Topic: Interfaces.
OOP concepts involved: Inheritance, Abstract Classes, Class Instantiation and Declaration, Class Members, Polymorphism, Constructor Methods, Class Modifiers, Interfaces, Getters and Setters.
Programming generic concepts involved: Access Modifiers, Functions, Primitive Variables, Variable Declaration, Function Parameters.

➢ Theoric introduction

Interfaces

In Java, an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods and nested types. Method bodies exist only for default and static methods.

An important factor is that Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces. Unless a class that implement an interface is abstract, all the methods of the interface must be defined in the class.

Declaring an interface is very similar to declaring a class:

```
[Access Modifier] interface [identifier]{ }
```

An interface may seem similar to an abstract class, but there are key differences between the two.

- ❏ All methods in an interface must be declared as abstract and public
- ❏ An interface cannot implement any method, all methods must be abstract
- ❏ An interface cannot be instanced
- ❏ A class can implement multiple interfaces, but it can only extend one class
- ❏ Classes are part of a hierarchy, while interfaces are not. Multiple classes can implement the same interface with no connection between them.

Implementing an Interface

After we have created our interface, we now need a way to implement it into a class. To do this we use the reserved keyword **implements** in the class header.

```
[Access Modifier] [Modifiers] class [identifier] implements [interface identifier]{ }
```

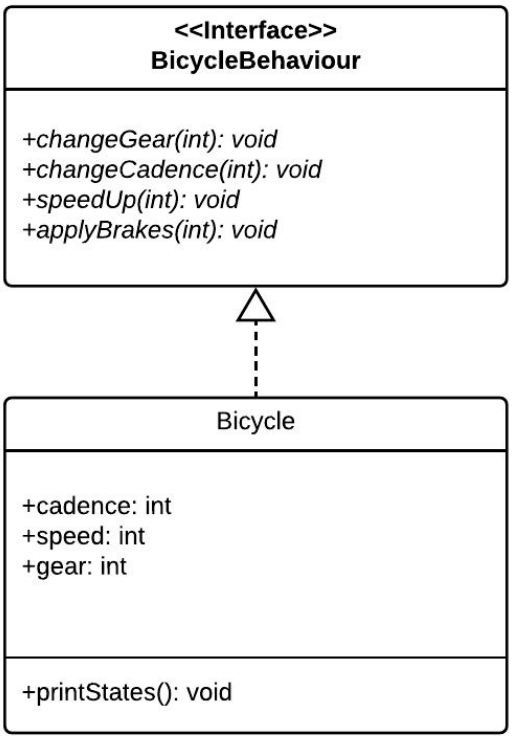
When implement an interface, we must declare all abstract methods inside the interface.

➢ Statement

For a bicycle simulation test, we need to create a simulation of the behaviour a bicycle should have.

Develop a program that creates an interface where the basic functions of a bicycle are: Changing cadence, Changing gear, speeding up and slowing down. Use and interface to do it, and create a class that implements that interface to test the functions.

➢ Class design (UML)



➢ Program Code

BicycleBehaviour.java

```
public interface BicycleBehaviour{

    //Declaring the interface abstract methods
    public void changeCadence(int newValue);

    public void changeGear(int newValue);

    public void speedUp(int increment);

    public void applyBrakes(int decrement);
}
```

Bicycle.java

```
public class Bicycle implements BicycleBehaviour {
       // Class attributes
       int cadence = 0;
       int speed = 0;
       int gear = 1;

        //Class Constructor
       public Bicycle(){}

        // The compiler will now require that methods: changeCadence, changeGear, speedUp,
        // and applyBrakes all be implemented. Compilation will fail if those methods are
        // missing from this class.

       public void changeCadence(int newValue) {
              this.cadence = newValue;
       }

       public void changeGear(int newValue) {
              this.gear = newValue;
       }

       public void speedUp(int increment) {
              this.speed = this.speed + increment;
       }

       public void applyBrakes(int decrement) {
              this.speed = this.speed - decrement;
       }
```

```
        public void printStates() {
                System.out.println("cadence:" + cadence);
                System.out.println(" speed:" + speed);
                System.out.println(" gear:" + gear);
        }
}
```

Main.java

```
import java.util.Scanner;
import java.lang.Integer;

public class Main {

        public static void main(String[] args) {
                // Declaring and instantiating a Bicycle Object
                Bicycle bb = new Bicycle();

                // Declaring the object to allow data reading predefining object System.in
               // for standard data input from the keyboard
                Scanner sc = new Scanner(System.in);

                // Capturing data through the console and assigning it
                System.out.println("What's the Bycicle's cadence?");
                bb.changeCadence(Integer.parseInt(sc.nextLine()));
                System.out.println("What's the Bycicle's gear?");
                bb.changeGear(Integer.parseInt(sc.nextLine()));
                System.out.println("What's the Bycicle's speed?");
                bb.speedUp(Integer.parseInt(sc.nextLine()));
                bb.printStates();

                sc.close(); // Closing data flow
        }
}
```

➢ Program execution

During the program execution, we add the values to the simulation to test the bicycle behaviour, getting the results of it at the end.

```
What's the Bycicle's cadence?
12
What's the Bycicle's gear?
12
What's the Bycicle's speed?
12
cadence:12
 speed:12
 gear:12
```

➢ Conclusions

Interfaces are a great tool in projects where multiple programmers are working at the same time. In this cases, interfaces work as kind of “contracts” that spells out how their software interacts. This way, all programmers should be able to write their code without knowledge of how other programmers wrote their code.