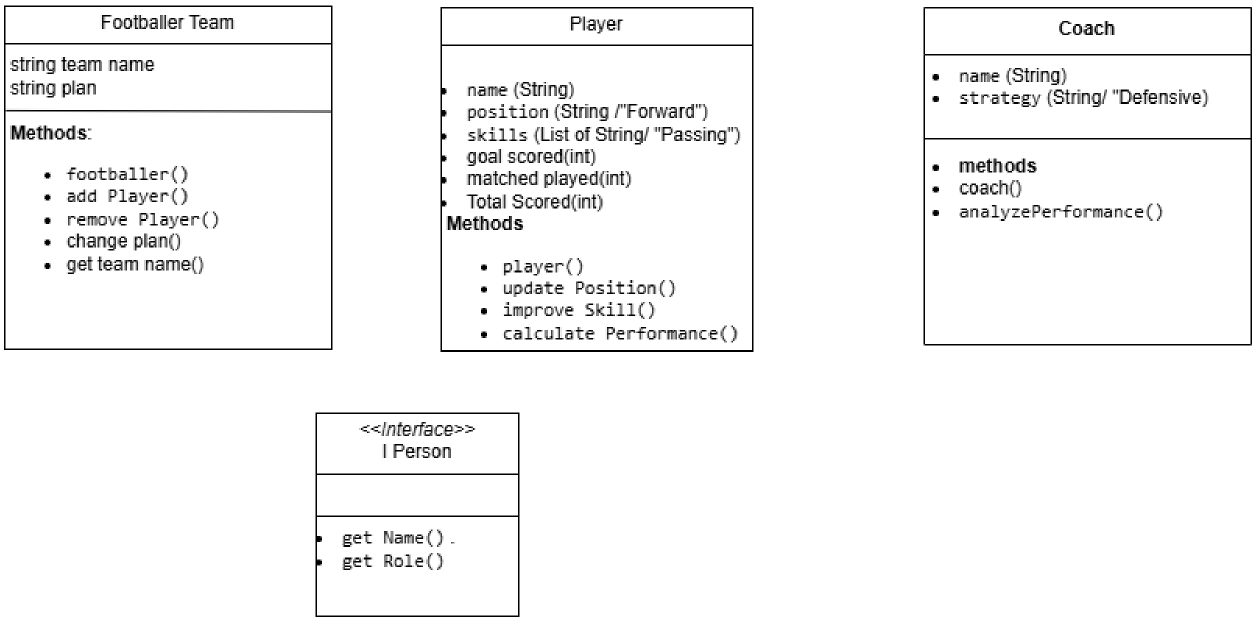
***Java*** is a programming paradigm that organizes software design around **objects**, which contain data (attributes) and methods (functions). Java is a widely-used OOP language because it provides a clear structure for programs and promotes code reuse, scalability, and maintainability.

## Use java lanaguge programming to implement this football system

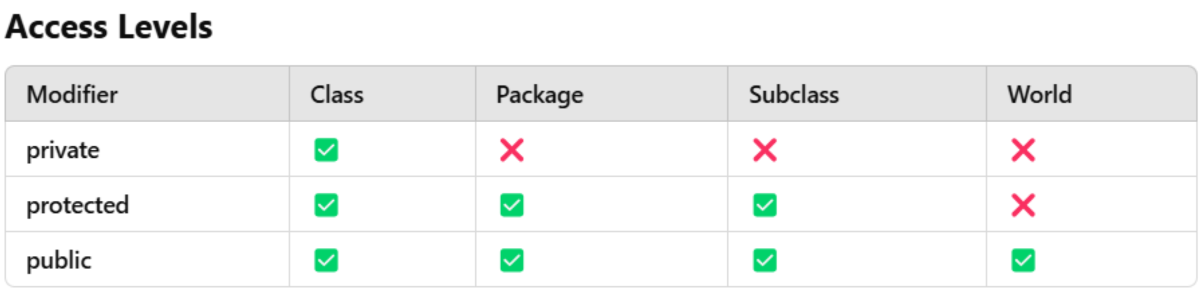
## ULM diagram



✓ ***OOP Principles in Java***

### Encapsulation

- Bundling data (fields) and methods (functions) together into a single unit (class).
- Access to data is controlled through access modifiers (`private`, `protected`, `public`).



**Access Levels**

| Modifier | Class | Package | Subclass | World |
|---|---|---|---|---|
| private | ✓ | ✗ | ✗ | ✗ |
| protected | ✓ | ✓ | ✓ | ✗ |
| public | ✓ | ✓ | ✓ | ✓ |

For example to use encapsulation:

Class player

```java
import java.util.List;

// make public class player
public class Player implements IPerson {
   public String name;
   public String position;
   public List<String> skills ;
   private int goalsscored;
   private int matchesplayd;



    // create set and get function for private attributes
    public int getGoalsscored() {
        return goalsscored;
    }

    public void setGoalsscored(int goalsscored) {
        this.goalsscored = goalsscored;
    }

    public int getMatchesplayd() {
        return matchesplayd;
    }

    public void setMatchesplayd(int matchesplayd) {
        this.matchesplayd = matchesplayd;
    }



    // contractor for class player
    public Player(String name, String position, List<String> skills ) {
        this.name=name;
        this.position=position;
        this.skills=skills;

    }
    // create update methods to change player position
   public void Update_position(String newposition){
      System.out.println(name+ " "+ "update to"+newposition);

   }
     // create improve skill methods to add new sill for player
     public void improveskill( String newskill){
         skills.add("shooting");
         skills.add("passing");
         // check if skill list contain new skill
          if (skills.contains(newskill)) {
              System.out.println("Skill already exists.");
          }
          else{
              System.out.println(name+ " "+ " add new skill"+(newskill) );}
     }
     // create methods to calculate performances scored
    public int calculate_performance(int goalsscored, int matchesplayd){
        return  (goalsscored * 4)  + matchesplayd;
    };


    // Override methods from interface Person

    @Override
    public String get_name( ) {
        return name;
    }

    @Override
    public String get_role() {
        return "Player";
    }


}
```

## Inheritance

- A mechanism where a new class (subclass) derives properties and behaviors from an existing class (superclass).
- Promotes code reuse.

```
• import java.util.List;

  public class Footballer extends Player {
      public String teamName;
      public String plan;

       // Create fun to return footballer name
      public String getTeamName(String teamName ) {
          return teamName;
      }
      // create constructor for football team class which extends from
  player
      Footballer(String name, String position, List<String> skills,
  String teamName ,String plan){
         // call constructor from super class Player
          super( name,position, skills);
           this.teamName=teamName;
           this.plan=plan;
      }


      // create method to add player to team
      public  void add_player( Player player3) {
            System.out.println(" Add player "+ player3.name);

      }
      // create method to remove player from team
      public  void remove_player(Player player3) {
          System.out.println(" remove "+ player3.name);
      }
      public  void change_plan(String Newplan){
          System.out.println( " the new plan" + Newplan);

      }



  }
```

## Polymorphism

- The ability of a single interface to represent different underlying forms (data types).
- Achieved through **method overriding** and **method overloading**.

```
• // make interface class every class inheritance from it must implement
  the methods
  public interface IPerson {

      String get_name();
      String get_role();
  }
```

```
•

public class Coach  extends Footballer implements IPerson{
        public String name;
        public String Strategy;


    // create contractor for class coach
    public Coach(String name, String position, List<String> skills,
                  String teamName,String plan){
        super(name,position,skills,teamName,plan);
        this.name=name;
        this.Strategy=Strategy;


    }

    // create method to analyze performance
    public void Analzeperfome() {
        System.out.println("Coach " + name + " is planning the " +
                Strategy + " strategy for the team: " + getTeamName("liver
pool"));
    }

    // Override methods from interface Person
    @Override
    public String get_name() {
        return name;
    }

    @Override
    public String get_role() {
        return "Coach";
    }
}
```

**Abstraction**

- Hiding implementation details and showing only the essential features of the object.
- Achieved through **abstract classes** and **interfaces**.

## Key OOP Concepts in Practice

- **Classes and Objects:**

```
•  import java.util.ArrayList;
   import java.util.List;

   public class Main {
       public static void main(String[] args) {
           List<String> skills = new ArrayList<>();
           // create instance from class player
           Player player1=new Player("Mo Salah","right wing",skills);
           player1.improveskill(" attacking");
           player1.Update_position("left wing");
           // call two private attribute goalscorer ,matches played
           player1.setGoalsscored(10);
           player1.setMatchesplayd(5);
           player1.setTotalscored(0);

           // handle try catch in main class to catch error
```

```
        try {
            System.out.println(  "the player1 performance
"+player1.calculate_performance(player1.getGoalsscored(),

player1.getMatchesplayd(),player1.getTotalscored()));
        }catch (ArithmeticException e){
            {
                System.out.println("Exception caught: " +
e.getMessage());
            }
        }

        System.out.println(player1.get_name());
        System.out.println(player1.get_role());
        // create instance from class coach
        Coach coach1=new Coach("Jürgen Kl
opp","trainer",skills,"liverpool","High Pressing");
        coach1.Analzeperfome();
        System.out.println(coach1.get_name());
        System.out.println( coach1.get_role());
        // create instance from class footballer
        Footballer team=new Footballer("liver
pool","central",skills,"liver pool" ,"High Pressing");
        Player player3=new Player("Virgil van Disk","left
wing",skills);
        System.out.println(team.getTeamName("liver pool"));
        team.remove_player(player1);
        team.change_plan("attacking");

}
}
```

## Benefits of OOP in Java

1. **Modularity:** Code is organized into classes and objects, making it modular.
2. **Code Reuse:** Inheritance allows reuse of existing code.
3. **Flexibility:** Polymorphism provides flexibility and ease of maintenance.
4. **Security:** Encapsulation hides data and implementation details.

## Common Exceptions

- `ArithmeticException`: Thrown when a number is divided by zero.
- `NullPointerException`: Thrown when a null object is accessed.
- `ArrayIndexOutOfBoundsException`: Thrown when accessing an invalid array index.
- `IOException`: Thrown during input/output operations.
- `FileNotFoundException`: Thrown when a file is not found.

Execute code :

```
"C:\Program Files\Java\jdk-23\bin\java.exe" --enable-preview "-javaagent:C:\Program
Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.4\lib\idea_rt.jar=58454:C:\Program
Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.4\bin" -Dfile.encoding=UTF-8 -
```

```
Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "D:\java tasks\java
project football\out\production\java project football" Main

    Mo Salah  add new skill attacking
    Mo Salah update toleft wing
    Exception caught: / by zero
    Mo Salah
    Player
    Coach Jürgen Kl opp is planning the null strategy for the team: liver pool
    Jürgen Kl opp
    Coach
    liver pool
     remove Mo Salah
     the new plan attacking

    Process finished with exit code 0
```