CMM024 Object Oriented Programming

Practical Session: 8

This lab is dedicated to Object Orientated Programming; focusing on inheritance. You are going to create your own classes and test them

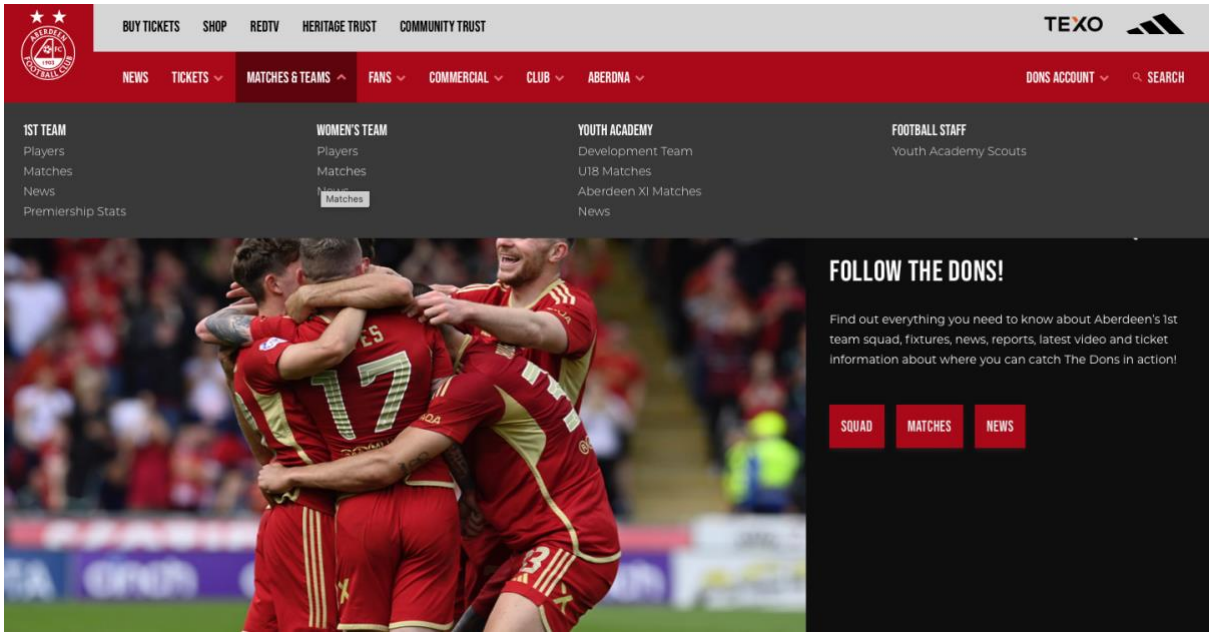
1. Player Class explanations (worked Example)

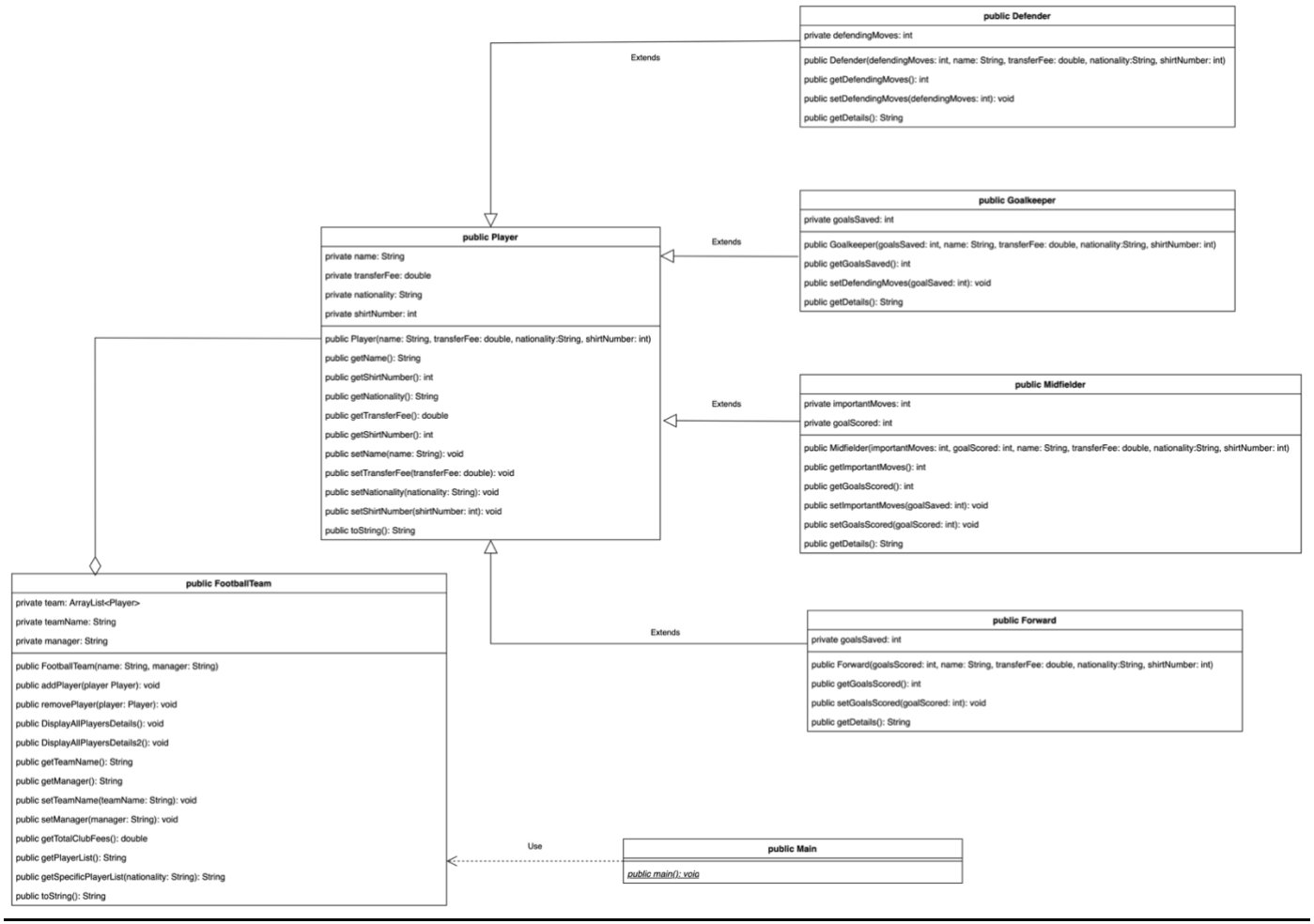
Please read this example and implement the code. This is needed when coding the second part of this lab. To test the last class (FootballTeam) you will have to download the Main.java file.

Brief:

This exercise is about modelling a football team as a set of classes using inheritance.

Inspiration:




public Defender
private defendingMoves: int
public Defender(defendingMoves: int, name: String, transferFee: double, nationality:String, shirtNumber: int)
public getDefendingMoves(): int
public setDefendingMoves(defendingMoves: int): void
public getDetails(): String
Extends
public Goalkeeper
private goalsSaved: int
public Goalkeeper(goalsSaved: int, name: String, transferFee: double, nationality:String, shirtNumber: int)
public getGoalsSaved(): int
public setDefendingMoves(goalSaved: int): void
public getDetails(): String
Extends
public Player
private name: String
private transferFee: double
private nationality: String
private shirtNumber: int
public Player(name: String, transferFee: double, nationality:String, shirtNumber: int)
public getName(): String
public getShirtNumber(): int
public getNationality(): String
public getTransferFee(): double
public getShirtNumber(): int
public setName(name: String): void
public setTransferFee(transferFee: double): void
public setNationality(nationality: String): void
public setShirtNumber(shirtNumber: int): void
public toString(): String
Extends
public Midfielder
private importantMoves: int
private goalScored: int
public Midfielder(importantMoves: int, goalScored: int, name: String, transferFee: double, nationality:String, shirtNumber: int)
public getImportantMoves(): int
public getGoalsScored(): int
public setImportantMoves(goalSaved: int): void
public setGoalsScored(goalScored: int): void
public getDetails(): String
public FootballTeam
private team: ArrayList<Player>
private teamName: String
private manager: String
public FootballTeam(name: String, manager: String)
public addPlayer(player Player): void
public removePlayer(player: Player): void
public DisplayAllPlayersDetails(): void
public DisplayAllPlayersDetails2(): void
public getTeamName(): String
public getManager(): String
public setTeamName(teamName: String): void
public setManager(manager: String): void
public getTotalClubFees(): double
public getPlayerList(): String
public getSpecificPlayerList(nationality: String): String
public toString(): String
Extends
public Forward
private goalsSaved: int
public Forward(goalsScored: int, name: String, transferFee: double, nationality:String, shirtNumber: int)
public getGoalsScored(): int
public setGoalsScored(goalScored: int): void
public getDetails(): String
Use
public Main
public main(): voig

Attributes for a Football Player

There two kinds of attributes often for a football player. The generic attributes relating to the person being part of team. Usually

- Name
- Age
- Transfer fee
- Nationality
- ~~Other clubs palyed~~
- ~~On loan or active~~

The other information relates to the role that player has on the field

- Goalkeepers
  - Goals Saved
- Defenders
  - Attacking moves
  - Defending Moves
  - ~~Areal Moves~~
  - ~~Etc.~~
- Midfielders (Note that a midfielder is able to score too)
  - Important moves
  - Goal Scored
  - ~~Etc.~~
- Forwards
  - Goal Scored
  - ~~Etc.~~

In order to model this, a superclass Player must be created to store the generic information, and Goalkeeper, Defender, Midfielder and Forward classes must be created to store the more specific information related to the position on the ground.

Player class Attributes Datatype mapping

Since we are modelling a generic Player in Java, we need to map each attribute with a suitable data type.

Attribute	Data Type
Name	String
Transfer Fee	double
Nationality	String
Shirt Number	int

Goalkeeper class Attribute Datatype mapping

Attribute	Data Type
Goals Saved	int

Defender class Attribute Datatype mapping

Attribute	Data Type
Defending moves	int

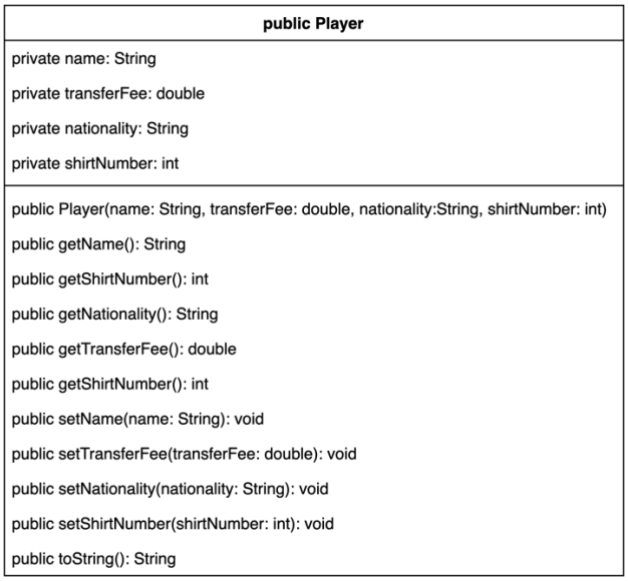
Midfielder class Attributes Datatype mapping

Attribute	Data Type
Important moves	int
Goals scored	int

Forward class Attribute Datatype mapping

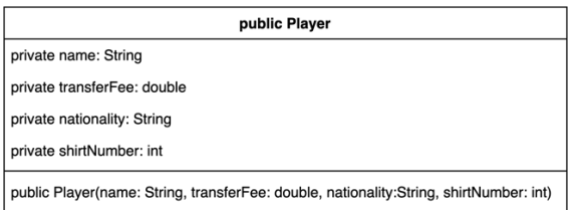
Attribute	Data Type
Goals scored	int

UML: modelling the Player class attributes (DrawIO)



UML: Creating a Constructor for the Player class

To create an Object instance from the Player class, we must create a constructor method that will pass enough information (in the parameters) to initialise all the attributes that must be set.



Creating Setter Methods for the Player class

Create setter method allow us to set or change the value of the attributes. Imagine that a mistake was made, how do you amend the existing information.

Attributes	Return type	Visibility	Method name	Method signature
Private name: String	void	public	setName	public void setName(String: name)
Private transferFee: double	void	public	setTransferFee	public void setTransferFee(double transferFee)
Private nationality: String	void	public	setNationality	public void setNationality (String nationality)
Private shirtNumber: int	void	public	setShirtNumber	public void setShirtNumber(int ShirtNumber)

Creating Getter Methods

All the attributes are private, which means that are invisible to other objects etc. However, we need to get access to these values, so we create public methods that return these values.

Attributes	Return type	Visibility	Method name	Method signature
Private name: String	String	public	getName	public void getName()
Private transferFee: double	double	public	getTransferFee	public void getTransferFee()
Private nationality: String	String	public	getNationality	public void getNationality ()
Private shirtNumber: int	int	public	getShirtNumber	public void getShirtNumber()

Creating toString() method

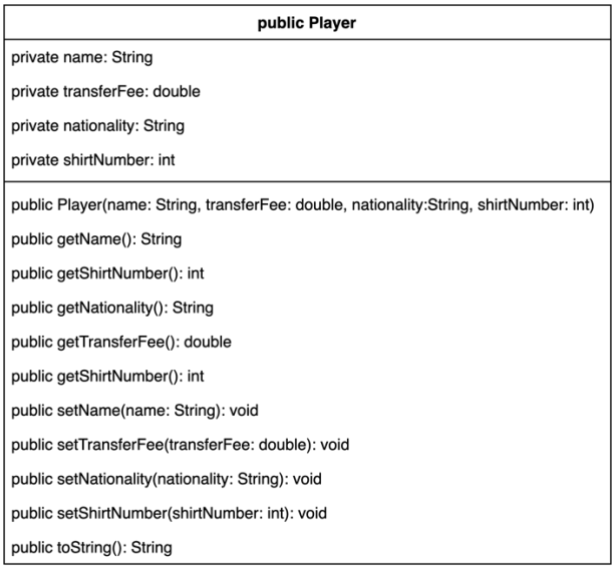
Often, we need to return the data information held in the attributes into a formatted string. This is also very useful for checking if the coding is right….when testing!

Attributes	Return type	Visibility	Method name	Method signature
Private name: String	String	public	toString	Public String toString()
Private transferFee: double				
Private nationality: String				
Private shirtNumber: int				

2. Player Class coding (worked Example)

UML: modelling the Player class functionality (DrawIO)

In terms of UML, this is the full UML diagram for the Player class. By perusing it, you can code the skeleton of that class.



Coding the Player Class Attributes

We first create a Player class, and then we declared all the object attributes we need. We thought about what data type to use in an early process.

1) Create a **Player.java** java file

```
public class Player {

    // Fields for Player superclass
    private String name;
    private double transferFee;
    private String nationality;
    private int shirtNumber;
```

Coding the Player Class constructor

As we already said above, the constructor is a special method to create an object for a particular class, here it is the Player class. Since all the attributes are necessary and need to be set, we must use parameter that will hold the values to initialise them.
Note: to differentiate between the parameter name and the object attribute name we use THIS. Which refers to the class attribute rather the parameters. You can use different names for the parameters. For example: the parameter String name can be String theName, thus avoiding issues etc.

```
// Full Specialised Constructor
public Player(String name, double transferFee, String nationality, int shirtNumber) {
    this.name = name;
    this.transferFee = transferFee;
    this.nationality = nationality;
    this.shirtNumber = shirtNumber;
}
```

Coding the Player Class Setter Methods

Setter methods have a parameter whose values is used to set the object attribute. i.e., the value in the parameter is used to set the value of the corresponding attribute. Of course, the data type for the parameter must be the same as the one used for the object attribute!

```java
// Setter Methods
public void setName(String name) {
    this.name = name;
}

public void setTransferFee(double transferFee) {
    this.transferFee = transferFee;
}

public void setNationality(String nationality) {
    this.nationality = nationality;
}

public void setShirtNumber(int shortNumber){
    this.shirtNumber = shortNumber;
}
```

```java
// Setter Methods
public void setName(String name) {…
public void setTransferFee(double transferFee) {…
public void setNationality(String nationality) {…
public void setShirtNumber(int shortNumber){…
```

Coding the Player Class Getter Methods

There will be no point of modelling something if there are no mechanisms to use the information and data in that object. Getter methods are returning the attributes' values.

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

public int getShirtNumber() {
    return shirtNumber;
}

public String getNationality() {
    return nationality;
}

public double getTransferFee() {
    return transferFee;
}

public int getShortNumber(){
    return shirtNumber;
}
```

```java
// Getter Methods
public String getName() {…
public int getShirtNumber() {…
public String getNationality() {…
public double getTransferFee() {…
public int getShortNumber(){…
```

Coding the Player class toString() Method

The central piece is a String where the value of the attributes are appended to it. We have used this before in other labs!

```java
// toString Method
public String toString() {
    return "Player{" + "Name = " + getName() + ", Position = " + getShirtNumber() +
    ", TransferFee = £" + (int) getTransferFee();
}
```

Testing the Player Class using its main(…) Method

We can add an extra static main method to test the class you have just coded. Note the keyword new. You have used it using the Scanner class in earlier labs!

```
Run | Debug
public static void main(String[] args) {
    Player player = new Player(name:"Rhys Williams", transferFee:416000, nationality:"England", shirtNumber:18);
    System.out.println(player);
}
```

```
Player{Name = Rhys Williams, Position = 18, TransferFee = £416000}
jean-claude@MacBook-Pro-2 footballteamproject %
```

3. Goalkeeper Class coding (worked Example)

UML: modelling the Goalkeeper class functionality (DrawIO)

public Goalkeeper
private goalsSaved: int
public Goalkeeper(goalsSaved: int, name: String, transferFee: double, nationality:String, shirtNumber: int) public getGoalsSaved(): int public setDefendingMoves(goalSaved: int): void public getDetails(): String

Coding the Goalkeeper Class Attributes

We first create a Goalkeeper class, and then we declared the only object attribute that is needed for this class. We thought about what data type to use in an early process.

1) Create a Goalkeeper.java file

Do not forget to code "extends Player" to tell Java that his class is a subclass for Player class!

```
public class Goalkeeper extends Player{

    //Field for Goalkeeper subclass
    private int goalsSaved;

    //Specialised Constructor
    public Goalkeeper(int goalsSaved, String name, double transferFee, String nationality, int shirtNumber){
```

Coding the Goalkeeper Class constructor

As we already said above, the constructor is a special method to create an object for a particular class, here it is the Goalkeeper class. Since all the attributes are necessary and need to be set, we must use parameters that will hold the values to initialise them.

This class is not a super class (i.e., concrete class) like the Player class we just coded. Its constructor must have enough parameters to initialise the goalsSaved object attribute as well as having all the necessary parameters to initialise the super class using the "super(name, transferFee, nationality, shirtNumber);" command. Hence it needs 5 parameters.

```
7       //Specialised Constructor
8       public Goalkeeper(int goalsSaved, String name, double transferFee, String nationality, int shirtNumber){
9
10          super(name, transferFee, nationality,shirtNumber );
11
12          this.goalsSaved = goalsSaved;
13      }
```

Coding the Goalkeeper Class Setter Method

Setter methods have a parameter whose values is used to set the object attribute. i.e., the value in the parameter is used to set the value of the corresponding attribute. Of course, the data type for the parameter must be the same as the one used for the object attribute!

```
//Setter Method
public void setGoalsSaved(int goalsSaved) {
    this.goalsSaved = goalsSaved;
}
```

Coding the Goalkeeper Class Getter Method

There will be no point of modelling something if there are no mechanisms to use the information and data in that object. Getter methods are returning the attributes' values.

```
//Getter Method
public int getGoalsSaved() {
    return goalsSaved;
}
```

Coding the Goalkeeper class other Method

The central piece is a String where the value of the attributes are appended to it. (just like the toString we did for the Player class)..

```
// toString Method
public String toString() {
    return "Player{" + "Name = " + getName() + ", Position = " + getShirtNumber() +
    ", TransferFee = £" + (int) getTransferFee();
}
```

Testing the Goalkeeper Class using its main(...) Method

You can copy the code below into the Goalkeeper class and then run it

```
public static void main(String[] args) {
    Goalkeeper goalkeeper24 = new Goalkeeper(72, "KELLE ROOS", 208000, "Netherlands", 24);
    System.out.println(goalkeeper24.getDetails());
    Goalkeeper goalkeeper25 = new Goalkeeper(0, "TOM RITCHIE", 36920, "Scotland", 25);
    System.out.println(goalkeeper25.getDetails());
    Goalkeeper goalkeeper31 = new Goalkeeper(51, "ROSS DOOHAN", 140400, "Scotland", 31);
    System.out.println(goalkeeper31.getDetails());
}
```

```
rage/7b69597119f8403098262e284f63b919/redhat.java/jdt_ws/footballteamproject_d9f56b58/bin Goalkeeper
Player{Name = KELLE ROOS, Position = 24, TransferFee = £208000} Goalkeeper{ Goal Saved = 72}
Player{Name = TOM RITCHIE, Position = 25, TransferFee = £36920} Goalkeeper{ Goal Saved = 0}
Player{Name = ROSS DOOHAN, Position = 31, TransferFee = £140400} Goalkeeper{ Goal Saved = 51}
jean-claude@MacBook-Pro-2 footballteamproject %
```

4. Defender Class coding (worked Example)

UML: modelling the Defender class functionality (DrawIO)

public Defender
private defendingMoves: int
public Defender(defendingMoves: int, name: String, transferFee: double, nationality:String, shirtNumber: int) public getDefendingMoves(): int public setDefendingMoves(defendingMoves: int): void public getDetails(): String

Coding the Defender Class Attributes

We first create a Defender class, and then we declared the only object attribute that is needed for this class. We thought about what data type to use in an early process.

1) Create a Defender.java file

Do not forget to code "extends Player" to tell Java that his class is a subclass for Player class!

```
public class Defender extends Player {

    // Field for Defender subclass
    // this is the attacking or defending moves added together
    private int defendingMoves;

    public Defender(int defendingMoves, String name, double transferFee, String nationality, int shirtNumber) {
```

Coding the Defender Class constructor

As we already said above, the constructor is a special method to create an object for a particular class, here it is the Defender class. Since all the attributes are necessary and need to be set, we must use parameters that will hold the values to initialise them.

This class is not a super class (i.e., concrete class) like the Player class we just coded. Its constructor must have enough parameters to initialise the "defendingMoves" object attribute as well as having all the necessary parameters to initialise the super class using the "super(name, transferFee, nationality, shirtNumber);" command. Hence it needs 5 parameters.

```
public Defender(int defendingMoves, String name, double transferFee, String nationality, int shirtNumber) {

    super(name, transferFee, nationality, shirtNumber);

    this.defendingMoves = defendingMoves;
}
```

Coding the Defender Class Setter Method

Code the setter method to retrieve the defendingMove attribute value in the same manner as you do for the Goalkeeper class.

Coding the Defender Class Getter Method

Code the setter method to set the defendingMove attribute value in the same manner as you do for the Goalkeeper class.

Coding the Defender class other Method

```
// Other methods
// return the player details (superclass) and adds specific details (fields)
public String getDetails() {
    return super.toString() + " Defender{" + " Defending moves = " + defendingMoves + '}';
}
```

Testing the Defender Class using its main(...) Method

You can copy the code below into the Defender class and then run it

```
public static void main(String[] args) {
Defender defender18 = new Defender(0, "RHYS WILLIAMS", 416000,"England",18);
System.out.println(defender18.getDetails());
Defender defender33 = new Defender(108, "SLOBODAN RUBEZIC", 135200,"Serbia",33);
System.out.println(defender33.getDetails());
Defender defender2 = new Defender(103, "NICKY DEVLIN", 150800, "Scotland",2);
System.out.println(defender2.getDetails());
Defender defender3 = new Defender(89, "JACK MACKENZIE", 109200,"Scotland",3);
System.out.println(defender3.getDetails());
Defender defender5 = new Defender(93, "RICHARD JENSEN", 140400,"Finland",5);
System.out.println(defender5.getDetails());
Defender defender6 = new Defender(115, "STEFAN GARTENMANN", 135200,"Denmark",6);
System.out.println(defender6.getDetails());
Defender defender15 = new Defender(98, "JAMES MCGARRY", 161200, "New Zealand",15);
System.out.println(defender15.getDetails());
Defender defender27 = new Defender(103, "ANGUS MACDONALD", 182000,"England",27);
System.out.println(defender27.getDetails());
Defender defender28 = new Defender(0, "JACK MILNE", 47320,"Scotland",28);
System.out.println(defender28.getDetails());
Defender defender30 = new Defender(0, "JOR DADIA", 93600,"Israel",30);
System.out.println(defender30.getDetails());
}
```

```
rage/7b69597119f8403098262e284f63b919/redhat.java/jdt_ws/footballteamproject_d9f56b58/bin Defender
Player{Name = RHYS WILLIAMS, Position = 18, TransferFee = £416000} Defender{ Defending moves = 0}
Player{Name = SLOBODAN RUBEZIC, Position = 33, TransferFee = £135200} Defender{ Defending moves = 108}
Player{Name = NICKY DEVLIN, Position = 2, TransferFee = £150800} Defender{ Defending moves = 103}
Player{Name = JACK MACKENZIE, Position = 3, TransferFee = £109200} Defender{ Defending moves = 89}
Player{Name = RICHARD JENSEN, Position = 5, TransferFee = £140400} Defender{ Defending moves = 93}
Player{Name = STEFAN GARTENMANN, Position = 6, TransferFee = £135200} Defender{ Defending moves = 115}
Player{Name = JAMES MCGARRY, Position = 15, TransferFee = £161200} Defender{ Defending moves = 98}
Player{Name = ANGUS MACDONALD, Position = 27, TransferFee = £182000} Defender{ Defending moves = 103}
Player{Name = JACK MILNE, Position = 28, TransferFee = £47320} Defender{ Defending moves = 0}
Player{Name = JOR DADIA, Position = 30, TransferFee = £93600} Defender{ Defending moves = 0}
jean-claude@MacBook-Pro-2 footballteamproject %
```

5. Midfielder Class coding (Semi Worked Example)

UML: modelling the Midfielder class functionality (DrawIO)

public Midfielder
private importantMoves: int private goalScored: int
public Midfielder(importantMoves: int, goalScored: int, name: String, transferFee: double, nationality:String, shirtNumber: int) public getImportantMoves(): int public getGoalsScored(): int public setImportantMoves(goalSaved: int): void public setGoalsScored(goalScored: int): void public getDetails(): String

Coding the Midfielder Class Attributes

We first create a Midfielder class, and then we declared the only object attribute that is needed for this class.

1) Create a Midfielder.java file

Repeat the same process as you did for the Defender class. Remember that this class has TWO fields.

Coding the Midfielder Class constructor

Remember that this class has two object attributes "importantMoves" and "goalScored". As this class is not a superclass (i.e., concrete class) like the Player class, its constructor must have enough parameters to initialise both these object attribute as well as having all the necessary parameters to initialise the Player super class using the "super(name, transferFee, nationality, shirtNumber);" command. Hence it needs 6 parameters.

```
// Full Specialised Constructor
public Midfielder(int importantMoves, int goalScored, String name, double transferFee, String nationality, int shirtNumber) {
    super(name, transferFee, nationality, shirtNumber);
    this.importantMoves = importantMoves;
    this.goalScored = goalScored;
}
```

Coding the Midfielder Class Setter Method

Code the setter method to retrieve the importantMoves and goalScored attributes value in the same manner as you do for the Goalkeeper class.

Coding the Midfielder Class Getter Method

Code the setter method to set the importantMoves and goalScored attributes value in the same manner as you do for the Goalkeeper class.

Coding the Midfielder class other Method

```
// Other Methods
// return the player details (superclass) and adds specific details (fields)
public String getDetails() {
    return super.toString() + " Midfielder{" + " Important Moves = " + importantMoves + '}';
}
```

Testing the Midfielder Class using its main(…) Method

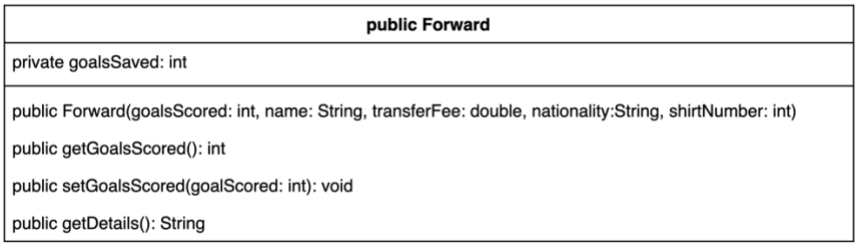
You can copy the code below into the Midfielder class and then run it

```
public static void main(String[] args) {
    Midfielder midfielder4 = new Midfielder(139, 15, "GRAEME SHINNIE", 218400, "Scotland", 4);
    System.out.println(midfielder4.getDetails());
    Midfielder midfielder7 = new Midfielder(155, 4, "JAMIE MCGRATH", 161200, "Ireland", 7);
    System.out.println(midfielder7.getDetails());
    Midfielder midfielder8 = new Midfielder(153, 1, "CONNOR BARRON", 45760, "Scotland", 8);
    System.out.println(midfielder8.getDetails());
    Midfielder midfielder23 = new Midfielder(0, 1, "RYAN DUNCAN", 36400, "Scotland", 23);
    System.out.println(midfielder23.getDetails());
    Midfielder midfielder32 = new Midfielder(0, 0, "FINDLAY MARSHALL", 28600, "Scotland", 32);
    System.out.println(midfielder32.getDetails());
    Midfielder midfielder10 = new Midfielder(110, 6, "LEIGHTON CLARKSON", 312000, "England", 10);
    System.out.println(midfielder10.getDetails());
    Midfielder midfielder22 = new Midfielder(0, 5, "VICENTE BESUIJEN", 78000, "Netherlands", 22);
    System.out.println(midfielder22.getDetails());
    Midfielder midfielder20 = new Midfielder(0, 0, "SHAYDEN MORRIS", 135200, "England", 20);
    System.out.println(midfielder20.getDetails());
    Midfielder midfielder21 = new Midfielder(155, 2, "DANTE POLVARA", 72800, "United States", 21);
    System.out.println(midfielder21.getDetails());
    Midfielder midfielder17 = new Midfielder(143, 28, "JONNY HAYES", 109200, "Ireland", 17);
    System.out.println(midfielder17.getDetails());
}
```

```
rage/7b69597119f8403098262e284f63b919/redhat.java/jdt_ws/footballteamproject_d9f56b58/bin Midfielder
Player{Name = GRAEME SHINNIE, Position = 4, TransferFee = £218400} Midfielder{ Important Moves = 139}
Player{Name = JAMIE MCGRATH, Position = 7, TransferFee = £161200} Midfielder{ Important Moves = 155}
Player{Name = CONNOR BARRON, Position = 8, TransferFee = £45760} Midfielder{ Important Moves = 153}
Player{Name = RYAN DUNCAN, Position = 23, TransferFee = £36400} Midfielder{ Important Moves = 0}
Player{Name = FINDLAY MARSHALL, Position = 32, TransferFee = £28600} Midfielder{ Important Moves = 0}
Player{Name = LEIGHTON CLARKSON, Position = 10, TransferFee = £312000} Midfielder{ Important Moves = 110}
Player{Name = VICENTE BESUIJEN, Position = 22, TransferFee = £78000} Midfielder{ Important Moves = 0}
Player{Name = SHAYDEN MORRIS, Position = 20, TransferFee = £135200} Midfielder{ Important Moves = 0}
Player{Name = DANTE POLVARA, Position = 21, TransferFee = £72800} Midfielder{ Important Moves = 155}
Player{Name = JONNY HAYES, Position = 17, TransferFee = £109200} Midfielder{ Important Moves = 143}
jean-claude@MacBook-Pro-2 footballteamproject %
```

6. Forward Class coding (Semi Worked Example)

UML: modelling the Forward class functionality (DrawIO)



Coding the Forward Class Attributes

We first create a Midfielder class, and then we declared the only object attribute that is needed for this class.

1) Create a Forward.java file

Repeat the same process as you did for the Defender class. Remember that this class has TWO fields.

Coding the Forward Class constructor

Remember that this class has two object attributes "goalScored".
As this class is not a superclass (i.e., concrete class) like the Player class, its constructor must have enough parameters to initialise both these object attribute as well as having all the necessary parameters to initialise the Player super class using the "super(name, transferFee, nationality, shirtNumber);" command. Hence it needs 5 parameters.

Repeat the same process as you did for the Midfielder class

Coding the Forward Class Setter Method

Code the setter method to retrieve "goalScored" attribute value in the same manner as you do for the Goalkeeper class.

Coding the Forward Class Getter Method

Code the setter method to set the "goalScored" attribute value in the same manner as you do for the Goalkeeper class.

Coding the Forward class other Method

```
// Other methods
// return the player details (superclass) and adds specific details (fields)
public String getDetails() {
    return super.toString() + " Forward{" + " Goals Scored = " + goalsScored + '}';
}
```

Testing the Forward Class using its main(...) Method

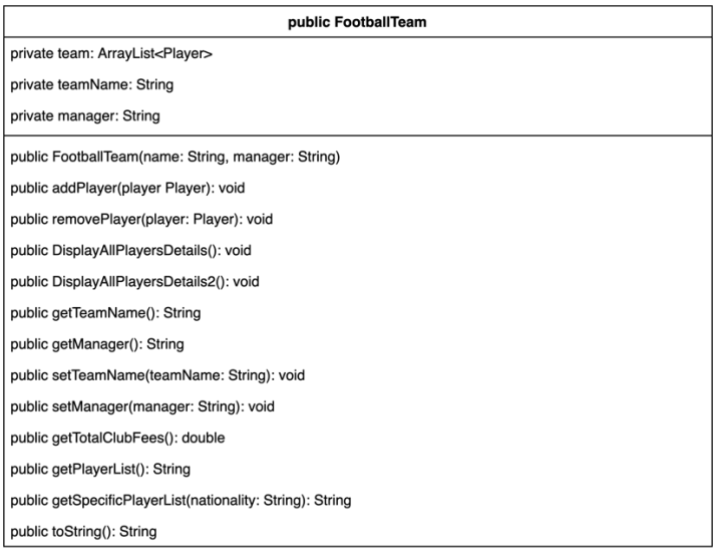
You can copy the code below into the Forward class and then run it

```
    public static void main(String[] args) {
    Forward forward9 = new Forward(26, "BOJAN MIOVSKI", 234000, "North Macedonia", 9);
    System.out.println(forward9.getDetails());
    Forward forward11 = new Forward(18, "DUK", 176800, "Cape Verde", 11);
    System.out.println(forward11.getDetails());
    Forward forward14 = new Forward(0, "PAPE GUÈYE", 145600, "Senegal", 14);
    System.out.println(forward14.getDetails());
    Forward forward19 = new Forward(1, "ESTER SOKLER", 119600, "Slovenia", 19);
    System.out.println(forward19.getDetails());
    Forward forward36 = new Forward(0, "ALFIE BAVIDGE", 0, "Scotland", 36);
    System.out.println(forward36.getDetails());
}
```

```
/bin Forward
Player{Name = BOJAN MIOVSKI, Position = 9, TransferFee = £234000} Forward{ Goals Scored = 26}
Player{Name = DUK, Position = 11, TransferFee = £176800} Forward{ Goals Scored = 18}
Player{Name = PAPE GUÈYE, Position = 14, TransferFee = £145600} Forward{ Goals Scored = 0}
Player{Name = ESTER SOKLER, Position = 19, TransferFee = £119600} Forward{ Goals Scored = 1}
Player{Name = ALFIE BAVIDGE, Position = 36, TransferFee = £0} Forward{ Goals Scored = 0}
jean-claude@MacBook-Pro-2 footballteamproject %
```

7. FootballTeam Class coding (Worked Example)

UML: modelling the FootballTeam class functionality (DrawIO)



Coding the FootballTeam Class Attributes

We first create a Midfielder class, and then we declared the only object attribute that is needed for this class.

1) Create a FootballTeam.java file

This class has 3 fields. One is the collection (ArrayList) that will store all the players, one is the team's name and one is the manager's full name.

```java
import java.util.ArrayList;

public class FootballTeam {

    // Fields for FootballTeam concrete class
    private ArrayList<Player> team;
    private String teamName;
    private String manager;
```

Coding the FootballTeam Class constructor

Remember that you only declared the ArrayList above, it must be CREATED! This is done in the constructor

```java
// Full Specialised Constructor
public FootballTeam(String name, String manager) {
    this.team = new ArrayList<>();
    this.teamName = name;
    this.manager = manager;
}
```

Coding the FootballTeam Class Setter Methods

```java
// Setter Methods
public void setTeamName(String teamName) {
    this.teamName = teamName;
}

public void setManager(String manager) {
    this.manager = manager;
}
```

Coding the FootballTeam Class Getter Methods

```java
// Getters Methods

public String getTeamName() {
    return teamName;
}

public String getManager() {
    return manager;
}
```

Coding the FootballTeam class other Methods

Here we will only code 3 of them that are needed to display all the player information

```java
// Other Methods

// Get the total players fees the club has to pay for all players
// we iterate through the list and invoke the getTransferFee() method
// to get the fees that are added to totalFees
public double getTotalClubFees() {
    double totalFee = 0;
    for (Player player : team) {
        totalFee += player.getTransferFee();
    }
    return totalFee;
}
```

```java
// this method return in a formated string a less detailed information
// about players. Name, Nationality and transer fees only
public String getPlayerList() {
    String output = "";
    for (Player player : team) {
        output += player.getName() + "( Nationality: " + player.getNationality() +
        "  Transfer Fees: " + player.getTransferFee() + ")" + "\n";
    }
    return output;

// display information invoking the superclass toString
// and then the specific getDetails() method
// once we detect what they are in the field
public void displayAllPlayerDetails2() {
    for (Player player : team) {
        if (player instanceof Defender) {
            Defender defender = (Defender) player;
            System.out.println(defender.getDetails());
        } else if (player instanceof Goalkeeper) {
            Goalkeeper goalkeeper = (Goalkeeper) player;
            System.out.println(goalkeeper.getDetails());
        } else if (player instanceof Forward) {
            Forward forward = (Forward) player;
            System.out.println(forward.getDetails());
        } else if (player instanceof Midfielder) {
            Midfielder midfielder = (Midfielder) player;
            System.out.println(midfielder.getDetails());
        }
    }
}
```

Testing the FootballTeam class

Download the Main.java file. Copy it in the project folder where the other java files are. It should appear in VC. Open and run it You may want to reduce the number of Aberdeen players!

```
Team Information:
The Football Team name is Aberdeen FC
Manager is BARRY ROBSON
Total Fees is 3829800.0

The Players are:
KELLE ROOS( Nationality: Netherlands  Transfer Fees: 208000.0)
TOM RITCHIE( Nationality: Scotland  Transfer Fees: 36920.0)
ROSS DOOHAN( Nationality: Scotland  Transfer Fees: 140400.0)
RHYS WILLIAMS( Nationality: England  Transfer Fees: 416000.0)
SLOBODAN RUBEZIC( Nationality: Serbia  Transfer Fees: 135200.0)
NICKY DEVLIN( Nationality: Scotland  Transfer Fees: 150800.0)
JACK MACKENZIE( Nationality: Scotland  Transfer Fees: 109200.0)
RICHARD JENSEN( Nationality: Finland  Transfer Fees: 140400.0)
STEFAN GARTENMANN( Nationality: Denmark  Transfer Fees: 135200.0)
JAMES MCGARRY( Nationality: New Zealand  Transfer Fees: 161200.0)
ANGUS MACDONALD( Nationality: England  Transfer Fees: 182000.0)
JACK MILNE( Nationality: Scotland  Transfer Fees: 47320.0)
JOR DADIA( Nationality: Israel  Transfer Fees: 93600.0)
GRAEME SHINNIE( Nationality: Scotland  Transfer Fees: 218400.0)
JAMIE MCGRATH( Nationality: Ireland  Transfer Fees: 161200.0)
CONNOR BARRON( Nationality: Scotland  Transfer Fees: 45760.0)
RYAN DUNCAN( Nationality: Scotland  Transfer Fees: 36400.0)
FINDLAY MARSHALL( Nationality: Scotland  Transfer Fees: 28600.0)
LEIGHTON CLARKSON( Nationality: England  Transfer Fees: 312000.0)
VICENTE BESUIJEN( Nationality: Netherlands  Transfer Fees: 78000.0)
SHAYDEN MORRIS( Nationality: England  Transfer Fees: 135200.0)
DANTE POLVARA( Nationality: United States  Transfer Fees: 72800.0)
JONNY HAYES( Nationality: Ireland  Transfer Fees: 109200.0)
BOJAN MIOVSKI( Nationality: North Macedonia  Transfer Fees: 234000.0)
DUK( Nationality: Cape Verde  Transfer Fees: 176800.0)
PAPE GUÈYE( Nationality: Senegal  Transfer Fees: 145600.0)
ESTER SOKLER( Nationality: Slovenia  Transfer Fees: 119600.0)
ALFIE BAVIDGE( Nationality: Scotland  Transfer Fees: 0.0)


--------------------------------

Player{Name = KELLE ROOS, Position = 24, TransferFee = £208000} Goalkeeper{ Goal Saved = 72}
Player{Name = TOM RITCHIE, Position = 25, TransferFee = £36920} Goalkeeper{ Goal Saved = 0}
Player{Name = ROSS DOOHAN, Position = 31, TransferFee = £140400} Goalkeeper{ Goal Saved = 51}
Player{Name = RHYS WILLIAMS, Position = 18, TransferFee = £416000} Defender{ Defending moves = 0}
Player{Name = SLOBODAN RUBEZIC, Position = 33, TransferFee = £135200} Defender{ Defending moves = 108}
Player{Name = NICKY DEVLIN, Position = 2, TransferFee = £150800} Defender{ Defending moves = 103}
Player{Name = JACK MACKENZIE, Position = 3, TransferFee = £109200} Defender{ Defending moves = 89}
Player{Name = RICHARD JENSEN, Position = 5, TransferFee = £140400} Defender{ Defending moves = 93}
Player{Name = STEFAN GARTENMANN, Position = 6, TransferFee = £135200} Defender{ Defending moves = 115}
Player{Name = JAMES MCGARRY, Position = 15, TransferFee = £161200} Defender{ Defending moves = 98}
Player{Name = ANGUS MACDONALD, Position = 27, TransferFee = £182000} Defender{ Defending moves = 103}
Player{Name = JACK MILNE, Position = 28, TransferFee = £47320} Defender{ Defending moves = 0}
Player{Name = JOR DADIA, Position = 30, TransferFee = £93600} Defender{ Defending moves = 0}
Player{Name = GRAEME SHINNIE, Position = 4, TransferFee = £218400} Midfielder{ Important Moves = 139}
Player{Name = JAMIE MCGRATH, Position = 7, TransferFee = £161200} Midfielder{ Important Moves = 155}
Player{Name = CONNOR BARRON, Position = 8, TransferFee = £45760} Midfielder{ Important Moves = 153}
```