**NAME: Shamveel Khan**

**CLASS: BsCs-2J**

**Assignment#01**

**Q1:**

#include <iostream>

#include <string>

using namespace std;

class Mentor;

class Skill {

private:

    int skillID;

    string skillName;

    string description;

public:

    Skill(int id, string name, string desc) : skillID(id), skillName(name), description(desc) {}

    void showSkillDetails() {

        cout << "Skill ID: " << skillID << ", Name: " << skillName << ", Desc: " << description << endl;

    }

    void updateSkillDescription(string newDescription) {

        description = newDescription;

    }

    int getSkillID() { return skillID; }

    string getSkillName() { return skillName; }

};

class Sport {

private:

    int sportID;

    string name;

    string description;

    Skill\* requiredSkills[10];

    int skillCount;

public:

    Sport(int id, string n, string desc) : sportID(id), name(n), description(desc), skillCount(0) {}

    void addSkill(Skill\* s) {

        if (skillCount < 10) {

            requiredSkills[skillCount++] = s;

            cout << "Skill " << s->getSkillName() << " added to " << name << endl;

        }

    }

    void removeSkill(Skill\* s) {

        for (int i = 0; i < skillCount; i++) {

            if (requiredSkills[i]->getSkillID() == s->getSkillID()) {

                for (int j = i; j < skillCount - 1; j++)

                    requiredSkills[j] = requiredSkills[j + 1];

                skillCount--;

                break;

            }

        }

    }

    int getSportID() { return sportID; }

    string getName() { return name; }

};

class Student {

private:

    int studentID;

    string name;

    int age;

    Sport\* sportsInterests[5];

    int sportsCount;

    Mentor\* mentorAssigned;

public:

    Student(int id, string n, int a) : studentID(id), name(n), age(a), sportsCount(0), mentorAssigned(nullptr) {}

    void registerForMentorship(Mentor\* m);

    void viewMentorDetails();

    void updateSportsInterest(Sport\* s) {

        if (sportsCount < 5) {

            sportsInterests[sportsCount++] = s;

            cout << s->getName() << " added to " << name << "'s interests" << endl;

        }

    }

    int getStudentID() { return studentID; }

    string getName() { return name; }

    void setMentorAssigned(Mentor\* m) { mentorAssigned = m; }

    Mentor\* getMentorAssigned() { return mentorAssigned; }

};

class Mentor {

private:

    int mentorID;

    string name;

    Sport\* sportsExpertise[5];

    int expertiseCount;

    int maxLearners;

    Student\* assignedLearners[10];

    int learnerCount;

public:

    Mentor(int id, string n, int max) : mentorID(id), name(n), expertiseCount(0), maxLearners(max), learnerCount(0) {}

    void addSportExpertise(Sport\* s) {

        if (expertiseCount < 5) {

            sportsExpertise[expertiseCount++] = s;

            cout << s->getName() << " added to " << name << "'s expertise" << endl;

        }

    }

    void assignLearner(Student\* s) {

        if (learnerCount >= maxLearners) {

            cout << "Cannot assign more learners. Maximum capacity reached." << endl;

            return;

        }

        for (int i = 0; i < learnerCount; i++) {

            if (assignedLearners[i]->getStudentID() == s->getStudentID()) {

                cout << s->getName() << " is already assigned to this mentor." << endl;

                return;

            }

        }

        assignedLearners[learnerCount++] = s;

        s->setMentorAssigned(this);

        cout << s->getName() << " has been assigned to mentor " << name << endl;

    }

    void removeLearner(Student\* s) {

        for (int i = 0; i < learnerCount; i++) {

            if (assignedLearners[i]->getStudentID() == s->getStudentID()) {

                assignedLearners[i]->setMentorAssigned(nullptr);

                for (int j = i; j < learnerCount - 1; j++)

                    assignedLearners[j] = assignedLearners[j + 1];

                learnerCount--;

                cout << s->getName() << " has been removed from mentor " << name << endl;

                return;

            }

        }

        cout << "Student not found in assigned learners." << endl;

    }

    void viewLearners() {

        if (learnerCount == 0) {

            cout << name << " has no assigned learners." << endl;

            return;

        }

        cout << name << "'s assigned learners:" << endl;

        for (int i = 0; i < learnerCount; i++) {

            cout << (i+1) << ". " << assignedLearners[i]->getName() << endl;

        }

    }

    void provideGuidance() {

        cout << name << " is providing guidance to " << learnerCount << " learners." << endl;

    }

    int getMentorID() { return mentorID; }

    string getName() { return name; }

    Sport\*\* getSportsExpertise() { return sportsExpertise; }

    int getExpertiseCount() { return expertiseCount; }

};

void Student::registerForMentorship(Mentor\* m) {

    if (mentorAssigned != nullptr) {

        cout << name << " is already assigned to a mentor. Please unregister first." << endl;

        return;

    }

    m->assignLearner(this);

}

void Student::viewMentorDetails() {

    if (mentorAssigned == nullptr) {

        cout << name << " is not assigned to any mentor yet." << endl;

        return;

    }

    cout << "Mentor Details: ID: " << mentorAssigned->getMentorID() << ", Name: " << mentorAssigned->getName() << endl;

    cout << "Expertise in:" << endl;

    Sport\*\* expertise = mentorAssigned->getSportsExpertise();

    int expertiseCount = mentorAssigned->getExpertiseCount();

    for (int i = 0; i < expertiseCount; i++) {

        cout << "- " << expertise[i]->getName() << endl;

    }

}

int main() {

    cout<<"\nSHAMVEEL KHAN\n24k-0962\n";

    Skill s1(1, "Forehand", "Tennis forehand");

    Skill s2(2, "Backhand", "Tennis backhand");

    Sport tennis(1, "Tennis", "Racket sport");

    tennis.addSkill(&s1);

    tennis.addSkill(&s2);

    Sport soccer(2, "Soccer", "Team sport");

    Mentor ali(1, "Ali", 3);

    ali.addSportExpertise(&tennis);

    Student saad(1, "Saad", 20);

    saad.updateSportsInterest(&tennis);

    Student khalid(2, "Khalid", 22);

    khalid.updateSportsInterest(&tennis);

    Student fatima(3, "Fatima", 19);

    fatima.updateSportsInterest(&tennis);

    Student omar(4, "Omar", 21);

    omar.updateSportsInterest(&tennis);

    saad.registerForMentorship(&ali);

    khalid.registerForMentorship(&ali);

    fatima.registerForMentorship(&ali);

    ali.viewLearners();

    omar.registerForMentorship(&ali);

    ali.removeLearner(&khalid);

    omar.registerForMentorship(&ali);

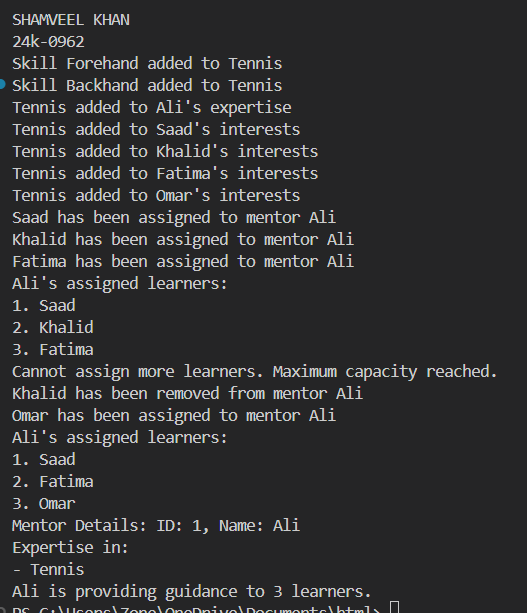
    ali.viewLearners();

    saad.viewMentorDetails();

    ali.provideGuidance();

    return 0;

}



**Q#2:**

#include <iostream>

#include <string>

using namespace std;

class Ball {

private:

    int x, y;

public:

    Ball(int initialX = 0, int initialY = 0) : x(initialX), y(initialY) {}

    int getX() const { return x; }

    int getY() const { return y; }

    void move(int dx, int dy) { x += dx; y += dy; }

    void getPosition(int& outX, int& outY) const { outX = x; outY = y; }

};

class Robot {

private:

    string name;

    int hits;

public:

    Robot(const string& robotName) : name(robotName), hits(0) {}

    void hitBall(int& ballX, int& ballY, const string& direction) {

        if (direction == "up") ballY += 1;

        else if (direction == "down") ballY -= 1;

        else if (direction == "left") ballX -= 1;

        else if (direction == "right") ballX += 1;

        hits++;

    }

    string getName() const { return name; }

    int getHits() const { return hits; }

};

class Goal {

private:

    int x, y;

public:

    Goal(int goalX = 3, int goalY = 3) : x(goalX), y(goalY) {}

    bool isGoalReached(int ballX, int ballY) const { return ballX == x && ballY == y; }

    int getX() const { return x; }

    int getY() const { return y; }

};

class Team {

private:

    string teamName;

    Robot\* robot;

public:

    Team(const string& name, Robot\* r) : teamName(name), robot(r) {}

    string getTeamName() const { return teamName; }

    Robot\* getRobot() const { return robot; }

};

class Game {

private:

    Team\* teamOne;

    Team\* teamTwo;

    Ball ball;

    Goal goal;

public:

    Game(Team\* t1, Team\* t2) : teamOne(t1), teamTwo(t2), ball(0, 0), goal(3, 3) {}

    void play(Team\* team) {

        Robot\* robot = team->getRobot();

        cout << "Team " << team->getTeamName() << " is playing." << endl;

        while (!goal.isGoalReached(ball.getX(), ball.getY())) {

            string direction;

            if (ball.getX() < goal.getX()) direction = "right";

            else if (ball.getX() > goal.getX()) direction = "left";

            else if (ball.getY() < goal.getY()) direction = "up";

            else direction = "down";

            int ballX = ball.getX(), ballY = ball.getY();

            robot->hitBall(ballX, ballY, direction);

            ball.move(ballX - ball.getX(), ballY - ball.getY());

            cout << "Robot " << robot->getName() << " hit the ball " << direction << ". Ball position: (" << ball.getX() << ", " << ball.getY() << ")" << endl;

        }

        cout << "Team " << team->getTeamName() << " reached the goal in " << robot->getHits() << " hits!" << endl;

        ball = Ball(0, 0);

    }

    void startGame() {

        cout << "Game started!" << endl;

        cout << "Goal position: (" << goal.getX() << ", " << goal.getY() << ")" << endl;

        play(teamOne);

        play(teamTwo);

        declareWinner();

    }

    void declareWinner() {

        Robot\* robotOne = teamOne->getRobot();

        Robot\* robotTwo = teamTwo->getRobot();

        cout << "\nGame Results:" << endl;

        cout << "Team " << teamOne->getTeamName() << " took " << robotOne->getHits() << " hits." << endl;

        cout << "Team " << teamTwo->getTeamName() << " took " << robotTwo->getHits() << " hits." << endl;

        if (robotOne->getHits() < robotTwo->getHits()) cout << "Team " << teamOne->getTeamName() << " wins!" << endl;

        else if (robotTwo->getHits() < robotOne->getHits()) cout << "Team " << teamTwo->getTeamName() << " wins!" << endl;

        else cout << "It's a tie!" << endl;

    }

};

int main() {

    cout<<"\nSHAMVEEL KHAN\n24K-0962\n";

    cout << "\n===== Football Game Simulation =====" << endl;

    cout << "\nCreating robots..." << endl;

    Robot\* robotA = new Robot("Striker-1");

    Robot\* robotB = new Robot("Kicker-2");

    cout << "Created robot: " << robotA->getName() << endl;

    cout << "Created robot: " << robotB->getName() << endl;

    cout << "\nForming teams..." << endl;

    Team\* teamA = new Team("Red Dragons", robotA);

    Team\* teamB = new Team("Blue Eagles", robotB);

    cout << "Team " << teamA->getTeamName() << " formed with robot " << teamA->getRobot()->getName() << endl;

    cout << "Team " << teamB->getTeamName() << " formed with robot " << teamB->getRobot()->getName() << endl;

    cout << "\n===== Starting the Actual Game =====" << endl;

    Game game(teamA, teamB);

    game.startGame();

    cout << "\nCleaning up resources..." << endl;

    delete robotA;

    delete robotB;

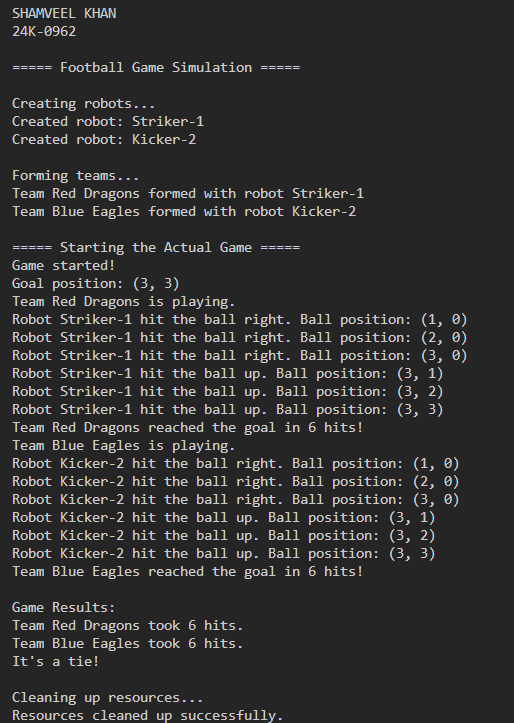
    delete teamA;

    delete teamB;

    cout << "Resources cleaned up successfully." << endl;

    return 0;

}



**Q#3:**

#include <iostream>

#include <string>

using namespace std;

enum LicenseType { LEARNER, INTERMEDIATE, FULL };

class User {

private:

    string name;

    int age;

    string userID;

    LicenseType license;

public:

    User(string n, int a, string id, LicenseType l)

        : name(n), age(a), userID(id), license(l) {}

    string getUserID() { return userID; }

    LicenseType getLicense() { return license; }

    void display() {

        cout << "Name: " << name << ", Age: " << age

             << ", ID: " << userID << ", License: " << license << endl;

    }

};

class Vehicle {

private:

    string model;

    double price;

    LicenseType minLicense;

    bool available;

public:

    Vehicle(string m, double p, LicenseType l)

        : model(m), price(p), minLicense(l), available(true) {}

    bool isAvailable() { return available; }

    LicenseType getMinLicense() { return minLicense; }

    void setAvailable(bool status) { available = status; }

    void display() {

        cout << "Model: " << model << ", Price: $" << price

             << ", Min License: " << minLicense

             << ", Available: " << (available ? "Yes" : "No") << endl;

    }

};

class RentalSystem {

private:

    User\*\* users;

    Vehicle\*\* vehicles;

    int userCount;

    int vehicleCount;

    int maxUsers;

    int maxVehicles;

public:

    RentalSystem(int maxU = 10, int maxV = 10) {

        maxUsers = maxU;

        maxVehicles = maxV;

        users = new User\*[maxUsers];

        vehicles = new Vehicle\*[maxVehicles];

        userCount = vehicleCount = 0;

    }

    ~RentalSystem() {

        for(int i = 0; i < userCount; i++) delete users[i];

        for(int i = 0; i < vehicleCount; i++) delete vehicles[i];

        delete[] users;

        delete[] vehicles;

    }

    void addUser(string name, int age, string id, LicenseType license) {

        if(userCount < maxUsers) {

            users[userCount++] = new User(name, age, id, license);

            cout << "User added successfully\n";

        }

    }

    void addVehicle(string model, double price, LicenseType minLicense) {

        if(vehicleCount < maxVehicles) {

            vehicles[vehicleCount++] = new Vehicle(model, price, minLicense);

            cout << "Vehicle added successfully\n";

        }

    }

    void showAvailableVehicles() {

        for(int i = 0; i < vehicleCount; i++) {

            if(vehicles[i]->isAvailable()) {

                cout << "Vehicle " << i << ": ";

                vehicles[i]->display();

            }

        }

    }

    void rentVehicle(string userID, int vehicleIndex) {

        User\* user = nullptr;

        for(int i = 0; i < userCount; i++) {

            if(users[i]->getUserID() == userID) {

                user = users[i];

                break;

            }

        }

        if(!user || vehicleIndex >= vehicleCount || !vehicles[vehicleIndex]->isAvailable()) {

            cout << "Invalid rental request\n";

            return;

        }

        if(user->getLicense() >= vehicles[vehicleIndex]->getMinLicense()) {

            vehicles[vehicleIndex]->setAvailable(false);

            cout << "Vehicle rented successfully\n";

        } else {

            cout << "License requirements not met\n";

        }

    }

};

int main() {

    RentalSystem system;

    cout<<"\nShamveel Khan\n24k-0962\n";

    cout << "\n=== Vehicle Rental System Demo ===\n";

    cout << "\n1. Adding Vehicles:\n";

    system.addVehicle("Toyota Corolla", 50.0, LEARNER);

    system.addVehicle("Honda Civic", 60.0, INTERMEDIATE);

    system.addVehicle("bugati chiron", 100.0, FULL);

    system.addVehicle("mehran dabba", 55.0, LEARNER);

    cout << "\n2. Initial Vehicle Inventory:\n";

    system.showAvailableVehicles();

    cout << "\n3. Registering Users:\n";

    system.addUser("Shamveel khan", 20, "U001", LEARNER);

    system.addUser("usman hassan", 25, "U002", INTERMEDIATE);

    system.addUser("moiz mughal", 30, "U003", FULL);

    cout << "\n4. Testing Rental Restrictions:\n";

    cout << "\nLearner License Tests (Shamveel - U001):\n";

    system.rentVehicle("U001", 0);

    system.rentVehicle("U001", 1);

    system.rentVehicle("U001", 2);

    cout << "\nIntermediate License Tests (Usman - U002):\n";

    system.rentVehicle("U002", 1);

    system.rentVehicle("U002", 2);

    cout << "\nFull License Tests (Moiz - U003):\n";

    system.rentVehicle("U003", 2);

    cout << "\n5. Updated Vehicle Inventory:\n";

    system.showAvailableVehicles();

    cout << "\n6. Testing Error Cases:\n";

    cout << "Attempting to rent an already rented vehicle:\n";

    system.rentVehicle("U002", 0);

    cout << "Attempting to rent with invalid user ID:\n";

    system.rentVehicle("U999", 3);

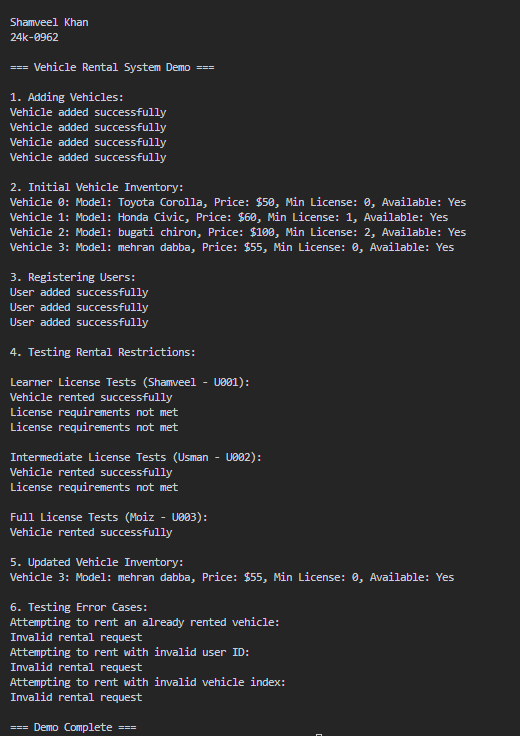
    cout << "Attempting to rent with invalid vehicle index:\n";

    system.rentVehicle("U003", 9);

    cout << "\n=== Demo Complete ===\n";

    return 0;

}



**Q#4:**

#include <iostream>

#include <string>

using namespace std;

class Stop {

private:

    string name;

    string location;

public:

    Stop(string n, string l): name(n), location(l) {}

    string getName() { return name; }

    string getLocation() { return location; }

};

class Route {

private:

    string routeId;

    Stop\*\* stops;

    int stopCount;

    static const int MAX\_STOPS = 10;

public:

    Route(string id): routeId(id), stopCount(0) {

        stops = new Stop\*[MAX\_STOPS];

    }

    ~Route() {

        delete[] stops;

    }

    void addStop(Stop\* stop) {

        if(stopCount < MAX\_STOPS) {

            stops[stopCount++] = stop;

            cout << "Stop added to route " << routeId << endl;

        }

    }

    void displayStops() {

        cout << "Route " << routeId << " stops:\n";

        for(int i = 0; i < stopCount; i++) {

            cout << stops[i]->getName() << " at " << stops[i]->getLocation() << endl;

        }

    }

};

class Student {

private:

    string id;

    string name;

    bool feePaid;

    string assignedStop;

    static int totalStudents;

public:

    Student(string i, string n, string stop):

        id(i), name(n), feePaid(false), assignedStop(stop) {

        totalStudents++;

    }

    string getId() { return id; }

    bool isFeesPaid() { return feePaid; }

    void payFees() {

        feePaid = true;

        cout << "Fees paid for student " << name << endl;

    }

    void recordAttendance() {

        if(feePaid) {

            cout << "Attendance recorded for " << name << endl;

        } else {

            cout << "Card inactive. Please pay fees." << endl;

        }

    }

    void display() {

        cout << "Student: " << name << " (ID: " << id << ")\n"

             << "Stop: " << assignedStop << "\n"

             << "Fees Status: " << (feePaid ? "Paid" : "Unpaid") << endl;

    }

};

int Student::totalStudents = 0;

class TransportSystem {

private:

    Student\*\* students;

    Route\*\* routes;

    int studentCount;

    int routeCount;

    static const int MAX\_STUDENTS = 100;

    static const int MAX\_ROUTES = 10;

public:

    TransportSystem() {

        students = new Student\*[MAX\_STUDENTS];

        routes = new Route\*[MAX\_ROUTES];

        studentCount = routeCount = 0;

    }

    ~TransportSystem() {

        for(int i = 0; i < studentCount; i++) delete students[i];

        for(int i = 0; i < routeCount; i++) delete routes[i];

        delete[] students;

        delete[] routes;

    }

    void registerStudent(string id, string name, string stop) {

        if(studentCount < MAX\_STUDENTS) {

            students[studentCount++] = new Student(id, name, stop);

            cout << "Student registered successfully\n";

        }

    }

    void addRoute(Route\* route) {

        if(routeCount < MAX\_ROUTES) {

            routes[routeCount++] = route;

        }

    }

    void processCardTap(string studentId) {

        for(int i = 0; i < studentCount; i++) {

            if(students[i]->getId() == studentId) {

                students[i]->recordAttendance();

                return;

            }

        }

        cout << "Student not found\n";

    }

    void displayAllStudents() {

        for(int i = 0; i < studentCount; i++) {

            students[i]->display();

            cout << endl;

        }

    }

};

int main() {

    TransportSystem system;

    cout<<"\nShamveel Khan\n24k-0962\n";

    cout << "=== FAST Transportation System Demo ===\n\n";

    Stop\* stop1 = new Stop("Stop1", "Location A");

    Stop\* stop2 = new Stop("Stop2", "Location B");

    Stop\* stop3 = new Stop("Stop3", "Location C");

    Route\* route1 = new Route("R1");

    route1->addStop(stop1);

    route1->addStop(stop2);

    route1->addStop(stop3);

    cout << "\nDisplaying Route Information:\n";

    route1->displayStops();

    cout << "\nRegistering Students:\n";

    system.registerStudent("S1", "Shamveel", "Stop1");

    system.registerStudent("S2", "Moiz", "Stop2");

    system.registerStudent("S3", "Usman", "Stop3");

    cout << "\nInitial Student Status:\n";

    system.displayAllStudents();

    cout << "\nTesting Card Taps (Before Fee Payment):\n";

    system.processCardTap("S1");

    system.processCardTap("S2");

    cout << "\nProcessing Fee Payments:\n";

    Student\* student1 = new Student("S1", "Shamveel", "Stop1");

    student1->payFees();

    cout << "\nTesting Card Taps (After Fee Payment):\n";

    system.processCardTap("S1");

    cout << "\nInvalid Card Tap Test:\n";

    system.processCardTap("S99");

    delete stop1;

    delete stop2;

    delete stop3;

    delete route1;

    delete student1;

    return 0;

}

