Employee Payroll System - Documentation

1. Overview

The **Employee Payroll System** is a C# console application designed to store employee details, calculate salaries, and manage payroll operations efficiently. This system follows **Object-Oriented Programming (OOP) principles** and includes features such as:

- Adding and displaying employee details
- Calculating individual salaries
- Managing payroll data
- Saving and retrieving employee information using file storage

2. Object-Oriented Structure

This system is built using OOP principles to ensure modularity, scalability, and maintainability.

2.1 BaseEmployee (Abstract Class)

```
abstract class BaseEmployee
{
    public string Name { get; set; }
    public int ID { get; set; }
    public string Role { get; set; }
    public double BasicPay { get; set; }
    public double Allowances { get; set; }
    public double Deductions { get; set; }

    public BaseEmployee(int id, string name, string role, double
basicPay, double allowances, double deductions)
    {
        ID = id;
        Name = name;
        Role = role;
        BasicPay = basicPay;
        Allowances = allowances;
        Deductions = deductions;
    }

    Public double CalculateSalary();
}
```

- **Abstract Class:** Provides common properties and ensures consistency across all employee types.
- **Method Overriding:** Forces derived classes to implement their own salary calculation logic.

2.2 Derived Classes (Manager, Developer, Intern)

```
class Manager : BaseEmployee
{
    public Manager(int id, string name, double basicPay, double
allowances, double deductions)
        : base(id, name, "Manager", basicPay, allowances, deductions)
{}

    public override double CalculateSalary()
    {
        return BasicPay + Allowances - Deductions;
    }
}
```

- **Inheritance:** Reduces code duplication by reusing `BaseEmployee` properties and methods.
- **Role-Specific Behavior:** Allows customization of salary calculations for different job roles.

3. Core Functionalities

3.1 Adding an Employee

```
void AddEmployee()
{
    Console.Write("Enter Employee ID: ");
    int id = int.Parse(Console.ReadLine());

    Console.Write("Enter Name: ");
    string name = Console.ReadLine();

    Console.Write("Enter Role (Manager/Developer/Intern): ");
    string role = Console.ReadLine();

    Console.Write("Enter Basic Pay: ");
    double basicPay = double.Parse(Console.ReadLine());

    Console.Write("Enter Allowances: ");
    double allowances = double.Parse(Console.ReadLine());

    Console.Write("Enter Deductions: ");
    double deductions = double.Parse(Console.ReadLine());

    BaseEmployee employee = role.ToLower() == "manager" ? new
Manager(id, name, basicPay, allowances, deductions) :
                            role.ToLower() == "developer" ? new
Developer(id, name, basicPay, allowances, deductions) :
                            new Intern(id, name, basicPay, allowances,
deductions);

    employees.Add(employee);
    Console.WriteLine("Employee Added Successfully!");
}
```

- **Takes user input dynamically**
- **Creates an employee object based on role**
- **Stores employee details for future reference**

3.2 Displaying All Employees

```
void DisplayEmployees()
{
    Console.WriteLine("Employee Details:");
    foreach (var emp in employees)
    {
        Console.WriteLine($"ID: {emp.ID}, Name: {emp.Name}, Role:
{emp.Role}, Salary: {emp.CalculateSalary()}");
    }
}
```

- **Loops through the employee list** to display details
- **Uses polymorphism** to call `CalculateSalary()` from `BaseEmployee`

3.3 Calculating Total Payroll

```
void CalculateTotalPayroll()
{
    double totalPayroll = employees.Sum(emp =>
emp.CalculateSalary());
    Console.WriteLine($"Total Payroll: {totalPayroll}");
}
```

- **Summarizes all salaries** into a single value
- **Ensures accurate payroll calculation**

4. Data Persistence

4.1 Saving Employee Data to File

```
void SaveEmployeesToFile()
{
    using (StreamWriter writer = new StreamWriter("employees.txt"))
    {
        foreach (var emp in employees)
        {

writer.WriteLine($"{emp.ID},{emp.Name},{emp.Role},{emp.BasicPay},{emp
.Allowances},{emp.Deductions}");
        }
    }
    Console.WriteLine("Employee Data Saved Successfully!");
}
```

- **Persists employee data** even after closing the application
- **Uses `StreamWriter` for efficient file handling**

4.2 Loading Employee Data from File

```
void LoadEmployeesFromFile()
{
    if (File.Exists("employees.txt"))
    {
```

```
        using (StreamReader reader = new
StreamReader("employees.txt"))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                string[] data = line.Split(',');
                int id = int.Parse(data[0]);
                string name = data[1];
                string role = data[2];
                double basicPay = double.Parse(data[3]);
                double allowances = double.Parse(data[4]);
                double deductions = double.Parse(data[5]);

                BaseEmployee employee = role == "Manager" ? new
Manager(id, name, basicPay, allowances, deductions) :
                                        role == "Developer" ? new
Developer(id, name, basicPay, allowances, deductions) :
                                        new Intern(id, name, basicPay,
allowances, deductions);

                employees.Add(employee);
            }
        }
        Console.WriteLine("Employee Data Loaded Successfully!");
    }
}
```

- **Restores data from the file** when the app starts
- **Ensures persistence** even after program restarts

5. User Interface (Main Menu)

```
void MainMenu()
{
    while (true)
    {
        Console.WriteLine("\nEmployee Payroll System");
        Console.WriteLine("1. Add Employee");
        Console.WriteLine("2. Display Employees");
        Console.WriteLine("3. Calculate Total Payroll");
        Console.WriteLine("4. Save Data");
        Console.WriteLine("5. Load Data");
        Console.WriteLine("6. Exit");
        Console.Write("Choose an option: ");

        int choice = int.Parse(Console.ReadLine());
        switch (choice)
        {
            case 1: AddEmployee(); break;
            case 2: DisplayEmployees(); break;
            case 3: CalculateTotalPayroll(); break;
            case 4: SaveEmployeesToFile(); break;
            case 5: LoadEmployeesFromFile(); break;
            case 6: return;
            default: Console.WriteLine("Invalid option!"); break;
        }
    }
}
```

Why?

- **Provides a structured user interface**
- **Uses a loop to keep the application running**
- **Handles user input efficiently**

6. Conclusion

This system efficiently manages employee payroll with:

- **OOP principles (abstraction, inheritance, polymorphism)**
- **Method Overriding for flexibility**
- **File Handling for data persistence**

Future enhancements can include **database integration** and **graphical UI support**.