Shamveel Khan

24K-0962

OOP theory Assignment 2

QUESTION 1:

#include <iostream>

#include <string>

using namespace std;

class person {

protected:

string id;

string name;

string assignedStop;

bool feePaid;

public:

person(string i, string n, string stop, bool fee)

: id(i), name(n), assignedStop(stop), feePaid(fee) {}

virtual void display() {

cout << "..................\n";

cout << "ID: " << id << endl;

cout << "Name: " << name << endl;

cout << "Assigned Stop: " << assignedStop << endl;

}

string getId() const { return id; }

bool getFeePaid() const { return feePaid; }

};

class Teacher : public person {

public:

Teacher(string i, string n, string stop)

: person(i, n, stop, false) {}

void payFees(int monthsPaid) {

if (monthsPaid >= 1) {

feePaid = true;

cout << name << " (Teacher) has paid fees for " << monthsPaid << " month(s).\n";

} else {

cout << "Invalid payment duration.\n";

}

}

void recordAttendance() {

if (feePaid) {

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

} else {

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

}

}

};

class Staff : public person {

public:

Staff(string i, string n, string stop)

: person(i, n, stop, true) {}

void recordAttendance() {

cout << "Attendance recorded for " << name << " (Staff, Free Transport)\n";

}

};

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;

}

}

bool operator==(const Route& other) const {

if (routeId != other.routeId) return false;

if (stopCount != other.stopCount) return false;

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

if (stops[i]->getName() != other.stops[i]->getName() ||

stops[i]->getLocation() != other.stops[i]->getLocation()) {

return false;

}

}

return true;

}

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 : public person {

private:

static int totalStudents;

public:

Student(string i, string n, string stop)

: person(i, n, stop, false) {

totalStudents++;

}

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;

Teacher\*\* teachers;

Staff\*\* staffMembers;

Route\*\* routes;

int studentCount, teacherCount, staffCount, routeCount;

static const int MAX\_STUDENTS = 100;

static const int MAX\_TEACHERS = 20;

static const int MAX\_STAFF = 20;

static const int MAX\_ROUTES = 10;

public:

TransportSystem() {

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

teachers = new Teacher\*[MAX\_TEACHERS];

staffMembers = new Staff\*[MAX\_STAFF];

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

studentCount = teacherCount = staffCount = routeCount = 0;

}

~TransportSystem() {

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

for(int i = 0; i < teacherCount; i++) delete teachers[i];

for(int i = 0; i < staffCount; i++) delete staffMembers[i];

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

delete[] students;

delete[] teachers;

delete[] staffMembers;

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 registerTeacher(string id, string name, string stop) {

if(teacherCount < MAX\_TEACHERS) {

teachers[teacherCount++] = new Teacher(id, name, stop);

cout << "Teacher registered successfully\n";

}

}

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

if(staffCount < MAX\_STAFF) {

staffMembers[staffCount++] = new Staff(id, name, stop);

cout << "Staff registered successfully\n";

}

}

void processCardTap(string personId) {

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

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

students[i]->recordAttendance();

return;

}

}

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

if(teachers[i]->getId() == personId) {

teachers[i]->recordAttendance();

return;

}

}

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

if(staffMembers[i]->getId() == personId) {

staffMembers[i]->recordAttendance();

return;

}

}

cout << "Person not found in the system\n";

}

void payStudentFee(string id) {

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

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

students[i]->payFees();

return;

}

}

cout << "Student not found!\n";

}

void payTeacherFee(string id, int months) {

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

if(teachers[i]->getId() == id) {

teachers[i]->payFees(months);

return;

}

}

cout << "Teacher not found!\n";

}

};

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 People:\n";

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

system.registerTeacher("T1", "Dr. Ahmed", "Stop2");

system.registerStaff("ST1", "Mr. Khan", "Stop3");

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

system.processCardTap("S1");

system.processCardTap("T1");

system.processCardTap("ST1");

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

system.payStudentFee("S1");

system.payTeacherFee("T1", 1);

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

system.processCardTap("S1");

system.processCardTap("T1");

system.processCardTap("ST1");

delete stop1;

delete stop2;

delete stop3;

delete route1;

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

QUESTION 2:

#include <iostream>

using namespace std;

class ghost {

protected:

string name;

string type;

int scareLevel;

public:

ghost(string n, int s, string t) : name(n), scareLevel(s), type(t) {}

virtual ~ghost() {}

int getScarelevel() {

return scareLevel;

}

virtual void haunt() const {

cout << name << " is haunting!" << endl;

}

virtual void display() const {

cout << "\n.................\n";

cout << "Name: " << name << endl;

cout << "Type: " << type << endl;

cout << "Scare Level: " << scareLevel << endl;

cout << "\n.................\n";

}

friend ostream& operator<<(ostream& out, const ghost& g) {

out << "Name: " << g.name << "\n"

<< "Type: " << g.type << "\n"

<< "Scare Level: " << g.scareLevel << "\n";

return out;

}

friend ghost operator+(const ghost& g1, const ghost& g2) {

string newName = g1.name + " & " + g2.name;

int newLevel = g1.scareLevel + g2.scareLevel;

return ghost(newName, newLevel, "Mixed ghost");

}

};

class Poltergeist : virtual public ghost {

public:

Poltergeist(string n, int s) : ghost(n, s, "Poltergeist") {}

void haunt() const override {

cout << name << " moves objects!" << endl;

}

};

class Banshee : public ghost {

public:

Banshee(string n, int s) : ghost(n, s, "Banshee") {}

void haunt() const override {

cout << name << " screams loudly!" << endl;

}

};

class Shadowghost : virtual public ghost {

public:

Shadowghost(string n, int s) : ghost(n, s, "Shadowghost") {}

void haunt() const override {

cout << name << " whispers creepily..." << endl;

}

};

class ShadowPoltergeist : public Poltergeist, public Shadowghost {

public:

ShadowPoltergeist(string n, int s)

: ghost(n, s, "ShadowPoltergeist"), Poltergeist(n, s), Shadowghost(n, s) {}

void haunt() const override {

Poltergeist::haunt();

Shadowghost::haunt();

}

};

class visitor {

string name;

int braveryLevel;

public:

visitor(string n,int bl):name(n),braveryLevel(bl) {}

void react(int g) {

if(g+1>braveryLevel) {

cout<<"\nVisitor ran away!!\n";

}

else if(g+1<braveryLevel) {

cout<<"\nVisitor laughs!!\n";

}

else {

cout<<"\nVisitor Stays!!\n";

}

}

void display() const {

cout << "\n.................\n";

cout<<"Name: "<<name<<endl;

cout<<"Bravery Level: "<<braveryLevel<<endl;

cout << "\n.................\n";

}

};

class hauntedHouse {

string name;

ghost\*\* listGhosts;

int ghostCount;

int TotalGhostCapacity;

public:

hauntedHouse(string n,int tgc): name(n),TotalGhostCapacity(tgc) {

listGhosts = new ghost\*[TotalGhostCapacity];

ghostCount=0;

}

void addGhost(ghost\* g) {

listGhosts[ghostCount++]=g;

}

void displayGhost() const {

cout<<"\n---Displaying All Ghost---\n";

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

listGhosts[i]->display();

cout<<" \n ";

}

cout<<".......................\n";

}

int totalScareLevels() {

int sum=0;

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

sum += listGhosts[i]->getScarelevel();

}

return sum;

}

void visit(visitor\* list,int visitorCount) {

int j=0;

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

list[i].react(listGhosts[j%ghostCount]->getScarelevel());

cout<<endl;

j++;

}

}

};

int main() {

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

hauntedHouse hh("Bhoot Bangla", 5);

ghost\* g1 = new Poltergeist("Shamveel", 5);

ghost\* g2 = new Banshee("Moiz", 8);

ghost\* g3 = new Shadowghost("Usman", 6);

ghost\* g4 = new ShadowPoltergeist("Kabeer", 10);

ghost\* g5 = new Poltergeist("Abdullah", 4);

hh.addGhost(g1);

hh.addGhost(g2);

hh.addGhost(g3);

hh.addGhost(g4);

hh.addGhost(g5);

hh.displayGhost();

cout << "Total Scare Level: " << hh.totalScareLevels() << endl;

visitor v1("Ali",7);

visitor v2("Hassan",9);

visitor v3("Hamza",5);

visitor visitors[3] = {v1,v2,v3};

hh.visit(visitors, 3);

delete g1;

delete g2;

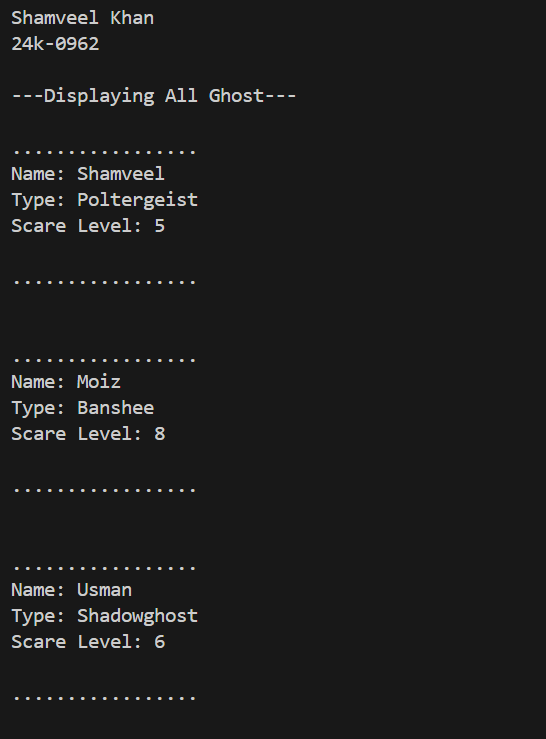
delete g3;

delete g4;

delete g5;

return 0;

}



A screenshot of a computer

AI-generated content may be incorrect.

QUESTION 3:

#include <iostream>

#include <string>

using namespace std;

class Vehicle

{

protected:

string id;

int speed;

int capacity;

int energyEfficiency;

static int totalDeliveries;

public:

Vehicle(string id) : id(id) {}

virtual void calculateRoute() = 0;

virtual double estimateTime(double distance) = 0;

virtual void command(string packageID)

{

totalDeliveries++;

cout << "Basic delivery command for package: " << packageID << endl;

}

virtual void command(string packageID, string urgency)

{

totalDeliveries++;

cout << "Urgent delivery command for package: " << packageID << endl;

}

static int getTotalDeliveries() { return totalDeliveries; }

friend bool operator==(const Vehicle &a, const Vehicle &b);

friend Vehicle \*resolveConflict(Vehicle \*a, Vehicle \*b);

};

int Vehicle::totalDeliveries = 0;

bool operator==(const Vehicle &a, const Vehicle &b)

{

return (a.speed == b.speed) &&

(a.capacity == b.capacity) &&

(a.energyEfficiency == b.energyEfficiency);

}

Vehicle \*resolveConflict(Vehicle \*a, Vehicle \*b)

{

int aScore = a->speed \* a->energyEfficiency + a->capacity;

int bScore = b->speed \* b->energyEfficiency + b->capacity;

return aScore >= bScore ? a : b;

}

class RamzanDrone : public Vehicle

{

public:

RamzanDrone(string id) : Vehicle(id)

{

speed = 100;

capacity = 10;

energyEfficiency = 80;

}

void calculateRoute() override

{

cout << "Calculating aerial route for high-speed delivery" << endl;

}

double estimateTime(double distance) override

{

return distance / speed;

}

void command(string packageID, string urgency) override

{

if (urgency == "urgent")

{

cout << "Drone activating turbo mode for Iftar meal: " << packageID << endl;

speed \*= 2;

}

Vehicle::command(packageID, urgency);

}

};

class RamzanTimeShip : public Vehicle

{

public:

RamzanTimeShip(string id) : Vehicle(id)

{

speed = 80;

capacity = 50;

energyEfficiency = 70;

}

void calculateRoute() override

{

cout << "Verifying historical consistency for time travel" << endl;

}

double estimateTime(double distance) override

{

return distance / speed;

}

void command(string packageID, string urgency) override

{

if (urgency == "urgent")

{

cout << "TimeShip validating historical records for: " << packageID << endl;

}

Vehicle::command(packageID, urgency);

}

};

class RamzanHyperPod : public Vehicle

{

public:

RamzanHyperPod(string id) : Vehicle(id)

{

speed = 120;

capacity = 1000;

energyEfficiency = 90;

}

void calculateRoute() override

{

cout << "Navigating underground hyperloop network" << endl;

}

double estimateTime(double distance) override

{

return distance / speed;

}

};

int main()

{

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

RamzanDrone drone("DRONE");

RamzanTimeShip timeship("TIMESHIP");

RamzanHyperPod pod("POD");

drone.command("IFTAR\_MEAL1", "urgent");

drone.command("IFTAR\_MEAL2", "urgent");

timeship.command("HISTORICAL\_ITEM", "urgent");

pod.calculateRoute();

cout << "Estimated delivery time: " << pod.estimateTime(360) << " hours" << endl;

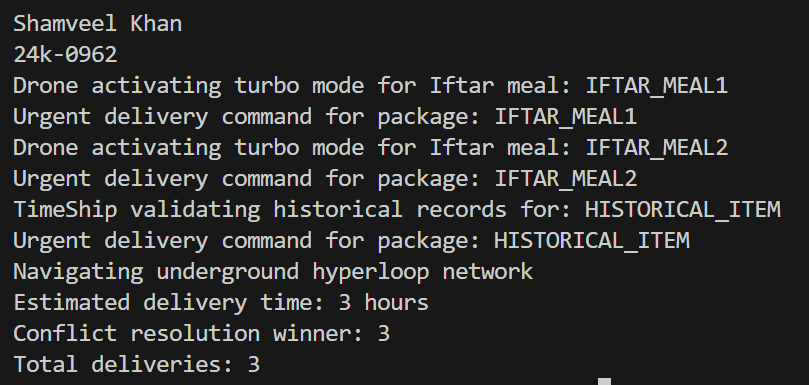
Vehicle \*result = resolveConflict(&drone, &pod);

cout << "Conflict resolution winner: " << result->getTotalDeliveries() << endl;

cout << "Total deliveries: " << Vehicle::getTotalDeliveries() << endl;

return 0;

}



QUESTION 4:

#include <iostream>

#include <string>

using namespace std;

class User

{

protected:

string name, id, email, password;

string role;

public:

User(string n, string i, string e, string pwd, string r)

: name(n), id(i), email(e), password(pwd), role(r) {}

bool authenticate(string pwd)

{

return password == pwd;

}

virtual void display()

{

cout << "Name: " << name << "\nID: " << id

<< "\nEmail: " << email << "\nRole: " << role << endl;

}

string getRole() { return role; }

};

class Student : public User

{

bool assignments[10] = {false};

int assignmentCount = 0;

public:

Student(string n, string i, string e, string pwd)

: User(n, i, e, pwd, "Student") {}

void addAssignment()

{

if (assignmentCount < 10)

assignmentCount++;

}

void submit(int index)

{

if (index < assignmentCount)

assignments[index] = true;

}

void display() override

{

User::display();

cout << "Assignments: " << assignmentCount << endl;

}

};

class TA : public Student

{

Student \*students[10];

string projects[2];

int studentCount = 0, projectCount = 0;

public:

TA(string n, string i, string e, string pwd)

: Student(n, i, e, pwd) { role = "TA"; }

bool addStudent(Student \*s)

{

if (studentCount < 10)

{

students[studentCount++] = s;

return true;

}

return false;

}

bool addProject(string p)

{

if (projectCount < 2)

{

projects[projectCount++] = p;

return true;

}

return false;

}

void display() override

{

User::display();

cout << "Managing " << studentCount << " students\n"

<< "Projects: " << projectCount << endl;

}

};

class Professor : public User

{

public:

Professor(string n, string i, string e, string pwd)

: User(n, i, e, pwd, "Professor") {}

bool assignProject(TA &ta, string p)

{

return ta.addProject(p);

}

void display() override

{

User::display();

cout << "Lab Supervisor" << endl;

}

};

void authenticateAndPerformAction(User \*user, string action)

{

string role = user->getRole();

if ((role == "Student" && action == "submit") ||

(role == "TA" && (action == "view" || action == "manage")) ||

(role == "Professor" && (action == "assign" || action == "full-access")))

{

cout << "Action permitted: " << action << endl;

}

else

{

cout << "Action denied: " << action << endl;

}

}

int main()

{

cout<<"Shamveel Khan -24K-0962\n";

Student s("Shamveel", "S001", "k240962@nu.edu.pk", "student123");

TA t("TA", "T001", "TA.com", "ta456");

Professor p("prof", "P001", "PROF.com", "prof789");

s.addAssignment();

s.submit(0);

t.addStudent(&s);

t.addProject("AI Research");

p.assignProject(t, "ML Project");

s.display();

cout << endl;

t.display();

cout << endl;

p.display();

cout << endl;

authenticateAndPerformAction(&s, "submit");

authenticateAndPerformAction(&t, "manage");

authenticateAndPerformAction(&p, "assign");

authenticateAndPerformAction(&s, "view");

cout << "\nAuthentication test: " << s.authenticate("student123") << endl;

return 0;

}

