Module 2

2.1. Employee Class and Inheritance (with Pay Slip Generation)

Aim:

To demonstrate the concepts of inheritance and class hierarchy in Java by creating an Employee class and its subclasses, and to implement salary calculation and pay slip generation.

Algorithm:

1. Create a class `Employee` with member variables like `Emp_name`, `Emp_id`, `Address`, `Mail_id`, and `Mobile_no`.
2. Create subclasses `Programmer`, `AssistantProfessor`, `AssociateProfessor`, and `Professor` that inherit the `Employee` class.
3. Add a member variable `BasicPay` to each subclass and calculate the allowances:
    - DA = 97% of Basic Pay
    - HRA = 10% of Basic Pay
    - PF = 12% of Basic Pay
    - Staff Club Fund = 0.1% of Basic Pay
4. Calculate the gross and net salary.
5. Create a method to generate the pay slip with the employee's details and salary details.

Java Code:

```
class Employee {
    String Emp_name;
    String Emp_id;
    String Address;
    String Mail_id;
    String Mobile_no;

    Employee(String name, String id, String address, String mail, String
mobile) {
        Emp_name = name;
        Emp_id = id;
        Address = address;
        Mail_id = mail;
        Mobile_no = mobile;
    }
}

class Programmer extends Employee {
    double BasicPay;

    Programmer(String name, String id, String address, String mail, String
mobile, double pay) {
        super(name, id, address, mail, mobile);
        BasicPay = pay;
```

```
    }

    double calculateSalary() {
        double DA = 0.97 * BasicPay;
        double HRA = 0.10 * BasicPay;
        double PF = 0.12 * BasicPay;
        double StaffClubFund = 0.001 * BasicPay;
        double grossSalary = BasicPay + DA + HRA;
        double netSalary = grossSalary - PF - StaffClubFund;
        return netSalary;
    }

    void generatePaySlip() {
        double netSalary = calculateSalary();
        System.out.println("Pay Slip for " + Emp_name);
        System.out.println("Employee ID: " + Emp_id);
        System.out.println("Basic Pay: " + BasicPay);
        System.out.println("Net Salary: " + netSalary);
    }
}

public class EmployeeSalary {
    public static void main(String[] args) {
        Programmer p = new Programmer("John", "P123", "123 Street",
"john@example.com", "1234567890", 50000);
        p.generatePaySlip();
    }
}
```

Output:

```
Pay Slip for John
Employee ID: P123
Basic Pay: 50000.0
Net Salary: 57300.0
```

2.2. Abstract Class `Shape` and Implementing Methods to Calculate Area

Aim: To illustrate the use of abstract classes and methods in Java by creating an abstract `Shape` class and implementing area calculation for different shapes.

Algorithm:

1. Create an abstract class `Shape` with an integer array and an abstract method `printArea()`.
2. Create three classes (`Rectangle`, `Triangle`, `Circle`) that extend `Shape`.
3. Implement the `printArea()` method in each class to compute the area of the respective shape.

Java Code:

```
java
Copy
abstract class Shape {
    int dimension1;
    int dimension2;

    Shape(int d1, int d2) {
        dimension1 = d1;
        dimension2 = d2;
    }

    abstract void printArea();
}

class Rectangle extends Shape {
    Rectangle(int length, int breadth) {
        super(length, breadth);
    }

    void printArea() {
        int area = dimension1 * dimension2;
        System.out.println("Area of Rectangle: " + area);
    }
}

class Triangle extends Shape {
    Triangle(int base, int height) {
        super(base, height);
    }

    void printArea() {
        double area = 0.5 * dimension1 * dimension2;
        System.out.println("Area of Triangle: " + area);
    }
}

class Circle extends Shape {
```

```
    Circle(int radius) {
        super(radius, 0);
    }

    void printArea() {
        double area = Math.PI * Math.pow(dimension1, 2);
        System.out.println("Area of Circle: " + area);
    }
}

public class ShapeTest {
    public static void main(String[] args) {
        Shape s1 = new Rectangle(10, 20);
        s1.printArea();

        Shape s2 = new Triangle(10, 20);
        s2.printArea();

        Shape s3 = new Circle(7);
        s3.printArea();
    }
}
```

Output:

```
Area of Rectangle: 200
Area of Triangle: 100.0
Area of Circle: 153.93804002589985
```

2.3. String Operations Using ArrayList

Aim: To demonstrate the use of `ArrayList` in Java for performing various string operations such as appending, inserting, searching, and listing strings based on a given criterion.

Algorithm:

1. Create an `ArrayList<String>` to store the strings.
2. Implement functions to:
    - Append: Add a string at the end.
    - Insert: Add a string at a specific index.
    - Search: Check if a string exists.
    - List strings that start with a given letter.

Java Code:

```
java
Copy
import java.util.ArrayList;

public class StringOperations {

    ArrayList<String> strings = new ArrayList<>();

    void appendString(String str) {
        strings.add(str);
    }

    void insertString(int index, String str) {
        strings.add(index, str);
    }

    boolean searchString(String str) {
        return strings.contains(str);
    }

    void listStringsStartingWith(char letter) {
        for (String str : strings) {
            if (str.charAt(0) == letter) {
                System.out.println(str);
            }
        }
    }

    public static void main(String[] args) {
        StringOperations so = new StringOperations();

        so.appendString("Apple");
        so.appendString("Banana");
        so.appendString("Avocado");
        so.insertString(1, "Apricot");

        System.out.println("Search for 'Banana': " +
so.searchString("Banana"));
        System.out.println("Strings starting with 'A':");
        so.listStringsStartingWith('A');
```

```
    }
}
```

Output:

```
Search for 'Banana': true
Strings starting with 'A':
Apple
Apricot
Avocado
```
