**JAVA FSD Phase 1**

**Day 1 (24 Oct 2022)**

**Git Steps**

1. git init inside of your local folder (Java Samples)

2) git status (it must be clean)

3) Create a new file inside of your folder by right clicking in the folder and name it as "hello.txt"

4) Check git status again you will see your file is in staging area

5) git add .

6) git status

1. git commit -m "Hello new file created" OR

sudo git commit -m "Hello new file created"

8) git status

9) git branch -M main

10) Create a new personal token in your github.com. In Settings --> Developer tools --> Generate New Token (classic) --> Copy the token and paste it in the below URL

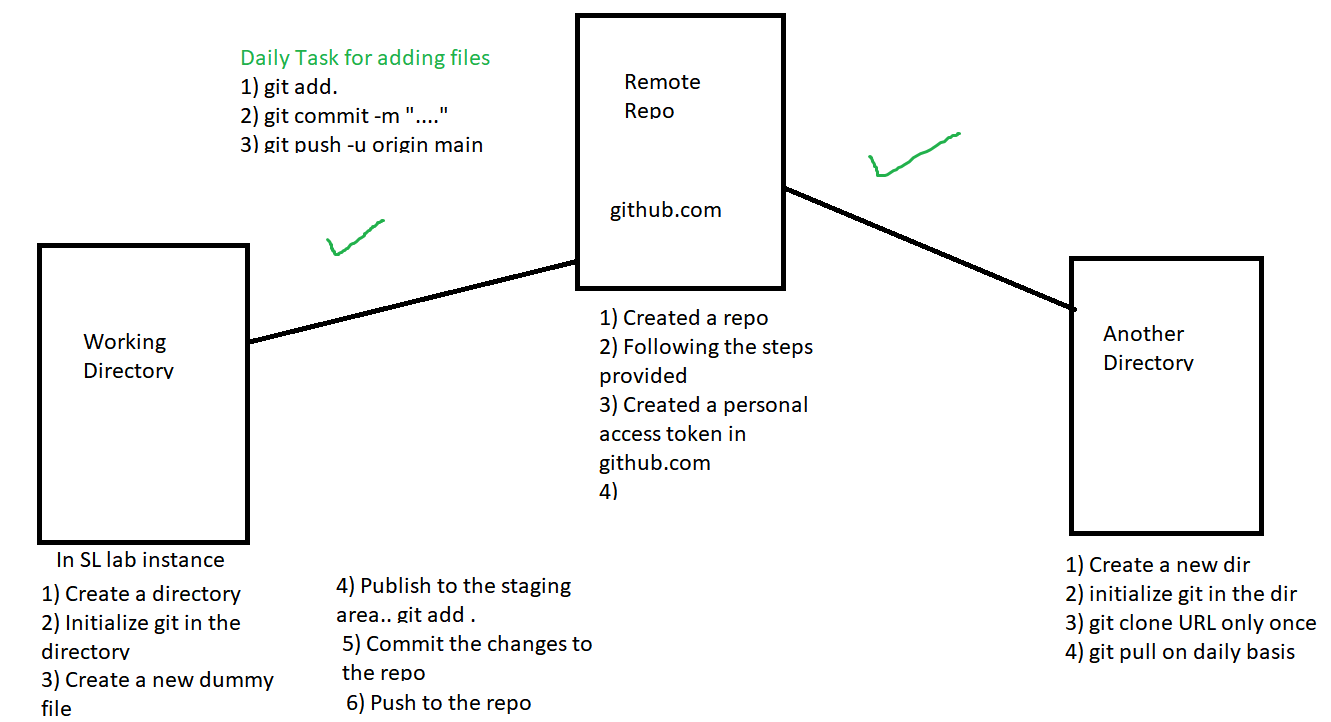
git remote add origin https://ghp\_bBsqoXv65HaLmgoWRVLFBhVvmglUVN4GLGnZ@github.com/raneemr05/javatraining.git

1. git push -u origin main

OR

sudo git push -u origin main

12) Check the git repo inside of your github.com to find the updated file added



1) Check if Java is installed (java -version)

2) If not then install using java installer

3) Set up enviromental variables for the java path

4) Write your first java program and compile and run it

/\* Welcome everyone

\* This is my first program in Java

\*/

public class HelloWorld

{

//This is the main method

public static void main(String args[])

{

//This method will print the message

System.out.println("Hello World"); System.out.println("Next line");

}

}

5) Compile it using **javac Filename.java**

6) After compilation you will get a .class file

7) You can run the file using **java Filename**

**Class Syntax**

**public (access modifer) class ClassName**

**{**

**Write methods/functions**

**Create variables**

**}**

**Comments in Java**

Lines which are not compiled or executed by the compiler

1. Single line comment //
2. Multi-line comment /\*\*/

**Variable:** Is a placeholder

**Syntax**

**Declare and initialize the variable**

int a = 10;

DataType VariableName = Value

**Declare a variable**

int a;

**Data Types**

Indicates what kind of data you can store in the variable

2 kind of data types

1. ***Primitive data type***

Store only value

1. Byte 1 byte
2. Short 2 byte
3. Int 4 byte
4. Long 8 byte
5. Float 4 byte
6. Double 8 byte
7. Char 2 byte
8. Boolean 1 byte
9. ***Non-primitive***

Store the value and reference of another data type also

Arrays, Class, Interfaces

**Conditional Statements: used to check conditions**

1. If
2. If else
3. If else if
4. Switch statement (multiple cases - suitable for creating a menu in application)

int a = 10;

If (a==10)

System.out.println(“a is 10”);

Else

System.out.println(“a is not equal to 10)

Switch(variable){

Case a: S.o.p (“You’re in 1st option);

}

**Loop**

To execute some line of code in iteration/multiple times.

3 parts of loop

1. Initialize the looping variable
2. Put condition for the looping variable
3. Increment or decrement looping variable
4. **For loop**

Syntax:

for(int a=0; a<10; a++) //a++ means a = a + 1 //increment

//if decrement I will write a--

for(int a=10; a>=0; a--)

{

S.o.p(“Value of a is”+a);

}

1. **While Loop**

Initialize the value

While (condition)

{

Increment/decrement

}

1. **Do while loop**

do{

Increment/decrement

} while(condition)

**Day 2 (26 oct 2022)**

1. **Methods**

**Method with no return type**

Void is used to return nothing

AccessModifier ReturnType MethodName()

{

//Method body

}

public void MyMessage(){

}

**Method with 1 parameter and call by value**

package com;

public class CallbyValueDemo {

int data = 10; //Instance Variable

//Parameterized Method - it accepts one int value as a parameter

void changevalue(int data)

{

data = data + 10; //Local variable

System.out.println("Value of local data" + data);

}

public static void main(String[] args) {

CallbyValueDemo demo = new CallbyValueDemo(); //Creating object

System.out.println("Before changing" + demo.data); //10

demo.changevalue(50); //Call by value

System.out.println("After changing" + demo.data); //10

}

}

**Method Overloading**

package com;

public class MethodOverloading {

//Method with same name and return type

void add(int a, int b)

{

int sum = a +b;

System.out.println("The value of sum is" + sum);

}

//Method with same name and return type

void add(int a, int b, int c)

{

int sum = a +b+c;

System.out.println("The value of sum is" + sum);

}

public static void main(String[] args) {

MethodOverloading obj1 = new MethodOverloading();

obj1.add(5, 5);

obj1.add(2,6,2);

}

}

**Classes and Objects**

Objects: any real world entity which has some properties and behaviour

Classes: Blue print or template of object

Example ( car class and 2 objects of class)

Car (Properties and methods/behaviors)

( color, company, engine no, wheels….)

(start() stop(), gear()…..)

Car2 Toyota

Company = Toyota

Color= Blue

Start()

Stop()

Car1 BMW

Company = BMW

Color= Blue

Start()

Stop()

package com;

public class CarApp {

//Properties of car - Instance variable

int carId;

String color;

int wheels;

static String CR = "12738791283";

//Methods of car

void start()

{

System.out.println("Car Starting");

}

void stop()

{

System.out.println("Car Stopping");

}

void display()

{

String company ="ABC Carshowroom"; //local Variable

System.out.println("Car Id is " + carId);

System.out.println("Color of the car is" + color);

System.out.println("Number of wheels" + wheels);

System.out.println("Cars owned by " + company);

}

public static void main(String[] args) {

//Creating 1st object

CarApp car1 = new CarApp();

car1.carId = 001;

car1.color ="Red";

car1.wheels = 4;

System.out.println("Car 1 object is created");

car1.display();

car1.start();

car1.stop();

}

}

**Types of variables**

1. Local Variable: defined inside the method body
2. Instance Variable: declared inside the class but outside the methods
3. Static variable: To set a value throughout the program static is used

Class A{

int a = 10; //Instance Variable

**static** String name = ”Java” //Static Variable

void method()

{

int b =5; //Local Variable

System.out.println(“Value of b is” + b);

}

int b = 10; //Instance Variable

**Constructor**

1. A special method used to initialized the object
2. Invoked at the time of object creation
3. Name of the constructor must be the same as of class name
4. It provides data for the object that’s why its know as constructor
5. It must have no explicit return type

**Syntax**

<class\_name>(){}

**Types of constructor**

1. Default or no-arg
2. Parameterized

package com;

public class CarApp {

//Properties of car - Instance variable

int carId;

String color;

int wheels;

static String CR = "12738791283";

//No-arg Constructor

CarApp(){

System.out.println("Object Created");

}

//Parameterized Constructor using this keyword

CarApp(int carId, String color,int wheels)

{

//calling the default constructor using this keyword

this();

this.carId= carId;

this.color = color;

this.wheels = wheels;

}

CarApp(int car, String col)

{

carId = car;

color = col;

}

//Methods of car

void start()

{

System.out.println("Car Starting");

}

void stop()

{

System.out.println("Car Stopping");

}

void display()

{

String company ="ABC Carshowroom"; //local Variable

System.out.println("Car Id is " + carId);

System.out.println("Color of the car is" + color);

System.out.println("Number of wheels" + wheels);

System.out.println("Cars owned by " + company);

}

public static void main(String[] args) {

//Creating 1st object

//new keyword allocates memory to the object

//Calling default constructor

CarApp car1 = new CarApp();

car1.carId = 1;

car1.color ="Red";

car1.wheels = 4;

System.out.println("Car 1 object is created");

car1.display();

car1.start();

car1.stop();

//Calling 3 parameter constructor

CarApp car2 = new CarApp(2,"Blue",4);

car2.display();

}

}

**This keyword**

*If instance variable and local variable have same name then local variable hide the visibility of instance variable. To refer to instance variable we have to use this keyword. This keyword is use to refer the current object.*

*this.instancevariableName*

**Day 3 (27 oct 2022)**

**Pillars of OOP/Java**

**4 Pillars (PAIE)**

**P**olymorphism

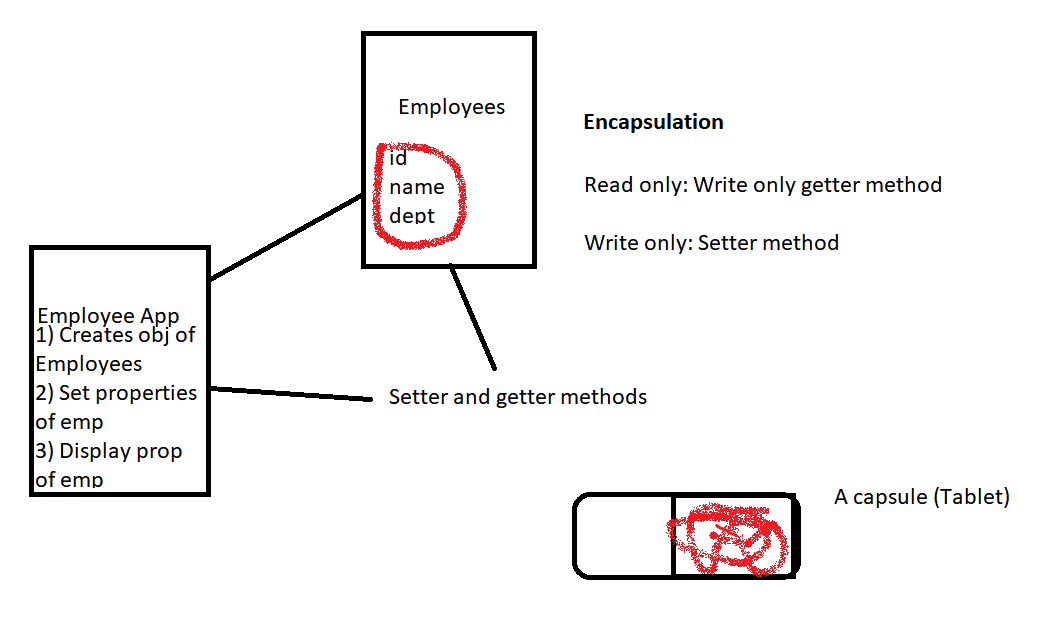
**A**bstraction

**I**nheritance

**E**ncapsulation

1. **Encapsulation**

Binding or wraping data(properties) and methods in a single unit



package com;

class Employees{

private int empId;

private String name;

private String dept;

private int Salary;

public void setEmpName(String name)

{

this.name = name;

}

//public void getEmpName()

//{

// System.out.println("Employee name is " + name);

// }

public void setSalary(int Salary)

{

if(Salary < 1000)

{

System.out.println("Sorry! the salary has been automatically set to a default value");

this.Salary = 1000;

}

else

{

this.Salary = Salary;

}

}

public void display()

{

System.out.println("Salary is:" + Salary);

System.out.println("Name of emp is " + name);

}

}

public class EmployeeApp {

public static void main(String[] args) {

// TODO Auto-generated method stub

Employees emp1 = new Employees();

//emp1.empId = 1;

//emp1.dept = "Sales";

//emp1.name = "Alice";

emp1.setEmpName("Alice");

emp1.setSalary(200);

emp1.display();

}

}

1. **Abstraction**

It is a process of hiding the implementation details and showing only the functionality to the user

**Abstract** class which can have abstract and non-abstract methods

Lets you focus on what the object does instead of how does it.

In java there are 2 ways to achieve abstraction

1. Abstract class (0 to 100%)
2. Interface (100%)

**Abstract Class Example**

package com;

abstract class Animal {

//abstract method

abstract void makeSound(); //Only method declaration

//Non-abstract method

public void run()

{

System.out.println("I can run");

}

}

class Dog extends Animal{

public void makeSound()

{

System.out.println("Bark Bark");

}

}

class Cat extends Animal{

public void makeSound()

{

System.out.println("Meow Meow");

}

}

class Animals

{

public static void main(String[] args)

{

Dog dog1 = new Dog();

dog1.makeSound();

dog1.run();

//Creating object for cat

//Using Abstract class as a reference on the left side and constructor of child class on the right side

Animal cat = new Cat();

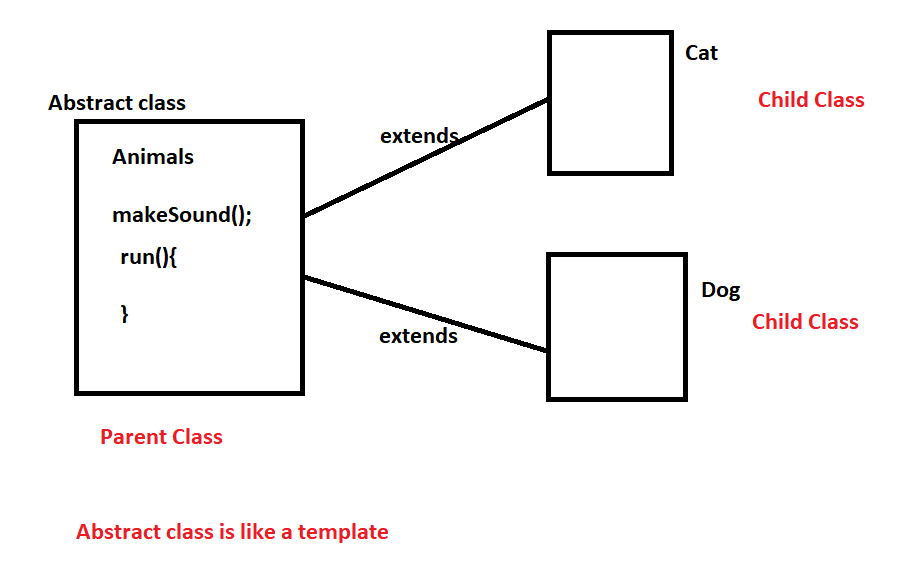
//Animal animal1 = new Animal();

cat.makeSound();

cat.run();

}

}



1. **Polymorphism**

Poly means “many” morphs means “forms”.

We can perform a single action in different ways.

2 types of polymorphism

* Compile-time poly
* Run-time poly

We can perform by

* Method overloading (different parameters list, same return type, same name)
* Method overriding

package com;

class Bike{

public void speed()

{

System.out.println("Standard Speed is 60km/h");

}

}

class Honda extends Bike{

public void color()

{

System.out.println("Color is Red");

}

}

class tvs extends Bike{

public void color()

{

System.out.println("Color is Blue");

}

public void speed()

{

super.speed(); //Super keyword is used to call parent class method

System.out.println("My Speed is 20km/h");

}

}

public class RuntimPoly {

public static void main(String[] args) {

// Creating objects for child class

Honda h1 = new Honda();

h1.color();

h1.speed(); //Calling super class method

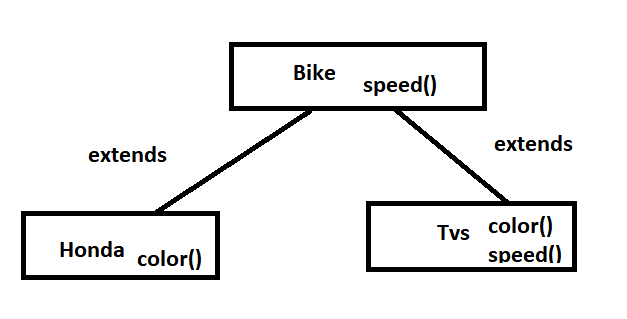
tvs tvs1 = new tvs();

tvs1.color();

tvs1.speed();

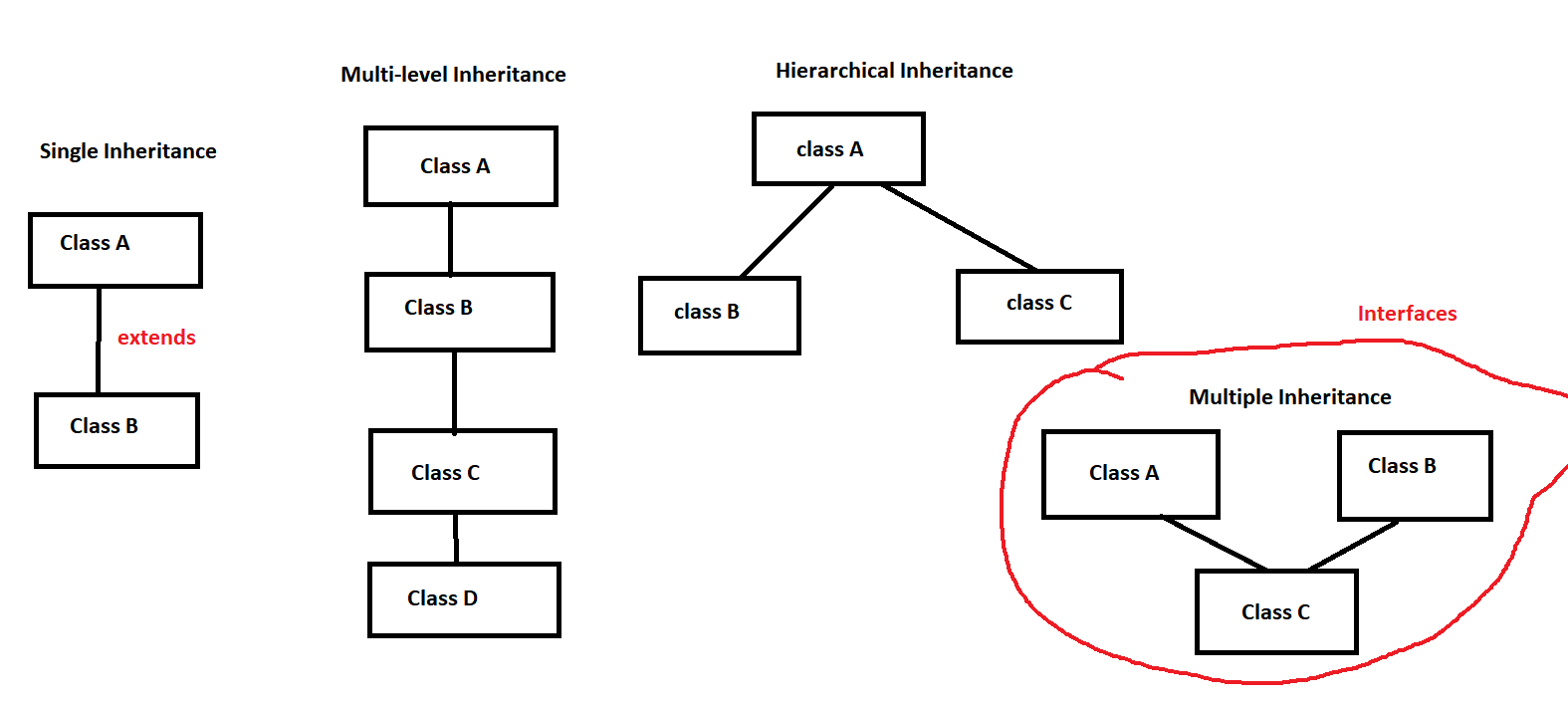
}

}



1. **Inheritance**

Types of inheritance



Class A{ }

Class B extends class A {}

Class C extends class B {}

**Association:** If we want to achieve ‘has a relationship’ we have to create the object of one class in another class

Class A{

B obj1 = new obj1();

}

**Aggregation:**

1. A child can exist independently of the parent
2. It is a ‘has-a’ relation
3. Weak association

Employee has a address

**Employee and Address**

class Employee{

Address add = new Adress();

}

class Address {

City, state, post code etc

Employee emp = new Employee(); //

}

**Composition**

1. Child can’t exist independently of the parent
2. Part-of relation
3. Strong association

StudentRecords is part of Student

Class Student{

StudentRecord sr = new StudentRecord();

}

class StudentRecord{

}

**Example**

package com;

import java.util.Scanner;

class Employee{

private int id;

private String name;

Address add = new Address();

Scanner sc = new Scanner(System.in);

public void setEmployee()

{

System.out.println("Enter the id");

id = sc.nextInt();

System.out.println("Enter the name");

name = sc.next();

}

public void getEmployee()

{

System.out.println("Id is " + id);

System.out.println("Name is " + name);

}

}

class Manager extends Employee{

int teamcount; //Instance variable

public void setMgr()

{

setEmployee();

System.out.println("Enter the count of team members you manage");

teamcount = sc.nextInt();

add.readAdd();

}

public void getMgr()

{

getEmployee();

System.out.println("The team members are "+ teamcount);

add.dispAdd();

}

}

class Address{

private String city;

private String state;

Scanner s = new Scanner(System.in);

public void readAdd()

{

System.out.println("Enter the city");

city = s.next();

System.out.println("Enter the state");

state = s.next();

}

public void dispAdd()

{

System.out.println("City is " + city);

System.out.println("State is " + state);

}

}

public class EmployeeSample {

public static void main(String[] args) {

Manager mgr = new Manager();

//To take input from the keyboard user

mgr.setMgr();

//To display manager details

mgr.getMgr();

}

}

**Interface (**Multiple Inheritance**)**

* It is a reference/non-primitive data type
* Provides 100% abstraction
* It supports multiple inheritance
* All the properties are public, static and final (psf)
* All the methods are public and abstract

**Syntax**

interface interfaceName{

Properties;

Methods;

}

**Example**

package com;

public interface InterfaceSample {

public static final String name = "Java";

//value of pi, you can keep it as final variable

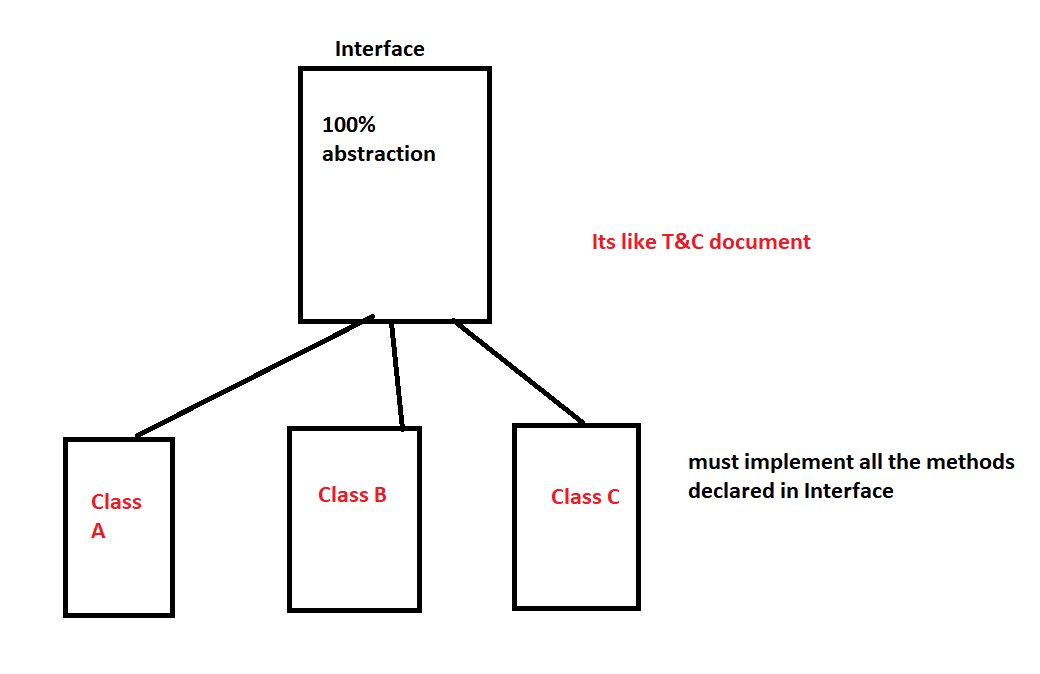
//only method declaration allowed

//all methods are abstract by default

public void display();

public void display2();

}



**Day 4 (28 oct 2022)**

package com;

//1st interface

interface abc{

String name = "Java";

//value of pi, you can keep it as final variable

//only method declaration allowed

//all methods are abstract by default

public void display();

public void display2();

}

//2nd Interface

interface xyz extends abc

{

public void display3();

}

//Class - implements used to do the implementation for the interface

class InterfaceImpl implements xyz{

@Override

public void display() {

// TODO Auto-generated method stub

System.out.println("This is from abc interface");

}

@Override

public void display2() {

// TODO Auto-generated method stub

System.out.println("This is from abc interface");

}

@Override

public void display3() {

// TODO Auto-generated method stub

System.out.println("This is from xyz interface");

}

}

public class InterfaceSample{

public static void main(String[] args)

{

InterfaceImpl imp = new InterfaceImpl();

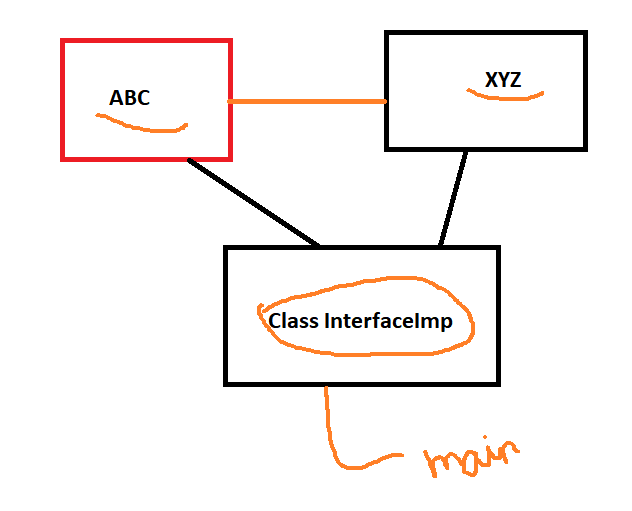
imp.display();

imp.display2();

imp.display3();

}

}



**Package**

Set of classes

Access Modifiers

1. Protected

package pack;

public class Mypkg {

public void msg() {

System.out.println("Hello from Mypkg");

}

}

package pack2;

import pack.\*;

public class B extends Mypkg {

public static void main(String[] args)

{

B obj1 = new B();

obj1.msg();

}

}

1. Public
2. Private
3. Default

**ARRAYS**

* Store multiple values
* Same data type
* Fixed sized elements

Name1, name2 …. name10

String[] learners\_names = new String[10];

String[] learners\_names ={“Scott”,”Israel”,”Crystal”,……};

package com;

public class ArraysSample {

public static void main(String[] args) {

// TODO Auto-generated method stub

String[] learners\_names ={"Scott","Israel","Crystal","Tim"};

System.out.println("Value of 0 index position" + learners\_names[0]);

System.out.println("Value of 3 index position" + learners\_names[3]);

System.out.println("Size of the array " + learners\_names.length);

System.out.println("Printing the array values using for loop");

for(int i=0; i < learners\_names.length; i++)

{

System.out.println(learners\_names[i]);

}

//Printing using the for each loop

System.out.println("Printing the array values using for each loop");

for(String n : learners\_names)

{

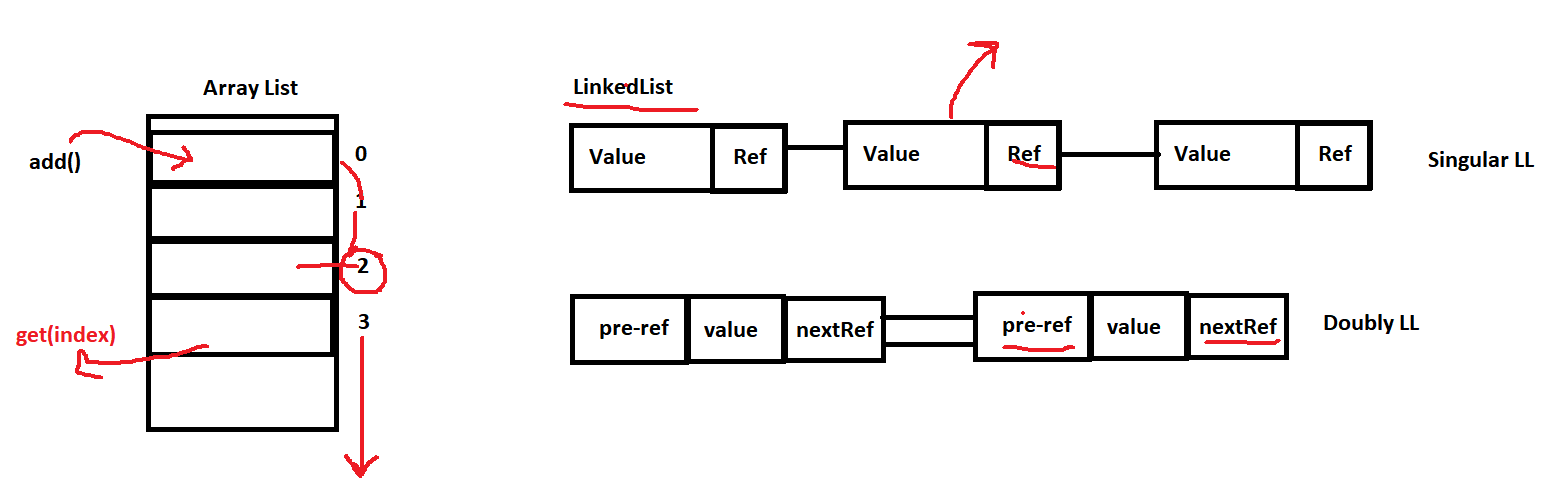
System.out.println(n);

}

}

}

**COLLECTIONS**



1. **List**
2. **ArrayList**

Arrays vs ArrayList

* In arrays you can store same data type value but in ArrayList you can store any types
* Arrays are fixed in size but the Array List provides dynamic memory.
* ArrayList provides extended functionality with pre-defined methods which makes it easier to program. Removing and add elements to an array is more complex

package com;

import java.util.ArrayList;

public class ArrayListDemo {

public static void main(String[] args)

{

ArrayList al = new ArrayList();

al.add(10);

al.add(20);

al.add(30);

al.add(40);

al.add("Java");

al.add(true);

System.out.println(al);

System.out.println("Print value at index 1 is:" + al.get(1));

al.add(1, 70);

System.out.println("Print value at index 1 after changing is:" + al.get(1));

al.remove(3);

System.out.println(al);

al.clear();

}

}

**LinkedList**

It uses node to store the value. Node is divided into 2 or parts, it depends upon the type of LL you create.

1. Singular LL
2. Doubly LL
3. Circular LL

package com;

import java.util.LinkedList;

public class LinkedListDemo {

public static void main(String[] args) {

LinkedList ll = new LinkedList();

ll.add(20);

ll.add(40);

ll.add(1, 55);

System.out.println(ll);

ll.remove(2);

System.out.println(ll);

ll.addFirst(1);

ll.addLast(100);

System.out.println(ll);

ll.removeFirst();

ll.removeLast();

System.out.println(ll);

}

}

**Queue**

package com;

import java.util.LinkedList;

import java.util.PriorityQueue;

import java.util.Queue;

public class QueueDemo {

public static void main(String[] args) {

Queue q1 = new PriorityQueue();

q1.add("Java");

q1.add("OOP");

q1.add("Trainings");

q1.add("Applications");

q1.add("Development");

System.out.println(q1);

System.out.println(q1.poll());

System.out.println(q1);

Queue q2 = new LinkedList();

q2.add("Java");

q2.add("OOP");

q2.add("Trainings");

q2.add("Applications");

q2.add("Development");

System.out.println(q2);

System.out.println(q2.poll());

}

}

**Sets (no duplication - similar to list)**

1. HashSet (unordered elements)
2. Linked HashSet (maintain same order)

package com;

import java.util.HashSet;

import java.util.LinkedHashSet;

public class HashSetDemo {

public static void main(String[] args) {

System.out.println("HashSet Demo");

HashSet hs = new HashSet();

hs.add(7);

hs.add(2);

hs.add(8);

hs.add(22);

System.out.println(hs);

HashSet hs1 = new HashSet();

hs1.add(2);

hs1.add(5);

hs.add(hs1);

System.out.println(hs1);

System.out.println("Size of the hashset is " + hs1.size());

System.out.println("Empty" + hs1.isEmpty());

System.out.println("Linked HashSet Demo");

LinkedHashSet lhs = new LinkedHashSet();

lhs.add(3);

lhs.add(4);

lhs.add(8);

lhs.add(11);

System.out.println(lhs);

}

}

1. TreeSet (Ascending order)

package com;

import java.util.TreeSet;

public class TreeSetDemo {

public static void main(String[] args) {

System.out.println("Using Tree Set API");

TreeSet ts = new TreeSet();

ts.add(1);

ts.add(100);

ts.add(48);

ts.add(4);

ts.add(8);

ts.add(11);

ts.add(2);

System.out.println(ts);

System.out.println(ts.headSet(3));

System.out.println(ts.tailSet(3));

System.out.println(ts.subSet(2, 8));

}

}

**Map (Dictionary)**

Uses key-value pair

Key is unique

Value may be duplicate

EMP001 --> “Alice”,”Manager”,”Sales”,1500

package com;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.TreeMap;

public class MapDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

//Map mm = new HashMap(); //unordered elements

//Map mm = new TreeMap(); //Ascending order

Map mm = new LinkedHashMap();

mm.put(1, "Alice");

mm.put(2, "Bob");

mm.put(100, "Charles");

mm.put(8, "Charles");

mm.put(4, "Charles");

mm.put(3, "Charles");

System.out.println(mm);

mm.put(2, "David");

System.out.println(mm);

System.out.println(mm.get(1));

System.out.println(mm.get(50));

System.out.println(mm.containsValue("David"));

System.out.println(mm.containsKey(5));

}

}

**Day 5 (31 oct 2022)**

C**ollection Framework with Generics**

CollectionClassName<Type> objectName = new CollectionClassName<Type> ();

Type:

Integer

String

Float

Double

Character

Or any user-defined Class

int a = 10 //Primitive data type for int

Integer //Its an object

**Retrieve the elements from the collection one by one**

Iterator<String> itr=names.iterator();

**while**(itr.hasNext()){

System.***out***.println(itr.next());

1. Create an object of Iterator class by using the collection object i.e. names
2. Use the Iterator object as a looping variable and hasNext method to check if the collection has more value in it
3. Next method is used to iterate the elements one by one and print them
4. Remove method is used to remove the element from the collection

**STACK**

package com;

public class StackDemo

{

//Defined the size of the stack

static final int MAX = 1000;

int top;

int a[] = new int[MAX];

boolean isEmpty()

{

return (top < 0);

}

StackDemo()

{

top = -1;

}

boolean push(int x)

{

if (top >= (MAX-1))

{

System.out.println("Stack Overflow");

return false;

}

else

{ // ++top is top = top + 1

a[++top] = x;

System.out.println(x + " pushed into stack");

return true;

}

}

int pop()

{

if (top < 0)

{

System.out.println("Stack Underflow");

return 0;

}

else

{

int x = a[top--];

return x;

}

}

public static void main(String args[])

{

StackDemo s = new StackDemo(); // top value sets to -1 means stack is empty

s.push(10);

s.push(20);

s.push(30);

s.push(50);

System.out.println(s.pop() + " Popped from stack");

}

}

**Inner Classes**

class Java\_Outer\_Class{

class Inner\_Class{

//Can access methods, variables of the outer class

}

}

**Strings**

* A predefined class in Java
* Part of java.lang package

2 ways to create the String Class object

* String course = “Java” //Literal style
* String course = new String(“Java Training”); //Creates an object using new keyword

package com;

public class StringDemo {

public static void main(String args[])

{

String course = "Welcome to the the java Training"; //Literal Style

String courseName = new String("Java Trainings"); //An object

System.out.println(course);

System.out.println(courseName);

System.out.println("Length of the string is" + course.length());

System.out.println(courseName.length());

System.out.println(courseName.toLowerCase());

System.out.println(courseName.toUpperCase());

System.out.println(courseName.indexOf('v'));

System.out.println(courseName.substring(4));

}

}

**RegEx**

Pattern: ^(.+) @ (.+) $

Values for matching:

[abc@example.com](mailto:abc@example.com)

[abc123@example.com](mailto:Abc123@example.com)

**Exception Handling**

**Exception**

* It is a pre-defined object
* It occurs when any abnormal condition occurs during the execution of the program
* To handle the generated exception using some technique is known as exception handling

After compiling you get a .class file (javac)

A bytecode is only understood by JVM and its WORA (platform independent)

Compile time error --> Syntax or type error

Java.lang package in which Error and Exception classes are pre-defined

Error: It generates at the run time which we can’t handle it.

Exception: Generates at the run time and we can handle it

In Java to handle the exception **5** keywords provided

1. try
2. catch
3. finally
4. throw
5. throws

**Syntax for try catch**

try {

}

catch(Exception e){

}

**Try with Single Catch**

Used generic exception class in catch parameter

package com;

public class TryCatchDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

System.out.println("Try Catch Demo");

int a = 5;

int b = 0;

try {

int res = a/b;

System.out.println("Result of a and b is "+ res);

}

catch(Exception e)

{

//Using custom message

System.out.println("No worries I will take care of this exception");

//Using Pre-defined message

System.out.println(e.getMessage());

//Using exception name and message

System.out.println(e.toString());

}

System.out.println("Ending of try catch demo");

System.out.println("Bye");

}

}

**Try with multiple catch block**

try{

}

catch(ArithmeticException e)

{

}

catch(ArrayIndexOurOfBoundException e){

}

catch(NullPointerException){

}

package com;

public class TryWithMultipleCatch {

public static void main(String[] args) {

// TODO Auto-generated method stub

System.out.println("Try with Multiple Catches");

int num[] = {10,20};

try {

int result = 10/num[2];

System.out.println("result is " + result);

}

catch(ArithmeticException e)

{

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

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Array Index " + e);

}

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

}

}

**Try Block:** The code which generates the exception, it can be one line or multi line code

**Catch Block:** Only execute if any exception occurs . Also known as exception solver block

**Finally Block:** It will always execute irrespective of your exception. Its mainly used to close resources i.e. files.

*Try - Catch*

*Try - Catch - Catch - Catch*

*Try - Catch - Finally*

*Try - Finally (Exception is not handled)*

**Throw:** Used to raise or generate any user-defined exception according to your own requirements.

Syntax

throw new Exception(); or throw new ExceptionSubClass();

package com;

public class ThrowExample {

public static void main(String[] args) {

int a = 10;

int b = 5;

try {

if (a > b)

{

throw new Exception("a>b");

}

else {

System.out.println("Not greater");

}

}

catch(Exception ex)

{

System.out.println(ex);

}

}

}

**Throws**

It can be used with the method signature to throw the exception to the caller method

Syntax

Void display() throws Exception{

}

package com;

public class ThrowsDemo {

static void display1() throws Exception{

int a = 10/0;

System.out.println("Display 1 method");

}

static void display2() throws Exception

{

//try {

display1();

//}

//catch(Exception e)

//{

System.out.println("display 2 method");

//}

}

public static void main(String[] args)

{

try {

display2();

}

catch(Exception e)

{

System.out.println("Main Method");

}

}

}

**Day 6 (1st Nov 2022)**

**File Handling**

* A class java.io.file package
* Allow us to work with different formats of files

**File Operations in Java**

Basic 4 operations on a file

* Create a File

package FilesDemo;

import java.io.File;

import java.io.IOException;

public class CreateFileDemo {

public static void main(String[] args) {

//Create a file object in the current directory

File file = new File("myfile.txt");

try {

//trying to create a file

boolean check = file.createNewFile();

if(check)

{

System.out.println("The new file is created");

}

else

{

System.out.println("The file already exists!");

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

* Get the file information

package FilesDemo;

import java.io.File;

public class FileInformation {

public static void main(String[] args) {

// TODO Auto-generated method stub

File file1 = new File("myfile.txt");

if (file1.exists()) {

//Returning the file name

System.out.println("File name:" + file1.getName());

//Returning the path of the file

System.out.println("Absolute Path: " + file1.getAbsolutePath());

//Checking if its Writeable

System.out.println("Writeable: " + file1.canWrite());

//Checking if its Readable

System.out.println("Readable: " + file1.canRead());

//Checking the file size

System.out.println("File size in bytes: " + file1.length());

}

else

{

System.out.println("File does not exist");

}

}

}

* Write some text to the file

package FilesDemo;

import java.io.FileWriter;

import java.io.IOException;

public class WriteFileDemo {

public static void main(String[] args) {

String data = "This is the data we are writing from a java file";

try {

//Create a writer object using FileWriter class

FileWriter output = new FileWriter("myfile.txt");

//Writes string data to the file

output.write(data);

System.out.println("Data is written to the file successfully");

output.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

* Read from a file

package FilesDemo;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class ReadFromFile {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

File myfile = new File("myfile.txt");

//Scanner object used to read the file

Scanner myReader = new Scanner(myfile);

//Iterate through the file and read the content and store values in a string variable

while(myReader.hasNextLine()) {

String data = myReader.nextLine();

System.out.println(data);

}

myReader.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

**Multi-threading**

**Program:**  Set of instructions to perform a specific task

**Processor:** Who is responsible to execute the task

**Process:** Time taken to execute the program

**Thread:** Small execution of a code within a process . Light weighted process. It takes less resources or memory of the machine

* In java there is one default thread that always execute i.e. main method
* It comes under java.lang package which contains a lot of pre-defined methods.
* Find the default thread details running inside a main method (currentThread)

**Syntax of creating Thread Class object**

Thread t = Thread

1. Default Thread using currentThread method

package Threading;

public class DefaultThreadDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

Thread t = Thread.currentThread();

System.out.println(t);

}

}

Thread[main,5,main]

Main --> name of the thread

5 --> Priority of the thread

Main --> group of the thread

Default thread priority is 5.

We can set min 1 and max 10.

**Example**

package Threading;

public class DefaultThreadDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

Thread t = Thread.currentThread();

System.out.println(t);

String name = t.getName();

System.out.println("Name of the thread is " + name);

int priority = t.getPriority();

System.out.println("Priority of thread is " + priority);

t.setPriority(8);

t.setPriority(Thread.MIN\_PRIORITY);

t.setName("My Thread");

System.out.println(t);

}

}

1. **Creating thread** 
   1. using extends Thread class
      1. Create a normal class but that class must extends Thread class
      2. Create the thread class reference
      3. Call the start() method using the reference. Start is a pre-defined method whch is used to start the thread
      4. Start method internally call run()
      5. If you want to write your custom code we have to override the run method inside the class

package Threading;

class mythread extends Thread{

@Override

public void run() {

// TODO Auto-generated method stub

super.run();

for(int i =0; i<10; i++)

{

System.out.println("value of I is" + i);

}

}

}

class anotherthread extends Thread{

@Override

public void run() {

// TODO Auto-generated method stub

super.run();

for(int j =0; j<10; j++)

{

System.out.println("value of j is" + j);

}

}

}

public class CreateThreadDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

mythread obj1 = new mythread(); //Indirectly referencing the thread class

obj1.start();

anotherthread obj2 = new anotherthread();

obj2.start();

}

}

* 1. Implementing Runnable interface
     1. Create a class but it must implements the Runnable interface
     2. Runnable interface must contains the run method so we have to override
     3. Create an object of your class
     4. Create a reference of thread class and call the start method

package Threading;

class firsthread implements Runnable{

@Override

public void run() {

// TODO Auto-generated method stub

for(int i =0; i<10; i++)

{

System.out.println("value of I is" + i);

}

}

}

class secondthread implements Runnable{

@Override

public void run() {

// TODO Auto-generated method stub

for(int j =0; j<10; j++)

{

System.out.println("value of J is" + j);

}

}

}

public class ImplementsRunnable {

public static void main(String[] args) {

// TODO Auto-generated method stub

firsthread obj1 = new firsthread();

secondthread obj2 = new secondthread();

Thread t1 = new Thread(obj1); //Create a reference of thread class

Thread t2 = new Thread(obj2);

t1.start(); //Call the start method

t2.start();

}

}

**Multiple Thread doing same task**

package Threading;

class Task implements Runnable{

@Override

public void run() {

// TODO Auto-generated method stub

Thread tt = Thread.currentThread();

String name = tt.getName();

for(int i=0; i<10; i++)

{

System.out.println(name + "=" + i);

}

}

}

public class MultipleThreads {

public static void main(String[] args) {

// TODO Auto-generated method stub

Thread tt = Thread.currentThread();

System.out.println(tt);

Runnable t = new Task(); //run-time polymorphism

Thread t1 = new Thread(t);

Thread t2 = new Thread(t);

Thread t3 = new Thread(t);

t1.setName("first thread");

t2.setName("second thread");

t3.setName("Third thread");

t1.start();

t2.start();

t3.start();

}

}

**Synchronization**

Help to block or lock the thread. It will allow to use only one thread all resource at a time.

You can use synchronized keyword

Keyword is used with method or block

**Wait, notify and notifyAll**

All belong to Object class

Wait is used to suspend the thread

Notify is used to resume or call back the suspended thread

**Inner Communication Thread Examples**

package Threading;

class Info implements Runnable{

@Override

public synchronized void run() {

// TODO Auto-generated method stub

Thread t = Thread.currentThread();

String name = t.getName();

for (int i = 0; i <10; i++)

{

try {

System.out.println(name + "=" +i);

Thread.sleep(400);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

public class InnerThreadDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

Info obj1 = new Info();

Thread t1 = new Thread(obj1);

Thread t2 = new Thread(obj1);

Thread t3 = new Thread(obj1);

t1.setName("Alice");

t2.setName("Bob");

t3.setName("Charles");

t1.start();

t2.start();

t3.start();

}

}

**Lifecycle of a thread**

**4/5 states**

1. New
2. Runnable (start())
   1. Non-Runnable (Blocked - optional state if you want to put on sleep)
3. Running (run())
4. Terminated

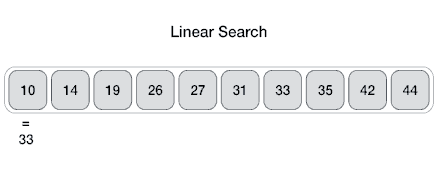
**Day 7 (2 - Nov -2022)**

**Searching**

int num[] = {3,6,2,8,9,4}

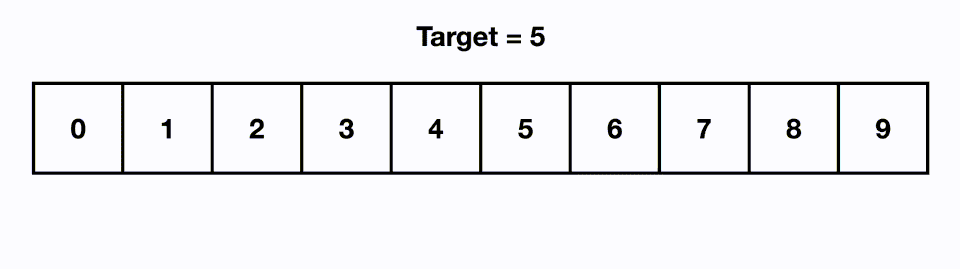
**Linear Search**

* It is sequential search.
* A key element = 8
* An Array
* +ve: When the search element matches the first element in array then linear search is the **best case** O(1) (Big O Notation)
* Worst case: Not a good fit for large arrays O(n)
* Good for small or medium array.
* Linear search doesn’t needs array elements to be sorted



**Binary Search**

* Divide and conquer technique
* 1st step is the array must be sorted before doing the search
* Find the mid value of the array: Total length of the array is 5: Mid value will be 5/2 = 2.5. the mid index will be 2
* Inside of the loop it will start searching and matching the value with the search value
* Once the value is found from the array it will return the index.
* Best Case: O(1)
* Worst Case O(log n)
* Large arrays, finite arrays

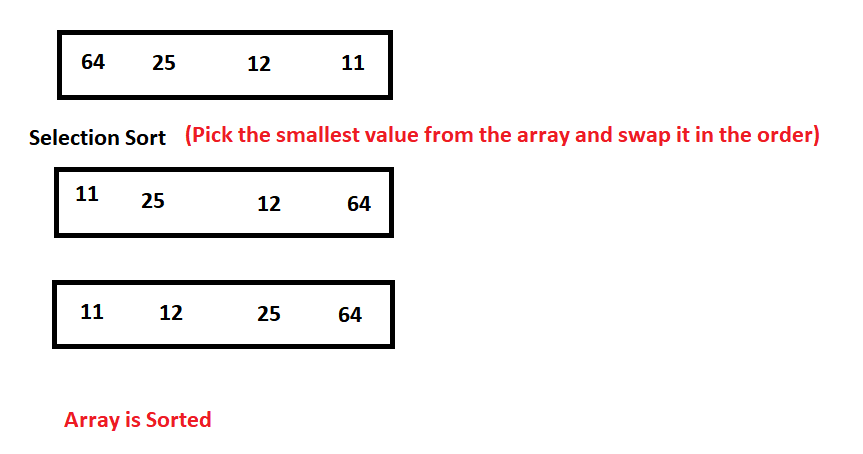


**Exponential Search**

* Works for unbounded array/searches where the size of the array is infinite
* It works better than Binary search for bounded arrays
* Complexity: O(log n)
* Works for good large arrays but unbounded

**Sorting**

1. **Selection Sort**



Complexity: Worst case it will be O(n^2)

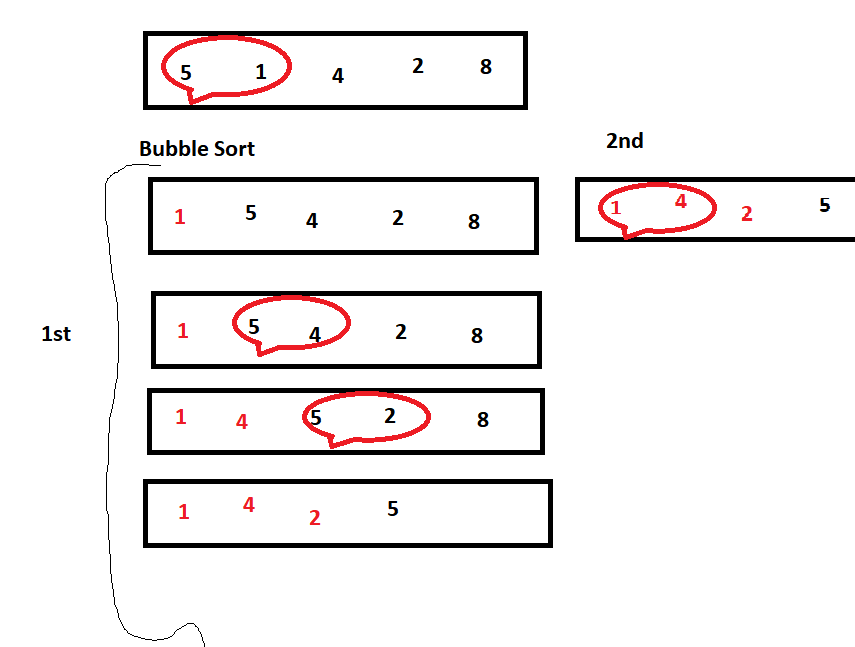
Not suitable for sorting large sets

1. **Bubble Sort**

Worst Case: If the elements arranged in descending order and you want to sort for ascending O(n^2)

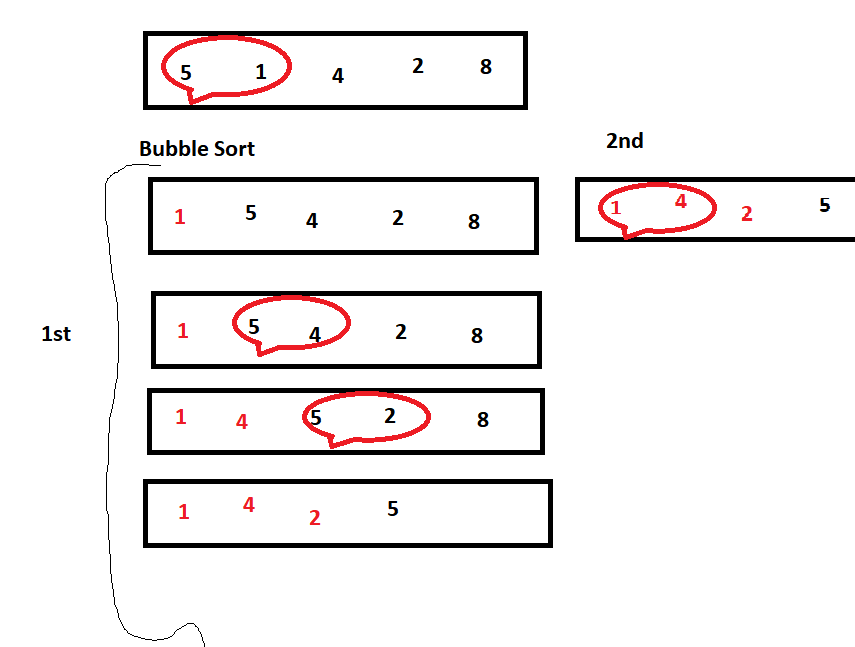
Best Case: O(n) if you have already sorted array

Not suitable for large data sets



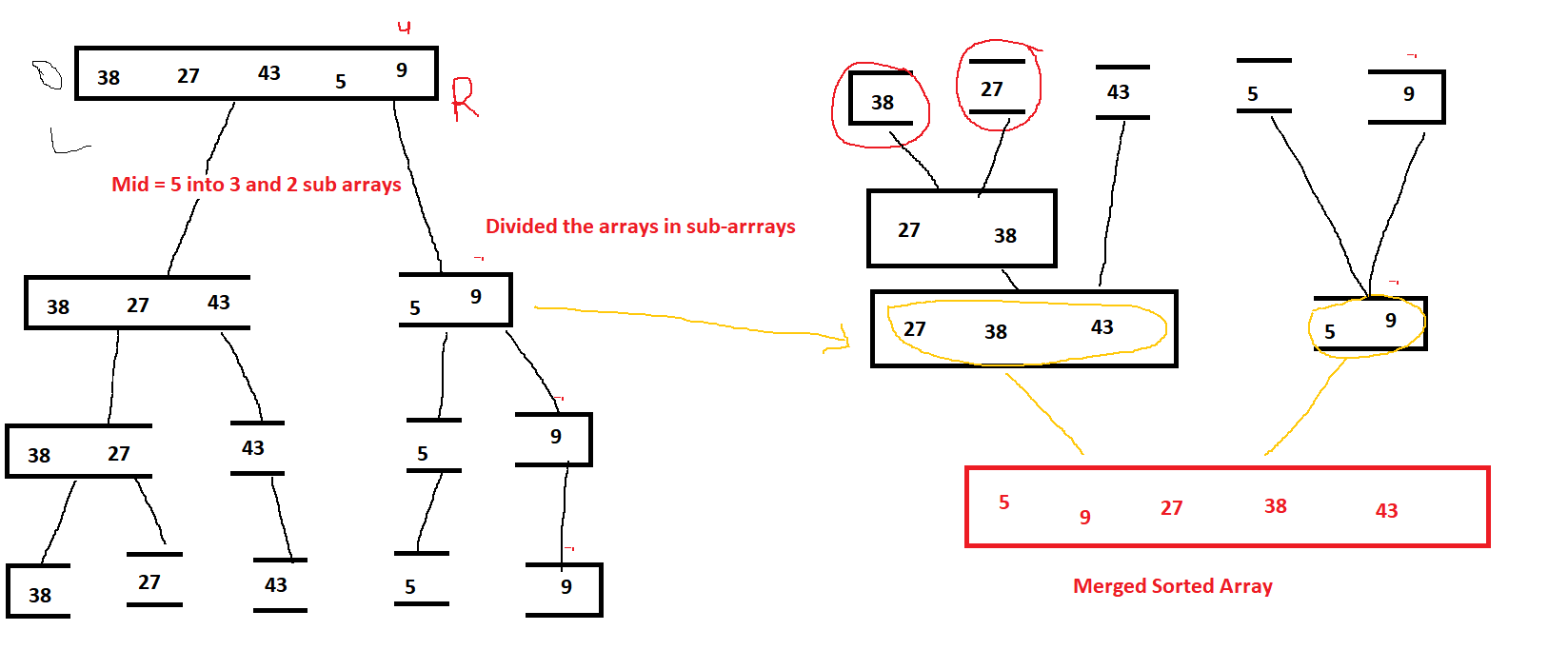
**Insertion Sort**

* Splits in sorted and unsorted array part
* Pick values from unsorted array and place in the correct position in sorted part
* Much simpler and less swaps in comparison to bubble sort
* Not suitable for large data sets
* **Complexity:** Worst O(n^2) Best O(n)



**Merge Sort**

* Divide and conquer technique
* Split in half
* When sorted then merge is applied
* Complexity: O(N log(N))
* It needs some additional space for temporary arrays



**Quick Sort**

* Divide and conquer algo
* Pick a pivot element
* Partition the array around the picked pivot element
* Best and Avg complexity: O(n log n)
* Worst complexity: O(n^2)

