|  | **National University of Computer and Emerging Sciences (Lahore)** | | | |
| --- | --- | --- | --- | --- |
| **Course:** | **OOP** | **Course code:** | **CS217** |
| **Section:** | **BSCS-2B** | **Semester:** | **Spring 2024** |
| **Duration:** | **40 minutes** | **TotalMarks:** | **10** |
| **Date:** |  | **ID:** | **A** |
| **Name:** |  | **Roll no:** |  |

**Question 1:**

**NOTE:** **Read the entire question first before attempting.**

A construction company requires a paycheque management system. It employs different types of employees who are all paid differently as described below. All types of employees must have a function that calculates their salaries but an employee can only be one of three types; manager, engineer or salesperson. All employees “must” belong to at least one of these categories. An unclassified employee “cannot be paid a salary”.

1. All Employees have the following attributes common:
   * name (string): The name of the employee.
   * id (int): The unique ID of the employee.
   * baseSalary (double):
2. Implement three derived classes: Manager, Engineer, and Salesperson, each inheriting from the Employee class with unique attributes:
   * For **Manager:** department (string), bonus (double), calculateSalary() method. They are paid a bonus in addition to their base salary.
   * For **Engineer:** rate (double), numProjects (int), calculateSalary() method. They are paid the product of their rate and no. of projects in addition to their base salary.
   * For **Salesperson:** salesAchieved (double), commissionRate (double), calculateSalary() method. They are paid a commission on each sale made in addition to base salary.
3. Implement default and parameterised constructors, destructors and a calculateSalary() method in each derived class to calculate the salary of the respective employee type based on the provided attributes.
4. Give output of the main given on the next page.

| **int main() {**  **// Create employee objects**  **vector<Employee\*> employees;**  **employees.push\_back(new Manager("Razan Usman", 100, 1000, "CS", 2000.0));**  **employees.push\_back(new Engineer("Armaghan Atiq", 420, 1000, 10.0, 5));**  **employees.push\_back(new Salesperson("Abdullah Ijaz", 666, 1000, 10, 10));**  **// Calculate and display salaries**  **for (Employee\* e : employees) {**  **cout << "Name: " << e->name << endl;**  **cout << "ID: " << e->id << endl;**  **cout << "Base Salary: $" << e->baseSalary << endl;**  **cout << "Total Salary: $" << e->calculateSalary() <<endl;**  **cout << endl;**  **}**  **// Free memory**  **for (Employee\* e : employees) {**  **delete employee;**  **}**  **return 0;**  **}** |
| --- |
| **Output:**  **Name: Razan Usman**  **ID: 100**  **Base Salary: $1000**  **Total Salary: $3000**  **Name: Armaghan Atiq**  **ID: 420**  **Base Salary: $1000**  **Total Salary: $1050**  **Name: Abdullah Ijaz**  **ID: 666**  **Base Salary: $1000**  **Total Salary: $1010** |

**Solution Code:**

class Employee {

public:

string name;

int id;

double baseSalary;

Employee(string name, int id, double baseSalary) : name(name), id(id), baseSalary(baseSalary) {}

virtual ~Employee() {}

virtual double calculateSalary() = 0; // Pure virtual function

};

class Manager : public Employee {

public:

string department;

double bonus;

Manager(string name, int id, double baseSalary, string department, double bonus)

: Employee(name, id, baseSalary), department(department), bonus(bonus) {}

double calculateSalary() override {

return baseSalary + bonus;

}

};

class Engineer : public Employee {

public:

double rate;

int numProjects;

Engineer(string name, int id, double baseSalary, double rate, int numProjects)

: Employee(name, id, baseSalary), rate(rate), numProjects(numProjects) {}

double calculateSalary() override {

return baseSalary + (rate \* numProjects);

}

};

class Salesperson : public Employee {

public:

double salesAchieved;

double commissionRate;

Salesperson(string name, int id, double baseSalary, double salesAchieved, double commissionRate)

: Employee(name, id, baseSalary), salesAchieved(salesAchieved), commissionRate(commissionRate) {}

double calculateSalary() override {

return baseSalary + (salesAchieved \* commissionRate);

}

};