Date / /
Page No.

Assignment - 2 Oops

#8 Problem Statement:

A local university requires a software to manage student information efficiently. The system should provide functionality to:

- Create and manage student records
- Track student academic information
- Perform basic operations on student data

1) Student Class Design -

Develop a Student class with the following specifications:

- Private data members to store
  - Student Name
  - Roll Number
  - CGPA
  - List of enrolled courses
- Constructors
  - Default Constructor
  - Parameterized constructor to initialize basic student detail
  - Copy Constructor
- Destructor to handle resource cleanup

Date / /
Page No.

→ // University System

```
#include <iostream>
#include <string>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;

// Base class representing a person in the university

class Person {
private:
    string name, id, contact;
    int age;
public:
    // Defaullt Constructor
    Person() : name(" "), age(0), id(" "), contact(" ") {}

    // Parameterized Constructor
    Person(string n, int a, string i, string c) {
        setName(n);
        setAge(a);
        id = i;
        contact = c;
    }

    // Virtual destructor for proper cleanup

    virtual ~Person() { cout << "Person object destroyed : " << name << endl;
```

Date / /
Page No.

```
// Setter functions with validation
void setName(string n) { if (!n.empty()) name = n; }
void setAge(int a) { if (a > 0 && a < 120) age = a; }
void setID(string i) { id = i; }
void setContact(string c) { contact = c; }

// Getter functions
string getName() { return name; }
int getAge() { return age; }
string getID() { return id; }
string getContact() { return contact; }

// Virtual function to display details
virtual void displayDetails() {
    cout << "Name: " << name << ", Age: " << age << In
    ", ID: " << id << ", Contact: " << contact << endl;

// Virtual function for payment calculation
virtual float calculatePayment() { return 0.0; }
};

// Derived class representing a student

class Student : public Person {
private:
    string enrollmentDate, program;
    float gpa;

public:
    // Default Constructor
    Student() : enrollmentDate(""), program(""), gpa(0.0) {
```

Date / /
Page No.

```
// Parameterized Constructor
Student(string n, int a, string i, string c, string e, string p, in
float g) : Person(n, a, i, c) {
        enrollmentDate = e;
        program = p;
        setGPA(g);
}

// Destructor
~Student() { cout << "Student object destroyed: " << getName(); }

void setGPA(float g) { if (g >= 0.0 && g <= 4.0) gpa = g; }
float getGPA() { return gpa; }
// Overriding display function
void displayDetails() override {
    Person::displayDetails();
    cout << "Program: " << program << ", GPA: " << gpa << endl
}
float calculatePayment() override { return 10000.0; }
};

// Derived class representing a Professor

class Professor : public Person {
private:
    string department, specialization, hireDate;
public:
    // Default constructor
    Professor() : department(""), specialization(""), hireDate("")

    // Parameterized Constructor
```

Date / /
Page No.

```
Professor(string n, int a, string i, string c, string d, string s, string h)
Person(n, a, i, c) {

        department = d;
        specialization = s;
        hireDate = h;
}

// Destructor
~Professor() { cout << "Professor object destroyed: " << getName() <<
 getName() << endl; }

// Overriding Display function
void displayDetails() override {
        Person::displayDetails();
        cout << "Department: " << department << ", Specialization: "
        << specialization << endl;
}
float calculatePayment() override { return 45000.0; }
};

// Class representing a Course
class Course {
private:
        string code, title, description;
        int credits;
public:
        // Default Constructor
        Course() : code(""), title(""), description(""), credits(0) {}

        // Parameterized Constructor
```

Date / /
Page No.

```
    Course(string c, string t, string d, int cr){
        code = c; title = t; description = d;
        setCredits(cr);
    }

    // Destructor
    ~Course() {cout << "Course object destroyed: " << title << endl;}

    void setCredits(int cr) { if(cr > 0) credits = cr; }

    // Getter functions for course details
    string getCode() { return code; }
    string getTitle() { return title; }
    string getDescription() { return description; }
    int getCredits() { return getCredits; }
};

// Class representing a Department
class Department {
private:
    string name, location;
    double budget;
public:
    // Default Constructor
    Department() : name(""), location(""), budget(0.0) {}

    // Parameterized Constructor
    Department(string n, string l, double b) {
        name = n; location = l; budget = b;
    }
    // Destructor
    ~Department() {cout << "Department object destroyed: " << name;}
```

```
    // Getter functions for Department details
    string getName() { return Name; }
    string getLocation() { return location; }
    double getBudget() { return budget; }
};

// Class representing a grade book
class GradeBook {
private:
    map<string, float> grades;
public:
    void addGrade(string studentID, float grade) {
        grades[studentID] = grade;
    }

    float calculateAverageGrade() {
        float total = 0;
        for (auto g : grades) total += g.second;
        return grades.empty() ? 0 : total / grades.size();
    }

// Class managing student enrollments
class EnrollmentManager {
private:
    map<string, vector<string>> enrollments;

public:
    // Function to enroll a student in a course
    void enrollStudent(string course, string studentID) {
        enrollments[course].push_back(studentID);
    }
```

Date / /
Page No.

```
// Function to drop a student from a course
void dropStudent(string course, string studentID) {
        auto& list = enrollments[course];
        list.erase(remove(list.begin(), list.end(), 
        studentID), list.end());}
};  int getEnrollmentCount(string course) {
        return enrollments[course].size();
    }
};

void showDetails(Person* p) {
    p->displayDetails();
    cout << "Payment: " << p->calculatePayment() << endl;
}
// Main function demonstrating the system

int main() {
    Student stu1("Raghav", 18, "ST101", "99999", "2024-01-01",
    "IT", 3.5);
    Student stu2("Kamal", 19, "ST102", "88888", "2024-01-01", "PTE", 3.0);
    Professor prof1("Arjun", 40, "PF201", "44444", "EE", "Signals",
    "2012-03-16");
    Professor prof2("Dr. Manoj", 35, "PF202", "77777", "BME", "Structures",
    "2008-06-01");

    // Displaying details
    showDetails(&stu1);
    showDetails(&stu2);
    showDetails(&prof1);
    showDetails(&prof2);
    return 0;
}
```

Date / /
Page No.

2) Student Management Functionality -

Implement methods in the Student class to :

- Add new courses to a student's record
- Update student CGPA
- Display complete student information
- Validate input data (eg. CGPA Range, Course addition)

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

// Base class representing a person in the university
class Person {
protected:
    string name, id, contact;
    int age;
public:
    // Default Constructor
    Person(string n = "", int a = 0, string i = "", string c = "")
    : name(n), age(a), id(i), contact(c) {}
    // Parameterized Constructor
    virtual ~Person() {}

    // Display basic details
    virtual void displayDetails() {
        cout << "Name: " << name << ", Age: " << age << ", ID: "
        << id << ", Contact: " << contact << endl;
    }
```

Date / /
Page No.

2) Student Management Functionality –

Implement methods in the Student class to :

- Add new courses to a student's record
- Update student CGPA
- Display complete student information
- Validate input data (eg. CGPA Range, Course addition)

→
```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

// Base class representing a person in the university
class Person {
protected:
    string name, id, contact;
    int age;
public:
    // Default Constructor
    Person(string n = "", int a = 0, string i = "", string c = "") :
    name(n), age(a), id(i), contact(c) {}
    // Parameterized Constructor
    virtual ~Person() {}

    // Display basic details
    virtual void displayDetails() {
        cout << "Name: " << name << ", Age: " << age << ", ID: "
        << id << ", Contact: " << contact << endl;
    }
```

```
    virtual float calculatePayment() { return 0.0; }
};

// Derived class representing a student
class Student : public Person {
private:
    string enrollmentDate, program;
    float gpa;
    vector<string> courses;
public:
    // Constructor to initialize student details
    Student(string n, int a, string i, string c, string e, string p,
    float g) : Person(n, a, i, c), enrollmentDate(e), program(p),
    gpa(g) {}

    // Function to update GPA with validation
    void setGPA(float g) { if (g >= 0.0 && g <= 4.0) gpa = g; }

    // Function to add a course
    void addCourse(string course) {
        if (find(courses.begin(), courses.end(), course) ==
        courses.end()) {
            courses.push_back(course);
            cout << "Course " << course << " added.\n";
        } else {
            cout << "Course " << course << " already added.\n";
        }
    }

    // Overriding Display Function

    void displayDetails() override {
```

Date / /
Page No.

```
    Person::displayDetails();
    cout << "Program: " << program << ", GPA: " << gpa << endl;
    cout << "Courses: ";
    for (const auto& course : courses) {
        cout << course << " ";
    }
    cout << endl;
}
float calculatePayment() override { return 15000.0; }
};

// Derived class representing a professor
class Professor : public Person {
private:
    string department, specialization, hireDate;
public:
    // Constructor to initialize Professor details
    Professor(string n, int a, string i, string c, string d, string s, string h) :
    Person(n, a, i, c), department(d), specialization(s), hireDate(h) { }

    // Overriding display function
    void displayDetails() override {
        Person::displayDetails();
        cout << "Dept: " << department << ", Spec: " << specialization;
    }
    float calculatePayment() override { return 40000.0; }
};

// Function to display details and payment of a person
void showDetails(Person* p) {
    p->displayDetails();
    cout << "Payment: Rs. " << p->calculatePayment() << endl;
}
```

Date / /
Page No.

```
// Main function demonstrating the system
int main() {
    // Creating student and professor objects
    Student s1("Kanishk", 18, "BUG101", "12345", "2023-08-01", "CSE",
    3.6);

    // Adding courses dynamically
    s1.addCourse("Maths");
    s1.addCourse("Physics");
    s1.addCourse("Maths");

    Professor p1("Dr. Manoj", 35, "APJ01", "77889", "CSE", "ML", in
    "2019-07-01");

    // Displaying details
    showDetails(&s1);
    showDetails(&p1);

    return 0;
}
```

Date / /
Page No.

3) Student Management System Class –

Create a Student Management System class with capabilities to :

- Add new students to the system
- Search for students by roll number
- Display all student records
- Manage a collection of student objects

★ Functional Requirements :–

The system should support the following operations :

1. Create student records with different initialization methods.
2. Add courses to a student's academic record
3. Update academic performance (CGPA)
4. Retrieve and display student information
5. Perform basic error checking and data manipulation

Date / /

Page No.

→
```
#include <iostream>
#include <string>
#include <map>
#include <vector>
using namespace std;

// Exceptional Handling Base Class
class UniversitySystemException {
protected:
    string message;
public:
    UniversitySystemException(string msg) : message(msg){}
    string what() { return message; }
};

// Specific exceptions
class InvalidDataException : public UniversitySystemException {
public:
    InvalidDataException(string msg) : \n
    UniversitySystemException("Invalid Data: " + msg) {}
};

// Student Class

class Student {
private:
    string name, rollNumber, program;
    float cgpa;
    vector<string> courses;
public:
    // Constructor
```

Date / /
Page No.

```
Student
  Student(string n, string r, string p, float c) {
  if (r.empty()) throw InvalidDataException("Roll No can't be empty");
  if (c < 0.0 || c > 4.0) throw InvalidDataException("CGPA must lie b/w 0 & 4");
  name = n;
  rollNumber = r;
  program = p;
  cgpa = c;
}
// Function to add a course
void addCourse(string course) {
  courses.push_back(course);
}

// Function to update CGPA
void updateCGPA(float newCGPA) {
  if(newCGPA < 0.0 || newCGPA > 4.0) throw InvalidDataException("\n
  ("CGPA must lie b/w 0.0 and 4.0");
  cgpa = newCGPA;
}

// Function to display student details
void displayDetails() {
  cout << "Name: " << name << ", Roll No: " << rollNumber << "\n
  "Program: " << program << ", CGPA: " << cgpa << endl;
  cout << "Courses: ";

  for(const auto& course : courses) {
      cout << course << " ";
  }
  cout << endl;
}
```

Date / /
Page No.

```
    // Getter for Roll Number
    string getRollNumber() { return rollNumber; }
};

// Student Management System Class

class StudentManagementSystem {
private:
    map<string, Student> students;
public:
    // Function to add a student
    void addStudent(Student s) {
        students[s.getRollNumber()] = s;
    }

    // Function to search for a student by Roll Number

    void searchStudent(string rollNumber) {
        if (students.find(rollNumber) != students.end()) {
            students[rollNumber].displayDetails();
        } else {
            cout << "Student with Roll No " << rollNumber << In
            "not found.\n";
        }
    }
    // Function to display all students
    void displayAllStudents() {
        for (auto& s : students) {
            s.second.displayDetails();
        }
    }
};
```

```
// Main function
int main() {
    try {
        StudentManagementSystem sms;

        // Adding students
        Student s1("Kanishk", "UG101", "CSE", 3.6);
        s1.addCourse("Maths");
        s1.addCourse("Physics");

        Student s2("Raghav", "PG202", "AI", 3.9);
        s2.addCourse("Machine Learning");

        sms.addStudent(s1);
        sms.addStudent(s2);
        // Searching for a student
        sms.searchStudent("UG101");
        // Displaying all students
        sms.displayAllStudents();
    } catch (UniversitySystemException& e) {
        cout << "[ERROR] " << e.what() << endl;
    }
    return 0;
}
```