syntax- for(datatype var name: arrayName or reference variable)

example

```
char[] arr = {'a','b','c'};
for(char a1: arr){
    System.out.println(a1);
}
```

Generic type of collection

In collection if we want to achieve generic type of collection we should angular braces - <datatype>

example

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

public class Collections {
     public static void main(String[] dp) {
         List<Integer> Q = new ArrayList<Integer>();
         Q.add(30);
         Q.add(20);
         Q.add(50);
         Q.add(40);
         System.out.println("ArrayList output");
         for(Object obj: Q) {
         System.out.println(obj);

     }
     }
}
```

Wrapper classes
These are the in built classes which belongs to java.lang package for all the 8 primitive datatype we have corresponding classes called wrapper classes i.e

primitive data type	wrapper classes
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
Boolean	Boolean

Boxing
It is a mechanism of converting primitive data type to the corresponding wrapper type classes is called as boxing

unBoxing
Converting from wrapper class object to the corresponding primitive datatype is called as unBoxing.

Example

```
Integer a1 = new Integer(10);//boxing
sop(a1);
int x = a1;//unboxing
sop(x);
```

How the elements are stored in the form of Collection?

When ever we add the element in the collection it will be boxed and then upcasted which is stored in the form of Object

```
ArrayList l1 = new ArrayList();
l1.add(10); // Object o1 = new Integer(10);
l1.add(20); // Object o2 = new Integer(20);
```

Map
It is an interface which belongs to java.util package when ever we want to store up on key and value the we should go for Map.
Map is by default generic
In map interface we have 3 sub classes
HashMap - It will not follow order of insertion
LinkedHashMap - It will follow order of insertion
TreeMap - It is auto sorted based on keys

In map it not allow duplicate keys but it will allow duplicate values.

HashMap

```
package classProgram;
import java.util.HashMap;


public class Collections {
     public static void main(String[] dp) {
         HashMap<String, Integer> h1 = new HashMap<String,Integer>();
         h1.put("dp", 007);
         h1.put("James Bond", 0101);

         System.out.println(h1);
     }

}
```

output

```
{James Bond=65, dp=7}
```

LinkedHashMap

```
package classProgram;
import java.util.HashMap;
import java.util.LinkedHashMap;


public class Collections {
     public static void main(String[] dp) {
         LinkedHashMap<String, Integer> h1 = new LinkedHashMap<String,Integer>();
         h1.put("dp", 007);
         h1.put("James Bond", 0101);
```

```
        System.out.println(h1);
    }

}
```

Output

```
{dp=7, James Bond=65}
```

TreeMap

```
package classProgram;

import java.util.TreeMap;


public class Collections {
    public static void main(String[] dp) {
        TreeMap<String, Integer> h1 = new TreeMap<String,Integer>();
        h1.put("dp",1748);
        h1.put("James Bond",198903);

        System.out.println(h1);
    }

}
```

Output

```
{James Bond=198903, dp=1748}
```

Cursors

ListIterator
Iterator

Helper method
ListIterator - listIterator();
Iterator - iterator();

ListIterator methods
  Object next();
  Boolean hasnext();
  void remove();
  Object pervious();
  Boolean hasprevious();
Iterator methods
  Object next();
  Boolean hasnext();
  void remove();

Example for ListIterator

```
ArrayList l1 = new ArrayList();
l1.add(10);
l1.add(20);
l1.add(30);
l1.add(40);
ListIterator itr = l1.listIterator();
while(itr.hasnext()){
    Object o = itr.next();
    System.out.println(o);
}
while(itr.hasprevious()){
    Object o = itr.next();
    System.out.println(o);
}
```

Example for Iterator

```
ArrayList l1 = new ArrayList();
l1.add(10);
l1.add(20);
l1.add(30);
l1.add(40);
Iterator itr = l1.iterator();
while(itr.hasnext()){
    Object o = itr.next();
    System.out.println(o);
}
```

Thread

Thread is an execution instance which owns it own CPU time and memory.
It is a class which belongs to java.lang package which will be imported by default
sleep()- it is a static method of thread class which will pause the execution of the program for few milli seconds

Example:

```
package tools.integer;

public class Mainclass  {
     public static void main(String[] dp)throws InterruptedException {

         for(int i = 1;i<=10;i++) {

             System.out.println(i);
             Thread.sleep(1000);
         }
     }
```

```
}
```

Runnable:
it is an interface which belongs to java.lang package it is a functional interface which has only one non-static method called run().

Write a program for multithreading

```
package tools.integer;



class Print100 extends Thread{

     public void run(){
         try {
             for(int i = 100; i <= 1000;i = i+100) {
                 System.out.println(i);
                 Thread.sleep(1000);
             }
         } catch (InterruptedException e) {
             // TODO Auto-generated catch block
             e.printStackTrace();
         }
     }
}


class Print10 extends Thread{
     public void run() {
         for(int i = 0; i <= 100; i = i+10) {
             System.out.println(i);
             try {
                 Thread.sleep(1000);
             } catch (InterruptedException e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
             }
         }
     }
}

public class Mainclass  {
     public static void main(String[] dp)throws InterruptedException{
         Print100 p1 = new Print100();
         Print10 p2 = new Print10();

         Thread t1 = new Thread(p1);
         Thread t2 = new Thread(p2);

         t1.start();
         t2.start();
     }
}
```

Wait()
It is a non-static method of object class which will be inherited to each and every class, wait method will make the thread wait till it receives the notification.
Notify()- it is a non-static method of the object class which will be inherited to each and every class it will just notify the thread which is about to access the resource.
NotifyAll()- it is a non-static method of the object class which will be inherited to each and every class, it will notify all the threads which is about to access the resource.

wait and sleep-

wait: it is a non-static method of object class
it will wait till it receives notification
it is used in the case of sync
sleep: it is a static method of thread class
it will pause for few milli seconds and then continue
it can be used in any code

Finialize method

it is a non-static non-final method of object class which will be inherited to each and every class, whenever an object is de-referenced
system.gc() method will invoke finialize method which will do garbage collection i.e it will free up the space for the lost object

What is the difference between final, finally, finialize()
final - it is a keyword
If we declare any member as final we can not redefine or re-init
finally- it is a block
this block will get executed mandatory even though exception is addressed or not
finialize()- it is non-static method of object class
this is used for garbage collection which free up the space for the de- referenced object.

```
package tools.integer;



public class Mainclass {



     public static void main(String[] dp){
         Mainclass obj1 = new Mainclass();
         Mainclass obj2 = new Mainclass();

         obj1 = null;
```

```
        obj2 = null;
        System.gc();
    }
    protected void finalize() {
        System.out.println("worked");
    }

}
```

What is constructors overloading ?
developing multiple constructor with in the class but variation in the argument list is called as constructor overloading

variation in argument list means

variation in data type.

variation in length of the argument.

variation in the order of occurrence of the argument.

```
package tools.integer;



public class Mainclass {

    Mainclass(){
        System.out.println("Constructor without parameter");
    }
    Mainclass(int a){
        System.out.println("Constructor with int paramether");
    }
    Mainclass(String val){
        System.out.println("Constructor with String parameter");

    }

    public static void main(String[] dp){
        Mainclass obj = new Mainclass();
        Mainclass obj1 = new Mainclass(123);
        Mainclass obj2 = new Mainclass("dp");
    }

}
```

This calling

It is used in the case of constructor over loading to call another constructor.
rules:
this calling statement should be the
we can not develop two this calling statement in the single constructor
this calling statement is used in the case of constructor overloading

```
package tools.integer;
```

```
public class Mainclass {

     Mainclass(){
         System.out.println("Constructor without parameter");
     }
     Mainclass(int a){
         this();
         System.out.println("Constructor with int paramether");
     }
     Mainclass(String val){
         this(4);
         System.out.println("Constructor with String parameter");

     }

     public static void main(String[] dp){
         Mainclass obj2 = new Mainclass("dp");
     }

}
```

output

```
Constructor without parameter
Constructor with int paramether
Constructor with String parameter
```

Super calling

this is used in the case of inheritance to call from sub class constructor with the immediate super class constructor.
Rules:
super calling statement should be the first statement inside the constructor
There should be is a relationship

Example:

```
package tools.integer;

class superClass{
     superClass(){
         System.out.println("Constructor of superClass");
     }
}
class subClass extends superClass{
     subClass(){
         System.out.println("Constructor of subClass");
     }
}


public class Mainclass extends subClass {
```

```
    Mainclass(){

        System.out.println("Constructor of Mainclass");

    }

    public static void main(String[] dp){

        Mainclass obj2 = new Mainclass();
    }

}
```

output

```
Constructor of superClass
Constructor of subClass
Constructor of Mainclass
```

Why multiple inheritance is not possible in java through classes?
we can not achieve multiple inheritance through classes
we can not develop two super calling statements in the subclass constructor
Diamond problem
each and every class extends object class all the super classes inheritites object class property and sub most class will have ambiguity or confusion from where should the property must be inherited.

to achieve multiple inheritance we use interface
interface doesn't support constructor, here we may not develop two super calling statements in the sub class constructor
interface doesn't extends any class interface itself so there will be no diamond problem.

```
package tools.integer;

interface first{
    int a = 10;
}
interface second{
    int b = 20;
}



public class Mainclass implements first,second{


    public static void main(String[] dp){

        Mainclass obj = new Mainclass();
        System.out.println(obj.a);
        System.out.println(obj.b);
    }

}
```

Constructor chaining

subclass constructor calling its immediate super class constructor intern super class constructor calling its immediate super class constructor is called as constructor chaining.

dynamic input from user
scanner class belongs to java.util package and it has following non-static methods
String next();
String nextLine();
int nextInt();
byte nextByte();
long nextLong();
float nextFloat();
double nextDouble();
Boolean nextBoolean();

```
package tools.integer;

import java.util.Scanner;



public class Mainclass {


     public static void main(String[] dp){

          Scanner out = new Scanner(System.in);
          System.out.println("Enter your name?");
          System.out.println("Hey, "+out.nextLine());
     }

}
```

Resume points
very goood knowledge on method overloading, inheritance, static and non-static, polymorphism, overriding, encapsulation, collections, exception handling.

1-In how many ways can a main method can be declared?
Main method can be declared in 4 ways
public static void main(String[] dp)
public static void main(String dp[])
public static void main(String... dp)
static public void main(String[] dp)
2- What are the features of java?
It is complied
It is Robust

It is highly secured lang
It is simple and easy to understand
High performance
Multithreaded
It is platform independent

3-why java object oriented programming lang?

because it supports the following concepts
Polymorphism
Abstraction
Encapsulation