Interfaces

Any Service Requirement Specification is called an Interface.

Any Client-Service Provider communication requirement is called an Interface.

It acts as a Blueprint of classes because it specifies only what a class must do, but not how to do that.

Meaning : - It can be considered as 100% Abstract class(till JDK 1.8).

All methods in an Interface are by default public and abstract.

So this is clear that each class which implements the interface will define its own implementation of all those public and abstract methods present in the Interface.

Diagram of Blueprints:-
Interface --> Classes --> Objects -->

Use case of Interfaces Concept:-

1. It provides Service Requirement Specification. Meaning is that, if I have to connect to Oracle Database or PostgreSQL database, these databases dont provide separate classes and interfaces. They just provide separate implementations to the JDBC API.

   The term API means Application Programming Interface. So, Interface is something which provides us what a class must do so as to achieve a task, but not how to achieve it. HOW part is left on the Database Vendors. Like MYSQL provides its own implementation based on JDBC API.

2. To separate the method from getting defined from the inheritance hierarchy.

3. It is used in Data Abstraction.

Using an Interface: -
Syntax: -

```
access-specifier interface name{
        final static type-name1 varName1 = value1;
        final static type-name2 varName2 = value2;

        return-type method-name(parameter list);
}
```

This is just a basic structure of an interface. Beginning from Line 1, "access-specifier" can be either public, default or protected. If the interface is declared as public then other interfaces in the file must not be declared using "public" keyword and the file name must be matched with the interface name.

"interface" is a keyword which tells the compiler that an interface(blueprint of class) begins.it is the required keyword
which is used to have an interface.

"name" is the name of the interface. It CAN BE ANY VALID IDENTIFIER NAME.

Note that all variables in an Interface in Java is by default a constant. That means, it is declared by using the "final" and "static"
keywords. So, "varName1" and "varName2" is declared using the "final" and "static" keywords.

The "type-name1" means the datatype of the variable "varName1". Eg: - int x = 10;
In the above example, "int" is a datatype, "x" is the variable name and "10" is a value assigned to the variable x.

"varName1" refers to the variable name which, again, can be any valid identifier.

Coming to the end of interface syntax, we have a method. This method is very special for and to Java. This is because
this method has no body. Which means that this method is "abstract". Abstract Methods have no bodies. Their implementation
is to be provided by the Service Provider or the Programmer(in this case). All Abstract Methods end with a ';' punctuator to
denote that it is an abstract method.

The "return-type" of that method means the type of data you would return as an output from the method.
"method-name" is the name of the method which can again be any valid Java Identifier. And the parameter list can be any set of valid parameter
list.

The ';' is a must in case of interfaces. Till JDK 7, the above syntax was must...But from JDK 8 and above, we can even have default and static methods...
But they are mostly unused in production line.

Example of an Interface in Java: -

Let's say that we need to find the Volume and Total Surface Area of 3D figures. Each figure would have different formulaes and
different implementations. Hence its better to create just a blueprint of 3D geometrical figure. If we go on to inherit an abstract class,
the method can virtually lose its implementation. Hence, its better to use Interfaces concept because it separates method definition

from the class hierarchy.

```
interface Three_Dimensions{
        public double volume();
        public double total_sur_ar();
}

public class Cube implements Three_Dimensions{
        private int length;
        @Override
        public double volume() {
                return Math.pow(length, 3);
        }

        @Override
        public double total_sur_ar() {
                return 6*Math.pow(length, 2);
        }

        public static void main(String[] args) {
                Cube cube = new Cube();
                cube.length=20;
                System.out.println(cube.volume());
                System.out.println(cube.total_sur_ar());
        }
}

//Interface Methods can only be "public".

/*
        * Why can't be interface methods defined by other access specifiers: ?
        * "private" : Can be done from JDK 8. Will learn how to use it at the end.
        * "protected" : Interface Methods cannot be protected by default.
        * "default" : Can be used since JDK 9.
        * "static" : Can be used since JDK 9.
*/
```

Here, the class Cube implements the Three_Dimensions because it has 3 dimensions and so it can have a volume and total surface area.
Hence the interface "Three_Dimensions" is blueprint of any 3d Geometrical shape. And also, each geometrical shape has its own volume and total
surface area. So, Cube class acts as a Service Provider and gives the implementation for its volume and total surface area.

Another example can be using a Audi A3 Car:

All Cars have functionalities like starting of engine, stopping of engine, opening and closing of door and different horn sound. These functionalities are available to all cars by default.

So lets say, Mr.Venky ordered for a Car Audi A3 from a Company X. Mr.Venky is the Client here and the Company is the Provider.

The Company X already has a blueprint of what all cars must contain. Now they will provide some special implementations of theirs in the car by implementing the features that all cars generally have. If they don't implement the start and stop engine methods, the Audi A3 car will not even start. Mr.Venky be like : "What a shit a car is this!!!! It isnt also starting up."

So each time, a client reaches out to the Company X for a Audi Car, the Company has a blueprint ready. It just has to implement some tasks into it which would make the Car, a "CAR" (no puns here). Now, each Car must have some methods that accelerate and slow the Car. Even, a Car has a speedometer which displays the current speed of the Car. And, doesn't each car have a music player...? Yeah, they do have. So these things are required to be present in the Class.
Hence, it means that Audi A3 implements and adds its implementations in the methods of the interface "Drivable" and extends all the Features that a Car has. But any Car does not have a concrete implementation. As you might have guessed correctly, Car is an Abstract Class. Hence, AUDI A3 being an Car, must extend the features of its parent class namely "CAR". Hence it also performs Inheritance.

```
public interface Driveable {

        //Abstract Methods
        public void start();
        public void stop();
        public void doorOpens();
        public void doorCloses();
        public void hornSound();
}
```

As described earlier, all cars can be described with an adjective "Driveable". For example, Mr.Venky is filled with sweetness. This means that Mr.Venky can be described by an adjective "Sweetness". Similarly, Cars can be driven hence they are "Driveable".

Now I want to programmatically say that Mr.Venky's Car is "Driveable". Otherwise, Mr.Venky cannot even start the Car.

Now, the Company X wants that the Mr.Venky's AUDI A3 programmatically becomes a Car. So it would ensure that AUDI A3 becomes a Car.

The Company X is very clever. It creates a partial implementation of everything that a Car consists of. They plan to edit that partial implementation. And even, successfully execute the plan.

```
public abstract class Car {

        //Abstract or Unimplemented Methods.
        public abstract void accelerate(int acc_speed);
        public abstract void slow(int retar_speed);
        public abstract int currentSpeed();

        //Concrete Method.
        public void playMusic() {
                System.out.println("Music Playing: See you Again Wiz Khalifa");
        }
}
```

The only thing common in any car is a music player. Otherwise, each car have their own rate of acceleration and retardation. Hence, now AUDI A3 "IS-A" Car ("IS-A" relationship is a "Child-Parent" relationship explained in the Inheritance Portion).

Now, successfully Company X can add the unique properties of a "CAR" and implement all those functionalities that makes a "CAR" drivable.

```
public class AudiA3 extends Car implements Driveable {
        int speed;
        AudiA3(int speed){
                this.speed = speed;
        }

        //All the Below Implemented Methods are now Concrete Methods.

        @Override
        public void start() {
                System.out.println("Engine Started of AudiA3");
        }

        @Override
        public void stop() {
                System.out.println("Engine Stopped! AudiA3 Resting");
        }

        @Override
        public void doorOpens() {
                System.out.println("Opens Upward!");
        }
        @Override
        public void doorCloses() {
                System.out.println("Closes Downward!");
        }

        @Override
        public void hornSound() {
                System.out.println("Beep Boop!");
        }

        @Override
        public void accelerate(int acc_speed) {
                this.speed+=acc_speed;
        }

        @Override
        public void slow(int retar_speed) {
                this.speed-=retar_speed;
        }

        public int currentSpeed() {
                return this.speed;
        }

        public static void main(String[] args) {
                AudiA3 venkyCar = new AudiA3(0);

                venkyCar.doorOpens();
                venkyCar.start();
                venkyCar.accelerate(20);
                venkyCar.slow(5);
                System.out.println(venkyCar.currentSpeed());
                venkyCar.stop();
        }
}
```

Now this AUDI A3 car will be given to Mr.Venky is finally complete. Now take money and pay respect and give car keys to Venky.

mission passed!
respect +