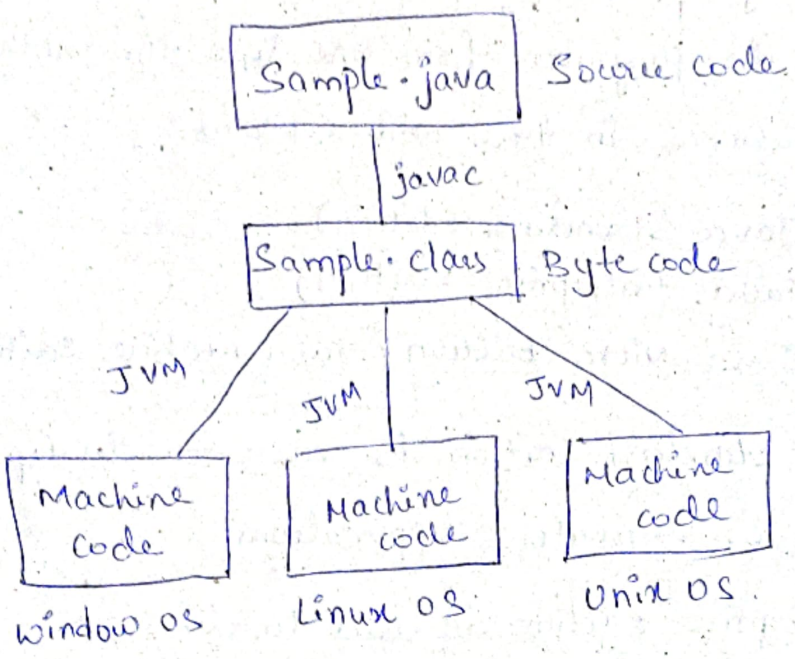
JAVA

19/12/22

Features of Java

* Java is a object oriented programing language.
* In oops, object is the main concept.
* Object means a real world Entity. Whatever we see in the realworld we can represent as an object.
* Java is called a realworld programing language.

→ Java is a platform independent/portable programing language. The Java applications will run on any operating System.



→ Java is a Secured programing language. The java has Eliminated the drawbacks from C and C++. The pointers Concept has been Eliminated in java. The pointers are used to work with address location of the variables.

→ Java is mainly used for Web applications, Enterprise application development. where Security is most important.

→ Java is a robust language. robust means Strong. Java has a Strong memory management mechanism.

→ JVM will be allocating the memory for the objects. and object will be Saved permanently.

→ The Garbage Collector of Java is used to deallocate the memory. Java has a Good Exception handling mechanism.

- Exception means an abnormal Situation. Exception handling is a mechanism which is used to handles Exceptions.

→ Java is a open Source technology The Java Software is freely available.

→ Java is a Simple programing language. The Java has Eleminated the drawbacks from C and C++. Java doesn't Support pointers and go to statement.

→ goto is a jump control Statement which moves the control of the program from one line to another line.

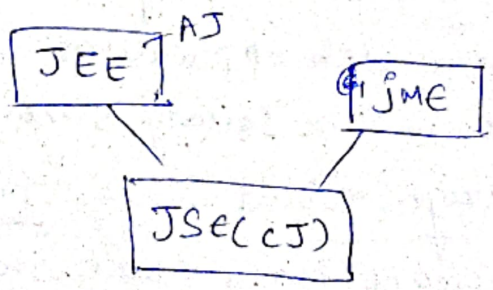
→ The Java Comes in three ~~edi~~ Editions:

1) JSE (Java Standard Edition)
2) JEE (Java Enterprise Edition)
3) JME (Java Micro Edition / Java Mobile Edition)

* The Java Standard Edition is used to develop desktop applications or Stand alone applications.

* Java Enterprise edition is used to developm webapplications and Enterprise applications.

* Java Micro Edition / Java Mobile Edition is used to develop mobile apps.



* JDK (JAVA Development Kit)

→ In the JDK of Java have all the Java tools.

Javac (Java compiler)

Java

Jar (Java archi-file)

Javap

Javadoc (Java documentation) and etc...

* JRE (Java Runtime Environment)

JRE will have all the Java predefined classes, interfaces etc.

↳ package

↳ Subpackages

↳ classes

interfaces

Methods

* JVM (Java Virtual Machine)

The JVM of Java will have three functions.

1. It is used to load a .class file in the Memory.
2. The memory will be allocated in different memory areas.
3. JVM converts byte code to the machine code.

20/12/22

→ In Java, Every Executable statement should be inside a class. Even main method also should be inside a class.

To create a class we use a keyword called class.

```
class classname
{

}
```

→ Outside of class we can write package declarsation, import statements and comments about the program.

→ Inside the main method of the program we create an object.

→ Objects are saved peminantly in the heap memory

→ The syntax of creating an object.

```
Classname objectname = new classname();
```

→ Object will have three properties:

1) State :- It represent the position of the object.

2) Behaviour :- It represent the purpose of using the object.

3) Identity :- The identification of the object.

→ A class represents a group of objects of similar type.

→ If animal is a class different types of animals are objects of animal class.

→ Object represents a real world Entity any physical Existing Entity can be represented as an object.

→ An object can also be represented as an instance of class.

→ The main method of java is

```
public Static void main (String[] args) {

}
```

> Main is a public method, the main method can be accesible to all the classes.

> Main is a static method, the static methods to call object is not required.

> Main method doesn't return any value, so, the return type of the main method is void.

> The main method takes the arguments as a string type.

> String is a class in Java and it represents group of characters.

> passing arguments to the main method at run time is called command line arguments.

→ To display the output to the user we use

System.out.println();

> System is a class which is used to connect with system resources.

> System class is available in <u>java.lang package</u>

> Java.lang is the default package of Java.

> The lang package is available to all the java classes.

→ System.out.print(); is used to display the output in same line.

→ System.out.printf(); will work as a printf(); method in C language and it uses format specifiers

<u>Eg:-</u> int a=5, b=6;

System.out.println("a="+a+"\n"+"b="+b);

<u>Output</u>: a=5
b=6.

<u>Eg2:-</u>

int a=5, b=6;

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

Output: a+b=11

Eg:-
```
class Demo
{
public static void main (String args[])
{
System.out.println ("Hello");
}
}
```

→ For Compilation we can use javac Demo.java

→ for Execution we use java Demo

21/12/22

Variable

→ Variable represents a location in the memory

→ A variable can hold ~~ea~~ only one value at a time

→ The value of the variable is not fixed

→ Variable name should ~~to~~ start with small letter eg: "a"

→ A variable name should not start with a digit but it can contain a digit.

→ Key words are not used for variable names.

Syntax for define variable

Datatype variablename;

Keywords :-

→ Keywords represents a reserved word, Every Keyword has a specific meaning, Keywords are not used for variables.

→ There are 52 Keywords in java.

Switch, case, default, try, catch, finally, Static, do, for, while ....

Identifiers:-

→ Identifiers represents symbolic names, the name given for class, variable, Method, interface or Identifiers.

Datatypes :-

There are two types of data types.

1) preemptive datatype
2) Nonpreemptive (or) reference datatype.

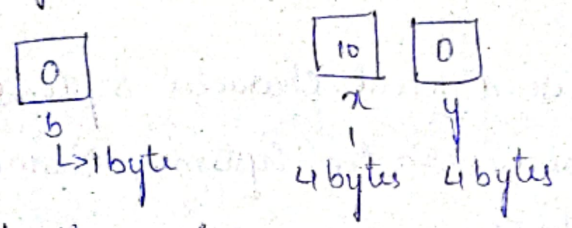
1) primitive datatypes.

* Integer type

→ byte - 1 byte
→ short - 2 bytes
→ int - 4 bytes
→ long - 8 bytes

Integer type is mainly used to store numbers.

byte b; int x=10, y;



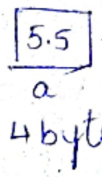
* floating point type

It is used to store decimal values. There are two data types. 1) float 2) double.

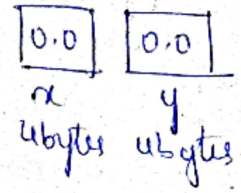
the default value is 0.0.

1) float - 4 bytes 2) Double - 8 bytes

float a=5.5f;



float x, y;



* Char type - 2 bytes

→ It can store only one character. character should be enclosed in single quotes (' ') The default value for the character is \U000

→ Unicode character system as ASCII, ISO, UTF are substypes

Eg:- char ch = 'x';

2 bytes

→ Java follows unicode character system.

* Why character occupies 2 bytes in Java?

→ The Java is used for web applications and enterprise applications development.

→ In this applications we use different character systems. To represent the different character systems, character required to store 2 bytes.

* Boolean:-

It has two possible states. They are

1) True
2) False

The default value is false

It takes only one bit (1 bit) in the memory

Eg:- boolean b;

boolean b_1 = true;

The default value for b is false. b will occupy 1 bit.

b_1 will take 1 bit and its value is true.

2) Non primitive (or) Reference data types

* String :- String is a class in Java and it represents group of character. The String class is available in java.lang package. Java.lang is the default package in Java

Eg:- String S_1 = "abc";
String S_2;

The S_2 value is NULL

* Array :- Array is an object in Java and it is used to represent group of elements of Same type.

22/12/22

Operators

→ Operators are used to perform a particular operation.

(a+b) expression.
a and b are operands and '+' is a operator

* Arithmatic Operators / Mathematical Operators

+ - Addition
- - Subtraction.
* - Multiplication
/ - Division operator will give the Quotient value
% - Modulus Operator will give the remainder

Eg:- 10/2 = 5 11/2 = 5
10%2 = 0 11%2 = 1

```
class Arithmaticoperators
{
public Static void main (String args[])
{
int a = 5, b = 5;
System.out.println ("a-b= "+(a-b));      output : a-b=0
System.out.println ("a/b = "+(a/b));      output : a/b=1
System.out.println ("a%b = "+(a%b));      output : a%b=0
}
```

* Relational Operators

→ The relational operators are used for comparision.

>	- Greater than	10>9 - True
<	- Less than	10>=9 - True
>=	- greater than or Equal	10<10 - False
<=	- less than or Equal	10<=10 - True
==	- is Equals to.	2==2 - True
!=	- not Equals to	2!=2 - False

Eg:-

```
public class RelationalOpDemo
{
  public Static void main (String args[])
  {
    System.out.println ("5<5::"+(5<5));   False
    System.out.print ln ("5>6::"+(5>6));   False
    System.out.println ("9<=10::"+(9<=10));   True
    System.out.print ln ("10==10::"+(10==10));   True
    System.out.print ln ("9!=8::"+(9!=8));   True
  }
}
```

* Logical Operators :-

→ The logical operators are used to combine two or more Expressions

Logical AND (&&)

Logical OR (||)

Logical Not (!)

→ The logical AND(&&) operator will return true when two or more Expressions are true.

Exp1	Exp2	R
T	T	T
T	F	F
F	T	F
F	F	F

$(2<3)$ && $(5>6)$

T && F → F

$(10==10)$ && $(9>8)$

T && T → T

→ Logical OR (||) :- The logical OR operator will return true when any one of the Expression is true.

Exp1	Exp2	R
T	T	T
T	F	T
F	T	T
F	F	F

$(1>2)$ || $(5<6)$

F || T — T

$(9>8)$ || $(10==10)$

T || T — T

→ Logical NOT (!)

Logical NOT operator will return opposite of the result

T F $!(5>6) \rightarrow T$

F T $!(10<11) \rightarrow F$

3/01/23

* Bitwise operators :-

Bitwise operators will work on bits (0's & 1's)

Bitwise AND (&)

Bitwise OR (|)

Bitwise XOR (^)

Binary Equivalent

2	8	0
2	4	0
2	2	0
	1	

8 - 1000

$2^3\ 2^2\ 2^1\ 2^0$

2	9	1
2	4	0
2	2	0
	1	

9 - 1001

$2^3\ 2^2\ 2^1\ 2^0$

1+8=9

→ The Bitwise AND operator will return true whenever both the bits are true

→ The Bitwise OR operator will return true whenever any one of the bit is true

→ The Bitwise XOR operator will return true whenever any one of the state is true and returns false whenever both the statements are same.

Eg:

Exp1	Exp2	R
0	0	0
0	1	1
1	0	1
1	1	0

AND (&)

9 - 1001
8 - 1000
1000

OR (|)

9 - 1001
8 - 1000
1001

XOR (^)

9 - 1001
8 - 1000
0001

* Increment / Decrement Operator

These operators are unary operators. The Unary operators works with single operand

++ → Increment operator.

-- → Decrement operator.

The Increment operator Increases the value by 1.

The Decrement operator Decreases the value by 1.

a++ post Increment ++a pre Increment

a-- post Decrement --a pre Decrement

Eg:- ~~post~~ Increment pre Increment post decrement pre decrement

int a=10, b;	int x=9, y;	int p=7, q;	int a=5, b;
b = a++;	y = ++x;	q = p--;	b = --a;
b=10	y=10	q=7	b=4
a=11	x=10	p=6	a=4.

→ In post Increment first the value will be assigned then it will Increment.

→ In preincrement first Incrementation will happen then It will assign.

→ In. post Decrement first the value will be assigned then it will Decrement.

→ In pre Decrement first Decrementation will happen then It will assign.

* Conditional / Ternary operator (?:)

The Conditional operator are the ternary operator works with more than two Expressions

Syntax:- (Exp1) ? (Exp2) : (Exp3)

→ First Exp1 will be checked If it is true Exp2 will ~~will~~ be Executed, If Exp1 is false It will Execute Exp3.

→ The Conditional operator is used as a short hand way of writing if Else Statement.

Eg:- (5>6) ? 5 : 6 ⟶ 6

(1<2) ? 1 : 2 ⟶ 1

(5<6) ? (5*2) : (10%2) ⟶ 10

* Assignment operators

+=

-=

*=

/=

%=

Eg:- int a=10;

a+=5; (a=a+5)

O/p: a value is 15.

int a=5;

a-=2; (a=a-2)

O/p:- a value is 3.

Control Statements / conditional Statements

* if Statement
* if else Statement
* if elseif else Statement
* Switch Statement.

*If Statement :-

→ The if Statement is used to check only one Condition.

Syntax:-

```
if (condition)
{
    Statements
}
```

→ If the Condition will be checked, if it is true then Statements inside the if will be executed, if it is false remaining code will be Executed in the program.

* Write a program to check the given number is equal to 100 or not.

```
public class IfDemo {
public Static void main(String[] args){
int a=100;
if(a==100)
{
System.Out.println("a value is 100");
}
System.out.println("Main Method");
}
}
```

O/p :- a value is 100
Main Method.

4/01/23

* If Else Statement :-

If else Statement is used to check two Conditions in an application

Syntax :-

```
if (condition)
{ statements
}
Else
{
Statement
}
```

First If Condition will be checked if it is true the Statement inside the if will be Executed. If the condition is false the else block will be executed.

* Write a program to check the given number is Even or odd.

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

        int num = 11;

        if (num % 2 == 0)
            System.out.println(num + " is a Even number");
        else
            System.out.println(num + " is a odd number");

    }
}
```

* Scanner class

→ Scanner is a pre defined class in java.util package

→ The Scanner class is used to read input from the user.

→ The Scanner class has various methods to read different types of data.

→ We have to use import statement to import the Scanner Class in the program

```
import java.util.Scanner;
import java.util.*;
```

→ The "*" represents all the classes from the util package

→ We can create Scanner class object in the following way.

```
Scanner S = new Scanner(System.in)
```

→ The Scanner class has various methods to read user input.

nextInt() → reads Integer Values

nextFloat() → reads a float value

nextDouble() → reads a double value

next() → reads a string without Spaces

nextLine() → reads a String with Spaces

next().charAt(0) → used to read a character

(P) WAP to accept age of a person and check the person is eligible for vote or not.

```
import java.util.Scanner;
public class AgeDemo
{
  public Static void main(String args[])
  {
    Scanner S = new Scanner(System.in)
    System.out.println("Enter the age");
    int age = S.nextInt();
    if (age >=18)
        System.out.println("~~not~~ Eligible for Vote");
    Else
        System.out.println("Not eligible for Vote");
  }
}
```

Output

Enter the age

15

Not eligible fo vote

(P) WAP to accept a character from the user and check the character is in upper case (or) lower case

```
import java.util.Scanner;
public class charcheck
{
  public Static void main (String args[])
  {
    Scanner s = new Scanner (System.in);
    System.out.println ("Enter a character");
    Char ch = s.next().charAt(0);

    if(ch>='A' && ch<='Z')
    {
      System.out.println("The given character is uppercase");
    }
    Else
    System.out.println("Given character is lowercase");
  }
}
```

(P) WAP to accept a no. from the user and check the no. is divisible by 3 and 5 both.

(P) WAP to accept height (ft) 5.7 and weight (in) 155 of a person and check the person is eligible for physical test or not

(P) WAP to accept a no. from the user and check the no. is between 0 to 9 or not.

Root of a quadratic equation.

$$\boxed{x = (-b \pm \sqrt{(b^2 - 4ac)/(2a)})}$$

* If else if else Statement /ladder 5/01/23

If else if else Statement is used to check multiple Conditions in an application.

Syntax:-

```
if (Condition)
{
  Statements
}
Else if (Condition)
{
  Statements
}
Else if (Condition)
{
  Statements
}
  :
Else
{
  Statement
}
```

① WAP to accept a number from a user and check the no. is positive, negative or zero.

→
```
import java.util.Scanner;
public class IfElseIfElseDemo {
    public static void main (String args[]) {
        Scanner S = New Scanner (System.in);
        int num = S.nextInt();
        if (num > 0)
            S.o.p (num + " is a positive number");
        Else if (num < 0)
            S.o.p (num + " is a negative number");
        Else
            S.o.p (num + " is a zero");
    }
}
```

(P) WAP to accept two numbers from the user and an operator (+, -, *, /). and perform the operation based on the operator.

```
class Main {
    Public Static Void main(String args[]){
        Scanner S = new Scanner (System.in);
        System.out.println("Enter two numbers:");
        int ~~num1~~a = S.nextInt();
        int ~~num2~~b = S.nextInt();
        System.out.println("Enter operator (+,-,*,/):");
        Char op = S.next().charAt(0);
        if (op == '+')
            System.out.println("Sum=" +(a+b));
        Else if (op == '-')
            System.out.println("Sub=" + (a-b));
        Else if (op == '*')
            System.out.println("mul=" + (a*b));
        Else if (op == '/')
            System.out.println("div=" +(a/b));
        Else
            System.out.println("Invalid operator");
    }
}
```

Output

```
Enter two numbers:
10
20
Enter operator (+,-,*,/):
+
Sum = 30
```

Switch Statement

Switch Statement is used to check multiple conditions in an application.

Syntax:-

```
Switch (Expression)
{
Case Constant1 : Statements
                 break;

case constant2 : Statements
                 break;

   :
   :

default:
        Statement

}
```

→ We can write Integer, character, String based Expressions in Switch Statement

→ Whenever the Switch Expression matches with any of the cases the Statements inside the case will be Executed.

→ If the Expression does not match with any of the case then default Statement will be Executed.

→ The break Statement in the Switch will move the control out of the Switch Statement.

(P) WAP to accept two numbers from the user and operator (+, -, *, /) and based on the operator perform the operation using Switch Statement.

```
class Main {
    public Static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        System.out.println("Enter two numbers:");
        int num1 = S.nextInt();
        int num2 = S.nextInt();
        System.out.println("Enter operator(+,-,*,/)");
        char op = S.next().charAt(0);

        Switch (op)
        {
         case '+':
            System.out.println ("Sum =" + (num1+num2));
            break;
         case '-':
            System.out.println ("sub=" + (num1-num2));
            break;
         case '*':
            System.out.println ("mul=" + (num1 * num2));
            break;
         case '/':
            System.out.println ("div=" + (num1 / num2));
            break;
         default:
            System.out.println ("Invalid operator");
        }
    }
}
```

(P) WAP to accept a number from the user and check the no. is Even or odd. using Switch Statement.

```
class Main {
    public Static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        System.out.println(" Enter the number:");
        int num = S.nextInt ();

        Switch (num % 2)
        {
          case '0' :
                System.out.println (num + "is Even");
                Break;
          default :
                System.out.println (num + " is odd");
        }
    }
}
```

(P) WAP to accept a character from the user and check the Character is a vowel or a Consonant.

```
class Main {
    public static void main (String args[]){
        Scanner S = new Scanner (System.in);
        System.out.println (" Enter a character: ");
        Char ch = S.next().charAt(0);

        Switch (ch )
        {
          case 'A':
          case 'E':
          case 'I':
          case 'O':
          case 'U':
          case 'a':
```

```
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            System.out.println("Given character is vowel");
            Break;
        default:
            System.out.println("Given character is Consonant");
    }
  }
}
```

(P) WAP to print the day of the week.

(P) WAP to accept a color from the user and check the Color is RGB colour or not.

(P) WAP to accept a character from the user and check the character is in upper case, lower case, ~~dio~~ its a digit or Special character. (if elseif else)

Loop Statements

6/01/23

→ The loop Statements are used to Executed set of instructions repeatedly per a particular no. of times.

→ There are three types of loops.

1) While loop
2) Do While loop
3) For loop.

1) While loop

The While loop is called pre-test loop, in while loop first the condition will be checked, if it is true the Statements inside the loop will be Executed.

If the condition is false the Statements inside the loop will not be Executed.

Syntax :-

```
while (Condition)
{
  Statements
}
```

(P) WAP to print the numbers from 1 to 10 using while loop.

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

(P) To print 10 to 1 numbers.

```
int i=10;
while (i>=1)
{
System.out.println(i);
i--;
}
```

(P) print Square of the numbers from 1 to 10.

```
int i=1;
while (i<=10)
{
System.out.println(i*i);
i++;
}
```

(P) WAP to print Sum of Even numbers and Sum of odd numbers from 1 to 10.

```
int i=1;
int esum=0, osum=0;
while (i<=10)
{
if (i%2==0)
esum = esum+i;
Else
osum = osum+i;
i++;
}
S.o.p ("Even Sum::"+esum);
S.o.p ("Odd Sum::"+osum);
}
```

i=1 osum=1
i=2 esum=2
i=3 osum=4
i=4 esum=6
i=5 osum=9
i=6 esum=12
i=7 osum=16
i=8 esum=20
i=9 osum=25
i=10 esum=30

(P) WAP to accept a number from the user and check the given number is palindrome or not.

* To check a number is palindrome, the reverse of the no. is equal to same number.

* When we modulo (%) a number by 10, we get each digit out of the number.

```
class palindrome {
    public static void main (String args[]) {
        Scanner s = new Scanner (System.in);
        int num = s.nextInt();
        int temp = num;
        int rev = 0;

        while (num != 0)
        {
            int d = num % 10;
            rev = rev * 10 + d;
            num = num / 10;
        }
        if (temp == rev)
            System.out.println ("Given no. is palindrome");
        Else
            System.out.println ("Given no is not palindrome");
    }
}
```

① 10)123(12
10
23
20
d = 3

rev = 0 * 10 + 3
= 3

num = 12

② 10)12(1
10
d = 2

rev = 3 * 10 + 2
= 32

num = 1

③ 1 % 10
d = 1

rev = 32 * 10 + 1
= 320 + 1
rev = 321

num = 521

temp = num

① 10) 521 (52
50
21
20
d = 1

rev = rev * 10 + d
= 0 * 10 + 1
= 1

num = num/10
= 52

② 10) 52 (5
50
d = 2

rev = 1 * 10 + 2
= 12

num = 5

③ ~~10~~) 5 % 10

d = 5

rev = 12 * 10 + 5
= 125

(P) WAP to accept a no. from the user and get Sum of digits of a number

num = 123

temp = num

Sum = 0;

① 10) 123 (12
10
23
20
d = 3

Sum = Sum + d
= 0 + 3
= 3

num = 12

② 10) 12 (1
10
d = 2

Sum = Sum + d
= 3 + 2
= 5

num = 1

③ 1 % 10

d = 1

Sum = Sum + d
= 5 + 1
= 6.

```
class Sum {
    public Static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        int num = S.nextInt();
    int Sum = 0;

        while (num != 0)
        {
            int d = num % 10;
            Sum = Sum + d;
            num = num / 10;
        }
        System.out.println ("Sum");
    }
}
```

9/01/23

(P) WAP to Gain display a count of digits of a number.

```
class Sum {
    public static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        int num = S.nextInt();
    int Count = 0;
        while (num != 0)
        {
            int d = num % 10;
            Count = Count + 1;
            num = num / 10;
        }
        System.out.println (Count);
    }
}
```

num = 123

Count = 0

10) 123 (12
10
23
20
d = (3)

Count = 0+1
= 1

num = num/10
= 12

10) 12 (1
10
d = (2)

Count = 1+1
= 2

num = num/10
= 1

1 % 10

d = 1

Count = 2+1
= 3

(P) WAP to accept 5 values from the user and print Sum of Even values and product of Odd values.

```
Class ProductSum {
    public static void main(String args[]){
        Scanner S = new Scanner(System.in);
        int i=1;
        int Sum = 0;
        int product = 1;
        while (i<=5)
        {
        System.Out.println("Enter a number");
        int num = S.nextInt();
        if (num%2==0)
        {
          Sum = Sum+num;
        }
```

```
Else
  { product = product * num;
  }
  i++;
 }
System.out.println ("Sum of even numb- values :: " +sum);
System.out.println ("product of odd values :: " + product);
}
}
```

Tracing

i = 1
Sum = 0
product = 1

i = 1
num = 4 → Even
Sum = Sum + num
= 0 + 4
= 4

i = 2
num = 3 → odd
product = 1 * 3
= 3

i = 3
num = 7 → odd
product = 3 * 7
= 21

i = 4
num = 8 → Even
Sum = 4 + 8
Sum = 12

i = 5
num = 5 → odd
product = 21 * 5
product = 105

(P) WAP to read n values from the user and find count of positive, negative and zero's

```
class CountDemo {
    public static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        System.out.println ("Enter n value");
        int n = S.nextInt();

        int i=1, pcount=0, ncount=0, zcount=0;

        while (i<=n)
        {
         System.out.println ("Enter a number ::");
         int num = S.nextInt();

         if (num>0)
         {
           pcount++;
         }
         Else if (num<0)
         {
           ncount++;
         }
         Else
         {
           zcount++;
         }
         i++;
        }
        System.out.println ("count of pcount :: " + pcount);
        System.out.println ("count of negative values :: " + ncount);
        System.out.println ("count of zero values :: " + zcount);
    }
}
```

(P) WAP to accept a number from the user and check the number is Armstrong number or not

10/01/23

```
class Armstrong {
    public static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        int num = S.nextInt();
        int temp = num;
        int count = 0;

        while (num != 0)
        {
            d = num % 10;
            count ++;
            num = num / 10;
        }

        while (num != 0)
        {
            d = num % 10;
        (double) Sum = Sum + Math.pow (d, count);
            num = num / 10;
        }
        if (temp == Sum)
            System.out.println ("Armstrong number");
        else
            S.o.p ("Not a Armstrong number");
```

```
class Armstrong {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        int num = s.nextInt();
        int temp = num;
        int count = 0, sum = 0 double sum = 0;
        while (temp > 0)
        {
            int d = temp % 10;
            count++;
            temp = temp/10;
        }
        temp = num;
        while (temp > 0)
        {
            int d = temp % 10;
            sum = sum + Math.pow(d, count);
            temp = temp/10;
        }
        if (num == sum)
        System.out.println("Armstrong");
        else
        System.o.p("Not Armstrong");
    }
```

$45 = 4^2 + 5^2 = 41$

$145 = 1^3 + 4^3 + 5^3 = \ldots$

If same

* First we have to get the count of digits, based on the count of digits we use Math.pow.
* Sum of cubes of individual digits of a given number is equal to the same number is called Armstrong number

* DO While loop :-

→ The DO While loop is called post test loop. The Do while loop will be executed atleast once, Even if the condition is false

Syntax :-

```
do
{
  Statements
} while(condition);
```

→ The do while loop is used in menuapplications

Eg:-

```
class DOWhileDemo {
    public static void main (String args[]){
        int i=100;
        do
        {
          System.out.println(i);
        } while(i<99);
    }
}
```

(P) WAP to display a menu to the user *

1. Addition
2. Sub
3. Mul
4. Div.

take the user choice from the menu and based on the user choice perform the operation.

→ Menu application using DoWhile

```
class MenuApp {
    public static void main(String args[]) {
        Scanner S = new Scanner(System.in);
        char ch1;

        do
        {
        S.op("1.Add" + "\n" + "2.sub" + "\n" + "3:mul" + "\n" + "4.div");
        S.o.p("Enter your choice");
        int ch = S.nextInt();
        S.o.p(" Enter two numbers");
        int a = S.nextInt();
        int b = S.nextInt();

        Switch(ch)
        {
        Case 1;
          System.out.println("sum=" + (a+b));
          break;
        Case 2;
          System.out.println(" Sub=" + (a-b));
          break;
        Case 3;
          System.out.println("mul=" + (a*b));
          break;
        Case 4;
          System.out.println("div=" + (a/b));
          break;
        default:
          System.out.println(" Invalid op values");
        }

        S.o.p("Do u want to continue press y for yes and
                                n for No");
```

```
~~char~~ ch1 = S.next().charAt(0);
if(ch1 = = 'n')
    break;
} while (ch1 = = 'y');
}
}
```

output:-

1. Add
2. Sub
3. Mul
4. div

Enter your choice

1

Enter two numbers

2

3

Sum = 5

Do u want to continue press y for yes and n for NO

y

if N

break

1. Add
2. Sub
3. Mul
4. div

Enter your choice

Repeat

Casting :-

→ Casting refers to converting the data from one type to another type. There are two types of casting.

They are : 1) Widening 2) Narrowing

* Conversion from lower type to higher type is called widening.

→ There is no data loss in widening.

byte - short - int - long - float - double.

Eg :- byte b = 9 ; int y = 100 ;
int x = b ; float f1 = y ;

* Narrowing :-

→ It will convert higher type to lower type.

→ Compiler cannot perform narrowing. We have to explicitly specify the conversion in brackets.

double - float - long - int - short - byte.

Eg :- float a = 5.5f ;
int x = (int) a ;

Happy Number

Number = 153 = $1^2 + 5^2 + 3^2$
= 35

$= 2^2 + 5^2 = 4 + 25$
= 29

$35 = 3^2 + 5^2$
= 34

& $2^2 + 4^2$
$= 9 + 16 \Rightarrow 25$

11/01/23

For Loop :-

→ In for loop we write initialization, condition and Increment/Decrement of the loop in a single statement.

Syntax :-

```
for (Initialization; Condition; Inc/dec)
{
  Statements
}
```

(P) Write a for loop to print the numbers from 1 to 10.

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

(P) print from 10 to 1

```
for( i=10 ; i>=10 ; i--)
    System.out.println (i);
```

(P) WAP to accept a number from the user and print the factors of that number.

```
Scanner S = new Scanner (System.in);
int num = S.nextInt ();
                    for(int i=1; i<=num ; i++)
~~ifor( i<=num==0)~~ { if (num% i==0)
                        S.o.p( i);
                    }
```

(P) WAP to check the given number is prime or not.

→ a prime number will have two factors 1 and same number.

```
class prime{
    public static void main (String args[]){
        Scanner S = new Scanner (System.in);
        int num = S.nextInt();
        int Count = 0;
        for (int i=1; i<=num; i++)
        {
            if (num % i ==0)
            Count ++;
        }
        if (count ==2)
         System.out.println(" prime");
        Else
         System.out.println("Not prime");
    }
}
```

Tracing

num = 5

i=1

5 % 1 == 0 (T)

Count = 0+1;

=1

i = 2

5 % 2 == 0 (F)

Count = 1

i = 3.

5 % 3 == 0 (F)

Count = 1

i = 4

5 % 4 == 0 (F)

Count = 1

i = 5

5 % 5 == 0 (T)

Count = 1+1

= 2

(P) WAP to print multiplication table of a given number

```
class TableMul {
    public static void main(String args[]){
        Scanner S = new Scanner(System.in);
        int num = S.nextInt();

        for(int i=1 ; i<=10 ; i++)
        {
            System.out.printf
                ("%d * %d = %d \n", num, i, (num*i));
        }
    }
}
```

(P) WAP to accept a number from the user and print the factorial of that number

```
Scanner S = new Scanner(System.in);
int num = S.nextInt();
int fact=1;
for(int i=num; i>=1 ; i--)
{ fact= fact* i;
}
System.out.println(fact);
```

Tracing

i=5	i=3	
5>=1	3>=1	i=5
fact = 1*5 = 5	fact = 20*3 = 60	1>=1
i=4	i=4	fact = 120*1=120
4>=1	2>=1	
fact = 5*4=20	fact = 60*2 = 120	

(P) WAP to check the given number is strong or not

```
public class strongNum{
  public static void main(String args[]){
    Scanner s = new Scanner (System.in);
    S.o.p("Enter a number");
    int num = S.nextInt();
    int temp = num;
    int sum = 0;
    while (temp != 0)
    {
      int d = temp % 10;
      int fact = 1;
      for (int i=1; i<=d; i++)
      {
        fact = fact * i;
      }
      sum = sum + fact;
      temp = temp / 10;
    }
    if (sum == num)
    {
      System.out.println("Strong");
    }
    else
    {
      System.out.println("Not a strong number");
    }
  }
}
```

Tracing num = 145

10)145(14
10
45
40
d = 5

i=1 1<=5

fact = 1*1 = 1

i=2 2<=5

fact = 1*2
= 2

i=3 3<=5

fact = 2*3 = 6

i=4 4<=5

fact = 6*4 = 24

i=5 5<=5

fact = 24*5

fact = 120

sum = sum + fact = 0+120

num = num/10
= 145/10
= 14

10)14(1
10
d = 4

i=1 1<=4

fact = 1*1 = 1

i=2 2<=4

fact = 1*2 = 2

i=3 3<=4

fact = 2*3 = 6

i=4 4<=4

fact = 6*4 = 24

fact = 24

sum = 120 + 24

sum = 144

num = num/10
= 14/10
= 1

d = 1

i=1 1<=1

fact = 1

sum = 144 + 1

sum = 145

sum = num

145 is a strong number

Strong num = 145

~~1²+4²+5²~~

~~= 1+16+25~~

1! + 4! + 5!

⇒ 1 + (4 × 3 × 2 × 1) + (5 × 4 × 3 × 2 × 1)

= 1 + 24 + 120

= 145

12/01/23

(P) WAP to print the fibonacci Series.

In fibonacci Series first two numbers we have to take in series, the next value to generate, we have to add the first two numbers

Eg:- 0 1 1 2 3 5 8 13 21 . .

```
public class FibSeries {
  public static void main (String[] args) {
    int n1=0, n2=1, n3;
    System.out.println(n1+"\n"+n2);
    for(int i=2; i<=10; i++)
    {
      n3=n1+n2;
      System.out.println(n3);
      n1=n2;
      n2=n3;
    }
  }
}
```

Break Statement

Break Statement is used to Exit from the loop. When we use break Statement inside a loop the loop will be terminated and remaining lines in the program will be Executed normally

(P) WAP to read 10 values from the user when user Enters input as 999 Stop the process and print the Sum of its previous numbers.

```
class BreakDemo {
    public static void main(String args[]) {
        Scanner S = new Scanner(System.in);
        int Sum = 0;
        for (i=1; i<=10; i++)
        {
            System.out.println("Enter a number:");
            int num = S.nextInt();
            if (num==99)
                Break;
            else
                Sum = Sum + num;
        }
        System.out.println("Sum = " + Sum);
    }
}
```

Tracing

i=1 num = 2
i<=10

Sum = 0+2 = 2

i=2 num = 5
2<=10

Sum = 2+5 = 7

i=3 num = 12
3<=10

Sum = 7+12
= 19

i=4 num = 50
4<=10

Sum = 19+50 = 69

i=5
5<=10 num = 8

Sum = 69+8 = 77

i=6
6<=10 num = 99

break;

$\boxed{\text{Sum} = 77}$

Continue Statement

The Continue Statement is used to continue the loop and it is used to skip the given condition and continue with the loop process.

(P) WAP to accept 10 values from the user when user Enters a negative number skip that number and continue the process for the remaining numbers and print sum of +ve values.

```
Public class Continue {
    public Static void main(String args[]) {
        Scanner S = new Scanner (System.in);
        int Sum = 0;
        for (i=1; i<=10; i++)
        {
        System.out.println("Enter a number:");
        int num = S.nextInt();
          if(num<0)
            ~~System~~ Continue;
          Else
            Sum= Sum+num;
        }
        System.out.println ("Sum =" +Sum);
    }
}
```

Happy Number :-

```
class Happy {
    public Static void main (String args[]) {
        Scanner S = new Scanner (System.in);
        System.out.println (" Enter a number ");
        int num = S.nextInt();
        int temp = num;
        int Sum = 0;
        while (temp > 9)
        {
            while( temp > 0)
            {
                int d = temp % 10;
                Sum = Sum + (d * d);
                temp = temp/10;
            }
            temp = Sum;
            Sum = 0;
        }
        if ( temp == 1)
            System.Out.println (" Happy ");
        Else
            System.out.println (" unhappy ");
    }
}
```

Roots of Quadratic Expression/Equation

→ Quadratic Equation has 2 roots and the roots depend on the discriminant. ~~In the a~~ to find the roots we use

$$\frac{-b \pm \sqrt{(b^2 - 4ac)}}{2a}$$

→ In the above formula, $(\sqrt{(b^2-4ac)})$ is called discriminant. The value of d may be positive, negative or zero.

→ If d is positive $(d>0)$, the root will be:
both roots are real and different.

→ ~~If~~ To find the roots ~~we~~ $\frac{-b \pm d}{2a}$

→ If $d==0$, roots are real and Equal. $\frac{-b}{2a}$.

→ If this two conditions are false, the roots are Imaginary

```
public class Main {
    public Static void main(String args[]) {
        Scanner S = new Scanner(System.in);
        System.out.println("Enter values for a, b & c");
        int a = ~~inputs~~.nextInt();
        int b = S.nextInt();
        int c = S.nextInt();
        int d = b*b - 4*a*c;

        if (d>0)
        {
        System.out.println("Roots are real and distinct");
        double r1 = (-b + Math.Sqrt(d)) / (2*a);
        double r2 = (-b - Math.Sqrt(d)) / (2*a);
```

```
        System.out.println("Root1 = "+ r1);
        System.out.println("Root2 = "+ r2);
    }
    Else if (d==0)
    {
        System.out.println("Roots are real and Equal");
        double r1 = -b/(2*a);
        System.out.println("Root1 = Root2 = "+r1);
    }
    Else
        System.out.println("Imaginary roots");
    }
}
```

17/11/23

```
* class pattern1
  {
    public static void main(String args[])
    {
      for(int i=1; i<=5; i++)
      {
        for(int j=1; j<=i; j++)
        {
          System.out.print(i+" ");
        }
        System.out.println();
      }
    }
  }
```

Tracing

i=1

j=1

i=2

j=1,2

i=3

j=1,2,3

i=4

j=1,2,3,4

i=5

j=1,2,3,4,5

output

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

```
* class pattern2
  {
    public static void main (String args[])
    {
      int k=1;
      for (int i=1; i<=3; i++)
      {
        for (int j=1; j<=i; j++)
        {
          System.out.print(k+" ");
          k++;
        }
        System.out.println();
      }
    }
  }
```

tracing

K=1

i=1 j=1

K=2

i=2 j=1

K=3

i=2 j=2

K=4

i=3 j=1

K=5

i=3 j=2

K=6

i=3 j=3

output

```
1
2 3
4 5 6
```

ASCII Range for upper case and lower case are

A = 65 → Z = 90

a = 97 → z = 122

* Pattern 3.

```
{
 public Static void main (String args[])
  {
   for (int i=1; i<=3; i++)
    {
     Char ch= 'A';
     for (int j=1; j<=i ; j++)
      {
       System.out.print(ch+" ");
       ch++;
      }
      System.out.print(n (* );
    }
  }
}
```

Output

A.

A B

A B C

* int rows=9;

```
System.out.println("The pattern is");
for(int i=1; i<=rows; i++)
{
 for(int i=1; i<=rows; j--)
  {
   System.out.print(" ");
  }
 for (int j=1; i<=i ; j++)
  {
   System.out.print("* ");
  }
 System.out.println();
 i++;
}
```

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

18/6/23

(P) WAP to check prime within a range.

```
public class primeRange {
    Public Static void main (String args[])
    {
        int S1, S2, S3, flag=0, i, j;
        Scanner S = new Scanner (System.in);
        System.out.println ("Enter the lower limit:");
        S1 = S.nextInt();
        System.out.printh ("Enter the upper limit:");
        S2 = S.nextInt();
        System.out.println ("The prime numbers in between the
                                    Entered limits are:")
        for (i = S1, i <= S2; i++)
        {
            for (j = 2; j < i; j++)
            {
                if (i % j == 0)
                {
                    flag = 0;
                    break;
                }
                Else
                {
                    flag = 1;
                }
            }
            if (flag == 1)
            {
                System.out.println (i);
            }
        }
    }
}
```

* pattern 4 {

```
public Static void main (String args[]) {
    for (int i=1 ; i<=5; i++)
    {
        for (int j=1 ; j<=i ; j++)
        {
            System.out.print ( j+ " ");
        }
        System.out.println();
    }
}
}
```

Tracing

i=1 j=1

1<=1

j=2

i=2 j=1 i=2, j=2

1<=2 2<=2

i=3, j=1 i=3, j=2 i=3, j=3

1<=3 2<=3 3<=3

i=4, j=1 i=4, j=2 i=4, j=3 i=4, j=4

1<=4 2<=4 3<=4 4<=4

i=5, j=1 i=5, j=2 i=5, j=3 i=5, j=4 i=5, j=5

1<=5 2<=5 3<=5 4<=5 5<=5

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

```
* pattern5 {
    public Static void main(String args[]) {
        char ch='A';
        for (int i=1; i<=5; i++)
        {
            for (int j=1; j<=i; j++)
            {
                System.out.print(ch+" ");
                ch++;
            }
            System.out.println();
        }
    }
}
```

A
B C
D E F
g h i j
K L M N O

Tracing

ch=A i=1, j=1 1<=1 ch++

ch=B i=2, j=1 1<=2 ch++

ch=C i=2, j=2 2<=2 ch++

ch=d i=3, j=1 1<=3 ch++

ch=E i=3, j=2 2<=3 ch++

ch=f i=3, j=3 3<=3 ch++

ch=g i=4, j=1 1<=4 ch++

ch=h i=4, j=2 2<=4 ch++

ch=i i=4, j=3 3<=4 ch++

ch=j i=4, j=4 4<=4 ch++

↓

ch=O i=5, i=5 5<=5

* Left Triangle pattern

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

        int i, j, row = 5;

        for (i=0; i<row; i++)
        {
            for (j=2*(row-i); j>=0; j--)
            {
                System.out.print(" ");
            }
            for (j=0; j<=i; j++)
            {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

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

Tracing :

① $i=0$, $j=10$ $10>=0$ $j--$

$i=0$, $j=9$ $9>=0$ $j--$

$i=0$, $j=8, 7, 6, 5, 4, 3, 2, 1$ $1>=0$

$0>=0$

$j=0$ $0<=0$ $j++$ (print)

$j=1$ $1<=0$ (F)

i=1 , جز

(2) i=1 , 1<5 , j=8 8>=0

j--

i=1 , j=7

j=6,5,4,3,2,1,0>=0

j=0 , 0<=1

j=1 , 1<=1

(3) i=2 , 2<5 , j=6 . 6>=0

i=2 , j=5,4,3,2,1,0>=0

j=0 , 0<=2

j=1 , 1<=2

j=2 2<=2

(4) i=3 , 3<5 , j=4 , 4>=0

i=3 , j=3,2,1,0>=0

j=0 , 0<=3

j=1 , 1<=3

j=2 , 2<=3

j=3 , 3<=3

(5)

i=4 , 4<5 , j=2 , 2>=0

i=4 , j=1,0>=0

j=0 , 0<=4 , j=1 1<=4 , j=2 2<=4 , 3<=4 , 4<=4

(6) i=5 , 0>=0

j=0	j=1	j=2	j=3	j=4	j=5
0<=5	1<=5	2<=5	3<=5	4<=5	5<=5

19/01/23

Arrays

→ Array is a object in java and it is used to store group of Elements of Same type.

→ The Array Elements will be Stored Sequentially in the memory

→ The first Element of the array will be Stored at index "0" position.

→ The last Element of the array will be Stored at Size-1 position.

→ length is the property of the array which is used to give the Size of the array

→ We can create an array in two ways. They are

1. primitive array
2. reference array

→ To create a primitive array we use data type followed with the array name.

Eg:- int a[] = {1,2,3,4};
char ch[] = {'a','b','c'};

Eg:- int b[] = {11,15,16,17};

11	15	16	17
0	1	2	3

b

→ To create a reference array we use new keyword.
The reference arrays are stored in heap memory.

Eg:- int x[] = new int[5];

→ For processing array elements we can use a loop.
To display the array elements we can use normal for loop (or) Enhanced for loop (or) for each loop.

(P) WAP to accept array elements and display.

```
public class ArrayDemo {
    public Static void main(String args[]) {
        int a[] = new int[5];

        Scanner s = new Scanner(System.in);

        System.out.println("Enter the array elements::");
        for (int i=0; i<a.length; i++)
            a[i] = s.nextInt();

        System.out.println("The array Elements are::");
        for (int i=0; i<a.length, i++)
            System.out.println(a[i]);
    }
}
```

Input:-

Enter the array elements::
1
2
3
4
5

output:-

The array elements are::
1
2
3
4
5

* Display array Elements with index

```
S.o.pln("Enter the array elements::");
for (int i=0; i<a.length; i++)
a[i] = S.nextInt();

System.out.println("The array elements are::");
for (int i=0; i<a.length; i++)
System.out.printf("a[%d]= %d \n", i, a[i]);
```

* WAP to display array elements in reverse order

```
System.out.println("The array elements with index are:");
for (int i=a.length-1; i>=0; i--)
System.out.printf("a[%d] = %d \n", i, a[i]);
```

~~Inp~~ output:

The array elements with index are:

a[4] = 5
a[3] = 4
a[2] = 3
a[1] = 2
a[0] = 1

Input:-
1
2
3
4
5

Input:

Enter the array Elements::

21
31
62
54
10

Output:

The array elements are::

a[0] = 21
a[1] = 31
a[2] = 62
a[3] = 54
a[4] = 10

(P) WAP to accept an array of 5 Subject marks from the user and display total marks.

```
for (int i=0; i<Sub.length ; i++)
    Sum = Sum + Sub[i];

System.out.println("Sum");
```

(P) WAP to accept an assay of 5 Subject marks, if marks ≤ 35 print the student is failed else print sum of marks.

```
int Sum=0;
System.out.println(" Enter the Subject Marks:");
for (int i=0; i< Sub.length; i++)
    Sub[i] = S.nextInt();

int flag=0;

for (int i=0; i< Sub.length ; i++)
{
    if (Sub [i] ≥35)
        Sum =Sum + Sub[i];

    Else
        flag=1;
}
if (flag==1)
    System.out.println(" The Student is failed");
Else
    System.out.println(" Total marks :: " + Sum);
}
}
```

20/01/23

(P) WAP to Search for an Element in an array

```
class SearchArray {
  public static void main(String args[])
  {
    int a[] = new int[10];
    Scanner s = new Scanner(System.in);
    System.out.println("Enter array Elements ::");
    for(int i=0; i<a.length; i++)
      {
        a[i] = s.nextInt();
      }
    System.out.println("Enter Search Elements::");
    int Search = nextInt();

    int flag = 0;
    int i;

    for(i=0; i< a.length; i++)
    {
      if( a[i] == Search)
        {
          flag = 1;
          break;
        }
    }
    if(flag==1)
    {
      System.out.println("search element is found at index"
                                                  +i);
    }
    else
    {
      Search("Search element is not found");
  }
}
```

(P) WAP to find maximum and minimum Element from the array

```
class MinMaxArray {
  public Static void main (String args[]) {
    Scanner S = new Scanner(System.in);
    ~~public Static void main (str~~
    S.o.p ("Enter n value::");
    int n = S.nextInt();

    int a[] = new int[n];

    System.out.println("Enter array Elements:");
    for(int i=0; i<a.length; i++)
    a[i] = S.nextInt();

    int max=a[0];
    int min = a[0];

    for(int i=0; i<a.length; i++)
    {
      if (a[i]>max)
          max = a[i];
      if(a[i]<min)
          min = a[i];
    }
    System.out.println("Max Element is::"+max);
    System.out.println("Min Element is::"+min);
  }
}
```

Tracing

n = 5

min = 5

max = 5

5	8	9	16	20
0	1	2	3	4

i = 0	i = 1	i = 2	i = 3	i = 4
5 > 5	8 > 5	9 > 8	16 > 9	20 > 16
5 < 5	max = 8	max = 9	max = 16	max = 20
min = 5	8 < 5	9 < 5	16 < 5	20 < 5
max = 5	min = 5	min = 5	min = 5	min = 5

* In java.util package array class is available. This class is used to work with one dimentional arrays. This class has static methods. They are:

Arrays.Sort() → Ascending to decending

Arrays.Equals()

Arrays.binarySearch()

Eg:-
```
import java.util.Arrays;
public class ArraysDemo {
    Public static void main(String[] args) {
        int a[] = {5, 7, 6, 19, 28, 1};
        Arrays.Sort(a);
        System.out.println(" After Sorting:: ");
        for (int i=0; i<a.length; i++)
        {
            System.out.print (a[i]);
        }
    }
}
```

Output :- After Sorting : 1 5 6 7 19 28

(P) WAP to accept two array Elements from the user and check array "a" 1st Element is Equal to array "b" last Element or not.

(P) WAP to print the average of middle elements of the array

Mid+1

a

5	4	3	2	1
0	1	2	3	4

b

1	2	3	4	5
0	1	2	3	4

a[0]

a

1	2	3	4	5	6
0	1	2	3	4	5

Sum

```
int a[] = new int[6];
Scanner S = new Scanner (System.in)
S.o.p ("Enter assay elements :");
for (int i=0; i<a.length; i++)
{
  a[i] = S.nextInt();
}
System.out.println("Avg of mid elements of an array");
int mid;
if (a.length>0)
{
  mid = (a.length-1)/2;
  int avg = (a[mid]+a[mid+1])/2;
  System.out.println (avg);
}
```

1+2+3+4+5 / 5 = 15/5 = 3

23/01/2023

Ⓟ WAP to Sort an array (Bubble Sort)

* Swapping of two elements using temporary variable

Eg:- int a=10, b=20;
int c;
c=a; // c=10
a=b; // a=20
b=c; // b=10

```
Public class BubbleSort {
  public Static void main (String args[]) {
    int a[] = new int[5];
    Scanner S = new Scanner (System.in);
    System.out.println("Enter the array elements::");
    for (int i=0; i<a.length; i++)
    {
      a[i] = S.nextInt();
    }
    for(int i=0; i<a.length; i++)
    {
      for (int j=i+1; j<a.length; j++)
      {
        if (a[i] > a[j]) //a[i]<a[j] is for descending order
        {
          int temp = a[i];
          a[i] = a[j];
          a[j] = temp;
        }
      }
    }
    System.out.println ("Array after Sorting");
    for (i=0; i<a.length; i++)
    System.out.println ("a[i]);
  }
}
```

Tracing

$a[] = \{5, 10, 32, 1, 8\}$

$i=0 \quad i<5 \quad \cancel{0>1}$
$j=1 \quad 1<5 \quad 5>10$

$i=0 \quad 0<5 \quad a[0]>a[2]$
$j=2 \quad 2<5 \quad 5>32$

$i=0 \quad 0<5 \quad a[0]>a[3]$ temp = 5 $\{1, 10, 32, 5, 8\}$
$j=3 \quad 3<5 \quad 5>1$ ✓ a[i] = 1
a[j] = 5

$i=0 \quad 0<5 \quad a[0]>a[4]$
$j=4 \quad 4<5 \quad 1>8$ ✗

[10, 32, 5, 8]

$i=1 \quad 1<5 \quad a[1]>a[2]$
$j=2 \quad 2<5 \quad 10>32$

$i=1 \quad 1<5 \quad a[1]>a[3]$ temp = 10 $\{1, 5, 32, 10, 8\}$
$j=3 \quad 3<5 \quad 10>5$ ✓ a[i] = 5
a[j] = 10

$i=2 \quad 2<5 \quad a[2]>a[3]$ temp = 32
$j=3 \quad 3<5 \quad 32>10$ ✓ a[i] = 10 $\{1, 5, 10, 32, 8\}$
a[j] = 32

$i=2 \quad 2<5 \quad a[2]>a[4]$ temp = 10
$j=4 \quad 4<5 \quad 10>8$ ✓ a[i] = 8 $\{1, 5, 8, 32, 10\}$
a[j] = 10

$i=3 \quad 3<5 \quad a[3]>a[4]$ temp = 32
$j=4 \quad 4<5 \quad 32>10$ a[i] = 10 $\{1, 5, 8, 10, 32\}$
a[j] = 32

* Enhanced for loop / for each loop

→ The Enhanced for loop is the Short hand way of writing a for loop. By default enhanced loop will start with '0', the Condition will be taken by length property and the loop will increment

→ Enhanced for loop is used for displaying array and collection Elements.

Eg:- ① int a[] = {1, 2, 3, 4};

```
for(int i:a)
System.out.println(i);
```

② char ch[] = {'a', 'b', 'c', 'd'};

```
for(char c:ch)
System.out.println(c);
```

③ String str[] = {"Sravs", "Sree", "Shanth"};

```
for(String s:str)
System.out.println(s);
```

* Two-Dimensional Arrays :

→ The 2D assays are used to store the data in 2D-format. (with respect to rows and columns).

Syntax :-

Datatype arrayname [][] = new Datatype [rsize][csize];

→ A two dimensional Array will be represented in matrix format.

Eg :- int a[][] = new int [3][3];

→ The no. of Elements in 2D array is row × column [3×3 = 9 Elements]

1	2	3
4	5	6
7	8	9

(P) WAP to accept the Element in 2D assay and display

```
int a[][] = new int[3][3];
Scanner S = new Scanner (System.in);
System.out.println(" Enter the assay Elements : ");
for (int i=0; i<a.length; i++)
{
    for(int j=0; j<a.length; j++)
    {
        a[i][j] = S.nextInt();
    }
}
System.out.println(" The assay Elements are : ");
for (int i=0; i<a.length; i++)
{
    for(int j=0; j<a.length; j++)
    {
        System.out.printf("a[%d][%d] = %d", i, j, a[i][j]);
    }
System.out.println();
}
```

Tracing

i=0 0<3	i=1	j=2	[0][0]	[0][1]	[0][2]
j=0 0<3	j=0	j=0	1	2	3
			[1][0]	[1][1]	[1][2]
i=0	i=1	j=2	4	5	6
j=1	j=1	j=1	[2][0]	[2][1]	[2][2]
i=0	j=1	j=2	7	8	9
j=2	j=2	j=2			

24/01/23

(P) WAP to accept the elements in a two dimensional array and display the transpose of that array

Eg:-

1	2	3		1	4	7
4	5	6	⟶	2	5	8
7	8	9		3	6	9
Original				Transpose		

```
Public class TwoDDemo {
  public Static void main(string args[]) {
    Scanner S = new Scanner (System.in);
    int a[][] = new int[3][3];
    System.out.println("Enter the array elements");
    for (int i=0; i<a.length; i++)
    {
      for (int j=0; j<a.length; j++)
      {
        a[i][j] = S.nextInt();
      }
    }
    for (int i=0; i<a.length; i++)
    {
```

```
for (int j=0; j<a.length; j++)
{
  System.out.print(a[j][i]);
}
System.out.println();
}
}
}
```

(P) WAP to print Sum of the diagonal Elements of the array

```
int a[][] = new int[3][3];
Scanner S = new Scanner(System.in);
int Sum=0;
System.out.println ("Enter the array elements");
for (int i=0; i<a.length; i++)
{ for (int j=0; j<a.length; j++)
  {
    a[i][j] = S.nextInt();
    if(i==j)
    {
      Sum = Sum+a[i][j];
    }
  }
}
System.out.println("Sum= "+Sum);
```

Eg:-

1	2	3
4	5	6
7	8	9

Sum of diagonal = 1+5+9 = 15

(P) WAP to print matrix addition.

(P) WAP to accept the elements in an array a and array b and display the common elements between the arrays.

* Binary Search

→ In Binary Search the array should be Sorted, if the search element is available in the middle of the array we can directly ~~to~~ display the Element.

→ If the search element is greater than the mid Element we Search Second half of the array, If the Search Element is less than mid Element, we Search 1st half of array.

```
import java.util.Random;
public class BinarySearchDemo
{
  public Static void main (String args[])
  {
    int [] arr = new int [10];
    Random rand = new Random();
    for (int i=0; i<arr.length; i++)
    {
      arr[i] = rand.nextInt(100);
    }
    Arrays.Sort(arr);
    System.out.println(" Array values");
    for (int i=0; i<arr.length; i++)
    {
      System.out.print(arr[i]+" ");
    }
    Scanner s = new Scanner(System.in);
    System.out.println("\n Enter Key value");
    int key = s.nextInt();
```

```
int lower=0, upper=arr.length-1 , mid;
boolean found = false;

while(upper >=lower)
{
    mid = (upper+lower)/2;
    if (key==arr[mid])
    {
      found = true;
    }
    System.out.println("Search Element is found");
    break;
    }
    Else if (key<arr[mid])
    {
        upper = mid-1;
        found = true;
    }
    Else
    {
        lower = mid+1;
        found = true;
    }
    }
    if (lower>upper)
    {
      System.out.println("Element not found");
    }
  }
}
```

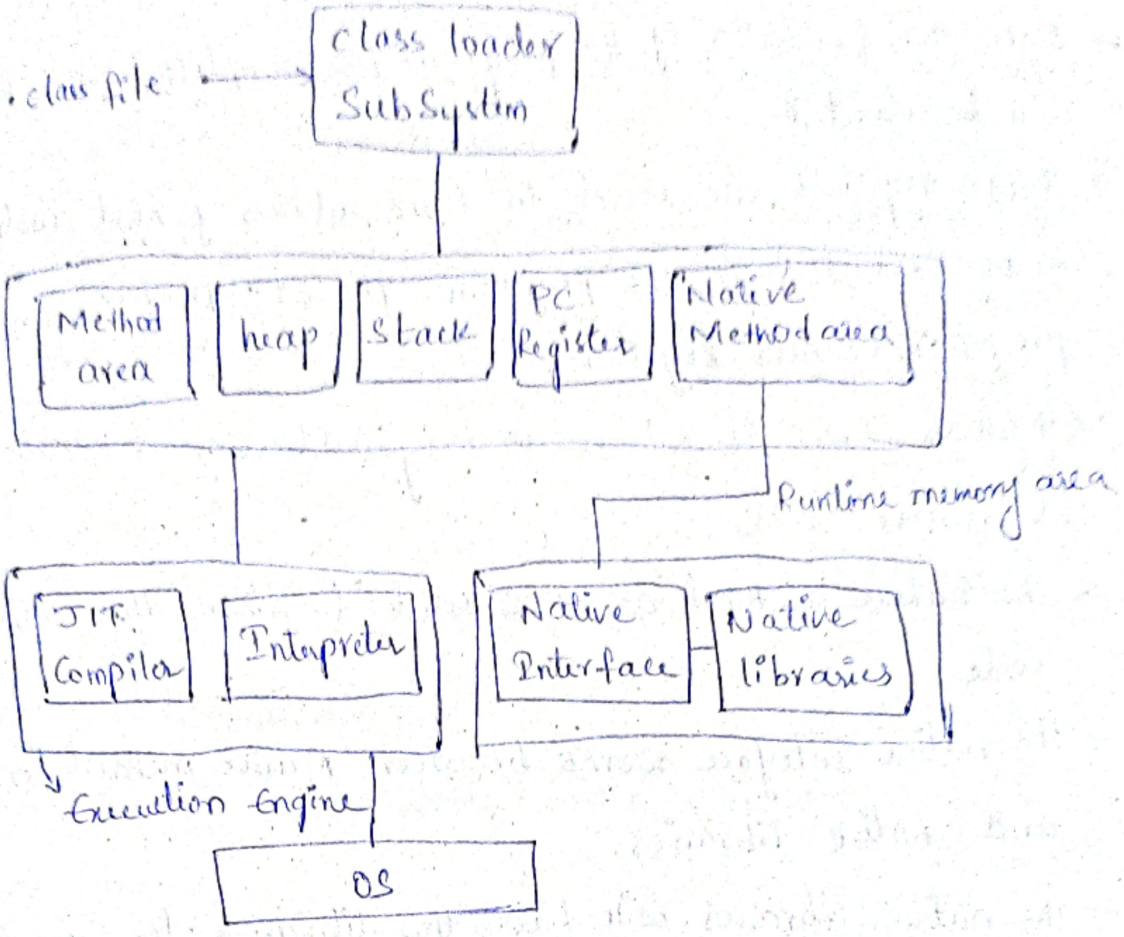
→ Random class is available in java.util package

→ The Random class is used to generate Random numbers

→ The nextInt() method of random class is used to generate random numbers from 0 to Specified range (Exclusive).

25/01/23

JVM Architecture



→ The class loader SubSystem is used to load a .class file in the memory and verifies the byte code.

→ If the byte code is proper then memory will be allocated in different memory blocks.

→ The method area will store static variables of the class at class loading time.

→ The heap memory stores the objects perminantly and every object will get a unique number called hashcode

→ The Stack is a temporary memory, whereas, heap is a permanant memory.

→ For every program execution the stack will be created. Every executable statement can be created as an entry in the stack.

→ Once the execution of the program is completed the stack will be deleted.

→ The PC registers are used to store address of next instructions to be executed in the program. PC stands for program counter register.

→ Other than Java if we write any code can be called as native code.

→ The native method area is used to store the native code.

→ The native interface works between native method area and native libraries.

→ The native libraries will have the libraries to execute native code.

→ The execution engine have two parts

* JIT compiler (Just in time compiler).
* Interpreter.

→ The instructions which are repeated can be executed by JIT compiler.

→ Interpreter executes the code line by line.

→ The JVM decides which code has to be executed by JIT compiler and interpreter.

→ The execution engine converts bytecode to machine code and it will be executed on the OS.

* String

→ String is a class in java in java.lang package.

→ A String represents group of characters.

→ String is an immutable class. Once the String is created we cannot change the value.

→ We can create a string in java in two ways.

1) String literal
2) String object

Eg:- String S_1 = "abc";

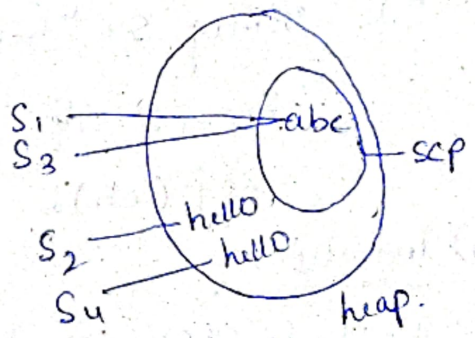
→ S_1 is called String literal. The literals are stored in SCP (String Constant pool).

→ SCP is designed to store literals and it is a part of heap memory.

Eg:- String S_2 = new String("hello");

→ S_2 is called String object. The objects are stored in the heap memory.

String S_1 = "abc";
String S_2 = new String("hello");
String S_3 = "abc";
String S_4 = new String("hello");



30/01/23

→ If Multiple String literals have Same value. A Single Copy of the value will be Stored in SCP (String Constant pool).

→ If multiple String objects have Same value different copies will be Stored in the heap memory.

→ The SCP is designed to adjust the memory requirements.

→ If Every object will be stored in the heap memory, memory does not Sufficient for project requirements. So, SCP is designed to adjust the memory requirements.

Methods of String Class :-

1) length()

length() is used to get the length of the String

Eg:- String S = "hello";

S.o.p (S.length()); // 5

2) charAt()

The charAt() will give you the character at the index position.

Eg:- String S = "hello";

char ch = S.charAt(0);

S.o.p (ch);

h	e	l	l	o
0	1	2	3	4

3) indexOf()

The indexOf() method will give the index position of a character

Eg:- String S1 = "hello";

S1.indexOf('h'); // 0

4) char[] toCharArray()

This method is used to convert a string to a character array.

Eg:- String str = "Welcome";
char ch[] = str.toCharArray();

5) Split()

The Split() is used to split the string into number of tokens. This method returns a string array.

Based on a delimiter, the Split() will split the string

Eg:- String str = "java is a programming language";
String s1[] = str.split(" ");
for (String s : s1)
System.out.println(s);

Output:- java
is
a
programming
language.

→ Java supports unicode character system. In unicode character system ASCII, UTF, ISO, ---- are subtypes.

→ The ASCII Range for A to Z is 65 to 90.

→ The ASCII Range for a to z is 97 to 122.

String Comparision

The Strings can be compared using i) == operator

ii) Equals()

iii) compareTo()

Eg:- String S_1 = "hi";
String S_2 = "hi";
System.out.println(S_1 == S_2) // true
System.out.println(S_1.Equals(S_2)); // true

String S_3 = new String("abc");
System.out.println(S_1 == S_3); // false
System.out.println(S_1.Equals(S_3)); // false

String S_4 = "HI";
System.out.println(S_1.Equals(S_4)); // false
System.out.println(S_1.EqualsIgnoreCase(S_4)); // true
System.out.println(S_4 == S_2); // false.

CompareTo()

→ CompareTo() is used to compare two strings and it can return 0 if the both the string are same along with the case
→ It returns a +ve value when one string character value is greater than the other string character value.
→ It returns -ve value when one character value is less than the other character value

Eg:- System.out.println("hi".compareTo("hi")); // 0
System.out.println("Hi".compareTo("hI")); // -32
System.out.println("welcome".compareTo("Welcome")); // 32
System.out.println("abc".compareTo("acb")); // -1
System.out.println("hi".compareToIgnoreCase("HI")); // 0

Substring()

String S = "hello";
System.out.println(S.Substring(0,3)); // hel

31/01/23

P) WAP to accept a String from a user and check the given String is palindrome or not (Without using reverse()).

```
Public class StringPalindrome {
  public Static void main(String args[]) {
    Scanner S = new Scanner(System.in);
    System.out.println("Enter the String:");
    String str = S.next();
    String rev = " ";
    for (int i = Str.length()-1 ; i>=0; i--)
    {
      rev = rev + Str.charAt(i);
    }
    if (Str.Equals(rev))
    {
      System.out.println("Given String is palindrome");
    }
    Else
    {
      System.out.println("Given String is not palindrome");
    }
  }
}.
```

output:-

```
Enter the String:
madam
Given String is palindrome
Enter the String:
hello
Given String is not palindrome
```

(P) WAP to check two Strings are Anagrams or not.

→ To check two Strings are Anagrams The length of both the Strings Should be Same.

→ After Sorting both the Strings the Strings Should be Equal.

Eg:- 1) Keep — 4
cekp (Sort)
2) peek — 4
cekp (Sort)

length

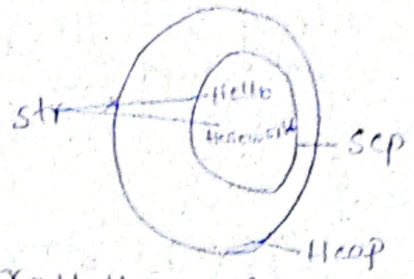
Equal

```
Scanner S = new Scanner(System.in);
System.out.println("Enter the first String");
String Str1 = S.next();
System.out.println("Enter the 2nd String");
String Str2 = S.next();
if (Str1.length() == Str2.length())
{
  Char ch1[] = Str1.toCharArray();
  Char ch2[] = Str2.toCharArray();

  Arrays.Sort(ch1);
  Arrays.Sort(ch2);
  boolean b = Arrays.Equals(ch1, ch2);

  if (b == true)
  {
    System.out.println("The Strings are anagrams");
  }
  Else
  {
    System.out.println("The Strings are not anagrams");
  }
}
Else
{ System.out.println("The Strings are not anagrams");
}
```

→ String is an Immutable class in Java

→ Immutable means we cannot change the value.

Eg:- String str = "hello";
str = str + " world";



→ In the above statement String str = Hellow will be created in the SCP, When we write str = str + world a new String called Hellowworld will be created in SCP, The hello is not replaced by hellowworld.

→ If hellow is not refered by any object then it is called unreferenced object (or) unused object.

→ When garbage collector Executes for that program then hello will be removed from the memory.

(P) WAP to accept a String from the user and print the no. of characters, words and Spaces.

```
String Str = "hello world";
int charCount = 0;
int Scount = 0;
for (int i=0; i< Str.length(); i++)
{
    Char ch = str.charAt(i);
    if (ch != ' ')
        charcount ++;
    else
        Scount ++;
}
System.out.println(charcount);
System.out.println (Scount);
```

Word count

```
public class WordCount {
  public static void main (String args[]) {
    String Str = "java is a programming language";
    int Count=1;
    for (int i=0; i<Str.length(); i++)
    {
      if (Str.charAt(i) == ' ')
      {
        Count++;
      }
    }
    System.out.println(Count);
  }
}
```

replace()

→ replace() m is used to replace a particular character or a word in a String.

→ replaceFirst() is used to replace the first occurance of the word from the String.

→ replaceAll() is used to replace a particular word from the Entire String.

* The difference between replace() and replaceAll().

→ The d replace() can replace a character and a word from the String but it doesnot follow regular Expression.

→ replaceAll() can replace a particular word from a String and can follow regular Expression pattern.

Eg:-
```
String Str = "java is a programming language";
String str1 = Str.replace ("a", "k");
System.out.println (Str1);
String str2 = "java is a programming is language";
String str3 = Str2.replace ("is", "was");
System.out.println(Str3);
String Str4 = Str2.replaceFirst ("is", "am");
System.out.println (Str4);

String Str5 = "hello hi hello hi";
String Str6 = Str5.replaceAll ("hi", "ab");
System.out.println (Str6);
```

Output:-
```
jkvk is k progrkmming lkangukge
java was a programming was language
java am a programming was language
hello ab hello ab
```

01/02/23

① WAP to convert each character in a string upper case to lower case (or) lower case to upper case.

```
Scanner S = new Scanner(System.in);
System.out.println("Enter the string");
String Str = S.next();
for(int i=0; i<=Str.length()-1; i++)
{
    char ch = Str.charAt(i);
    if (ch>='A' && ch<='Z')
    {
        int ch1 = ch+32;
        System.out.print((char) ch1);
    }
    Else
    {
        int ch2 = ch-32;
        System.out.print ((char)ch2);
    }
```

StringBuffer

→ A StringBuffer class is available in java.lang package it is a mutable class and it is thread safe.

→ Multiple threads can simultaneously can work on StringBuffer class

→ The Methods of StringBuffer class are thread safe (Synchronized).

→ There are two ways to add the data to StringBuffer

1) StringBuffer sb = new StringBuffer();
sb.append("hello");

2) StringBuffer sb1 = new StringBuffer("hi");

Eg:
```
StringBuffer sb = new StringBuffer();
sb.append("hello");
System.out.println(sb);
sb.insert(1,"java");
System.out.println("sb after insert:"+sb);
sb.deleteCharAt(1);
System.out.println("sb after delete:"+sb);
```

(P) WAP to check given string is palindrome or not using reverse().

```
Scanner s = new Scanner(System.in);
System.out.println("Enter a String:");
String str = s.next();

StringBuffer sb = new StringBuffer(str);

sb.reverse();
String str1 = sb.toString();
if(str.Equals(str1))
{
 System.out.println("palindrome");
}
else
{
 System.out.println("Not palindrome");
}
```

→ The toString() is used to convert StringBuffer to String type.

StringBuilder:

→ StringBuilder class is available in java.lang package. It is a mutable class and it is not thread safe

→ The methods of StringBuilder class are not non Synchronized.

(P). WAP to accept a String from the user and remove a particular word from the string.

2/02/23

Object Oriented programming

→ object means a real world Entity. Whatever we See in the real world can be represented as an object.

→ A class represents group of objects of Similar type.

Types of variables :-

There are 3 types of variables :-

1. Instance variable
2. static variables
3. local variable.

* Instance variables :-

→ The instance variables are also called as Non-static variables.

→ The instance variables will be Created whenever the object is created.

→ Every object will get a Seperate copy of instance variables.

→ We can access instance variables of the class using object name. variable name and using instance methods, Static methods.

Eg :-
```
class Student
{
    int sid;
    String name;
    int phno;
    Static String course = "java";
}
```

```
Student S1 = new Student();
Student S2 = new Student();

S1.Sid = 1001        S1.phnone = 34243
S1.name = "abc";
```

O/p :- Course = "java"

S_1	S_2
Sid = 1001	Sid
name = 'abc'	name
Phno = 34243	phno

* Static variables :-

→ The Static variables are created by using Static Keyword.

→ The static variables are created in method area at class loading time.

→ only a single copy of static variables will be created in the memory.

→ All the objects can access the static variables directly.

Eg:-
```
class student
{
    int Sid;
    String name;
    int phno;
    Static String course = "java";
    public Static void main (String args[])
    {
```

```
Student S1 = new student();
S1.sid = 1001;
S1.name = "abc";
S1.phno = 34243;
System.out.println("course = " + course);
System.out.println("S1.sid = " + S1.sid + "\n" +
        S1.name = " + S1.name + "\n" + "S1.phno = " +
                                        S1.phno);
}
}
```

O/P :- course = java
S1.sid = 1001
S1.name = abc
S1.phno = 34243

* Local variables :-

→ The variables which are created inside a method or a block are called local variables.

→ The local variables are accessible only up-to the method or a block, which it is created.

Eg :-
```
class Test
{
  void m1()
  {
    int x=100;
  }
}
```

→ x is a local variable, the accessibility of x is upto m1 method only

* Create a class called Employee and identify instance variables and static variables and assign values and display.

```
Class Employee
{
    int Empid;
    String name;
    int phno;
    Static String Company = "TCS"
    public Static void main (String args[ ])
    {
        Employee E1 = new Employee ();
        E1.Empid = 101;
        E1.name = "abc";
        E1.phno = 12123;
        System.out.println("Company = " + Company)
        System.out.println ("E1.Empid = " + E1.Empid + "\n"
          + " E1.name = " + E1.name + "\n" + " E1.phno = "
          + E1.phno);
    }
}
```

<u>o/p:-</u>

```
Company = TCS
E1.Empid = 101
E1.name = abc
E1.phno = 12123
```

Creating Methods :-

→ Methods are used to perform a particular task or an operation.

→ There are 2 types :-

1. predefined
2. User defined.

→ We can define methods in 2 ways :-

1. Method with return type.
2. Method with no return type.

→ void indicates no return type.

Syntax :- returntype Methodname (parameters)

```
{

}
```

Eg :- Method with no return type

```
void add (int a, int b)
{
    System.out.println ("Sum = " + (a+b));
}
```

6/2/23

(P) WAP to create a class called Employee with Empid, empname, dept as the instance variables, accept the Employee details using accept() and display using display().

```
public class Employee1 {
    int empid;
    String name;
    String dept;
    Static String Companyname = "packetprep";
```

```
void accept()
{
 Scanner s = new Scanner (System.in);
 System.out.println ("Enter Empid");
 Empid = s.nextInt();
 System.out.println(" Enter name");
 Name = s.next();
 System.out.println(" Enter dept ");
 dept = s.next();
}
void display()
{
 System.out.println ("Companyname :: " + Companyname);
 System.out.println ("Empid = " + Empid + " \n" + " name = "
        + name + " \n" + " dept = " + dept );
}
public static void main(String args[]) {
 Employee1 e = new Employee1();
 e. accept();
 e. display();
}
}
```

(P) Create a class called Customer and Identify Instance and static variable accept the Customer details and display (cId, cname, cadd)

```
public class Customer {
  int cId;
  String cname;
  String cadd;
  int phno;
  Static String company = "Amazon";
  void accept()
  {
    Scanner S = new Scanner(System.in);
    System.out.println(" Enter customerId ");
    cId = S.nextInt();
    System.out.println (" Enter customer name");
    Cname = S.next();
    System.out.println (" Enter Customer Address");
    Cadd = S.next();
    System.out.println(" Enter phonenumber");
    phno = S.nextInt();
  }
  void display()
  {
    System.out.println (" companyname :: " + Company);
    System.out.println (" CId :: " + cId + "\n" + " cname :: " +
      Cname + "\n" + " cadd :: " + cadd + "\n" + " phno :: " +
        phno);
  }
  public static void main (String args[]) {
    Customer C = new customer();
    C.accept();
    C.display();
  }
}
```

(P) Create a class called car and Identify Instance and static variable and define the methods (vehicleTypes, turnupcost()) Accept the details and display.

```
public class car {

  int Vnum;
  int vcost;
  String vcolor;
  Static String vtype = "i20";

  Void vehicleType()
  {
    System.out.println("vehicleType::"+ vType);
  }

  void turnupCost()
  {
    Scanner S = new Scanner(System.in);
    System.out.println("Enter vehicle cost");
    Vcost = S.nextInt();
  }

  void accept()
  {
    Scanner S = new Scanner(System.in);
    System.out.println("Enter vehiclenumber");
    Vnum = S.nextInt();
    System.out.println("Enter vechicle color");
    vcolor = S.next();
  }

  void display()
  {
    System.out.println("vehicle num::"+ vnum + "\n"+
                        "vColor::"+ v color);
```

```
public static void main (String args[]) {
    Car c = new Car();
        c.accept();
        c.vehicleType();
        c.turnupcost()
        c.display();
    }
}
```

7/02/23

Eg:-

```
class Demo
{
    void add(int a, int b)
    {
        System.out.println((a+b));
    }
    Public Static void main (String args[])
    {
        Demo d = new Demo();
        d.add(10,20);
    }
}
```

* Method With return type

→ A Method can return only one value

→ we have to use return Statement as the last Statement.

→ return Statement can return the value to the method calling.

Note:-

In java a method can return an array, object, interface, collection and etc...

Eg:-

```
class Demo
{
  int add ( int a, int b)
  {
    return a+b;
  }
  public Static void main (String args[])
  {
    Demo d = new Demo();
    System.out.println (d.add(1,2));
  }
}
```

(P) Create a method called max() and pass two parameters and return maximum value

```
class ~~Maximum~~ Demo
{
  int max ( int a, int b)
  {
    if (a>b)
      return a;
    Else
      return b;
  }
  public Static void main (String args[])
  {
    Demo d = new Demo();
    System.out.println (d.max(4,5));
  }
}
```

(P) WAP to pass an array of 5 Subject marks to a method and return total marks.

→ To pass an array as a parameter to method, only name of the array we have to pass.

```
public class ArrayDemo {
int getTotalMarks( int marks[])
{
  int Sum=0;
  for(int i: marks)
  {
    Sum = Sum+i;
  }
  return Sum;
}
public static void main(String args[])
{
  ArrayDemo a = new ArrayDemo();
  Scanner s = new Scanner(System.in);
  int marks[] = new int [5];
  System.out.println(" Enter the marks ");
  for (int i=0; i<marks.length; i++)
  {
    marks [i] = s.nextInt();
  }
  System.out.println(" Total Marks :: " + a.getTotal
                                          Marks(marks));
}
}
```

(P) WAP to pass name as a parameter to a method and return a welcome message Welcome name.

(P) WAP to pass two arrays as a parameter to a method and return true if array a first Element is equal to array b last Element, otherwise return false.

(3) WAP to pass an array as a parameter to a method and return Sum of last three Elements:

~~(a) public class Welcome(String name)~~

```
(1) public class WelcomeReturn {
    String Message (String name)
    {
      return ~~name;~~ "welcome" + name;
    }
    public Static void main (String args[])
    {
      WelcomeReturn w = new WelcomeReturn();
      System.out.println (w. ~~welcome~~ Message ( " ~~welcome~~ india"));
    }
}
```

```
String getMessage(String name)
{
return "welcome" + name;
}
```

(2)

```
import java.util.Scanner;
public class CommonElement {
  boolean Common ( int a[], int b[])
  {
    ~~if (a[0] == b[4])~~
    if (a[0] == b[b.length-1])
    {
      return true;
    }
    Else
    {
      return false;
    }
  }
  public Static void main (String args[]) {
    CommonElement c = new CommonElement();
    int a[] = new int[5];
    int b[] = new int[5];
    System.out.println("Enter ~~array~~ elements of a ~~Elements~~ array:");
    for (int i=0; i<5; i++)
    {
      a[i] = S.nextInt();
    }
    System.out.println("Enter array elements of b:");
    for (int i=0; i<5; i++)
    {
      b[i] = S.nextInt();
    }
    System.out.println(c.common(a,b));
  }
}
```

③

```
import java.util.Scanner;
public class LastSumReturn
{
    int sum=0;
    int array (int a[])
    {
      for (int i=a.length-1; i>=a.length-3; i--)
      {
        sum = sum +a[i];
      }
      return sum;
    }
    public static void main (String args [])
    {
      LastsumReturn l = new LastSumReturn();
      Scanner s = new Scanner (System.in);
      int a[] = new int [6];
      System.out.println("Enter array elements:");
      for (int i=0; i<a.length ; i++)
      {
        a[i] = s.nextInt();
      }
      System.out.println(l.array(a));
    }
}
```

Access Modifiers.

- private
- protected
- default
- public

→ private members are accessable within the Same class.
private members can't be accessable to the Subclasses.

→ protected members are accessable within the same package and Subclasses of the other package.

Note:- A class can't be defined as private and protected in java.

→ default or package access modifier

default members are accesable upto the Same package.
The default access modifier in java is default.
When we don't define access modifier for class, variable, method java will treat it as default.

→ public

public is universal access modifier.
public members are accessable any where

10/02/23.

Object class

→ object class is in java.lang package.

→ Object is the Superclass for all the classes in java.

→ There are 11 methods in object class, all the 11 methods are available to all the java classes.

1) hashcode()
2) Equals()
3) String toString()
4) class getClass()
5) clone()
6) finalize()
7) wait()
8) wait(long)
9) wait(long, int)
10) notify()
11) notifyAll()

(P) WAP on HashCode.

```
public class HashCodeDemo {
    public Static void main (String args[]) {
        HashCodeDemo h = new HashCodeDemo();
        System.out.println(h.hashCode());
    }
}
```

Output:

1072591677

→ object class all the methods we can use on any java c class.

→ hashCode() method will return and unique number called hashCode.

* Static Keyword

→ Static Keyword can be used for

1) Variables
2) methods
3) blocks
4) nested classes

* Static variables

→ When we define a variable as a static, the value of the variable will be defined only once.

→ Single copy of the static variables will be available in the memory

* Static methods

→ We can define a method as a static.

```
static void display()
{
}
```

→ Static methods can access Static variables and instance Variables of the class. can't be accessable by static Method.

→ We can call Static methods without using object (or) classname.methodname.

13/03/23

Why Static methods can't access instance variables of the class?

A) When static methods are called, the copy of the instance variables are not available in the memory.

*Static Block

→ Static block is created by using static keyword.

→ Static block is called by JVM at class loading time.

→ block doesn't have any name

```
static
{
  System.out.println("static block");
}
```

Eg:-

```
public class StaticDemo {
  Static
  {
    System.out.println("Static block1");
  }
  Static
  {
    System.out.println("static block2");
  }
  public static void main(String args[])
  {
    System.out.println("Main Method");
  }
}
```

output : Static block1
static block2
Main Method

final Keyword

→ final Keyword is used for
variable
method
class

→ final variable value can't be changed.

```
final int x = 10;
```

→ x value 10 is fixed. It is not possible to increment or decrement or change the value.

→ final methods can't override.
The Subclass can't override the final methods.

→ final class can't be Subclassed.

→ we can't create a Subclass for final class.

Eg:-
```
final class A
{
}
class B Extends A ——xx
{
}
```

* Static nested class

```
class A ——→ outer class
{
    class B ——→ nested class
    {
    }
}
```

We can make static as nested class.

Constructor

→ Constructors are used to initialize instance variable of class.

→ Constructor have same name as class name.

→ Constructors are Executed when object of the class is Created.

→ Constructor doesn't return any value.

→ There are two types of Constructors

> non-parameterized

> parameterized

→ When we don't create a Constructor, the default Constructor will be called by jvm.

Eg:-

```
public class Student {
    Student()
    {
        System.out.println("non-parameterized Constructor");
    }
    public static void main(String[] args)
    {
        Student s = new Student();
    }
```

Output:-

non-parameterized Constructor.

Eg:- public class Student {

```
int Sid;
String name;
int age;
Student()
{
System.out.println("non-parameterized Constructor");
}
Student(int Sid, String name, int age)
{
this.Sid = Sid;
this.name = name;
this.age = age;
}
public static void main(String args[])
{
Student S1 = new Student(1001, "abc", 22);
}
}
```

→ In the above program, the this keyword is used to identify the instance variable.

→ this always refers to instance variables.

→ In a program we can write multiple constructors.

→ When we create a parameterized constructor object should be created to call the constructor.

* Differences between method and Constructor

→ method can have any name

*→ Constructor have same name as the class name

→ Method can return a value

*→ Constructor doesn't return a value

→ methods are used to perform the calculations

*→ Constructors are used to initialize instance variables

→ methods has to be called Explicitly

*→ Constructors will be called when object of the class is Created.

Constructor chaining

→ Calling one Constructor inside another Constructor within the Same class is called Constructor chaining

→ we can use this() in Constructor chaining

Eg:-
```
public class Emp{
    int Empid;
    String name;
    Emp() {
    System.out.println("non-parameterized Constructor");
    }
    Emp(int Empid)
    {
      this();
      this.Empid = Empid;
      System.out.println ("Empid :: "+Empid);
    }
    Emp (int Empid, String name)
    {
      this (Empid);
      this.Empid = Empid;
      this.name = name;
      System.out.println ("Empid::"+Empid+"\n"+"name::"
                                    +name);
    }
```

```
public static void main(String args[]) {
    Emp e = new Emp(1002, "abc");
  }
}
```

14/02/23

Encapsulation

→ Encapsulation is the process of wrapping up data and methods in a single unit is called class.

→ We can achive Encapsulation by defining the variables of the class as a private and using public setter and getter method.

→ Setter methods are used to save the data

→ getter methods are used to get the data.

Eg:-
```
public class Customer {
    private int cid;
    private String cname;
    public void setcid(int cid)
    {
      this.cid = cid;
    }
    public void setcname(int cname)
    {
      this.cname = cname;
    }
    public int getcid()
    {
      return cid;
    }
    public String getcname()
    {
      return cname;
    }
```

```
public static void main (String args[])
{
    Customer c = new customer();
    Scanner S = new Scanner (System.in);
    System.out.println ("Enter cid");
    int cid = S.nextInt();
    c.Setname (cno
    c.setcid (cid);
    System.out.println(" Enter cname");
    String cname = S.next();
    c.setcname (cname);
    System.out.println ("CID = " + c.getcid());
    System.out.println ("CName = " + c.getCname());
  }
}
```

→ Encapsulation provides data Hiding

Abstraction

→ Abstraction is used to Hide unnecessary data and provides only necessary data.

→ Abstraction can be achieved by using abstract class and interface

→ Abstract class does not provide 100% abstraction. Interface provides 100% abstraction.

Eg:- If we take remote as an object How Each button in the remote will work as a user we don't know. But How to operate a remote as user can know.

→ Abstract class of java can have abstract and non abstract methods.

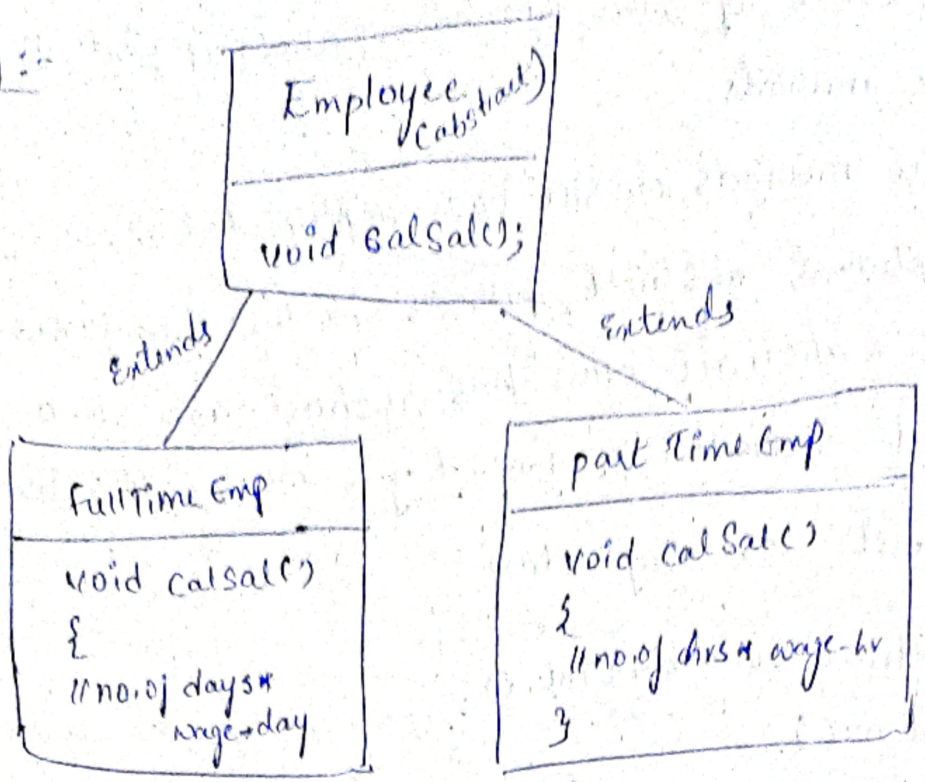
→ Abstract methods doesn't have implementation.

→ We implement abstract methods in the Subclasses.

→ Because abstract class has abstract and non abstract methods, it doesn't give 100% abstraction.

Eg:-
```
abstract class Superclass
{
  abstract void cal(int x);
  void m1()
  {
    System.out.println("m1 method");
  }
}
class Subclass1 Extends Superclass
{
  void cal(int x)
  {
    System.out.println("Square val" + (x * x));
  }
}
class Subclass2 Extends Superclass
{
  void cal(int x)
  {
    System.out.println("Square val" + (x * x * x));
  }
  public static void main(String args[]){
    Subclass1 s1 = new Subclass1();
    s1.m1();
    s1.cal(90);
    Subclass2 s2 = new Subclass2();
    s2.m1();
    s2.cal(100);
  }
}
```

Eg :-



Eg :- File 1

```
public abstract class Employee {
int cid;
String ename;
Employee (int cid, String ename)
{
   this.cid = cid;
   this.ename = ename;
   System.out.println(" Eid :: "+ cid+ " \n" + "Ename:" + ename);
}
abstract void calsal();
}
```

File 2

```
public class FullTimeEmp Extends Employee {
   FullTimeEmp (int cid, String ename)
   {
      super(cid, ename);
   }
```

```
void calSal()
{
  Scanner S = new Scanner(System.in);
  System.out.println("Enter no. of days");
  int days = S.nextInt();
  System.out.println("Enter wage per day");
  int wage_day = S.nextInt();
  System.out.println("Sal for FullTimeEmp::" +
                    (days * wage_day));
  }
}
```

File 3

```
public class partTimeEmp Extends Employee {
  partTimeEmp(int Eid, String Ename) {
    Super(Eid, Ename);
  }
  void calsal() {
  Scanner S = new Scanner(System.in);
  System.out.println("Enter no. of hrs");
  int hrs = S.nextInt();
  System.out.println("Enter wage per hr");
  int wage_hr = S.nextInt();
  System.out.println("Salary for partTime::" + hrs *
                    wage_hr));
  }
  public static void main(String args[]) {
  Employee e1 = new FullTimeEmp(1001, "abc");
  e1.calsal();
  Employee e2 = new partTimeEmp(1002, "xyz");
  e2.calSal();
  }
}
```

15/02/23

→ We can't create object of abstract class and an Interface.

→ Abstract class can have Constructor.

→ We can call abstract class Constructor from Subclass using Super keyword.

Why we can't create object of abstract class and an interface?

A) When object is created the memory should be allocated. Abstract class have abstract methods, which are unimplemented and interface all the methods are public and abstract by default. So, jvm can't decide how much memory to allocate for the objects.

Interface.

→ Interface provides 100% abstraction.

→ All the methods of an interface are by default public and abstract.

→ from jdk 1.8 onwards we can also define default and static methods in the interface.

→ interface is defined by using interface keyword.

→ variables of an interface are public static final.

```
interface I1
{
    int x=10;   // public static final
    void m1();  // public abstract
}
```

→ Between class and an interface we use implements keyword.

→ When class implements from an interface all the methods of the interface should implement in subclasses.

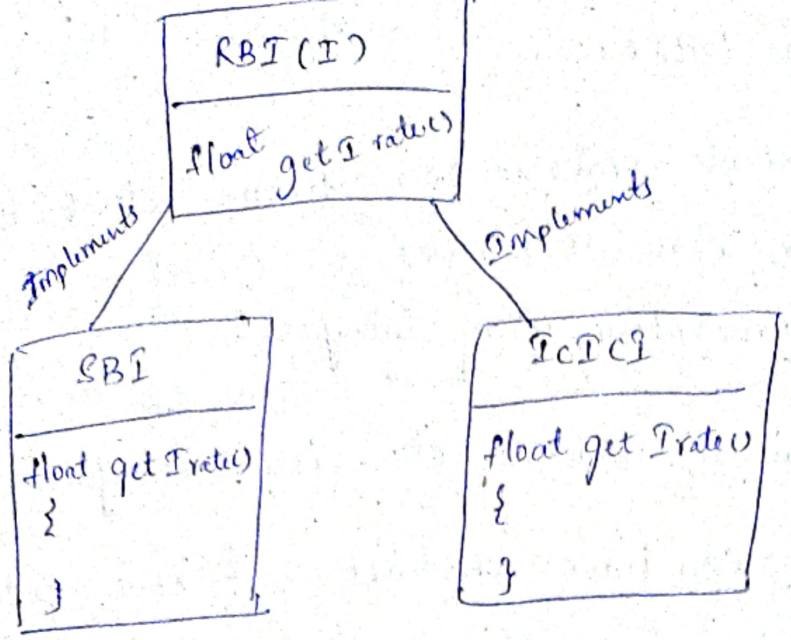
→ In Java a class can Extend from a single class but can implement from multiple interfaces.

Eg:-

```
interface I1
{
  void m1();
  void m2();
}
class A implements I1
{
  public void m1()
  {
    System.out.println("m1 method of I1 Interface");
  }
  public void m2()
  {
    System.out.println("m2 method of I1 interface");
  }
  public static void main (String args[])
  {
    A a = new A();
    a.m1();
    a.m2();
    I1 i = new A();
    i.m1();
    i.m2();
  }
}
```

Eg:- for Subinterface

```
interface I1
{
  void m1();
}
interface I2 Extends I1
{
  void m2();
}
Class A implements I2
{
  public void m1()
  {
    System.out.println("m1 method");
  }
  public void m2()
  {
    System.out.println("m2 method");
  }
  public static void main(String args[])
  {
    A a = new A();
    a.m1();
    a.m2();
    I2 i = new A();
    i.m1();
    i.m2();
    I1 i1 = new A();
    i1.m1();
  }
}
```



Eg:- interface RBI

```
{
  float getIrate();
}
Class SBI implements RBI
{ public float getIrate()
  { return 8.75f;
  }
}
public class ICICI implements RBI {
  public float getIrate()
  {
    return 8.9f;
  }
  public static void main (String[] args) {
  RBI r = new SBI();
  System.out.println("SBI :: "+r.getIrate());
  RBI r1 = new ICICI();
  System.out.println(" ICICI :: "+r1.getIrate());
  }
}
```

RBI r = new SBI();

→ interface reference variable can ~~referene~~ refer to the implementation class at runtime is called Runtime polymorphism using interface

* Difference between abstract class and interface:

→ Abstract class can have abstract and non abstract methods

→ Interface all the methods are by default pubic and abstract by default. From jdk1.8 onwards we can create Static and default methods.

→ Abstract class doesn't provide 100% abstraction
→ Interface provides 100% abstraction.

→ Abstract class abstract methods are unimplemented.
→ Interface all the methods are unimplemented

→ Abstract class is defined using abstract keyword
→ Interface is defined using interface keyword

→ Multiple inheritance can't be achieved to using class
→ Multiple inheritance can done using interface.

17/02/23

Inheritance

→ Inheritance is a process of acquiring properties and methods from Superclass to Subclass.

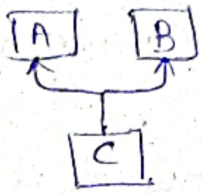
→ Inheritance provides code reusability.

→ We use Extends Keyword in inheritance

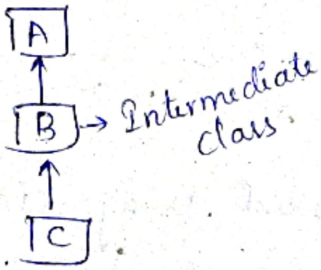
Types of Inheritance

1) Single Inheritance

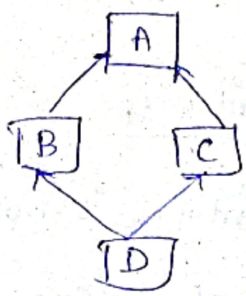
2) Multiple Inheritance



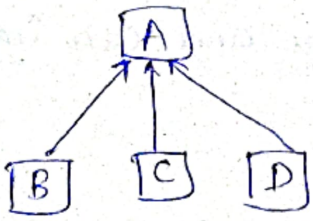
3) Multilevel Inheritance



4) Hybrid Inheritance



5) Hierarchical Inheritance



1) Single Inheritance

→ In Single Inheritance we have Single Superclass and Single Subclass.

→ The Subclass can take the properties from Superclass

Eg:-

```
class A
{
  void methodA()
  {
    System.out.println("methodA of class A");
  }
}
class B Extends A
{
  void methodB()
  {
    System.out.println("methodB of class B");
  }
  public static void main(String args[])
  {
    B b = new B();
    b.methodA();
    b.methodB();
  }
}
```

2) Multiple Inheritance

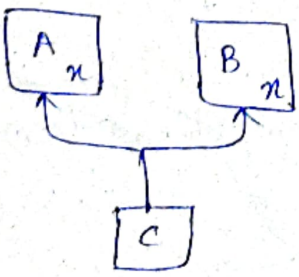
→ Multiple Inheritance is not Supported using class

→ In java a class cannot Extend from multiple classes But, a class can implement from multiple Interfaces.

→ A class can Extend from a class and can implement from a interface

→ Multiple Inheritance can be achieved using Interface.

• Why java dosnot Support multiple Inheritance using class?



→ In the above diagram class A and B having same variable x.

→ When we create an object of C class

```
C c = new C();
```

and when we access C.x;

which x variable it has to access (A class x variable or B class x variable is a confusion to the compiler)

```
class A
{
}
Class B
{
}
```

```
Class C Extends A,B ——xxx
{
}
```

→ Class Implements from multiple Interfaces

```
interface I1
{
}
interface I2
{
}
```

```
Class A implements I1,I2
{
}
```

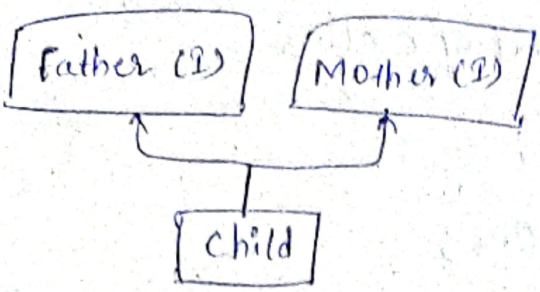
→ class Extends from a single class implements from multiple interfaces.

```
interface I1
{
}
interface I2
{
}
```

```
class A
{
}
```

```
class B Extends A implements I1,I2
{
}
```

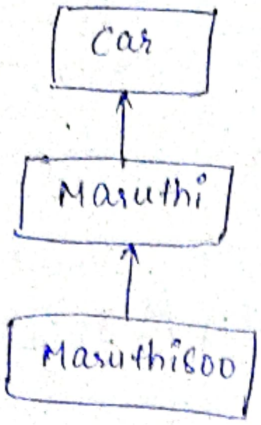
Multiple Inheritance using Interface



→ In java multiple Inheritance can achieved using Interface

```
interface Father
{
  float ht = 5.7f;
}
interface Mother
{
  float ht = 5.4f;
}
class child implements Father, mother
{
  void calHeight()
  {
    System.out.println("avg child height :: " +
                        (Father.ht + Mother.ht)/2);
  }
  public static void main (String args[])
  {
    Child c = new child();
    c.calHeight();
  }
}.
```

3) Multilevel Inheritance




```
class Car
{
 void vehicleType()
 {
  System.out.println("Vehicle Type :: car");
 }
}
class Maruthi Extends car
{
 void Speed_Maruthi()
  {
   System.out.println("Speed of maruthi :: 80kmph");
  }
}
class Maruthi800 Extends Maruthi
{
  void Speed_maruthi800()
  {
   System.out.println(" Speed of maruthi800 :: 90kmph");
  }
  public Static void main (String args[])
  {
    Maruthi800 m = new Maruthi800();
    m.vehicleType();
    m.Speed_maruthi();
    m.speed_maruthi800();
  }
}
```

5) Hierarchical Interface

```
class A
{
  void method()
  {
    System.out.println("Method of A");
  }
}
class B Extends A
{
  void method1()
  {
    System.out.println("Method1 of B");
  }
}
class C Extends A
{
  void method2()
  {
    System.out.println("Method2 of C");
  }
  public Static void main(String args[])
  {
    C c = new C();
    c.method();
    c.method1();
    c.method2();
  }
}
```

20/2/23

polymorphism

→ poly means many morphs means forms.

→ The polymorphism means ability to have multiple forms

Eg:- A single Stadium is used to play multiple games.
A single object is used in multiple ways is an example of polymorphism.

Eg:- A Software developer perform multiple operations to develop a code.

* First he has to understand the requirements, requirements has to be converted to design, from the designing coding has to be done, after coding the application has to be tested.

* This multiple operations is performed by programmer

→ polymorphism can be done in two ways:
1) Compile time (Static polymorphism)
2) Run time (Dynamic polymorphism).

→ The Method overloading comes under Static polymorphism.

→ In Method overloading we write two or more methods within the Same class but with different in parameters (no. of parameters, type of parameters, order of parameters).

→ The java Compiler will identify the method to Execution based on the parameters

→ Method Overloading will be done within the Same Class.

Eg:-

```
class OverloadingDemo {
    void add (int a, int b)
    {
      System.out.println ("sum = "+(a+b));
    }
    void add (int a, int b, int c)
    {
      System.out.println ("sum = "+ (a+b+c));
    }
    void add ( int a, float b)
    {
      System.out.println( "sum:" + (a+b));
    }
    void add (float a, int b)
    {
      System.out.println ("sum = "+ (a+b));
    }
    public static void main (String args[])
    {
      OverloadingDemo O = new OverloadingDemo();
      O.add (10, 20);
      O.add (10, 20, 30);
      O.add (1, 5.0f).
      O.add ( 5.6f , 10);
    }
}
```

Method overriding

→ In Method overriding, The Subclass will override Superclass method.

→ A Subclass can access the properties and methods from the Superclass.

→ If a Subclass want to provide a new functionality to the Superclass method, i.e. It can override the method.

→ At run time the JVM will call the Superclass method if we use Super keyword.

Eg:-
```
class A
{
 void cal (int x)
  {
   System.out.println("Square::"+(x*x));
  }
}
class B Extends A
{
 void cal (int x)
  {
   Super.cal(5);
   System.out.println("cube::"+(x*x*x));
  }
  public Static void main(String args[])
  {
    B b= new B();
    b.cal(3);
  }
}
```

Output

25
27

21/03/23

* Differences between Method overloading & Method Overriding

→ Method overloading comes under compile time polymorphism
→ Method overriding comes under run time polymorphism

→ Overloading will be done between the Same class
→ Overriding will be done from Superclass to Subclass

→ In Method Overloading we write two or more methods with the Same name, but different with parameters
→ In Method overriding the Subclass Method should be identical to the Superclass.

→ The Compiler will identify the method at compile time based on the methods parameters.
→ By using Super Keyword we can call the Superclass Method in the Subclass.
During the run time the JVM will identify Superclass method to Execute.

Aggregation

→ In Aggregation one class can be the part of another class, One Class properties we can use in another class.

→ The aggregation is called has-a relationship

Eg:-

```
class Address        class Emp        class Student
{                    {                {
int streetno;        int Empid;       int Sid;
String city,state;   String name;     String name;
                     Address a;       ~~Emp e;~~
}                    }                Address a;
                                      }
```

→ The Emp class have has a relationship with address class.

→ we can use address properties in another classes.

Eg:- public class Address

```
{
  int streetno;
  String city, State, zipcode;

  Address (int streetno, String city, String state,
                                String zipcode)
  {
     this.streetno = streetno;
     this.city = city;
     this.state = state;
     this.zipcode = zipcode;
  }
}
```

File2: Eg:-

```
public class Emp {
  int empid;
  String name;
  Address address;

  Emp (int empid, stringname, Address address)
  {
    this.empid = empid;
    this.name = name;
    this.address = address;
    System.out.println("Empid:: "+ empid + "\n" + "Name=" +
                                                  name);
    System.out.println ("Streetno:: " + address.streetno + "\n" +
      "City:: " + address.City + "\n" + "State:: " + address.State + "\n" + "zipcode:"
      + address.zipcode);
  }
```

```
public static void main (String[] args)
{
    Address a = new Address (13, "hyd", "ts", "500072");
    Emp e = new Emp (101, "abc", a);
}
}
```

→ Aggregation shows weak relationship between the Entities.

→ In Aggregation one class can Exist without an another class also.

Composition

→ one class is a composition of another class, one class cannot Exists without an another class, is called composition.

→ Composition shows Strong relationship between the Entities.

Eg:- The relationship between car and Engine

Eg:- file-1:

```
public class Engine
{
    public void work()
    {
        System.out.println(" Engine is working");
    }
}
```

file 2

```
public class car
{ final Engine Engine;
  public car ( Engine Engine)
  { this. Engine = Engine;
  }
  void Start()
  {
    Engine.Work();
    System.out.println("car Started");
  }
  public static void main( String args[])
  {
    Engine  c = new Engine();
    Car  c = new Car(c);
    c.Start();
  }
}
```

Exception Handling

→ Exception means abnormal Situation which interupts the normal flow of the program.

→ Exception Handling is the mechanism to handle the abnormal Situation.

→ The Commonly occur Exceptions are when user Searching for a file, if the file is not available in the given path. then Exception will occur.

→ When user uploads a wrong format file then also Exception will occur

→ In Java all the Exceptions and Errors throwable is the Super Class.

→ Throwable class has two subclasses one is Exception class and second is Error Class.

→ The Errors will occur due to the problem of resources.

→ Errors are irrecoverable. The Common Errors occurs due to the problem of memory and resources.

→ The throwable class is defined in java.lang package.

23/02/23

There are two types of Exceptions : They are

1) Checked

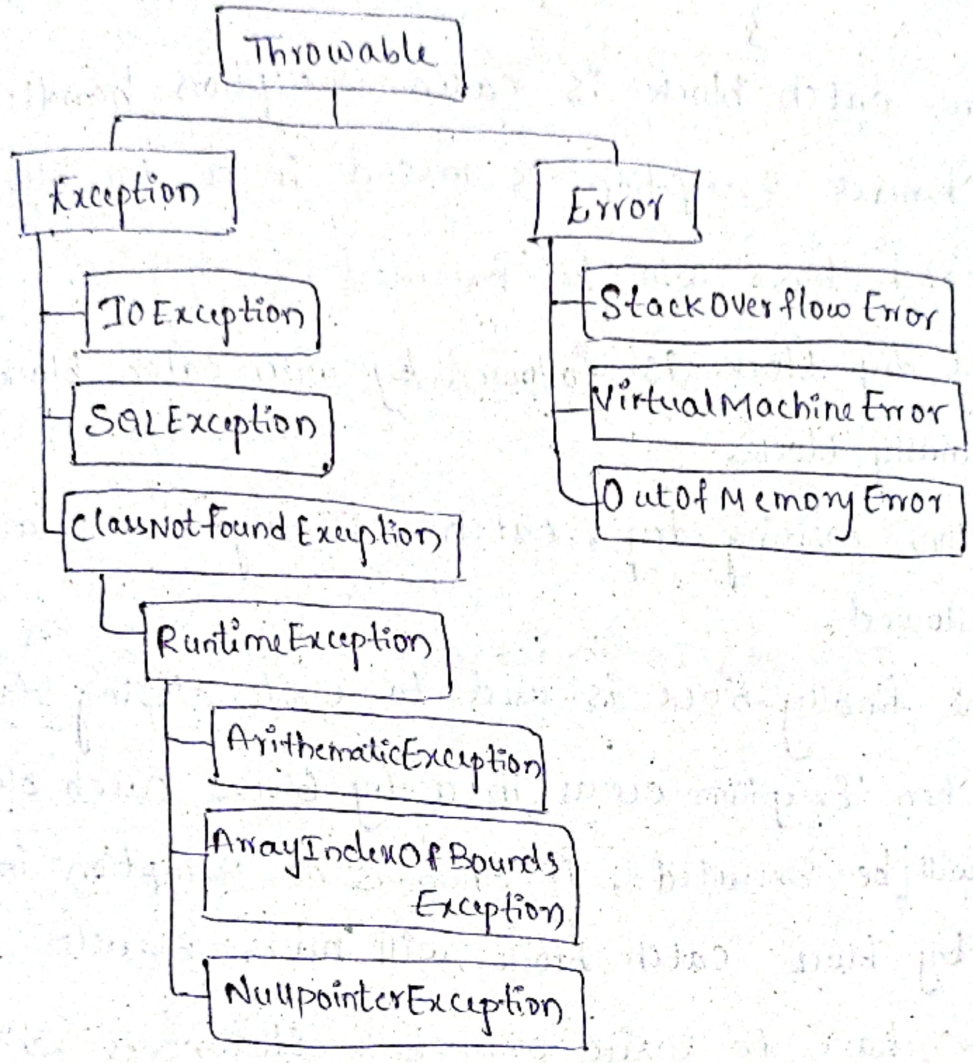
2) Unchecked

Checked Exceptions will be raised at Compile time and Unchecked Exceptions will be raised at run time

→ The programmer compulsory should handle the checked exceptions.

→ unchecked Exceptions are raised at run time. it is not compulsory to handle Exception.

→ when we handle the Exception the normal flow of the program will be maintained.



→ The classes which are directly the subclasses of Exception class are called checked Exceptions.

The classes which are subclasses of runtime Exception class are called unchecked Exceptions.

Exception Handling Keywords

1) try 2) catch

The try block we have the Statement which gives Exception.

```
try        catch()
{          {
}          }
```

→ The catch block is called Exception handler. Whenever Exception is raised in a try block catch block will be Executed.

→ The try block is followed by with catch block (or) finally block.

→ Alone writing try, catch, finally blocks are not allowed.

→ The finally block is used to write closing Statements

→ When Exception occurs in a try block, catch block will be Executed, if there is no Exception in the try block catch block will never Execute.

→ we have to write only both statements which will give an Exception inside the try block

→ A try block can follow with multiple catch blocks but with the Single finally block

* There are three way to print Exception details to the user.

1) printing the Exception Object

When we print Exception object it will print Exception class name, Exception message but it will not print

the line number.

2) Using getMessage()

It will print only the Exception message. It is available in throwable class.

3) printStackTrace()

This Method will print Exception class name, line number and Exception Message.

This method is available in throwable class.

```
try
{
 System.out.println(100/0);
}
catch(ArithmeticException ae)
{
 System.out.println(ae);
}
```

output : java.lang.ArithmeticException: / by zero.

Arithmetic Exception :-

When we divide a number by zero, we will get ArithmeticException.

~~int a[] = {1,2,3,4,5}~~

Eg :- System.out.println(100/0);

ArrayIndexOutOfBoundsException

```
int a[] = {1,2,3,4,5}
System.out.println(a[5]);
```

The above code will give ArrayIndexOutOfBoundsException

→ When we try to access an array element out of the Index we will get ArrayIndexOutOfBoundsException.

Null Pointer Exception.

```
String s = null;
System.out.println (s.length());
```

The Null pointer Exception will be raised when a user try to ~~refer~~ refer a null referenced object.

Eg:-
```
public class ExceptionDemo {
  public static void main (String args[])
  {
    try()
    {
      System.out.println (100/0);
      System.out.println ("hi");
    }
    catch(ArithmeticException e)
    {
      e.printStackTrace();
    }
  }
}
```

Output:-

```
java.lang.ArithmeticException : / by Zero.
  at P1. ExceptionDemo. main (ExceptionDemo.java:8)
```

In the above program System.out.println ("Hi"); is not giving any Exception. So we should not write that statement inside the try block

Ex:-
```
public class Demo {
  public static void main (String args[]) {

    int a[] = {1,2,3,4,5};
```

```
try
{
  System.out.println(a[5]);
}
Catch ( ArrayIndexOutOfBoundsException ae)
{
  System.out.println(ae);
}
finally
{ System.out.println("-finally block");
}
}
}
```

Output

java.lang.ArrayIndexOutOfBoundsException : Index 5 out of bounds for length 5

finally block.

→ The finally block always will Execute regardless of an Exception.

→ we write only the closing statements inside the finally block

→ The possible ways of writing try, catch, finally blocks

1)
```
try
{
}
Catch()
{
}
```

2)
```
try
{
}
finally
{
}
```

3)
```
try
{
}
Catch()
{
}
finally
{
}
```

4)
```
try
{
}
Catch()
{
}
Catch()
{
}
finally
{
}
```

5)
```
try
{

}
```
——— xx

6)
```
catch()
{

}
```
——— xx

7)
```
finally
{

}
```
——— xx

(Nested try catch)

8)
```
try
{
}
Catch()
{
  try
  {
  }
} Catch()
  {
  }
}
```

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

10)
```
try
{
  try
  {
  }
  Catch()
  {
  }
}
Catch()
{
}
```

try with multiple catch blocks

→ A try block can follow with multiple catch() blocks, with a single finally block

→ The statements which we write inside the try block if it leads to multiple Exceptions then, try with multiple catch() blocks we can write

→ We can pass arguments to the main Method at run time, is called command line arguments.

→ Main Method takes the arguments as String type.

→ When we pass an integer arguments to the main method, we can convert these arguments to integer format by using Integer.parseInt().

→ If we pass the arguments as String when we convert into the number type we get number format Exception.

Eg:-
```
public class Test {
    public static void main (String args[]) {
    try
    {
      int n1 = Integer.parseInt(args[0]);
      int n2 = Integer.parseInt(args[1]);
      System.out.println(n1+n2);
    }
    catch (ArrayIndexOutOfBoundsException ne)
    {
      ne.printStackTrace();
    }
  }
}
```

```
javac Test.java
java Test 10 20
```

In the above program we are passing two integer arguments, there is no Exception in the program.

O/P: 30

```
javac Test.java
java Test
```

It will lead to ArrayIndexOutOfBoundsException

```
javac Test.java
java Test abc xyz
```

It will lead to NumberFormatException.

24/02/23

throw Keyword

→ the throw Keyword is used to throw the Exception Explicitly.

→ The throw is followed with Exception class object.

→ When we throw the Exception Explicitly, we have to handle the Exception. using throws (or try, catch

→ The throws Keyword is used to declare the Exception in the application.

→ Use throw and throws keywords a user defined Exception Handling

User Defined Exceptions (or Custom Exceptions

→ The programmer can define its own Exceptions are called Custom Exceptions.

→ In custom Exceptions the Exception class should Extends from ~~the~~ Exception class (or run time Exception class.

(P) WAP to accept age of a person, and check if the person is Eligible for the vote or not.

Create an Exception class InvalidAgeException

Eg:-
```
public class InvalidAgeException Extends Exception
{
  InvalidAgeException(String str)
  {
    Super (Str);
  }
}
```

```
public class AgeDemo {
void accept (int age) throws InvalidAgeException
{
 if (age<18)
   throw new InvalidAgeException ("age is <18");
 else
   System.out.println("Welcome to vote");
}
public static void main (String args[]) throws
                        InvalidAgeException {
  AgeDemo a = new AgeDemo();
  a.accept(12);
  }
}
```

(P) WAP to create a class called Bank with deposit(), withdraw() and balanceCheck() as the methods. If withdrawamount is more than balance amount create an Exception class called InsufficientFundsException.

file(1)

```
public class InsufficientFundsException Extends Exception
{
  InsufficientFundsException (String str)
  {
    Super(str);
  }
```

File ①

```
public class Bank {

    int amount = 1000;

    void deposit ()
    {
        Scanner s = new Scanner (System.in);
        System.out.println (" Enter deposite amount ");
        int depAmount = S.nextInt ();
        amount = amount + depAmount;
        System.out.println (" amount ~~+depAmount~~ );
    }

    withdraw () throws InsufficientFunds Exception
    {
        Scanner S = new Scanner (System.in);
        System.out.println ("Enter withdraw amount");
        int withdrawAmt = S.nextInt();

        if (withdrawAmt > amount)
        {
            throw new InsufficientsFunds Exception ("No Enough amount");
        }
        Else
        {
            amount = amount - withdrawAmt;
            System.out.println (amount); ~~(withdrawAmt)~~;
        }
    void balance check ()
    {
        System.out.println ("Balance amount : " + amount);
    }
    public static void main (String args[]) throws
                                InsufficientFunds Exception
    {
```

```
Bank b = new Bank();
    b.deposit();
    b.withdraw();
    b.balanceCheck();
  }
}
```

StackOverflowError

→ When stack memory is not sufficient to store the entries StackOverflowError will occur.

→ This exception will occur when condition is not specified for the loop, loop will be infinite.

→ When we don't give condition in recursion program this error will occur.

Outofmemory error

→ When heap memory is not sufficient to store the object this error will occur.

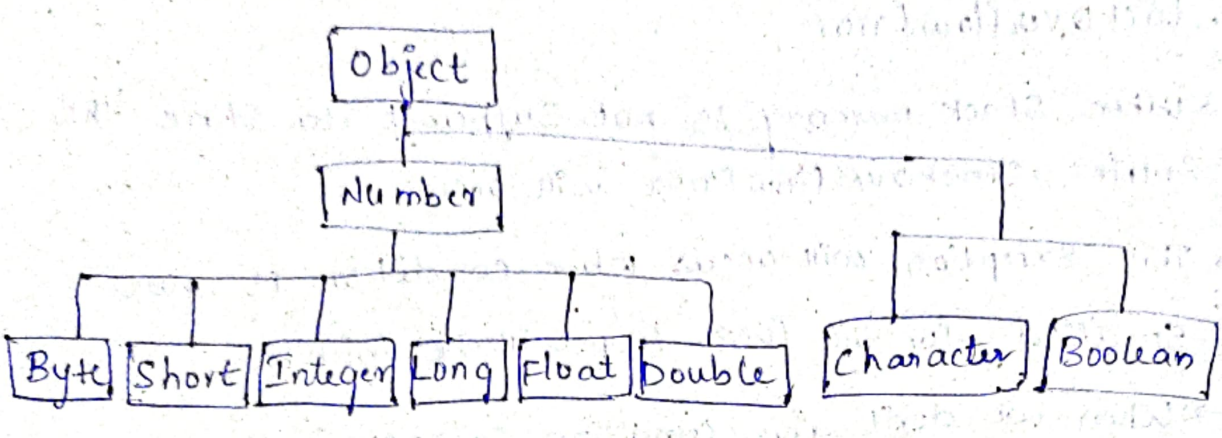
→ we can expand the heap memory explicitly.

27/2/23

Wrapper classes

→ For Each primitive type of java there is an Equivalent wrapper class type is defined.

→ The Collections frame work of java we use Objects and generic collections, In generic generic collections we use wrapper classes.



For Each primitive type there is an Equivalent Wrapper class type.

DataTypes	Wrapperclass
byte	Byte
Short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

There are two types of Conversion in wrapper classes

1) auto-boxing
2) unboxing

→ In auto-Boxing we convert primitive type to an equivalent wrapper class type.

Eg:- int a = 10;
Integer i = new Integer(a);

→ In unboxing we convert wrapper class type to primitive type.

Eg:- Integer x = new Integer(100);
int y = x;

Collection Framework

→ Collection represents group of objects. A framework will have an architecture set of class and interfaces to work with collections.

→ The Collection Framework is represented in java.util package

→ There are two types of collections.

1) Non-Generic Collection (or) Raw Collection:

* The non-Generic collection works with any type of objects.

2) Generic Collection

* Generic collection works with same type of objects.
* Generic collection provides type safty because it will allow same type of objects.

→ Collection frame work internally uses data structures to organise the data.

Differences between Array and Collection.

① → The Array can store same type of values.
→ The collection can store same type of objects and different types of objects.

② → In Array the Elements are stored in Index position.
→ We can do Index based and non-Index based operations in Collections.

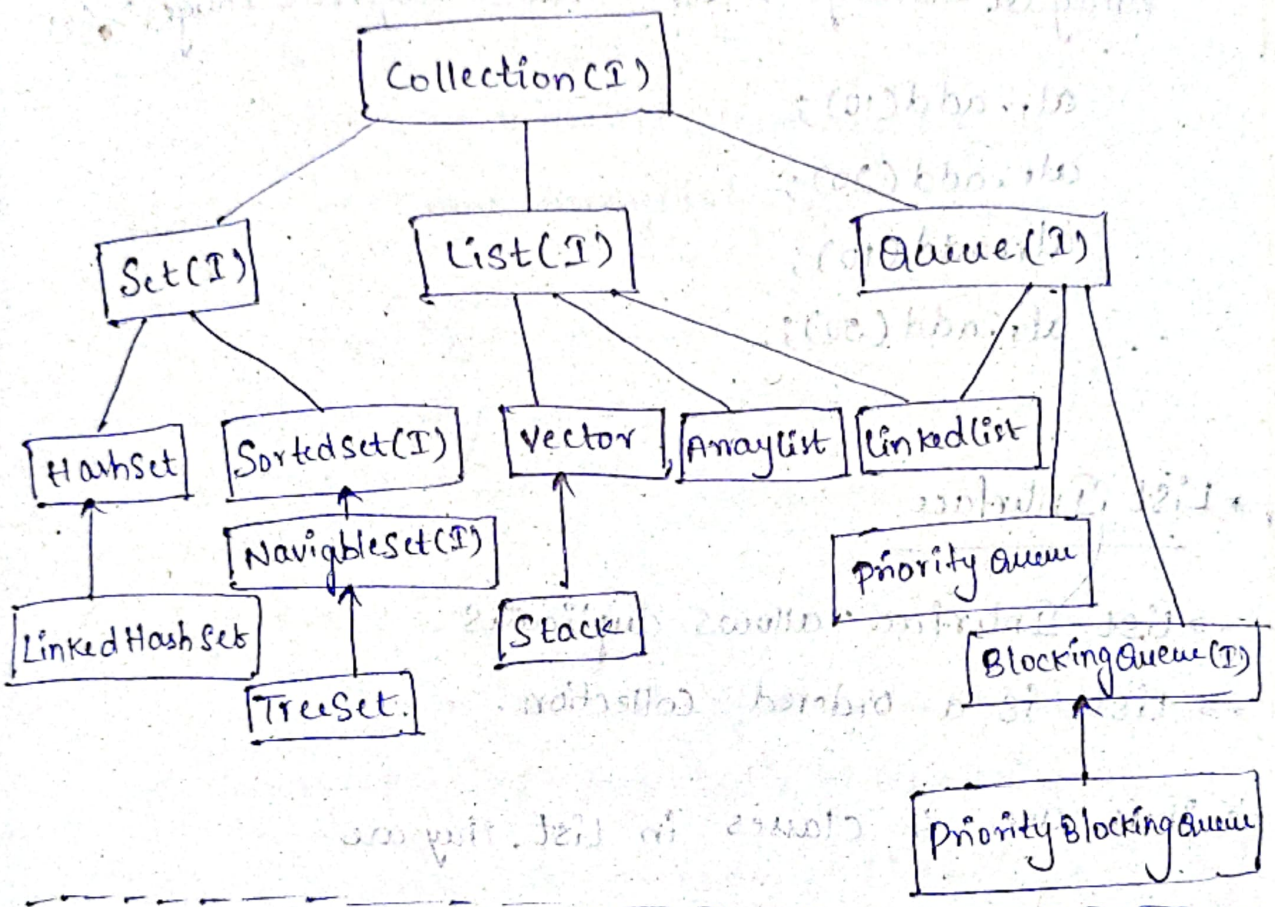
Q7 Adding and deleting an Element from the middle of the Array is difficult to perform.

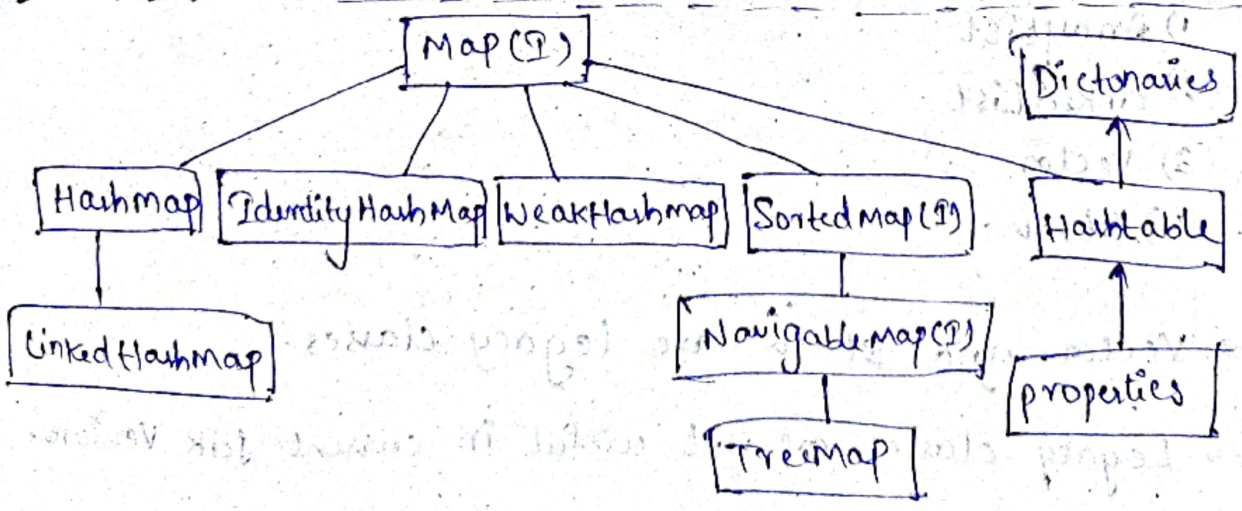
→ We can add an delete an Element from any position of the Collection.

Iterable

→ Iterable is the Super Interface of Collection Interface

→ from the Collection Interface there are three Sub Interfaces. They are : 1) List 2) Set 3) Queue





Example for non-Generic

```
Arraylist al = new Arraylist();
    al.add(10);
    al.add("abc");
    al.add(10);
    al.add(6.5);
```

Example for Generic

```
Arraylist<Integer> al1 = new Arraylist<Integer>();
    al1.add(10);
    al1.add(20);
    al1.add(10);
    al1.add(50);
```

* List Interface

→ List Interface allows duplicates.

→ List is a ordered collection.

→ There are 4 classes in List. They are

1) ArrayList
2) LinkedList
3) Vector
4) Stack

→ Vector and stack are legacy classes.

→ Legacy classes are not useful in current JDK Versions.

1) Arraylist

- Arraylist implements List Interface.
- Arraylist internal datastructure is growable array or dynamic array.
- The initial capacity of ArrayList is 10.
- When initial capacity is increased, further the size will be increased by 50% of the current capacity.

Eg:-
```
public class ArrayListEg {
    public Static void main(String args[]) {
        Arraylist al = new ArrayList();
        al.add(10);
        al.add("abc");
        al.add(5.5);
        al.add("abc");
        al.add(30);
        System.out.println(al);
        System.out.println("index of abc::"+al.indexOf("abc"));
        System.out.println("last index of abc::"+
                al.lastIndexOf("abc"));
        al.remove(1);
        System.out.println("al after remove::"+al);
        al.add(1,90);
        System.out.println("al after adding 90::"+al);
    }
}
```

Generic Example

```
import java.util.Arraylist();
[public static void main (String args[])]
public class GenericDemo {
  public static void main (String args[]) {
  Arraylist <Integer> al = new Arraylist <Integer>();
  al.add(90);
  al.add(100);
  al.add(90);
  al.add(110);
  al.add(120);
  System.out.println(al);
  System.out.println("Contains 100 :: "+al.contains(100));
  System.out.println("Element at index 0 :: "+al.get(0));
  System.out.println("checking collection is Empty :: "+
                                al.isEmpty());
  al.clear(); //removes all the elements
  System.out.println("checking collection is Empty after
                        clear :: "+al.isEmpty());
  }
}
```

output :-

```
[90, 100, 90, 110, 120]
Contains 100 :: true
Element at index 0 :: 90
checking collection is Empty :: false
checking collection is Empty after clear :: true
```

Cursor

→ Cursors are used to display the Elements of the collection.

→ There are 3 cursors

Iterator
ListIterator
Enumeration

→ Iterator is used to display the Elements of the collection in forward direction.

→ Iterator ~~has~~ is called universal cursor. It will work on any collection.

hasNext() - checks the Elements in available in collection or not.

next() - methods returns the element

→ ListIterator is used to display both forward and backward direction.

ListIterator extends Iterator.

hasprevious() checks the previous element

previous() returns previous element

→ Enumeration interface works on legacy Collection

hasMoreElements() - checks the Element

nextElement() returns the Element.

* All Cursors are Interfaces.

* Collections work on user defined objects.

Eg :-

```
Class Student
{
  int Sid;
  String name;
  int age;
  Student (int Sid, String name, int age)
  {
    this.Sid = Sid;
    this.name = name;
    this.age = age;
  }
  public Static void main(String args[]) {
    Arraylist<Student> al = new Arraylist<Student>();
    Student S1 = new Student (1001, "abc", 22);
    Student S2 = new Student (1002, "xyz", 23);
    al.add (S1);
    al.add (S2);
    Iterator it = al.iterator ();
    while ( it.hasNext())
    {
      Student s = (Student) it.next();
      System.out.println(S.Sid+" "+S.name+" "+S.age);
    }
  }
}
```

Note: Methods of java can return an array, Collection, Object, Interface.

iterator() returns Iterator interface

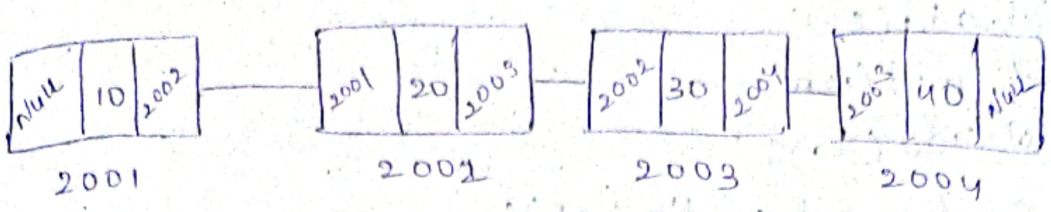
listIterator() returns ListIterator interface

LinkedList

→ LinkedList implements from list and Queue Interfaces.

→ LinkedList Internally uses double LinkedList as a datastructure.

Double LinkedList



→ node have three representations.

* previous node address
* value
* next node address.

Each node will be linked to next node with address.

(P) WAP to display of LinkedList in reverse order

```
import java.util.LinkedList;
public class LinkedListDemo {
  public static void main(String args[]) {
   LinkedList <String> l = new LinkedList<String>();
   l.add("php");
   l.add("oracle");
   l.add("java");
   l.add("php");
   ListIterator<String> lt = l.ListIterator(l.size());
   while(lt.hasprevious())
   {
    System.out.println(lt.previous());
   }
  }
}
```

(P) WAP to Convert an Arraylist to Array

```
public class ArraylistArray {
 public static void main(String args[]) {
  Arraylist<Integer> al = new Arraylist<Integer>();
  al.add(10);
  al.add(100);
  al.add(50);
  al-add(120);
  al.add(70);
  System.out.println(al);
  Collections.sort(al);
  Integer[] arr = al.toArray(new Integer[0]);
  System.out.printl"after Converting arraylist to Array::");
  for (int S: al)
   { System.out.print(S+" ");
   }
 }
}
```

```
Scanner S = new Scanner(System.in);
int a[] = new int(5);
for(int i=0; i<a.length; i++)
{
  a[i]=S.nextInt();
}
Arrays-Sort(a);
for (int i=0; i<a.length; i++)
{ System.out.print(a[i]+" ");
}
List<Integer> arr = new ArrayList<Integer>();
for(int S: a)
{
  arr.add(S);
}
System.out.println("After Conversion " + arr);
```

Vector

→ Vector implements from List Interface and it is a legacy class. Legacy classes are not useful in current jdk versions.

→ Vector internally uses growable array as a datastructure

Eg:-

```
import java.util.vector;
public class VectorExample {
  public static void main(String args[]) {
    Vector<Integer> V= new vector<Integer>();
    V.addElement (90);
    V.addElement (190);
    V.addElement (110);
    V.addElement (190);
    V.addElement (100);

    Enumeration<Integer> e = V.Elements();

    while (e.hasMoreElements())
    {
      System.out.println (e.nextElement());
    }
  }
}
```

→ The initial capacity of vector is 10

→ It will create a space for 10 Elements.

Stack

→ Stack Extends vector.

→ Stack is a legacy class.

→ Stack follows LIFO order (Last in first out)

→ The element which is added last to the Stack will be returned first from the stack.

→ Stack has three operation.

push() - adds the object in Stack.

pop() - deletes the Element (in LIFO order)

peek() - top most element from the stack

Eg:-

```
import java.util.Stack;
public class StackDemo{
public Static void main(String args[]){
Stack<Integer> S1 = new Stack<Integer>();
S1.push(10);
S1.push(20);
S1.push(30);
S1.push(40);
System.out.println("top most Element::"+S1.peek());
S1.pop();
System.out.println("Stack after pop::"+S1);
}
}
```

Output : [10, 20, 30, 40]

top most Element :: 40

Stack after pop :: [10, 20, 30]

* Set Interface

1/3/23

→ Set doesnot allow duplicate objects.

→ HashSet class implements from Set Interface.

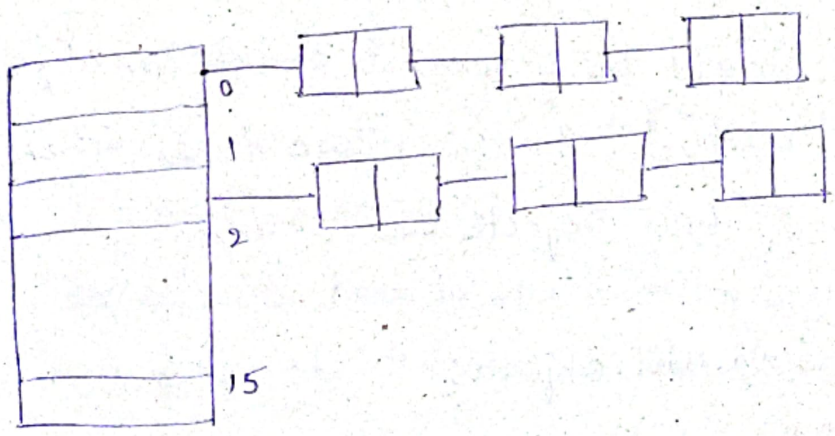
→ Hash Set internally uses HashTable as a DS.

→ The capacity of Hashset is unordered collection.

→ The capacity of Hashset is 16. with a load factor of 0.75.

→ when intially capacity increases further the memory will be extended for 12 new Elements.

16 * 0.75 = 12



→ The HashTable will be created from 0 to 15 buckets.

→ Each bucket will be Connected to a node.

→ Node will have two partitions data & address of next node.

→ Hashset allows Null values.

Overriding hashcode() and Equals() methods

→ When we store user defined objects in Set we have to overide hashcode() and Equals() methods.

→ hashcode() is used to get hashcode of the object.

→ Equals() is used to Compare two objects.

→ hashcode() and Equals() methods are available in object class.

→ Two objects are Equal their hashcode should be Equal.

→ hashcode Should be generated based on object data.

→ Equals() method should return true when hashcode of two objects are same

Eg:-
```
import java.util.Objects;
class Emp
{
  int eid;
  Stringname;
  String dept;
  Emp(int eid, String ename, String dept)
  {
    this.eid = eid;
    this.name = name;
    this.dept = dept;
  }

  // generate hashcode and Equals from Source
}
```

```
public class HashSetExample {
public static void main (String args[]) {
HashSet <Emp> h = new HashSet<Emp>();
Emp e1 = new Emp(1001, "abc", "hr");
Emp e2 = new Emp (1001, "abc", "hr");
h.add(e1);
h.add(e2);
for(Emp e:h)
System.out.println(e.eid+" "+e.ename+" "
                                    +e.dept);
}
}
```

→ In the above program Emp e1 = new (1001, "abc", "hr");
Emp e2 = new (1001, "abc", "hr");

→ e1 and e2 objects are same.
Because we override hashcode() and Equals() ,

→ HashSet will allow one e1 only in the collection.

→ getclass() return runtime class of the object.

LinkedHashSet

→ LinkedHashSet Extends HashSet.

→ It follows Insertion order

→ The data Structure used by LinkedHashset is hashtable with double Linked List.

Eg:-
```
import java.util.LinkedHashSet;
public class LinkedHashSetEg {
public static void main (String args[]) {
```

```
LinkedHashSet<Integer> lh1 = new LinkedHashSet<Integer>();
lh1.add(10);
lh1.add(30);
lh1.add(40);
lh1.add(90);
System.out.println("lh1::"+lh1);

LinkedHashSet<Integer> lh2 = new LinkedHashSet<Integer>();
lh2.add(90);
lh2.add(110);
lh2.add(150);
lh2.add(170);
System.out.println("lh2::"+lh2);

lh1.addAll(lh2);
System.out.println("After addAll lh1::"+lh1);
}
}
```

Output :

```
lh1 : [10, 30, 40, 90]
lh2 : [90, 110, 150, 170]
After addAll lh1 :: [10, 30, 40, 90, 110, 150, 170]
```

Tree Set

→ Tree Set implements NavigableSet.

→ NavigableSet Extends SortedSet.

→ Tree Set follows ascending order. It doesn't allow duplicates and null values.

→ TreeSet follows Balanced binary tree as a data structure.

→ We can store only Comparable objects in treeset.

Eg:-

```
import java.util.TreeSet;
public class TreeSetEg {
public static void main (String args[]) {
  TreeSet<String> t = new TreeSet<String>();
  t.add("java");
  t.add("sql");
  t.add("html");
  t.add("php");
  System.out.println("TreeSet::"+t);
  System.out.println("first element to remove::"+
                        t.pollFirst());
  System.out.println("last Element to remove::"+
                        t.pollLast());
  System.out.println("first Element in the Set::"+
                        t.first());
  System.out.println("last Element in the Set::"+
                        t.last());
  }
}
```

output:- [html, java, php, SqL]

first Element to remove :: html

Last Element to remove :: SQL

first Element in the Set :: java

Last Element in the set :: php.

→pollFirst() retrives and removes the first Element

→pollLast() retrives and removes the Last Element

→ first() returns the first Element

→Last() returns the Last Element.

2/03/23

Queue Interface

→ Queue Interface follows FIFO order (First in first out).

→ LinkedList class implements from Queue and List interfaces.

→ Queue is an ordered Collection.

Eg:-
```
import java.util.Queue;
public class LinkedListAsQueue {
public Static void main(String args[]) {
  Queue<Integer> q = new ~~Queue~~ LinkedList<Integer>();
  q.add(10);
  q.add(20);
  q.add(90);
  q.add(100);
  q.add(30);
  System.out.println("100 is available :: "+q.contains(100));
  System.out.println("size of Queue :: "+q.size());
  System.out.println("Queue is Empty :: "+q.isEmpty());
  System.out.println("head of the queue to remove :: "+
                                                q.poll());
}
}
```

Output:-

100 is available :: true

Size of Queue :: 5

Queue is Empty :: false

head of the queue to remove :: 10

→ Insertion will be done from the tail of the Queue.
→ deletion will be done from the head of the Queue.

priority Queue

```
import java.util.priorityQueue;
public class priorityQueueEg {
  public static void main (String args[]){
    priorityQueue<Integer> p = new priorityQueue<Integer>();
    p.add(100);
    p.add(110);
    p.add(90);
    p.add(120);
    p.add(100);
    System.out.println(p);
      Object o[] = p.toArray();
      for(Object i: o)
        System.out.println(i);
    }
  }
```

output :-

```
[90, 100, 100, 120, 110]
 90
 100
 100
 120
 110
```

→ priorityQueue with the default initialcapacity (11)
→ toArray() Converts Queue to an Array
→ The return type of toArray() is Object[].

PriorityBlockingQueue

→ PriorityBlockingQueue implements from BlockingQueue Interface.

→ PriorityBlockingQueue with the default initial capacity(11) that orders its Elements according to their natural Ordering.

MapInterface

→ MapInterface Stores Entries

→ A key and value pair together is called an Entry.

→ Keys should not be duplicate but value can be duplicate

→ One key can map to one value atmost.

→ When we use Same key, to map a new value it Overrides previous value

Eg:-

SID (Key)	name (value)
1001	abc → Entry
1002	xyz
1003	pqr
1001	A

→ In the above Eg. 1001 key maps with A

* HashMap class implements from Map Interface.

→ HashMap uses Hashtable as a datastructure.

→ The initial capacity of HashMap is 16 with the lower factor of 0.75.

→ When initial capacity increases, further memory will be Extended for 12 new Elements.

Hashmap m = Hashmap();

→ It is a non-Generic collection and unordered

→ we can store key and value of any type.

```
Hashmap<Integer, String> m1 = new Hashmap<Integer, String>();
    m1.put(1001, "abc");
    m1.put(1002, "xyz");
    m1.put(1001, "pqr");
```

→ m1 is a generic collection. In this collection key is integer type and value is String type.

→ There are three main Methods of Map (I)

Set KeySet() - returns all the keys

Collection values() - returns all the values

Set EntrySet() - returns keys & values (Entries).

Eg:-
```
import java.util.HashMap;
public class HashmapEg {
  public Static void main(String args[]) {
    HashMap<Integer, String> h = new <Integer, String>();
      h.put(1001, "Abc");
      h.put(1002, "xyz");
      h.put(1003, "pqr");
      h.put(1001, "A");

      System.out.println("Keys::" + h.keys());
      System.out.println("values::" + h.values());
      System.out.println("Entries::" + h.EntrySet());
  }
}
```

3/03/23

Recursion

Recursion is a process of calling the same method itself until it comes to a particular condition.

* Factorial using recursion

```
public class RecursionDemo {
 int fact (int n)
 {
   if(n==1)
     return 1;
   Else
     return n* fact(n-1);
 }
 public static void main (String args[]) {
  RecursionDemo r = new RecursionDemo();
  System.out.println(r.fact(5));
 }
}
```

1 return
2 * fact(1)
3 * fact(2)
4 * fact(3)
5 * fact(5-1)

Stack

* Fibonacci Series using recursion

```
Public class fibonacciRecursion {
 int fib(int n)
 {
   if(n==0)
     return n;
   if (n==1)
     return n;
   Else
     return fib(n-1)+fib(n-2);
 }
 Public Static void main (String args[]) {
  fibonacciSeries f = new fibonacciSeries();
  for (int i=0; i<=10; i++)
   System.out.println(f.fib(i));
} }
```

* Hashmap allows, one Null key & multiple Null values.

LinkedHashMap

→ LinkedHashMap Extends HashMap

→ LinkedHashMap uses Hashtable with double LinkedList.

→ LinkedHashMap allows one null key and multiple null values.

→ It follows insertion order based on keys.

Eg:-
```
import java.util.LinkedHashMap;
public class LinkedHashMapEg {
 public static void main (String args[]) {
  LinkedHashMap<Integer, String> l = new
                    LinkedHashMap<Integer, String>();
  l.put (1001, "abc");
  l.put (1002, "xyz");
  l.put (1003, "pqr");
  l.put (1001, "A");
  Set s = l.EntrySet();
  Iterator it = s.iterator();
  while (it.hasNext())
  {
   Map.Entry<Integer, String> m = (Entry<Integer,
                    String>) it.next();
   System.out.println (m.getKey()+" "+m.getvalue());
  }
 }
}
```

→ Map.Entry is a nested interface of Map Interface
→ This Interface has getKey() which ~~w~~ returns all the keys, getvalue() which returns the values.

IdentityHashMap

→ IdentityHashMap Implements from MapInterface
→ This class doesnot follow Map Contract.
→ When we Store user defined objects in the map, it uses Hashcode() and Equals().
→ The Hashcode() ~~of a key~~ will return Hashcode of a key.
→ The Equals method compares the Entries and it return false when keys are different.
→ IdentityHashMap uses == operator for comparision of the objects.

WeakHashMap

→ WeakHashMap is used to store WeakEntries.
→ The Entries which are no longer used are called weak Entries.
→ We can Store weakEntries in a weakHashMap, these Entries can garbage Collected to remove.
→ We can also call System.gc() to force garbage Collector to Execute.
→ When we call System.gc(), The garbage Collector may not Execute immediatly, If it occupies

with a process, whenever it will get free it will execute.

TreeMap

- TreeMap implements from NavigableMap.
- NavigableMap Extends SortedMap.
- TreeMap follows ascending order based on keys.
- TreeMap doesn't allow null Entries.
- When we Store null Key or null Value it will throw NullpointerExcyption.
- TreeMap uses Balanced binary tree as a data Structure.
- we have to Store Comparable Objects in TreeMap.

* How HashSet will work internally?
* How HashMap will work internally?

Eg:-

```
import java.util.TreeMap;
public Class TreeMapEg {
public Static void main(String args[]) {
TreeMap<Integer, String> t = new TreeMap<Integer, String>();
t.put(101, "java");
t.put(102, "php");
t.put(103, "sql");
t.put(104, ".net");
System.out.println(t);
for (Map.Entry<Integer, String> m : t.entrySet())
{
System.out.println(m.getKey() + " " + m.getvalue());
}
}
}
```

Hashtable

→ Hashtable is a legacy class

→ It implements Hashtable as data Structure.

→ Null key & Null values are not allowed.

→ Hashtable Extends from Dictionary & implements from MapInterface

→ Initial default capacity is 11 with the load factor of 0.75.

→ Hashtable is Synchronized.

In Synchronization only one thread can access the resource at a time.

→ Synchronization is the process of controlling multiple threads to access a resource at a time

Difference between Hashmap and Hashtable

In Java, both Hashmap and Hashtable are used to store key-value pairs, but there are some differences:

→ HashMap is not Synchronized and it is not thread-safe.
→ Hashtable is Synchronized which means it is thread-safe and can be accessed by multiple threads Concurrently without any problem.
→ Hashmap allows one null key and any number of null values.
→ Hashtable does not allow null keys or values.
→ Hashmap have better performance than Hashtable because its non Synchronized nature.
→ Hashtable may perform better due to its Synchronized nature in situations where thread-safty is imp.
→ The Iterator for Hashmap is fail-fast which means that it throws a ConcurrentModificatioException if the collection is modified during iteration.
→ The Iterator for Hashtable is not fail-safe which means that if the collection is modified during iteration, it may result in unexpected behaviour.
→ HashMap does not maintain the insertion order of elements
→ Hashtable maintain the insertion order of elements.

* Why is the map not from collection in java.

→ The map interface is not a subtype of the collection interface because maps and collections are fundamentally different types of data structures.

→ collections are a group of objects, usually of the same type, that can be accessed in a particular order.

→ The most common operations on collections include adding and removing elements, checking if an element is present, and iterating over the collection.

→ map is a collection of key-value pairs, while each key is associated with a value.

→ The common operations on maps include adding and removing key-value pairs, looking up a value based on its key, and iterating over the keys or values.

→ Maps and collection serve different purpose, they have different interface in Java.

→ Collection interface is designed for collections that are accessed by position or iteration.

→ the map interface is designed for mappings that are accessed by key.

* Why we should override hashcode() and Equals()

→ Hashcode() and Equals() methods are used for object equality and to ensure that objects are correctly stored and retrived from collections like HashSet, Hashmap, and ~~Ind~~ Hashtable.

→ The Equals() method is used to compare two objects from content equality.

→ The Equals() Compares object references, which means that it returns 'true' only if the two objects being compared are the Exact Same object in memory. therefore, we need to override the Equals() to provide custom comparison logic.

→ Hashcodes are used by Collections to quickly identify the location where an object should be stored or retrieved.

→ when two objects are Equal they must have the Same hash code

Therefore, if we override Equals(), we must also override hashcode() to Ensure that two Equal objects have the Same hash code.

Difference between Arraylist and Linked List

→ Arraylist is implemented using an array.

→ Linkedlist is implemented using a doubly linkedlist.

→ Arraylist is faster random access times because Elements can be accessed using an index.

→ Linkedlis ~~can~~ ~~inso~~ has Slower random access times because elements must be traversed Sequentially to find the desired element.

→ Arraylist is slower than linkedlist when it comes to inserting or deleting elements from the middle of the list.

→ Linkedlist can insert and delete Elements in Constant time by adjusting the links between nodes.

→ Arraylist is not thread-safe by default, but it can be synchronized using External Synchronization.

→ LinkedList is also not thread-safe by default, but it can be made thread-safe using External Synchronization.

* Difference between List and set

→ List allows duplicate Elements.

→ Set does not allows duplicate elements.

→ In List, we can add same Elements multiple times.

→ In Set, adding an Element that already Exists has no Effect.

→ List Implementations, Such as Array List and LinkedList, maintain the order of Elements in which they are added to the collection.

→ Set implementations, such as Hashset and Treeset, do not maintain the insertion order of elements.

→ In List, Equals() is used to determine if two lists have same elements in the same order.

→ In Set, Equals() is used to determine if two objects are the same.

→ List is used when you need a collection of elements that allows duplicates and maintains their order.

→ Set is used when you need a collection of elements that doesnot allow duplicates and does not maintain their order.

* Difference between iterator and listiterator in

→ Iterator can only iterate in the forward direction, from the beginning to the End of the collection.

→ ListIterator allows you to iterate in both forward and backward directions, and also Enables you to modify elements in the list during iteration.

→ Iterator does not maintain the current position and only allows you to remove Elements from the Collection during iteration.

→ ListIteration maintains the current position in the List, and allows you to add, set or remove Elements at the current position.

→ Iterator is implemented by all Collections that implement the Iterable interface, Such as Arraylist, Linkedlist and Hashset.

→ ListIterator is only implemented by List Collections Such as Arraylist and Linkedlist.

→ Iterator is a basic interface that can be used to iterate over any collection that implements the Iterable Interface

→ ListIterator is a more advanced interface that can be used to iterate over and modify List Collections in both forward and backward direction.

→ Iterator ~~is a~~ more has additional methods that are not present in the Iterator interface like previous(), hasprevious(), add(), Set(), and remove().

* Difference between Collection and Collections

→ Collection is an interface in the Java Collection Framework that represents a group of objects known as Elements.

→ It is a root interface that has Several Subinterfaces Such as list, set and Queue.

→ Collection Interface provides Several methods of adding, removing and querying Elements in the Collection.

→ Collection is an interface that defines the behaviour of a group of objects.

Collections.

→ Collections is a utility class in the Java Collections frame work that contains static methods for operating on objects of type Collections.

→ It provides Several methods for Sorting, Searching, and manipulating Collection objects.

→ Some of the methods provided by the Collections class include sort(), reverse(), Shuffle(), binarySearch() and max().

→ Collections is a utility class that provides methods for working with Collection objects.

* Difference between array and collection

→ An array has a fixed size that is determined at the time of creation.

→ collection can dynamically grow or shrink in size as elements are added or removed.

→ Array in java are type-safe, meaning that they can only store elements of the same data type.

→ collection can store elements of different data types, but this can lead to type safety issues.

→ Accessing elements in an array is faster than accessing elements in a collection because arrays are implemented as a contiguous block of memory.

→ collection may require extra operations to traverse the elements.

→ collections offer many more methods and functionalities than arrays, such as sorting, searching and filtering.

→ Arrays are generally more memory-efficient than collections. because,

They do not require the additional overhead associated with dynamic resizing and other features offered by collections.

* Difference between Array and Arraylist.

→ Array has a fixed size that is determined at the time of creation.

→ Arraylist can dynamically grow or shrink in size as elements are added or removed.

→ Arrays are type-safe, meaning that they can only store elements of the same data type.

→ Arraylist can store elements of different data types, but this can lead to type safety issues.

→ Array does not allow generics, though multidimensional in nature.

→ Arraylist enables the use of generics and single dimensional in nature

→ Functions such as indexof() and remove() are not supported by Arrays in Java.

→ operations such as indexof(), remove() are supported by Arraylist in java

→ Array members can be accessed by using [];

→ Arraylist can access elements using a set of methods and modify them.

→ Array is a fixed sized data structure

→ Arraylist is not a fixed sized data structure.

* difference between Comparable and Comparator

→ Comparable provides a single sorting sequence and we can sort the collection on the basis of a single element such as id, name and price.

→ Comparator provides a ~~d~~ multiple sorting sequences and we can sort the collection on the basis of multiple elements such as id, name and price etc.

→ Comparable affects the original class i.e; the actual class is modified.

→ Comparator doesn't affect the original class i.e; the actual class is not modified.

→ Comparable provides compareTo() to sort elements.

→ Comparator provides compare() to sort elements.

→ Comparable is present in java.lang package.

→ Comparator is present in java.util package.

→ we can sort the list elements of comparable type by collections.sort(List) method.

→ we can sort the list elements of comparator type by collections.sort(list, comparator) method.

→ Comparable is implemented by the class itself.

→ Comparator is implemented by a seperate class that is passed to the sorting method.

* Difference between final, finally and finalize()

→ final keyword is used with the classes, methods and variables as "unchangable".

→ once declared, final variable becomes constant and cannot be modified.

→ final method is executed only when we call it.

*→ finally block is always related to the try and catch block in exception handling.

→ finally block is executed as soon as the try-catch block is executed.

→ it's execution is not dependant on the exception.

*→ finalize() method is used with the objects.

→ finalize() is used to perform clean up processing just before object is garbage collected.

→ finalize method is executed just before the object is destroyed.

* Difference between java.util.Date and java.sql.Date

→ java.util.Date represents a point in time with millisecond precision.

→ java.sql.Date, represents a data without time zone information and is designed for use with database

→ It is not tied to any Specific time zone or database managment System.

→ It is designed for use with databaes and is used to store or retrive datas in the SQL DATA format.

→ This class provides basic functionalities such as getting and Setting the data, time and milliseconds.

→ This class provides additional methods for getting and Setting the year, month and day seperately.

Properties class

6/03/23

→ properties class is used to read data from properties file.

→ properties file storage stores the data as a key & value pair.

Eg:- Student.properties

```
username = admin
password = admin.
```

propertiesDemo

```
Public class propertiesDemo {
Public Static void main(String args[]) throws IOException {
FileInputStream fis = new FileInputStream("path");
Properties P = new properties();
P.load(fis);

Scanner S = new Scanner(System.in);
System.out.println("Enter username");
String username = S.next();
System.out.println("Enter password");
String password = S.next();

String uname = P.getProperty("username");
String pwd = P.getproperty("password");
if (uname.Equals(username) && pwd.Equals(password))
{
System.out.println("Login Success");
}
else
{
System.out.println("Login again");
}
}
}
```

→ The load() of properties class takes Input Stream object as a parameter.

→ getproperty() method takes property name as a parameter and returns String.

Collections Class

→ Collection is an Interface where as Collections is a class.

→ Collections class has Static method to work on Collection.

Eg:-
```
public class CollectionsDemo {
public Static void main (String args[]) {
Arraylist <Integer> al = new Arraylist <Integer>();
al.add ("100");
al.add("200");
al.add("80");
al.add("100);
al.add (70);
System.out.println (al);
Collections.Sort (al);
System.out.println ("collections after Sorting ::" + al);
Collections.Sort (al, Collections.reverseorder());
System.out.println ("Collection in descending order ::" + al);
System.out.println ("Max Element ::" + Collections.max(al));
System.out.println("min Element ::" + Collections.min(al));
Collections.shuffle (al, new Random());
System.out.println ("Collection Shuffled in random order ::" + al);
}
}
```

→ Collections. Shuffle() is used to shuffle Collection.

Dictionary class

The Dictionary class is the abstract parent of Hashtable, which maps keys to values. Every key and every value is an object. In any one Dictionary object, every key is associated with at most one value.

Example on Comparator Interface

```
class Emp1
{
 int eid;
 String name;
 double sal;
 Emp1 (int eid, String name, double sal)
 {
    this.eid = eid;
    this.name = name;
    this.sal = sal;
 }
}
public class ComparatorDemo implements Comparator
                                        <Emp1> {
 public int compare (Emp1 o1, Emp1 o2) {
   if (o1.eid == o2.eid)
     return 0;
   else if (o1.eid < o2.eid)
     return -1;
   else
     return 1;
 }
```

```
public Static void main (String args[])
{
ArrayList<Emp1> al = new ArrayList <Emp1>();
al.add (new Emp1 (1004, "abc", 19000));
al.add (new Emp1 (1002, "xyz", 21000));
al.add (new Emp1 (1003, "A", 22000));

Collections.Sort (al, new ComparatorDemo());
for (Emp1 e : al)
    System.out.println (e.eid + " " + e.name + " " + e.Sal);
  }
}
```

→ Comparator Interface provides ordering of collection objects.

→ Comparator can be passed to a Sort() (Collections.Sort)

→ This interface has Compare method, It take two objects as parameters.

→ If two objects are Equal it returns 0, One object value is less than other object value it returns -ve value and one object value is greater than other object value it returns +ve value

7/03/23

Comparable Interface

→ Comparable Interface is used to sort user defined Objects.

→ This Interface has CompareTo() method.

Eg:-
```
class Student1 implements Comparable<Student1>
{
    int Sid;
    String name;
    int age;
    Student1 (int Sid, String name, int age)
    {
        this.Sid = Sid;
        this.name = name;
        this.age = age;
    }
    @Override
    Public int CompareTo (Student1 o)
    {
        if (this.Sid == o.Sid)
            return 0;
        Else if (this.Sid > o.Sid)
            return 1;
        Else
            return -1;
    }
}
Public class ComparableDemo {
public Static void main (String args[]) {
ArrayList<Student1> al = new ArrayList<Student1>();
Student1 S1 = new Student1 (1001, "abc", 22);
Student1 S2 = new Student1 (1002, "xyz", 21);
Student1 S3 = new Student1 (1000, "pqr", 20);
```

```
al.add(S1);
al.add(S2);
al.add(S3);

Collections.Sort(al);

for(Student1 S:al)

  System.out.println(S.sid+" "+S.name+" "S.age);
 }
}
```

output: 1000 Pqr 20
1001 abc 22
1002 xyz 21

Multithreading

→ Multi means more than one, thread means a light weight process.

→ In multithreading more than one thread can Execute Simultaneously.

→ The advantage of multithreading is we can utilize cpu ideal time

→ Multithreading is used in game based applications, the Servers internally uses multithreading model.

→ A thread Executes in the memory Seperately.

→ A thread can also be called as ligth weight process.

→ Even multiple threads are created in an application, the cpu allocates a processing time for a thread to Execute.

→ A thread shows Seperate path of Execution.

→ In multithreading one thread process can be independent of another thread.

→ The thread Scheduler Executes a thread.

→ Thread Scheduler can follow any of the two algorithms.

i) pre Emptive Scheduling
ii) time Slicing.

→ In pre Emptive Scheduling, the thread with the highest priority will be Executed first.

→ In time Slicing, Each thread will be given with Slice of time to Execute.

There are two ways to create a thread.

i) implements from Runnable Interface
ii) Extends from Thread class.

→ Runnable Interface and Thread class are available in java.lang package

→ Runnable Interface have a Single method called Run method.

→ When a class implements from Runnable Interface we have to override run method.

```
class MyRunnable implements Runnable
{
    public void run()
    {
    }
}
```

→ Thread class implements from Runnable Interface.

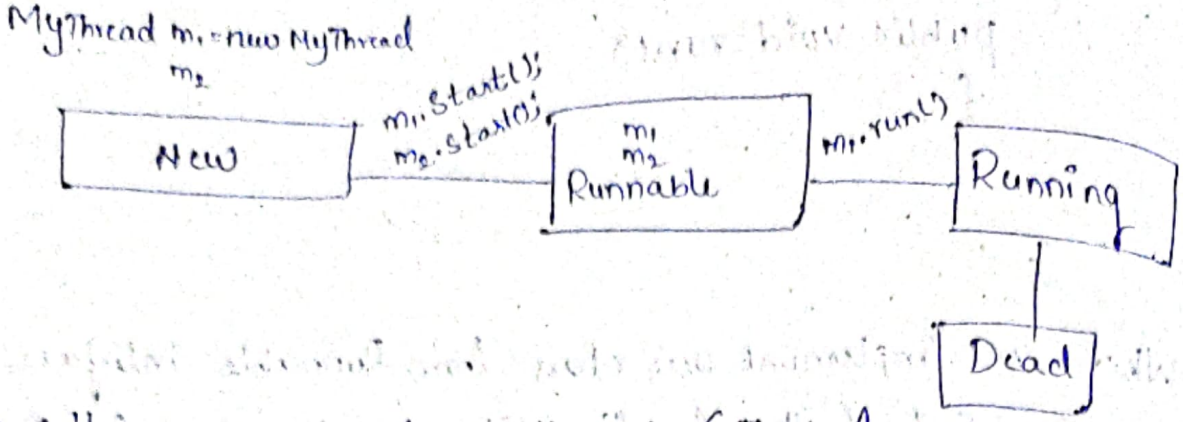
```
class MyThread Extends Thread
{
    public void run()
    {
    }
}
```

→ When we implement our class from Runnable interface, we can Extend the functionality of the class further.

→ Whe we Extend our class from thread class, it is not possible to Extend the functionality further.

Eg:-
```
public class MyThread Extends Thread {
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            System.out.println("ThreadName :: " + Thread.
                CurrentThread().getName());
            System.out.println(i);
        }
    }
    public static void main (String args[])
    {
        MyThread m1 = new MyThread();
        m1.start();
        MyThread m2 = new MyThread();
        m2.start();
    }
}
```

Threal life cycle



→ When we create a thread (Thread t = new Thread()) thread will be New State

→ When we call start() the thread will be moved to Runnable state.

→ In Runnable state multiple threads will be there, when thread Schedulas calls run() the thread will be ~~re~~ moved to running state.

→ In running state the thread will execute.

→ Once all the life cycle stages of the thread is Completed, thread will be ~~removed~~ moved to dead state.

Thread class methods

Start() is used to Start the thread.

run() is used to run the thread.

CurrentThread() is used to return the reference of the cussent thread.

SetName() is used to give name for the thread.

getName() is used to get the name

Setpriority() is used to give priority for the thread.

getpriority() is used to get the priority for the thread

Thread priority

→ Thread priority are from 1 to 10.

1 is the minimum priority
10 is the maximum priority
5 is the normal priority

Thread.MAX_PRIORITY → 10

Thread.MIN_PRIORITY → 1

Thread.NORM_PRIORITY → 5

The thread with the highest priority will be Executed first.

→ Runnable Interface has a single Method called Run().

Eg:-
```
public class MyRunnable implements Runnable {
    public void run() {
        for (int i=1; i<=5; i++)
        {
            System.out.println(" Thread Name :: " +
                Thread.CurrentThread().getName());
            System.out.println("Thread priority :: " +
                Thread.CurrentThread().getpriority());
            System.out.println("hi");
        }
    }
    public static void main(String args[]) {
        MyRunnable m1 = new MyRunnable();
        Thread t1 = new Thread(m1);
        t1.SetName("thread-1");
        t1.Setpriority(10);
        t1.start();
```

```
MyRunnable m2 = new MyRunnable();
Thread t2 = new Thread(m2);
t2.SetName("thread-2");
t2.Setpriority(7);
t2.Start();
}
}
```

Sleep()

- Sleep() is a static method of thread class.
- Sleep() is used to pause thread Execution.

Thread.sleep(long milliseconds)
Thread.sleep(long milliseconds, int nanoseconds)

→ When we use sleep() we have to handle InterruptedException.

→ When sleep() is called on a thread, thread will be moved to not runnable state.

→ In the sleep time of the thread, Scheduler will give chance for other thread to Execute.

→ once the sleep time of the thread is completed, thread will be moved to runnable state.

→ from runnable state, Scheduler again will Schedule the thread to Execute.

```
public class SleepDemo Extends Thread {
public void run() {
  for(int i=1; i<=5; i++)
  {
  System.out.println ("Thread Name ::"+
        Thread.currentThread().getName());
  System.out.println("Thread priority ::"+
        Thread.currentThread().getpriority());
  System.out.println(i);
  try
  {
   Thread.Sleep(3000);
  }
  catch (InterruptedExeption e)
  {
    e.printStackTrace();
  }
}
public static void main (String args[]).
{
  SleepDemo S1 = new SleepDemo();
  S1.SetName ("thread-1");
  S1.Setpriority(10);
  S1.Start();

  SleepDemo S2 = new SleepDemo();
  S2.SetName ("thread-2");
  S2.Setpriority(5);
  S2.Start();
 }
}
```

10/03/23

Thread Synchronization

→ Synchronization is the process of controlling Multiple threads to access a resource at a time.

→ Synchronization is used to avoid data inconsistancy problems

→ When there is a limited data is available we can use thread Synchronization to avoid data inconsistancy problems.

→ Even Java Supports multithreading, Synchronization can be used in certain cases.

Eg:- For a particular train if there is only one ticket is available, if we allow multiple people to book the ticket, one ticket will be alloted to multiple people.

If we allow only one person we can avoid data inconsistancy problems.

→ The Synchronized keyword can be used for a method and a block.

→ Synchronized method has more scope compare to Synchronized block.

Eg:-
```
public synchronized void display()  // Method
{

}

synchronized(this)   // Block
{

}
```

```
public class ThreadSynchronizationDemo Extends Thread {
String str[] = {"this", "is", "a", "synchronized", "demo"};
public void run()
{
String name = Thread.currentThread().getName();
display (name);
}
synchronized void display (String name)
{
System.out.println (" Thread Name ::" +
                    Thread.CurrentThread().getName());
for (String s : str)
{
System.out.println (s);
}
}
public Static void main (String args[]) {
ThreadSynchronizationDemo t1 = new ThreadSynchronizationDemo
                                                    ();
t1.SetName ("thread-1");
t1.Start();

ThreadSynchronizationDemo t2 = new ThreadSynchronizationDemo();
t2.SetName ("thread -2");
t2.Start();
}
}
```

Synchronized Block

```
Public class SynchronizedBlockDemo Extends Thread
{
  public void run()
  {
    int a[] = {1,2,3,4,5};
    Synchronized (this)
    {
      for (int i : a)
        System.out.println(i);
    }
  }
  public Static void main(String args[]) {
    SynchronizedBlockDemo S1 = new SynchronizedBlockDemo();
    S1.SetName ("thread-1");
    S1.setpriority (Thread.MAX_PRIORITY);
    S1.Start();

    SynchronizedBlockDemo S2 = new SynchronizedBlockDemo();
    S2.SetName ("thread-2");
    S2.setpriority (4);
    S2.Start();
  }
}
```

Dead Lock

→ Deadlock is the process of Continuous waiting State of the threads.

→ In a application, multiple threads are Continuously waiting to aquire a lock on a resource, it can lead to deadlock Situation.

→ Dead lock is not an Exception or Error, it occurs only waiting State for the threads.

→ We can avoid dead lock Situation by giving a particular time to Execute a thread.

→ A proper way of obtaining the locks can avoid dead lock Situation.

→ Dead lock occurs only in thread Synchronization.

I/O Streams (IO)

13/03/23

→ Stream is a flow of data from one place to another place.

→ IO refers to Input output.

→ We can Send the data to the file (using outputStream)

→ We can open a file and read the data from the file using (Input Stream)

→ There are two types of Streams.

Byte Stream

Character Stream.

→ Byte Streams can Send only byte and can read only one byte.

→ Byte Streams can work with any type of data.

→ Character Stream can send one character at a time and read one character at a time.

→ Character streams can work on text or String data only.

→ ByteStream classes Ends with Stream.

→ CharacterStreams Ends with reader or writer as the word.

→ FileOutput Stream class opens the file and writes the data to the file.

→ FileInputStream class opens the file and reads the data from the file.

Eg:-
```
public class FileWriteDemo {
    public static void main (String args[]) {
    FileOutputStream fout = null;
    try
    {
     fout = new FileOutputStream ("myfile.txt", true);
     String str = "Hello";
     byte b[] = str. ~~str~~ getBytes ();
     fout .write (b);
    }
    Catch (IOException e)
    {
      e.printStackTrace ();
    }
    finally
    {
```

```
        try
        {
          fout.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
  }
}
```

Eg:-

```
public class FileReadDemo {
public static void main (String args[]) {
FileInputStream fis = null;

try
{
 fis = new FileInputStream("myfile.txt");
 int i;
 while((i = fis.read()) != -1)
     System.out.print((char) i);
}
catch (IOException e)
{
  e.printStackTrace();
}
finally
{
  try
  {
    fis.close();
  }
  catch (IOException e)
  {
    e.printStackTrace();
  }
}
}
}
```

FileWriter

FileWriter is a character stream class.

This class opens the file writes the data

FileReader is a character stream class.

This class opens the file and reads the data from the file.

Eg:-

```
public class FileWriterDemo {
    public Static void main(String args[]) {
        FileWriter fw=null;
        try
        {
            fw=new FileWriter("abc.txt",true);
            fw.write("java is a programming languge");
        }
        Catch (IOException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                fw.close();
            }
            Catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

Eg:-

```
public class FileReaderDemo {
public Static void main (String args[]) {
  FileReader fr = null;

  try
  {
    fr = new FileReader ("myfile abc.txt", true);
    ~~fw = write ("java is a prg)~~
    int i;
    while ((i = fr.read ()) != -1)
        System.out.print ((char) i);
  }
  Catch (IoException e)
  {
    e.printStackTrace ();
  }
  finally
  {
    try
    {
      fw.close ();
    }
    catch (IOexception e)
    {
      e.printStackTrace ();
    }
  }
}
}
```

14/03/23

BufferedReader

* The BufferedReader class is used to take input from the user.
* The readline method takes the input in String format.
* The BufferedStreams are used to improve Efficiency of IO operations.
* Bufferedstreams internally maintains a buffer (temporary space to store the data).

Eg:-

```
public class BufferedDemo {
    public Static void main (String args[]) throws
                NumberFormatException, IOException {
    BufferedReader br = new BufferedReader
                (new InputStreamReader (System.in));
    System.out.println ("Enter Sid");
    int Sid = Integer.parseInt (br.readLine());
    System.out.println ("Enter the name");
    String name = br.readLine();
    System.out.println("Enter the marks");
    float marks = Float.parseFloat(br.readLine());
    System.out.println ("Sid:: "+ Sid + "\n" + "name="+
        name + "\n"+ "marks=" + marks );
    }
}
```

output :- Enter Sid
1001
Enter the name
abc
Enter the marks
50.5
Sid :: 1001
name :: abc
marks :: 50.5

Serialization

→ Serialization is the process of Saving an object as a ByteStream.

→ To perform Serialization the class Should implement from Serializable interface.

→ Serializable Interface is in java.Io package.

→ Serializable Interface is a marker Interface

→ marker Interfaces are Empty Interfaces. In marker Interfaces there are no methods and fields.

→ The other marker Interfaces in java are cloneable and Remote.

* If the marker Interfaces doesn't have fields and methods, why the class should Implement?

→ When a class implements from marker interface, the functionality will be added to the class.

Example for Serialization

```
public class Employee implements Serializable
{
    int Empid;
    String name;
    double Sal;
    Employee (int empid, String name, double Sal)
    {
        this.Empid = empid;
        this.name = name;
        this.Sal = Sal;
    }
    public static void main (String args[]) throws
                                IOException {
    FileOutputStream fout = new FileOutputStream("Emp.ser
    ObjectOutputStream oop = new ObjectOutputStream
                                (fout);
    Employee e = new Employee (1001, "abc", 12000);
    oop.writeObject(e);
    }
}
```

→ In Serialization process we use Serializable file (.ser)

→ ObjectOutputStream class writeObject() is used to store object in the .ser file.

DeSerialization:

→ In Deserialization process we retrive an object from Serialization.

→ It is the reverse process of Serialization.

→ ObjectInputStream class readObject() reads object from Serializable file.

→ The returntype of this method is object type.

Example of Deserialization

```
public class Deserialization {
 public static void main(String args[]) throws
        IOException , classNotFoundException {

  FileInputStream fis = new FileInputStream ("Emp.ser");
  ObjectInputStream ois = new ObjectInputStream (fis);

  Employee e = (Employee) ois.readObject();
  System.out.println (e.Empid + " "+ e.name + " "+ e.sal);
 }
}
```