**Q.1. A university is developing a student management system where each student has attributes like name, roll**

**number, and courses enrolled. The system needs to allow students to perform actions such as enrolling in courses**

**and checking grades.**

**Question: Explain the relationship between objects, classes, data members, and methods in OOP with reference to**

**this system.**

ANS:-

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Student {

private:

string name;

int rollNumber;

vector<string> coursesEnrolled;

vector<string> grades;

public:

Student(string studentName, int roll) {

name = studentName;

rollNumber = roll;

void enrollInCourse(string course) {

coursesEnrolled.push\_back(course);

grades.push\_back("Not graded"); // default grade

}

void assignGrade(string course, string grade) {

for (size\_t i = 0; i < coursesEnrolled.size(); ++i) {

if (coursesEnrolled[i] == course) {

grades[i] = grade;

return;

}

}

cout << "Course not found for this student." << endl;

}

void displayCourses() {

cout << "Courses enrolled by " << name << " (Roll No: " << rollNumber << "):" << endl;

for (const string& course : coursesEnrolled) {

cout << "- " << course << endl;

}

cout << endl;

}

void checkGrades() {

cout << "Grades for " << name << " (Roll No: " << rollNumber << "):" << endl;

for (size\_t i = 0; i < coursesEnrolled.size(); ++i) {

cout << coursesEnrolled[i] << ": " << grades[i] << endl;

}

cout << endl;

}

};

int main() {

Student s1("Gaurav", 101);

Student s2("Sachin", 102);

s1.enrollInCourse("Math");

s1.enrollInCourse("Physics");

s2.enrollInCourse("Chemistry");

s1.assignGrade("Math", "A");

s1.assignGrade("Physics", "B+");

s2.assignGrade("Chemistry", "A-");

s1.displayCourses();

s2.displayCourses();

s1.checkGrades();

s2.checkGrades();

return 0;

}

**Q.2. A company is designing an employee management system. Some employee details, such as salary and bank details,**

**should be hidden from other employees but accessible to HR. Other general details, like name and department,**

**should be visible to everyone.**

**Question: How does the private modifier differ from the protected modifier in OOP, and how can they be used to**

**implement data security in this scenario?**

ANS:-

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

double salary;

string bankDetails;

protected:

int employeeID;

double getSalary() const {

return salary;

}

string getBankDetails() const {

return bankDetails;

}

public:

string name;

string department;

Employee(string n, string dept, double sal, string bank, int id)

: name(n), department(dept), salary(sal), bankDetails(bank), employeeID(id) {}

void displayGeneralInfo() const {

cout << "Name: " << name << "\nDepartment: " << department << endl;

}

};

class HR : public Employee {

public:

HR(string n, string dept, double sal, string bank, int id)

: Employee(n, dept, sal, bank, id) {}

void viewSensitiveInfo() const {

cout << "\n[HR Access] Sensitive Information:" << endl;

cout << "Employee ID: " << employeeID << endl;

cout << "Salary: $" << getSalary() << endl;

cout << "Bank Details: " << getBankDetails() << endl;

}

};

int main() {

Employee emp1("Gaurav Naik", "IT", 50000.0, "HDFC - Acc#123456", 1001);

HR hr1("Yogita Naik", "HR", 60000.0, "SBI - Acc#987654", 1002);

cout << "=== General Employee Info ===" << endl;

emp1.displayGeneralInfo();

cout << "\n=== HR Viewing Info of HR Employee ===" << endl;

hr1.displayGeneralInfo();

hr1.viewSensitiveInfo();

return 0;

}

Q.3**.) An e-commerce application allows users to apply different discount coupons. Some coupons apply a flat discount,**

**while others apply a percentage-based discount. The system has a method applyDiscount() that behaves differently**

**based on the type of coupon**

**ANS:-**

#include <iostream>

using namespace std;

class Discount {

public:

void applyDiscount(double amount, double flat) {

cout << "[Compile-Time] Flat discount of ₹" << flat

<< " applied. Final amount: ₹" << amount - flat << endl;

}

void applyDiscount(double amount, int percent) {

double discount = (percent / 100.0) \* amount;

cout << "[Compile-Time] " << percent << "% discount applied. Final amount: ₹"

<< amount - discount << endl;

}

class Coupon {

public:

virtual void applyDiscount(double amount) {

cout << "[Run-Time] No discount applied. Amount: ₹" << amount << endl;

}

virtual ~Coupon() {}

};

class FlatDiscount : public Coupon {

public:

void applyDiscount(double amount) override {

double flat = 100.0;

cout << "[Run-Time] Flat ₹" << flat

<< " discount applied. Final amount: ₹" << amount - flat << endl;

}

};

class PercentageDiscount : public Coupon {

public:

void applyDiscount(double amount) override {

double percent = 10.0;

double discount = (percent / 100.0) \* amount;

cout << "[Run-Time] " << percent << "% discount applied. Final amount: ₹"

<< amount - discount << endl;

}

};

int main() {

cout << "=== Compile-Time Polymorphism ===" << endl;

Discount d;

d.applyDiscount(1000.0, 100.0);

d.applyDiscount(1000.0, 10);

cout << "\n=== Run-Time Polymorphism ===" << endl;

Coupon\* c1 = new FlatDiscount();

Coupon\* c2 = new PercentageDiscount();

c1->applyDiscount(1000.0); // Calls FlatDiscount version

c2->applyDiscount(1000.0); // Calls PercentageDiscount version

delete c1;

delete c2;

return 0;

}

**Q.4. A vehicle rental service provides different rental price calculations based on vehicle type. Cars have a base price**

**per day, trucks have an additional weight-based charge, and motorcycles have a mileage-based price**

**ANS:-**

#include <iostream>

using namespace std;

class Vehicle {

protected:

double basePricePerDay;

public:

Vehicle(double price) : basePricePerDay(price) {}

virtual double calculateRentalPrice(int days) {

return basePricePerDay \* days;

}

virtual void showRentalDetails(int days) {

cout << "Total Rental Price for " << days << " day(s): ₹" << calculateRentalPrice(days) << endl;

}

virtual ~Vehicle() {}

};

class Car : public Vehicle {

public:

Car(double price) : Vehicle(price) {}

double calculateRentalPrice(int days) override {

return basePricePerDay \* days; // No extra charges

}

};

class Truck : public Vehicle {

private:

double weight;

public:

Truck(double price, double w) : Vehicle(price), weight(w) {}

double calculateRentalPrice(int days) override {

double weightCharge = 500 \* weight; // Example charge per ton

return (basePricePerDay + weightCharge) \* days;

}

};

class Motorcycle : public Vehicle {

private:

double mileage;

public:

Motorcycle(double price, double m) : Vehicle(price), mileage(m) {}

double calculateRentalPrice(int days) override {

double mileageFactor = (mileage < 40) ? 1.2 : 1.0; // More charge if low mileage

return basePricePerDay \* mileageFactor \* days;

}

};

int main() {

int rentalDays = 3;

Car car(1000);

Truck truck(1500, 2.5);

Motorcycle bike(500, 35);

cout << "=== Car Rental ===" << endl;

car.showRentalDetails(rentalDays);

cout << "\n=== Truck Rental ===" << endl;

truck.showRentalDetails(rentalDays);

cout << "\n=== Motorcycle Rental ===" << endl;

bike.showRentalDetails(rentalDays);

return 0;

}

**Q5. A banking application needs to handle multiple types of exceptions, such as InsufficientFundsException,**

**InvalidAccountException, and NetworkFailureException. These exceptions can occur simultaneously when processing**

**transactions.**

**ANS:-**

#include <iostream>

#include <string>

#include <stdexcept>

using namespace std;

class BankException : public exception {

public:

virtual const char\* what() const noexcept override {

return "Bank exception occurred.";

}

};

class InsufficientFundsException : public BankException {

public:

const char\* what() const noexcept override {

return "Insufficient funds in the account.";

}

};

class InvalidAccountException : public BankException {

public:

const char\* what() const noexcept override {

return "The account number provided is invalid.";

}

};

class NetworkFailureException : public BankException {

public:

const char\* what() const noexcept override {

return "Network error. Please try again later.";

}

};

void processTransaction(string accountNumber, double balance, double amountToWithdraw, bool networkUp) {

if (!networkUp) {

throw NetworkFailureException();

}

if (accountNumber != "123456") {

throw InvalidAccountException();

}

if (balance < amountToWithdraw) {

throw InsufficientFundsException();

}

cout << "Transaction successful! ₹" << amountToWithdraw << " withdrawn." << endl;

}

int main()

string account = "123456";

double balance = 5000.0;

double withdrawAmount = 6000.0;

bool networkAvailable = true;

try {

processTransaction(account, balance, withdrawAmount, networkAvailable);

}

catch (const InsufficientFundsException& e) {

cout << "[Error] " << e.what() << endl;

}

catch (const InvalidAccountException& e) {

cout << "[Error] " << e.what() << endl;

}

catch (const NetworkFailureException& e) {

cout << "[Error] " << e.what() << endl;

}

catch (const BankException& e) {

cout << "[Error] " << e.what() << endl;

}

catch (...) {

cout << "[Error] Unknown exception occurred." << endl;

}

return 0;

}

**Q.6. // Interface Animal**

**interface Animal {**

**void makeSound();**

**}**

**// Interface Cat**

**interface Cat {**

**void purr();**

**}**

**// Dog class implementing both interfaces**

**class Dog implements Animal, Cat {**

**public void makeSound() {**

**System.out.println("Dog barks!");**

**}**

**public void purr() {**

**System.out.println("Dog cannot purr, but method must be implemented!");**

**}**

**}**

**// Main class**

**public class Main {**

**public static void main(String[] args) {**

**Dog dog = new Dog();**

**dog.makeSound();**

**dog.purr();**

**}**

**}**

**ANS:**

interface Animal {

void makeSound();

}

interface Cat {

void purr();

}

class Dog implements Animal, Cat {

public void makeSound() {

System.out.println("Dog barks!");

}

public void purr() {

System.out.println("Dog cannot purr, but method must be implemented!");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.makeSound();

dog.purr();

}

}

**Q.7. A flight booking system allows customers to enter their age while booking a ticket. If the user enters a negative**

**number, the system should throw an exception. However, missing a required field should cause a different exception.**

**ANS:-**

#include <iostream>

#include <stdexcept>

#include <string>

using namespace std;

int main() {

string input;

cout << "Enter your age: ";

getline(cin, input);

try {

if (input.empty()) {

throw runtime\_error("Age is a required field.");

}

int age = stoi(input); 1

if (age < 0) {

throw invalid\_argument("Age cannot be negative.");

}

cout << "Age entered: " << age << endl;

cout << "Ticket booking successful!" << endl;

}

catch (const invalid\_argument& e) {

cerr << "Invalid input: " << e.what() << endl;

}

catch (const runtime\_error& e) {

cerr << "Missing input: " << e.what() << endl;

}

catch (...) {

cerr << "An unknown error occurred." << endl;

}

return 0;

}

**Q.8. What will happen if an exception occurs and there is no catch block?**

Ans:-

#include <iostream>

#include <stdexcept>

using namespace std;

int main() {

cout << "Program started." << endl;

throw runtime\_error("Something went wrong!");

cout << "This line will not be executed." << endl;

return 0;

}

**Q.9 A software development company is working on an employee attendance system. Each employee has a unique ID**

**and login time. The system initializes employee details when they log in and provides methods to retrieve and update**

**their attendance data.**

**Question: What is the difference between a method and a constructor in OOP? Explain with reference to the**

**employee attendance system.**

**Ans:-**

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

int employeeID;

string loginTime;

string attendanceStatus;

public:

Employee(int id, string time) {

employeeID = id;

loginTime = time;

attendanceStatus = "Present";

cout << "Employee " << employeeID << " logged in at " << loginTime << ", Status: " << attendanceStatus << endl;

}

void updateLoginTime(string newTime) {

loginTime = newTime;

cout << "Login time updated for Employee " << employeeID << ": " << loginTime << endl;

}

void updateAttendance(string status) {

attendanceStatus = status;

cout << "Attendance updated for Employee " << employeeID << ": " << attendanceStatus << endl;

}

string getLoginTime() {

return loginTime;

}

string getAttendanceStatus() {

return attendanceStatus;

}

void printDetails() {

cout << "Employee ID: " << employeeID << endl;

cout << "Login Time: " << loginTime << endl;

cout << "Attendance Status: " << attendanceStatus << endl;

}

};

int main() {

Employee emp1(101, "09:00 AM");

emp1.updateLoginTime("09:15 AM");

emp1.updateAttendance("Absent");

cout << "\nFinal Employee Details:" << endl;

emp1.printDetails();

return 0;

}

**Q.10 A bank application stores customer account details securely. The account balance should not be directly**

**accessible but can be retrieved or updated using specific methods.**

**Question: Explain how getter and setter methods contribute to encapsulation. Provide a code example**

**demonstrating how they can be used in this banking scenario.**

**Ans:-**

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

private:

string accountHolder;

double balance;

public:

BankAccount(string name, double initialBalance) {

accountHolder = name;

if (initialBalance >= 0)

balance = initialBalance;

else {

balance = 0;

cout << "Invalid initial balance. Set to 0." << endl;

}

}

double getBalance() const {

return balance;

}

void deposit(double amount) {

if (amount > 0) {

balance += amount;

cout << "Deposited: $" << amount << endl;

} else {

cout << "Invalid deposit amount!" << endl;

}

}

void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

cout << "Withdrawn: $" << amount << endl;

} else {

cout << "Invalid withdrawal attempt!" << endl;

}

}

void showAccount() const {

cout << "Account Holder: " << accountHolder << endl;

cout << "Current Balance: $" << balance << endl;

}

};

int main() {

BankAccount account("Gaurav Naik", 500.0);

account.showAccount();

account.deposit(150.0);

account.withdraw(100.0);

cout << "Final Balance: $" << account.getBalance() << endl;

return 0;

}

**Q.11. A university wants to model its staff hierarchy. A Professor is a type of Employee, and a Department has multiple**

**Professors.**

**Question: Describe the "is-a" and "has-a" relationships in OOP with reference to this scenario. Provide an example**

**Ans:-**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Employee {

protected:

string name;

int employeeID;

public:

Employee(string n, int id) : name(n), employeeID(id) {}

void displayInfo() const {

cout << "Name: " << name << ", ID: " << employeeID << endl;

}

};

class Professor : public Employee {

private:

string subject;

public:

Professor(string n, int id, string sub) : Employee(n, id), subject(sub) {}

void displayProfessor() const {

displayInfo();

cout << "Subject: " << subject << endl;

}

};

class Department {

private:

string deptName;

vector<Professor> professors;

public:

Department(string name) : deptName(name) {}

void addProfessor(const Professor& prof) {

professors.push\_back(prof);

}

void showDepartment() const {

cout << "Department: " << deptName << endl;

cout << "Professors:\n";

for (const auto& prof : professors) {

prof.displayProfessor();

cout << "-------------------\n";

}

}

};

int main() {

Professor prof1("Dr. Tanay Gosh", 1001, "Physics");

Professor prof2("Prajakta BHand", 1002, "Mathematics");

Department AIML("AIML Department");

AIML.addProfessor(prof1);

AIML.addProfessor(prof2);

AIML.showDepartment();

return 0;

}

**Q. 12) A game development team is using both C++ and Java to design a game engine. They notice that Java requires**

**interface and implements keywords, while C++ does not.**

**Question: Why does C++ not require the interface or implements keyword like Java? Explain with a code example.**

**Ans:-**

**Java code ex:-**

interface GameObject {

void update();

}

class Player implements GameObject {

@Override

public void update() {

System.out.println("Player updated");

}

}

class Enemy implements GameObject {

@Override

public void update() {

System.out.println("Enemy updated");

}

}

public class Main {

public static void main(String[] args) {

// Polymorphism: reference of type GameObject

GameObject playerObj = new Player();

GameObject enemyObj = new Enemy();

playerObj.update();

enemyObj.update();

}

}

**C++ code ex.**

#include <iostream>

using namespace std;

class GameObject {

public:

virtual void update() = 0;

virtual ~GameObject() {}

};

class Player : public GameObject {

public:

void update() override {

cout << "Player updated" << endl;

}

};

class Enemy : public GameObject {

public:

void update() override {

cout << "Enemy updated" << endl;

}

};

int main() {

GameObject\* playerObj = new Player();

GameObject\* enemyObj = new Enemy();

playerObj->update();

enemyObj->update();

delete playerObj;

delete enemyObj;

return 0;

}

**Q. 13) A social media platform is implementing user profiles. Some details, such as username and profile picture, are**

**visible to everyone, while other details, like contact number and email, are hidden and can only be accessed via a**

**request system.**

**Question: What is the difference between Encapsulation and Abstraction in OOP? How would you use these concepts**

**to design the privacy settings in this social media application?**

Ans:-

#include <iostream>

#include <string>

using namespace std;

class UserProfile {

private:

string username;

string profilePicture;

string contactNumber;

string email;

bool contactVisible;

bool emailVisible;

public:

UserProfile(string user, string pic, string contact, string mail)

: username(user), profilePicture(pic), contactNumber(contact), email(mail),

contactVisible(false), emailVisible(false) {}

string getUsername() {

return username;

}

string getProfilePicture() {

return profilePicture;

}

string getContactNumber() {

if (contactVisible) {

return contactNumber;

} else {

return "Request to view contact number.";

}

}

string getEmail() {

if (emailVisible) {

return email;

} else {

return "Request to view email.";

}

}

void setContactVisible(bool visible) {

contactVisible = visible;

}

void setEmailVisible(bool visible) {

emailVisible = visible;

}

void showPublicProfile() {

cout << "Username: " << getUsername() << endl;

cout << "Profile Picture: " << getProfilePicture() << endl;

}

void showFullProfile() {

showPublicProfile();

cout << "Contact Number: " << getContactNumber() << endl;

cout << "Email: " << getEmail() << endl;

}

};

int main() {

UserProfile user("Gaurav\_Naik", "profile\_pic.jpg", "7517574393", "gaurav@gmail.com");

cout << "Public Profile: " << endl;

user.showPublicProfile();

cout << "\nAttempting to show full profile without permission:" << endl;

user.showFullProfile();

user.setContactVisible(true);

user.setEmailVisible(true);

cout << "\nFull Profile after setting privacy permissions:" << endl;

user.showFullProfile();

return 0;

}

**Q.14 A student information system needs to store student records in a file. It should allow reading student details**

**when needed and updating them efficiently.**

**Question: Explain how to read and write files in C++ using fstream. Provide an example of how the student**

**information system could implement this feature.**

**Ans:-**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

struct Student {

string name;

int rollNumber;

double grade;

void display() const {

cout << "Name: " << name << ", Roll Number: " << rollNumber << ", Grade: " << grade << endl;

}

};

// Function to save student details to the file

void saveStudentDetails(const vector<Student>& students) {

ofstream outFile("students.txt", ios::out); // Open file for writing

if (!outFile) {

cerr << "Error opening file for writing!" << endl;

return;

}

// Write each student's details to the file

for (const auto& student : students) {

outFile << student.name << "\n" << student.rollNumber << "\n" << student.grade << "\n";

}

outFile.close(); // Close the file

}

// Function to read student details from the file

vector<Student> readStudentDetails() {

vector<Student> students;

ifstream inFile("students.txt", ios::in); // Open file for reading

if (!inFile) {

cerr << "Error opening file for reading!" << endl;

return students;

}

Student student;

while (getline(inFile, student.name)) { // Read name from file

inFile >> student.rollNumber; // Read roll number

inFile >> student.grade; // Read grade

inFile.ignore(); // Ignore newline after grade

students.push\_back(student); // Add student to vector

}

inFile.close(); // Close the file

return students;

}

// Function to update a student's record by roll number

void updateStudentRecord(int rollNumber, const Student& updatedStudent) {

vector<Student> students = readStudentDetails(); // Read the current student details from file

ofstream outFile("students.txt", ios::out); // Open file for writing (will overwrite the old file)

if (!outFile) {

cerr << "Error opening file for writing!" << endl;

return;

}

for (auto& student : students) {

if (student.rollNumber == rollNumber) {

student = updatedStudent; // Update the student details if roll number matches

}

outFile << student.name << "\n" << student.rollNumber << "\n" << student.grade << "\n"; // Write student details

}

outFile.close(); // Close the file after writing

}

int main() {

// Sample student data

vector<Student> students = {

{"Gaurav", 1, 85.5},

{"Yogita", 2, 90.0},

{"Sachin", 3, 78.5}

};

// Save student details to file

saveStudentDetails(students);

cout << "Student details saved to file." << endl;

// Read and display student details from file

vector<Student> readStudents = readStudentDetails();

cout << "\nStudent details read from file:\n";

for (const auto& student : readStudents) {

student.display();

}

// Create updated student information

Student updatedStudent = {"Bob", 2, 95.0};

// Update student record

updateStudentRecord(2, updatedStudent);

// Display updated student details

cout << "\nUpdated student details:\n";

readStudents = readStudentDetails();

for (const auto& student : readStudents) {

student.display();

}

return 0;

}

**Q.15 Identify and Fix the Error in the Given Code**

**public class ExceptionExample {**

**public static void main(String[] args) {**

**try {**

**int num = 5 / 0;**

**}**

**catch () {**

**System.out.println("Cannot divide by zero");**

**}**

**}**

**}**

**Ans:-**

**public class ExceptionExample {**

**public static void main(String[] args) {**

**try {**

**int num = 5 / 0; // This will cause an ArithmeticException**

**}**

**catch (ArithmeticException e) { // Catch the specific exception**

**System.out.println("Cannot divide by zero");**

**}**

**}**

**}**

**1. A payment gateway processes multiple types of transactions. While executing a transaction, different exceptions**

**may occur, such as InsufficientBalanceException, InvalidCardException, or TransactionTimeoutException. The system**

**must handle all these exceptions to ensure smooth transactions.**

#include <iostream> #include <stdexcept> #include <string>

class InsufficientBalanceException : public std::exception { public:

const char\* what() const noexcept override {

return "Insufficient balance to complete the transaction."; }

};

class InvalidCardException : public std::exception { public:

const char\* what() const noexcept override { return "Invalid card number.";

} };

class TransactionTimeoutException : public std::exception {

public:

const char\* what() const noexcept override { return "Transaction timed out.";

} };

class PaymentGateway { public:

void processTransaction(double balance, const std::string& cardNumber, bool isTimeout) {

if (balance < 100.0) {

throw InsufficientBalanceException(); }

if (cardNumber.empty() || cardNumber.length() != 16) { throw InvalidCardException();

}

if (isTimeout) {

throw TransactionTimeoutException(); }

std::cout << "Transaction processed successfully!" << std::endl; }

};

int main() {

PaymentGateway gateway;

std::cout << "Test 1: Insufficient Balance" << std::endl; try {

gateway.processTransaction(50.0, "1234567812345678", false); } catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << "\n" << std::endl; }

std::cout << "Test 2: Invalid Card" << std::endl; try {

gateway.processTransaction(200.0, "12345", false); } catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << "\n" << std::endl; }

std::cout << "Test 3: Transaction Timeout" << std::endl; try {

gateway.processTransaction(200.0, "1234567812345678", true); } catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << "\n" << std::endl; }

std::cout << "Test 4: Successful Transaction" << std::endl; try {

gateway.processTransaction(200.0, "1234567812345678", false);

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << "\n" << std::endl; }

return 0; }

**2. A hospital management system stores patient records. Each patient has a unique ID, name, and medical history.**

**The system should allow doctors to create and access patient records efficiently.**

**Question: What is the difference between a Class and an Object in OOP? Explain using this hospital management**

**system as an example.**

#include <iostream> #include <vector> using namespace std; class Patient {

public:

int patientID; string name;

string medicalHistory; void displayInfo() {

cout << "Patient ID: " << patientID << endl; cout << "Name: " << name << endl;

cout << "Medical History: " << medicalHistory << endl; cout << "-----------------------------" << endl;

} };

int main() { Patient p1, p2;

p1.patientID = 101; p1.name = "John Doe";

p1.medicalHistory = "Diabetes, High Blood Pressure"; p2.patientID = 102;

p2.name = "Jane Smith"; p2.medicalHistory = "Asthma"; cout << "Patient Records:\n"; p1.displayInfo(); p2.displayInfo();

return 0; }

**3. A company is developing a role-based access control system where different users have different levels of access.**

**Admin can modify and delete data, Manager can view and edit, while Employees can only view certain details. Question: Create a comparison table of access modifiers (private, protected, public, and default) and explain how**

**they can be used in this system to restrict access appropriately.**

#include <iostream> using namespace std; class User { protected:

string name; string role;

public:

User(string uname, string urole) : name(uname), role(urole) {}

virtual void viewData() {

cout << role << " " << name << " is viewing data." << endl; }

virtual void editData() {

cout << role << " " << name << " does not have permission to edit data." << endl;

}

virtual void deleteData() {

cout << role << " " << name << " does not have permission to delete data." << endl;

} };

class Employee : public User { public:

Employee(string uname) : User(uname, "Employee") {} };

class Manager : public User { public:

Manager(string uname) : User(uname, "Manager") {}

void editData() override {

cout << role << " " << name << " is editing data." << endl; }

};

class Admin : public User { private:

void secureAccessLog() {

cout << "Admin is accessing secure logs." << endl; }

public:

Admin(string uname) : User(uname, "Admin") {}

void editData() override {

cout << role << " " << name << " is editing data." << endl; }

void deleteData() override {

cout << role << " " << name << " is deleting data." << endl; secureAccessLog(); // Only Admin can call this

} };

int main() {

Employee emp("Gaurav"); Manager mgr("Sachin"); Admin adm("Krushna");

cout << "\n--- Employee Access ---" << endl; emp.viewData();

emp.editData(); emp.deleteData();

cout << "\n--- Manager Access ---" << endl; mgr.viewData();

mgr.editData();

mgr.deleteData();

cout << "\n--- Admin Access ---" << endl; adm.viewData();

adm.editData(); adm.deleteData();

return 0; }

**4. A robotics startup is using C++ and Java to develop a robotic arm control system. While coding the interfaces for different sensors, they observe that Java requires interface and implements keywords, whereas C++ does not. Question: Why does C++ not require the interface or**

**implements keyword like Java? Explain with a code example.**

#include <iostream> using namespace std; class Sensor {

public:

virtual void readData() = 0; virtual ~Sensor() {}

};

class TemperatureSensor : public Sensor { public:

void readData() override {

cout << "Reading temperature..." << endl; }

};

class PressureSensor : public Sensor { public:

void readData() override {

cout << "Reading pressure..." << endl; }

};

int main() {

Sensor\* sensor1 = new TemperatureSensor(); Sensor\* sensor2 = new PressureSensor();

sensor1->readData(); sensor2->readData(); delete sensor1; delete sensor2;

return 0; }

**5. A school management system has a Student class that inherits from the Person class. The Person class has a**

**method getDetails(), which returns basic information, but the Student class needs to provide additional student-**

**specific details.**

**Question: How do the super and this keywords work in Java? Explain their use in this school management system.**

#include <iostream> #include <string> using namespace std; class Person { protected:

string name;

public:

Person(string name) { this->name = name;

}

virtual void getDetails() {

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

};

class Student : public Person { private:

int rollNo; public:

Student(string name, int rollNo) : Person(name) { this->rollNo = rollNo;

}

void getDetails() override { Person::getDetails();

cout << "Roll No: " << rollNo << endl; }

};

int main() {

Student student("Gaurav", 1066); student.getDetails();

return 0; }

**6. A vehicle rental management system allows customers to rent vehicles. Customers can specify rental duration,**

**vehicle type, or even add extra services like insurance. Question: How would you implement method overloading for a rent\_vehicle() method to support different customer**

**needs? Provide an example.**

#include <iostream> #include <string> using namespace std;

class VehicleRental { public:

// 1. Rent by duration only void rent\_vehicle(int days) {

cout << "Vehicle rented for " << days << " day(s)." << endl; }

// 2. Rent by vehicle type and duration

void rent\_vehicle(string vehicleType, int days) {

cout << vehicleType << " rented for " << days << " day(s)." << endl;

}

// 3. Rent with extra services like insurance

void rent\_vehicle(string vehicleType, int days, bool insurance) {

cout << vehicleType << " rented for " << days << " day(s)." << endl;

if (insurance)

cout << "Insurance included." << endl;

else

cout << "No insurance selected." << endl; }

};

int main() {

VehicleRental rentalSystem;

cout << "--- Basic Rental ---" << endl; rentalSystem.rent\_vehicle(3); // 3 days

cout << "\n--- Vehicle Type and Duration ---" << endl; rentalSystem.rent\_vehicle("Car", 5); // Car for 5 days

cout << "\n--- Vehicle with Insurance ---" << endl;

rentalSystem.rent\_vehicle("Bike", 2, true); // Bike for 2 days with insurance

return 0; }

**7. A banking application processes transactions. If an error occurs, such as insufficient funds or invalid account**

**number, the system should handle it gracefully instead of crashing.**

**Question: Define Exception Handling and Error Handling and explain how they help in building reliable banking**

**applications.**

#include <iostream> #include <stdexcept> using namespace std;

class InsufficientFundsException : public runtime\_error { public:

InsufficientFundsException() : runtime\_error("Error: Insufficient funds for the transaction.") {}

};

class InvalidAccountException : public runtime\_error { public:

InvalidAccountException() : runtime\_error("Error: Invalid account number.") {}

};

class BankAccount { private:

int accountNumber; double balance;

public:

BankAccount(int accNo, double bal) { if (accNo <= 0) {

throw InvalidAccountException(); }

accountNumber = accNo; balance = bal;

}

void withdraw(double amount) { if (amount > balance) {

throw InsufficientFundsException(); }

balance -= amount;

cout << "Withdrawal successful. Remaining balance: $" << balance << endl;

}

void display() {

cout << "Account Number: " << accountNumber << ", Balance: $" << balance << endl;

} };

int main() { try {

BankAccount account(12345, 500.00); account.display(); account.withdraw(700.00); // This will throw

InsufficientFundsException

} catch (const InvalidAccountException& e) { cerr << e.what() << endl;

} catch (const InsufficientFundsException& e) { cerr << e.what() << endl;

} catch (const exception& e) {

cerr << "General error: " << e.what() << endl; }

return 0; }

**8. Identify and Fix the Error in the Given Code class Student {**

**private int age = 20; }**

**public class Main {**

**public static void main(String[] args) { Student s = new Student(); System.out.println(s.age);**

**} }**

#include <iostream> using namespace std;

class Student { private:

public:

int getAge() { return age;

} };

int main() { Student s;

cout << s.getAge() << endl; return 0;

}

**9. A flight booking system allows customers to book tickets. If an invalid seat number is entered, the system should**

**immediately generate an exception. However, some exceptions, like invalid passport numbers, should be declared in**

**the method signature for the calling function to handle.**

**Question: What is the difference between throw and throws in Java? Explain with reference to this flight booking**

**system.**

#include <iostream> #include <stdexcept> using namespace std;

class InvalidSeatException : public exception { public:

const char\* what() const throw() { return "Invalid seat number.";

} };

class InvalidPassportException : public exception { public:

const char\* what() const throw() { return "Invalid passport number.";

} };

class FlightBookingSystem { public:

void bookTicket(string passportNumber, int seatNumber) {

if (passportNumber.length() != 9) { throw InvalidPassportException();

}

if (seatNumber <= 0 || seatNumber > 100) { throw InvalidSeatException();

}

cout << "Ticket booked successfully!" << endl; }

};

int main() { FlightBookingSystem system;

try {

system.bookTicket("12345", 10);

} catch (InvalidPassportException& e) {

cout << "Booking failed: " << e.what() << endl; } catch (InvalidSeatException& e) {

cout << "Booking failed: " << e.what() << endl; }

return 0; }