* Hardwork is a small distance to success, intelligence is sometimes fails but hardwork never fails *
* Success is not a luck or accident it's a result of planning, commitment, attitude & hard work *
* Talk less work hard *

INDEX

net-informations.com

Name

Standard Section Roll No.

Subject

School / College

S. No.	Date	Title	Page No.	Teacher's Sign
I 1		Different data types in Java	1	
2		What is string concatenation & if else condition.	2	
3		Learn for and while loops	4	
4		What is static array in java	5	
5		What is 2-D array in java	7	
6		What is class & object	8	
7		What is function/Method	9	
8		What is Method overloading	10	
9.		Diff. b/w Local & Global vars	11	
10		String manipulation	11	
11		Static and non-static	13	
12		Wrapper classes	14	
13		Call by value & Call by reference	15	
II 14		Method overriding, Inheritance & Polymorphism.	17	
15		Interface & Multiple Inheritance	18	
16		can we overload main() method	23	
17		Abstraction in java - Interface vs Abstract	20	
18		constructor & this keyword	24	
19		This & super with constructor examples.	25	
		final vs finally vs finalize		

CS

S. No.	Date	Title	Page No.	Teacher's Sign
		Why string is immutable in Java	28	
		Stack & Heap – Memory management in Java	28	
		Singleton design pattern in Java – How to achieve it in selenium	29	
		Collections		
1		Overview of collection framework	31	
2		Arraylist	31	
3		Linkedlist	34	
4		Map Interface	35	
5		How to use HashMap in Java	35	
6		Hash Table	37	
7		ArrayList vs Hashtable	39	
8		How to use HashMap in selenium webdriver.	41	

JAVA

Instal Java

Download JDK jar file
Jdk-8u181-windows-x64 } → Run the jar file → Next → Accept the license Agreement → Next

→ close.

Set Jdk path

My computer → Right click on this PC → Properties → Advanced system settings → Environmental variables → Under system variables: Double click on path → New → Paste jdk\bin path: c:\Program files\Java\jdk1.8.0_181\bin → OK → OK → Apply → OK

check installed or not

open command prompt → Enter: java → Enter → will see the information
Enter: java -version → will see java version details.

① Different data types in java

→ Open eclipse → Create project: file → New → Project → java → Java Project → Next → Project name: JavaTrainingSession → Next → Finish
[src] → New → Package → Name: JavaBasics → Finish
[JavaBasics] → New → class → DataTypes → ☑ public static void main(String[] args) → Finish

```
⇒ public class DataTypes {
    // His this is my first java code - single line comment
    /*His this
      is my first
      java code
      and I'm so happy */
    // main() method - starting/execution point of the program
⇒   public static void main (String[] args) {
        // Primitive data types: int, char, double, boolean
        //int:
        int i=10;
            i=20;
        int j=30
        System.out.println(j);   // shot out → sysо → ctrl+space
        int k=40;
        int l=-1;
        System.out.println(i+j);

        //2. double
        double d=12.33;
        double d1=34.14;
        system.out.println(d1);
        double d2=100;  // 100.00
```

```
// 3. char : only single digit value, should be written within single quotes.
char c='a';
char c1='A';
char c2='1';
char c3='$';

// 4. boolean
boolean b1= true;
boolean b2 = false;

// 5. String : is a class, not a datatype
String s= "Hello world";
String s1= "selenium";
String s2 = "Hi this is my java code";
String s3 = "1000";
String s4 = "12.33";
System.out.println (s2);
}
```

Run: Right click → RunAs → Java application.

o/p:
```
30
-1
50
34.44
Hi this is not java code
```

② What is String concatenation and if else condition

JavaBasics → class → StringConcatenation → public class StringConcatenation{

```
public static void main (String[] args){
  // + : is concateration operator.
  // println - is used to print on the console with a new line
  // print - is used to print on the console.
    int a= 100;
    int b = 200;
    String x= "Hello";
    String y= "world";
    double c = 12.33;
    double d = 10.33;
    system.out.println (a+b);
    sop (x+y);
    sop (a+b+x+y);
    sop (x+y+a+b);
    sop (x+y+(a+b));
    sop(a+b+x+y+a+x+b+y);
    sop(a+b+x+y+a+b);
    sop (c+d);
    sop (x+y+c+d);
```

```
sop(" Hellow word ");
sop(a);
sop("The value of a is: "+a);
sop("The value of b is : "+b);
sop("The addition of a and b is: "+(a+b));

System.out.print("Hello Selenium");
System.out.println("Hello testing");   // Hello seleniumHello testing
}
}
```

o/p:-

```
300
HelloWorld
300HelloWorld
HelloWorld100200
Helloword 300
300HelloWorld100Hello200World
300HelloWorld100200
22.66
HelloWorld12.3310.33
HelloWorld
100
The value of a is: 100
The value of b is : 200
The addition of a and b is: 300
HelloSeleniumHelloTesting.
```

JavaBasics
└ IfElseConcept

```
⇒ public class IfElseConcept {
    public static void main(String[] args) {
    int a = 30;
    int b = 20;
    if (b > a) {
        sop(" b is greater than a ");
    else {
        sop(" a is greater than b");
    }

    // comparison operators:
    // < > <= >= == !=
    int c = 40;
    int d = 50;
    if (c == d) {
        sop(" c and d are equal ");
    } else {
        sop("c and d are not equal");
    }
    // writ a logic to find the highest number
    int a1 = 400;
    int b1 = 500;
    int c1 = 300;
```

```
// Nested if-else
if (a1>b1 & a1>c1) {
        sop( " a is greatest ");
}
else if (b1>c1){
        sop(" b1 is greatest ");
}
else{
        sop ("c1 is greatest ");
}
}
```

O/P:- a is greater than b
c and d are not equal
b1 is greatest.

③ Learn for and while loops in java

⊞ JavaBasics
↳ LoopsConcepts

```
⇒ public class LoopsConcepts{
    public static void main(String[] args){
     //1. while loop:
     // Disadvantage of while loop is it will generates infinite loop if we don't give
     // incremental or decremental part.

     // print 1 to 10
     int i=1; // initialization
     while(i<=10){ // condition
       sop(i);
       i=i+1; // incremental/ decremental.
     }
     sop("* * * * * * * ");
     // j++ means j=j+1

     //2. for loop:
     // print 1 to 10
     for(int j=1; j<=10; j++){ // initialization, condition, updation.
         sop(j);
     }
     sop("* * * * * * * ");
     // -1 > -10 true
     // 1 > 10 false
     // print 10 to 1
     // k-- means k=k-1
     // 10 9 8 7 6 5 4 3 2 1
     for (int k=10; k>=1; i--){
        sop(k);
     }
}
```

JavaBasics
↳ IncrementAndDecrementOperator

```
⇒ public class IncrementAndDecrementOperator{
    public static void main (String[] args){
        // ++
        // --
        int i = 1;
        int j = i++; // post increment
        sop(i); // 2
        sop(j); // 1
        int a = 1;
        int b = ++a; // pre increment
        sop(a); // 2
        sop(b); // 2
        int m = 2
        int n = m--; // post decrement
        sop(m); // 1
        sop(n); // 2
        int p = 2;
        int q = --p; // pre decrement
        sop(p); // 1
        sop(q); // 1
    }
}
```

④ What is static array in java

JavaBasics
↳ ArrayConcept

```
⇒ public class ArrayConcept{
    public static void main(String[] args){
        // Array : To store similar datatype values in a array variable.
        // 1. int array:
        // lowest bound/index = 0
        // Upper bound/index = n-1 (n is size of array)
        // one-dimensional array.
        // Disadvantages of array:
        // 1. size is fixed - static array - To overcome this problem - we use collections
        //    - Array list, Hash Table - Use dynamic array.
        // 2. stores only similar datatypes - To overcome this problem, we use object arr
```

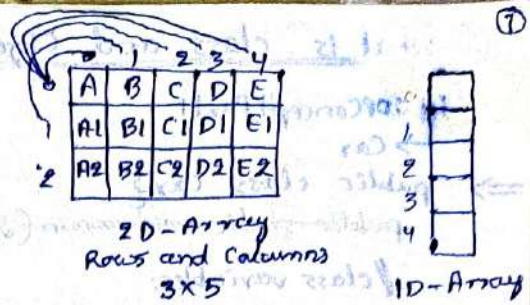
```
int i[] = new int[4];
i[0] = 10;
i[1] = 20;
i[2] = 30;
i[3] = 40;
```

```
sop(i[2]); // 30
sop(i[3]); // 40
// sop(i[4]); // ArrayIndexOutOfBoundsException
```

```
sop(i.length); // size or length of array
//print all values of array: use for loop
for(int j=0; j<i.legth; j++){
    sop(i[j]);
}

//2. double array:
double d[] = new double[3];
  d[0] = 12.33;
  d[1] = 13.44;
  d[2] = 45.55;
sop(d[2]);

//3. char array:
char c[] = new char[3];
  c[0] = 'a';
  c[1] = z;
  c[2] = '$';

//4. boolean array:
boolean b[] = new boolean[2];
  b[0] = true;
  b[1] = false;

//5. String array:
String s[] = new String[3];
  s[0] = "test";
  s[1] = "Hello";
  s[2] = "world";
sop(s.length); // 3
sop(s[1]);     // Hello

//6. Object Array: (Object is a class) - is used to store diff. data type values.
Object ob[] = new Object[6];
  ob[0] = "Tom";
  ob[1] = 25;
  ob[2] = 12.33;
  ob[3] = "1/1/1990";
  ob[4] = 'M';
  ob[5] = "London";
sop(ob[5]); // London
sop(ob.length); // 6
}
}
```

Note: int i[] = new int[4] } Both are same.
int [] i = new int[4]

5 What is Two Dimension (2D) Array in Java



⊞ JavaBasics
↳ TwoDimArray

```
⇒ public class TwoDimArray {
    public static void main(String[] args) {
        String x[][] = new String[3][5];
        sop(x.length); // total no. of rows = 3
        sop(x[0].length); // total no. of columns = 5

        // 1st row
        x[0][0] = "A";
        x[0][1] = "B";
        x[0][2] = "C";
        x[0][3] = "D";
        x[0][4] = "E";

        // 2nd row
        x[1][0] = "A1";
        x[1][1] = "B1";
        x[1][2] = "C1";
        x[1][3] = "D1";
        x[1][4] = "E1";

        // 3rd row
        x[2][0] = "A2";
        x[2][1] = "B2";
        x[2][2] = "C2";
        x[2][3] = "D2";
        x[2][4] = "E2";

        sop(x[1][2]); // C1
        sop(x[2][2]); // C2
        sop(x[0][4]); // E

        // Print all the values of 2d array: use 2 for loops/ nested for loops/ inner for
        // row=0, col=0 to 4
        // row=1, col=0 to 4
        // row=2, col=0 to 4
        for(int row=0; row<x.length; row++) {
            for(int col=0; col<x[0].length; col++) {
                sop(x[row][col]);
            }
        }
    }
```

~~6 What is Class and Object in Java~~

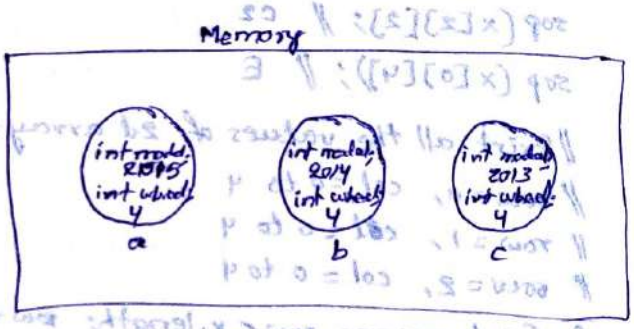
6 What is class and Object In java

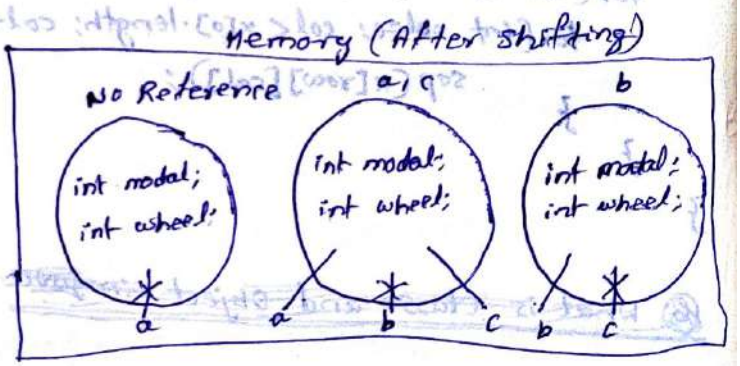
OOPConceptPart1
→ Car

→ A class is a template for creating objects which define its state and behavior. A class contains fields and methods to define the state and behavior of object.

→ A class is the creation of prototype and object is the actual implementation of that prototype. Technically, class is a template that describes what state and behavior an instance of this class can have. Object implements the state and behavior in the form of variables and methods.

```
⇒ public class Car{

    // class variables:
    int modal;
    int wheel;

public static void main (String[] args){

    // new car();  → This is the object of car class.
    // new keyword is used to create the object
    // a, b, c → Object reference variables.

    Car a = new Car();
    Car b = new Car();
    Car c = new Car();

    a.modal = 2015;
    a.wheel = 4;
    b.modal = 2014;
    b.wheel = 4;
    c.modal = 2013;
    c.wheel = 4;

    sop(a.modal); // 2015
    sop(a.wheel); // 4
    sop(b.modal); // 2014
    sop(b.wheel); // 4
    sop(c.modal); // 2013
    sop(c.wheel); // 4
}
}
```





~~7 What is Function or Method in java~~

Shifting Object References

```
⇒ public class Car{
    int modal;
    int wheel;
public static void main (String[] args){
    Car a = new Car();
    Car b = new Car();
    Car c = new Car();

    a = b;
    b = c;
    c = a;
    a.modal = 10;
    sop(a.modal); // 10
    c.modal = 20;
    sop(a.modal); // 20
    sop(c.mod); // 20
    }
}
```

7) What is Function or Method in Java

⊞ OOPConceptPart1
↳ FunctionsInJava

```
→ public class FunctionsInJava {
    // main method → starting point of execution
    public static void main (String[] args) {
        FunctionsInJava obj = new FunctionsInJava();
        // one object will be created, obj is the reference variable, refering to this
          object.
        // after creating the object, the copy of all non-static methods will be given
          to this object.

        obj.test();  // test method
        int l = obj.pqr();
        sop(l);  //30

        String s1 = obj.qa();
        sop(s1);  // selenium

        int div = obj.division(30,10);
        sop(div);  // 3
        // main method is void-never returns a value

    }
    // non static methods

    // void - doesn't return any value
    // return type=void
    public void test(){  // no input, no output
      sop("test method");
    }

    // return type = int
    public int pqr(){ no input, no output
      sop("pqr method");
      int a=10;
      int b=20;
      int c = a+b;
      return c;
    }

    // return type = String
    public String qa(){ // no input, some output
        sop("qa method");

        String s= "Selenium";
        return s;
    }
    // return type = int
    // x,y → Input parameters/arguments
    public int division (int x, int y){
      sop("division method");
      int d= x/y;
      return d;
}
```

⑧ What is Method Overloading

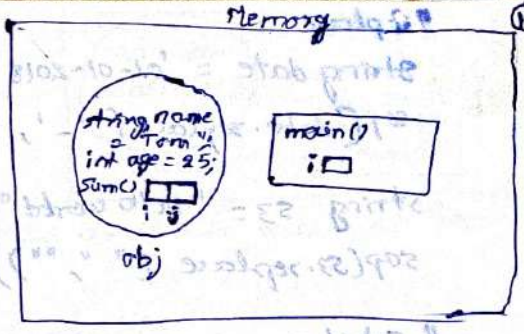
OOPConceptPart1
↳ MethodOverloading

```
⇒ public class MethodOverloading {
    public static void main(String[] args) {
        MethodOverloading obj = new MethodOverloading();
        obj.sum();
        obj.sum(10);
        obj.sum(10, 5);
    }
    public static void main(int p){
    }
    public static void main(int q, int h){
    }

    // we can overload main method also.
    // you cannot create method inside method.
    // duplicate methods i.e same method name with same no. of args not allowed.

    // Method Overloading → When the method name is same with different
       arguments or input parameters within the same class.
    // Methods can be differed by, Type of args., no. of args, order of args.
    public void sum() { // 0 input parameters
        sop("Sum method - zero parameters");
    }
    public void sum(double d) { // 1 input parameters
        sop("Sum method - ~~zero~~ 1 parameter");
    }
    public void sum(int i) { // 1 input parameter
        sop("sum method - 1 input parameter");
        sop(i);
    }
    public void sum(int k, int l){ // 2 input parameters.
        sop("Sum method - 2 input parameters");
        sop(k+l);
    }
}
```

output

```
Sum method - zero parameters
Sum method - 1 parameter
Sum method - 1 input parameter
10
Sum method - 2 input parameters
15
```

9 Differences between Local and Global variables



⊞ OOPConceptPart1
↳ LocalVsGlobalVariables.

```
public class LocalVsGlobalVariables {
    // Global variables or class variables.
    String name = "Tom";
    int age = 25;

    public static void main(String[] args) {
        int i = 10; // local variable for main method
        sop(i); // 10
        LocalVsGlobalVariables obj = new LocalVsGlobalVariables();
        sop(obj.name); // Tom
        sop(obj.age); // 25
    }

    public void sum() {
        int i = 15; // local variable for sum method
        int j = 20;
    }
}
```

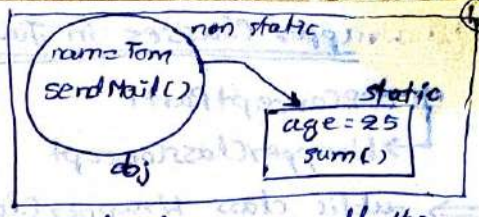
10 String Manipulation

⊞ OOPConceptPart1
↳ StringManipulation.

```
public class StringManipulation {
    public static void main(String[] args) {
        String s = "The trains have started here selenium";
        sop(s.length());
        sop(s.charAt(5));
        sop(s.charAt(26));
        sop(s.charAt(37)); // StringIndexOutOfBoundsException

        sop(s.indexOf('a'));
        sop(s.indexOf('s')); // 8 -> First occurance of s
        sop(s.indexOf('s', s.indexOf('s')+1)); // 15 -> second occurance of s
        sop(s.indexOf('s', s.indexOf('s', s.indexOf('s')+1)+1)); // 28 - 3rd occurance of s

        sop(s.indexOf("have"));
        sop(s.indexOf("hello")); // -1

        string s1 = "The trains have started here";
        sop(s.equals(s1));
        sop(s.equalsIgnoreCase(s1));
        // Trim
        String str = "    Hello world    ";
        sop(str.trim());
```

```
// Replace
string date = '01-01-2018'; // 01/01/2018
sop(date.replace('-', '/'));

string s3 = "Helb world";
sop(s3.replace(" ", ""));

// substring
string s4 = " The rains have started here";
sop(s4.substring(0, 9));

// split
string s5 = "Hellow_Selenium_Testing";
String arr[] = s5.split("_");
sop(arr[0]);
sop(arr[1]);
sop(arr[2]);

string h = "XxTestXxTestXxTesting";
string d[] = h.split("Xx");
for(int i=0; i<d.length; i++) {
    sop(i+"-->"+d[i]);
}

String firstName = "Tom;Naveen;Felix;Ipsi;Sharmi";
string firstName1[] = firstName.split(";");
for(int i=0; i<firstName1.length; i++) {
    sop(firstName1[i]);
}
}
```

(11) What is Static and Non-static in java



⊞ OOPConceptPart1
↳ StaticAndNonStaticConcept

⇒ public class StaticAndNonStaticConcept{

//Global vars: The scope of global variables will be available across all the functions with the same condition.

```
String name = "Tom";  // non-static global variable
static int age = 25;  // static global var.

public static void main(String[] args){
    // How to call static methods and vars:
    // 1. Direct calling
    sum();
    // 2. calling by class name
    StaticAndNonStaticConcept.sum();
    sop(age);
    sop(StaticAndNonStaticConcept.age);

    // How to call non-static methods and vars:
    StaticAndNonStaticConcept obj = new StaticAndNonStaticConcept();
    obj.sendmail();
    sop(obj.name);

    // Can I access static methods by using object reference? yes
    obj.sum();  // waring will be given - static method should be
                                        accessed in static way

}
public void sendMail(){  // non-static method
    sop("send mail method");
}
public static void sum(){  // static method
    sop("sum method");
}
}
```

o/p:

```
Sum method
25
25
send mail method
Tom
```

(12) Wrapper Classes in Java

OOPConceptPart1
→ WrapperClassConcept

```
⇒ public class WrapperClassConcept {
    public static void main(String[] args) {
// Integer, Double, Boolean
        String x = "100";
        sop(x+20); // 10020

// string to double conversion
        String y = "12.33";
        double d = Double.parseDouble(y);
        sop(d); // 22.33

// string to boolean
        String k = "true";
        boolean b = Boolean.parseBoolean(k);
        sop(b); // true

// Int to String conversion
        int j = 200;
        sop(j+20); // 220

        String s = String.valueOf(j); // "200"
        sop(s+20); // 20020

        String u = "100A";
        Integer.parseInt(u); // NumberFormatException - For input String: "100A"
    }
}
```

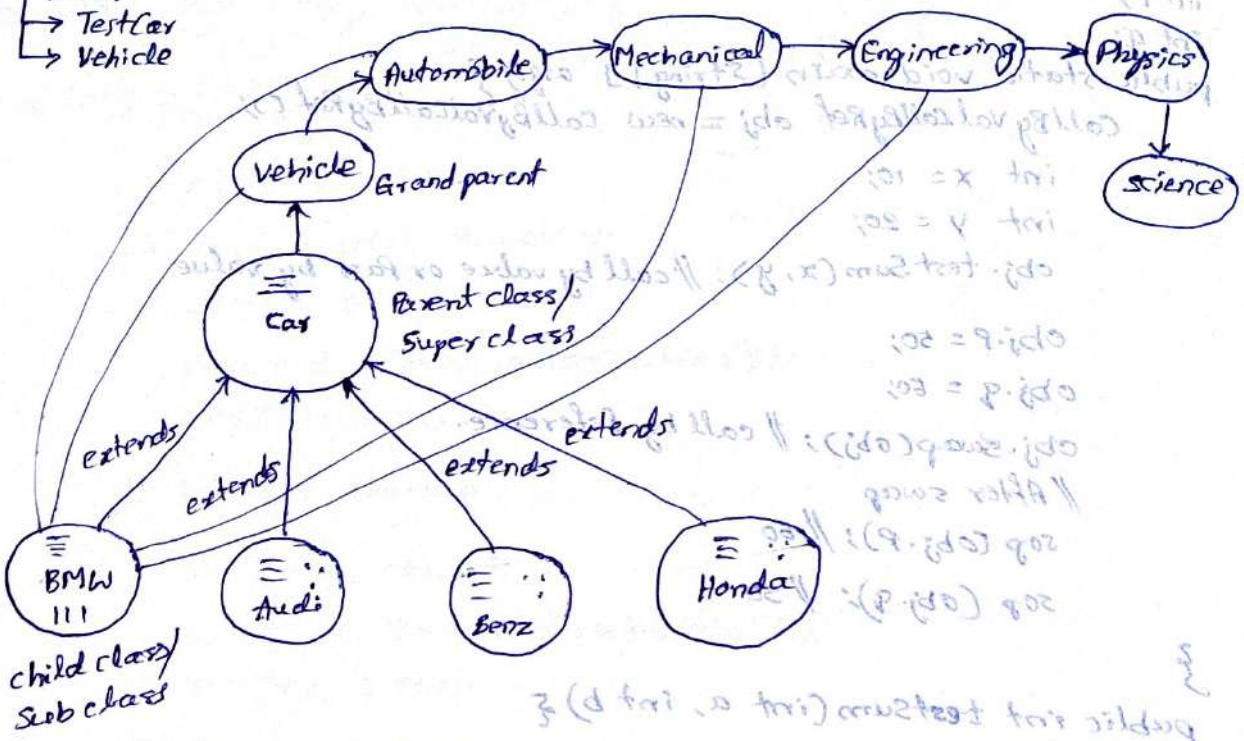
(13) What is call by value & call by reference in java

OOPConcept Part1
→ CallByValCallByRef

```
⇒ public class CallByValCallByRef {
    int p;
    int q;
    public static void main (String [] args) {
        CallByValCallByRef obj = new CallByValCallByRef ();
        int x = 10;
        int y = 20;
        obj.testSum (x, y); // call by value or Pass by value

        obj.p = 50;
        obj.q = 50;
        obj.swap(obj); // call by Reference.
        // After swap
        sop (obj.p); // 60
        sop (obj.q); // 50
    }
    public int testSum (int a, int b) {
        a = 30;
        b = 40;
        int c = a+b;
        return c;
    }
    public void swap (CallByVarCallByRef t) {
        int temp;
        temp = t.p; // temp = 50
        t.p = t.q ; // t.p = 60
        t.q = temp; // t.q = 50
    }
}
```

(19) Method Overriding, Inheritance and Polymorphism in Java

- OOPConceptPart2
  - BMW
  - Car
  - TestCar
  - Vehicle




```
⇒ public class Vehicle {
      public void engine() {
          sop("Vehicle - Engine");
      }
  }

⇒ public class Car extends Vehicle {
      public void start() {
          sop("car - start");
      }
      public void stop() {
          sop("Car - stop");
      }
      public void refuel() {
          sop("car - refuel");
      }
  }
```

⟹ public class BMW extends Car { // has-a relationship

// When same method is present in parent class as well as in child class with same name and same number of arguments is called Method-overriding

```
public void ~~theftSafety~~ start() { //overridden method
    sop("BMW - start");
}
public void theftSafety() {
    sop("BMW - theftSafety");
}
}
```

⟹ public class TestCar {

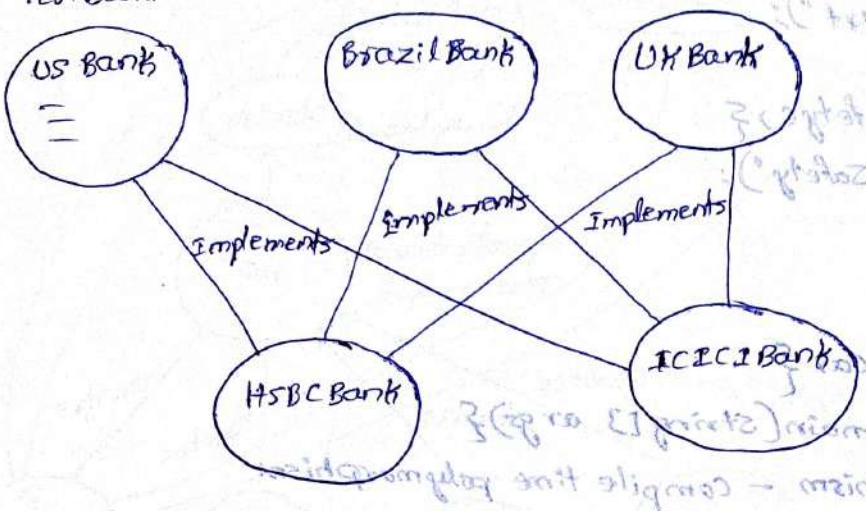
```
public static void main(String[] args) {
// static polymorphism - compile time polymorphism
    BMW b = new BMW();
    b.start();   // BMW - start
    b.stop();    // car - stop
    b.refuel();  // car - refuel
    b.theftSafety();  // BMW - theftSafety
    b.engine();  // vehicle - Engine
    sop("* * * * * * * *");
    Car c = new Car();
    c.start();   // car - start
    c.stop();    // car - stop
    c.refuel();  // car - refuel
    sop("* * * * * * * *");

    // Top casting (or) Up casting
    Car c1 = new BMW();  // Child class object can be refered by parent class reference variable - Dynamic Polymorphism (or) Run-time Polymorphism
    c1.start();  // BMW - start
    c1.stop();   // car - stop
    c1.refuel();  // car - refuel

    // Down casting
    BMW b = (BMW) new Car();  // classCastException
}
}
```

If we don't cast

→ When we use Dynamic polymorphism overridden methods in child class can be accessed ~~and~~ ~~all~~ in all parents.

(15) Interface Concept and Multiple Inheritance in Java

- OOPConceptPart2
  - USBank
  - BrazilBank
  - HSBCBank
  - TestBank



⇒ public interface USBank {

```
int min_bal = 100;
public void credit();
public void debit();
public void transferMoney();
```

// only method decleration
// no method body - only method prototype
// In interface, we can decleare the variables, variables are by default static in nature.
// variables will not be changed, its final/constant in nature
// no static method in interface
// no main method in interface
// we can't create the object of interface
// Interface is abstract in nature.
// Abstraction - 100%

}

⇒ public interface BrazilBank {

```
public void mutualFund();
```

}

⇒ public class HSBCBank implements USBank, BrazilBank { // we are achieving multiple inheritance

// Is-a relationship

// If a class is implementing any interface, then its mandatory to define/overide all methods of interface.

```
//overriding from USBank
public void debit (){
  sop ("HSBC - debit");
}
public void transferMoney (){
  sop ("HSBC - transfeMoney");
}
public void credit (){
  sop ("HSBC - credit");
}

// separate methods/local methods of HSBCBank class
public void educationLoan (){
  sop ("HSBC Bank - edu loan");
}
public void carLoan (){
  sop ("HSBC Bank - car Loan");
}

//overriding from BrazilBank
public void mutualFund (){
  sop ("HSBC - mutualFund");
}
}
```

IS-A Vs HAS-A

① IS-A simply means inheritance

eg:- class Animal { }

class Dog extends Animal { }

This means Dog IS-A animal

② HAS-A simply means reference

eg:- class Nip { }

class Dog { private Nip nip; }

This means Dog HAS-A Nip

⇒ public class TestBank {

```
public static void main (String[] args) {
  sop (USBank.min_bal);
  HSBCBank hs = new HSBCBank ();
  hs.credit (); // HSBC - credit
  hs.debit (); // HSBC - debit
  hs.transferMoney (); // HSBC - money transfer
  hs.educationLoan (); // HSBC - edu loan
  hs.carLoan (); // HSBC - car Loan
  // Dynamic polymorphism:
  // child classobject can be refered by parent Interface reference variable
  USBank b = new HSBCBank ();
  b.credit (); // HSBC - credit
  b.debit (); // HSBC - debit
  b.transferMoney (); // HSBC money transfer
}
}
```

(16) Abstraction in Java - Interface vs Abstract

OOP3
- Shape
- Test

→ Abstraction is the process of hiding the implimentation details, but actual business logic will be defined a in the overridden methods.

```
⇒ public abstract class Shape { // while creating class check abstract
    shape(){
     sop("shape class constructor");
    }
    int color;
    abstract void drawing();
    public void fill(){
        sop("shape -- fill");
    }
    // cann't create the object of abstract class like interface.

}

⇒ public class Test extends Shape {
    Test(){
      sop("Test class constructor");
    }
    public static void main(String[] args){
      // Shape s = new Test();
      //   s.drawing(); // Test -- drawing
      //   s.fill(); // Shape -- fill

    Test test = new Test();

      // shape class constructor
      // Test class constructor

    }

    @Override
    void drawing(){
        sop("Test -- drawing");
    }
}
```

Note:

→ When ever we create the object of inherited class of abstract class, abstract class constructor will be called first

→ Parent abstract (or) class constructor will be called when child class object is created

→ Whenever constructor of abstract class created will be called when object (or) of child class created.

OOP3
- Animal
- Dog

```
⇒ public abstract Animal {
    public void eat ( ){
        sop ("Animal -- eat ");
    }
    // 0 - 100% abstraction (partial abstraction)
}

⇒ public class Dog extends Animal {
    public static void main (String[] args) {
        Dog d = new Dog();
        d.eat();
    }
```

(17) Exception handling in Java

```
⇒ public class ExceptionHandling {
    // int a = 10;
    // static ExceptionHanding obj;
    public static void main(String[] args) {
        // uncaught exception:
        // int i = 9/0;
        // sop (i);

        // caught exception:
        // Thread.sleep (2000);
        //①. try-catch block :
        try{ int i = 9/0 ; // this code will throw an exception
        }
        catch (ArithmeticException e) {
            e.printstacktrace();
            sop (e.getMessage());
        }   sop ("This is / by zero error");
        sop ("ABC");
        sop ("ABC");
    }
}
```

→ Exception occured is printed at catch block and remaining statement will be executed after catch block even the exeeption occured.

→ Instead of 'ArithmeticException' if we use 'Exception' it will take care of any exception.

```
⇒ public class Throwskeyword {
  public static void main(String[] args){
    Throwskeyword obj = new Throwskeyword();
    obj.sum();
    sop("ABC");
  }
  public void sum(){
    try{
        div();
    } catch (ArithmeticException e){

    }
  }
  public void div() throws ArithmeticException{
      int i=9/0; // Exception line
  }
}
```

Throwable — object

Error Exception
- Arithmetic exception
- NullpointerException
- ...
- ...

Note

Exception flow: occured at div() and it is thrown to sum() to handle

```
⇒ public class Throwkeyword {
  public static void main(String[] args){
    sop("ABC");
    try {
        throw new Exception("Naveen Exception");
    } catch (Exception e){
        e.printStacktrace();
    }
    sop("PQR");
    String Execflag = " ";
    if (Execflag.equals(" ")){
    try {throw new Exception("Exec flag is Blank Exception");
    } catch (Exception e){
        e.printStackTrace();
    }
    }
    sop("Test");
  }
```

o/p:
```
ABC
PQR
java.lang.Exception : Naveen Exception
                    : Exec flag is Blank Exception
Test
```

→ To customize the exceptions we use throw keyword (∵ To give user defined exceptions)

(18) Can we overload main() method?

```
=> public class TestBase {
    //JVM -- main (String args[]) -- public static void
    public static void main (String[] args) {
        sop ("main method - 1");
        main ("Testing");
        main (10);
        main (10, 20);
    }

    public static void main (String args) {
        sop ("main method - 2");
    }

    public static void main (int a) {
        sop ("main method - 3");
    }

    public static void main (int a, int b) {
        sop ("main method - 4");
    }
}
```

o/p:- main method - 1
main method - 2
main method - 3
main method - 4

→ JVM first looks for the signature "public static void" and with "String[] args" or "String args[]" arguments

→ If we comment "main method - 1" we will'nt get the option to execute i.e "Run-As java application".

```
=> public class A {
    public static void main (String[] args) {
        sop ("A - main method");
        B.main (args);
    }
}

=> public class B {
    public static void main (String[] args) {
        sop ("B - main method");
        // A.main (args); // If we call again A-main it will becomes infinite loop.
    }
}
```

o/p:- A - main method
B - main method.

(19) Constructor concept and this keyword in Java

```
⇒ public class ConstructorConcept{
    // class vars : instance vars
    String name;
    int age;

    public ConstructorConcept(){ // 0 parameter
        sop("Default constructor");
    }

    public ConstructorConcept(int i) { // 1 parameter
        sop("1 parameter constructor");
        sop(i);
    }

    pubic ConstructorConcept(int i, int j) { // 2 parameter
        sop("2 parameter constructor");
        sop( i + "&" + j);
    }

    public constructorConcept(String name, int age) {
        this.name = name; // this.classvar = localvar
        this.age = age;
        // name = name1; // without this keyword
        // age = age1;
    }

    public static void main(String[] args){
        ConstructorConcept obj = new ConstructorConcept(); // def const will call
        ConstructorConcept obj1 = new ConstructorConcept(10);
        ConstructorConcept obj2 = new ConstructorConcept(10,20);
        ConstructorConcept obj3 = new ConstructorConcept("Tom", 25);
        sop(obj3.name);
        sop(obj3.age);
    }
}
```

o/p:- Default constructor
1 parameter constructor
10
2 parameter constructor
10&20
Tom
25

```
⇒ public class Car {
    String name;
    String modelNumber;
    int price;
    String engine;

    public Car(String name, String modelNumber, int price, String engine) {
        this.name = name;
        this.modelNumber = modelNumber;
        this.price = price;
        this.engine = engine;
    }

    public static void main(String[] args) {
        Car obj1 = new Car("BMW", "320d", 10, "Automatic");
        Car obj2 = new Car("Audi", "A8", 25, "Automatic");
        Car obj3 = new Car("Honda", "City", 5, "Manual");
        sop("obj1.name + " " + obj1.modelNumber + " " + obj1.price + " " + obj1.engine);
    }
}
```

O/P:- BMW 320d 10 Automatic

(20) This and Super keywords with Constructor Examples in Java

Super() - is used to call parent class constructor

```
⇒ public class A {
      public A() {
          sop("Parent class constructor");
      }
      public A(int i) {
          sop("Parent class constructor with value of i"+i);
      }
      public A(int i, int j) {
          sop("Parent class constructor with value of i"+i);
          sop("Parent class constructor with value of j"+j);
      }
}

⇒ public class B extends A {
      public B() {
          super();
          //super(i);
          //super(i,j);
      }
```

→ Not allowed to have multiple super keyword in one constructor

→ super keyword must be the first statement in the constructor.

```
      public B(int i) {
          super(i);
      }
      public B(int i, int j) {
          super(i, j);
      }
```

```
public static void main (String args[]) {
    B obj = new B();  // Default const will be called
    B obj1 = new B(10);  // 1 arg const "
    B obj2 = new B(10, 20);  // 2 arg const "
}
}
```

O/P:- Parent class constructor
Parent class constructor with value of i 10
Parent class constructor with value of j 10
Parent class constructor with value of j 20

(21) Final vs Finally vs Finalize (Reserve keywords)

Final: (It's a keyword)

→ Final keyword is used to define the constant values.
→ Final keyword is used to prevent the inheritance - If we use final keyword at class we can't inherit it (we can't extend to any child class).
→ Final keyword prevents the overriding also - If any method as final we can't override in child classes.

Finally: (It's a block)

→ Finally keyword is used with try-catch block - finally block will be executed after try-catch block.
→ Finally block code will be executed even the exception raised in try is handled or not in catch, even no exception raised finally block will be executed.

⇒ public class FinallyConcept {

```
    public static void main (String[] args) {
        // test1();
        // test2();
        Division();
    }
    public static void test1() {    // With Exception
        try {  sop("Inside test1 method");
            throw new RuntimeException("test");
        } catch (Exeeption e) {
            sop("Inside catch block");
        }
        finally {  sop("Inside finally block");
        }
    }
    public static void test2() {    // Without exception
        try {  sop("Inside test 2");
        }
        finally {  sop("Finally code inside test 2");
        }
    }
```

```
public static void division () {   // With irrelevent exception
    int i=10;
    try {  sop ("Inside try block");
           int k= i/0;  // Arithmetic exception
    }
    catch (NullPointerException e) {
        sop("Inside catch block");
        sop (" Devide by zero error");
    }
    finally {
          sop( "Execute this code even after any exception");
    }
}
```

Finalize : (It's a method)

→ It is used to perform the cleanup memory processing (Garbage collection).

→ Before garbage collector finalize() method will be called for garbage collection. we donn't need to call finalize method manually by obj ref.

→ This is just for cleanup processing the unused objects.

⇒

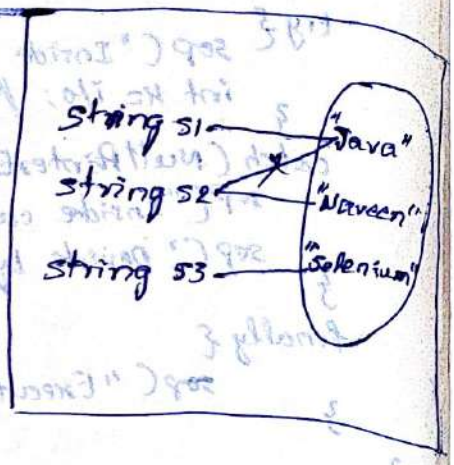
```
public class FinalizeConcept {
    public void finalize() {
        sop(" finalize method");
    }

    public static void main (String[] args) {
        FinalizeConcept f1 = new FinalizeConcept();
        FinalizeConcept f2 = new FinalizeConcept();
        f1 = null;
        f2 = null;

        System.gc(); // calling garbage collector manually
    }
}
```

o/p:- finalize method.

22 Why String is immutable and Final in Java

① String pool in method Area

```
String s1 = "Java";   // id=15
String s2 = "Java";   // id=15 → new id = 23
      s2 = "Naveen";
String s3 = "selenium";
```



② Security

③ Class loading mechanism

④ Optimization and performance

⑤ Integer, Long, Double, Float - Wrapper classes - Immutable and final

```
public class StringConcept {
    public static void main(String[] args) {
        String s1 = "Java";   // id = 15
        String s2 = "Java";   // id = 15 → new id = 23 (after changing s2)
                s2 = "Naveen";
        String s3 = "Selenium";

        sop(s1);
        sop(s2);
    }
}
```

23 Stack & Heap - Memory management in Java

Java Memory Management

Heap Memory

Young Generation

GC1
GC2
GC3
GCn

Minor GC will be performed

Eden

Survivor Memory

S0

Survivor memory

S1

Old Generation

Major GC
- Takes longer time
TimeOut error

Memory Pool:
Is used to store immutable objects:
StringClass, String pools
- runtime by JVM memory managers

Heap memory:
① stores objects
② JRE classes
③ GC runs on heap memory only to free some memory by destroying some objects which don't have any references

Permanent Generation

- Metadata of classes/methods
- Not part of heap memory
- will be generated at the run time

Method Area:
- class structure
- static vars
- constants

Runtime Constant pool:
- static vars
- Constant vars

Java Stack memory:
- Used to execute threads
- Method specific values
- local vars (int i=0)
- object references which are refering some objects on heap memory Test t1 = new Test();
- LIFO - Last In first out

```
public void m1() {
    int i=10;
}
public void m2() {
}
```

method block
whenever method invocation happens that particular method will occupies space here and will be cleared once done

→ stack memory size is very less as compared to heap memory.

24) Singleton Design Pattern in Java - How to achieve it in selenium

⇒ public class Singleton {

// In OOP, a singleton class is a class that can have only one object (instance of the class) at a time.

// How to design singleton class:

~~For normal class we use constructor where as for singleton class~~

① Make constructor as private

② Write a public static method (getInstance) that has return type of object of this singleton class (Lazy Initialization).

// Diff between normal class and singleton class:

for normal class we use constructor where as for singleton class we use getInstance() method for instantiation.

```
private static Singleton singleton_instance = null;
public String str;

private Singleton() {
    str = "Hey I am using singleton class pattern";
}

public static Singleton getInstance() {
    if(singleton_instance == null)
        singleton_instance = new Singleton();
    return singleton_instance;
}

public static void main(String[] args) {
    Singleton x = Singleton.getInstance();
    Singleton y = Singleton.getInstance();
    Singleton z = Singleton.getInstance();

    x.str = (x.str).toUpperCase();
    sop(x.str); // HEY I AM USING SINGLETON CLASS PATTERN
    sop(y.str); //        "                 "
    sop(z.str); //        "                 "

    z.str = (z.str).toLowerCase();

    sop(x.str); // hey i am using singleton class pattern
    sop(y.str); //        "                 "
    sop(z.str); //        "                 "

}
}
```

How to Achieve singleton design pattern in selenium:

```
⇒ public class DriverInit {
    static DriverInit instanceDriver = null;
    private
    private WebDriver driver;

    private DriverInit() {

    }
    public WebDriver openBrowser() {
        System.setProperty("webdriver.chrome.driver", ".\drivers\chromedriver.exe");
        driver = new ChromeDriver();
        return driver;
    }
```

```
public static DriverInit getInstance(){
    if(instanceDriver == null)
        instanceDriver = new DriverInit();
    return instanceDriver;
}
}
```

⟹

```
public class PageTest {
    WebDriver driver;
    @BeforeMethod
    public void setUp(){
        DriverInit instanceDriver = DriverInit.getInstance();
        driver = instanceDriver.openBrowser();
    }

    @Test
    public void testMethod1(){
        driver.get("http://www.google.com");
        sop(driver.getTitle());
    }

    @AfterMethod
    public void tearDown(){
        driver.quit();
    }
}
```

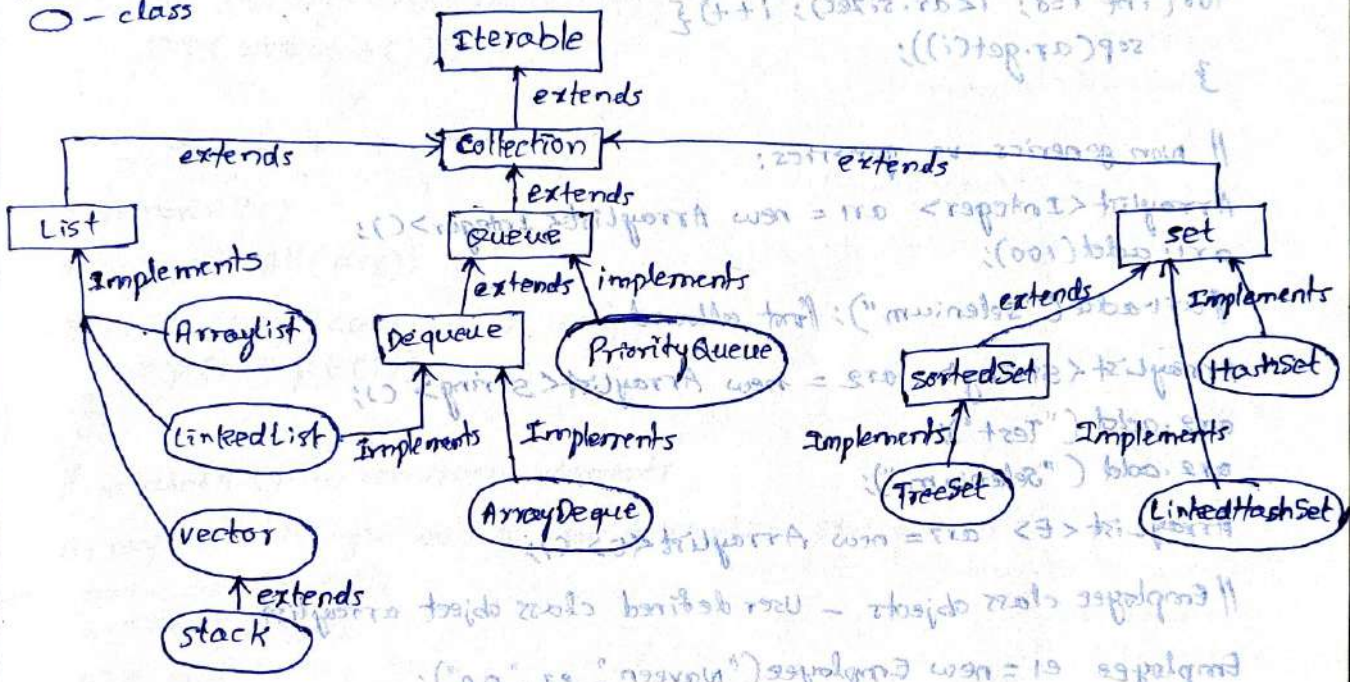
① Overview of Collection Framework

Framework - to store and manipulate the groups of objects

operations - Searching, Insertion, Manipulation, Deletion, Sorting

Interface:	Classes:
Iterable	
Collection	
List	Arrylist, LinkedList, Vector, Stack
Queue	
Dequeue	Priority Queue, ArrayDeque
Set	HashSet, LinkedHashSet, TreeSet
SortedSet	

▭ - Interface
⬭ - class



② Arraylist

CollectionsSessions (Project)
- ListConcept
  - ArrayListConcept
  - Employee

```
⟹ public class ArrayListConcept{
    public static void main (String[] args){
        int a[] = new int[3]; // static array - size is fixed.
        // Dynamic array - ArrayList:
        // 1. can contains duplicate values/elements
        // 2. Maintains insertion order
        // 3. Synchronized
        // 4. Allows random access to fetch the element because it stores the values
           on the basis of indexes

        Arraylist ar = new ArrayList();
        ar.add(10); // 0
        ar.add(20); // 1
        ar.add(30); // 2
        sop(ar.size()); // 3
```

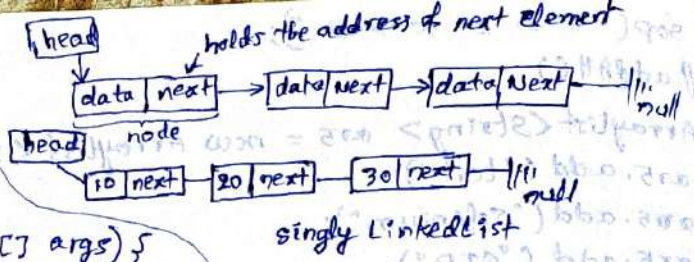
```
ar.add(40); //3
ar.add(50); //4
ar.add(50); //5
ar.add(12.33); //6
ar.add("Test"); //7
ar.add('a'); //8
ar.add(true); //9

sop(ar.size()); // size of arraylist
sop(ar.get(4)); // to get the values from an index

// To print all the values from arraylist:
1. for loop
2. using iterator

for(int i=0; i<ar.size(); i++){
    sop(ar.get(i));
}

// Non generics vs generics:
ArrayList<Integer> ar1 = new ArrayList<Integer>();
ar1.add(100);
// ar1.add("selenium"); // not allowed
ArrayList<String> ar2 = new ArrayList<String>();
ar2.add("Test");
ar2.add("Selenium");
ArrayList<E> ar3 = new ArrayList<E>();

// Employee class objects - User defined class object arraylist
Employee e1 = new Employee("Naveen", 27, "QA");
Employee e2 = new Employee("Tom", 28, "Dev");
Employee e3 = new Employee("Peter", 29, "Admin");

// create arrayList
ArrayList<Employee> ar4 = new ArrayList<Employee>();
ar4.add(e1);
ar4.add(e2);
ar4.add(e3);

// Iterator to traverse the values:
Iterator<Employee> it = ar4.iterator();
while(it.hasNext()){
Employee emp = it.next();
sop(emp.name);
sop(emp.age);
sop(emp.dept);
}
```

```
sop("* * * * * * ");
// addAll()
Arraylist<String> ar5 = new Arraylist<String>();
ar5.add("test");
ar5.add("selenium");
ar5.add("QTP");

Arraylist<String> ar6 = new ArrayList<String>();
ar6.add("dev ");
ar6.add(" Java");
ar6.add("Javascript");

ar5.addAll(ar.6);

for(int i=0; i<ar5.size(); i++){
    sop(ar5.get(i));
}

sop("* * * * * * * ");

// removeAll()
ar5.removeAll(ar6);

for(int i=0; i<ar5.size(); i++){
    sop(ar5.get(i));
}

// retainAll() - common elements
ArrayList<String> ar7 = new ArrayList<String>();
ar7.add("test"); ✓
ar7.add("selenium");
ar7.add("QTP");

ArrayList<String> ar8 = new ArrayList<String>();
ar8.add("test"); ✓
ar8.add(" Java");
ar8.add("Javascript");

ar7.retainAll(ar8);
for(int i=0; i<ar7.size(); i++){
    sop(ar7.get(i));
}

}
```

⟹

```
public class Employee{
    String name;
    int age;
    String dept;
    Employee(String name, int age, String dept){
        this.name = name;
        this.age = age;
        this.dept = dept;
    }
}
```

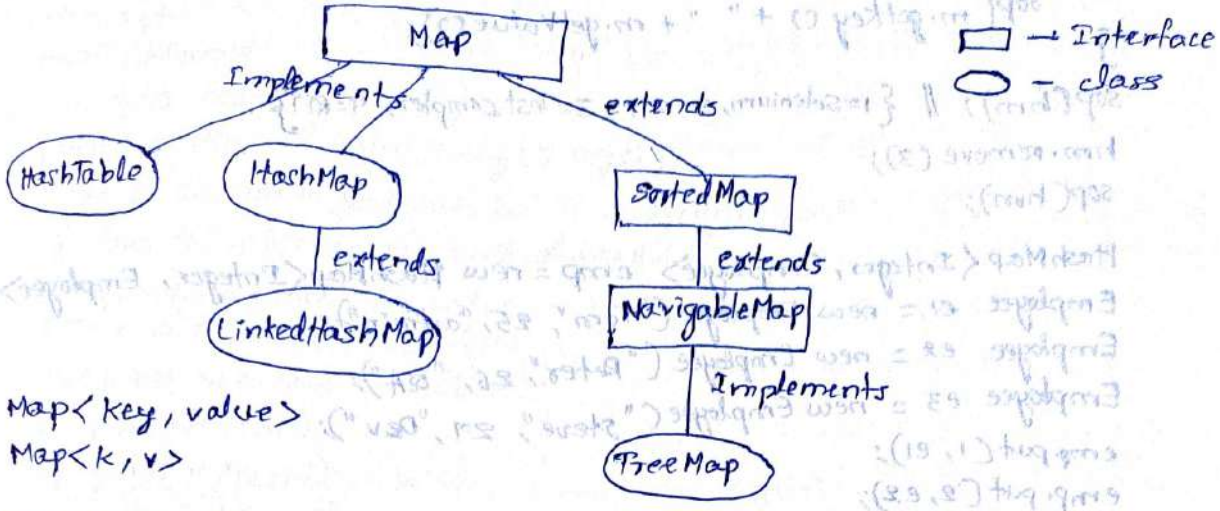
③ Linked List



- List Concept
  - ↳ LinkedListConcept

```
public class LinkedListConcept {
  public static void main (String[] args) {
    LinkedList<String> ll = new LinkedList<String>();
    //add:
    ll.add("test");
    ll.add("qtp");
    ll.add("selenium");
    ll.add("RPA");
    ll.add("RFT");

    // print:
    sop(" contents of Linked List:"+ ll); //contents of LinkedList: [test, qtp, selenium, RPA, RFT]

    //addfirst
    ll.addFirst("Naveen");

    // addLast
    ll.addLast("Automation");
    sop("contents of linkedlist: "+ll);

    // get:
    sop(ll.get(0));

    // set:
    ll.set(0,"Tom");
    sop(ll.get(0));

    // Remove first and last element:
    ll.removefirst();
    ll.removeLast();
    sop("contents of linkedlist: "+ll);
    ll.remove(2); // remove element at 2 index
    sop("contents of linkedlist:"+ll);

    // how to print all the values of LinkedList:
    // for loop
    sop("*** Using for loop ***");
    for(int n=0; n<ll.size(); n++){
        sop(ll.get(n));
    }

    // Advanced for loop - foreach loop
    sop("*** Using advanced for loop ***");
    for (String str : ll){
      sop(str);
    }

    // iterator
    sop("*** Using iterator ***");
    Iterator<String> it = ll.iterator();
    while(it.hasNext()){
        sop(it.next());
    }

    //while loop
    sop("*** Using while loop ***");
    int num=0;
    while(ll.size() > num){
      sop(ll.get(num));
      num++;
    }
```

④ Map Interface



Map<key, value>
Map<k, v>

```
Map<String, String> m = new Map<String, String>();
m.put("name", "Tom");
m.put("dept", "admin");

m.get("name"); // Tom
```

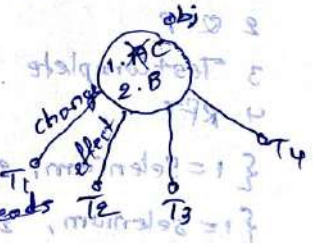
Examples of Map:

1. Error codes (1.2.1: "fatal error")
2. Map of zip codes
3. Map of managers and employees
4. Test data storage
5. Configuration/environmental variables
6. Headers in Rest API

Methods:

1. put()
2. putAll()
3. get(key)
4. containskey (key)
5. remove(key)
6. keySet()
7. entrySet()

⑤ How to use HashMap in Java



~~ClassConcept~~
↳ HashMapConcept

```
⇒ public class HashMapConcept{
    public static void main (String[] args){
  // HashMap is a class implements Map Interface
  // Extends AbstractMap
  // It contains only unique elements
  // Stores the values - key & value pair
  // It may have one null key and multiple null values
  // It maintains no order
  // hashmap is non-synchronized → not thread safe
  // concurrent modification exception → Fail - fast condition

  HashMap<Integer, String> hm = new HashMap<Integer, String>();
  hm.put(1, "Selenium");
  hm.put(2, "QTP");
  hm.put(3, "Test Complete");
  hm.put(4, "RFT");
  sop(hm.get(1));
  sop(hm.get(4));
```

```
for(Entry m: hm.entrySet()) {
    sop(m.getKey() + " " + m.getValue());
}
sop(hm); // {1=Selenium, 2=QTP, 3=TestComplete, 4=RFT}
hm.remove(3);
sop(hm);

HashMap<Integer, Employee> emp = new HashMap<Integer, Employee>();
Employee e1 = new Employee("Tom", 25, "admin");
Employee e2 = new Employee("Peter", 26, "QA");
Employee e3 = new Employee("Steve", 27, "Dev");
emp.put(1, e1);
emp.put(2, e2);
emp.put(3, e3);

// Traverse the HashMap:
for(Entry<Integer, Employee> m: emp.entrySet()) {
    int key = m.getKey();
    Employee e = m.getValue();
    sop("Employee " + key + "Info ");
    sop(e.name + " " + e.age + " " + e.dept);
}
}
```

o/p:

```
1 Selenium
2 QTP
3 TestComplete
4 RFT
{1=Selenium, 2=QTP, 3=TestComplete, 4=RFT}
{1=Selenium, 2=QTP, 4=RFT}
Employee 1 Info
    Tom 25 admin
Employee 2 Info
    Peter 26 QA
    Employee 3 Info
    Steve 27 Dev
```

⑥ HashTable

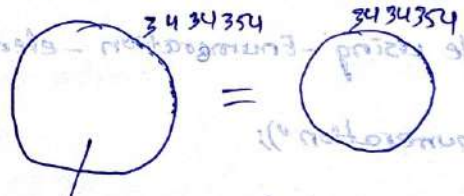
- ⊞ ListConcept
  - ↳ HashTableConcept

```
⇒ public class HashTableConcept {
    public static void main(String[] args) {
      // It is similar to hashmap, but it is synchronized
      // Stores the value on the basis of key & val
      // key → Object - HashCode & value
      // no null key and null values
      Hashtable h1 = new Hashtable();
      h1.put(1, "Tom");
      h1.put(2, "Test");
      h1.put(3, "Java");

      // create a clone copy/shallow copy:
      Hashtable h2 = new Hashtable();
      h2 = (Hashtable) h1.clone();
      sop("values from h1:" + h1);
      sop("values from h2:" + h2);

      h1.clear();
      sop("values from h1: " + h1);
      sop("values from h2: " + h2);
      // contains value:
      Hashtable st = new Hashtable();
      st.put("A", "Naveen");
      st.put("B", "Manager");
      st.put("C", "Selenium");
      if (st.containsValue("Naveen"))
          sop("value is found");
      // print all the values from hashtable using - Enumeration - elements()
      Enumeration e = st.elements();
      sop("print values from st using enumeration");
      while (e.hasMoreElements()) {
          sop(e.nextElement());
      }
      // get all the values from hashtable using - entrySet() - set of hashtable values
      sop("print values from st using entrySet");
      Set s = st.entrySet();
      sop(s);

      Hashtable st1 = new Hashtable();
      st1.put("A", "Naveen");
      st1.put("B", "Manager");
      st1.put("C", "Selenium");
      st1.put("C", "Selenium"); // It contains only unique values
      // no null key and null values
      // st1.put("D", null); // nullPointerException
```

```
sop("values from st1 : ");
sop (st1);
// check both the hashtables are equal or not:
if (st.equals(st1))
        sop ("Both are equal");
// get the value from a key:
sop (st1.get("A"));
// get the hashcode of hashtable object:
sop ("hash code value of st1 : " + st1.hashCode());

// generics:
Hashtable<String, String> st3 = new Hashtable<String,String>();

}
}
```

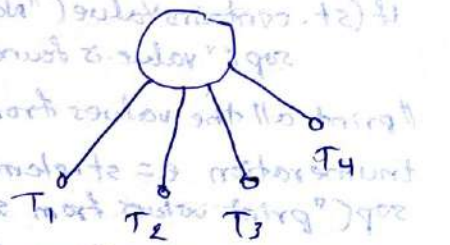
o/p:- values from h1 {3=Java, 2=Test, 1=Tom}
values from h2 {3=Java, 2=Test 1=Tom}
values from h1 {}
values from h2 {3=Java, 2=Test, 1=Tom}
value is found
print values from st using enumeration
Naveen
selenium
Manager
print values from st using entry set
[A=naveen, c=selenium, B=Manager]

hashcode number - 32 bit int



Test t1 = new Test();

if two objects are equal returns the same values, contains same hashcode



Threads run one by one (thread-safe)

① ArrayList vs Hashtable

```
⇒ public class ArrayConcept {
    public static void main (String[] args) {
    // Disadvantage of array:
    1. size is fixed: static array: To overcome from this problem, we use collections
       - dynamic array - ArrayList
    2. It stores only similar datatype: To overcome this problem, we use Object array.

    // Int Array:
    int i[] = new int[4];
    i[0] = 10;
    i[1] = 20;
    i[2] = 30;
    i[3] = 40;
    // i[4] = 50; // ArrayIndexOutofBoundsException: 4
    sop (i[2]);
    sop (i[3]);
    // size of Array
    sop(i.length);
    // To print all values of array: we use for loop
    for (int j=0; j<i.length; j++) {
        sop(i[j]);
    }
}
⇒ public class ArrayListConcept {
    public static void main (String[] args) {
    ArrayList ar = new ArrayList();
    ar.add (100); //0
    ar.add (200); //1
    ar.add (300); //2
    sop(ar.size());
    ar.add(400); // 3
    ar.add(500); //4
    sop(ar.size());
    ar.add ("Tom"); //5
    ar.add ("Hello"); //6
    ar.add (12.33); //7
    ar.add ('M'); //8
    ar.add (600); //9
    sop(ar.size());
    ar.remove(9);
    sop(ar.size());
    sop(ar.get(4));
    // sop(ar.get(9)); // IndexOutofBoundsException.

    // to print all values of ArrayList: for loop
    for (int i=0; i<ar.size(); i++) {
        sop(ar.get(i));
    }
}
```

```
HashTable Concept {
public class
    public static void main (String[] args) {
        Hashtable h = new Hashtable();
        h.put("A", "Test");
        h.put("B", "Hello");
        h.put("c", "world");
        sop (h.size());
        h.put (1, 100);
        h.put (2, 200);
        sop( h.size());
        sop( h.get(2));
        sop( h.get("c"));
        h.put ( 3, "Tom");
        Hashtable<Integer, Integer> h1 = new Hashtable<Integer, Integer>();
        h1.put (1, 100);
        // h.put ("A", 200); // not allowed

        Hashtable<Integer, String> h2 = new Hashtable<Integer, String>();
        h2.put (1, "Tom");
```

(8) How to use HashMap in selenium WebDriver

```
⇒ public class Data {
    public static HashMap<String, String> getUserLoginInfo(){
        HashMap<String, String> userMap = new HashMap<String, String>();
        userMap.put("customer", "naveenk_test@123");
        userMap.put("admin", "adminuser_test@123");
        userMap.put("superadmin", "superadminuser_test@123");
        userMap.put("director", "directoruser_test@123");
        return userMap;
    }

    public static HashMap<Integer, String> monthMap(){
        HashMap<Integer, String> monthMap = new HashMap<Integer, String>();
        monthMap.put(1, "January");
        monthMap.put(2, "February");
        monthMap.put(3, "March");
        monthMap.put(4, "April");
        monthMap.put(5, "May");
        monthMap.put(6, "June");
        monthMap.put(7, "July");
        monthMap.put(8, "August");
        monthMap.put(9, "September");
        monthMap.put(10, "October");
        monthMap.put(11, "November");
        monthMap.put(12, "December");

        return monthMap;
    }
}

⇒ public class FreeCRMTest {
    public WebDriver driver;
    public String credentials;
    public String credentialsInfo[];
    @BeforeMethod
    public void setUp(){
        System.setProperty("webdriver.chrome.driver", "<chrome drivers path>");
        driver = new ChromeDriver();
        driver.get("https://www.freecrm.com");
    }
    @Test(priority=1)
    public void loginWithAdminUserTest(){
        credentials = Data.getUserLoginInfo().get("admin");
        credentialsInfo = credentials.split("_");
        driver.findElement(By.name("username")).sendKeys(credentialsInfo[0]);
        driver.findElement(By.name("password")).sendKeys(credentialsInfo[1]);
        WebElement loginBtn = driver.findElement(By.xpath("//input[@type='submit']"));
        JavaScriptExecutor js = (JavaScriptExecutor) driver;
        js.executeScript("arguments[0].click();", loginBtn);
```

```
    Thread.sleep(3000);
    driver.switchTo().frame("mainpanel");
    Select s = new Select(driver.findElement(By.name("selectMonth")));
    s.selectByVisibleText(Data.monthMap().get(10));
}

@Test(priority=2)
public void loginWithCustomerUserTest(){
  ~~driver~~
  credentials = Data.getUserLoginInfo().get("customer");
  credentialsInfo = credentials.split("_");

  driver.findElement(By.name("username")).sendKeys(credentialsInfo[0]);
  driver.findElement(By.name("password")).sendKeys(credentialsInfo[1]);
  WebElement loginBtn = driver.findElement(By.xpath("//input[@type='submit']"));
  JavascriptExecutor js = (JavascriptExecutor) driver;
  js.executeScript("arguments[0].click();", loginBtn);
}

/Assignment:
/* Url
   login with customer user
   place the order
   Logout

   login with admin user
   check the order
   check all the orders for the day
   logout
*/

}
```

① How to find duplicate Elements in an array

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

        String names[] = {"Java", "JavaScript", "Ruby", "c", "python", "Java"};

    // 1. compare each element : o (nxn) ..... worst solution
    for (int i=0; i< names.length; i++) {
        for (int j=i+1; j< names.length; j++) {
            if( names[i].equals (names[j])) {
                sop( "Duplicate element is : "+ names[i]);
            }
        }
    }
    sop ("= = = = = = = = = = = = = ");

    // 2. Using Hash set : Java collection : It stores unique values : o(n)
    Set<String> store = new HashSet<String> ();
    for (String name : names) {
        if( store.add (name) = = false) {
            sop ("Duplicate element is: "+ name);
        }
    }
    sop("= = = = = = = = = = = = ");

    Iterator itr= store.iterator();
    while( itr.hasNext()) {
        sop ("Duplicate: "+ itr.next());
    }

    // 3. Using HashMap : o (2n)
    Map <String, Integer> storeMap = new HashMap <String, Integer> ();
    for(String name : names) {
        Integer count = storeMap.get (name);
        If (count = = null) {
            storeMap.put(name, 1);
        }
        else {
            storeMap.put (name, ++count);
        }
    }

    // printing duplicates
    for(String s : storeMap.keySet ()) {
        if (storeMap.get (s) >1) {
            sop ("Duplicate element is : "+ s);
        }
    }
}
```

no. of occurance in storeMap

② How to find missing number in Array.

```
⇒ public class MissingNumber {
  public static void main (String[] args){
  // int a[] = {1, 2, 3, 4, 5, 7, 8, 9, 10, ..... 100};
  // 1+2+4+5 = 12
  // 1+2+3+4+5 = 15
  // 15-12 = 3
  // 22
  // 28 - 22 = 6
  int a[] = {-1, 0, 1, 2, 3, 5, 6, 7, 8, 9, 10};
  int sum = 0;
  for (int i=0; i< a.length ; i++){
      sum = sum + a[i]; // 50
  }
  sop (sum);
  int sum1 = 0;
  for (int j = -1; j<=10 ; j++){
      sum1 = sum + j ; // 54
  }
  sop (sum1);
  sop ("Missing number is : "+ (sum1 - sum)); // 4
}
```

first value in the sequence

last value in the sequence

★ → To do this elements should be in sorted order

③ How to Reverse a String

```
⇒ public class ReverseString {
  public static void main(String[] args) {
  // Reverse a String
  // Diff. b/w string and stringBuffer ? string - immutable
                                         stringBuffer - mutable
  // Do we have reverse function in String ? No
  String s = "Selenium";
  // 1. Using for loop
  int len = s.length(); //8
  String rev = " ";
  for (int i=len-1 ; i>=0; i--){
      rev = rev + s.charAt(i); // muineles
  }
  sop(rev);
  // 2. Using String Buffer class
  StringBuffer sf = new StringBuffer(s);
      sop(sf.reverse()); // muineles
  }
}
```

```
⇒ public class ReverseStringUsingRecursion{
    psvm (String arg) {
        String s = "Selenium";
        String revString = reverseString (s);
        sop("Reversed String is:" + revString);
    }
    public static String reverseString(String s){
        if(s.isEmpty()){
            return s;
        }
        return reverseString(s.subString(1))
                    + s.charAt(0);
    }
}
```

④ How to Reverse an Integer

```
⇒ public class ReverseInteger{
  public static void main(String[] args){
  //1. Using Algorithm
    long num = 12345; // 54321
    long rev = 0;
    while (num != 0){
      rev = rev * 10 + num % 10; // 54321
      num = num / 10 ; // To remove last digit from 12345
    }
    sop ("Reverse number is : " + rev);
  // 2. Using String Buffer method
    long num1 = 123456;
    sop (new StringBuffer(String.valueof (num1)).reverse());
  }
}
```

⑤ How to find the largest number in the Given Integer Array
How to find the smallest number in the Given Integer Array

```
⇒ public class SmallestAndLargestIntegerNumber{
    int numbers[] = {-10, 24, 50, -88, 987656};
    int largest = numbers[0];
    int smallest = numbers[0];
    for (int i = 1; i < numbers.length; i++){
      if (numbers[i] > largest){
        largest = numbers[i]; // 987656
      }
      else if (numbers[i] < smallest){
        smallest = numbers[i]; // -88
      }
      sop ("\n Given array is: " + Arrays.toString(numbers));
      sop ("Largest number is: " + largest);
      sop ("Smallest number is: " + smallest);
    }
}
```

O/P:- Given array is: [-10, 24, 50, -88, 987656]
Largest number is: 987656
Smallest number is: -88

⑥ How will you Remove Junk - special characters in a String

```
⇒ public class RemoveJunk {
  public static void main (String[] args) {
      String s= "#@!β Latin string 0123456789 0";
      // Regular Expression : [^a-zA-Z0-9]   ← Except these
      s = s.replaceAll ("[^a-zA-Z0-9]", " ");
      sop (s);
  }
}
```

o/p:- Latin string 0123456789

⑦ String Manipulation in Java

```
⇒ public class stringManipulation {
  public static void main (String[] args) {
      String str = "The rains have started here";
      String str1 = "The rains Hare started here";
      sop(str.length ());
      sop(str.charAt (5));
      sop(str.indexOf('s')); // Ist occurance of s
      sop(str.indexOf('s', str.indexOf('s')+1); // 2nd occurance of s
      sop(str.indexOf("have");
      sop(str.indexOf("Hello")); // -1
      // String Comparison:
      sop(str.equals (str1));
      sop(str.equalsIgnoreCase (str1));
      // SubString:
      sop(str.subString (0,9));
      // Trim:
      String s= "  How world  "; // Hello world
      sop(s.trim ());
      sop(s.replace (" ", ""));
      String date = "01-01-2017"; // 01/01/2017
      sop(date.replace ("-", "/");
      // Split:
      String test = "Hello world_test_selenium";
      String testval[] = test.split("_");
      for(int i=0; i<testval.length; i++){
          sop(testval [i]);
      }
```

```
String s2 = "cares";
sop(s2.concat("s"));
String x = "Hello";
String y = "world";
int a = 100;
inb b = 200;
sop(x+y);
sop(a+b);
sop(x+y+a+b); // Helloworld100200
sop(a+b+x+y); // 300Helloworld
sop(x+y+(a+b)); // Helloworld 300
}
}
```

⑦ <u>How to swap two Integers without using temp/ third variable</u>

```
⇒ public class SwapWithoutThirdVar {
  public static void main(String[] args){
    int x = 5; // 0101
    int y = 10; // 1010
    // o/p should get: x = 10, y = 5.
    // 1. with using third variable: t
      int t;
      t = x; // 5
      x = y; // 10
      y = t; // 5
    // 2. without using third var: using + operator
      x = x + y; // 15
      y = x - y; // 5
      x = x - y; // 10
    // 3. without using third var: using * operator
      x = x * y; // 50
      y = x / y; // 5
      x = x / y; // 10
    // 4. using XOR : ^
      x = x ^ y; // 15 → 1111
      y = x ^ y; // 10 → 1010
      x = x ^ y; // 5 → 0101
      sop(x); // 10
      sop(y); // 5
  }
}
```

8) Swap two strings without using third variable

```
⇒ public class StringSwap {
    public static void main (String [] args) {
        String a = "Hello";
        String b = "world";
        sop(" Before swaping ");
        sop("The value of a is: "+a);
        sop(" The value of b is: "+b);
    // 1. Append a and b
        a = a+b; // Hello world
    // 2. store initial string a in string b:
        b = a.substring (0, a.length () - b.length ()); // ~~world~~ Hello
    // 3. store initial string b in string a:
        a = a.substring (b.length ()); // ~~Hello~~ world
        sop(" The value of a & b after swapping ");
        sop ("The value of a is:" + a);
        sop ("The value of b is: " + b);
```

O/P:
Before swapping
The value of a is: Hello
" b " : world
After swapping
The value of a is: world
" b " : Hello

9) Prime Number

```
⇒ public class PrimeNumber {
    psvm(String[] args){
    sop("Enter number");
    Scanner s = new Scanner(System.in);
    int n = s.nextInt();
    primeChecker(n);
    }
    public static void primeChecker(int num){
    int count = 0;
    for (int i=1; i<=num; i++){
        if (num%i == 0){
            count++;
        }
    }
    if(count == 2){
    sop(" Prime number");
    }else{
    sop("Not a prime number");
    }
    }
}
```

O/P:- Enter number:
7
Prime number

```
⇒ public class primeNumbersBetween1toN {
    psvm(String[] args){
    sop("Enter n value: ");
    Scanner s = new Scanner(System.in);
    int n = s.nextInt();
    sop("Prime numbers between 1 to "+n+" are:");
    for(int i=2; i<=n; i++){
        int count=0;
        for(int j=1; j<=i; j++){
            if(i%j==0){
                count++;
            }
        }
        if(count==2){
            sop(i+" ");
        }
    }
    }
}
```

O/P:- Enter n value:
100
2 3 5 7 11 13 17 19 23 29 31
41 43 47 53 59 61 67 71 73 79 83
97

(25) Can we override a static method in child class?

→ we can't override a static method in child class. but we can overload.

```
⇒ public class car {
    // Method Hiding
    public static void start () {
        sop ( "car --- start");
    }
    public static void start(int a) {
        sop ( "car --- start");
    }
    public static void start (String price) {
        sop ( " car --- start");
    }
    public void stop () {
        sop ( "car --- stop");
    }
    public void refuel () {
        sop ( " car --- refuel");
    }
}
⇒ public class BMW extends car {
    /*
    @Override
    public void start () {
        sop( " BMW --- start ");
    }
    */
    // @Override - can't be overriden
    public static void start () {
        sop ( " BMW --- start");
    }
}
⇒ public class TestCar {
    public static void main (String [] args) {
    BMW b = new BMW ();
    b.start();
    b.stop();
    b.refuel();
}
```

// This instance method can't override from car
// quick fix : Remove 'static' modifier of 'car' static

// waring - static method start() in BMW should be accessed in static way.

// we can never access the start () of car class using BMW obj. ref.
i.e super class start method is Hidden

o/p:- BMW --- start
car --- stop
car --- refuel

① Count the no. of occurance of each word

```
→ public class CountWords {
    public static void main (String[] args){
      String str = "I am learning learning java java programming";
      Map<String, Integer> map = new HashMap<String, Integer> ();
       Integer count = 1;
      String[] arr = str.split(" ");
      for (int i=0; i<arr.length; i++) {
          if(!map.containskey (arr[i]){
             map.put (arr[i], count);
          }else{
                map.put(arr[i], map.get(arr[i])+1);
          }
      }
      // Printing the output
      for (String x : map.keySet()){
          sop(" The count of word : "+x+" = "+map.get(x));
      }
    }
}
```

simple way

```
Map<String, Integer> m = new HashMap<String, Integer> ();
    String str = str.split(" ");
      for( String name : str){
      Integer count = m.get ( name);
        if (count == null) {
           m.put ( name, 1);
        }else{ m.put(name, ++count);
        }
      }
```

O/P:-

The count of word : Java = 2
The count of word : I = 1
The count of word : learning = 2
The count of word : am = 1
The count of word : programming = 1

② Count the occurance of given character from a word/ sentence

```
→ public class CountCharacterOccurance{
    public static void main (String[] args){
        String str = "Java is object oriented language";
        // Jv is object oriented lnguge
      int result = str.length() - str.replaceAll("a", "").length();
      sop("Result is:"+ result);
    }
}
```

O/P:- Result is : 4

③ Find duplicate characters in a string

→
```
public class FindDuplicateCharacters{
    public static void main(String[] args){
        String str = "Programming";
        char[] arr = str.toCharArray();
        Map<Character, Integer> map = new HashMap<Character, Integer>();
        int count = 1;
        for(int i=0; i<arr.length; i++){
            if(!map.containsKey(arr[i])){
                map.put(arr[i], count);
            }else{
                map.put(arr[i], map.get(arr[i])+1);
            }
        }
        // printing duplicates
        for(Character key : map.keySet()){
            if(map.get(key)>1){
                sop(key + ":" + map.get(key));
            }
        }
    }
}
```

```
Char[] arr = str.toCharArray();
Map<Character, Integer> m = new HashMap<C,Z>();
for(Char c : arr){
Integer count = m.get(c);
if(count == null){
    m.put(c,1);
}else{ m.put(c, ++count);}
}
```

O/P: r: 2
g: 2
m: 2

④ Check if two given strings are Anagram

two words contains same characters and same length.

eg: strp, mary ×
stop, tops ✓
army, mary ×

→
```
public class AnagramStrings{
    public static void main(String[] args){
        String str1 = "army";
        String str2 = "mary";
        char[] arr1 = str1.toLowerCase().toCharArray();
        char[] arr2 = str2.toLowerCase().toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        if(Arrays.equals(arr1, arr2)){
            sop("Given strings are anagram");
        }else{
            sop("Given strings are NOT anagram");
        }
    }
}
```

O/P:- Given strings are anagram

⑤ find first non-Repeated character of a given string

eg: swiss (1st, 2nd) programming (1st)

→ public class FirstNonRepeatedCharacter {

```
    public static void main(String[] args) {
        String str = "swiss";
        // String str = "programming";

        Map<Character, Integer> map = new LinkedHashMap<Character, Integer>();
        int count = 1;
        for (int i=0; i< str.length(); i++) {
            if (!map.containskey(str.charAt[i])) {
                map.put(str.charAt(i), count);
            } else {
                map.put(str.charAt(i), map.get(str.charAt(i))+1);
            }
        }

        // Printing first non-repeated character from map.
        for (Entry<Character, Integer> entry : map.entrySet()) {
            if (entry.getValue() == 1) {
                sop("First non-repeated character is: "+ entry.getKey());
                break;
            }
        }
    }
```

maintains insertion order

returns set view of map

∵ char[] arr = str.toCharArray();
arr ⇔ str.charAt[i]

o/p:- First non-repeated character is: p

```
String str = "swiss";
char[] arr = str.toCharArray();
Map<Character, Integer> m = new LinkedHashMap<Character, Integer>();
for(Character c: arr){
    Integer count = m.get(c);
    if(count == null){
        m.put(c, 1);
    }else{
        m.put(c, ++count);
    }
}
// Printing non repeated character
for(Character key: m.keySet()){
    if (m.get(k) == 1){
        sop("1st non-repeated character is:" + key);
    }
}
```

covered

⑥ How to Reverse Given String

→ public class StringReverse {

```
    public static void main(String[] args) {
        String str = "programming";
        // grimmargorp
        // #1 Using StringBuffer class
        StringBuffer sb = new StringBuffer(str);
        String result = sb.reverse().toString();
        sop(result);
        // #2 Without StringBuffer
        String result1 = "";
        for(int i=str.length()-1; i>=0; i--) {
            result1 = result1 + str.charAt(i);
        }
        sop(result1);
    }
}
```

o/p:- grimmargorp
grimmargorp

⑦ How to extract last four characters from a string

→ public class ExtractStringPart {
public static void main (String[] args) {

```
String str = "www.erostrum.com";
// www. -> firstfour
// .com -> lastfour
// erostrum
String firstFour = str.substring(0, 4); // end index is exclusive
```

Began index is inclusive

```
sop(firstFour);
String lastFour = str.substring(str.length() - 4, str.length());
sop(lastFour);
String website = str.substring(4, str.length() - 4);
sop(website);
}
}
```

o/p:- www.
.com
erostrum

⑧ Check the given string is palindrome or not using recursion

function calling itself.

ex: nitin, malayalam, abcx

→ public class PalindromeChecker {
public static void main (String[] args) {

```
String str = "nitin";
boolean result = isPalindrome(str);
sop(str + "IsPalindrome = " + result);
}
public static boolean isPalindrome(String str) {
    if (str == null) {
        return false;
    }
    if (str.length() <= 1) {
        return true;
    }
    String first = str.substring(0, 1); // first character
    String last = str.substring(str.length() - 1, str.length()); // last character
    if (!first.equals(last)) {
        return false;
    } else {
        return isPalindrome(str.substring(1, str.length() - 1)); // iti
    }
}
}
```

o/p:- nitin IsPalindrome = false

⑦ How to extract last four characters from a string

→ public class ExtractStringPart {

```
public static void main(String[] args) {
    String str = "www.erostrum.com";
    // www. -> firstfour
    // .com -> lastfour
    // erostrum
    String firstFour = str.substring(0, 4);  // end Index is exclusive
                                                Began index is inclusive
    sop(firstFour);

    String lastFour = str.substring(str.length() - 4, str.length());
    sop(lastFour);

    String website = str.substring(4, str.length() - 4);
    sop(website);
  }
}
```

o/p:- www.
.com
erostrum

⑧ Check the given string is palindrome or not using recursion

function calling itself.

ex: nitin, malayalam, abcx

→ public class PalindromeChecker {

```
public static void main(String[] args) {
    String str = "nitin";
    boolean result = isPalindrome(str);
    sop(str + " IsPalindrome = " + result);
  }
  public static boolean isPalindrome(String str) {
    if (str == null) {
        return false;
    }
    if (str.length() <= 1) {
        return true;
    }
    String first = str.substring(0, 1);  // first character
    String last = str.substring(str.length() - 1, str.length());  // last character
    if (!first.equals(last)) {
        return false;
    } else {
        return isPalindrome(str.substring(1, str.length() - 1));  // iti
    }
  }
}
```

o/p:- nitin IsPalindrome = false

(9) How to reverse words in a sentence

→
```
public class ReverseWords {
    public static void main (String[] args){
        String str = "I am working on my pc"; //expected: pc my on working am I
        String result = "";
        String[] arr = str.split ("\\s");  // can use " " or \\s to separate words by space
        for (int i=arr.length -1; i>=0 ; i--){
            result = result + arr[i] + " ";
        }
        sop (result);
    }
}
```

o/p:- pc my on working am I

(10) How to reverse each(every) word in a sentence

→
```
public class ReverseEachWord {
    public static void main(String[] args) {
        String str = "How are you"; //expected: woh era uoy
        String result = "";
        String[] arr = str.split ("\\s");
        for (String word : arr) {
            String reverseword = "";
            int j = word.length() - 1;
            while (j >= 0){
                char ch = word. charAt (j);
                reverseword = reverseword + ch;
                j--;
            }
            result = result + reverseword + " ";
        }
        sop (result);
    }
}
```

```
String str = "How are you";
String[] arr = str.split (" ");
String result = "";
for (int=0; i < arr.length; i++) {
char[] arr1 = arr[i]. toCharArray();
    String reverseword = "";
    for(int j = arr1.length-1; j>=0; j--){
        reverseword = reverseword + arr1[j];
    }
    result = result + reverseword + " ";
}
sop (result);
```

o/p:- woH era uoy

(11) How to find Largest and smallest element in an array

→
```
public class LargestSmallestElementInArray {
    public static void main (String[] args) {
        int[] arr = {5,6,-3,9,-25,234,7,0,2};
        int largest = arr[0];
        int smallest = arr[0];
        for (int i=1 ; i< arr.length; i++) {
            if (arr [i] > largest){
                largest = arr[i];
            }else if (arr[i] < smallest) {
                smallest = arr[i];
            }
        }
        sop(Arrays.toString(arr));
        sop("The largest element is : "+largest);
        sop("The smallest element is : "+ smallest);
    }
}
```

o/p:- [5,6,-3,9,-25,234,7,0,2]
The largest element is : 234
The smallest element is : -25

(12) Find the missing number in a array

→
```
public class MissingNumberFinder {
    public static void main (String[] args) {
        int[] numArr = {1,2,3,5,6,7,8,9,10};
        int sum=0;
        int expectedsum=0;
        for (int i=0; i< numArray.length; i++) {
            sum = sum + numArr[i];
        }
        for (int i=1; i<=10; i++) {
            expectedsum = expectedsum + i;
        }
        sop(Arrays.toString(numArr));
        sop("The missing number is: " + (expectedsum - sum));
    }
}
```

act: 51
exp: 55
missing 4 ✓

o/p:- [1,2,3,5,6,7,8,9,10]
The missing number is: 4

(13) How to find duplicates in an array using Brute force method

→
```
public class DuplicatesInArray {
    public static void main (String[] args) {
        int[] inputArr = {2,5,6,7,6,8,2,9};
        DuplicatesInArray.findDuplicateUsingBruteForce (inputArr);
    }
    private static void findDuplicateUsingBruteForce (int[] arr) {
        for (int i=0; i< arr.length; i++) {
            for (int j= i+1; j< arr.length; j++) {
                if (arr[i] == arr[j] {
                    sop("Duplicate element is: " + arr[i]);
                }
            }
        }
    }
}
```

o/p:- Duplicate element is: 2
Duplicate element is: 6

(14) Find duplicates in array using sorting method

2, 5, 6, 7, 6, 8, 2, 9
sort → 2, 2, 5, 6, 6, 7, 8, 9

→
```
public class DuplicatesInArray {
    public static void main (String[] args) {
        int[] inputArr = {2,5,6,7,6,8,2,9};
        DuplicatesInArray.findDuplicatesUsingSorting (inputArr);
    }
    private static void findDuplicatesUsingSorting (int[] arr) {
        Array.sort(arr);
        for (int i=0; i< arr.length -1; i++) {
            if (arr[i] == arr[i+1]) {
                sop("Duplicates element is: " + arr[i]); // o/p:- 2 & 6
            }
        }
    }
}
```

(15) Find duplicates in array using Hashset

* set will not allow duplicates

→
```
public class DuplicatesInArray {
    public static void main (String[] args) {
        int[] inputArr = { 2, 5, 6, 7, 6, 8, 2, 9, 8};
        DuplicatesInArray.findDuplicatesUsingSet (intputArr);
    }
    private static void findDuplicatesUsingSet (int[] arr) {
        Set<Integer> hs = new HashSet<Integer>();
        for (int x : arr) {
            if (!hs.add(x)) {
                sop("The duplicate element is : "+x); // o/p:- 2, 6, 8
            }
        }
    }
}
}
```

returns false if element already exists

(16) Find duplicates in array using Hashmap

→
```
public class DuplicatesInArray {
    public static void main (String[] args) {
        int[] inputArr = { 2, 5, 6, 7, 6, 8, 2, 9, 8};
        DuplicatesInArray.findDuplicatesUsingHashMap (inputArr);
    }
    private static void findDuplicatesUsingHashmap (int[] arr) {
        Map<Integer, Integer> hm = new HashMap<Integer, Integer>();
        for (int x : arr) {
            if (!hm.containsKey(x)) {
                hm.put(x, 1);
            } else {
                hm.put(x, get(x) + 1);
            }
        }
        for (Integer x : hm.KeySet()) {
            if (hm.get(x) > 1) {
                sop("Duplicate element is: "+x); // o/p:- 2, 6, 8
            }
        }
    }
}
}
```

o/p:- Duplicate element is: 2
Duplicate element is: 6
Duplicate element is: 8

(17) How to Remove Duplicate words from a given sentence

```
→ public class RemoveDuplicateWords {
    public static void main(String[] args) {
        // i/p = " I am am rahul rahul ";
        // o/p = " I am rahul ";
        String input = " I am am rahul rahul ";
        String[] inputArr = input.split(" ");
        // Set<String> myset = new HashSet<String>();
        Set<String> myset = new LinkedHashSet<String>();
        for (String x : inputArr) {
            myset.add(x);
        }
        Iterator itr = myset.iterator();
        while(itr.hasNext()) {
            sop(itr.next() + " ");
        }
    }
}
```

will not maintains order

not contains duplicates & mains order

// Iterator - to print elements of set

o/p:- I am rahul

(18) How to Reverse a number in java

If we divide any no. by 10

% - modulus - gives the last no.

/ - devide - Removes last and gives the rest of the no.

```
→ public class ReverseNumber {
    public static void main(String[] args) {
        // i/p: 1234   o/p: 4321
        sop(reverseNumber(1234));
    }
    public static int reverseNumber(int num) {
        int result = 0
        while(num > 0) {
            result = result * 10 + num % 10;
            num = num / 10;
        }
        return result;
    }
```

o/p:- 4321

(19) Fibonacci series

→ Addition of first 2 no.s gives the 3rd no.

```
→ public class Fibonacci {
    public static void main(String[] args) {
        // 0 1 1 2 3 5 8 13 21 ...
        int first = 0;
        int second = 1;
        int n = 10;
        for (int i = 1; i <= n; i++) {
            sop(first + " ");
            int third = first + second;
            first = second;
            second = third;
        }
    }
```

o/p:- 0 1 1 2 3 5 8 13 21 34

20 Factorial

$n=5$
result $= 5\times4\times3\times2\times1 = 120$
$n! = n(n-1)\ldots$

```
→public class Factorial {
    public static void main (String[] args) {
        int n = 5;
        int fact = 1;
        for (int i=1; i<=n; i++){
            fact = fact * i;
        }
        sop ("The factorial of "+n+" is = "+fact);
    }
}
```

o/p:- 120

21 Factorial Using Recursion ← a function calling it self

```
→ public class FactorialUsingRecursion {
    public static int factorialFinder (int n) {
        if (n<=1)
            return 1;
        else
            return n * factorialFinder(n-1);
    }
    public static void main (String[] args) {
        int n = 5;
        sop ("The factorial of "+n+" is = "+factorialFinder(n));
    }
}
```

o/p:- 120

22 Fibonacci Series Using Recursion

```
→ public class FibonacciSeries {
    static int n1=0, n2=1;
    public static void fibonacciFinder (int n) {
        if(n>0){
            sop(n1+" ");
            int n3 = n1+n2;
            n1 = n2;
            n2 = n3;
            fibonacciFinder(n-1);
        }
    }
    public static void main(String[] args) {
        int n=10;
        fibonacciFinder(n);
    }
}
```

o/p:- 0 1 1 2 3 5 8 13 21 34

23 Patterns Printing

→ public class ~~Diamond~~Patterns{

```
public static void main (String[] args){
    diamondPattern();
}
public static void diamondPattern(){
    Scanner sc = new Scanner(System.in);
    Sop("Enter N: ");
    int n = sc.nextInt();
    Sop("Enter Symbol: ");
    char c = sc.next().charAt(0);
    for(int i=1; i<=n; i++){
        for(int j=1; j<n-i; j++){
            sop(" "); // print()
        }
        for(int j=1; j<i*2-1; j++){
            sop(c);
        }
        sopln();
    }
    // Repeat again in reverse order
    for(int i=n-1; i>0; i--){
        for(int j=1; j<=n-i; j++){
            sop(" ");
        }
        for(int j=1; j<=i*2-1; j++){
            sop(c);
        }
        sopln();
    }
}
```

O/P:- Enter N: 5
Enter Symbol: *

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
 * * * * * * *
  * * * * *
   * * *
    *
```

→ Public static void hallowDiamond(){

```
    Scanner sc = new Scanner(System.in);
    sop("Enter N: ");
    int n = sc.nextInt();
    sop("Enter symbol: ");
    char c = sc.next().charAt(0);
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n+1-i; j++){
            sop("* ");
        }
        for(int k=1; k<(2*i-1); k++){
            sop(" ");
        }
        for(int j=1; j<n+1-i; j++){
            sop("* ");
        }
        sopln();
    }
    // second half
    for(int i=n-1; i>=1; i--){
        for(int j=n; j>=i; j--){
            sop("* ");
        }
        for(int k=1; k<(2*i-1); k++){
            sop(" ");
        }
        for(int j=n; j>=i; j--){
            sop("* ");
        }
        sopln();
    }
}
```

O/P:- Enter N: 5
Enter Symbol: *

```
* * * * * * * * *
* * * *     * * *
* * *       * * *
* *         * *
*           *
*           *
* *         * *
* * *       * * *
* * * * * * * * *
```

→ public static void leftHalfDiamond () {

```
    Scanner sc = new Scanner(System.in);
    sop ("Enter N: ");
int n = sc.nextInt ();
    sop ("Enter symbol: ");
    char c = sc.next().charAt(0);
    for (int i=1; i<=n; i++){
        for (int j=1; j<=n-i; j++){
            sop(" ");
        }
        for (int k=1; k<=i; k++){
            sop(" * ");
        }
        sopln();
    }
    //second half
    for (int i=2; i<=n; i++) {
        for (int j=1; j<i; j++){
            sop (" ");
        }
        for (int k=i; k<=n; k++){
            sop (" * ");
        }
        sopln ();
    }
}
```

O/P:- Enter N = 5
Enter symbol: *

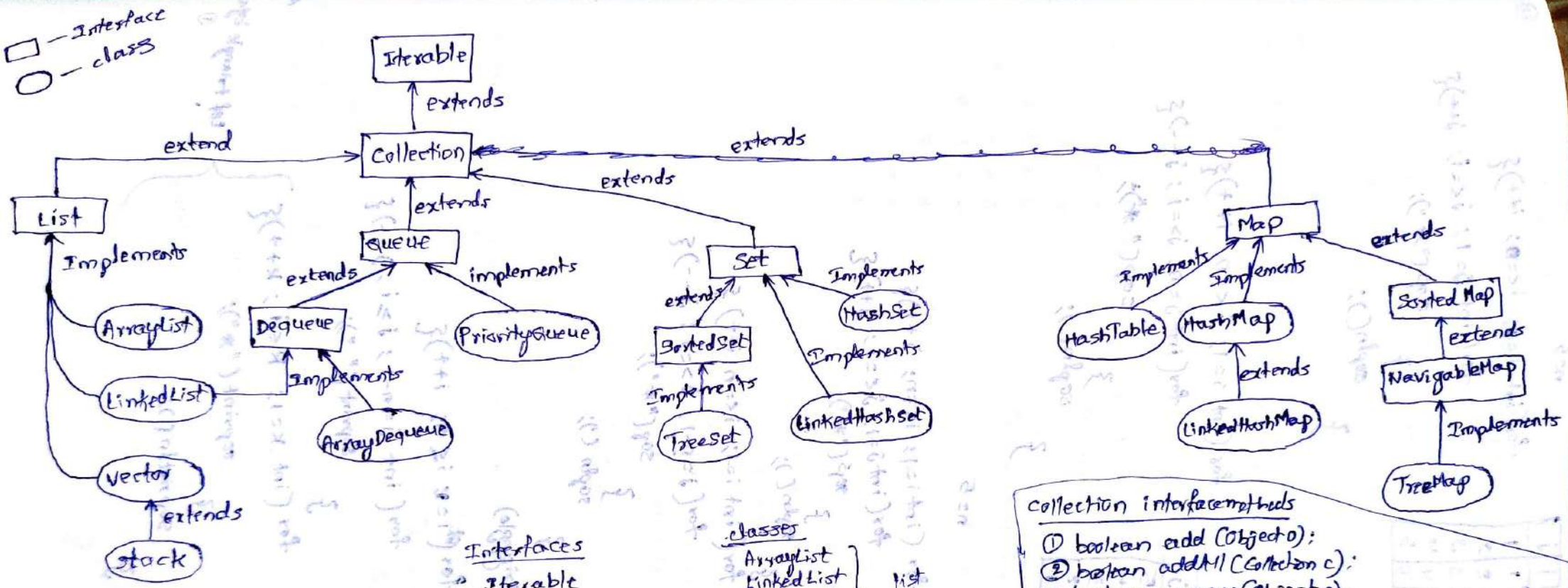
```
*   *
    * *
      * *
        * *
* * * * *
    * * * *
      * * *
        * *
          *
```

→ public static RightHalfDiamond () {

```
    Scanner sc = new Scanner (System.in);
    sop("Enter N: ");
    int n = sc.nextInt ();
    sop("Enter Symbol: ");
    char c = sc.next().charAt(0);
    for (int i=1; i<=n; i++){
        for (int j=1; j<=i; j++){
            sop("*");
        }
        sopln();
    //Print lower half
    for (int i=n-1; i>=1; i--){
        for (int j=1; j<=i; j++){
            sop("*");
        }
        sopln();
    }
}
```

O/P:-
Enter N: 5
Enter Symbol: *

```
*
* *
* * *
* * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```



Interfaces

- Iterable
- Collection
- List
- Queue
- DeQueue
- Set
- SortedSet
- Map
- SortedMap
- NavigableMap

classes

- ArrayList, LinkedList, Vector, stack } list
- PriorityQueue, ArrayDequeque } queue
- HashSet, LinkedHashSet, Treeset } set
- HashTable, HashMap, LinkedHashMap, TreeMap } Map

collection interface methods

1. boolean add (Object o);
2. boolean addAll (Collection c);
3. boolean remove(Object o);
4. boolean removeAll(Object o);
5. boolean retainAll (Collection c);
6. void clear();
7. boolean contains (Object o);
8. boolean containsAll (Collection c)
9. boolean isEmpty ();
10. int size();
11. Object[] toArray();
12. Iterator iterator();

Star Pattern Programs

① Left tringle

n=5

i=row
j=col

i \ j	1	2	3	4	5
1	*				
2	*	*			
3	*	*	*		
4	*	*	*	*	
5	*	*	*	*	*

i	j
1	1
2	2
3	3
4	4
5	5

n=5

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=i; j++){
        sop("* ");
    }
    sopln();
}
```

② Left Reverse tringle

i \ j	1	2	3	4	5
1	*	*	*	*	*
2	*	*	*	*	
3	*	*	*		
4	*	*			
5	*				

i	j
1	5
2	4
3	3
4	2
5	1

```
for (int i=1; i<=n; i++){
    for(int j=n; j>=i; j--){
        sop("*");
    }
    sopln();
}
```

n=5

③ ① + ②

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * * *
* * *
*
```

```
for (int i=1; i<=n; i++){
    for(int j=1; j<=i; j++){
        sop("*");
    }
    sopln();
}
for(int i=1; i<=n i++){
    for(j=n-1; j>=i; j--){
        sop("*");
    }
    sopln();
}
```

Right triangle: (mirror to left triangle) ①

```
    *
   * *
  * * *
 * * * *
* * * * *
```

n=5
i=1
j=5-1=4
k=1

```
for(int i=1; i≤n; i++){
    for( int j=n-1; j≥i; j--){
        Soprint(" ");
    }
    for (int k=1; k≤i; k++){
        soprint("*");
    }
    soprintln();
}
```

left triangle logic ①

⑤ Right Reverse triangle (mirror to ②)

n=5

*	*	*	*	*
-	*	*	*	*
-		*	*	*
			*	*
				*

```
- * * * * *
- - * * * *
- - - * * *
- - - - * *
- - - - - *
```

```
for (int i=1; i≤n; i++){
    for(int j=1; j≤i; j++){   // can start with j=2 to print exact root spaces.
        print(" ");
    }
    for(int k=n; k≥i; k--){
        print("*");
    }
    println();
}
```

②

⑥

```
        *
      * *
    * * *
  * * * *
* * * * *
  * * * *
    * * *
      * *
        *
```

④+⑤

```
for (int i=1; i<=n; i++){
    for(int j=n-1; j≥i; j--){
        print(" ");
    }
    for(int k=1; k<=i; k++){
        print("* ");
    }
    println();
}
for (int i=1; i≤n; i++){
    for(int j=1; j≤i; j++){
        print(" ");
    }
    for(int k=n-1; k≥i; k--){
        print("* ");
    }
    println();
}
```

⑦ Star Pyramid (Add space befor star in ④ then star pyramid will print)

Method ①:

```
    *
  * *
* * *
* * * *
* * * *
```

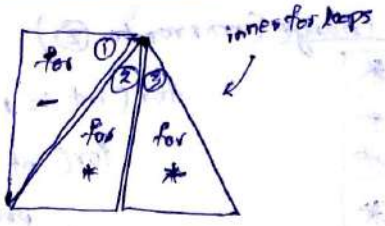
```
for (int i=1; i≤n; i++){
    for (int j=n-1; j≥i; j--){
        print(" ");
    }
    for (int k=1; k≤i; k++){
        print("_*");
    }
    println();
}
```

Method ②: Using if else

```
for (int i=1; i≤n; i++){
    for (int j=n-1; j≥1; j--){
        if(j>i){
            print(" ");
        }
        else{
            print("_*");
        }
    }
    println();
}
```

⑧

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```



n=5

Method ①:

```
for(int i=1; i≤n; i++){
    for(int j=n; j≥i; j--){
        print(" ");
    }
    for(int k=1; k≤i; k++){
        print("*");
    }
    for(int l=2; l≤i; l++){
        print("*");
    }
    println();
}
```

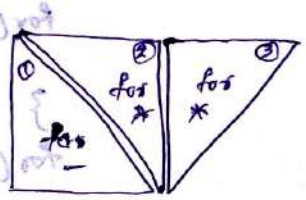
j=n (or) j=n-1

Method ②: Remove 3rd inner for loop and change 2nd for loop condition to k<(i*2)

```
for(int i=1; i≤n; i++){
    for(int j=n; j≥i; j--){
        print(" ");
    }
    for(int k=1; k<(i*2); k++){
        print("*");
    }
    println();
}
```

⑨

```
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```



Method ①:

```
for(int i=1; i<=4; i++){
    for(int j=1; j≤i; j++){
        print(" ");
    }
    for(int k=4; k≥i; k--){
        print("*");
    }
    for(int l=3; l≥i; l--){
        print("*");
    }
    println();
}
```

Method ②:

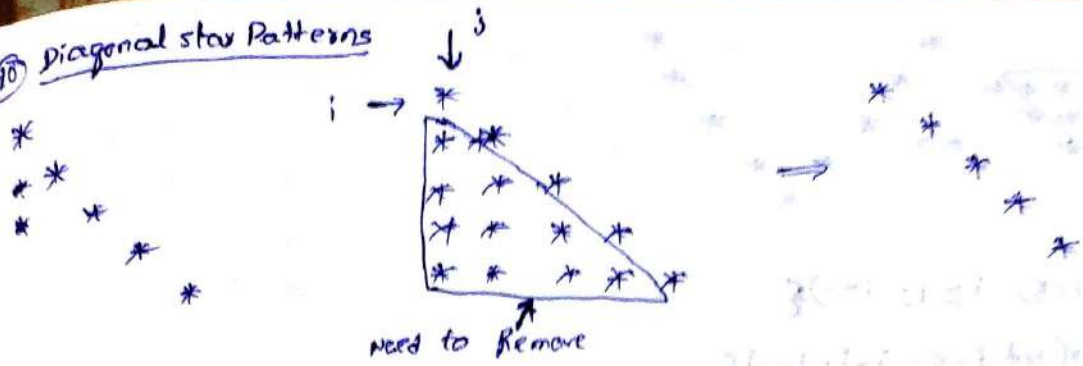
```
for(int i=1; i≤4; i++){
    for(int j=1; j≤i; j++){
        print(" ");
    }
    for(int k=9; k>(i*2); k--){
        print("*");
    }
    println();
}
```

Method ③:

```
for(int i=5; i≥1; i--){
    for(int j=5; j>i; j--){
        print(" ");
    }
    for(int k=1; k<(i*2); k++){
        print("*");
    }
    println();
}
```

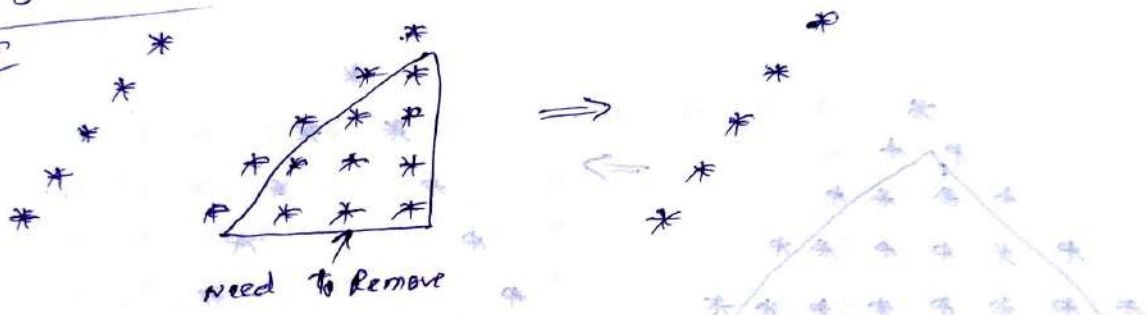
(10) Diagonal star Patterns

I




```
for(int i=1; i<=5; i++){
    for(int j=1; j<=i; j++){
        if (i>=2 && j<=i-1){
            print(" ");
        }
        else{ print("*");
        }
    }
    println();
}
```

II

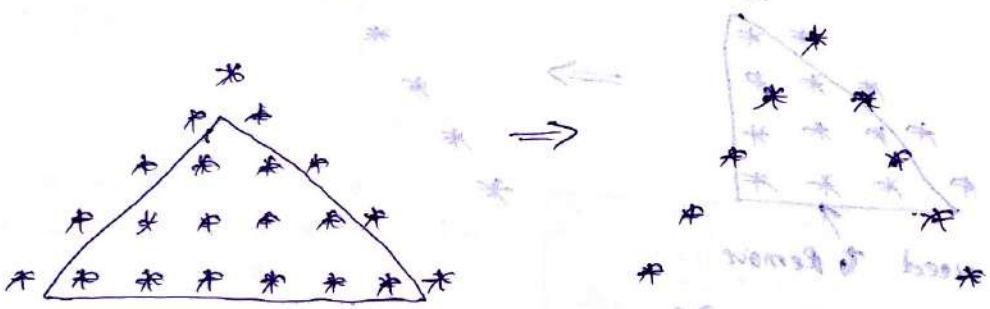



```
for(int i=1; i<=5; i++){
    for(int j=4; j>i; j--){
        print(" ");
    }
    for(int k=1; k<=i; k++){
        if(i>=2 && k>1){
            print(" ");
        }
        else{ print("*");
        }
    }
    println();
}
```

III

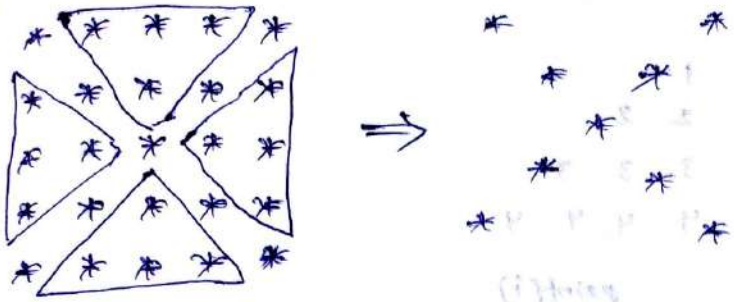
```
for(int i=5; i≥1; i--){
    for(int j=5; j>i; j--){
        print(" ");
    }
    for(int k=1; k<(i*2); k++){
        if(k>1 && k<(i*2)-1){
            print(" ");
        }
        else{ print("*");
        }
    }
    println();
}
```

IV



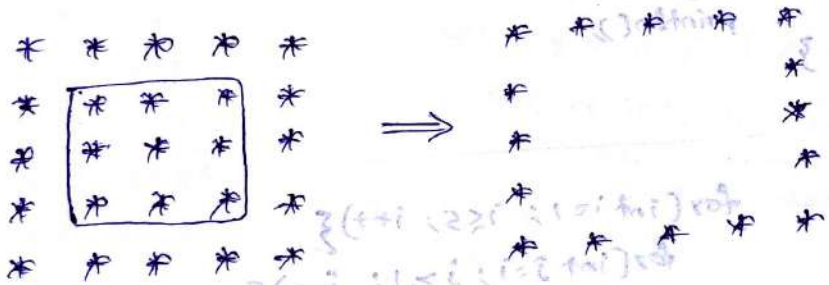
```
for(int i=1; i≤5; i++){
    for(int k=5; k≥i; k--){
        print(" ");
    }
    for(int j=1; j<(i*2); j++){
        if(j>1 && j<(i*2)-1){
            print(" ");
        }
        else{
            print("*");
        }
    }
    println();
}
```

(11)



```
for(int i=0; i<5; i++){
    for(int j=0; j<5; j++){
        if(i==j || i+j==5-1){
            print("*");
        }
        else{
            print(" ");
        }
    }
    println();
}
```

(12)



```
for(int i=1; i≤5; i++){
    for(int j=1; j≤5; j++){
        if(i>2 && j>2 && i≤4 && j≤4){
            print(" ");
        }
        else{
            print("*");
        }
    }
    println();
}
```

Number Pattern Programs

①

```
*
* *
* * *
* * * *
```

print("*")

```
1
1 2
1 2 3
1 2 3 4
```

print(j)

```
1
2 2
3 3 3
4 4 4 4
```

print(i)

```
for(int i=1; i≤4; i++){
    for(int j=1; j≤i; j++){
        print(i);
    }
    println();
}
```

②

```
1
2 3
4 5 6
7 8 9 10
```

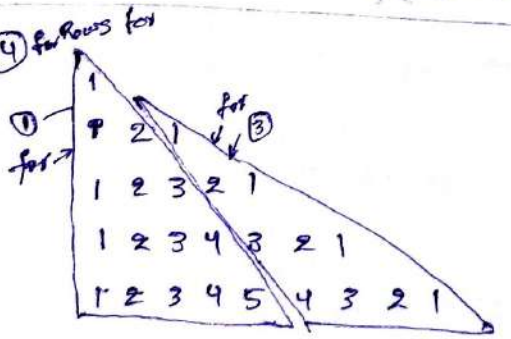
```
int count=0;
for(int i=1; i≤4; i++){
    for(int j=1; j≤i; j++){
        count=count+1;
        print(count+" ");
    }
    println();
}
```

③

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

```
for(int i=1; i≤5; i++){
    for(int j=i; j≥1; j--){
        print(j+" ");
    }
    println();
}
```

④ for rows for




```
for(int i=1; i≤5; i++){
    for(int j=1; j≤i; j++){
        print(j+" ");
    }
    for(int k=i-1; k≥1; k--){    // k=i-1
        print(k+" ");
    }
    println();
}
```

⑤

```
* * * * *      5 4 3 2 1
* * * *        5 4 3 2
* * *          5 4 3
* *            5 4
*              5
```

print("*") print(j)

```
for(int i=1; i≤5; i++){
    for(int j=5; j≥i; j--){
        print(j+" ");
    }
    println();
}
```

⑥

1
2 6
3 7 10
4 8 11 13
5 9 12 14 15
4 3 2 1

```
for(int i=1; i≤5; i++){
    int no=i;
    for(int j=1; j≤i; j++){
        print( no+" ");
        no=no+5-j;
    }
    println();
}
```

i=1
no=1
j=1
1+5-1=5

⑦

I
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15

```
int count=0;
for(int i=1; i≤5; i++){
    for(int j=1; j<3; j++){
        count=count+1;
        print(count+" ");
    }
    println();
}
```

II Snake number pattern:

6 – rows
5 – columns

1 2 3 4 5
10 9 8 7 6
11 12 13 14 15
20 19 18 17 16
21 22 23 24 25
30 29 28 27 26

```
int count=0;
for(int i=1; i<6; i++){
    if(i%2 != 0){ // odd no. of row
        for(int j=1; j<5; j++){
            count=count+1;
            print(count+" ");
        }
    }
    else{ int temp=count+1;        // j=count+5
        for(int j=count+5; j≥temp; j--){
            count=count+1;
            print(j+" ");
        }
    }
    println();
}
```

① Difference between final, finally and finalize

final:

→ final is a modifier applicable for classes, methods and variables.
If a class is declared as final then we can't extend that class.
i.e we can't create child class for that class.

→ If a method declared as final then we can't override that method in the child class.

→ If a variable declared as final then it will become constant and we can't perform reassignment for that variable

finally:

→ finally is a block always associated with try catch to maintain cleanup code.

```
try {
    // risky code
}
catch (X e){
    // Handling code
}
finally {
    // cleanup code  eg. closing database connection
}
```

finalize():

→ finalize() is a method which is always invoked by garbage collector just before destroying an object to perform cleanup activities.
(cleanup activities - closing database connection).

Note:

→ finally meant for cleanup activities related to try block.
where as finalize() meant for cleanup activities related to object.

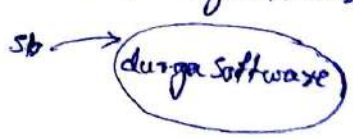
② Differences between String and StringBuffer

String:

```
String s = new String("durga");
s.concat("software");
sopln(s); // durga
```

StringBuffer:

```
StringBuffer sb = new StringBuffer("durga");
sb.append("software");
sopln(sb); // durgaSoftware
```



→ String objects are immutable where as StringBuffer objects are mutable.

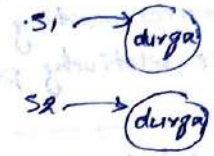
→ Once we create a String object we can't perform any changes in the existing object. If we try to perform any changes, with those changes a new object will be created. This non-changeable nature is nothing but immutability of the string object.

→ Once we create a StringBuffer object we can perform any type of changes in the existing object. This changeable nature is nothing but mutability of the StringBuffer object.

③ Difference between "==" operator and ".equals()" method in java

→ In general we use == operator for reference/address comparison where as .equals() method for content comparison.

```
String s1 = new String("durga");
String s2 = new String("durga");
sopln(s1==s2); // false
sopln(s1.equals(s2)); // true
```



Note:

→ .equals() method present in object class also meant for reference/address comparison only, based on our requirement we can override for content comparison.

→ In String class, all wrapper class and collection classes .equals() method is overriden for content comparison.

④ What are the various modifiers available in java

I → The modifiers which are applicable for inner classes but not for outer classes — private, protected, static

→ The modifiers which are applicable for classes but not for interfaces — final

→ The modifiers which are applicable for classes but not for enums — final, abstract

→ The modifiers which are applicable only for methods and which can't use anywhere else — native

→ The only modifiers which are applicable for constructor are — public, private, protected, default

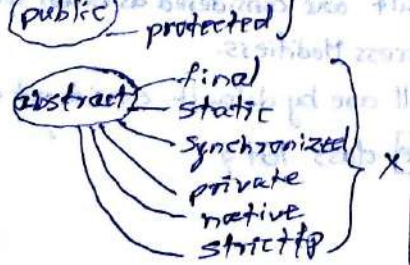
→ The only modifier applicable for local variables — final

II → The interface which is declared inside a class is always static whether we are declaring or not.

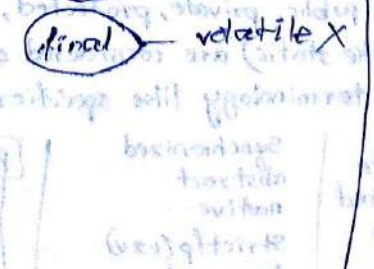
→ The interface which is declared inside interface is always public and static whether we are declaring or not

Illegal combinations:

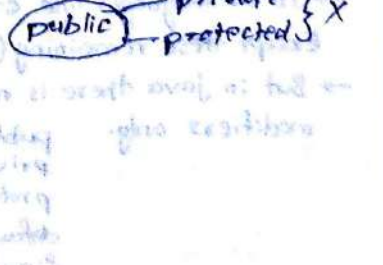
At Methods level:



At variables level:



At class level:



5) Difference between StringBuffer and StringBuilder

→ StringBuilder is exactly same as StringBuffer (Including methods and constructors) except the following differences.

String Buffer	StringBuilder
→ Every method present in StringBuffer is Synchronized.	→ No method is synchronized in StringBuilder
→ At a time only one thread is allowed to operate on StringBuffer object. Hence StringBuffer object is thread safe.	→ At a time multiple threads are allowed to operate on StringBuilder object and hence StringBuilder object is not thread safe.
→ It increases waiting time of threads and hence relatively performance is low.	→ Threads are not required to wait to operate on StringBuilder object and hence relatively performance is high.
→ Introduced in 1.0 version	→ Introduced in 1.5 version.

6) Difference between Interface vs abstract class vs concrete class

→ If we don't know anything about implementation, we just have requirement specification (100% Abstraction) then we should go for interface.

ex:- Servlet

→ If we are talking about implementation but not completely (partial implementation) then we should go for Abstract class.

ex:- GenericServlet & HTTPServlet

→ If we are talking about implementation completely and ready to provide service then we should go for concrete class.

ex:- MyOwnServlet

Interface (I) → Servlet (I) → Plan

Abstract class → GenericServlet & HTTPservlet → Partially completed Building

concrete class → MyOwnServlet → Fully completed Building

→ In reality we require (use) only concrete classes, Interface and Abstract class are intermediate stages to get concrete class.

8) Access Specifiers vs Access Modifiers

→ In old languages like c++ public, private, protected, default are considered as Access specifiers Except these remaining (like static) are considered as Access Modifiers

→ But in java there is no terminology like specifiers, all are by default considered as modifiers only.

public
private
protected
default
final
static
	Synchronized
	abstract
	native
	strictfp (1.2v)
	transient
	volatile
		private class Test {
		}
		↳ compile time error:
		(modifier) private not allowed here

⑥ String vs StringBuffer vs StringBuilder

→ If the content is fixed and won't change frequently then we should go for String.

→ If the content is not fixed and keep on changing but Thread Safety is required then we should go for StringBuffer.

→ If the content is not fixed and keep on changing and Thread safety is not required then we should go for StringBuilder.

⑨ Difference between Interface and Abstract class

Interface	Abstract class
→ If we don't know anything about implementation just we have requirement specification then we should go for interface.	→ If we are talking about implementation but not completely (partial implementation) then we should go for Abstract class.
→ Inside interface every method is always public and abstract whether we are declaring or not. Hence interface is also considered as 100% pure Abstract class.	→ Every method present in abstract class need not be public and Abstract. In addition to abstract methods we can take concrete methods also.
→ We can't declare interface method with the following modifiers, public → private, protected; Abstract → final, static, synchronized, native, strictfp	→ There are no restrictions on Abstract class method modifiers.
→ Every variable present inside interface is always public, static and final whether we are declaring or not.	→ The variables present inside Astract class need not be public, static and final.
→ We can't declare interface variables with the following modifiers. private, protected, tranciert, volatile	→ There are not restrictions on Abstract class variable modifiers.
→ For interface variables compulsory we should perform initialization at the time of decleration otherwise we will get compile time error.	→ For Abstract class variables it is not required to perform initialization at the time of declaration.
→ Inside interface we can't declare instance and static blocks. Otherwise we will get compile time error.	→ Inside Abstract class we can declare instance and static blocks.
→ Inside interface we cannot declare constructors.	→ Inside Abstract class we can declare constructor, which will be executed at the time of child object creation.

⑩ Explain System.out.println();

⇒
```
class Test{
    static String s="Java";
}
Test.s.length();
```

→ Test is a class name

→ 's' is a static variable present in Test class of type String

→ length() is a method present in String class.

⇒
```
class System{
    static PrintStream out;
}
System.out.println("Hello");
```

→ System is a class present in java.lang package.

→ 'out' is a static variable present in system class of type PrintStream.

→ Println() is a method present in PrintStream class.

(11) Explain about public static void main(String[] args); Part-(1)

→ Whether class contains main() method or not and whether main() method is declared according to requirement or not these things won't be checked by compiler. At runtime, JVM is responsible to check these things.

→ At runtime if JVM is unable to find required main() method then we will get runtime exception saying NoSuchMethodError: main

```
class Test {

}
javac Test.java
java Test
RuntimeException: NoSuchMethodError: main
```

→ public → To call by JVM from anywhere

static → without existing object also JVM has to call this method & main method no way related to any object

void → main() method won't return anything to JVM

main(String[] args)
- main → this is name which is configured inside JVM
- (String[] args) → command line arguments.

→ The above syntax is very strict if we perform any change we will get runtime exception saying NoSuchMethodError: main

→ Even though the above syntax is very strict the following changes are acceptable.

→ We can declare "String[]" in any acceptable form.

```
main(String[] args)
main(String []args)
main(String args[])
```

→ Instead of 'args' we can take any valid java identifier

→ we can replace String[] with var arg parameter

main(String[] args) ==> main(String... args)

→ We can declare main() method with the following modifiers also.

final
synchronized
strictfp

→
```
class Test {
    final static synchronized strictfp public void main (String... durga) {
        sopln("valid main");
    }
}
javac Test.java
java Test
```

o/p: valid main

→ which of the following are valid main method declarations?

1) public static void main(String[] args) ✗
2) public static (m) main(String[] args) ✗
3) public static void main(String args) ✗
4) public final synchronized strictfp void main(String[] args) ✗ (∵ static missing)
5) public static final synchronized strictfp void main(String[] args) ✓
6) public static void main(String... args) ✓

→ In which of the above cases we will get compile time error?

- we won't get compile time error anywhere but at runtime we will get exception in all cases except last two.

Part ②

case 1:

→ Overloading of main method is posible but JVM will always call String[] argument main method only.

→ the other overloaded method we have to call explicitly then it will be executed just as a normal method call.

```
class Test {
    public static void main (String[] args) {
        sopln ("String[]");
    }
    // overloaded methods
    public static void main (int[] args) {
        sopln (" int[]");
    }
}
```

o/p:- String[]

case 2:

→ Inheritance concept applicable for the main method. Hence while executing child class if child class doesn't contain main method then parent class main method will be executed.

```
class P{
 public static void main (String[] args) {
    println ("parent main");
 }
}
class C extends P{
}
```

javac P.java

P.class C.class

java P
o/p:- Parent main

java C
o/p:- Parent main

case 3:

→ It seems overriding concept applicable for main method but it is not overriding it is method hiding.

```
class P{
 public static void main (String[] args) {
    sopln ("parent main");
 }
}
class C extends P{
 public static void main (String[] args) {
    sopln ("child main");
 }
}
```

It is method hiding but not overriding.

java C
o/p:- child main
java P
o/p:- Parent main

Note:

→ For main method inheritance and overloading concepts are applicable but overriding concept is not applicable, instead of overriding method hiding concept is applicable.

Part ③ 1.7 Version Enhancements w.r.t main() method

case 1 :

→ Until 1.6 version if the class doesn't contain main() method then we will get runtime exception saying no such method error. But from 1.7 version onwards instead of NoSuchMethodError we will get more meaningful error information.

```
class Test{

}
```

1.6 version	1.7 version
javac test.java	javac Test.java
java Test	java Test
RE: NoSuchMethodError: main	Error: Main method not found in class Test, please define main method as public static void main(String args)

case 2 :

→ From 1.7 version onwards to run a java program main method is mandatory. Hence, even though class contains static blocks they won't be executed if the class doesn't contain main() method.

eg:-1)

```
class Test{
    static{
        sopln("static block");
    }
}
```

1.6 version	1.7 version
javac Test.java	javac Test.java
java Test	java Test
o/p:- static block	Error: main method not found in class
RE: NoSuchMethodError: main	

eg:-2)

```
class Test{
    static{
        sopln("static block");
        System.exit(0);
    }
}
```

1.6 version	1.7 version
javac Test.java	javac Test.java
java Test	java Test
o/p:- static block	o/p: Error: main method not found in class.

eg:- 3)

→ If the class contains main() method whether it is 1.6 or 1.7 version there is no change in execution sequence.

eg: 3)
```
class Test{
    static{
        sopln("static block");
    }
    public static void main(String[] args){
        sopln("main method");
    }
}
```

1.6 version	1.7 version
javac Test.java	javac Test.java
java Test	java Test
o/p:- static block main method	o/p:- static block main method

→ without writing main() method is it possible to print some statements to console?

A: Yes we can print by using static block.
But this rule is applicable until 1.6 version only, from 1.7 version onwards main() method is mandatory to print some statements to console

(12) Differences between Overloading & Overriding

Part ①

Overloading:

→ Two methods are said to be overloaded if and only if both methods having the same name but different argument types.

```
class Test{
    public void m1(int i){
    }
    public void m(long i){
    }
}
```
} overloaded methods

Overriding:

→ What ever methods parent has, by default available to the child through inheritance. Some times child may not satisfy with parent method implementation. Then child is allow to re define that method based on it's requirement. This process is called overriding.

→ The parent class method which is overridden is called overridden method.

→ The child class method which is overridden is called overriding method.

```
class P{
    public void property(){
        sopln("cash + Land + Gold");
    }
    public void marriage(){
        sopln("Subbalakshmi");
    }
}
```
overridden method

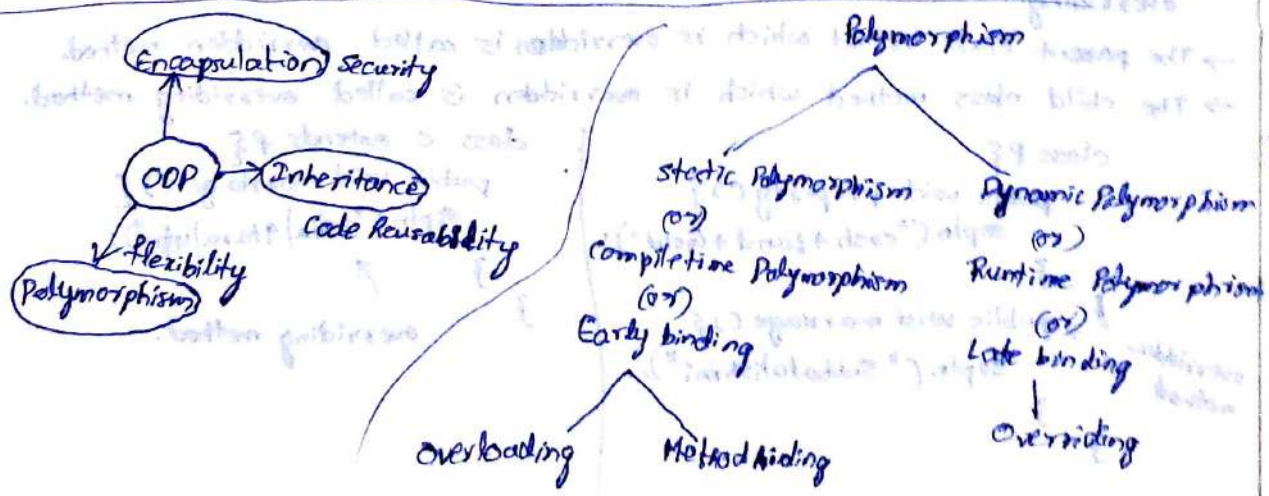
```
class C extends P{
    public void marriage(){
        sopln("3sba/9tara/4fre");
    }
}
```
overriding method.

Property	Overloading	Overriding
① Method names	Must be same	Must be same
② Argument Types	Must be different (at least order)	Must be same (Including order)
③ Method signatures	Must be different	Must be same
④ Return type	No restrictions	Must be same but this rule is applicable untill 1.4 version only. From 1.5 onwards co-variant return types are allowed.
⑤ private, static & final methods	Can be overloaded	Can'nt be overridden
⑥ Access modifiers	No Restrictions	We can't reduce scope of access modifier but we can increase.
⑦ throws clause	No Restrictions	If child class method throws any checked exception, compulsorylly parent class method should throw the same checked exception. otherwise we will get compile time error but there are no restrictions for unchecked exceptions.
⑧ Method Resolution	Always takes care by compiler based on reference type.	Always takes care by JVM based on Runtime object.
⑨ Also known as	Compile time polymorphism or static polymorphism or early binding.	Runtime polymorphism, dynamic polymorphism or late binding.

Note :

→ In overloading we have to check only method names (must be same) and argument types (must be different) except this the remaing like return types, access modifiers etc. are not required to check.

But in overriding everything we have to check like method names argument types, return types, access modifiers, etc.



(13) Difference between Parent p = new Child(); and Child c = new Child();

Child c = new Child();	Parent p = new Child();
→ If we know exact runtime type of object then we should use this approach.	→ If we dont know exact runtime type of object then we should use this approach. (Polymorphism).
→ By using child reference we can call both parent and child class methods.	→ By using parent reference we can call only methods available in parent class and child specific methods we can't call.
→ We can use child reference to hold only for that particular child class object only.	→ We can use parent reference to hold any child class object.

(14) Various possible combinations of try-catch-finally

→ Whenever we are writing try block compulsorily we should write catch or finally. That is 'try' without catch or finally is invalid syntax.

→ Whenever we are writing catch block compulsorily we should write try block. that is 'catch' without try is invalid.

→ Whenever we are writing finally block compulsorily we should write try block. That is 'finally' without try is invalid.

→ In try catch finally, order is important.

→ 'try' with multiple catch blocks is valid but the order is important, compulsorily we should take from child to parent. by mistake if we are trying to take from parent to child then we will get compile time error.

→ If we are defining two catch blocks for the same exception we will get compile time error.

→ We can define try-catch-finally within the try, with in the catch and within the finally blocks. Hence nesting of try-catch-finally is valid.

→ For try-catch-finally curly braces are mandatory.

① try{ } catch(x e){ }	② try{ } catch(x e){ } catch(x e){ } CE: exception x has already been caught.	③ try{ } catch(x e){ } catch(y e){ }	④ try{ } catch(Exception e){ } catch(ArithmeticException e){ } (AE) CE: exception java.lang.AE has already been caught
⑤ try{ } catch(AE e){ } catch(Exception e){ }	⑥ try{ } CE: try without catch or finally	⑦ catch(x e){ } CE: catch without try	⑧ finally{ } CE: finally without try

(9)
```
try{
}
finally{
}
```

(10)
```
try{
}
sopln("Hello");
catch(x e){
}
```
CE: try without catch or finally

CE: catch without try

(11)
```
try{
}
catch(x e){
}
sopln("Hello");
catch(y e){
}
```
CE: catch without try

(12)
```
try{
}
catch(x e){
}
sopln("Hello");
finally{
}
```
CE: finally without try

(13)
```
try{
}
catch(x e){
}
try{
}
finally{
}
```

(14)
```
try{
}
finally{
}
catch(x e){
}
```
CE: catch without try

(15)
```
try{
}
catch(x e){
}
finally{
}
finally{
}
```
CE: finally without try

(16)
```
try{
  try{
  }
  catch(x e){
  }
  finally{
  }
}
catch(x e){
}
```

(17)
```
try{
}
catch(x e){
  try{
  }
  finally{
  }
}
```

(18)
```
try{
}
catch(x e){
}
finally{
}
try{
}
finally{
}
}
```

(19)
```
try{
  try{
  }
}
catch(x e){
}
```
CE: try without catch or finally

(20)
```
try{
  sopln("Hello");
catch(x e){
}
```

(21)
```
try{
}
catch(x e)
  sopln("catch");
```

(22)
```
try{
}
catch(x e){
}
finally
  sopln("finally");
```

(15) Differences between Exception and Error

Throwable (class) → Exception, Error

→ Throwable acts as root for java exception hierarchy

Exception:

→ Most of the cases Exceptions are caused by our program and these are recoverable

ex:-
→ For example if our program requirement is to read data from a remote file locating at London. At runtime if the London file is not available then we will get FileNotFoundException.

→ If FileNotFoundException occurs then we can provide a local file and rest of the program will be continued normally.

```
try{
    // Read data from a remote file location at London
} catch (FileNotFoundException e){
    // use local file & continue rest of the program normally
}
```

Errors:

→ Most of the times errors are not caused by our program, these are due to lack of system resources.

→ Errors are non recoverable.

ex:- → For Example if OutOfMemory error occurs being a programmer we can't do anything and program will be terminated abnormally.
→ System admin or server admin is responsible to increase heap memory.

(16) Difference between checked Exceptions and Unchecked Exceptions

Checked Exceptions:

→ The Exceptions which are checked by compiler for smooth execution of the program at runtime are called checked Exceptions.

ex:- FileNotFoundException ✓
HallTicketMissingException
PenNotWorkingException.

→ In the case of checked exceptions compiler will check whether we are handling exception or not, if the programmer not handling then we will get compile time error.

Unchecked Exceptions:

→ The exceptions which are not checked by compiler are called unchecked exceptions.

Example: NullPointerException ✓
ArithmeticException ✓
BombBlastException

→ In case of unchecked exceptions compiler won't check whether the programmer handling exception or not.

Note:

→ Whether exception is checked or unchecked, compulsorily it will occur only at Runtime. There is not chance of occuring any exception at compile time.

→ Runtime exception and it's child classes, error and it's child classes are unchecked, except this remaining are checked exceptions.

(17) Differences between fully checked and partially checked exceptions

→ A checked exception is said to be fully checked if and only if all of its child classes are also checked.

ex:- IOException ✓
InterruptedException. ✓

→ A checked exception is said to be partially checked exception if and only if some of its child classes are unchecked.

ex:- Exception

Note:

→ The only possible partially checked exceptions in Java are
1. Exception ✓
2. Throwable ✓

(18) Control flow in try-catch (Java Exception Handling)

```
try {
    statement 1;
    statement 2;
    statement 3;
}
catch (X e) {
    statemement 4;
}
statement 5;
```

case 1: If there is no exception
1, 2, 3, 5 Normal Termination

case 2: If an exception raised at statement 2 and corresponding catch block matched.
1, 2, 4, 5 Normal Termination

case 3: If an exception raised at statement 2 and corresponding catch block not matched
1, Abnormal termination

case 4: If an exception raised at statement 4 or statement 5
- It is always abnormal termination.

Conclusions

→ Within try block if anywhere exception raised then rest of try block won't be executed even though we handled that exception. Hence length of try block should be as less as posible. And we have to take only risky code within try block.

→ In addition to try block there may be a chance of raising an exception inside catch and finally blocks.

→ If any statement raises an exception, which is not a part of try block then it is always Abnormal termination.

(19) Control flow in try-catch-finally (Java Exception Handling)

```
try{ stmt 1;
     stmt 2;
     stmt 3;
}
catch (X e){
     stmt 4;
}finally{
     stmt 5;
}
stmt 6;
```

Case 1: If there is no exception
1, 2, 3, 5, 6. normal termination

case 2: If an exception raised at stmt 2 and corresponding catch block matched
1, 4, 5, 6 normal termination

case 3: If an exception raised at stmt 2 and corresponding catch block not matched
1, 5 abnormal termination

case 4: If an exception raised at stmt 4
It is always Abnormal termination but before Abnormal termination finally block will be executed.

case 5: If an exception raised at stmt 5 or 6
It is always abnormal termination.

(20) Differences between Arrays and Collections

Arrays	Collections
→ Arrays are fixed in size	→ Collections are growable in nature i.e based on our requirement we can increase or decrease the size.
→ Wrt memory arrays are not recommended to use.	→ Wrt memory collections are recommended to use.
→ Wrt performance arrays are recommended to use	→ wrt performance collections are not recommended to use.
→ Arrays can hold only homogeneous data type elements.	→ Collections can hold both homogeneous and heterogeneous elements.
→ There is no underlying data structure for arrays and hence readymade method support is not available.	→ Every collections class is implemented based on some standard data structure. Hence readymade method support is available for every requirement.
→ Arrays can hold both primitives and object types	→ Collections can hold only objects but not primitives.

(21) Difference between List and Set

List	Set
→ Duplicates are allowed	→ Duplicates are not allowed
→ Insersion order preserved	→ Insertion order not preserved.

(22) Difference between Collection and collections

→ Collection is an interface which can be used to represent a group of individual objects as a single entity.

→ Collections is an utility class present in java.util.package to define several utility methods (like sorting, searching...) for collection objects.

(23) Difference between ArrayList and Vector

ArrayList	Vector
→ Every method present in ArrayList is non synchronize.	→ Every method present in LinkedList is synchronize.
→ At a time multiple threads are allowed to operate on ArrayList object and hence ArrayList is not thread safe	→ At a time only one thread is allowed to operate on vector object is thread safe
→ Threads are not required to wait to operate on ArrayList, hence reletively performance is high.	→ Threads are required to wait to operate on vector object and hence performance is relatively low.
→ Introduced in 1.2 version And it is non legacy class	→ Introduced in 1.0 version and it is a legacy class.

(24) Difference between ArrayList and LinkedList

ArrayList	LinkedList
→ It is the best choice if our frequent operation is retieval.	→ It is the best choice if our frequent operation is insertion and deletion
→ Array list is the worst choice if our frequent operation is insertion and deletion.	→ LinkedList is the worst choice if our frequent operation is retrieval operation.
→ Underlying data structure for ArrayList is resizable or growable.	→ Underlying data structure is Double LinkedList.
→ ArrayList implements Random Access Interface	→ LinkedList doesn't implement RandomAccess interface.

Note: → Both are list only, in both cases duplicates are allowed, in both cases insertion order is preserved, in both cases serializable clonable is there.

(25) How to get Synchronized version of ArrayList

→ By defaut ArrayList object is non-synchronized but we can get synchronized version of ArrayList by using Collection class synchronizedList() method.

```
public static List synchronizedList(List l);
```

Non-synchronized:

```
ArrayList l1 = new ArrayList();
```

Synchronized:

```
List l = Collections.synchronizedList(l1);
```

→ similarly we can get Synchronized version of Set, Map objects by using the following methods of Collections class.

```
public static Set synchronizedSet(Set s);
public static Set synchronizedMap(Map m);
```

(26) Comparison of three cursors of java

Property	Enumeration	Iterator	ListIterator
Applicable for	only legacy classes	Any collection classes	only List classes
Movement	only forward direction (single direction)	only forward direction (single direction)	Both forward and backward direction (bidirectional)
Accessibility	only read access	Both read and remove	Read, Remove, replace, addition of new objects
How to get it?	By using elements() method of vector class.	By using iterator() method of Collection interface.	By using listIterator() method of List interface.
Methods	2 methods hasMoreElements(), nextElement()	3 methods hasNext(), next(), remove()	9 methods (see next page)
Is it legacy	"yes" (1.0v)	"no" (1.2v)	"no" (1.2v)

(27) Differences between comparable and comparator

comparable	comparator
→ It is meant for default natural sorting order	→ It is meant for customized sorting order.
→ Present in java.lang. package	→ Present in java.util package.
→ This interface defines only one method compareTo()	→ This interface defines two methods compare() and equals()
→ All wrapper classes and string class implements comparable interface	→ The only implemented classes of Comparator are Collator and RuleBasedCollator.

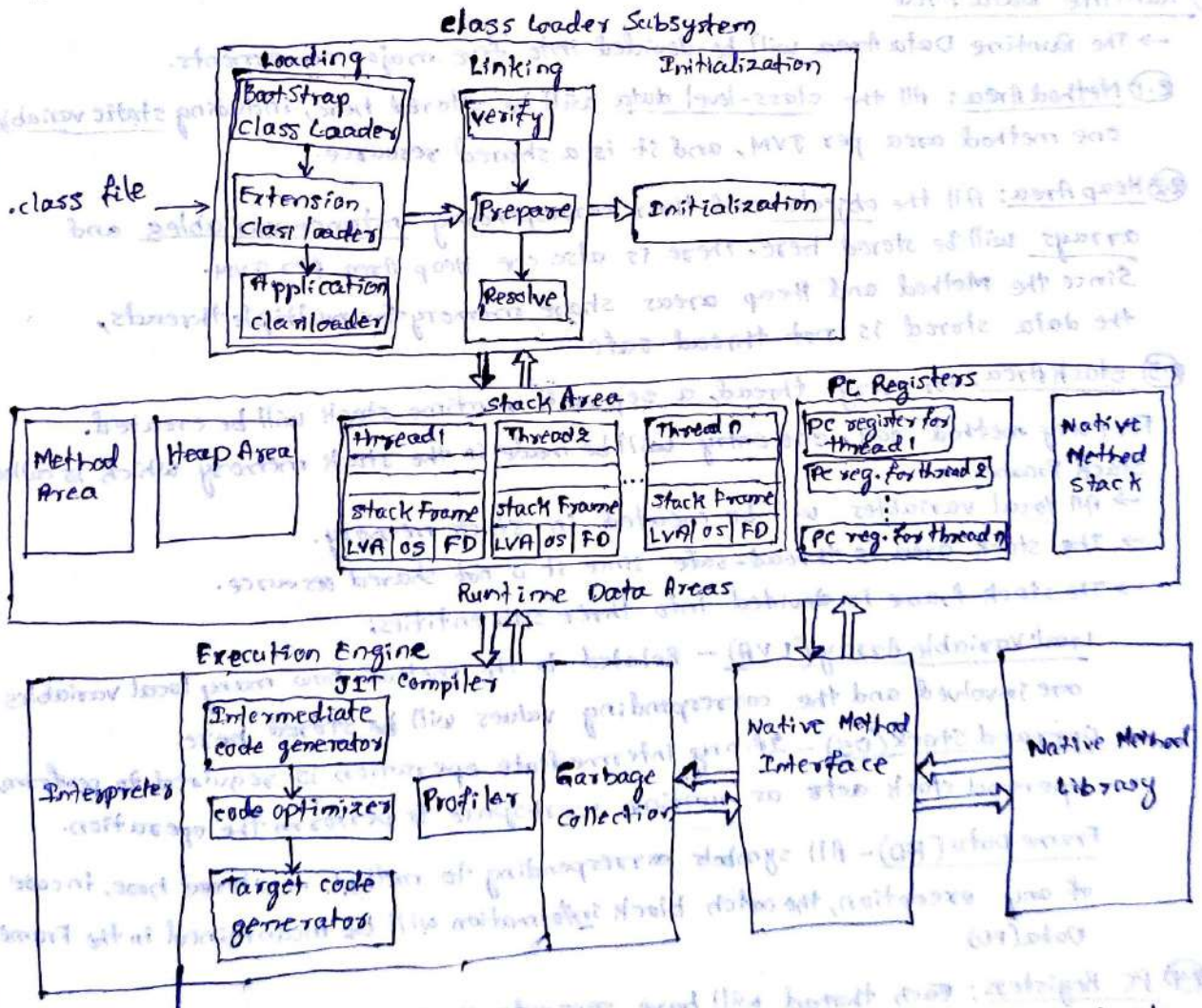
(28) JVM Architecture

What is JVM?

→ A virtual machine is a software implementation of physical machine. Java developped with concept of WORA (Write Once Run Anywhere), which runs on a VM.

→ The compiler compiles the java file into a java .class file, then that .class file is input into the JVM, which loads and executes the class file.

JVM Architecture:



① Class Loader Sub System:

→ Java's dynamic class loading functionality is handled by class loader subsystem. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time.

(i) Loading: → classes will be loaded by this component. Bootstrap class loader, Extention class loader and Application class loader will help in achieving it.

Bootstrap class loader – Loads the classes from bootstrap classpath, nothing but rt.jar. Highest priority will be given to this loader.

Extension class loader – loads the classes which are inside the ext folder (jre/lib)

Application class loader – Loads the application level classpath, path mentioned in Environment variable, etc.

(1.2) Linking:

verify – Byte code verifier will verify ~~the~~ whether the generated byte code is proper or not. if verification fails we will get the verification error.

Prepare – For all static variables memory will be allocated and assigned with default values.

Resolve – All symbolic memory references will be replaced with the original references from method area.

(1.3) Initialization:

→ This is the final phase of class loading, here all static variables will be assigned with the original values, and the static block's will be executed.

(2) Runtime Data Area

→ The Runtime Data Area will be devided into five major components.

(2.1) Method Area: All the class-level data will be stored here, including static variables one method area per JVM, and it is a shared resource.

(2.2) Heap Area: All the objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread safe.

(2.3) Stack Area: For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called Stack Frame.

→ All local variables will be created in stack memory.

→ The stack area is thread-safe since it is not shared resource.

→ The stack frame is devided into three sub entities;

Local Variable Array (LVA) – Related to the method how many local variables are involved and the corresponding values will be stored here.

Operand Stack (OS) – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.

Frame Data (FD) – All symbols corresponding to method is stored here, in case of any exception, the catch block information will be maintained in the Frame Data (FD).

(2.4) PC Registers: Each thread will have separate PC Registers, to hold the address of current executing instruction, once the instruction is executed the pc register will be updated with next instruction.

(2.5) Native Method Stack: Native Method Stack holds native method information. For every thread, a ~~separate~~ separate native method stack will be created.

③ Execution Engine

→ The byte code, which is assigned to the Runtime Data Area, will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece (line by line).

3.1) Interpreter: The Interpreter interprets the bytecode faster but executes slowly. The disadvantage of interpreter is that when one method is called multiple times everytime a new interpretation is required.

3.2) JIT Compiler: The JIT compiler neutralizes the disadvantage of the interpreter. The execution engine will be using the help of the interpreter in converting bytecode, but when it finds repeated code it uses the JIT compiler, which compiles entire byte code and changes it to native code. This native code will be used directly for repeated method calls, which improves the performance of the system.

Intermediate code Generator - Produces intermediate code.

Code Optimizer - Optimizes the intermediate code generated above

Target Code Generator - Generates Machine code or Native code

Profiler - It is a special component, responsible for finding hotspots, i.e whether the method is called multiple times or not.

3.3) Garbage Collector: Collects and removes unused/unreferenced objects. Garbage collection can be triggered by calling System.gc(), but the execution is not guaranted.

Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries, which is required for the execution engine.

List Iterator methods:

1. public boolean hasNext();
2. public Object next();
3. public int nextIndex();

} forward movement

4. public boolean hasPrevious();
5. public Object prious();
6. public int previousIndex();

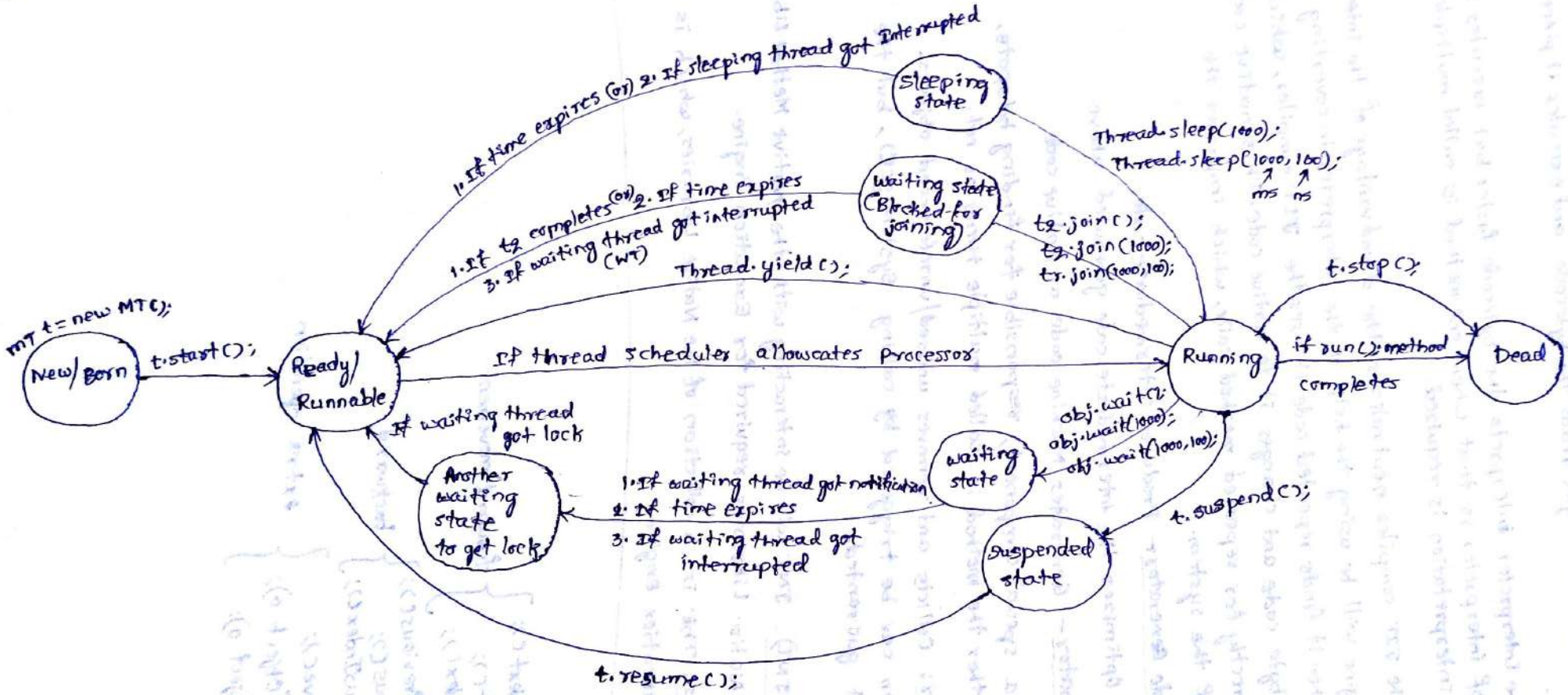
} backward

7. public void remove();
8. public void add (Object o);
9. public void set (Object o);

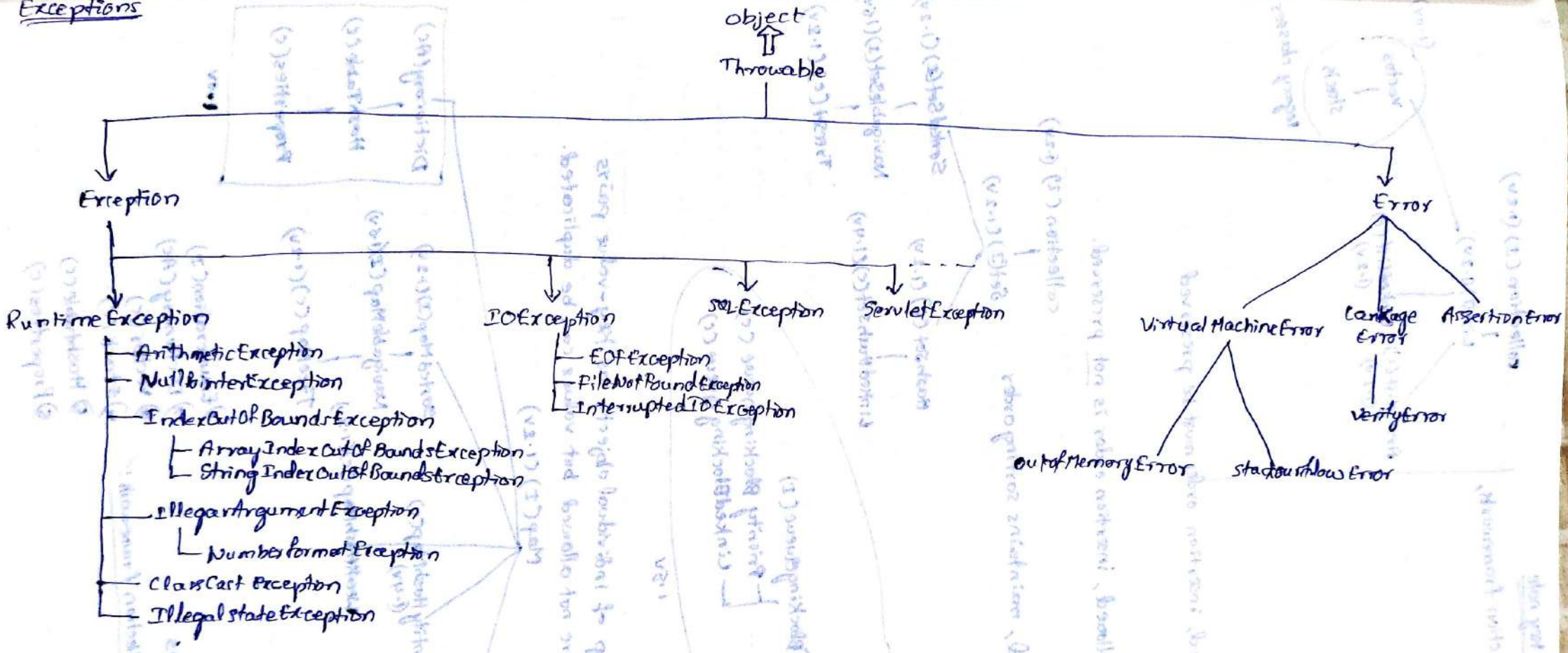
} extra operation

Thread Life Cycle

MT - my thread



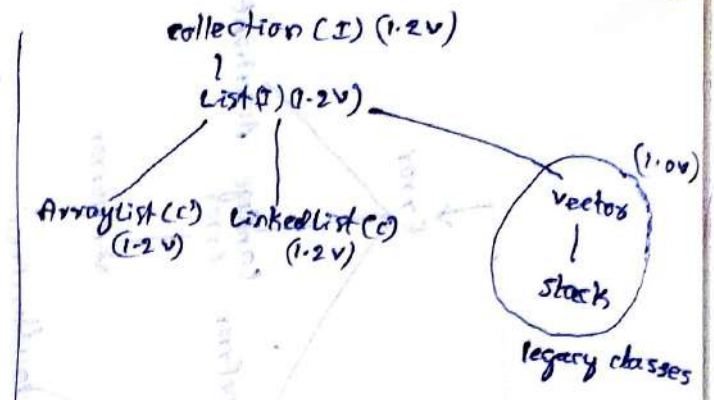
Exceptions


object
Throwable
Exception
Error
RuntimeException
ArithmeticException
NullPointerException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException
IllegalArgumentException
NumberFormatException
ClassCastException
IllegalStateException
IOException
EOFException
FileNotFoundException
InterruptedIOException
SQLException
ServletException
VirtualMachineError
LinkageError
AssertionError
VerifyError
OutOfMemoryError
StackOverflowError

Durgasoft - Collections key note

9 key interfaces of collection framework,

1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. Navigable Map
9. Sorted Map



List(I)

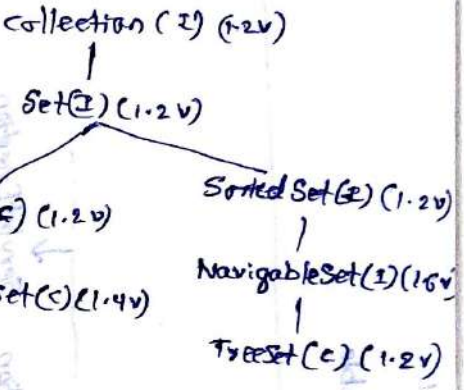
→ Duplicates ~~not~~ are allowed, insertion order must be preserved

Set(I)

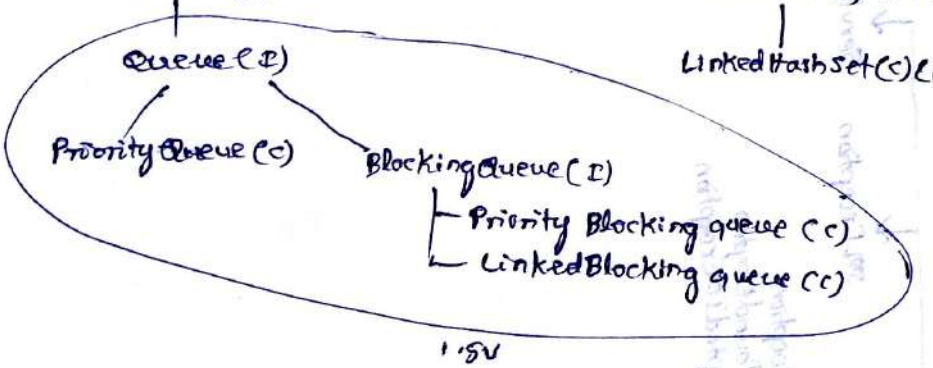
→ Duplicates are Not allowed, insertion order is not preserved.

SortedSet(I)

→ Duplicates not allowed, maintains sorting order



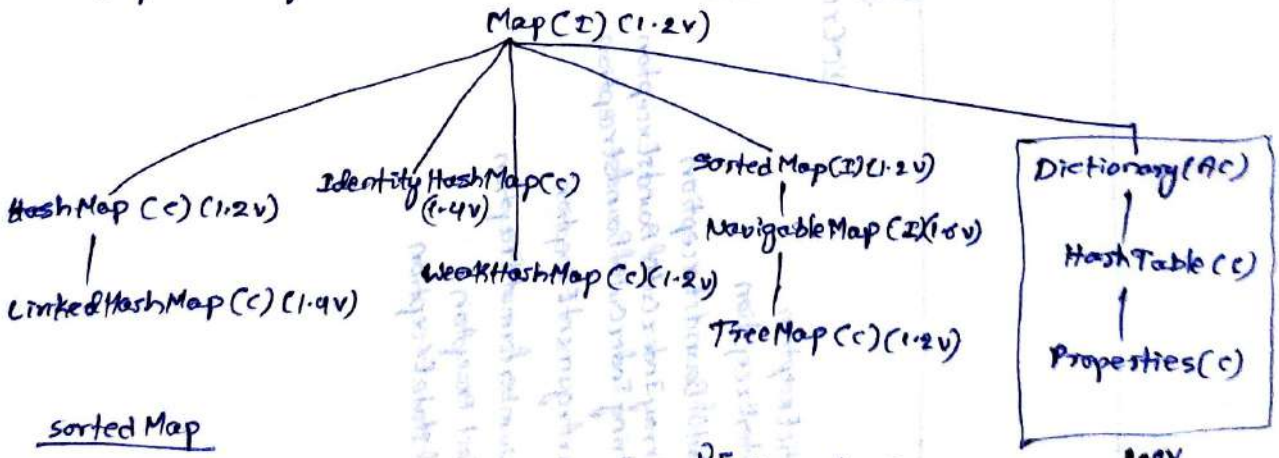
Queue (I)



Map (I)

→ Represents group of individual objects as key-value pairs

→ Duplicate keys are not allowed but values can be duplicated.



sorted Map

→ Key-value pair

→ Sorting order of keys

Legcy characters in collection framework:

1) Enumeration (I)
2) Dictionary (AC)
3) vector (c)
4) stack (c)
5) Hashtable (c)
6) Properties (c)

Collection (I)

→ List of methods present in collection interface which can be applicable for any collection object.

1) boolean add (Object o);
2) boolean addAll (Collection c);
3) boolean remove (Object o);
4) boolean removeAll (Object o);
5) boolean retainAll (Collection c); // To remove all objects except those present in c
6) void clear ();
7) boolean contains (Object o);
8) boolean containsAll (Collection c)
9) boolean isEmpty ();
10) Int size ();
11) Object [] toArray ();
12) Iterator iterator ();

→ There is no concrete class which implements collection interface directly.

List (I)

→ Duplicates are allowed and insertion order is preserved.
→ Maintains order by index
→ List interface defines the following specific methods,

1) boolean add (int index, Object o);
2) boolean addAll (int index, Collection c);
3) Object get (int index);
4) Object remove (int index);
5) Object set (int index, object new); // to replace
6) Int indexOf (Object o); // Returns index of first occurence of 'o'.
7) Int lastIndexOf (Object o);
8) ListIterator listIterator();

ArrayList (C)

→ The underlying data structure is resizable array (or) growable array.
→ Duplicate objects are allowed.
→ Insertion order preserved.
→ Heterogeneous objects are allowed.
→ Null insertion is possible.

Constructors:-

1) ArrayList a = new ArrayList();
   → Default initial capacity 10, new capacity = $\left(\text{current capacity} \times \frac{3}{2}\right) + 1$
2) ArrayList a = new ArrayList (int initialCapacity);
3) ArrayList a = new ArrayList (Collection c); // creates equivalent collection for the given collection

Getting synchronized version of ArrayList obj:-

```
Public static List synchronizedList (List l);
    ArrayList a = new ArrayList();
    List l1 = collections.synchronizedList(a);
```

similarly for set & Map,

```
public static Set synchronizedSet (Set s);
public static Map synchronizedMap(Map m);
```

→ ArrayList & Vector classes implements RandomAccess interface, so that any random element can access with same speed.
→ RandomAccess interface present in util package and doesn't contain any methods. It's a marker interface.

LinkedList(C)

→ Underlying data structure is double LinkedList
→ Insertion & deletion in the middle is possible
→ For retrieval operation LinkedList is worst choice
→ Duplicate objects are allowed.
→ Insertion ~~obje~~ order is preserved.
→ Heterogeneous objects are allowed.
→ Null insertion is posible.
→ Implements Serializable and Coloneable interfaces but not RandomAccess.
→ Linked List methods,

1) void addFirst(Object o);
2) void addLast(Object o);
3) Object getFirst();
4) Object getLast();
5) Object removeFirst();
6) Object removeLast();

constructors:

① LinkedList l = new LinkedList();
② LinkedList l = new LinkedList(Collection c);

Vector

→ Underlying data structure is resizeable array (or) growable array.
→ Duplicate objects are allowed.
→ Insertion order is preserved.
→ Heterogeneous objects are allowed.
→ Null insertion is possible.
→ Implements Serializable, Cloneable and RandomAccess Interfaces.
→ Every method present in vector is synchronized and hence Vector is Thread safe.
→ vector methods,

To add Objects:

1) add(Object o); — collection
2) add(int index, object o); — List
3) addElement(Object o); — vector

To remove elements:

1) remove(Object o); — Collection
2) remove(int index); — List
3) removeElement(Object o); — Vector
4) removeElementAt(int index); — Vector
5) removeAllElements(); — vector
6) clear(); — Collection

To get Objects:

1) Object get(int index); — List
2) Object elementAt(int index); — Vector
3) Object firstElement(); — vector
4) Object lastElement(); — vector

other methods:

1) Int size(); // How many objects added
2) Int capacity(); // Total capacity
3) Enumeration elements();

constructors:

1) vector v = new Vector();
→ Initial capacity 10
→ new capacity = currentcapacity × 2
2) Vector v = new Vector(int initialcapacity);

3) vector v = new Vector (int initialcapacity, int incrementalcapacity);

4) vector v = new vector (collection c);

Stack

→ Its a child class of vector.

→ whenever LIFO order required then we should go for Stack.

Stack s = new Stack();

→ Stack methods,

1) Object push(Object o); // To insert object into stack
2) Object pop(); // to remove and return top of stack
3) Object peek(); // To return top of the stack without removal
4) boolean emty(); // Return true if stack is empty.
5) Int search(Object o); // Returns offset if the element is available otherwise returns "-1"

The 3 cursors of java

1) Enumeration
2) Iterator
3) ListIterator

Enumeration:

→ To get objects one by one from legacy collection objects.

→ we can create Enumeration obj by using elements() method.

Enumeration e = v.elements();
└ vector obj

→ Enumeration interface defines the following two methods.

1) public boolean hasMoreElements();
2) public Object nextElement();

limitations of Enumeration:-

→ Applies for legacy classes and it is not a universal cursor

→ we can get only read access and can't do remove operation.

→ To overcome this we use Iterator concept in 1.2v.

Iterator:

→ To get objects one by one from any collection obj.

→ It's an Universal cursor.

→ Can perform both read and remove operations.

→ we can get Iterator object by using iterator() method.

Iterator itr = c.iterator();

→ Iterator methods,

1) public boolean hasNext();
2) public Object next();
3) public void remove();

Limitations of iterator:-

→ Both enumeration and iterator are single direction cursors only. we can't move to the backward direction.

→ only read and remove operations can be done, we can't perform ~~read~~ replacement and addition of new objects.

→ To overcome these limitations we go for ListIterator.

ListIterator

→ It is a child interface of iterator.

→ Can move either to the forward or backward direction hence it is a bidirectional cursor.

→ Can perform replacement and addition of new objects. in addition to read and remove operations.

→ Using list iterator method we can create listIterator object.

```
ListIterator itr = l.listIterator();
```

→ List iterator has following 9 methods,

1) public boolean hasNext();
2) public Object next();
3) public int nextIndex();

} forward

4) public boolean hasPrevious();
5) public Object previous();
6) public int previousIndex();

} backward

7) public void remove();
8) public void set(Object o);
9) public void add(Object new);

Set (I)

→ It is the child interface of collection.

→ Duplicates not allowed and insertion order is not preserved.

→ Set doesn't contains any new methods, we use collection interface methods only.

HashSet

→ Underlying data structure is Hashtable.

→ Insertion order not preserved and it is based on hash code of objects.

→ Duplicates not allowed.

→ If we are trying to insert duplicate objects we won't get compile time error and runtime error, add() method simply returns false.

→ Heterogeneous objects are allowed.

→ Null insertion is possible.

→ Implements Serializable and Cloneable interfaces but not Random Access.

Constructors:

1) HashSet h = new HashSet();
   → Defaut initial capacity 16 and default fill ratio (or) load factor × 0.75

2) HashSet h = new HashSet(int initialCapacity);
   → Creates empty hashset with specified initial capacity and default fill ratio 0.75.

3) HashSet h = new HashSet(int initialCapacity, float fillratio);

4) HashSet h = new HashSet(Collection c);

LinkedHashSet:

→ It is child class of HashSet

→ LinkedHashSet is exactly same as Hashset except the following diff.

HashSet	LinkedHashSet
1) Underlying data structure is Hashtable.	1) Underlying datastructure is combination of LinkedList and Hashtable.
2) Insertion order is not preserved	2) Insertion order is preserved.
3) Introduced in 1.2V	3) Introduced in 1.4V

Note:- LinkedHashSet and LinkedHashMap commonly used for implementing "cache" applications where order must be preserved and duplicates are not allowed.

SortedSet (I)

→ child interface of Set.
→ If we want to represent a group of "unique objects" according to some sorting order then we should go for sortedset interface.
→ That sorting can be either default natural sorting (or) customized sorting order.
→ Sorted set methods,

1) Object first();
2) Object last();
3) SortedSet headSet(Object o); // Returns sorted set whose elements are <=o
4) SortedSet tailSet(Object o); // " " " >=o
5) SortedSet subSet(Object o1, Object o2); // " " >=o but < o2
6) Comparator comparator(); // Returns comparator obj that describes underlying sorting technique.
If we are following defaut natural sorting order then this method returns null.

100
101
104
106
109
110
120

first() → 100
last() → 120
headSet(109) → [100, 101, 104, 106]
tailSet(109) → [109, 110, 120]
subSet(104, 110) → [104, 106, 109]
comparator() → null

TreeSet

→ Underlying data structure is balanced tree.
→ Duplicate objects are not allowed
→ Insertion order is not preserved and it is based on some sorting order of objs
→ Heterogeneous objects are not allowed if we are trying to insert heterogeneous obj we will get classcastException.
→ Null insertion is possible (only once).

Constructors:

1) TreeSet t = new TreeSet();
→ Creates empty Treeset where all elements inserted according to default natural sorting order.
2) Treeset t = new Treeset(Comparator c);
→ Creates empty Treeset where all objects will be inserted according to customized sorting order specified by comparator object.
3) Treeset t = new Treeset(Sortedset s);
4) Treeset t = new TreeSet(Collection c);

Null acceptance:
→ For the empty Treeset as the 1st element "null" insertion is possible but after inserting that null if we are trying to insert any other we'll get NullPointerException.

Comparable Interface(I):

→ It is present in java.lang package and it contains only one method, compareTo() method.

```
public int compareTo(Object obj);
    obj1.compareTo(obj2);
```

→ returns -ve if and only if obj1 has to come before obj2
→ returns +ve if and only if obj1 has to come after obj2
→ returns (zero) 0 if and only if obj1 and obj2 are equal.

→ If we are depending on default natural sorting order then internally JVM will use compareTo() method to arrange objects in sorting order.

→ comparable meant for default natural sorting order.
→ comparator meant for customized sorting order

Comparator Interface(I):

→ It is present java.util package this interface defines following 2 methods.

1) public int compare(Object obj1, Object obj2);
2) public boolean equals(Object obj);

property	HashSet	LinkedHashSet	Treeset
1) Underlying data structure	Hashtable	LinkedList + Hashtable	Balanced Tree
2) Insertion order	Not preserved	Preserved	Not preserved (By default)
3) Duplicate objects	Not allowed	Not allowed	Not allowed
4) Sorting order	Not applicable	Not applicable	Applicable
5) Heterogeneous objects	Allowed	Allowed	Not allowed
6) Null insertion	Allowed	Allowed	For the empty Treeset as the 1st element null insertion is possible in all other cases we will get NPE

Map(I)

→ If we want to represent group of objects as "key-value" pair the we should go for Map interface.
→ Both the key and value are objects only.
→ Duplicate keys are not allowed but values can be duplicated.
→ Each key-value pair is called "one entry".

→ Map is not a child interface of collection and hence we can't apply collection interface methods here.

→ Map specific methods,

1) Object put(Object key, Object value);
   → To add entry to the map, if key is already available then the old value replaced with new value and old value will be returned.
2) void putAll(Map m);
3) Object get(Object key);
4) Object remove(Object key);
   → Removes entry associated with specified key and returns corresponding value.

5) boolean containsKey(Object key);
6) boolean containsValue(Object value);
7) boolean isEmpty();
8) int size();
9) void clear();
10) Set keySet(); // The set of keys we are getting.
11) Collection values(); // The set of values we are getting
12) Set entrySet(); // The set of entrySet we are getting.

HashMap

→ Underlying data structure is Hashtable
→ Duplicate keys are not allowed but values can be duplicated.
→ Insertion order is not preserved and it is based on hash code of the keys.
→ Heterogeneous objects are allowed for both key and value.
→ Null is allowed for keys (only once) and for values (any number).

Diff. between HashMap and Hashtable:-

HashMap	Hashtable
1) No method is synchronized	1) Every method is synchronized.
2) Multiple threads can operate simultaneously on HashMap object and hence it is not thread safe.	2) Multiple threds can't operate simultaneously on Hashtable object and hence Hashtable obj is thredd safe.
3) Relatively performance is high	3) Relatively performance is low
4) Null is not allowed for key and value.	4) Null is not allowed for key and value otherwise we will get NPE.
5) It is non legacy and introduced in 1.2v	5) It is legacy and introduced in 1.0v

How to get synchronized version of HashMap:-
public static Map synchronizedMap(Map m);

Constructors:
1) HashMap m = new HashMap();
→ Intial capcity 16 and default fill ratio 0.75
2) HashMap m = new HashMap(int initialcapacity);
3) HashMap m = new HashMap(int initialcapacity, float fillratio);
4) HashMap m = new HashMap(Map m);

LinkedHashMap:-

HashMap	LinkedHashMap
1) Underlying data structure is Hashtable	1) Underlying data structure is combination of Hashtable + LinkedList
2) Insertion order is not preserved	2) Insertion order is preserved.
3) Introduced in 1.2v	3) Introduced in 1.4v

Note:- In general we can use LinkedHashSet and LinkedHashMap for implementing cache operations.

Identity HashMap:
→ It is exactly same as HashMap except the following diff,
→ In case of HashMap JVM will always use ".equals()" method to identify duplicate keys.
→ But in identityHashMap JVM will use "==" (double equals operator) to identify duplicate keys.

~~SortedMap~~

HashTable

→ Underlying data structure is Hashtable
→ Insertion order is not preserved and it is based on Hashcode of Keys.
→ Heterogeneous objects are allowed for both keys and values.
→ Null key (or) null values not allowed otherwise we get NPE.
→ Duplicate keys are allowed but values can be duplicated.

Constructors:-

1) Hashtable h = new Hashtable();
   → Default initial capacity 11 and fill ratio 0.75
2) Hashtable h = new Hashtable (int initialCapacity);
3) Hashtable h = new Hashtable(int initial capacity, float fillratio);
4) Hashtable h = new Hashtable (Map m);

Properties

→ child class of hashtable
→ By using properties object we can read and hold properties from property files into java application.

Constructor:-

Properties p = new Properties();

→ If properties both key and value should be String type only

Methods:-

1) String getProperty(String propertyname); // Returns value associated with specified property.
2) String setProperty (String propertyname, String propertyvalue);
3) Enumeration propertyNames();
4) void load (InputStream is); // Any input stream we can pass
5) void store (OutputStream os, String comment); // Any o/p stream we can pass

Collections class

→ Collections class defines several utility methods for collection objects.

1) public static void sort (List l);
2) public static void sort(List l, Comparator c); // list should n't be empty otherwise we get NPE

ex:- Collections.sort(l);
← ArrayList obj

Searching the elements of List:-

1) public static int binarySearch (List l, object obj);
2) public static int binarySearch (List l, Object obj, Comparator c);

Reverse the elements of List:-

public static void reverse(List l);

Collections.reverse (l);

APP- Synopsis

List

→ List extends Collection interface
→ Provides ordered collection of objects.
→ Classes in implements List interface are, ArrayList, vector and LinkedList. all these classes provides ordered collection of objects and allows duplicate values.

1) ArrayList:-
→ Growable array with fast iteration and fast random access.
→ Ordered by index and is not sorted in nature.

2) Vector:-
→ It is almost same as the ArrayList except that, vector methods are synchronized and hence thread safe.

3) LinkedList:-
→ Elements are ordered by index and are double-linked to each other.
→ Because of this it provides adding and removing elements from both sides and hence it may iterate slow than ArrayList but insertion and deletion is much faster than ArrayList.

Set

→ Set extends collection interface.
→ It doesn't allow duplicate values.
→ classes implements are, HashSet and TreeSet

1) HashSet:-
→ Unsorted and unordered set, it uses hash code of objects being inserted
→ Doesn't allow duplicates.

2) TreeSet:-
→ Its a sorted set which ensures elements will be in ascending order naturally.
→ Uses red-black tree structure to store elements.

3) LinkedHashSet:-
→ Its a ordered version of HashSet.

Map

→ Map doesn't extends collection interface
→ Provides key-value pairs of objects.
→ Classes implements are, HashMap, HashTable and TreeMap.

1) HashMap:-
→ Provides unsorted and unordered key-value pairs.
→ Allows one null key and multiple null values.

2) HashTable:-
→ Its a synchronized version of HashMap (all methods are thread safe)
→ Doesn't allows null keys or null values.

3) TreeMap:-
→ Like TreeSet, TreeMap is a ordered Map which persists elements in ascending order.

Diff between List and Set

→ Set is unordered collection where List is ordered collection (based on zero based index).

→ List allows duplicate elements but set doesn't allow duplicates

→ List doesn't prevent inserting null elements (as many you like), but set will allow only one null element.

Diff between Arrays and ArrayList

→ Arrays are created of fixed size where as ArrayList is dynamic in nature and can vary its length. Also Array size can't be incremented but Arraylist size is variable.

→ Once Array is created elements cannot be added or deleted from it. But with ArrayList elements can be added and deleted at runtime. Arraylist is one dimensional but array can be multidimentional.

→ Arrays stores objects of same data type (Homogeneous). ArrayList if not used with generic can store objects of different data types.

Diff between List and ArrayList

List	ArrayList
→ It is an interface	→ It is a class, implementation of List interface
→ Length is fixed, you can't resize it.	→ Length is dynamic in nature, you can resize it.
→ Store both primitives and objects	→ Can't stores primitives, stores only objects