FootyBettor

Sports Betting Platform (SBP)

Software Design Document v2- Sprint 2

Prepared by:
Zahra Cheeseman
Esa Chen
Junhao Qu
Shamdeed Kabir

Release Date:
Tuesday 21st March, 2023

Contents

Executive Summary

Sports Betting Platform (SBP) is a unique form of entertainment that combines the excitement of sports with the thrill of gambling. SBP is intended to exist in similar markets as other sports betting websites, but provides more data-backed informed odd calculations, and uncluttered, understandable, and easy-to-use operation pages. SBP softwares will be distributed as standalone applications for use on end-user hardware without network connectivity.

This document provides technical information regarding the software design of SBP.

Document Versioning

Date	Owner	Comment
03/03/2023	Zahra Cheeseman, Shamdeed Kabir, Esa Chen, Junhao Qu	Executive Summary and Project Description
03/06/2023	Zahra Cheeseman, Shamdeed Kabir, Esa Chen, Junhao Qu	Feature Matrix and Feature Discussion
03/06/2023	Shamdeed Kabir	High-level System Architecture
03/06/2023	Zahra Cheeseman and Esa Chen	Major Components
03/06/2023	Zahra Cheeseman and Esa Chen	Detail Design
03/06/2023	Shamdeed Kabir and Junhao Qu	UML Diagrams
03/06/2023	Zahra Cheeseman and Esa Chen	Data Format
03/12/2023	Zahra Cheeseman, Shamdeed Kabir, Esa Chen, Junhao Qu	Updated Detail Design, UML Design and Data Format
03/15/2023	Esa Chen and Shamdeed Kabir	Updated Detail Design
03/15/2023	Zahra Cheeseman	Updated Project Description
03/15/2023	Shamdeed Kabir and Junhao Qu	Updated UML Diagrams
03/15/2023	Esa Chen and Zahra Cheeseman	Updated Feature Matrix and Feature Discussion
03/15/2023	Zahra Cheeseman	Updated Data Format
03/21/2023	Shamdeed Kabir	Entity Relationship Diagram
03/21/2023	Esa Chen	Updated Class Descriptions
03/21/2023	Junhao Qu and Shamdeed Kabir	Updated UML Diagrams
03/21/2023	Zahra Cheeseman	Updated Components Section and Feature Discussion

Project Description

FootyBettor is looking to enter the sports gambling market. This market is saturated with platforms with clunky and complex operation pages, limited data availability and implicit probability distribution. Sports Betting Platform (SBP) is FootyBettor's creation which provides a solution to these problems which traditional betting platforms present which create difficulty in users being able to make informed decisions, place winning bets and succeed in the highly competitive sports betting industry. SBP allows for impressive data visualisation of previous season team statistics and a smooth user experience for easy betting on the result of matches between sports teams. Users can create an account in which they navigate through the program, and view a balance which they can add to and which gets updated in relation to the return on their bets. SBP generates precise odds which are informed by a multitude of statistics used in calculations which optimise accuracy.

SBP is highly extendable and thus would have potential to be able to support different applications such as a variation in the sport selected and type of bet available.

Additionally, SBP supports a graphical user interface. The GUI allows SBP to provide a sleek user interface, which gives the user a fun and accessible user experience.

Features

The feature matrix enumerates the technical features required to support each of the business requirements. The discussion section provides details regarding the constraints and functionality of the feature. The ids are used for traceability. Features that can be removed should strike-through the feature id and have a comment added to identify why this feature can be removed without impacting the BRD requested functionality.

Priority Codes:

H - High, a must-have feature for the product to be viable and must be present for launch
M - Medium, a strongly desirable feature but product could launch without
L - Low, a feature that could be dropped if needed

Feature Matrix

ID	Pri	Feature Name	Comment	BRD ID
P.1	H	Language	Implementation of JAVA	s.1 , s.3
I.1	H	GUI		ux.1 , e.2
C.1	H	System Loop		e.1 , e.3
C.2	H	Save Data	Store user profile	e.4 , e.5 , e.10
C.3	H	Actions		e.5 , e.6 , e.7 , e.8
C.4	M	Represent Data		e.6
T.1	H	Error Handling		ux.2 , e.1

Feature Discussion

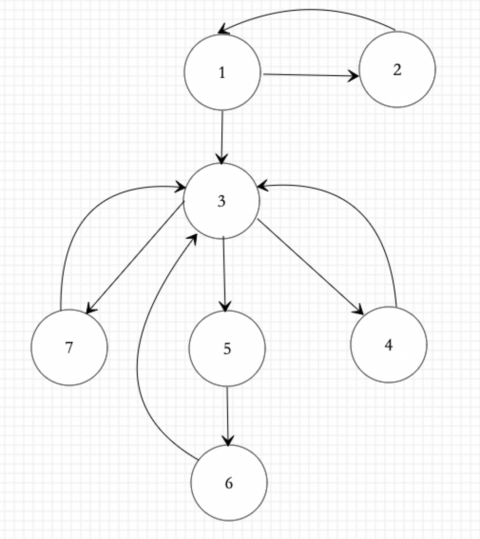
P.1 - Language

There are many possibilities for the choice of programming language that SBP will utilise. Java provides the best fit for the capabilities that SBP must support. Java has cross-platform support, includes capabilities for making executable bundles and has consistent GUI behaviour across platforms.

I.1 - GUI

SBP provides a graphic user interface (GUI) for interacting with the software, thus it must have a user-friendly, easy-to-use, and aesthetic interface pattern that is well suited for GUI operations.

C.1 - System Loop



All SBP will use the same basic loop system.

1. User log-in
2. User sign up and return to sign in
3. Get the user's choice
4. Show previous session stats and return to 3
5. Bet on a game (including selection of odd type and insurance/ donation preferences)
6. Show the betting result for the user and return to 3
7. Show balance and return to 3

C.2 - Save Data

Save Data means to store the user's profile including username, password, and balance. Each user profile is stored in a mySQL database in a table named user, which has a username, password and balance column. SBP provides options for logging in and signing up for an account. To log in to SBP, the user can enter the username and password, then the system would check whether there is a corresponding user in the table. If the user exists, the system would check the password and record the user's account balance. For the option of signing up for an account, the platform asks for the user's input for username and password. A new entry in the user table will be created which will contain all of the new account information and initialise the balance as 0, allowing the user to log in.

C.3 - Actions

Actions means all of the actions the platform will perform when users make a choice. The platform provides a menu that contains all of the actions for users to choose from. These actions can store user data, represent data, and track balance.

C.4 - Represent Data

An integral part to SBP is that previous team statistics can be viewed by the user, in order for them to make an informed decision on their bet. They can view both complete previous season statistics and data specific to the teams that they select. All of this data will be stored in tables in the mySQL database. The teamstat table stores the previous season data including teams and

the number of games they won, lost and tied, and the fixture table stores the current season data regarding the fixture, the result, and the home and away teams.

T.1 - Error Handling

If the system reaches an unsafe state or the provided AL/EL has system stopping defects during parsing or execution, error information should be provided to inform the end user that an error has occurred. The user should then be presented with an option to have SBP produce a more detailed error report that could be sent back to the game developer as part of a game bug report.

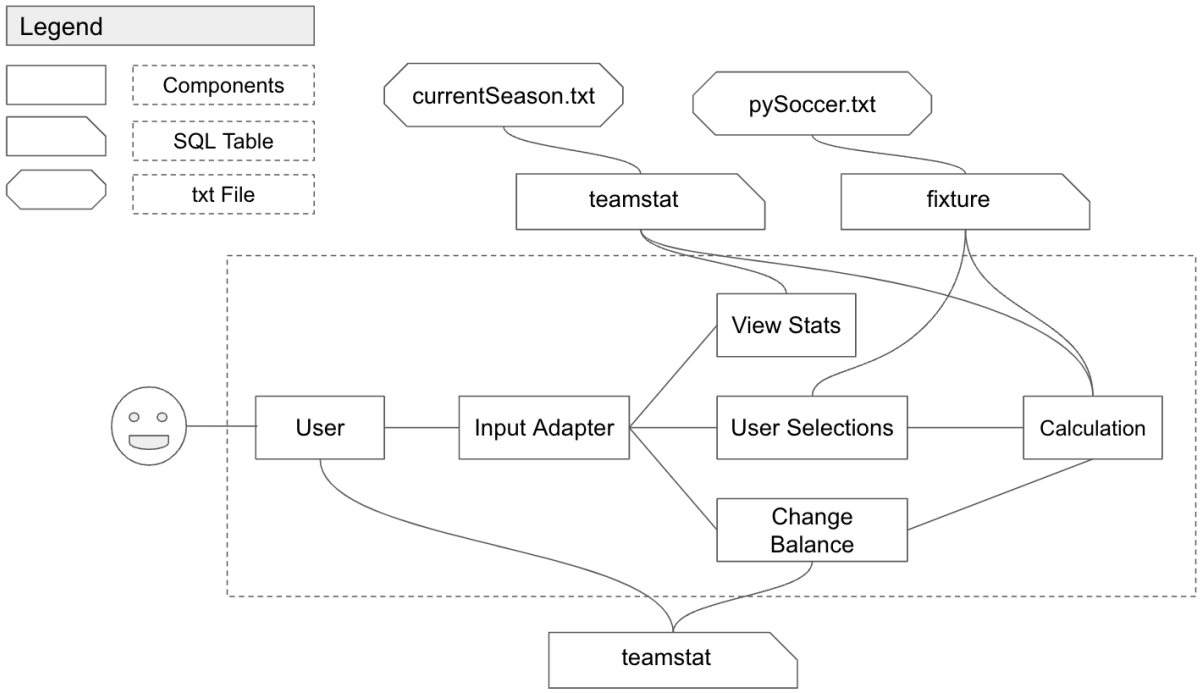
System Design

This section describes the system design in detail. An overview of the major system components and their roles will be provided, followed by more detailed discussion of components. Component discussion will reference the technical features the component helps satisfy.

Note: Design co-evolves with implementation. It is important to start with some sense of components and their interactions, but design docs should be updated during implementation to capture the actual system as-built.

Architecture Overview

High-level System Architecture



High-level Architecture Diagram

Major Components

InputAdapter

I.1, C.1
InputAdapter is responsible for managing the input choice of the user and then determining which actions to run in response to that.

User

C.2
This component is responsible for storing user information in the database when users sign up and checking the user's password when they log in.

User selections

C.3
This component controls all user selections- including the teams which they select, their predicted result, their wager, the odds that they choose and the selection on the donation/insurance page.

Change/Update balance

C.2, C.3
The Change/Update balance component is responsible for the functionality of displaying the current balance and adding money to the current balance. After the change is made by the user, the system should update the balance stored in the useraccount table in the mySQL database.

Calculation

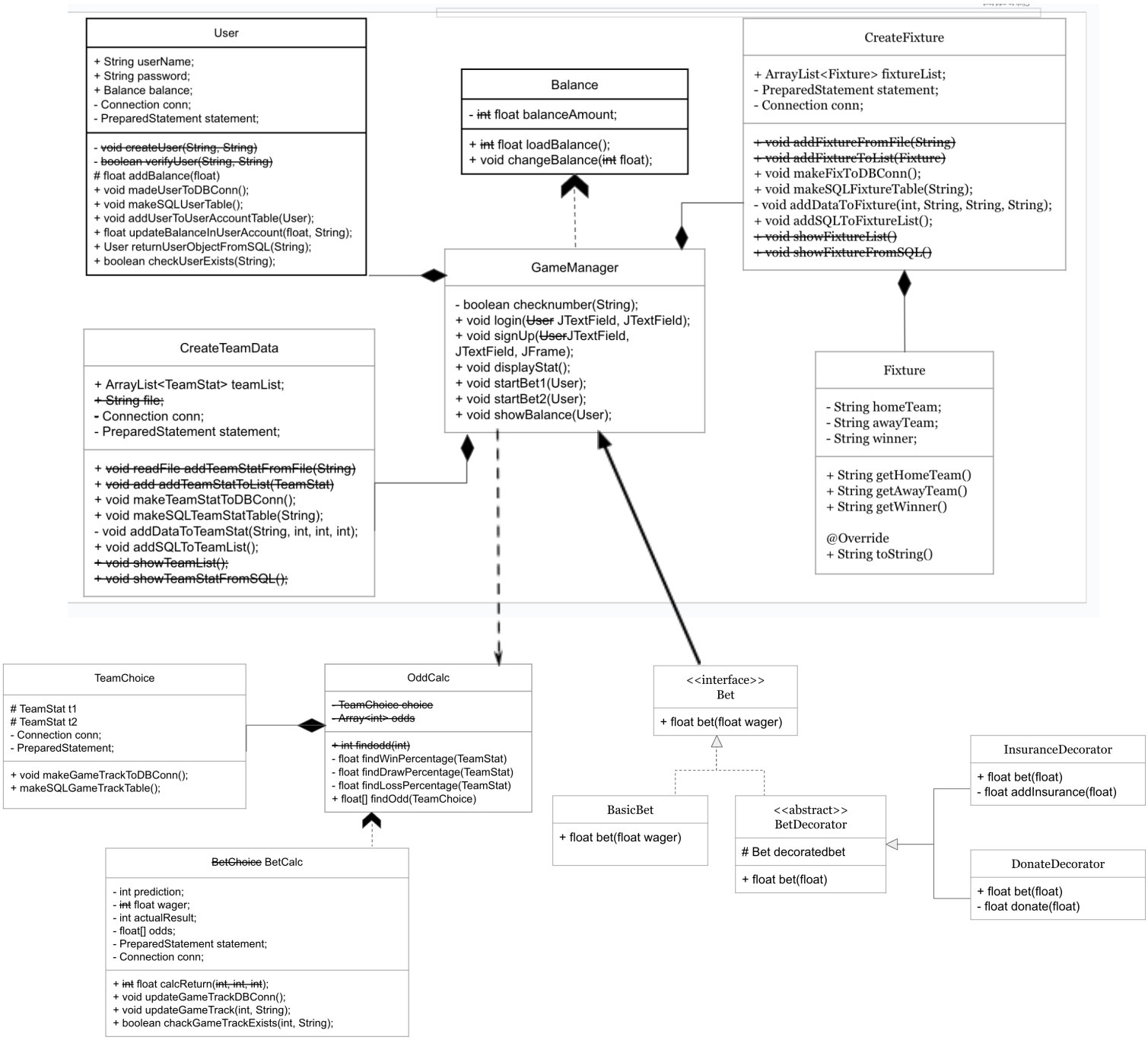
C.3
The calculation component manages all calculations- including taking in data in order to generate the different odds and returns. The calculation component implements the decorator and strategy pattern.

View Stats

C.4
Controls the display of the statistics from the previous season's team data stored in the teamstat table.

Detail Design

InputAdapter Component



Class Diagram

GameManager

The GameManager class which has only methods which are used in Main to control the system state/ user pathway. It includes the login(), signUp(), displayStat(), ~~starBet(User)~~, startBet1(User), startBet2(User), and showBalance(User) methods. It interacts with all of the other classes. The startBet1(User) is called in the Main class for the user to bet by entering a game number. The startBet2(User) supports the function of allowing the user to bet by entering two team names, one for the home team, and the other for the away team. The method checknumber(String) supports checking whether the user input is a positive number (including decimals) or not.

CreateFixture

The CreateFixture class reads the text file (pySoccer.txt) containing current season data of all the matched games line by line by the method ~~addFixtureFromFile(String)~~ makeSQLFixtureTable(String) which requires an input of filename and creates FIXTURE table in the local SQL database. As each line in the file represents a game, the class creates an object Fixture for each line and stores all the Fixture objects in an ArrayList<Fixture> fixtureList by using the ~~addFixtureToList(Fixture)~~ adddSQLToFixtureList() method which adds the input Fixture object to the ArrayList fixtureList. The makeFixToDBConn() method is used to connect the SQL database.

Fixture

The Fixture class stores data of the user-selected game. The home team name, away team name, and the winner of the matched game are stored as private string variables in this class. The methods getHomeTeam(), getAwayTeam(), and getWinner() are public getters for those three variables. The method toString() supports the functionality of printing out all the information of the selected game in one single string.

Decorator Pattern

The Decorator Pattern is implemented in the following four classes and one interface for adding features such as bet insurance and bet donation to the basic bet when the user bet on a game in SBP.

Bet

The Bet interface defines the fundamental behavior of a bet and is common to the concrete classes: the BasicBet class and the BetDecorator abstract class. The only method contained in this class is the bet(float) method, which requires a float number as a parameter and returns a float number as well.

BasicBet

The BasicBet class is the concrete class that implements the Bet interface and overrides the bet(float) method in the Bet interface. The bet(float) method returns 0 as a basic bet, which does not require any other additional fee.

BetDecorator

The BetDecorator abstract class implements the Bet interface and defines the basic behaviors of the decorators and features added to the basic bet, including bet insurance and bet donation. It stores a Bet object called decoratedbet when calling its constructor, and the Bet object is used in the override method bet(float).

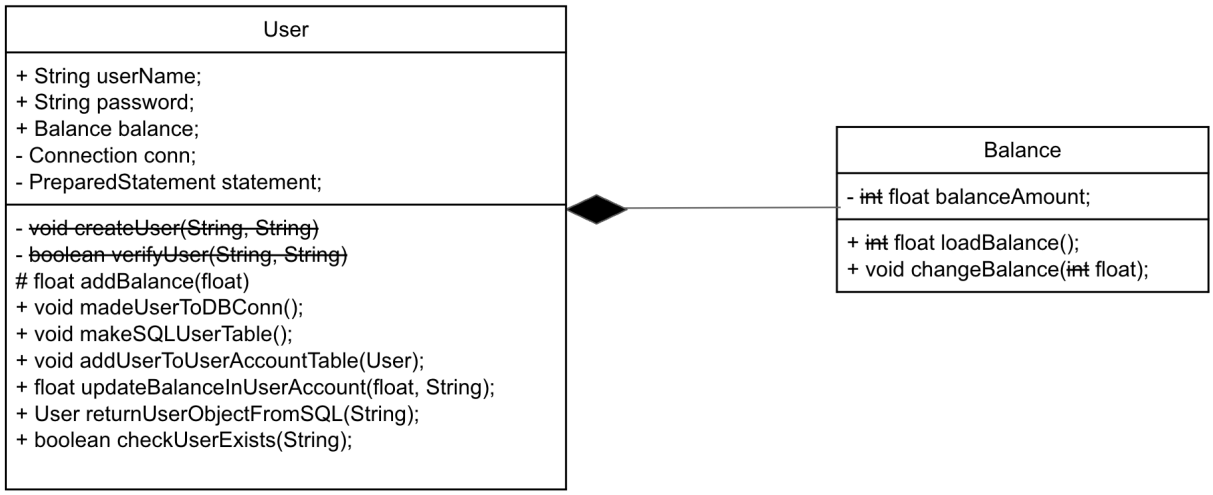
DonateDecorator

The DonateDecorator class is the concrete class that extends the BetDecorator abstract class. It contains its public constructor, the public override bet(float) method, and the private donate(float) method. Its constructor requires a Bet object as a parameter, which is used in the super() method contained in the constructor. The bet(float) method calls the donate(float) method to calculate the amount of donation based on the user's wager (5% of wager) and returns the float number the donate(float) method returns.

InsuranceDecorator

The InsuranceDecorator concrete class extends the BetDecorator abstract class. There are three methods contained in this class: the public constructor, the public override bet(float) method, and the private addInsurance(float) method. Its constructor requires a Bet object as a parameter, which is used in the super() method contained in the constructor. The bet(float) method calls the addInsurance(float) method to calculate the insurance fee based on the user's wager (5% of wager) and returns the float number the addInsurance(float) method returns.

User Component



Class Diagram

User

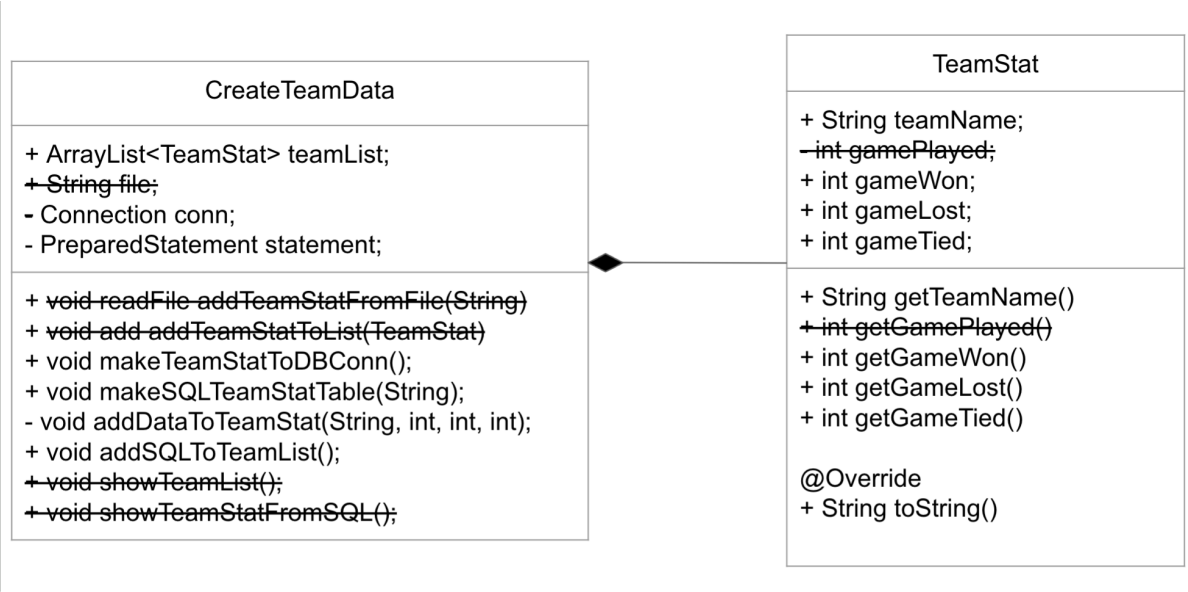
Creates a user object which defines a user by their username, password, and their balance. The userName and password are stored as string variables, and the balance is an instance of the

Balance class. The ~~addBalance(float)~~ updateBalanceInUsesrAccount (float, String) method is used to add money to the balanceAmount of the balance object stored in the User class by calling the changeBalance(float) method in the Balance class, and then this method updates the balance in the user's account stored in the userAccount table in the SQL database. The makeSQLUserTable() function connects the database. The addUserToUserAccount(User) method adds a new user to the userAccount table with the username and password. The checkUserExists(String) method is used to check whether the user account exists or not and returns the boolean value. The returnUserObjectFromSQL(String) method takes a String username as a parameter in order to check if the input password matches the existing account's corresponding password which is stored in the table in database. Then, it obtains the balance of the account if both the password and username are correct.

Balance

The Balance class is responsible for the user's balance management. It stores the current balance in the user account as a float variable. The loadBalance() method is the public getter method for obtaining the value of balance. The changeBalance(float) method supports the function of changing the balance by adding the input float number.

View Stats Component



Class Diagram

CreateTeamData

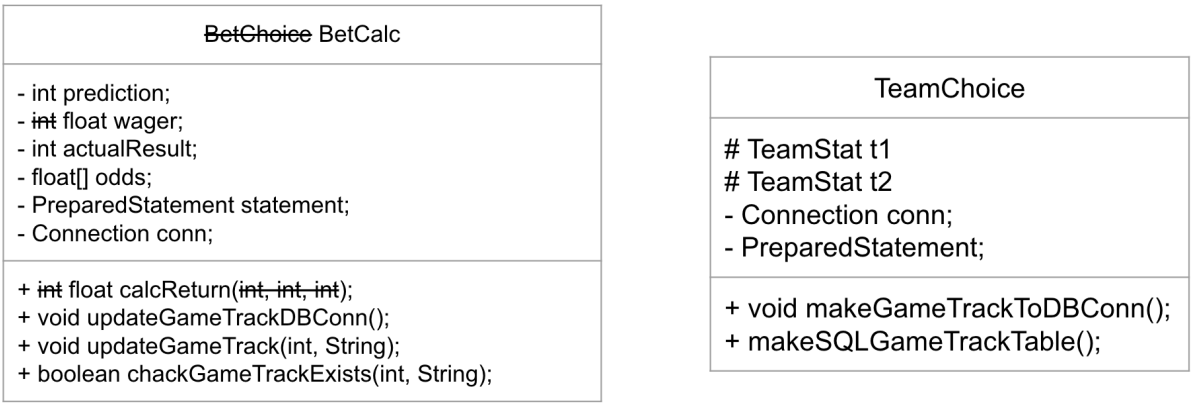
The CreateTeamData class reads the text file (pastSeason.txt) containing data of all the teams line by line by the method ~~addTeamStatFromFile(String)~~ makeSQLTeamStatTable(String) which requires an input of filename and creates a TEAMSTAT table in the local SQL database. As each line in the file represents a team, the class creates an object TeamStat for each line and stores all the TeamStat objects in an ArrayList<TeamStat> teamList by using the

~~addTeamStatToList(TeamStat)~~ addSQLToTeamList() method which adds the input TeamStat object to the ArrayList teamList. The makeTeamStatToDBConn() method is used to connect the SQL database.

TeamStat

The TeamStat class manages the team data by storing the information of a team such as teamName, gameWon, gameLost and gameTied. The getTeamName(), getGameTied(), getGameWon(), and getGameLost() methods are public getter methods that support the function of returning the private variables stored in the TeamStat class- including String teamName, int gameTied, int gameWon, and int gameLost. The toString method is also overridden, allowing the display of all of these variables in an easy-to-read format.

User selections Component



Class Diagram

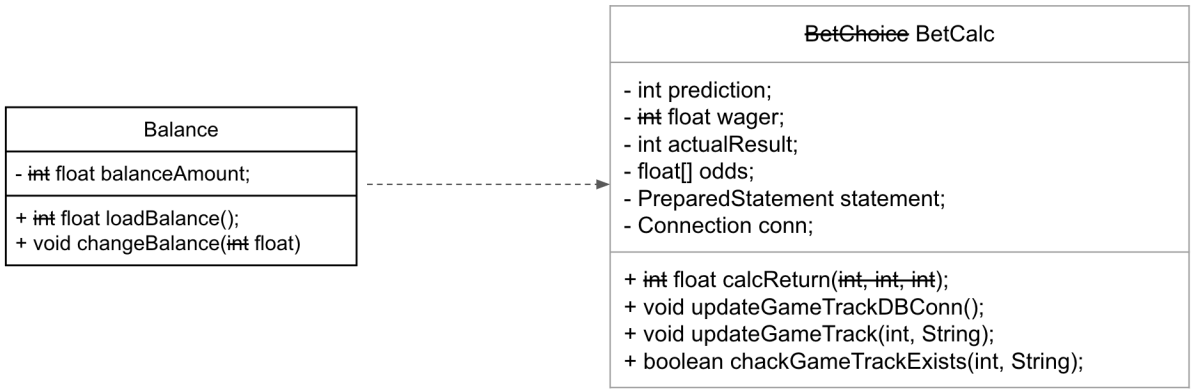
BetCalc

The BetCalc class takes in the user's prediction, their wager, the actualResult and the odds of each outcome to calculate the return for the user in the method calcReturn(). The updateGameTrackDBConn() method is used to connect the SQL database. The updateGameTrack() method supports the function of updating the GameTrack table in the database to store the game the user has already bet on.

TeamChoice

The TeamChoice class stores the user's two selected teams by creating two TeamStat objects, t1 and t2. These two objects can be accessed directly without getter methods since they are public. The makeGameTrackToDBConn() method is used to connect the SQL database. The makeSQLGameTrackTable() method creates the GameTrack table in the database.

Change/Update balance Component



Class Diagram

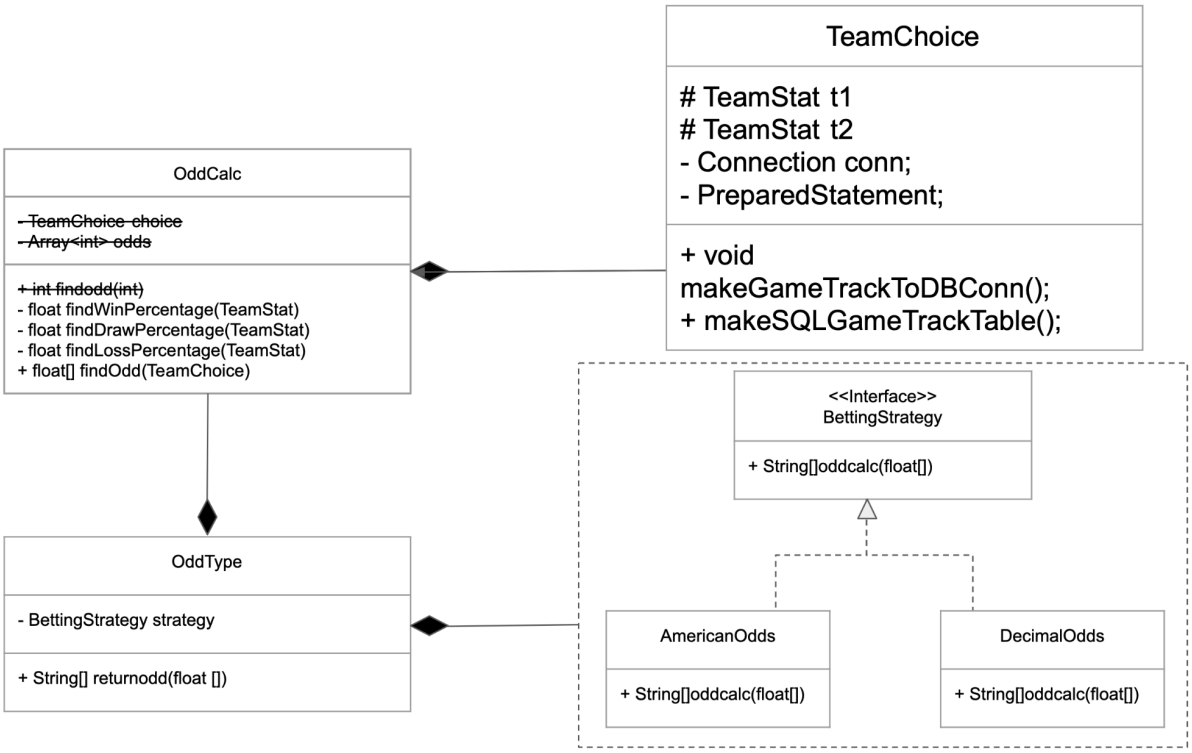
User

Creates a user object which defines a user by their username, password, and their balance. The userName and password are stored as string variables, and the balance is an instance of the Balance class. The ~~addBalance(float)~~ updateBalanceInUsesrAccount (float, String) method is used to add money to the balanceAmount of the balance object stored in the User class by calling the changeBalance(float) method in the Balance class, and then this method updates the balance in the user's account stored in the userAccount table in the SQL database. The makeSQLUserTable() function connects the database. The addUserToUserAccount(User) method adds a new user to the userAccount table with the username and password. The checkUserExists(String) method is used to check whether the user account exists or not and returns the boolean value. The returnUserObjectFromSQL(String) method takes a String username as a parameter in order to check if the input password matches the existing account's corresponding password which is stored in the table in database. Then, it obtains the balance of the account if both the password and username are correct.

Balance

The Balance class is responsible for the user's balance management. It stores the current balance in the user account as a float variable. The loadBalance() method is the public getter method for obtaining the value of balance. The changeBalance(float) method supports the function of changing the balance by adding the input float number.

Calculation Component



Class Diagram

oddCalc

This class contains the algorithm which calculates the odds for the three situations of the game result between the two teams that the user has selected: one team wins, the other team wins or there is a draw. These odds are stored by and can be accessed from an Array and has the method findOdd(TeamChoice) which takes in their choice of teams and produces the odds. The findOdd(TeamChoice) method calls on the private methods: findDrawPercentage(TeamStat), findWinPercentage(TeamStat), and findLossPercentage(TeamStat), in order to calculate the odds.

TeamChoice

The TeamChoice class stores the user's two selected teams by creating two TeamStat objects, t1 and t2. These two objects can be accessed directly without getter methods since they are public. The makeGameTrackToDBConn() method is used to connect the SQL database. The makeSQLGameTrackTable() method creates the GameTrack table in the database.

TeamStat

The TeamStat class manages the team data by storing the information of a team such as teamName, gameWon, gameLost and gameTied. The getTeamName(), getGameTied(), getGameWon(), and getGameLost() methods are public getter methods that support the function of returning the private variables stored in the TeamStat class- including String

teamName, int gameTied, int gameWon, and int gameLost. The toString method is also overridden, allowing the display of all of these variables in an easy-to-read format.

Strategy Pattern

The Strategy Pattern is implemented in the following three classes and one interface to support the functionality of allowing the user to select two different ways of odds calculation for a chosen game to bet on. It includes the American odds and the Decimal odds calculation.

Betting Strategy

The Betting Strategy interface is common to all concrete odds calculation strategies: American odds and Decimal odds. It declares a method oddcalc(float[]) asking for an array of floats and returning an array of strings.

Oddtype

The Oddtype class supports the function of choosing a specific odds calculation strategy by calling its constructor. It stores the selected strategy as an instance of the Betting Strategy class, called strategy, which is declared privately. The returnodd(float[]) method returns the calculated odds based on the specific strategy.

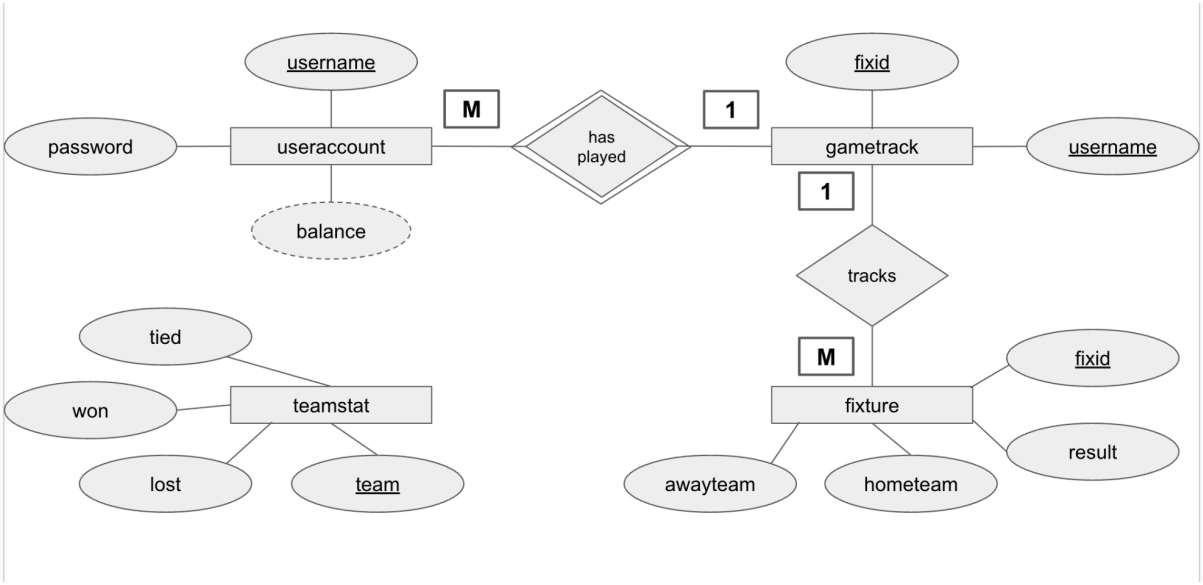
AmericanOdds

The AmericanOdds class implements the BettingStrategy class. It contains the override method oddcalc(float[]) to do the algorithm and calculation of the American odds.

DecimalOdds

The DeciamlOdds class implements the BettingStrategy class. It overrides the method oddcalc(float[]) to implement the algorithm of the Decimal odds.

Entity-Relationship Schema Diagram



Entity-Relationship Schema Diagram

Please see below for a more detailed description of the tables in the database.

Data Format

The two text files, pastSeason.txt and pySoccer.txt, which stores data of teams in the past season and all matched games in the current season respectively, are used to set up the database tables TeamStat and Fixture on the local computer initially when SBP is run and initialized.

Database Tables

SBP must store data in a MySQL database, in 4 different tables. In the visualisation below, the primary keys are underlined.

UserAccount

The account data for every user is stored in the MySQL database in a table named useraccount. The username is the primary key. Below is the structure of the table:

username	password	balance

GameTrack

GameTrack is a table which SBP uses to document which specific games the user has bet on, allowing SBP to prevent users from betting on the same game twice. The fixid and username is the composite primary key. User accesses which fixtures the user has bet on by the username. Below is the structure of the table:

fixid	username

TeamStat

The past season data is stored in a table named teamstat. It records all the results of games and contains team data of team name, number of games won, number of games lost and number of games tied in the 19/20 premier league season. Each row in the table represents a team. The primary key is team. Below is the structure of the table:

team	won	lost	tied

Fixture

The current season data is stored in a table called fixture. It contains the results of every game played between two teams in the 20/21 premier league season. It stores the fixture id, the home team, the away team and the result. The primary key is fixid. Below is the structure of the table:

fixid	hometeam	awayteam	result