Football Betting Game

Ingegneria del Software

Luca Leuter 6207353

May 2020

1 Introduzione

Lo scopo del progetto è creare un software per la simulazione di un sistema di scommesse per campionati di calcio. A tal proposito sono stati combinati tra di loro i design pattern Singleton, Builder e Observer. Il software permette scommesse del tipo "classico", cioè si può scommettere sulla vittoria della squadra di casa, sul pareggio o sulla vittoria della squadra ospite. Partendo da 10,00 euro l'obbiettivo e arrivare a guadagnare 1000,00 euro prima della scadenza delle giornate, che dipendono dal numero delle squadre (n di squadre - 1). Il gioco finisce quando non si ha soldi sufficienti per effettuare una scommessa (meno di 1,00 euro) oppure quando finiscono le giornate. Il progetto è stato versionato con Git ed è disponibile su GitHub al link https://github.com/Luca0079CM/ProgettoSWE

1.1 Design Pattern

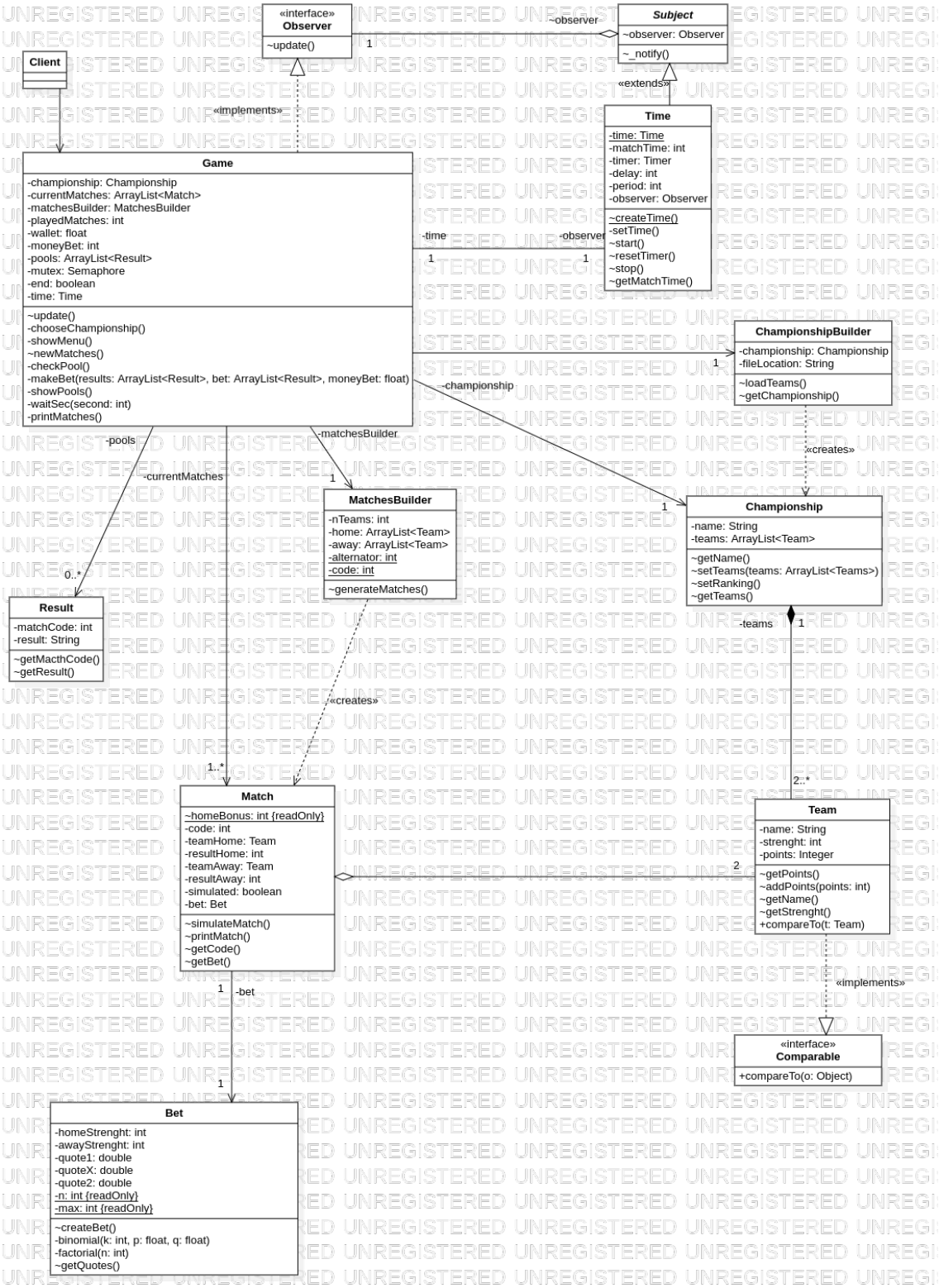
Vengono brevemente presentati i design pattern usati

- **Singleton**: Il singleton Pattern ha lo scopo di assicurarsi che una classe abbia solo un'istanza e fornire un punto di accesso globale a essa. Gli elementi che lo caratterizzano sono il costruttore privato, per evitare la creazione di oggetti da classi esterne e l'uso di un metodo statico, per accedere all'unica istanza dell'oggetto.

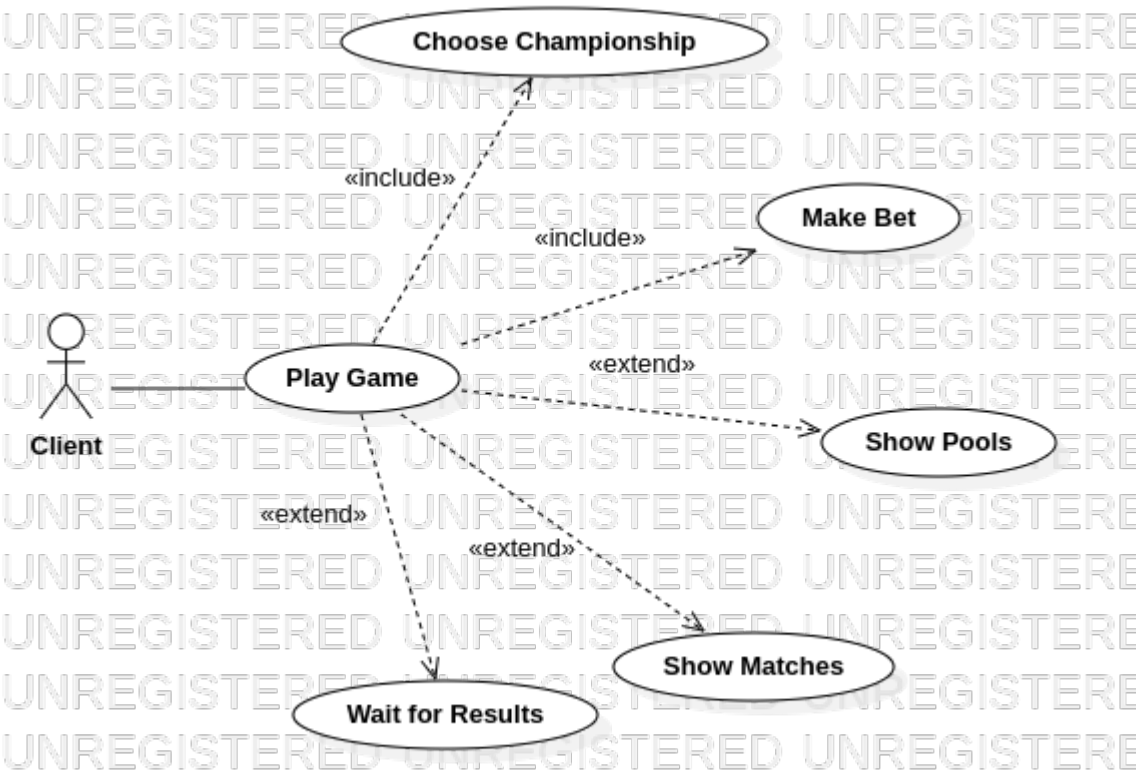
- **Builder**: Il design pattern Builder separa la costruzione di un oggetto dalla sua rappresentazione cosicché il processo di costruzione stesso possa creare diverse rappresentazioni. Serve quindi per separare l'algoritmo di costruzione di una struttura di oggetti dalla creazione dei singoli oggetti.

- **Observer**: Il design pattern Observer permette di definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato interno, ciascuno degli oggetti dipendenti da esso viene notificato e aggiornato automaticamente. L'Observer nasce dall'esigenza di mantenere un alto livello di consistenza fra classi correlate, senza produrre situazioni di forte dipendenza e di accoppiamento elevato.

1.2 Class Diagram:



1.3 Use Case Diagram:



2 Partecipanti:

Vengono presentate le classi del progetto, con una descrizione delle loro funzionalità.

- **Observer:** definisce un interfaccia implementata dagli Observer concreti. Per questa implementazione definisce solo il metodo UPDATE().
- **Subject:** è una classe astratta per rappresentare i Subject come descritto dal design pattern Observer. Per questa implementazione ha un solo un attributo Observer e il metodo NOTIFY().
- **Game:**La classe Game è la classe principale del progetto poiché gestisce l'interfaccia del gioco con l'utente. E' un observer concreto, perchè deve osservare la classe time (subject) in modo tale da far andare avanti i turni ogni 90 secondi. Inizialmente permette di scegliere il campionato con il metodo CHOOSECHAMPIONSHIP(), poi crea la prima giornata con il metodo NEWMATCHES() che verrà chiamato ogni 90 secondi dal metodo UPDATE(). L'observer pattern è stato implementato in modalità *push*(è l'observer a prendere il dato dal subject): ad ogni chiamata se il numero delle giornate non è finito e se il giocatore non ha finito i soldi controlla se il matchTime è scaduto, e in tal caso controlla la schedina giocata dall'utente e genera una nuova giornata. L'eventuale vincita è il prodotto tra i soldi scommessi dall'utente e le quote delle scommesse scelte dall'utente relative ai match.
- **Time:** è il subject del design pattern Observer, inoltre è implementato come singleton. Quando viene chiamato il metodo START() viene instanziato un *Timer* e un oggetto di tipo *Timer Task*, il quale ogni secondo esegue il metodo SETTIME() che a sua volta chiama NOTIFY().
- **Championship:** definisce la classe che contiene i vari team e che ordina la classifica in base ai punti con il metodo SETRANKING().
- **ChampionshipBuilder:** rappresenta il builder dei vari Championship, creati in base alla scelta iniziale dell'utente. Crea un campionato e carica i team dal file relativo alla scelta con il metodo LOADTEAMS(), che per ognuno setta il nome e dà dei punti forza in modo casuale tra 25 e 100.
- **Team:** rappresenta una squadra del campionato. Ad ognuna di esse sono associati un nome, dei punti forza e i punti per la classifica. Implementa l'interfaccia Comparable, in modo tale da poter permette l'ordinamento della classifica con il metodo COMPARETO(). Il metodo ADDPOINTS() aggiunge i punti in base al risultato.

- **Match:** rappresenta una partita tra due squadre. Utilizzando i parametri di forza delle due squadre, il metodo SIMULATEMATCH() genera il risultato, ritornando la stringa "1" se vince la squadra di casa, "X" per il pareggio e "2" se vince la squadra ospite. Per ogni squadra vengono fatti 10 tentativi di gol, ognuno dei quali ha una possibilità del $\frac{strenght \pm homeBonus}{300}$ di successo, dove il parametro *homeBonus* viene aggiunto o tolto in base a se la squadra sta giocando in casa oppure no. Ogni match genera anche un BET relativo.

- **MatchesBuilder:** rappresenta il builder dei match di una giornata di campionato. Per generare i match utilizza *l'algoritmo di Berger*: due squadre vengono abbinate in base a 2 criteri:

  - non sono ancora state abbinate
  - le restanti squadre, non selezionate, non siano a loro volta già state accoppiate

  In questo modo viene generato un calendario "all'italiana", dove ogni squadra fronteggia 1 volta tutte le altre del campionato.

- **Bet:** rappresenta una scommessa effettuabile relativa a un match. Utilizzando i parametri di forza delle due squadre calcola le quote delle vittorie e del pareggio. Il calcolo avviene utilizzando una distribuzione binomiale per ogni possibile risultato (da 0-0 a 10-10) e salvando poi ogni risultato in base alla tipologia (es. 2-2 fa parte dei risultati "X"). Si divide poi il parametro di guadagno (in questo caso 1.1) per la somma di ogni categoria, i risultati sono le quote.

- **Result:** rappresenta un risultato di un match o di una scommessa. Serve per controllare la schedina.

2.1 Codice:

vengono riportati frammenti di codice per ogni classe. Sono stati tralasciati gli import, i getter e setter e alcune funzioni di stampa

---

```
public class Game implements Observer {
    Game() {
        time = Time.createTime(this);
        pools = new ArrayList<>();
        wallet = 10;
        moneyBet = 0;
        chooseChampionship();
    }
    private void chooseChampionship(){
        //Lista campionati e numeri corrispondenti
        try {
            Scanner sc = new Scanner(System.in);
            c = sc.nextInt();
        } catch (Exception e) {
            System.out.println("Il numero inserito non corrisponde a
                nessun campionato");
            System.exit(0);
        }
        ChampionshipBuilder championshipBuilder;
        switch (c) {
            default:
                championshipBuilder = null;
                break;
            case 1:
                championshipBuilder = new
                    ChampionshipBuilder("./ChampionshipFiles/seriea",
                    "Serie A");
                break;
            //Casi per gli altri campionati
        if (championshipBuilder != null) {
            try {
                championshipBuilder.loadTeams();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
            championship = championshipBuilder.getChampionship();
            System.out.println("\nBene! Hai scelto il campionato
                "+championship.getName()+" a cui partecipano "
                   +championship.getTeams().size()+" squadre");
            championship.setRanking();
            waitSec(10);
            time.start();
            matchesBuilder = new MatchesBuilder(championship);
        } else {
            System.out.println("Il numero inserito non corrisponde a
```

```
            nessun campionato");
        System.exit(0);
    }
}
@Override
public void update() {
    int matchTime = time.getMatchTime();
    if(matchTime%30 == 0){
        System.out.println("Rimangono "+matchTime+" secondi
            all'inizio della prossima giornata!");
    }
    if(matchTime == 20){
        System.out.println("Rimangono solo 20 secondi all'inizio
            della prossima giornata!\nAffrettati a completare le tue
            operazioni, ricorda che non verranno conteggiate dopo la
            fine del tempo");
    }
    if (matchTime==1) {
        ArrayList<Result> results = new ArrayList<>();
        for (Match m : currentMatches) {
            results.add(new Result(m.getCode(), m.simulateMatch()));
            m.printMatch();
        }
        wallet += checkPools(results, pools, moneyBet);
        playedMatches++;
        championship.setRanking();
        moneyBet = 0;
        pools.clear();
        if (playedMatches == championship.getTeams().size() - 1) {
            System.out.println("---------FINE CAMPIONATO----------");
            System.out.println("Classifica finale:");
            championship.setRanking();
            if(wallet >=1000)
                System.out.println("COMPLIMENTI! Hai raggiunto
                    l'obbiettivo");
            else
                System.out.println("Peccato, sarai pi fortunato la
                    prossima volta");
            System.out.println("\nSoldi nel
                portafoglio:"+df.format(wallet));
            time.stop();
            end = true;
            mutex.release();
        }else if(wallet < 1){
            System.out.println("-----HAI FINITO I SOLDI-----");
            time.stop();
            end = true;
            mutex.release();
        }else {
            waitSec(7);
```

```
            newMatches();
            time.resetTimer();
            mutex.release();
        }
    }
}
private void showMenu() throws InterruptedException{
    while (!end) {
        try {
            Scanner input = new Scanner(System.in);
            System.out.println("\nPortafoglio:"+df.format(wallet)+"
                euro");
            System.out.println("Che operazione vuoi fare? (Digita il
                numero corrispondente)");
            System.out.println("1-Vedi i match della giornata n" +
                (playedMatches + 1));
            System.out.println("2-Scommetti su un match della
                giornata");
            System.out.println("3-Vedi la schedina");
            System.out.println("4-Attendi i match");
            System.out.println("5-Inserisci l'importo da
                scommettere");
            if(!pools.isEmpty() && moneyBet==0){
                System.out.println("\nATTENZIONE sono presenti
                    scommesse ma non l'importo, premi 5 per
                    inserirlo");
            }
            int c = input.nextInt();
            switch (c) {
                default:
                    System.out.println("\nNon c' nessun operazione
                        corrispondente al numero inserito");
                    break;
                case 1:
                    printMatches();
                    break;
                case 2:
                    makeBet();
                    break;
                case 3:
                    showPools();
                    break;
                case 4:
                    System.out.println("Ok! Attendi il prossimo
                        turno");
                    mutex.acquire();
                    break;
                case 5:
                    if(moneyBet != 0){
                        System.out.println("\nHai gi inserito la
```

```
                        scommessa");
                    break;
                }
                System.out.println("\nSono accettati solo importi
                    interi");
                Scanner sc = new Scanner(System.in);
                int tmp1 = sc.nextInt();
                if(tmp1 > 0 && tmp1 <= wallet){
                    moneyBet = tmp1;
                    wallet -= moneyBet;
                }else{
                    System.out.println("L'importo inserito non
                        valido");
                }
                break;
            }
        } catch (NoSuchElementException e) {
            System.out.println("\nOperazione non valida");
        }
    }
}
void newMatches() {
    currentMatches = matchesBuilder.generateMatches();
    printMatches();
    try {
        if(playedMatches==0){
            showMenu();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
 private float checkPool(ArrayList<Result> results,
     ArrayList<Result> bets, float moneyBet){
    boolean win = true;
    if(bets.size()==0){
        System.out.println("\nNon avevi scommesso niente");
        return 0;
    }
    for (Result b : bets) {
        for(Result r : results){
            if(b.getMatchCode() == r.getMatchCode() &&
                !b.getResult().equals(r.getResult())) {
                win = false;
                System.out.println("\nPeccato, non hai vinto");
                break;
            }
        }
    }
    if(win){
```

```
        for (Result b : bets) {
            for(Match m : currentMatches){
                if(b.getMatchCode() == m.getCode()) {
                    String s = b.getResult();
                    switch (s) {
                        case "1":
                            moneyBet *= m.getBet().getQuotes()[0];
                            break;
                        case "X":
                            moneyBet *= m.getBet().getQuotes()[1];
                        case "2":
                            moneyBet *= m.getBet().getQuotes()[2];
                    }
                }
            }
        }
        System.out.println("\nComplimenti!! Hai vinto
            "+df.format(moneyBet)+" euro");
    }else
        moneyBet = 0;

    return moneyBet;
}
private void makeBet(){
    System.out.println("Inserisci il codice della partita su cui
        vuoi scommettere:");
    Scanner sc = new Scanner(System.in);
    int code = sc.nextInt();
    boolean found = false;
    for (Match m : currentMatches){
        if (code == m.getCode()){
            found = true;
            System.out.println("Match trovato! Inserisci la
                scommessa:");
            System.out.println("1-Vittoria della squadra di casa");
            System.out.println("X-Pareggio");
            System.out.println("2-Vittoria della squadra ospite");
            break;
        }
    }
    if (!found){
        System.out.println("Il codice inserito non corrisponde a
            nessun match della giornata, riprova");
    }else{
        String s = sc.next();
        boolean alreadyBet = false;
        if (s.equals("1")||s.equals("X")||s.equals("2")){
            for(Result b : pools){
                if(code == b.getMatchCode() &&
                    s.equals(b.getResult())){
```

```
                    alreadyBet = true;
                    break;
                }
            }
            if(alreadyBet) {
                System.out.println("La scommessa inserita  gi presente
                    nella schedina");
            }else {
                pools.add(new Result(code, s));
            }
        }else {
            System.out.println("Scommessa non valida!");
        }
    }
}
private void showPools(){
    float tmp = 1;
    if (!pools.isEmpty()) {
        System.out.println("--Schedina--");
        for(Result b : pools) {
            for(Match m : currentMatches){
                if(m.getCode() == b.getMatchCode()) {
                    System.out.println();
                    m.printMatch();
                    System.out.println("Scommessa: "+b.getResult());
                    if(b.getResult().equals("1")){
                        tmp *= m.getBet().getQuotes()[0];
                    }else if(b.getResult().equals("X")){
                        tmp *= m.getBet().getQuotes()[1];
                    }else{
                        tmp *= m.getBet().getQuotes()[2];
                    }
                }
            }
        }
        System.out.println("\nScommessi "+df.format(moneyBet)+"
            euro");
        System.out.println("Potenziale vincita
            "+df.format(moneyBet*tmp)+" euro");
    } else {
        System.out.println("La schedina  vuota");
    }

}
private void waitSec(int seconds){
    try {
        Thread.sleep(seconds*1000);
    }catch (InterruptedException e){
        e.printStackTrace();
    }
```

```
    }
    private void printMatches(){
        System.out.println("\nMatches:");
        for(Match m : currentMatches)
            m.printMatch();
        System.out.println("\n");
    }
}
```

```
class Championship {
    private String name;
    private ArrayList<Team> teams;

    Championship(String name){
        this.name = name;
    }
    void setRanking(){
        teams.sort(Comparator.reverseOrder());
        System.out.println("\nSquadre del campionato:");
        for (Team t : teams)
            System.out.println(t.getName()+"-forza:"+t.getStrength()+"-punti:"+t.getPoints());
    }
}
```

```
class ChampionshipBuilder {
    private Championship championship;
    private String fileLocation;
    ChampionshipBuilder(String fileLocation, String name){
        this.fileLocation = fileLocation;
        this.championship = new Championship(name);
    }
    void loadTeams() throws FileNotFoundException{
        Scanner scanner = new Scanner(new File(fileLocation));
        ArrayList<Team> teams = new ArrayList<>();
        while(scanner.hasNextLine()){
            String line = scanner.nextLine();
            Team t = new Team(line, (int)(Math.random()*((100 -
                25)+1)+25));
            teams.add(t);
        }
        championship.setTeams(teams);
    }
}
```

```
class Match {
    static final int homeBonus = 10;
    private int code;
```

```
    private Team home;
    private int resultHome;
    private Team away;
    private int resultAway;
    private boolean simulated;
    private Bet bet;
    Match(int code, Team home, Team away){
        this.code = code;
        this.home = home;
        this.away = away;
        simulated = false;
        bet = new Bet(home.getStrength(), away.getStrength());
    }
    String simulateMatch(){
        simulated = true;
        for(int i=0;i<10;i++){
            if(home.getStrength() + homeBonus >=Math.random()*300)
                resultHome++;
            if(away.getStrength() - homeBonus >=Math.random()*300)
                resultAway++;
        }
        if(resultHome>resultAway) {
            home.addPoints(3);
            return "1";
        }
        else if(resultHome<resultAway) {
            away.addPoints(3);
            return "2";
        }
        else {
            home.addPoints(1);
            away.addPoints(1);
            return "X";
        }
    }
}
```

```
class MatchesBuilder {
    private int nTeams;
    private ArrayList<Team> home;
    private ArrayList<Team> away;
    private static int alternator = 0;
    private static int code = 0;

    MatchesBuilder(Championship championship){
        ArrayList<Team> teams = championship.getTeams();
        nTeams = teams.size();
        Collections.shuffle(teams);
        home = new ArrayList<>();
```

```
        away = new ArrayList<>();
        for (int i = 0; i < nTeams /2; i++) {
            home.add(teams.get(i));
            away.add(teams.get(teams.size()-1-i));
        }
    }

    //Algoritmo di Berger
    ArrayList<Match> generateMatches(){
        ArrayList<Match> matches = new ArrayList<>();
        if (alternator % 2 == 1) {
            for (int j = 0; j < nTeams /2 ; j++)
                matches.add(new Match(++code, away.get(j), home.get(j)));
        }else {
            for (int j = 0; j < nTeams /2 ; j++)
                matches.add(new Match(++code, home.get(j), away.get(j)));

        }
        away.add(0, home.get(1));
        Team riporto = away.remove(away.size()-1);
        home.remove(1);
        home.add(riporto);
        alternator++;
        return matches;
    }
}
```

```
class Result {
    private int matchCode;
    private String result;
    Result(int matchCode, String result){
        this.matchCode = matchCode;
        this.result = result;
    }
}
```

```
public class Team implements Comparable <Team> {
    private String name;
    private int strength;
    private Integer points;

    Team(String name, int strength){
        this.name = name;
        this.strength = strength;
        this.points = 0;
    }
    @Override
    public int compareTo(Team t) {
```

```
        return this.getPoints().compareTo(t.getPoints());
    }
    void addPoints(int p){
        points += p;
    }
}
```

```
class Time extends Subject {
    private static Time time;
    private static int matchTime;
    private static Timer timer;
    private static int delay;
    private static int period;

    static Time createTime(Observer observer){
        if(time==null)
            time = new Time(observer);
        return time;
    }
    private Time(Observer observer){
        this.observer = observer;
        delay = 1000;
        period = 1000;
        resetTimer();
    }
    private void setTime() {
        _notify();
        matchTime--;
    }
    void start(){
        timer = new Timer();
        TimerTask timerTask = new TimerTask() {
            @Override
            public void run() {
                setTime();
            }
        };
        timer.scheduleAtFixedRate(timerTask, delay, period);
    }
    void resetTimer(){
        matchTime = 91;
    }
    void stop() {
        timer.cancel();
    }
}
```

```
class Bet {
```

```
private int homeStrenght;
private int awayStrenght;
private double quote1, quoteX, quote2;
private static final int n = 10;
private static final int max = 300;

Bet(int homeStrenght, int awayStrenght){
    this.homeStrenght = homeStrenght;
    this.awayStrenght = awayStrenght;
    createBet();
}

private void createBet() {
    float pHome = (float)(homeStrenght+Match.homeBonus)/max;
    float pAway = (float)(awayStrenght+Match.homeBonus)/max;
    float qHome = 1 - pHome;
    float qAway = 1 - pAway;

    float[] probCasa = new float[n+1];
    float[] probTras = new float[n+1];
    float prob1 = 0;
    float probX = 0;
    float prob2 = 0;


    for(int i=0; i<=n; i++) {
        probCasa[i] = binomial(i, pHome, qHome);
        probTras[i] = binomial(i,pAway,qAway);
    }

    for (int i=0; i<=n; i++){
        for(int j=0; j<=n; j++){
            if(i>j) {
                prob1 += probCasa[i] * probTras[j];
            }else if(i<j){
                prob2 += probCasa[i]*probTras[j];
            }else{
                probX += probCasa[i]*probTras[j];
            }
        }
    }
    quote1 = 1.1/prob1;
    quoteX = 1.1/probX;
    quote2 = 1.1/prob2;
}

private float binomial(int k, float p, float q) {
    float prob= (float)(factorial(n) / (factorial(k)*
        factorial(n-k)) );
    for(int i=0;i<k;i++) {
```

```
            prob *= p;
        }
        for(int i=0;i<(n-k);i++) {
            prob *= q;
        }
        return prob;
    }

    private int factorial(int n) {
        if(n==0)
            return 1;
        else
            return(n* factorial(n-1));
    }

    double[] getQuotes(){
        double[] tmp = new double[3];
        tmp[0] = quote1;
        tmp[1] = quoteX;
        tmp[2] = quote2;
        return tmp;
    }
}
```

2.2 Test:

Per i test è stato usato il framework di unit testing **Junit 5**. E' stato effettuato il test del funzionamento delle varie classi e in particolare del funzionamento dei vari design pattern utilizzati.

```
class BetTest {
    private Bet testBet;

    @BeforeEach
    void setTestBet(){
        testBet = new Bet(80, 40);
    }

    @Test
    void testGetQuotes() {
        assertTrue(testBet.getQuotes()[0] > 1.6 &&
            testBet.getQuotes()[0] < 1.7);
        assertTrue( testBet.getQuotes()[1] > 6.4 &&
            testBet.getQuotes()[1] < 6.5 );
        assertTrue(testBet.getQuotes()[2] > 7.5 &&
            testBet.getQuotes()[2] < 7.6);
    }
}
```

```
class ChampionshipBuilderTest {
    private ChampionshipBuilder testChampionshipBuilder;

    @BeforeEach
    void setTestChampionshipFactory(){
        testChampionshipBuilder = new
            ChampionshipBuilder("./ChampionshipFiles/seriea", "Serie A");
        try {
            testChampionshipBuilder.loadTeams();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    @Test
    void testLoadTeams() throws FileNotFoundException {
        Championship testChampionship = new Championship("Serie A");
        Scanner scanner = new Scanner(new
            File("./ChampionshipFiles/seriea"));
        ArrayList<Team> testTeams = new ArrayList<>();
        while(scanner.hasNextLine()){
            String line = scanner.nextLine();
            Team t = new Team(line, 100);
            testTeams.add(t);
```

```
        }
        testChampionship.setTeams(testTeams);

        for(int i = 0; i<testTeams.size(); i++){
            assertEquals(testTeams.get(i).getName(),
                testChampionshipBuilder.getChampionship().getTeams().get(i).getName());
        }
    }

    @Test
    void getChampionship() {
        assertEquals("Serie A",
            testChampionshipBuilder.getChampionship().getName());
    }

    @Test
    void testEveryTeamHasLessThanMaxStrenght(){
        int max = 100;
        for(Team t :
            testChampionshipBuilder.getChampionship().getTeams())
            assertTrue(t.getStrength() <= max);
    }



    @Test
    void testFailingLoad() {
        ChampionshipBuilder testChampionshipBuilder2 = new
            ChampionshipBuilder("./Championship/nothing", " ");
        Throwable exception = assertThrows(Exception.class,
            testChampionshipBuilder2::loadTeams);
        assertEquals("./Championship/nothing (File o directory non
            esistente)", exception.getMessage());
    }
}
```

```
class ChampionshipTest {
    private Championship testChampionship;
    private ArrayList<Team> testTeamArrayList;

    @BeforeEach
    void setChampionshipTest(){
        testChampionship = new Championship("Test Championship");
        testTeamArrayList = new ArrayList<>();
        for(int i = 0; i<10; i++){
            testTeamArrayList.add(new Team("Team "+i, i+1));
        }
    }
    @Test
```

```
    void getName() {
        assertEquals("Test Championship", testChampionship.getName());
    }
    @Test
    void getTeams() {
        int i = 0;
        for(Team t : testTeamArrayList) {
            assertEquals("Team "+i, t.getName());
            assertEquals(i+1, t.getStrength());
            i++;
        }
    }
}
```

```
class MatchTest {
    private Match testMatch;

    @BeforeEach
    void setTestMatch(){
        Team testTeam1 = new Team("Test Team 1", 100);
        Team testTeam2 = new Team("Test Team 2", 50);
        testMatch = new Match(1, testTeam1, testTeam2);
    }
    @Test
    void simulateMatch() {
        assertTrue(testMatch.simulateMatch().equals("1") ||
                testMatch.simulateMatch().equals("2")||
                testMatch.simulateMatch().equals("X"));
    }
    @Test
    void getCode() {
        assertEquals(1, testMatch.getCode());
    }
    @Test
    void getBet() {
        Bet testBet = new Bet(100 ,50);
        assertEquals(testBet.getQuotes()[0],
            testMatch.getBet().getQuotes()[0]);
        assertEquals(testBet.getQuotes()[1],
            testMatch.getBet().getQuotes()[1]);
        assertEquals(testBet.getQuotes()[2],
            testMatch.getBet().getQuotes()[2]);
    }
}
```

```
class ResultTest {
    private Result testResult;
```

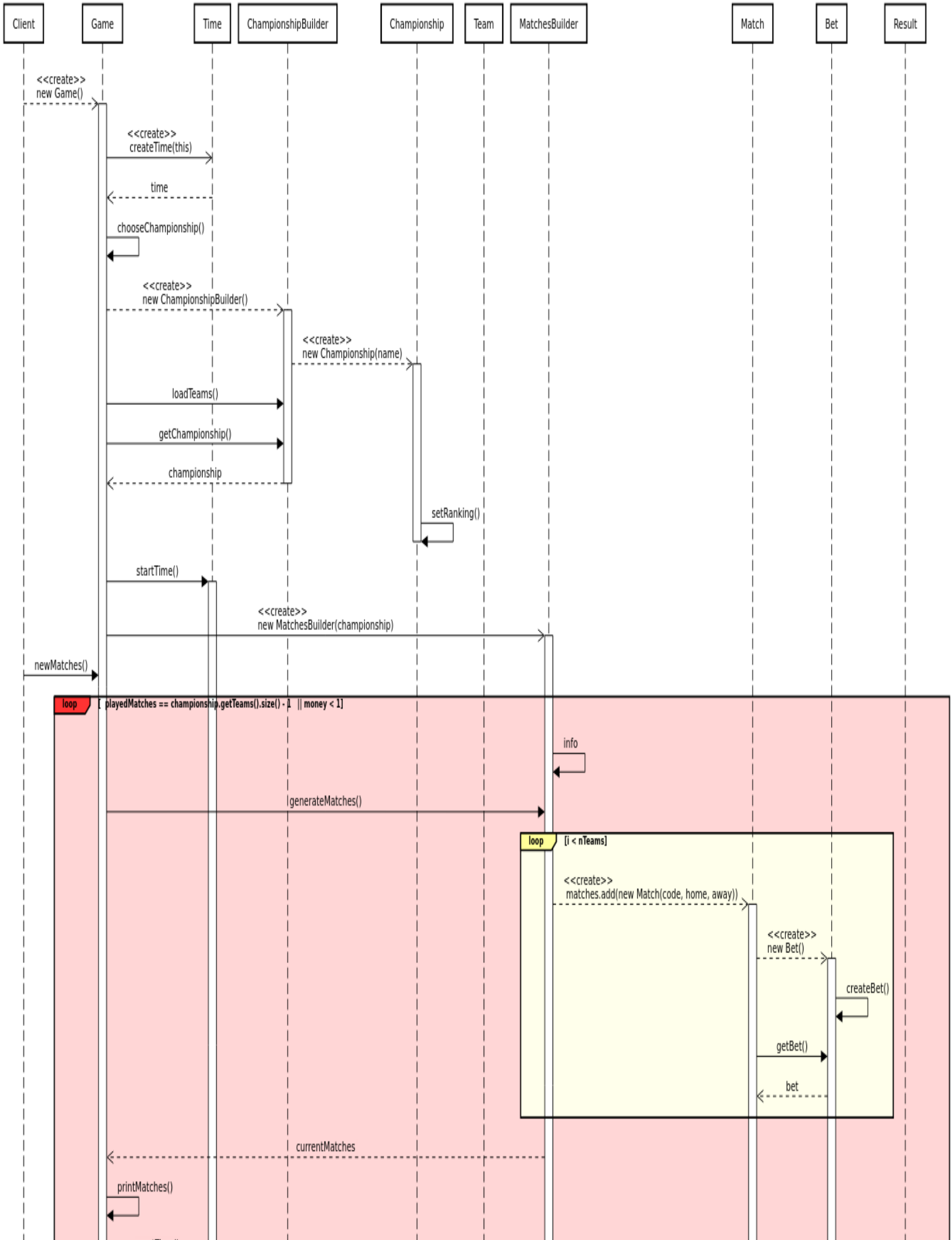
```
    @BeforeEach
    void setTestResult(){
        testResult = new Result(1, "X");
    }
    @Test
    void getMatchCode() {
        assertEquals(1, testResult.getMatchCode());
    }
    @Test
    void getResult() {
        assertEquals("X", testResult.getResult());
    }
}
```

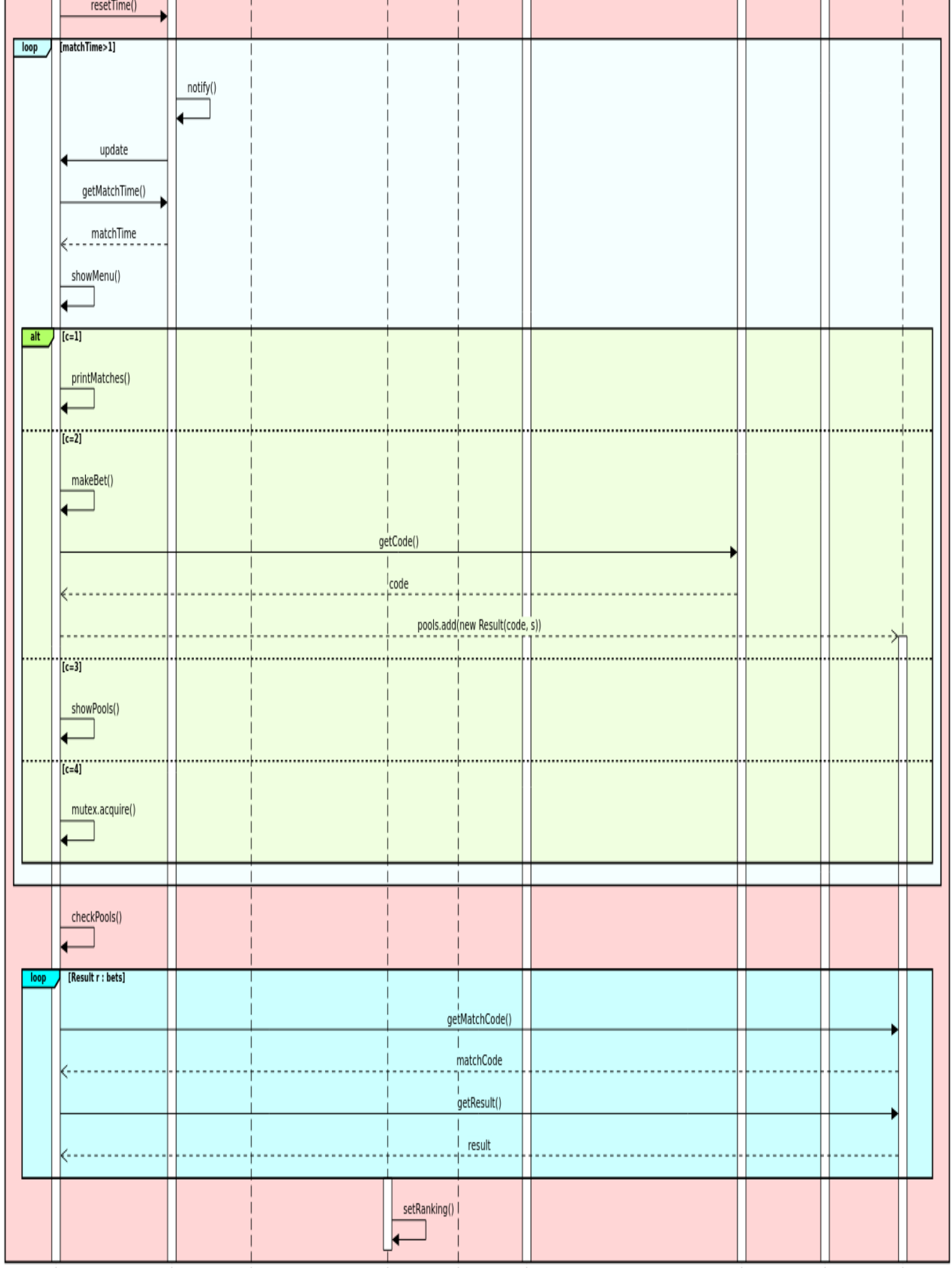
```
class TeamTest {
    private Team testTeam;

    @BeforeEach
    void setTeam(){
        testTeam = new Team("Test Team", 100);
    }
    @Test
    void compareTo() {
        Team testTeam2 = new Team("Test Team 2", 50);
        testTeam.addPoints(3);
        assertTrue(testTeam.compareTo(testTeam2)>0);
        assertTrue(testTeam2.compareTo(testTeam)<0);
        testTeam2.addPoints(3);
        assertEquals(0, testTeam.compareTo(testTeam2));
    }
    @Test
    void getPoints() {
        assertEquals(0, testTeam.getPoints());
    }
    @Test
    void addPoints() {
        int tmp = testTeam.getPoints();
        testTeam.addPoints(3);
        assertEquals(tmp + 3, testTeam.getPoints());
    }
    @Test
    void getName() {
        assertEquals("Test Team", testTeam.getName());
    }
    @Test
    void getStrength() {
        assertEquals(100, testTeam.getStrength());
    }
}
```

```
class TimeTest {
    class TestObserver implements Observer{
        boolean called = false;
        @Override
        public void update() {
            called = true;
        }
    }

    private TestObserver testObserver = new TestObserver();
    private Time testTime;

    @BeforeEach
    void createTimer() {
        testTime = Time.createTime(testObserver);
        testTime.start();
    }
    @Test
    void testSingleton(){
        Time testTime2 = Time.createTime(testObserver);
        assertEquals(testTime, testTime2);
    }
    @Test
    void resetTimer() {
        testTime.resetTimer();
        assertEquals(91, testTime.getMatchTime());
    }
    @Test
    void testCallUpdate(){
        testTime.observer = testObserver;
        testTime._notify();
        assertTrue(testObserver.called);
    }
}
```

3 Sequence Diagram:

Nelle seguenti 2 pagine è rappresentato un esempio con un sequence diagram


Client
Game
Time
ChampionshipBuilder
Championship
Team
MatchesBuilder
Match
Bet
Result
<<create>>
new Game()
<<create>>
createTime(this)
time
chooseChampionship()
<<create>>
new ChampionshipBuilder()
<<create>>
new Championship(name)
loadTeams()
getChampionship()
championship
setRanking()
startTime()
<<create>>
new MatchesBuilder(championship)
newMatches()
loop
[playedMatches == championship.getTeams().size() - 1 || money < 1]
info
generateMatches()
loop
[i < nTeams]
<<create>>
matches.add(new Match(code, home, away))
<<create>>
new Bet()
createBet()
getBet()
bet
currentMatches
printMatches()


resetTime()
loop
[matchTime>1]
notify()
update
getMatchTime()
matchTime
showMenu()
alt
[c=1]
printMatches()
[c=2]
makeBet()
getCode()
code
pools.add(new Result(code, s))
[c=3]
showPools()
[c=4]
mutex.acquire()
checkPools()
loop
[Result r : bets]
getMatchCode()
matchCode
getResult()
result
setRanking()