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: Объектно-ориентированное программирование
тема: «Программа моделирования спортивной игры (футбол)»

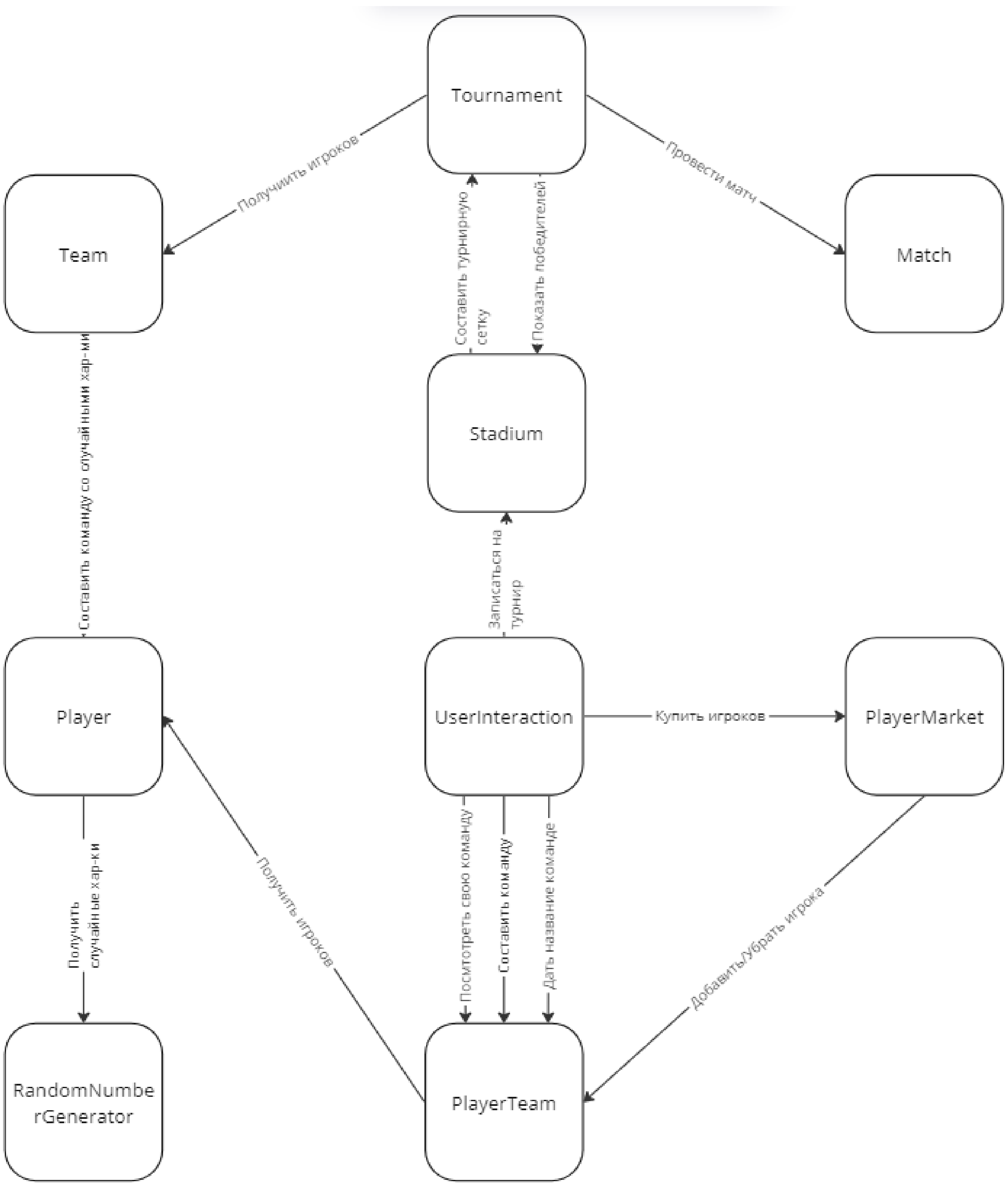
Выполнил: ст. группы ПВ-202
Аладиб Язан
Проверил:
Буханов Дмитрий Геннадьевич

Белгород 2022 г.

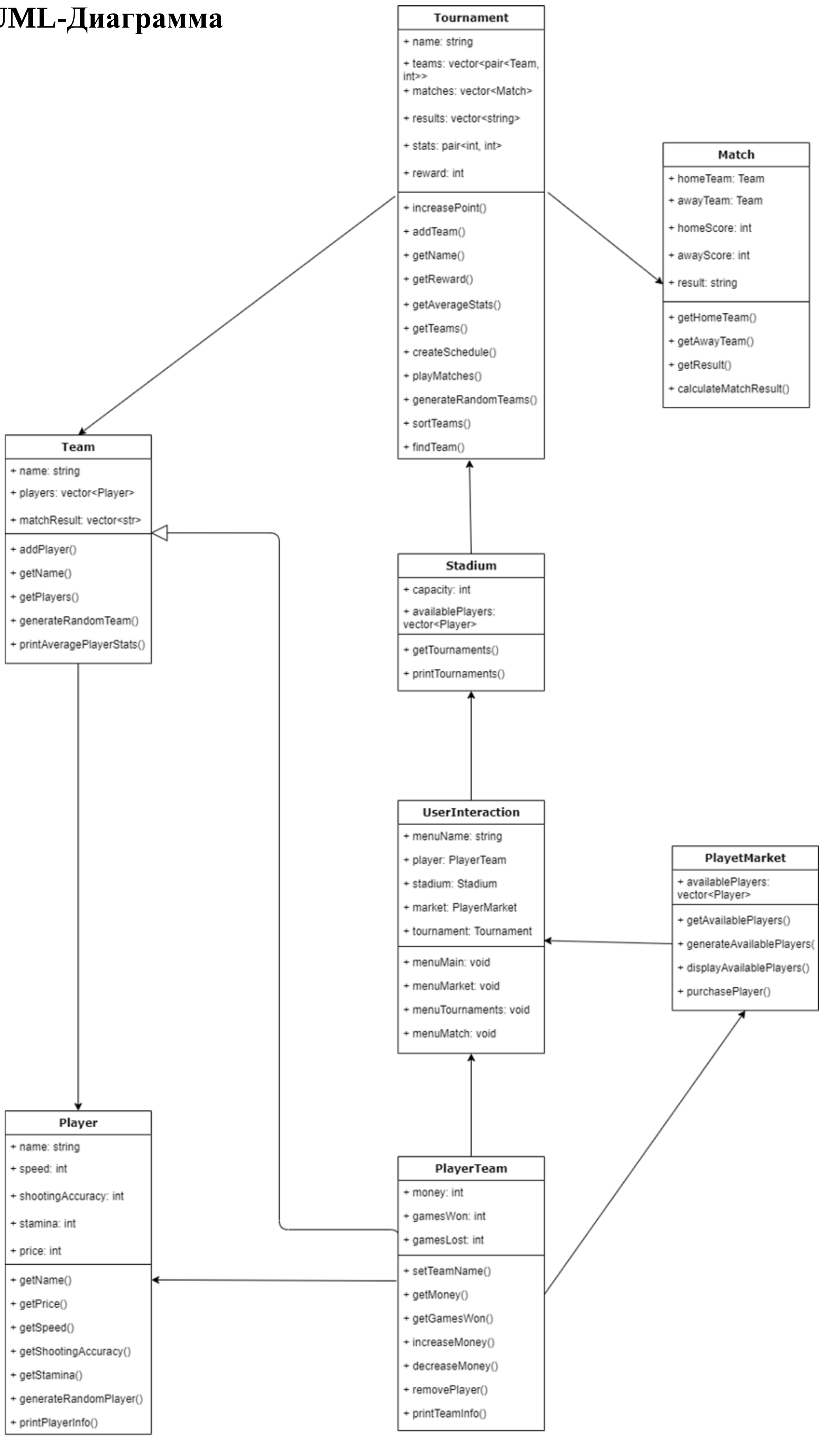
Постановка задачи:

Программа моделирования спортивной игры (футбол). Программа должна обеспечить возможность учета игроков различных команд, моделирования матчей команд, учета результатов матчей в различных турнирах. Каждая команда имеет отличительные свойства (например, настрой, умение нападать, умение защищаться), которые влияют на результаты матчей с участием этой команды. Каждый игрок характеризуются некоторыми атрибутами, которые влияют на его поведение во время матча (например, скорость, точность удара, выносливость и т.п.).

Объектная декомпозиция задачи



UML-Диаграмма



Код программы:

```
#include <iostream>
#include <string>
#include <utility>
#include <vector>
#include <cstdlib>
#include <ctime>
#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

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

  [[nodiscard]] int getPrice() const {
    return price;
  }

  [[nodiscard]] int getSpeed() const {
    return speed;
  }

  [[nodiscard]] int getShootingAccuracy() const {
    return shootingAccuracy;
  }
  [[nodiscard]] 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() = default;
  explicit 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

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

  [[nodiscard]] 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("noobs league", (std::pair<int, int>) {0, 15}, 1000);
    Tournament medium("medium league", (std::pair<int, int>) {15, 30}, 5000);
    Tournament pro("pro league", (std::pair<int, int>) {30, 60}, 10000);
    Tournament secret("secret league", (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 <<
                ", Prize money: " << 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("Player " + 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 << "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 << "Available commands:\n";
    std::cout << "1. Players market\n";
    std::cout << "2. Sign up for tournament\n";
    std::cout << "3. My team\n";
    std::cout << "0. Exit program\n";
    std::cout << "Enter command: ";

    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 << "The program is complete.\n";
      exit(0);
    } else {
      std::cout << "Invalid command selection. Try again.\n";
    }

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

  void menuMarket() {
    std::cout << "Available commands:\n";
    std::cout << "1. Sell player\n";
    std::cout << "2. Buy player\n";
    std::cout << "3. Back\n";
    std::cout << "0. Exit program\n";
    std::cout << "Enter command: ";

    int choice;
    std::cin >> choice;
    std::cout << "\n";
    if (choice == 1) {
      std::cout << "\nAvailable players for sale:" << std::endl;
      player.printTeamInfo();

      int playerIndex;
      std::cout << "Enter the index of the player you want to sell (0 - if you changed
your mind): ";
      std::cin >> playerIndex;
      playerIndex--;
      if (playerIndex >= 0 && playerIndex < player.getPlayers().size()) {
        player.removePlayer(playerIndex);
        std::cout << "You have successfully sold the player!\n" << std::endl;
      } else if (playerIndex == -1) {
        std::cout << "Returning to the main menu...\n" << std::endl;
      } else {
        std::cout << "Invalid index.\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 << "Enter the index of the player you want to buy (0 - if you changed
your mind): ";
      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 << "You have successfully acquired a new player for your team!\n"
<< std::endl;
        } else {
          std::cout << "You do not have enough money to buy this player\n" <<
std::endl;
        }
      } else if (playerIndex == 0) {
        std::cout << "Returning to the main menu...\n" << std::endl;
      } else if (player.getPlayers().size() >= 10) {
        std::cout << "More than 10 players are not allowed.\n" << std::endl;
      } else {
        std::cout << "Invalid index.\n" << std::endl;
      }
      menuName = "main";
    } else if (choice == 3) {
      menuName = "main";
    } else if (choice == 0) {
      std::cout << "Program terminated.\n";
      exit(0);
    } else {
      std::cout << "Invalid team selection. Please try again.\n";
    }
  }

  void menuTournaments() {
    stadium.printTournaments();
    std::cout << "Select the league you are interested in: ";

    int choice;
    std::cin >> choice;
    if (choice > 0 && choice <= stadium.getTournaments().size()) {
      tournament = stadium.getTournaments()[choice - 1];
      std::cout << "You have signed up for the tournament in the League: " <<
tournament.getName() << std::endl;
      menuName = "match";
    } else {
      std::cout << "Failed to recognize the command" << std::endl;
      menuName = "main";
    }
  }

  void menuMatch() {
    std::cout << "Available options:\n";
    std::cout << "1. Start the tournament\n";
    std::cout << "2. Leave\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 << "Place: " << i + 1 << ", Team: " << team.first.getName()
                  << ", Points: " << team.second << ", Prize: " << 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 << "Failed to recognize the command." << std::endl;
    }
  }
};

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

int main() {
  SetConsoleOutputCP(CP_UTF8);
  std::cout << "Enter the name of your team: ";
  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();
  }
}
```