Coding notes

▼ Bubble Sort

Bubble Sort is the simplest **sorting algorithm** that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

We run the loop from 0 to (n-2) because we wont be able to compare the last element of array (n-1) as it will give “ArrayindexOutOfBounds”.

https://www.geeksforgeeks.org/bubble-sort/

▼ Selection Sort

Selection sort *is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.*

e.g. Find the minimum element in arr[0…n-1] and place it at beginning. Repeat this for next adjacent elements to get a Sorted array.

▼ Interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body.

It is used to achieve abstraction and multiple inheritance in Java.

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects. An interface cannot contain a constructor

- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `public` and `abstract`.
- Interface attributes are by default `public`, `static` and `final`
- Interface can extend other Intefaces.

▼ Polymorphism

Polymorphism in Java is basically a concept by which we can perform a *single action in different ways*.
Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

There are two types of polymorphism in Java: *Compile-time polymorphism* and *Runtime polymorphism*.
We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism.

Method overloading :

method overloading occurs when multiple methods can have the same name with different parameters.

```
static int plusMethod(int x, int y) {
  return x + y;
}

static double plusMethod(double x, double y) {
  return x + y;
}

public static void main(String[] args) {
  int myNum1 = plusMethod(8, 5);
  double myNum2 = plusMethod(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

Method overriding:

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.
(same name, same arguements)

```
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){
     System.out.println("Vehicle is running");
  }
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){
        System.out.println("Bike is running safely");
    }

  public static void main(String args[]){

    Vehicle ob = new Vehicle();//creating object
  ob.run(); //prints run from Vehicles class

  Bike2 obj = new Bike2();//creating object
  obj.run();//Overrides the run method of Vehicle class
  }
}
```

▼ Encapsulation

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines. The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users

To create a fully encapsulated class in Java :

- declare class variables/attributes as `private`
- provide public **getter** and **setter** methods to access and update the value of a `private` variable

By providing only a *setter* or *getter* method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

It is a way to achieve ***data hiding*** in Java because other class will not be able to access the data through the private data members.

```
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
public class Main {
  public static void main(String[] args) {

    Person myObj = new Person();
    myObj.setName("John"); // Set the value of the name va
riable to "John"
```

```
        System.out.println(myObj.getName());
    }
}
// Outputs "John"
```

▼ Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.
A class which is declared with the
abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

```
public class Abstraction2 {

    public static void main(String[] args) {

        Honda obj = new Honda();
        obj.run();
        obj.changeGear();

    }

}
```

```
//Example of an abstract class that has abstract, non-abst
ract methods and Constructor
abstract class Car {

    Car() {                                             //Cons
tructor
        System.out.println("bike is created");
    }

    abstract void run();                                //abst
ract method

    void changeGear() {                                 //non-
abstract method
        System.out.println("gear changed");
    }
}

//Creating a Child class which inherits Abstract class
class Honda extends Car {

    void run() {
        System.out.println("running safely..");
    }
}
```