Assignment 1

Rafay Siddiqui 24K-0009

# Q1

#include <iostream>

#include <string>

using namespace std;

const int MAX\_ARR\_SIZE = 2;

class Skill

{

private:

    static int timesCalled;

    string skillName,description;

    int skillID;

public:

    //default constructor

    Skill(){

        this->skillID = -1;

        this->skillName = "NA";

        this->description = "NA";

    };

    // para constructor

    Skill(int skillID, string skillName, string description){

        this->skillID = skillID;

        this->skillName = skillName;

        this->description = description;

    }

    // copy constructor

    Skill(Skill &s){

        this->skillID = s.skillID;

        this->skillName = s.skillName;

        this->description = s.description;

    }

    int getSkillID(){return this->skillID;}

    void setSkillID(int sID){this->skillID=sID;}

    void showSkillDetails(){

        cout << "Skill ID: " << this->skillID << endl;

        cout << "skillName: " << this->skillName << endl;

        cout << "description: " << this->description << endl;

    }

    void updateSkillDescription(string newDescription){

        this->description = newDescription;

    }

    ~Skill(){

        cout << timesCalled++ << " Skill object destroyed." << endl;

    }

};

int Skill::timesCalled = 0;

class Sport

{

private:

    int skillsCount=0;

    string name,description;

    Skill requiredSkills[MAX\_ARR\_SIZE];

    int sportID;

public:

    Sport(){

        this->sportID = -1;

        this->name = "NA";

        this->description = "NA";

    }

    Sport(int sportID, string name, string description, Skill requiredSkills[])

    {

        this->sportID = sportID;

        this->name = name;

        this->description = description;

        for (size\_t i = 0; i < MAX\_ARR\_SIZE; i++)

        {

            this->requiredSkills[i] = requiredSkills[i];

        }

    }

    void addSkill(Skill s){

        requiredSkills[skillsCount++] = s;

    }

    void removeSkill(Skill s){

        for (size\_t i = 0; i < skillsCount; i++)

        {

            if (requiredSkills[i].getSkillID() == s.getSkillID())

            {

                requiredSkills[i].setSkillID(-1);

                skillsCount--;

            }

        }

    }

    ~Sport(){

        cout << "Sport object destroyed." << endl;

    }

};

class Mentor;

class Student

{

private:

    int sportsInterest=0;

    string name;

    Sport sportsInterests[MAX\_ARR\_SIZE];

    string mentorAssigned;

    int age,studentID;

public:

    Student(){

        this->age = -1;

        this->studentID = -1;

        this->name = "NA";

    }

    Student(int age, int studentID, string name,string mentorAssigned, Sport sportsInterests[]){

        this->age = age;

        this->studentID = studentID;

        this->name = name;

        this->mentorAssigned = mentorAssigned;

        for (size\_t i = 0; i < MAX\_ARR\_SIZE; i++)

        {

            this->sportsInterests[i] = sportsInterests[i];

        }

    }

    void setStudentID(int studentID){this->studentID= studentID;}

    int getStudentID(){return this->studentID;}

    int getStudentAge(){return this->age;}

    string getStudentName(){return this->name;}

    void registerForMentorship(Mentor \*m);

    /\* {

        m->assignLearner(\*this);

    } \*/

    void viewMentorDetails(Mentor \*m);

    /\* {

        m->showMentorDetails();

    } \*/

    void updateSportsInterest(Sport s){

        sportsInterests[sportsInterest++] = s;

    }

    ~Student(){

        cout << "Student object destroyed." << endl;

    }

};

class Mentor

{

    int underMentorship=0;

    string name,sportsExpertise[MAX\_ARR\_SIZE];

    int mentorID,maxLearners = 2;

    Student assignedLearners[MAX\_ARR\_SIZE]; // 10 is the max limit of learners a mentor can have

public:

    // default constructor

    Mentor(){

        this->mentorID = -1;

        this->maxLearners = -1;

        this->name = "NA";

        for (int i = 0; i < MAX\_ARR\_SIZE; i++) {

            this->sportsExpertise[i] = "";

        }

        for (int i = 0; i < MAX\_ARR\_SIZE; i++) {

            this->assignedLearners[i] = Student();

        }

    }

    // parameterized constructor

    Mentor(int mentorID, string name,  int maxLearners,Student assignedLearners[]){

        this->mentorID = mentorID;

        this->maxLearners = maxLearners;

        this->name = name;

        for (int i = 0; i < MAX\_ARR\_SIZE; i++) {

            this->sportsExpertise[i] = "";

        }

        for (size\_t i = 0; i < maxLearners; i++)

        {

            this->assignedLearners[i] = assignedLearners[i];

        }

    }

    void showMentorDetails(){

        cout << "Mentor ID: " << this->mentorID << endl;

        cout << "Mentor Name: " << this->name << endl;

        cout << "Mentor Expertise: " << this->sportsExpertise << endl;

    }

    void assignLearner(Student &s){

        if (underMentorship < maxLearners) {

            assignedLearners[underMentorship++] = s;

        } else {

            cout << "Cannot assign more learners, capacity full." << endl;

        }

    }

    void removeLearner(Student s){

        for (size\_t i = 0; i < maxLearners; i++)

        {

            if (assignedLearners[i].getStudentID() == s.getStudentID())

            {

                assignedLearners[i].setStudentID(-1);

                underMentorship--;

            }

            /\* code \*/

        }

    }

    void viewLearners(){

        for (size\_t i = 0; i < underMentorship; i++)

        {

            cout << "Student ID: " << assignedLearners[i].getStudentID() << endl;

            cout << "Student Age: " << assignedLearners[i].getStudentAge() << endl;

            cout << "Student Name: " << assignedLearners[i].getStudentName() << endl;

        }

    }

    void provideGuidance(){

        cout << "Guidance provided." << endl;

    }

    ~Mentor(){

        cout << "Mentor object destroyed." << endl;

    }

};

// int Mentor::underMentorship = 0;

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

        m->assignLearner(\*this);

    }

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

    m->showMentorDetails();

}

int main() {

    // Skills function correctly

    cout << "Skills" << endl;

    Skill skill1(1, "Skill1", "Skill1 Description");

    Skill skill2(2, "Skill2", "Skill2 Description");

    Skill skill3(3, "Skill3", "Skill3 Description");

    cout << "skills function correctly" << endl;

    Skill skills[MAX\_ARR\_SIZE] = {skill1, skill2};

    // Sports function correctly

    cout << "Sports" << endl;

    Sport sp1(1, "Sport1", "Sport1 Description", skills);

    Sport sp2(2, "Sport2", "Sport2 Description", skills);

    cout << "sports function correctly" << endl;

    Student s1, s2;

    cout << "students function correctly" << endl;

    Student Learners[MAX\_ARR\_SIZE] = {s1, s2};

    cout << "Mentor" << endl;

    Mentor m1(1, "Mentor1", 2, Learners);

    cout << "Mentor function correctly" << endl;

    Sport Sports[MAX\_ARR\_SIZE] = {sp1, sp2};

    cout << "new sports arr" << endl;

    cout << "new student defined by para const" << endl;

    Student st1(20, 1, "Student1", "mentor1", Sports);

    cout << "defined successfully" << endl;

    st1.registerForMentorship(&m1);

    st1.viewMentorDetails(&m1);

    cout << "exiting..." << endl;

    return 0;

}

## Output

Skills

skills function correctly

Sports

sports function correctly

students function correctly

Mentor

Mentor function correctly

new sports arr

new student defined by para const

defined successfully

Mentor ID: 1

Mentor Name: Mentor1

Mentor Expertise: 0x72313fe6a8

exiting...

Student object destroyed.

Sport object destroyed.

0 Skill object destroyed.

1 Skill object destroyed.

Sport object destroyed.

2 Skill object destroyed.

3 Skill object destroyed.

Sport object destroyed.

4 Skill object destroyed.

5 Skill object destroyed.

Sport object destroyed.

6 Skill object destroyed.

7 Skill object destroyed.

Mentor object destroyed.

Student object destroyed.

Sport object destroyed.

8 Skill object destroyed.

9 Skill object destroyed.

Sport object destroyed.

10 Skill object destroyed.

11 Skill object destroyed.

Student object destroyed.

Sport object destroyed.

12 Skill object destroyed.

13 Skill object destroyed.

Sport object destroyed.

14 Skill object destroyed.

15 Skill object destroyed.

Student object destroyed.

Sport object destroyed.

16 Skill object destroyed.

17 Skill object destroyed.

Sport object destroyed.

18 Skill object destroyed.

19 Skill object destroyed.

Student object destroyed.

Sport object destroyed.

20 Skill object destroyed.

21 Skill object destroyed.

Sport object destroyed.

22 Skill object destroyed.

23 Skill object destroyed.

Student object destroyed.

Sport object destroyed.

24 Skill object destroyed.

25 Skill object destroyed.

Sport object destroyed.

26 Skill object destroyed.

27 Skill object destroyed.

Student object destroyed.

Sport object destroyed.

28 Skill object destroyed.

29 Skill object destroyed.

Sport object destroyed.

30 Skill object destroyed.

31 Skill object destroyed.

Sport object destroyed.

32 Skill object destroyed.

33 Skill object destroyed.

Sport object destroyed.

34 Skill object destroyed.

35 Skill object destroyed.

36 Skill object destroyed.

37 Skill object destroyed.

38 Skill object destroyed.

39 Skill object destroyed.

40 Skill object destroyed.

# Q2

#include <iostream>

using namespace std;

class Robot

{

public:

    string name;

    int hits;

    void hitBall(){

        this->hits++;

    }

    Robot(){

        this->name = "NA";

        this->hits = 0;

    }

};

class ball

{

    int x,y;

public:

    int getX(){

        return x;

    }

    int getY(){

        return y;

    }

    void move(int dx, int dy,string move){

        if (move == "up")

        {

            dx\*=0;

            dy\*=1;

        }

        if (move == "down")

        {

            dx\*=0;

            dy\*=-1;

        }

        if (move == "left")

        {

            dx\*=-1;

            dy\*=0;

        }

        if (move == "right")

        {

            dx\*=1;

            dy\*=0;

        }

        this->x += dx;

        this->y += dy;

    }

    /\* ball(){

        this->x = 0;

        this->y = 0;

    } \*/

    ball(int x=0, int y=0){

        this->x = x;

        this->y = y;

    }

};

class goal

{

public:

    int x,y;

    bool isGoalReached(int &ballX, int &ballY){

       if (ballX == this->x && ballY == this->y)

       {

           return true;

       }

       return false;

    }

    goal(int x = 3, int y = 3){

        this->x = x;

        this->y = y;

    }

};

class Team

{

public:

    string teamName;

    Robot \*robot;

    Team(string teamName="NA", Robot \*robot = new Robot()){

        this->teamName = teamName;

        this->robot = robot;

    }

    ~Team(){

        cout << "Team object destroyed." << endl;

        delete robot;

    }

};

class game

{

public:

    Team teamOne, teamTwo;

    ball Ball;

    goal Goal;

    void play(Team \*team){

        int ballX = Ball.getX();

        int ballY = Ball.getY();

        string direction;

        cout<< "ball position: " << Ball.getX()<<", "<< Ball.getY() << endl;

        cout<< "enter direction for " << team->teamName << " robot: ";

        cin >> direction;

        cout<< "enter x direction for " << team->teamName << " robot: ";

        cin >> ballX;

        cout<< "enter y direction for " << team->teamName << " robot: ";

        cin >> ballY;

        // TODO: hit ball / move ball conflict, which one should move the ball?,

        team->robot->hitBall();

        Ball.move(ballX, ballY,direction);

    }

    void declareWinner(){

        if (teamOne.robot->hits < teamTwo.robot->hits)

        {

            cout << "Team One Wins!" << endl;

        }

        else if (teamOne.robot->hits > teamTwo.robot->hits)

        {

            cout << "Team Two Wins!" << endl;

        }

        else

        {

            cout << "It's a draw!" << endl;

        }

    }

    void startGame(){

        int ballX = Ball.getX();

        int ballY = Ball.getY();

        cout << "Game Started!" << endl;

        cout << "Team One: " << teamOne.teamName << endl;

        cout << "Team Two: " << teamTwo.teamName << endl;

        cout << "goal position: " << Goal.x<<", "<< Goal.y<< endl;

        while (true)

        {

            if (!Goal.isGoalReached(ballX, ballY))

            {

                play(&teamOne);

                ballX = Ball.getX();

                ballY = Ball.getY();

            }

            if (!Goal.isGoalReached(ballX, ballY))

            {

                play(&teamTwo);

                ballX = Ball.getX();

                ballY = Ball.getY();

            }

            else

            {

                break;

            }

        }

        declareWinner();

    }

    game(){

        teamOne;

        teamTwo;

        Ball = ball(0, 0);

        Goal = goal();

    }

};

int main(){

    // all members are public

    game game1;

    game1.teamOne.teamName = "Team One";

    game1.teamTwo.teamName = "Team Two";

    game1.teamOne.robot->name = "Robot One";

    game1.teamTwo.robot->name = "Robot Two";

    game1.startGame();

    return 0;

}

## Output

Game Started!

Team One: Team One

Team Two: Team Two

goal position: 3, 3

ball position: 0, 0

enter direction for Team One robot: up

enter x direction for Team One robot: 4

enter y direction for Team One robot: 3

ball position: 0, 3

enter direction for Team Two robot: left

enter x direction for Team Two robot: 0

enter y direction for Team Two robot: 2

ball position: 0, 3

enter direction for Team One robot: left

enter x direction for Team One robot: 1

enter y direction for Team One robot: 2

ball position: -1, 3

enter direction for Team Two robot: right

enter x direction for Team Two robot: 3

enter y direction for Team Two robot: 2

ball position: 2, 3

enter direction for Team One robot: right

enter x direction for Team One robot: 1

enter y direction for Team One robot: 1

Team Two Wins!

Team object destroyed.

Team object destroyed.

# Q3

#include <iostream>

using namespace std;

class User

{

static int counter;

    int Age, UserID; // will increment for each new user

    string licenseType,contactInfo;

    public:

    User(int age = 18, string licenseType = "Learner",string contactInfo = "123456789")

        :   Age(age),

            licenseType(licenseType),

            contactInfo(contactInfo)

            {

                this->UserID=counter++;

            }

            /\* { by default counter will assign id

        this->Age = age;

        this->licenseType = licenseType;

        this->contactInfo = contactInfo;

        this->UserID=counter++;

    } \*/

/\*

    // setters not needed as constructor is doing the job

    void setAge(int age){

        this->Age = age;

    }

    void setLicenseType(string licenseType){

        this->licenseType = licenseType;

    }

    void setContactInfo(string contactInfo){

        this->contactInfo = contactInfo;

    } \*/

    // getters

    int getAge(){

        return this->Age;

    }

    string getLicenseType(){

        return this->licenseType;

    }

    string getContactInfo(){

        return this->contactInfo;

    }

};

int User::counter = 0;

class vehicle

{

private:

    string Model; //  Eligibility; for different license types

    double rentalPrice;

    bool rented;

public:

    vehicle(string mod= "NA", double price=0.0)

    {

        this->Model = mod;

        this->rentalPrice = price;

        this->rented = false;

    }

    string getModel(){

        return this->Model;

    }

    double getRentalPrice(){

        return this->rentalPrice;

    }

    double isRented(){

        return this->rented;

    }

    void setRentStatus(bool rented){

        this->rented = rented;

    }

};

class System

{

private:

    string learner[3] = {"Toyota Yaris", "Honda Fit", "Ford Fiesta"};

    string intermediate[3] = {"Toyota Corolla", "Honda Civic", "Ford Focus"};

    string fullLicense[3] = {"Toyota Camry", "Honda Accord", "Ford Mustang"};

public:

    bool Eligible(User &user, vehicle &vehicle){

        // check licence validity

        if (user.getLicenseType()=="learner")

        {

            if (

                vehicle.getModel() == learner[0] ||

                vehicle.getModel() == learner[1] ||

                vehicle.getModel() == learner[2]

                )

            {

                return true;

            }

            else

            {

                return false;

            }

        }

        else if (user.getLicenseType()=="intermediate")

        {

            if (

                vehicle.getModel() == intermediate[0] ||

                vehicle.getModel() == intermediate[1] ||

                vehicle.getModel() == intermediate[2]

                )

            {

                return true;

            }

            else

            {

                return false;

            }

        }

        else if (user.getLicenseType()=="full")

        {

            if (

                vehicle.getModel() == fullLicense[0] ||

                vehicle.getModel() == fullLicense[1] ||

                vehicle.getModel() == fullLicense[2]

                )

            {

                return true;

            }

            else

            {

                return false;

            }

        }

        else

        {

            return false;

        }

    }

    void rentVehicle(User user, vehicle &vehicle){

        if (Eligible(user, vehicle))

        {

            vehicle.setRentStatus(true);

            cout << "Vehicle Rented: " << vehicle.getModel() << " for " << vehicle.getRentalPrice() << " per day" << endl;

        }

        else

        {

            cout << "You are not eligible to rent this vehicle" << endl;

        }

    }

};

int main(){

    /\* string learner[3] = {"Toyota Yaris", "Honda Fit", "Ford Fiesta"};

    string intermediate[3] = {"Toyota Corolla", "Honda Civic", "Ford Focus"};

    string fullLicense[3] = {"Toyota Camry", "Honda Accord", "Ford Mustang"}; \*/

    System system;

    User

    user1(20, "full", "123456789"),

    user2(18, "learner", "987654321"),

    user3(25, "intermediate", "123123123");

    vehicle

    \*vehicle1 = new vehicle("Toyota Camry", 100.0),

    \*vehicle2 = new vehicle("Toyota Corolla", 90.0),

    \*vehicle3 = new vehicle("Toyota Yaris", 80.0);

    // only user1 can rent vehicle1, vehicle2, vehicle3

    system.rentVehicle(user1, \*vehicle1);

    system.rentVehicle(user2, \*vehicle2);

    system.rentVehicle(user3, \*vehicle3);

    return 0;

}

## Output

Vehicle Rented: Toyota Camry for 100 per day

You are not eligible to rent this vehicle

You are not eligible to rent this vehicle

# Q4

#include <iostream>

using namespace std;

const int MAX\_ARR\_SIZE = 5,daysInMonth = 30 ;

class Card

{

private:

static int incrementor;

    bool active,isTapped;

    int cardID;

public:

    bool getActive(){

        return this->active;

    }

    void setActive(bool b){

        this->active = b;

    }

    bool getTapped(){

        return this->isTapped;

    }

    void setTapped(bool b){

        this->isTapped = b;

    }

    int getCardID(){

        return this->cardID;

    }

    Card(bool active=false, bool isTapped=false ){

        this->active = active;

        this->isTapped = isTapped;

        this->cardID = incrementor++;

    }

    ~Card(){

        cout << "Card object destroyed." << endl;

    }

};

int Card::incrementor = 0;

class Student{

static int incrementor;

    // a student can pay fees but not have an active card, a card can only be active if the student has paid the fees, a transport can only provided if both are true

    bool paidSemFee,registeredForTransport, attendance[daysInMonth];

    string name;

    int rollNO;

    Card \*card;

public:

    string getName(){

        return this->name;

    }

    bool getPaidSemFee(){

        return this->paidSemFee;

    }

    bool getCardActive(){

        return this->card->getActive();

    }

    void setCardActive(bool b){

        this->card->setActive(b);

    }

    bool getTransportStatus(){

        return this->registeredForTransport;

    }

    void setTransportStatus(bool b){

        this->registeredForTransport = b;

    }

    Student(

        bool cardActive = false,

        bool paidSemFee=false,

        bool registeredForTransport = false,

        string name="NA")

        :

        card(new Card(cardActive)),

        paidSemFee(paidSemFee),

        registeredForTransport(registeredForTransport),

        name(name),

        rollNO(incrementor++)

        {

        /\* this->paidSemFee = paidSemFee;

        this->registeredForTransport = registeredForTransport;

        this->name = name;

        this->rollNO = incrementor++;\*/

        for (size\_t i = 0; i < daysInMonth; i++)

        {

            attendance[i] = false; // for the entire month

        }

    }

    void markAttendance(int dayOfMonth){

        if (getTransportStatus()){

            attendance[dayOfMonth] = true;

        } else {

            cout << "illegal student being marked." << endl;

        }

    }

    void displaySheet(){

        for (size\_t i = 0; i < daysInMonth; i++)

        {

            cout <<  attendance[i] << " ";

        }

    }

    ~Student(){

        delete card;

        cout << "Student object destroyed." << endl;

    }

};

int Student::incrementor = 0;

// class does not explicitly  define a bus or a van, rather the overall service

class Transport

{

    private:

    static int incrementor,transport;

    string route[2];

    int TransportID;

    Student \*students[MAX\_ARR\_SIZE];

public:

    Transport(){

        this->TransportID = incrementor++;

        this->route[0] = "NA0";

        this->route[1] = "NA1";

        for (int i = 0; i < MAX\_ARR\_SIZE; i++) {

            this->students[i] = nullptr; // initialize array with nullptr

        }

    }

    void registerStudent(Student \*s){

        if (s->getPaidSemFee() == true && transport < MAX\_ARR\_SIZE && s->getCardActive() == true){

            students[transport++] = s;

            s->setTransportStatus(true);

        } else {

            s->setTransportStatus(false);

            cout << "error registering " << s->getName()<<" (check for card status, sem fee status, system transport capability status)." << endl;

        }

    }

    ~Transport(){

        // delete all students

        for (size\_t i = 0; i < MAX\_ARR\_SIZE; i++)

        {

            delete students[i];

        }

        cout << "Transport object destroyed." << endl;

    }

};

int Transport::incrementor = 0;

int Transport::transport = 0;

int main(){

    // cout << "creating Transport object ." << endl;

    Transport t1;

    // cout << "Transport object created." << endl;

    //

    // cout << "creating Student object1 ." << endl;

    // rafay has not paid the fees, so his attendance will not be marked

    Student s1(false, true, true, "Rafay");

    // cout << "created Student object1 ." << endl;

    // cout << "creating Student object2 ." << endl;

    // ali has paid the fees and has an active card and has registered for transport, so his attendance is marked

    Student s2(true, true, true, "Ali");

    // cout << "created Student object2 ." << endl;

    // cout << "creating Student object3 ." << endl;

    // ali has paid the fees and has an active card but did not register for transport, so his attendance will be marked

    Student s3(true, true, false, "Saad");

    // cout << "created Student object3 ." << endl;

    // cout << "registering s1." << endl;

    t1.registerStudent(&s1); // s1 will fail registration as he has not paid the fees

    // cout << "registered s1." << endl;

    // cout << "registering s2." << endl;

    t1.registerStudent(&s2);

    // cout << "registered s2." << endl;

    // cout << "registering s3." << endl;

    t1.registerStudent(&s3);

    // cout << "registered s3." << endl;

    // simulating attendance for 4 days

    int day = 0;

    // day 1

    s1.markAttendance(day);

    s2.markAttendance(day);

    s3.markAttendance(day);

    // day 2

    day++;

    s1.markAttendance(day);

    s2.markAttendance(day);

    // day 3

    day++;

    s1.markAttendance(day);

    s2.markAttendance(day);

    // day 4

    day++;

    s1.markAttendance(day);

    s3.markAttendance(day);

    cout << "Attendance for s1: ";

    s1.displaySheet();

    cout << endl;

    cout << "Attendance for s2: ";

    s2.displaySheet();

    cout << endl;

    cout << "Attendance for s3: ";

    s3.displaySheet();

    cout << endl;

    return 0;

}

error registering Rafay (check for card status, sem fee status, system transport capability status).

illegal student being marked.

illegal student being marked.

illegal student being marked.

illegal student being marked.

Attendance for s1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Attendance for s2: 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Attendance for s3: 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Card object destroyed.

Student object destroyed.

Card object destroyed.

Student object destroyed.

Card object destroyed.

Student object destroyed.

Card object destroyed.