#include <iostream>

#include <vector>

using namespace std;

//exception for payment

void Exception(float amount)

{

if (amount <= 0) {

throw logic\_error("PaymentException: Payment amount must be positive.");

}}

//CLASS PERSON

class Person

{ int age;

int id;

string contact;

protected:

string name;

public:

Person() //default constructor

{ cout<<"Person default constructor called"<<endl;

name="unknown";

age=0;

id=0;

contact="0";

}

void Name(string a)

{if(!a.empty()) name=a;

else cout<<"Empty name"<<endl;}

void Age(int b)

{if(b>0 && b<100) age=b;

else cout<<"Invalid age"<<endl;}

Person(string a,int b,int c,string d) //parametrised constructor

{ cout<<"Person parametrised constructor called"<<endl;

Name(a);

Age(b);

id=c;

contact=d;

}

void setterPerson(string a, int b, int c, string d) //setter function

{

Name(a);

Age(b);

id=c;

contact=d;

}

string getname() const {

return name;

}

//display function

virtual void displaydetails() const

{

cout<<"PERSON DETAILS"<<endl;

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

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

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

cout<<"Contact : "<<contact<<endl;

}

virtual float calculatePayment() const

{Exception(0.0);

return 0.0;}

virtual ~Person() //destructor

{cout<<"Person Destructor called"<<endl;}

};

//CLASS PROFESSOR

class Professor:public Person

{ string department;

string specialization;

struct hire\_date{

int day;

int month;

int year;

};

hire\_date date;

protected:

float basesalary=0.0;

public:

Professor() //default constructor

{ cout<<"Professor default constructor called"<<endl;

department="Unknown";

specialization="Unknown";

date.day=1;

date.month=1;

date.year=2000;

}

Professor(string a, int b,int c, string d, string a1, string b1, int c1, int d1, int e) : Person(a,b,c,d) //parametrised constructor

{ cout<<"Professor parametrised constructor called"<<endl;

department=a1;

specialization=b1;

date.day=c1;

date.month=d1;

date.year=e;

}

void setterProfessor(string a, string b, int c, int d, int e) //setter function

{

department=a;

specialization=b;

date.day=c;

date.month=d;

date.year=e;

}

float calculatePayment() const override

{Exception(89000.0);

return 89000.0; }

//display function

void displaydetails() const override

{ cout<<"PROFESSOR DETAILS"<<endl;

cout<<"Department : "<<department<<endl;

cout<<"Specialization : "<<specialization<<endl;

cout << "Hire Date : " << date.day << "/" << date.month << "/" << date.year << endl;

}

~Professor() //destructor

{cout<<"Professor Destructor called"<<endl;}

};

//CLASS COURSE

class Course

{ int code;

string title;

int credits;

string description;

vector<Course\*> courses;

Professor instructor;

public:

//default constructor

Course() :instructor()

{ cout<<"Course default constructor called"<<endl;

code=0;

title="Unknown";

credits=0;

description="None";

}

void Credits(int c)

{if(c>0) credits=c;

else cout<<"Credits can't be negative"<<endl;}

Course(int a, string b, int c, string d,Professor prof) //parametrised constructor

{ cout<<"Course parametrised constructor called"<<endl;

code=a;

title=b;

Credits(c);

description=d;

instructor=prof;

}

void setterCourse(int a, string b, int c, string d,Professor prof) //setter function

{

code=a;

title=b;

Credits(c);

description=d;

instructor=prof;

}

vector<Course\*> getallCourse() const

{

return courses;

}

string getCoursetitle() const

{return title;}

void display() const //display function

{

cout<<"COURSE DETAILS"<<endl;

cout<<"Code : "<<code<<endl;

cout<<"Title : "<<title<<endl;

cout<<"Credits : "<<credits<<endl;

cout<<"Description : "<<description<<endl;

cout << "Instructor: ";

instructor.displaydetails(); // Display instructor details

}

~Course() //destructor

{cout<<"Course Destructor called"<<endl;}

};

//CLASS STUDENT

class Student: public Person

{ struct enrollment\_date{

int day;

int month;

int year;

};

enrollment\_date date;

string program;

float gpa;

vector<Course> courses;

public:

Student() //default constructor

{ cout<<"Student default constructor called"<<endl;

date.day=1;

date.month=1;

date.year=2000;

program="Unknown";

gpa=0.0;

}

void Gpa(float e)

{if(e>0.0 || e<4.0) gpa=e;

else cout<<"Invalid GPA"<<endl;}

Student(string a, int b, int c, string d, int a1, int b1, int c1, string d1, float e) : Person(a, b, c, d) //parametrised constructor

{ cout<<"Student parametrised constructor called"<<endl;

date.day=a1;

date.month=b1;

date.year=c1;

program=d1;

Gpa(e);

}

void setterStudent(int a, int b, int c, string d, float e) //setter function

{

date.day=a;

date.month=b;

date.year=c;

program=d;

Gpa(e);

}

float calculatePayment() const override

{ Exception(30500.0);

return 30500.0;}

void addcourse(const Course cou)

{

courses.push\_back(cou);

}

vector<Course> getallCourses() const

{

return courses;

}

//display function

void displaydetails() const override

{ cout<<"STUDENT DETAILS"<<endl;

cout << "Program : " << program << endl;

cout << "GPA : " << gpa << endl;

cout << "Enrollment Date : " << date.day << "/" << date.month << "/" << date.year << endl;

cout<<"Enrolled courses: "<<courses.size()<<endl;

for(const auto& Course:courses)

{Course.display();

cout<<endl;}

}

~Student() //destructor

{cout<<"Student Destructor called"<<endl;}

};

//test function pointer to call polymorphic functions

void test(const Person\* p) {

p->displaydetails();

cout<<"Payment: "<<p->calculatePayment()<<endl;

}

//CLASS DEPARTMENT

class Department

{ string name;

string location;

int budget;

vector<Professor\*> professors;

public:

Department() //default constructor

{ cout<<"Department default constructor called"<<endl;

name="Unknown";

location="Unknown";

budget=0;

}

Department(string a,string b, int c) //parametrised constructor

{ cout<<"Department parametrised constructor called"<<endl;

name=a;

location=b;

budget=c;

}

void setterDepartment(string a , string b, int c) //setter function

{

name=a;

location=b;

budget=c;

}

void addProfessor(Professor\* prof)

{

professors.push\_back(prof);

}

vector<Professor\*> getallProfessor() const

{

return professors;

}

string getDepartmentname() const

{return name;}

void display() const //display function

{

cout<<"DEPARTMENT DETAILS"<<endl;

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

cout<<"Location : "<<location<<endl;

cout<<"Budget : "<<budget<<endl;

}

~Department() // destructor

{cout<<"Department Destructor called"<<endl;}

};

//CLASS GRADEBOOK

class GradeBook

{ int stuId[50];

float grade[50];

int size;

public:

GradeBook() //default constructor

{size=0; }

void addgrade(int id,float gr){

if(gr<0 || gr>100){

cout<<"Invalid grade"<<endl;

return;}

if(size<50) {

stuId[size]=id;

grade[size]=gr;

size++;

cout<<"Grade added successfully"<<endl;}

else

cout<<"Grades are full"<<endl;

}

float calculateAverageGrade()

{

if(size==0) return 0.0;

int total=0;

for(int i=0;i<size;i++)

{

total+=grade[i];

}

return (float)total/size;

}

int GetHighestGrade()

{

if(size==0) return -1;

int max=grade[0];

for(int i=0;i<size;i++)

{ if(grade[i]>max)

max=grade[i];

}

return max;}

void getFailingStudents()

{

cout<<"The failing students(below 40) are"<<endl;

for(int i=0;i<size;i++)

{ if(grade[i]<40)

cout<<stuId[i]<<endl;

}}

//display function

void display() const

{ cout<<"GRADEBOOK DETAILS"<<endl;

cout<<"No. of grades added: "<<size<<endl;

}

~GradeBook() //destructor

{cout<<"GARDEBOOK DESTRUCTOR CALLED"<<endl;}

};

//CLASS ENROLLMENTMANAGER

class EnrollmentManager

{ int StuId[50];

int coursecode[50];

int size;

public:

EnrollmentManager() //default constructor

{ size=0; }

void enrollStudent(int Id, int code)

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

if (StuId[i] == Id && coursecode[i] == code) {

throw runtime\_error("EnrollmentError: Student is already enrolled in this course.");

}

}

if(size<50)

{ StuId[size]=Id;

coursecode[size]=code;

size++;

cout<<"Student enrolled"<<endl;

}

else

{ cout<<"Enrollment is full"<<endl;}

}

void dropStudent(int Id,int code)

{

bool present=false;

int i,j;

for(i=0;i<size; i++)

{if(coursecode[i]==code && StuId[i]==Id)

{for(j=i;j<size-1;j++)

{StuId[j]=StuId[j+1];

coursecode[j]=coursecode[j+1];}}

size--;

present=true;

cout<<"Student dropped"<<endl; }

if(!present)

{ throw invalid\_argument("GradeException: No student found with the given id and coursecode"); }

}

int getEnrollmentCount(int code)

{

int count=0;

for(int i=0;i<size;i++)

{if(coursecode[i]==code)

count++;}

return count;

}

//display function

void display() const

{ cout<<"ENROLLMENT MANAGER DETAILS"<<endl;

cout<<"No. of students enrolled: "<<size<<endl;

}

~EnrollmentManager() //destructor

{cout<<"ENROLLMENTMANAGER DESTRUCTOR CALLED"<<endl;}

};

//CLASS UNDERGRADUATESTUDENT

class UndergraduateStudent:public Student

{ string minor;

string major;

string expGraddate;

public:

UndergraduateStudent(string a, int b, int c, string d , int a1, int b1, int c1, string d1, float e, string minor, string major, string expGraddate):

Student(a,b,c,d,a1,b1,c1,d1,e),minor(minor), major(major), expGraddate(expGraddate) {} //parametrised constructor

//display function

void displaydetails() const override

{cout<<"UNDERGRADUATE STUDENT DETAILS"<<endl;

cout<<"Minor: "<<minor<<endl;

cout<<"Major: "<<major<<endl;

cout<<"Expected Graduation Date: "<<expGraddate<<endl;

}

float calculatePayment() const override

{ Exception(45000.0);

return 45000.0;}

~UndergraduateStudent() //destructor

{cout<<"UNDERGRADUATESTUDENT DESTRUCTOR CALLED"<<endl; }

};

//CLASS GRADUATESTUDENT

class GraduateStudent:public Student

{ string researchtopic;

Professor advisor;

string Thesistopic;

bool hasResearchAssistantship;

public:

GraduateStudent(string a, int b, int c, string d , int a1, int b1, int c1, string d1, float e,string researchtopic,Professor adv, string Thesistopic ):

Student(a,b,c,d,a1,b1,c1,d1,e),researchtopic(researchtopic), advisor(adv),Thesistopic(Thesistopic), hasResearchAssistantship(false) {} //parametrised constructor

//display function

void displaydetails() const override

{ cout<<"GRADUATE STUDENT DETAILS"<<endl;

cout<<"Name: "<< getname()<<endl;

cout<<"Research topic: "<<researchtopic<<endl;

cout << "Advisor: ";

advisor.displaydetails();

cout<<"Teacher assistance assiged"<<endl; // Display instructor details

cout<<"Thesis topic: "<<Thesistopic<<endl;

cout << "Has Research Assistantship: " << (hasResearchAssistantship ? "Yes" : "No") << endl;

}

float calculatePayment() const override

{Exception(60000.0);

return 60000.0;}

~GraduateStudent() //destructor

{cout<<"GRADUATETUDENT DESTRUCTOR CALLED"<<endl;}

};

//CLASS ASSISTANTPROFESSOR

class AssistantProfessor:public Professor

{ float years;

public:

AssistantProfessor(string a, int b, int c, string d , string a1, string b1, int c1, int d1, int e,float years):

Professor(a,b,c,d,a1,b1,c1,d1,e) //parametrised constructor

{

this->years=years;

basesalary=75000.0;

}

//display function

void displaydetails() const override

{ cout<<"ASSISTANT PROFESSOR DETAILS"<<endl;

cout<<"Years of Probation are "<<years<<endl;

}

float calculatePayment() const override

{ float rankbonus=10000.0;

float expbonus=years\*5000.0;

float p=rankbonus+expbonus+basesalary;

Exception(p);

return p; }

~AssistantProfessor() //destructor

{cout<<"ASSISTANTPROFESSOR DESTRUCTOR CALLED"<<endl;}

};

//CLASS ASSOCIATEPROFESSOR

class AssociateProfessor:public Professor

{

float yearsofexp;

public:

AssociateProfessor(string a, int b, int c, string d , string a1, string b1, int c1, int d1, int e,float yearsofexp):

Professor(a,b,c,d,a1,b1,c1,d1,e) //parametrised constructor

{

this->yearsofexp=yearsofexp;

basesalary=100000.0;

}

//display function

void displaydetails() const override

{ cout<<"ASSOCIATE SPROFESSOR DETAILS"<<endl;

cout<<"Years of experience"<<yearsofexp<<endl;}

float calculatePayment() const override

{ float rankbonus=15000.0;

float expbonus=yearsofexp\*5500.0;

float p=rankbonus+expbonus+basesalary;

Exception(p);

return p;}

~AssociateProfessor() //destructor

{cout<<"ASSOCIATEPROFESSOR DESTRUCTOR CALLED"<<endl;}

};

//CLASS FULLPROFESSOR

class FullProfessor:public Professor

{ float years;

int publicationsrequired;

public:

FullProfessor(string a, int b, int c, string d , string a1, string b1, int c1, int d1, int e,float f,int g):

Professor(a,b,c,d,a1,b1,c1,d1,e) //parametrised constructor

{

years=f;

publicationsrequired=g;

basesalary=150000.0;

}

//display function

void displaydetails() const override

{ cout<<"FULL PROFESSOR DETAILS"<<endl;

cout<<"No of years "<<years<<endl;

cout<<"Number of publications required are "<<publicationsrequired<<endl;}

float calculatePayment() const override

{ float rankbonus=20000.0;

float expbonus=years\*6000.0;

float publishbonus= publicationsrequired\* 3000;

float p=rankbonus+expbonus+publishbonus+basesalary;

Exception(p);

return p;}

~FullProfessor() //destructor

{cout<<"FULLPROFESSOR DESTRUCTOR CALLED"<<endl;}

};

class University

{

vector<Department> departments;

string name;

int count=0;

public:

University(string n) //parametrised constructor

{

name=n;

cout<<"University "<<name<<endl;

}

void addDepartment(const Department dept) {

if (count >= 15) {

throw invalid\_argument("UniversitySystemException: Cannot add more departments. Max limit reached.");

}

departments.push\_back(dept);

count++;

cout << "Department added: " << dept.getDepartmentname() << endl;

}

void deletedept(const string& dept)

{ for (auto it = departments.begin(); it != departments.end(); ++it) {

if ((it)->getDepartmentname()==dept) {

departments.erase(it);

cout << "Department " << dept << " removed.\n";

return;

}

}

cout << "Department " << dept << " not found.\n";

}

void getallProfessor() const {

cout << "\nProfessors across all departments:\n";

for (const auto& dept : departments) {

dept.display(); // assumes your Department class has this

}

}

void getallCourse() const {

cout << "\nCourses across all departments:\n";

for (const auto& dept : departments) {

dept.display(); // assumes your Department class has this

}

}

~University() //destrcutor

{cout<<"University destructor called"<<endl;}

};

//CLASS CLASSROOM

class Classroom

{ int roomno;

public:

Classroom(int roomno) //parametrised constructor

{

this->roomno=roomno; }

int getroomno() const

{return roomno; }

void display() const //display function

{

cout<<"Classroom number is "<<roomno<<endl;

}

~Classroom() //destructor

{

cout<<"CLASSROOM DESTRUCTOR CALLED"<<endl;

}

};

//CLASS SCHEDULE

class Schedule

{ Course c;

Classroom cr;

string timeslot;

public:

Schedule(Course c, Classroom cr, string t): cr(cr), c(c), timeslot(t) {} //parametrised constructor

void display() const //display function

{

cout << "Schedule:"<<endl;

c.display();

cr.display();

cout << "Time Slot: " << timeslot << endl;

}

~Schedule() //destructor

{cout<<"SCHEDULE DESTRUCTOR CALLED"<<endl;}

};

//CLASS UNIVERSTYSYSTEM

class UniversitySystem

{ vector<Student> students;

vector<Professor> professors;

vector<Course> courses;

vector<Department> departments;

EnrollmentManager enrollmentmanager;

GradeBook gradebook;

struct Semester {

string name;

int year;

int number;

};

Semester currentSemester;

public:

void addDepartment(const Department& dept) {

departments.push\_back(dept);

cout << "Department added: " << dept.getDepartmentname() << endl;

}

void addCourse(const Course& course) {

courses.push\_back(course);

cout << "Course added: " << course.getCoursetitle() << endl;

}

void enrollStudent(int studentId, int courseId) {

try {

enrollmentmanager.enrollStudent(studentId, courseId);

cout << "Student enrolled in course." << endl;

} catch (const exception& e) {

cout << "Enrollment Error: " << e.what() << endl;

}

}

void dropStudent(int studentId, int courseId) {

try {

enrollmentmanager.dropStudent(studentId, courseId);

cout << "Student dropped from course." << endl;

} catch (const exception& e) {

cout << "Error while dropping student: " << e.what() << endl;

}

}

void processGrades(int studentId, int courseId, double grade) {

try {

gradebook.addgrade(studentId, grade);

cout << "Grade processed for student." << endl;

} catch (const exception& e) {

cout << "Grade Processing Error: " << e.what() << endl;

}

}

void calculateProfessorPayment(const Professor& prof) {

try {

double payment = prof.calculatePayment();

cout << "Payment for Professor " << prof.getname() << ": ₹" << payment << endl;

} catch (const exception& e) {

cout << "Payment Error: " << e.what() << endl;

}

}

void addSemester() {

cout << "Enter semester name: "<<endl;

cin >> currentSemester.name;

cout << "Enter semester year: "<<endl;

cin >> currentSemester.year;

cout << "Enter semester number: "<<endl;

cin >> currentSemester.number;

cout << "Semester Created: " << currentSemester.name << " " << currentSemester.year << " (Semester " << currentSemester.number << ")" << endl;

}

void generateReport() const {

cout << "University Report:" << endl;

for (const auto& student : students) {

cout << "Student: " <<endl;

student.displaydetails();

cout<< endl; }

for (const auto& prof : professors) {

cout << "Professor: " <<endl;

prof.displaydetails();

cout<< endl; }

for (const auto& dept : departments) {

cout << "Department: " <<endl;

dept.display();

cout<< endl; }

for (const auto& course : courses) {

cout << "Course: " <<endl;

course.display();

cout<< endl;

}

cout<<"END OF REPORT"<<endl;

}

};

//main function

int main()

{ cout<<"The University System can do the following tasks"<<endl;

cout<<"1. Create Person, Student, Professor"<<endl;

cout<<"2. Add a new grades,department,course,semester"<<endl;

cout<<"3. Enroll or drop a student"<<endl;

cout<<"4. Process grades and find the average, highest grade or who is failing"<<endl;

cout<<"5. Calculate payments for all"<<endl;

cout<<"6. Schedule a class for a course"<<endl;

cout<<"7. Make a detailed report for all"<<endl;

//class objects

Person person("Jenna", 28, 1002, "9876543210");

person.displaydetails();

cout<<endl;

Professor professor("Dr. Ritu", 48, 3002, "4444444444", "ECE", "Embedded", 15, 6, 2012);

professor.displaydetails();

cout<<"Payment: "<<professor.calculatePayment()<<endl;

cout<<endl;

Course course(102, "Database Systems", 3, "Relational databases and SQL",professor);

course.display();

cout<<endl;

Student student("Alice", 20, 2001, "1111111111",1, 9, 2022, "B.Tech", 3.6);

student.addcourse(course);

student.displaydetails();

cout<<"Payment: "<<student.calculatePayment()<<endl;

cout<<endl;

Department department("Computer Science", "Building A", 500000);

department.addProfessor(&professor);

department.display();

cout<<endl;

//array of person pointer

Person\* p1[2];

Student s;

Professor p;

s.setterPerson("Naina",20,210,"7852456398");

p.setterPerson("Rahul",26,852,"7854965213");

p1[0]=&s;

p1[0]->displaydetails();

cout<<endl;

p1[1]=&p;

p1[1]->displaydetails();

cout<<endl;

for(int i=0;i<2;i++)

{test(p1[i]);

delete p1[i];}

GradeBook gradebook;

gradebook.addgrade(2001, 85.3);

gradebook.addgrade(1003,39.1);

gradebook.getFailingStudents();

gradebook.display();

cout<<endl;

EnrollmentManager em;

try

{ em.enrollStudent(3001, 101);

em.enrollStudent(3002, 101);

cout << "Enrollment Count for Course 101: " << em.getEnrollmentCount(101) << endl;

em.display();

} catch (exception &exception) {

cout << exception.what() << endl;

}

cout << endl;

UndergraduateStudent ugs("Radhika", 20, 4001, "1122334455", 1, 9, 2023, "B.Sc", 3.5, "Math", "CS", "May 2026");

ugs.displaydetails();

cout << "Payment: " << ugs.calculatePayment() << endl;

cout<<endl;

GraduateStudent gs("Rashi", 24, 2001, "1234567890", 1, 9, 2023, "AI", 8.5, "Neural Networks", professor, "Thesis on Reinforcement Learning");

gs.displaydetails();

AssistantProfessor ap("Neha", 35, 1022, "Female", "P002", "Physics", 5, 3, 2, 2.5);

ap.displaydetails();

cout << "Payment: " << ap.calculatePayment() << endl;

cout<<endl;

AssociateProfessor asp("Rohit", 40, 1023, "Male", "P003", "Chemistry", 6, 4, 3, 7.0);

asp.displaydetails();

cout << "Payment: " << asp.calculatePayment() << endl;

cout<<endl;

FullProfessor fp("Sneha", 50, 1024, "Female", "P004", "Mathematics", 9, 6, 4, 15.0, 20);

fp.displaydetails();

cout << "Payment: " << fp.calculatePayment() << endl;

cout<<endl;

Classroom room(101);

room.display();

cout<<endl;

Schedule schedule1(course, room, "Mon 9:00 AM - 11:00 AM");

schedule1.display();

cout<<endl;

University university("NCU");

university.getallCourse();

university.getallProfessor();

UniversitySystem unis;

unis.addDepartment(department);

unis.addCourse(course);

unis.enrollStudent(2001, 101);

unis.processGrades(2001, 101, 88.5);

unis.calculateProfessorPayment(professor);

unis.addSemester();

unis.generateReport();

return 0;

}