МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Курсовая работа

по дисциплине: «ООП»

Выполнил: ст. группы ПВ-211

Чувилко Илья Романович

Проверил:

Буханов Дмитрий Геннадьевич

Харитонов Сергей Дмитриевич

Белгород 2023 г.

Тема: Реализация моделирования футбольных матчей
Цель: разработке и реализации моделирования спортивной игры (футбол) с использованием принципов ООП.

Постановка задачи:

В рамках данной курсовой работы разрабатывается программа моделирования спортивной игры, а именно футбола. Цель программы заключается в создании среды, способной учитывать игроков различных команд, проводить моделирование матчей между командами и регистрировать результаты этих матчей в различных турнирах.

Каждый игрок представляется в программе с набором атрибутов, которые описывают его характеристики и влияют на его поведение во время матча. Эти атрибуты могут включать такие параметры, как скорость, точность удара, выносливость. Учет этих атрибутов позволяет программе создавать реалистичные сценарии игры, где каждый игрок демонстрирует свои индивидуальные особенности.

Основной функционал программы включает возможность создания и редактирования команд, добавление новых игроков с указанием их атрибутов, а также проведение матчей между командами. При моделировании матчей программа учитывает значения атрибутов игроков. Результаты матчей фиксируются и сохраняются, что позволяет программе проводить турниры и отслеживать статистику команд и игроков.

Таким образом, разрабатываемая программа моделирования спортивной игры футбол предоставляет среду для проведения игр, позволяя учитывать индивидуальные навыки игроков и проводить турниры.

Обзор предметной области:

1. Создание игроков:
    - Случайное создание игрока с заданными характеристиками (скорость, точность удара, выносливость) и случайным именем.
    - Случайное создание игрока с заданными характеристиками и заданным именем.
    - Случайное создание игрока с заданными характеристиками в заданном диапазоне и случайным именем.
    - Случайное создание игрока с заданными характеристиками в заданном диапазоне и заданным именем.
2. Создание команд:
    - Создание команды с заданным именем и случайными игроками.
    - Добавление игрока в команду.
3. Создание турнира:
    - Создание турнира с заданным уровнем сложности (название, диапазон характеристик игроков, призовые).
    - Добавление команды в турнир.
    - Создание расписания матчей для турнира.
4. Проведение матчей:
    - Расчет результатов матча на основе характеристик игроков.
    - Подсчет очков для команд в зависимости от результатов матчей.
5. Предстоящие турниры на стадионе:
    - Просмотр доступных турниров на стадионе.
    - Вывод информации о командах в турнире (средние характеристики игроков, результаты матчей, очки).
    - Вывод доступных турниров со своими характеристиками и призовыми.
6. Управление командой игрока:
    - Создание команды игрока с заданными игроками.
    - Просмотр и изменение имени команды игрока.
    - Просмотр и изменение количества денег игрока.
    - Просмотр и изменение количества выигранных и проигранных игр игрока.

Используемые паттерны проектирования:

В работе применен паттерн проектирования **"Одиночка" (Singleton)**. Он реализован следующим образом:

1. В классе `UserInteraction` присутствует статический указатель `instance` на единственный экземпляр класса.
2. Конструктор `UserInteraction` объявлен как приватный, чтобы предотвратить прямое создание объектов класса извне.
3. Метод `getInstance()` является статическим и предоставляет доступ к единственному экземпляру класса. Если экземпляр еще не создан, метод создает новый экземпляр и возвращает его, иначе возвращает существующий экземпляр.
4. В функции `main()` вызывается `UserInteraction::getInstance()` для получения единственного экземпляра класса, и через него вызываются различные методы для работы с пользовательским интерфейсом.

Паттерн "Одиночка" применяется, когда требуется иметь только один экземпляр класса, который обеспечивает доступ к своим методам и данным. В данном случае класс `UserInteraction` является синглтоном, предоставляющим доступ к различным меню и функциональности пользовательского взаимодействия в игре.

Фабричный метод (Factory Method) является порождающим паттерном проектирования, который определяет интерфейс для создания объекта, но позволяет подклассам решать, какой класс инстанцировать. В представленном коде фабричный метод может быть использован в статических методах класса `Team и PlayerTeam`, которые могут создать как объект `Team` для создания команд противников, так и `PlayerTeam` для создания команды игрока с отдельными характеристиками.

Наблюдатель (Observer): В коде неявно используется паттерн Наблюдатель. Например, в методе `playMatches` класса `Tournament` после каждого матча результат добавляется в вектор `results`. Это позволяет другим частям программы быть в курсе результатов матчей и предпринимать соответствующие действия.

Описание каждого класса:

1. Класс `RandomNumberGenerator` представляет генератор случайных чисел. Он используется для генерации случайных характеристик игроков и случайных чисел во время матчей.

2. Класс `Player` представляет игрока. Он имеет атрибуты, такие как имя, скорость, точность удара, выносливость и цена. Класс содержит методы для генерации случайного игрока и вывода информации о нем.

3. Класс `Team` представляет команду. Он имеет атрибуты, такие как имя команды, список игроков и результаты матчей. Класс содержит методы для добавления игрока в команду, генерации случайной команды и вывода средних показателей игроков.

4. Класс `Match` представляет матч между двумя командами. Он имеет атрибуты, такие как домашняя команда, гостевая команда, счет и результат. Класс содержит методы для расчета результата матча.

5. Класс `Tournament` представляет турнир. Он имеет атрибуты, такие как имя, список команд, список матчей и результаты. Класс содержит методы для добавления команды в турнир, создания расписания матчей, проведения матчей и генерации случайных команд.

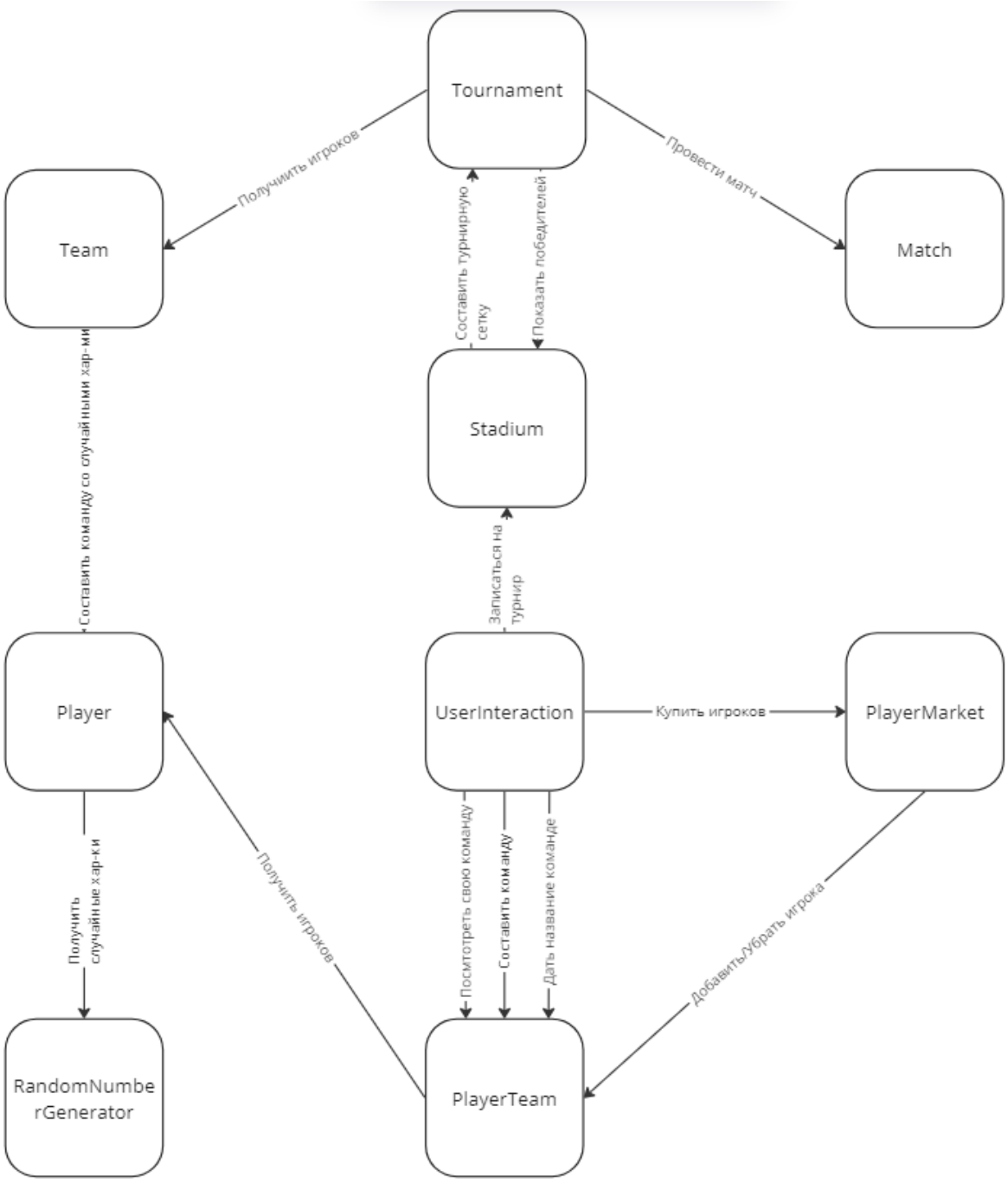
6. Класс `Stadium` представляет стадион. Он имеет атрибуты, такие как вместимость и список турниров. Класс содержит методы для получения списка доступных турниров.

7. Класс `PlayerTeam` представляет команду игрока. Он наследует класс `Team` и добавляет дополнительные атрибуты, такие как деньги, количество выигранных и проигранных игр. Класс содержит методы для создания случайной команды игрока.

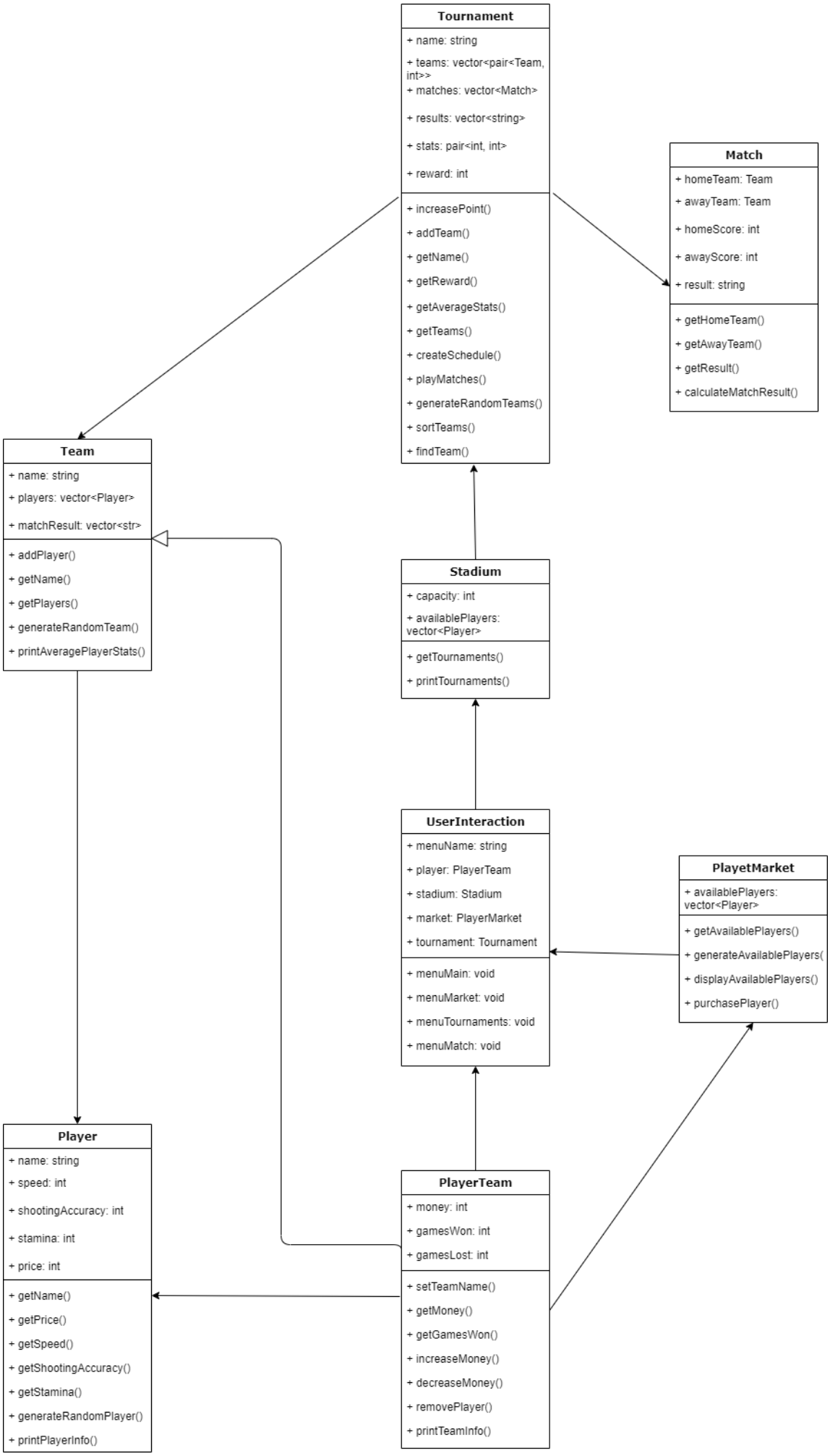
8. Класс `PlayerMarket` представляет рынок игроков, где игроки могут быть куплены и проданы. Он отвечает за управление доступными игроками и проведение торговых операций

9. Класс `UserInteraction` представляет пользовательский интерфейс для взаимодействия с игровым приложением. Он отвечает за отображение информации и обработку пользовательских действий.

Объектная декомпозиция задачи



UML-Диаграмма



Код программы:

```
#include <iostream>
#include <string>
#include <utility>
#include <vector>
#include <cstdlib> // Для генерации случайных чисел
#include <ctime> // Для инициализации генератора случайных чисел
#include <map>
#include <valarray>
#include "windows.h"


// Класс Генератор случайных чисел
class RandomNumberGenerator {
public:
  static int generateRandomNumber(int min, int max) {
    static bool initialized = false;
    if (!initialized) {
      std::srand(static_cast<unsigned int>(std::time(nullptr)));
      initialized = true;
    }

    return min + std::rand() % (max - min + 1);
  }
};

// Класс Игрок
class Player {
private:
  std::string name;
  int speed;
  int shootingAccuracy;
  int stamina;
  int price;

public:
  Player(std::string playerName, int playerSpeed,
         int playerShootingAccuracy, int playerStamina)
         : name(std::move(playerName)), speed(playerSpeed),
           shootingAccuracy(playerShootingAccuracy), stamina(playerStamina) {
    price = (speed * 10 + shootingAccuracy * 10 + stamina * 10) *
            RandomNumberGenerator::generateRandomNumber(50, 150) / 100;
  }

  // Геттеры и сеттеры для атрибутов игрока

  std::string getName() const {
    return name;
  }

  int getPrice() const {
    return price;
  }

  int getSpeed() const {
    return speed;
  }

  int getShootingAccuracy() const {
    return shootingAccuracy;
  }
```

```
 int getStamina() const {
  return stamina;
 }

 static Player generateRandomPlayer() {
  std::string playerName;
  int speed = RandomNumberGenerator::generateRandomNumber(0, 100);
  int shootingAccuracy = RandomNumberGenerator::generateRandomNumber(0, 100);
  int stamina = RandomNumberGenerator::generateRandomNumber(0, 100);

  return {std::move(playerName), speed, shootingAccuracy, stamina};
 }

 static Player generateRandomPlayer(std::string playerName) {
  int speed = RandomNumberGenerator::generateRandomNumber(50, 100);
  int shootingAccuracy = RandomNumberGenerator::generateRandomNumber(60, 100);
  int stamina = RandomNumberGenerator::generateRandomNumber(70, 100);

  return {std::move(playerName), speed, shootingAccuracy, stamina};
 }

 static Player generateRandomPlayer(std::pair<int, int> Stats) {
  std::string playerName = "Player";
  int speed = RandomNumberGenerator::generateRandomNumber(Stats.first, Stats.second);
  int shootingAccuracy = RandomNumberGenerator::generateRandomNumber(Stats.first, Stats.second);
  int stamina = RandomNumberGenerator::generateRandomNumber(Stats.first, Stats.second);

  return {playerName, speed, shootingAccuracy, stamina};
 }

 static Player generateRandomPlayer(std::string name, std::pair<int, int> Stats) {
  std::string playerName = std::move(name);
  int speed = RandomNumberGenerator::generateRandomNumber(Stats.first, Stats.second);
  int shootingAccuracy = RandomNumberGenerator::generateRandomNumber(Stats.first, Stats.second);
  int stamina = RandomNumberGenerator::generateRandomNumber(Stats.first, Stats.second);

  return {playerName, speed, shootingAccuracy, stamina};
 }

 void printPlayerInfo() {
  std::cout << "Name: " << name
            << ", Speed: " << speed
            << ", Accuracy: " << shootingAccuracy
            << ", Stamina: " << stamina
            << ", Price: " << price << std::endl;
 }
};

// Класс Команда
class Team {
protected:
 std::string name;
 std::vector<Player> players;
 std::vector<std::string> matchResults;

public:
 Team() {}

 Team(std::string teamName)
       : name(std::move(teamName)) {}

 // Метод для добавления игрока в команду
 void addPlayer(const Player &player) {
```

```
    players.push_back(player);
  }

  // Методы для получения информации о команде и игроках

  std::string getName() const {
    return name;
  }

  std::vector<Player> getPlayers() const {
    return players;
  }

  void generateRandomTeam(int numPlayers) {
    players.clear();
    // Генерация случайных характеристик для каждого игрока
    for (int i = 0; i < numPlayers; i++)
      players.push_back(Player::generateRandomPlayer());
  }

  void generateRandomTeam(int numPlayers,
                          std::pair<int, int> Stats) {
    players.clear();
    // Генерация случайных характеристик для каждого игрока
    for (int i = 0; i < numPlayers; i++)
      players.push_back(Player::generateRandomPlayer(Stats));
  }

  // Метод для вывода средних показателей игроков
  void printAveragePlayerStats() const {
    std::cout << "Average Player Stats for Team " << name << ":" << std::endl;
    int totalSpeed = 0;
    int totalShootingAccuracy = 0;
    int totalStamina = 0;

    for (const Player &player: players) {
      totalSpeed += player.getSpeed();
      totalShootingAccuracy += player.getShootingAccuracy();
      totalStamina += player.getStamina();
    }

    int numPlayers = players.size();
    double averageSpeed = static_cast<double>(totalSpeed) / numPlayers;
    double averageShootingAccuracy = static_cast<double>(totalShootingAccuracy) / numPlayers;
    double averageStamina = static_cast<double>(totalStamina) / numPlayers;

    std::cout << "Average Speed: " << averageSpeed << std::endl;
    std::cout << "Average Shooting Accuracy: " << averageShootingAccuracy << std::endl;
    std::cout << "Average Stamina: " << averageStamina << std::endl;
  }

  bool operator==(Team &rhs) {
    if (name != rhs.getName())
      return false;
    return true;
  }
};

// Класс Матч
class Match {
private:
  Team homeTeam;
  Team awayTeam;
```

```
 int homeScore;
 int awayScore;
 std::string result;

public:
 Match(Team team1, Team team2)
       : homeTeam(std::move(team1)), awayTeam(std::move(team2)), homeScore(0),
         awayScore(0) {}

 // Геттеры и сеттеры для атрибутов матча
 [[nodiscard]] Team getHomeTeam() const {
  return homeTeam;
 }

 Team getAwayTeam() const {
  return awayTeam;
 }

 int getHomeScore() const {
  return homeScore;
 }

 int getAwayScore() const {
  return awayScore;
 }

 std::string getResult() const {
  return result;
 }

 void setHomeScore(int score) {
  homeScore = score;
 }

 void setAwayScore(int score) {
  awayScore = score;
 }

 void setResult(const std::string &matchResult) {
  result = matchResult;
 }

 void calculateMatchResult() {
  // Получение игроков из команд
  std::vector<Player> homePlayers = homeTeam.getPlayers();
  std::vector<Player> awayPlayers = awayTeam.getPlayers();

  homeScore = 0;
  for (const Player &homePlayer: homePlayers) {
   // Расчет способности игрока влиять на результаты матча
   int playerAbility =
         (homePlayer.getSpeed() + homePlayer.getShootingAccuracy() +
          homePlayer.getStamina()) / 3;

   // Генерация случайного числа для определения результата действия игрока
   int randomNumber = RandomNumberGenerator::generateRandomNumber(0, 100);

   if (randomNumber <= playerAbility) {
    // Действие игрока успешно - гол для домашней команды
    homeScore++;
   }
  }
```

```
    awayScore = 0;
    for (const Player &awayPlayer: awayPlayers) {
      // Расчет способности игрока влиять на результаты матча
      int playerAbility =
              (awayPlayer.getSpeed() + awayPlayer.getShootingAccuracy() +
               awayPlayer.getStamina()) / 3;

      // Генерация случайного числа для определения результата действия игрока
      int randomNumber = RandomNumberGenerator::generateRandomNumber(0, 110);

      if (randomNumber <= playerAbility) {
        // Действие игрока успешно - гол для домашней команды
        awayScore++;
      }
    }

    // Обновление счета и результата матча
    if (homeScore > awayScore) {
      result = "Home";
    } else if (homeScore < awayScore) {
      result = "Away";
    } else {
      result = "Draw";
    }
  }
};

// Класс Турнир
class Tournament {
private:
  std::string name;
  std::vector<std::pair<Team, int>> teams;
  std::vector<Match> matches;
  std::vector<std::string> results;
  std::pair<int, int> stats;
  int reward;

  void increasePoint(Team team, int score) {
    for (auto &i: teams) {
      if (i.first == team) {
        i.second += score;
        return;
      }
    }
  }

public:
  Tournament() {}

  Tournament(std::string difficult, std::pair<int, int> stats, int reward)
          : name(std::move(difficult)), stats(std::move(stats)), reward(reward) {}

  // Метод для добавления команды в турнир
  void addTeam(const Team &team) {
    teams.push_back({team, 0});
  }

  std::string getName() {
    return name;
  }

  int getReward() {
    return reward;
```

```
}

std::pair<int, int> getAverageStats() {
  return stats;
}

std::vector<std::string> getResults() const {
  return results;
}

std::vector<std::pair<Team, int>> getTeams() {
  return teams;
}

// Метод для создания расписания матчей
void createSchedule() {
  matches.clear();
  for (int i = 0; i < teams.size() - 1; i++) {
    for (int j = i + 1; j < teams.size(); j++) {
      // Создание матча между командами i и j
      Match match(teams[i].first, teams[j].first);
      matches.push_back(match);
    }
  }
}

// Метод для проведения матчей в турнире
void playMatches() {
  results.clear();
  for (Match &match: matches) {
    match.calculateMatchResult();
    if (match.getResult() == "Home")
      increasePoint(match.getHomeTeam(), 3);
    if (match.getResult() == "Away")
      increasePoint(match.getAwayTeam(), 3);
    else {
      increasePoint(match.getHomeTeam(), 1);
      increasePoint(match.getAwayTeam(), 1);
    }
    results.push_back(match.getResult());
  }
}

// Метод для генерации заданного количества команд со случайными характеристиками
void generateRandomTeams(int numTeams) {
  teams.clear();
  for (int i = 0; i < numTeams; i++) {
    Team team("Team " + std::to_string(i + 1));
    team.generateRandomTeam(10, stats);
    teams.push_back({team, 0});
  }
}

void sortTeams() {
  std::sort(teams.begin(), teams.end(), [](auto &left, auto &right) {
    return left.second > right.second;
  });
}

int findTeam(Team team) {
  for (int i = 0; i < teams.size(); i++)
    if (teams[i].first == team)
      return i;
```

```
    return -1;
  }
};

// Класс Стадион
class Stadium {
private:
  int capacity;
  std::vector<Tournament> tournaments;

public:
  Stadium() {
    Tournament noobs("Низшая лига", (std::pair<int, int>) {0, 15}, 1000);
    Tournament medium("Средняя лига", (std::pair<int, int>) {15, 30}, 5000);
    Tournament pro("Высшая лига", (std::pair<int, int>) {30, 60}, 10000);
    Tournament secret("Секретная лига", (std::pair<int, int>) {60, 90}, 50000);
    tournaments = {noobs, medium, pro, secret};
  }

  // Геттеры и сеттеры для атрибутов стадиона
  int getCapacity() const {
    return capacity;
  }

  std::vector<Tournament> getTournaments() const {
    return tournaments;
  }

  void printTournaments() {
    std::cout << "Available Tournaments:\n" << std::endl;
    for (int i = 0; i < tournaments.size(); i++) {
      Tournament tournament = tournaments[i];
      std::cout << "id: " << i + 1 << ", " << tournament.getName() << ": " <<
                tournament.getAverageStats().first << " - " <<
                tournament.getAverageStats().second <<
                ", Призовые: " << tournament.getReward() << std::endl;
    }
    for (Tournament &tournament: tournaments) {
    }
    std::cout << std::endl;
  }
};

class PlayerTeam : public Team {
private:
  int money;
  int gamesWon;
  int gamesLost;

public:
  PlayerTeam() : Team(""), money(1000), gamesWon(0), gamesLost(0) {
    for (auto i = 0; i < 10; i++) {
      Player player = Player::generateRandomPlayer("Игрок " + std::to_string(i + 1), (std::pair<int, int>) {5, 15});
      players.push_back(player);
    }
  }

  void setTeamName(std::string teamName) {
    name = std::move(teamName);
  }

  [[nodiscard]] int getMoney() {
```

```
    return money;
  }

  [[nodiscard]] int getGamesWon() {
    return gamesWon;
  }

  [[nodiscard]] int getGamesLost() {
    return gamesLost;
  }

  void increaseMoney(int amount) {
    money += amount;
  }

  void decreaseMoney(int amount) {
    money -= amount;
  }

  void increaseGamesWon() {
    gamesWon++;
  }

  void increaseGamesLost() {
    gamesLost++;
  }

  void removePlayer(int playerIndex) {
    if (playerIndex >= 0 && playerIndex < players.size()) {
      Player removedPlayer = players[playerIndex];
      increaseMoney(removedPlayer.getPrice());
      players.erase(players.begin() + playerIndex);
    }
  }

  void printTeamInfo() {
    std::cout << "Team Name: " << getName() << std::endl;
    std::cout << "Money: " << getMoney() << std::endl;
    std::cout << "Games Won: " << getGamesWon() << std::endl;
    std::cout << "Games Lost: " << getGamesLost() << std::endl;

    std::cout << "Players:" << std::endl;
    for (int i = 0; i < players.size(); i++) {
      std::cout << "id: " << i + 1 << "; ";
      players[i].printPlayerInfo();
    }
  }
};

class PlayerMarket {
private:
  std::vector<Player> availablePlayers;

public:
  PlayerMarket() {
    generateAvailablePlayers();
  }

  std::vector<Player> getAvailablePlayers() {
    return availablePlayers;
  }

  void generateAvailablePlayers() {
```

```
    availablePlayers.clear();
    for (int i = 0; i < 10; i++) {
      availablePlayers.push_back(Player::generateRandomPlayer());
    }
  }

  void displayAvailablePlayers() {
    std::cout << "Players:" << std::endl;
    for (int i = 0; i < availablePlayers.size(); i++) {
      std::cout << "id: " << i + 1 << "; ";
      availablePlayers[i].printPlayerInfo();
    }
  }

  void purchasePlayer(int playerIndex, Team &team, PlayerTeam &player) {
    if (playerIndex >= 0 && playerIndex < availablePlayers.size()) {
      Player purchasedPlayer = availablePlayers[playerIndex];
      if (purchasedPlayer.getPrice() <= player.getMoney()) {
        team.addPlayer(purchasedPlayer);

        // Удаление приобретенного игрока из доступных игроков на рынке
        availablePlayers.erase(availablePlayers.begin() + playerIndex);
        std::cout << "Player " << purchasedPlayer.getName()
                  << " purchased successfully!" << std::endl;
      } else {
        std::cout << "Недостаточно денег.\n" << std::endl;
      }
    } else {
      std::cout << "Invalid player index.\n" << std::endl;
    }
  }
};

class UserInteraction {
private:
  static UserInteraction *instance;  // Статический указатель на единственный экземпляр класса

  UserInteraction() {}  // Приватный конструктор для предотвращения прямого создания объектов

public:
  std::string menuName;
  PlayerTeam player;
  Stadium stadium;
  PlayerMarket market;
  Tournament tournament;

  static UserInteraction *getInstance() {
    if (!instance) {
      instance = new UserInteraction();
    }
    return instance;
  }

  void menuMain() {
    std::cout << "Доступные команды:\n";
    std::cout << "1. Рынок игроков\n";
    std::cout << "2. Записаться на турнир\n";
    std::cout << "3. Моя команда\n";
    std::cout << "0. Выйти из программы\n";
    std::cout << "Введите команду: ";

    int choice;
    std::cin >> choice;
```

```
  std::cout << "\n";
  if (choice == 1) {
    menuName = "market";
  } else if (choice == 2) {
    menuName = "tournaments";
  } else if (choice == 3) {
    player.printTeamInfo();
  } else if (choice == 0) {
    // Выход из программы
    std::cout << "Программа завершена.\n";
    exit(0);
  } else {
    std::cout << "Неверный выбор команды. Попробуйте снова.\n";
  }

  std::cout << "\n";
}

void menuMarket() {
  std::cout << "Доступные команды:\n";
  std::cout << "1. Продать игрока\n";
  std::cout << "2. Купить игрока\n";
  std::cout << "3. Назад\n";
  std::cout << "0. Выйти из программы\n";
  std::cout << "Введите команду: ";

  int choice;
  std::cin >> choice;
  std::cout << "\n";
  if (choice == 1) {
    std::cout << "\nДоступные для продажи игроки:" << std::endl;
    player.printTeamInfo();

    int playerIndex;
    std::cout << "Введите индекс игрока, которого хотите продать (0 - если передумали): ";
    std::cin >> playerIndex;
    playerIndex--;
    if (playerIndex >= 0 && playerIndex < player.getPlayers().size()) {
      player.removePlayer(playerIndex);
      std::cout << "Вы успешно продали игрока!\n" << std::endl;
    } else if (playerIndex == -1) {
      std::cout << "Возвращаю в главное меню...\n" << std::endl;
    } else {
      std::cout << "Неверный индекс.\n" << std::endl;
    }
    menuName = "main";


  } else if (choice == 2) {
    std::cout << "\nДоступные для покупки игроки:" << std::endl;
    market.generateAvailablePlayers();
    market.displayAvailablePlayers();

    int playerIndex;
    std::cout << "Введите индекс игрока, которого хотите купить (0 - если передумали): ";
    std::cin >> playerIndex;

    if (playerIndex > 0 && playerIndex <= market.getAvailablePlayers().size() && player.getPlayers().size() < 10) {
      Player purchasedPlayer = market.getAvailablePlayers()[playerIndex - 1];
      if (player.getMoney() >= purchasedPlayer.getPrice()) {
        player.addPlayer(purchasedPlayer);
        player.decreaseMoney(purchasedPlayer.getPrice());
        market.generateAvailablePlayers();
```

```
        std::cout << "Вы успешно приобрели нового игрока вам в команду!\n" << std::endl;
      } else {
        std::cout << "У вас недостаточно денег для покупки этого игрока\n" << std::endl;
      }
    } else if (playerIndex == 0) {
      std::cout << "Возвращаю в главное меню...\n" << std::endl;
    } else if (player.getPlayers().size() >= 10) {
      std::cout << "Больше 10 игроков нельзя.\n" << std::endl;
    } else {
      std::cout << "Неверный индекс.\n" << std::endl;
    }
    menuName = "main";
  } else if (choice == 3) {
    menuName = "main";
  } else if (choice == 0) {
    std::cout << "Программа завершена.\n";
    exit(0);
  } else {
    std::cout << "Неверный выбор команды. Попробуйте снова.\n";
  }
}

void menuTournaments() {
  stadium.printTournaments();
  std::cout << "Выберите интересующую вас лигу: ";

  int choice;
  std::cin >> choice;
  if (choice > 0 && choice <= stadium.getTournaments().size()) {
    tournament = stadium.getTournaments()[choice];
    std::cout << "Вы записаны на турнир в Лигу: " << tournament.getName() << std::endl;
    menuName = "match";
  } else {
    std::cout << "Не удалось распознать командую" << std::endl;
    menuName = "main";
  }
}

void menuMatch() {
  std::cout << "Доступные команды:\n";
  std::cout << "1. Начать турнир\n";
  std::cout << "2. Уйти\n";

  int choice;
  std::cin >> choice;
  if (choice == 1) {
    tournament.generateRandomTeams(10);
    tournament.addTeam(player);
    tournament.createSchedule();
    tournament.playMatches();
    tournament.sortTeams();
    int current = tournament.getReward();
    int place = tournament.findTeam(player);
    for (int i = 0; i < tournament.getTeams().size(); i++) {
      auto team = tournament.getTeams()[i];
      current /= 2;
      std::cout << "Место: " << i + 1 << ", Команда: " << team.first.getName()
                << ", Очки: " << team.second << ", Призовые: " << current << std::endl;
    }
    int playerReward = tournament.getReward() / std::pow(2, place + 1);
    player.increaseMoney(playerReward);
    std::cout << "\n\n";
    menuName = "main";
```

```
    } else if (choice == 2) {
      menuName = "main";
    } else {
      std::cout << "Не удалось распознать команду." << std::endl;
    }
  }
};

// Инициализация статического указателя на ноль
UserInteraction* UserInteraction::instance = nullptr;

int main() {
  SetConsoleOutputCP(CP_UTF8);
  std::cout << "Введите название вашей команды: ";
  std::string teamName;
  std::cin >> teamName;
  UserInteraction::getInstance()->menuName = "main";
  UserInteraction::getInstance()->player.setTeamName(teamName);

  while (true) {
    if (UserInteraction::getInstance()->menuName == "main")
      UserInteraction::getInstance()->menuMain();
    else if (UserInteraction::getInstance()->menuName == "market")
      UserInteraction::getInstance()->menuMarket();
    else if (UserInteraction::getInstance()->menuName == "tournaments")
      UserInteraction::getInstance()->menuTournaments();
    else if (UserInteraction::getInstance()->menuName == "match")
      UserInteraction::getInstance()->menuMatch();
  }
}
```

Примеры работы программы:

Раздел моя команда:

```
Введите название вашей команды:dfdf
 Доступные команды:
1. Рынок игроков
2. Записаться на турнир
3. Моя команда
0. Выйти из программы
Введите команду:3

Team Name: dfdf
Money: 1000
Games Won: 0
Games Lost: 0
Players:
id: 1; Name: Игрок 1, Speed: 10, Accuracy: 11, Stamina: 9, Price: 420
id: 2; Name: Игрок 2, Speed: 8, Accuracy: 13, Stamina: 14, Price: 325
id: 3; Name: Игрок 3, Speed: 9, Accuracy: 5, Stamina: 13, Price: 324
id: 4; Name: Игрок 4, Speed: 6, Accuracy: 5, Stamina: 9, Price: 106
id: 5; Name: Игрок 5, Speed: 9, Accuracy: 14, Stamina: 15, Price: 570
id: 6; Name: Игрок 6, Speed: 15, Accuracy: 9, Stamina: 6, Price: 336
id: 7; Name: Игрок 7, Speed: 5, Accuracy: 15, Stamina: 11, Price: 353
id: 8; Name: Игрок 8, Speed: 6, Accuracy: 11, Stamina: 14, Price: 198
id: 9; Name: Игрок 9, Speed: 9, Accuracy: 7, Stamina: 11, Price: 324
id: 10; Name: Игрок 10, Speed: 11, Accuracy: 15, Stamina: 14, Price: 576
```

Пример продажи игрока:

```
Доступные команды:
1. Продать игрока
2. Купить игрока
3. Назад
0. Выйти из программы
Введите команду:1


Доступные для продажи игроки:
Team Name: adsad
Money: 1000
Games Won: 0
Games Lost: 0
Players:
id: 1; Name: Игрок 1, Speed: 5, Accuracy: 13, Stamina: 7, Price: 265
id: 2; Name: Игрок 2, Speed: 13, Accuracy: 13, Stamina: 7, Price: 240
id: 3; Name: Игрок 3, Speed: 13, Accuracy: 6, Stamina: 9, Price: 179
id: 4; Name: Игрок 4, Speed: 9, Accuracy: 11, Stamina: 7, Price: 359
id: 5; Name: Игрок 5, Speed: 8, Accuracy: 12, Stamina: 15, Price: 490
id: 6; Name: Игрок 6, Speed: 9, Accuracy: 15, Stamina: 12, Price: 475
id: 7; Name: Игрок 7, Speed: 11, Accuracy: 15, Stamina: 14, Price: 344
id: 8; Name: Игрок 8, Speed: 6, Accuracy: 15, Stamina: 9, Price: 162
id: 9; Name: Игрок 9, Speed: 8, Accuracy: 12, Stamina: 12, Price: 192
id: 10; Name: Игрок 10, Speed: 6, Accuracy: 11, Stamina: 14, Price: 350
Введите индекс игрока, которого хотите продать (0 - если передумали):1
 Вы успешно продали игрока!
```

Пример покупки игрока:

```
Доступные команды:
1. Продать игрока
2. Купить игрока
3. Назад
0. Выйти из программы
Введите команду:2


Доступные для покупки игроки:
Players:
id: 1; Name: , Speed: 45, Accuracy: 79, Stamina: 12, Price: 1944
id: 2; Name: , Speed: 16, Accuracy: 26, Stamina: 89, Price: 681
id: 3; Name: , Speed: 34, Accuracy: 4, Stamina: 53, Price: 1319
id: 4; Name: , Speed: 58, Accuracy: 80, Stamina: 11, Price: 834
id: 5; Name: , Speed: 18, Accuracy: 9, Stamina: 84, Price: 1209
id: 6; Name: , Speed: 90, Accuracy: 50, Stamina: 49, Price: 2249
id: 7; Name: , Speed: 48, Accuracy: 73, Stamina: 32, Price: 1728
id: 8; Name: , Speed: 68, Accuracy: 6, Stamina: 78, Price: 2036
id: 9; Name: , Speed: 52, Accuracy: 46, Stamina: 53, Price: 1419
id: 10; Name: , Speed: 84, Accuracy: 0, Stamina: 77, Price: 1127
Введите индекс игрока, которого хотите купить (0 - если передумали):2
 Вы успешно приобрели нового игрока вам в команду!
```

Пример турнира:

```
Доступные команды:
1. Рынок игроков
2. Записаться на турнир
3. Моя команда
0. Выйти из программы
Введите команду:2


Available Tournaments:

id: 1, Низшая лига: 0 - 15, Призовые: 1000
id: 2, Средняя лига: 15 - 30, Призовые: 5000
id: 3, Высшая лига: 30 - 60, Призовые: 10000
id: 4, Секретная лига: 60 - 90, Призовые: 50000

Выберите интересующую вас лигу:1
 Вы записаны на турнир в Лигу: Средняя лига
Доступные команды:
1. Начать турнир
2. Уйти
1
Место: 1, Команда: Team 2, Очки: 24, Призовые: 2500
Место: 2, Команда: Team 4, Очки: 22, Призовые: 1250
Место: 3, Команда: Team 9, Очки: 22, Призовые: 625
Место: 4, Команда: Team 10, Очки: 21, Призовые: 312
Место: 5, Команда: Team 1, Очки: 19, Призовые: 156
Место: 6, Команда: Team 5, Очки: 18, Призовые: 78
Место: 7, Команда: Team 6, Очки: 18, Призовые: 39
Место: 8, Команда: Team 3, Очки: 17, Призовые: 19
Место: 9, Команда: Team 8, Очки: 17, Призовые: 9
Место: 10, Команда: Team 7, Очки: 16, Призовые: 4
Место: 11, Команда: adsad, Очки: 14, Призовые: 2
```