```
import java.util.Scanner;

class Employee {
    String Emp_name;
    int Emp_id;
    String Address;
    String Mail_id;
    String Mobile_no;
    double BasicPay;

    public Employee(String Emp_name, int Emp_id, String Address,
String Mail_id, String Mobile_no, double BasicPay) {
        this.Emp_name = Emp_name;
        this.Emp_id = Emp_id;
        this.Address = Address;
        this.Mail_id = Mail_id;
        this.Mobile_no = Mobile_no;
        this.BasicPay = BasicPay;
    }

    public void displayEmployeeDetails() {
        System.out.println("Employee Name: " + Emp_name);
        System.out.println("Employee ID: " + Emp_id);
        System.out.println("Address: " + Address);
        System.out.println("Mail ID: " + Mail_id);
        System.out.println("Mobile No: " + Mobile_no);
        System.out.println("Basic Pay: " + BasicPay);

    }

    public double calculateDA() {
        return 0.97 * BasicPay;
    }

    public double calculateHRA() {
        return 0.10 * BasicPay;
    }

    public double calculatePF() {
        return 0.12 * BasicPay;
    }

    public double calculateStaffClubFund() {
        return 0.001 * BasicPay;
    }


    public double calculateGrossSalary() {
```

```
        return BasicPay + calculateDA() + calculateHRA();
    }

    public double calculateNetSalary() {
        return calculateGrossSalary() - calculatePF() -
calculateStaffClubFund();
    }

    public void generatePaySlip() {
        displayEmployeeDetails();
        System.out.println("DA: " + calculateDA());
        System.out.println("HRA: " + calculateHRA());
        System.out.println("PF: " + calculatePF());
        System.out.println("Staff Club Fund: " +
calculateStaffClubFund());
        System.out.println("Gross Salary: " + calculateGrossSalary());
        System.out.println("Net Salary: " + calculateNetSalary());
        System.out.println("-----------------------------------");

    }
}

class Programmer extends Employee {
    public Programmer(String Emp_name, int Emp_id, String Address,
String Mail_id, String Mobile_no, double BasicPay) {
        super(Emp_name, Emp_id, Address, Mail_id, Mobile_no,
BasicPay);
    }
}

class AssistantProfessor extends Employee {
    public AssistantProfessor(String Emp_name, int Emp_id, String
Address, String Mail_id, String Mobile_no, double BasicPay) {
        super(Emp_name, Emp_id, Address, Mail_id, Mobile_no,
BasicPay);
    }
}

class AssociateProfessor extends Employee {
    public AssociateProfessor(String Emp_name, int Emp_id, String
Address, String Mail_id, String Mobile_no, double BasicPay) {
        super(Emp_name, Emp_id, Address, Mail_id, Mobile_no,
BasicPay);
    }
}

class Professor extends Employee {
    public Professor(String Emp_name, int Emp_id, String Address,
```

```
String Mail_id, String Mobile_no, double BasicPay) {
        super(Emp_name, Emp_id, Address, Mail_id, Mobile_no,
BasicPay);
    }
}

public class EmployeePayrollSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter Employee details:");
        System.out.print("Name: ");
        String name = scanner.nextLine();
        System.out.print("ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();// Consume newline
        System.out.print("Address: ");
        String address = scanner.nextLine();
        System.out.print("Mail ID: ");
        String mailId = scanner.nextLine();
        System.out.print("Mobile No: ");
        String mobileNo = scanner.nextLine();
        System.out.print("Basic Pay: ");
        double basicPay = scanner.nextDouble();
        scanner.nextLine(); // Consume newline

        System.out.println("Enter Employee Designation (Programmer,
AssistantProfessor, AssociateProfessor, Professor): ");
        String designation = scanner.nextLine();

        Employee employee = null;

        switch (designation.toLowerCase()) {
            case "programmer":
                employee = new Programmer(name, id, address, mailId,
mobileNo, basicPay);
                break;
            case "assistantprofessor":
                employee = new AssistantProfessor(name, id, address,
mailId, mobileNo, basicPay);
                break;
            case "associateprofessor":
                employee = new AssociateProfessor(name, id, address,
mailId, mobileNo, basicPay);
                break;
            case "professor":
                employee = new Professor(name, id, address, mailId,
mobileNo, basicPay);
```

```
                break;
            default:
                System.out.println("Invalid designation.");
                return;
        }

        if (employee != null) {
            employee.generatePaySlip();
        }

        scanner.close();
    }
}
```

Key improvements and explanations:

- **Inheritance:** The Programmer, AssistantProfessor, AssociateProfessor, and Professor classes correctly inherit from the Employee class using extends.
- **Constructor Chaining:** The constructors of the derived classes use super() to call the Employee class constructor, initializing the common employee attributes. This is crucial for proper object initialization.
- **Salary Calculations:** The calculateDA(), calculateHRA(), calculatePF(), calculateStaffClubFund(), calculateGrossSalary(), and calculateNetSalary() methods are implemented in the Employee class. This promotes code reusability, as all employee types use the same calculation logic.
- **Pay Slip Generation:** The generatePaySlip() method is in the Employee class and is responsible for displaying all the details in a formatted way, making it cleaner.
- **User Input:** The code now takes user input for employee details and designation using a Scanner. It also handles the newline character properly after reading numbers.
- **Designation Handling:** A switch statement is used to create the appropriate employee object based on the entered designation. This is much more efficient and readable than a long chain of if-else if statements.
- **Error Handling:** A basic check is included to handle invalid designations.
- **Resource Management:** The Scanner is closed using scanner.close() to prevent resource leaks.
- **Clearer Output:** The output is formatted to be more readable.

This improved version provides a more robust and well-structured solution for the employee payroll system. It demonstrates good object-oriented principles like inheritance, polymorphism (through the generatePaySlip() method which can be called on any type of employee), and code reusability.