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.

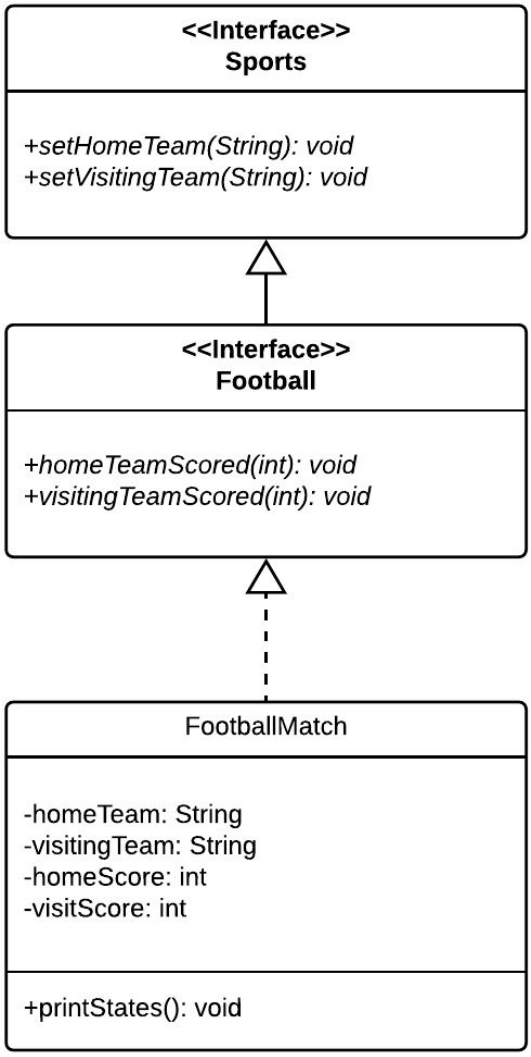
```
[Access Modifier] [Modifiers] class [identifier] implements [interface identifier]{ }
```

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

➢ **Statement**

Develop a program that implements a Football match with all its basic attributes, Home Team Score and Visiting Team Score. Also create a father interface for future uses for other Sports.

➢ **Class design (UML)**



➢ Program Code

Sports.java

```
public interface Sports {
	//Declaring the Interface abstract methods
	public void setHomeTeam(String name);
	public void setVisitingTeam(String name);

}
```

Football.java

```
public interface Football extends Sports {
	//An interface can also extend another interface, and just like classes, the
Subinterface inherits all of the father Interface abstract methods, so a class implementing
this subInterface must define all methods included in this and the father interface.
	//Declaring the Interface abstract methods
	public void homeTeamScored(int points);
	public void visitingTeamScored(int points);
}
```

FootballMatch.java

```
public class FootballMatch implements Football{

	private String homeTeam, visitingTeam;
	private int homeScore, visitScore;
	//Class Constructor
	public FootballMatch(){
	}

   // The compiler will now require that methods setHomeTeam, setVisitingTeam,
   // homeTeamScored and visitingTeamScored all be implemented.
   // Compilation will fail if those methods are missing from this class.

	public void setHomeTeam(String name){
			this.homeTeam = name;
	}
	public void setVisitingTeam(String name){
		this.visitingTeam = name;
	}
	public void homeTeamScored(int points){
		this.homeScore = points;
	}
	public void visitingTeamScored(int points){
		this.visitScore = points;
```

```
	}

	public void printStates(){
		System.out.println("\n Home Team: " + this.homeTeam + " - " +
this.homeScore);
		System.out.println("Visiting Team: " + this.visitingTeam + " - " +
this.visitScore);
	}
}
```

Main.java

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

public class Main {

	public static void main(String[] args) {
		// Declaring and instantiating a FootballMatch object
		FootballMatch fb = new FootballMatch();

		// 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 Home Team name?");
		fb.setHomeTeam(sc.nextLine());
		System.out.println("What's the Visiting Team name?");
		fb.setVisitingTeam(sc.nextLine());
		System.out.println("What's the Home Team score?");
		fb.homeTeamScored(Integer.parseInt(sc.nextLine()));
		System.out.println("What's the Visiting Team score?");
		fb.visitingTeamScored(Integer.parseInt(sc.nextLine()));
		fb.printStates();

		sc.close(); // Closing data flow

	}
}
```

➢ Program execution

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

```
What's the Home Team name?
XOLOS
What's the Visiting Team name?
PUEBLA
What's the Home Team score?
4
What's the Visiting Team score?
0

 Home Team: XOLOS - 4
Visiting Team: PUEBLA - 0
```

➢ 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.