DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 2210

Object Oriented Analysis and Design

SPORTS LEAGUE MANAGEMENT SYSTEM

by

Cafer Hakan Acar

Muhammed Aydoğan

Umut Devrim Kaya

CHAPTER ONE

INTRODUCTION

The subject of this project is to create a league system in any sports branch we have determined to code. In line with this sport branch we choose, all the necessary information for the league will be taken from the file. After all teams and players have been created, our league will begin. Until the league system is completed, the teams will be able to transfer player according to their transfer budgets. At the end of this whole league, one team will win the league as a champion. The rules within the league will vary according to the sports branch we have determined to code. As a group, our current idea is to form a league system on the football branch. However, since the logic of the league systems is very close to each other, this algorithm we have developed will be very easily implemented in other systems.

The aim of our project is to be a champion according to the rules mentioned before.In this project you have your own team. You can transfer the players from other teams if the rules allow also create your players as you want and also project have a league and you can match the other teams like a real league and collect the points to end of the final match.The algorithm of that determines who will win will be determined according to the strengths of the teams. If user wants to be a champion of the league he/she should built the team carefully and should manage transfer relations.

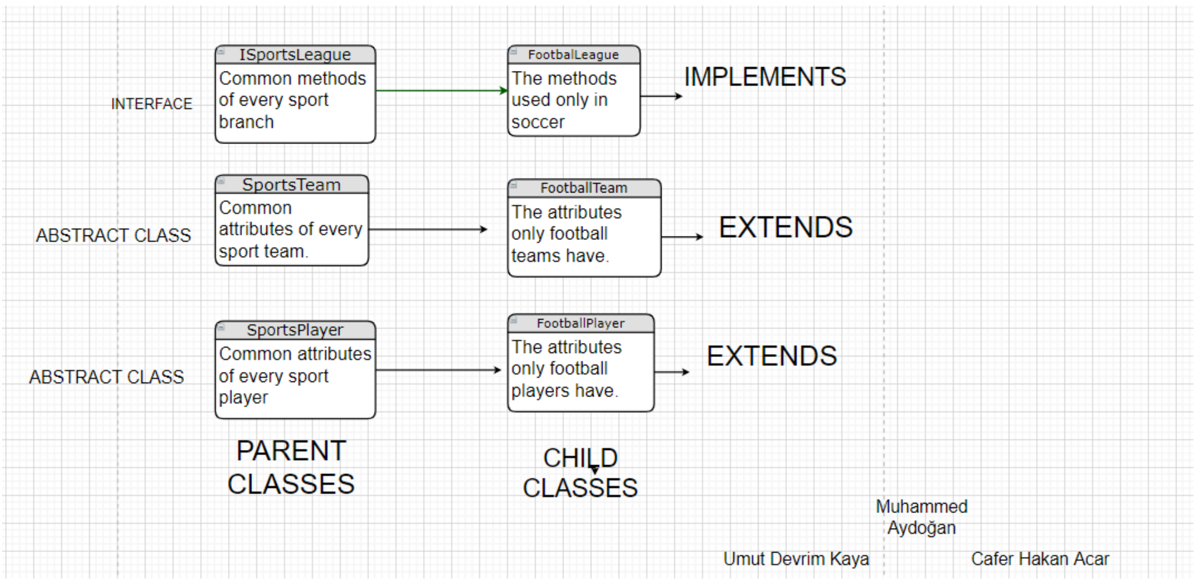
Project’s scope can be defined as customer’s order depending on the sport branch. Customer can always get feedback about the statement of their progress,league,transfer system or economical situation of the team. Given timeline allows to develop a well-designed system for our strong-designed project. In final stages of the project, it will be reviewed for additional features depend on the customer feedback. Any kind of algorithm problem will be handled from a programmer’s view according to algortihm efficiency techniques.

CHAPTER TWO

REQUIREMENTS

The strength of the teams will be formed according to the strength of the players in the team (attack, defense, morale, etc.). Team’s winning matches will contribute to the morale of the players and the team's budget. File processes will be impletemented using java reading functions and results will be written to the text file to store log.

Project’s main structure can be shown as follows:


INTERFACE
ISportsLeague
Common methods of every sport branch
FootbalLeague
The methods used only in soccer
IMPLEMENTS
ABSTRACT CLASS
SportsTeam
Common attributes of every sport team.
FootballTeam
The attributes only football teams have.
EXTENDS
ABSTRACT CLASS
SportsPlayer
Common attributes of every sport player
FootballPlayer
The attributes only football players have.
EXTENDS
PARENT CLASSES
CHILD CLASSES
Muhammed Aydoğan
Umut Devrim Kaya
Cafer Hakan Acar

```java
public interface ILeague {

        public FootballTeam matchTeams(FootballTeam team1,FootballTeam team2);
        public void increasePoint(FootballTeam team);
        public void createAllTeams();
        public void createFixture();
        public boolean isBudgetEnough(FootballTeam team) {}
        public boolean isSizeofTeamConvenient(FootballTeam team) {}
        public void makeTransfer(Player playerToTransfer) {}

}
```

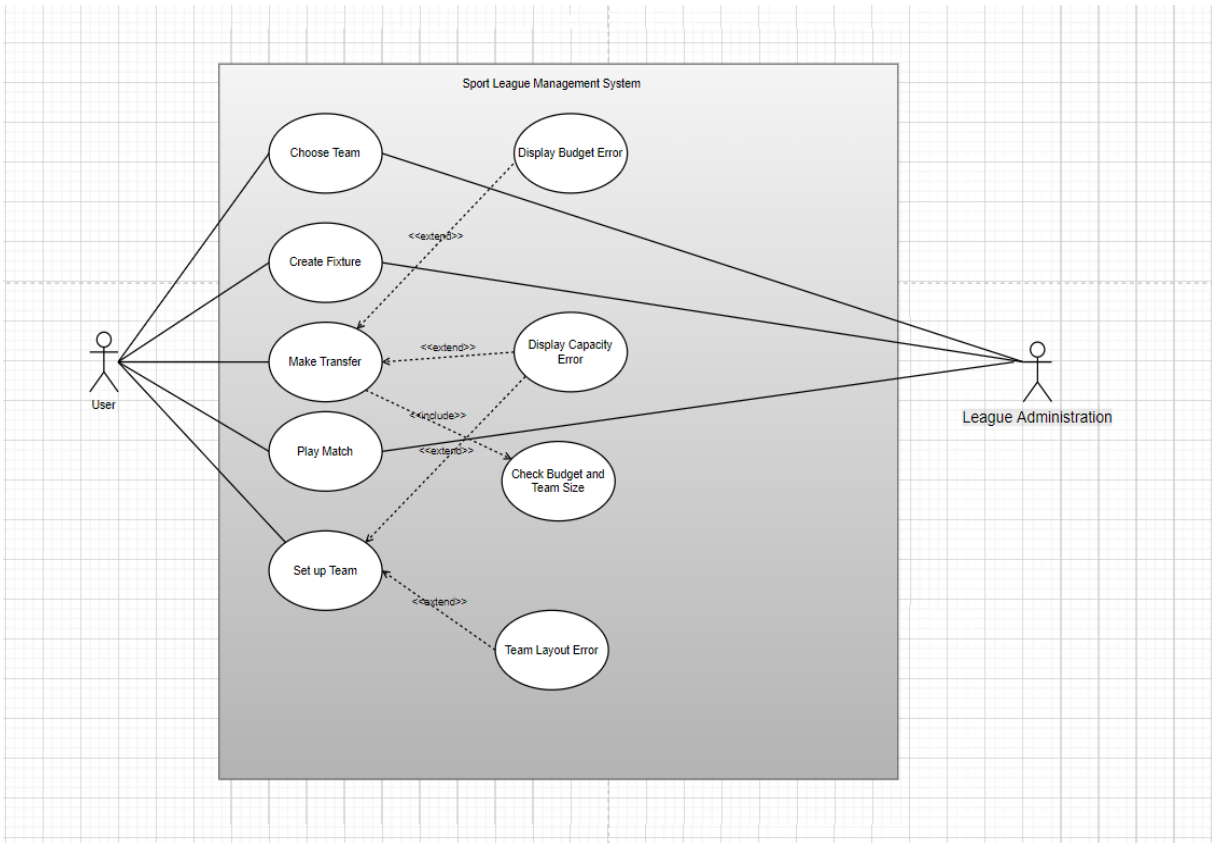
```
public abstract class SportsTeam {

	private String name;
	private int point=0;
	private int winCount=0;
	private int lossCount=0;
	private ArrayList<Player> players;
	private int MAX_PLAYER;
	private int teamStrength;
	private int teamMoral;
	private int budget;
	private int currentPlayer;



	public int getPoint() {
		return point;
	}
	public void setPoint(int point) {
		this.point = point;
	}

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getWinCount() {
		return winCount;
	}
	public void setWinCount(int winCount) {
		this.winCount = winCount;
	}
	public int getLossCount() {
		return lossCount;
	}
	public void setLossCount(int lossCount) {
		this.lossCount = lossCount;
	}
	public ArrayList<Player> getPlayers() {
		return players;
	}
	public void setPlayers(ArrayList<Player> players) {
		this.players = players;
	}
	public int getMAX_PLAYER() {
		return MAX_PLAYER;
	}
	public void setMAX_PLAYER(int mAX_PLAYER) {
		MAX_PLAYER = mAX_PLAYER;
	}
	public int getTeamStrength() {
		return teamStrength;
	}
	public void setTeamStrength(int teamStrength) {
		this.teamStrength = teamStrength;
	}
	public int getTeamMoral() {
		return teamMoral;
	}
	public void setTeamMoral(int teamMoral) {
		this.teamMoral = teamMoral;
	}
	public int getBudget() {
		return budget;
	}
	public void setBudget(int budget) {
		this.budget = budget;
	}
	public int getCurrentPlayer() {
		return currentPlayer;
	}
	public void setCurrentPlayer(int currentPlayer) {
		this.currentPlayer = currentPlayer;
	}

}

public abstract class SportsPlayer {

	private int moral;
	private int strength;
	private String name;
	private int age;
	private SportsTeam team;



	public int getMoral() {
		return moral;
	}
	public void setMoral(int moral) {
		this.moral = moral;
	}
	public int getStrength() {
		return strength;
	}
	public void setStrength(int strength) {
		this.strength = strength;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}
	public SportsTeam getTeam() {
		return team;
	}
	public void setTeam(SportsTeam team) {
		this.team = team;
	}

}


public class FootballLeague implements ILeague {


	private ArrayList<FootballTeam> teams;




	@Override
	public FootballTeam matchTeams(FootballTeam team1, FootballTeam team2) {



	}
	@Override
	public void increasePoint(FootballTeam team) {


	}
	@Override
	public void createAllTeams() {

	}
	@Override
	public void createFixture() {


	}
	@Override
	public boolean isBudgetEnough(FootballTeam team) {
		// TODO Auto-generated method stub
		return false;
	}
	@Override
	public boolean isSizeofTeamConvenient(FootballTeam team) {
		// TODO Auto-generated method stub
		return false;
	}
	@Override
	public void makeTransfer(Player playerToTransfer) {
		// TODO Auto-generated method stub

	}
```

```
}



public class FootballTeam extends SportsTeam {



        private int tieCount;
        private int goalScored;
        private int goalTaken;

        private int determineTeamPower(ArrayList<Player> players) {
_______________
        }

        private int determineTeamMoral(ArrayList<Player> players) {
_______________
        }


}

public class Player extends SportsPlayer {

        private String position;
        private boolean foot;
        public String getPosition() {
                return position;
        }
        public void setPosition(String position) {
                this.position = position;
        }
        public boolean isFoot() {
                return foot;
        }
        public void setFoot(boolean foot) {
                this.foot = foot;
        }

}
```

CHAPTER THREE

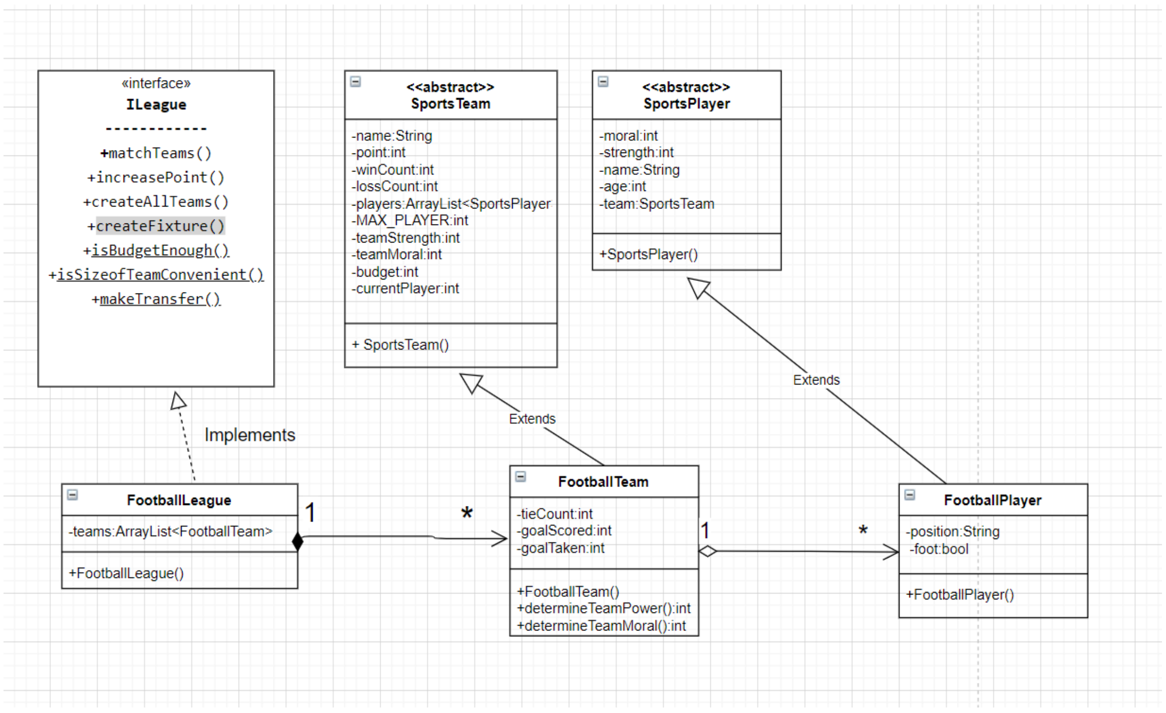
UML DIAGRAMS

Use Case Diagram



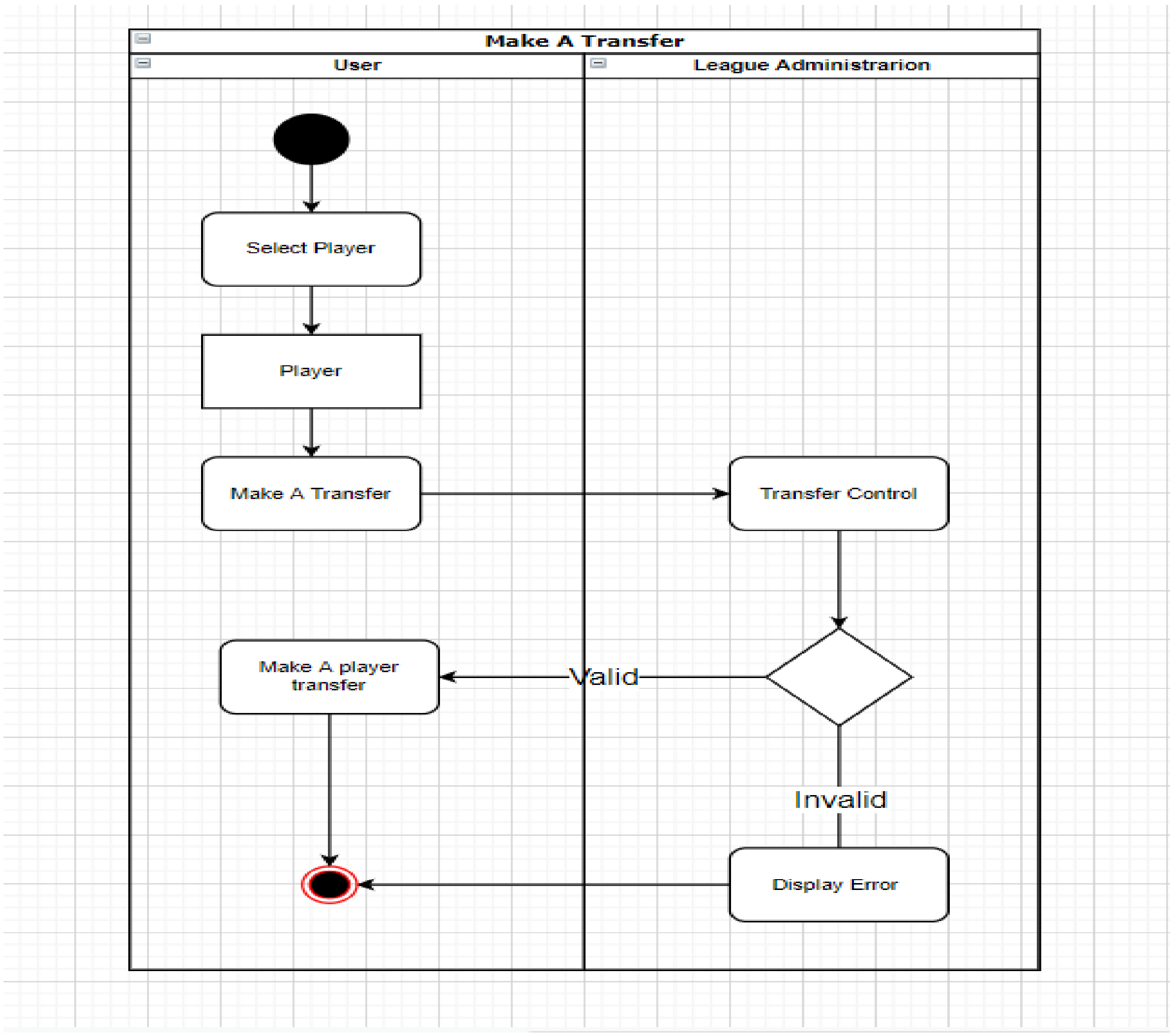
We want to show the links between our actions in the use case diagram. The customer can interact with the program by selecting the options here. The user has simple configurations such as choosing team, transfering, playing match, setting up team. The conditions on which the configurations depend on are shown for these actions to occur. The actors who fulfill these conditions are bound to the necessary places.

Class Diagram



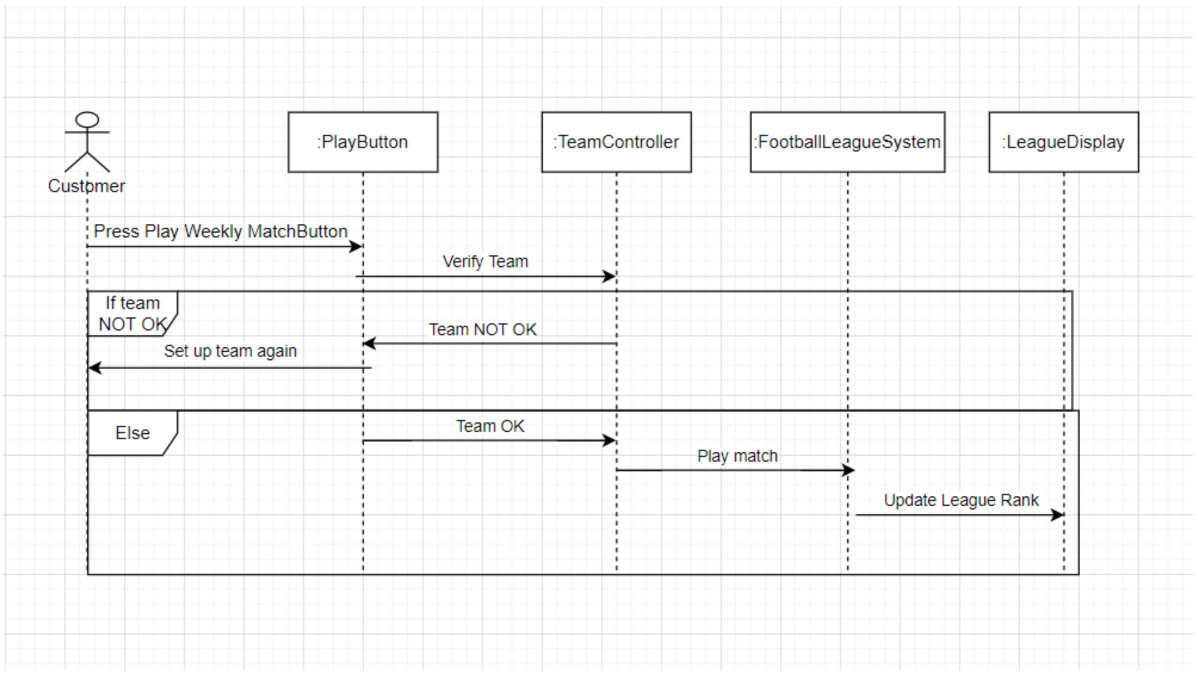
A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system. The basic classes of our project and their properties and methods are listed in the class diagram. The abstraction and inheritance features, which are the basic principles of OOP, have been implemented to achieve a more robust structure of our project.

Activity Diagram



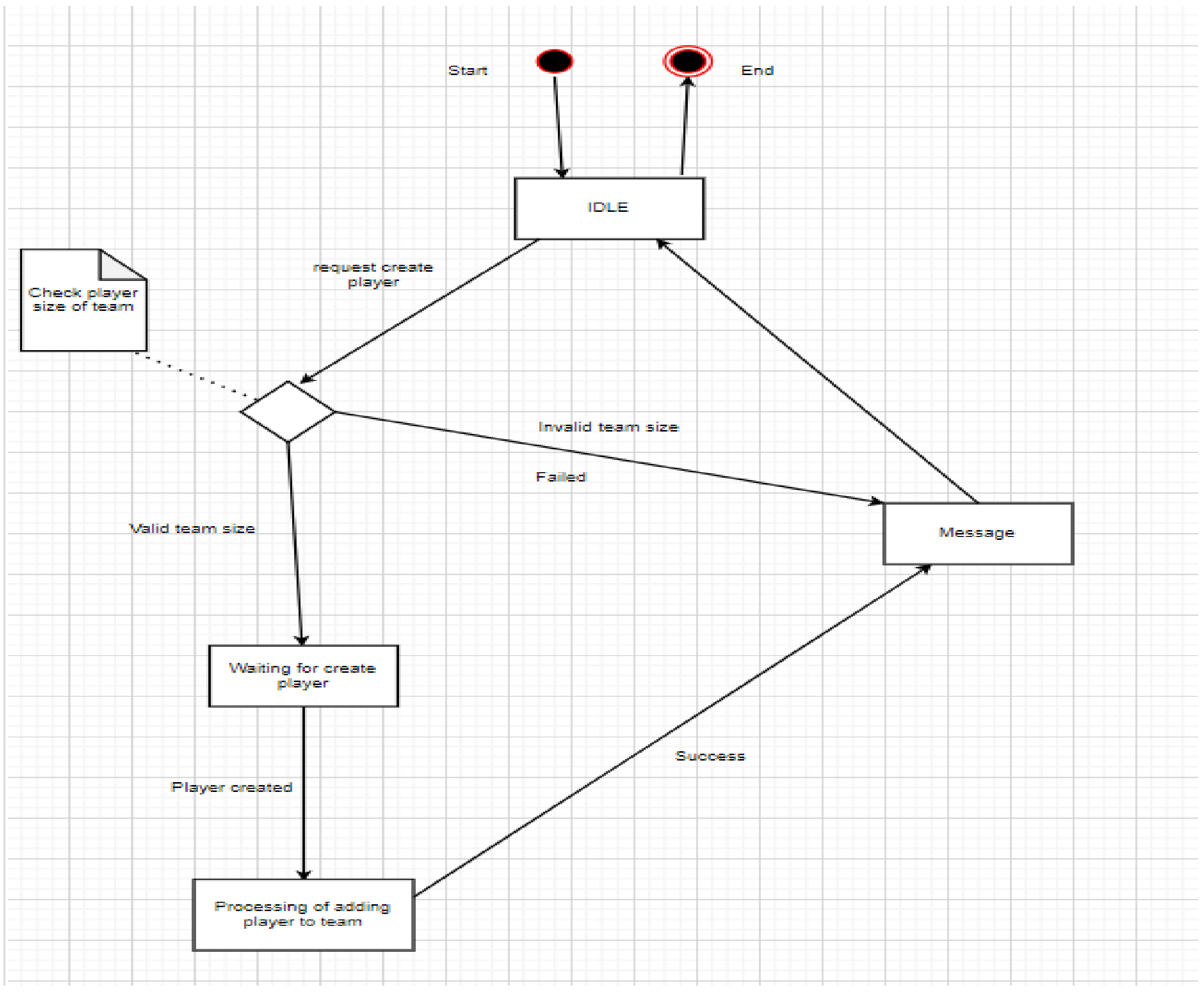
Activity diagram was made for Transfer Implementation. In this process, transfer control (capacity and budget control) was performed before the transfer was made. If it passed all the controls, the transaction was made. Otherwise, an error message was given by the system.

Sequence Diagram



A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. In this diagram player hits the playButton to play the weekly match. After that TeamController objects checks the team whether it is appropriate or not. After the team is OK leaugeSystem runs the match.

State Diagram



State diagram is status of a System/Object/Subsystem at a particular time.In our state diagram we wanted to show status of a create player.So firstly user has to make a request and the system controls the team size.If size of team is not valid system gives an error message to the screen.If size is valid system waits for information about player .Then player created and added to team.After that system gives a success message for user and diagram ends

CHAPTER FOUR

IMPLEMENTATION

Singleton design pattern used in the project for sustain robust architecture. League object created only once in private constructor to apply it. Fixture System is implemented with the algorithm given in reference part. User can arrange their team by selecting from the current players and the reserve players which is hold in arraylists. After that according to the position, players placed on the team line-up display.

CreateAllTeams() this function reads the text file which is included players list and then create players and teams.End of the this function createBudgetAllTeams() function runs.This also assign budget to all teams to make transfer.

MakeMatch(Match match): This function provides making match between 2 teams
and kept it's statistic.But how? First, the average power of the teams is determined and then generating 2 random numbers if difference between that two number is less or equal than 10 match ends in a draw.İf powerteam1>powerteam2 host team wins the match.Else away team wins the match.The scores of the teams are determined by their strength. And then determine scorer and assister player.When determining them we kept the probability of strikers to score higher than the othersThen we determine goal minutes finally adding goal to goals arraylist in match class.

MakeTransfer(Player player,FootballTeam newTeam): Basically, there are certain rules that need to be checked before the transfer is made.Team size has to be greater than 20 and less than 40.Then the player can not transfer to his own team.And finally size of position is convenient ,transfer takes place.So player's new team is determined and is deducted from the budget of the new team

getBestScorerPlayer()	in these function, we use stack
getBestAssisterPlayer()	and queue effectively.So this
getBestScorerTeams()	function returns stack or
getLeastScorerTeams()	queue and in statistic tab
mostGoalTakenTeams()	we call these functions
leastGoalTakenTeams()	and writing the top 5.

DetermineFirstEleven(): This function automatically adds the strongest players to the top 11 players

In the Scoreboard process, a ranking was made according to the scores obtained after weekly matches were played. This ranking was sorted according to the number of points, averages, goals and printed on the score board screen.

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

In this project we learned new skill called swing and improved our java skills.We worked together as a team on project.So every team member has done his tasks and during the making project we encountered errors.We kept in touch throughout the project.So we overcome the problems.We finished entire Project despite this diffucult and intense process.We could have done a few more nice touches but this semester we did more than one homework and exam for each course separately.But despite everything we successfully finished the Project.

We have created league system with only for soccer teams. In future other sports may involve the project. Architecture is designed according to oop and design patterns used for sustainability of the project. Details of the matches will be optional to display in later works. In addition to the system, stadium properties may have part of the system for the income level of the teams. Medical healthcare crew, player training system (for increasing their power), improved team tactic arrangement system , and many more can be done in the future.

Fetching data from txt file can seem primitive today.In order to overcome this situation DB Management Systems may have form the fundamental of data modeling. In the next terms authentication may be added to the system. With that user will be able to hold their records and passwords in the database. Basic CRUD operations will be handled by the way. For foreign players multiple language support may show up.

REFERENCES

HTTP://WWW.NUHAZGINOGLU.COM/TAG/FIXTURE-ALGORITHM/

HTTPS://WWW.EDUCATIVE.IO/EDPRESSO/HOW-TO-USE-THE-STACK-CLASS-IN-JAVA