Problem 1: Abstract Shape

Description: Create an abstract class Shape to represent different geometric shapes. Each shape should be able to calculate its area.

Requirements:

1. Create an abstract class Shape with an abstract method calculateArea().
2. Create concrete classes Circle, Rectangle, and Triangle that extend Shape and implement the calculateArea() method.
3. Add attributes to each concrete class: Circle should have radius, Rectangle should have length and width, and Triangle should have base and height.
4. Implement a main method to create instances of each shape and display their areas.

Problem 2: Abstract Employee

Description: Create an abstract class Employee to represent different types of employees. Each employee should have a method to calculate their salary.

Requirements:

1. Create an abstract class Employee with an abstract method calculateSalary().
2. Create concrete classes FullTimeEmployee, PartTimeEmployee, and Contractor that extend Employee and implement the calculateSalary() method.
3. Add attributes to each concrete class: FullTimeEmployee should have monthlySalary, PartTimeEmployee should have hourlyRate and hoursWorked, and Contractor should have fixedAmount.
4. Implement a main method to create instances of each employee type and display their salaries.

Problem 3: Abstract Appliance

Description: Create an abstract class Appliance to represent different types of home appliances. Each appliance should have a method to turn on and turn off.

Requirements:

1. Create an abstract class Appliance with abstract methods turnOn() and turnOff().
2. Create concrete classes WashingMachine, Refrigerator, and Microwave that extend Appliance and implement the turnOn() and turnOff() methods.
3. Add attributes to each concrete class: WashingMachine should have loadCapacity, Refrigerator should have volume, and Microwave should have power.
4. Implement a main method to create instances of each appliance and demonstrate turning them on and off.

Problem 4: Abstract Account

Description: Create an abstract class `Account` to represent different types of bank accounts. Each account should have a method to calculate the interest.

Requirements:

1. Create an abstract class `Account` with an abstract method `calculateInterest()`.
2. Create concrete classes `SavingsAccount`, `CurrentAccount`, and `FixedDepositAccount` that extend `Account` and implement the `calculateInterest()` method.
3. Add attributes to each concrete class: `SavingsAccount` should have `balance` and `interestRate`, `CurrentAccount` should have `balance` and `overdraftLimit`, and `FixedDepositAccount` should have `principalAmount` and `tenure`.
4. Implement a `main` method to create instances of each account type and display their calculated interest.

Problem 5: Abstract Vehicle

Description: Create an abstract class `Vehicle` to represent different types of vehicles. Each vehicle should have methods to start and stop.

Requirements:

1. Create an abstract class `Vehicle` with abstract methods `start()` and `stop()`.
2. Create concrete classes `Car`, `Bike`, and `Truck` that extend `Vehicle` and implement the `start()` and `stop()` methods.
3. Add attributes to each concrete class: `Car` should have `numberOfDoors`, `Bike` should have `engineCapacity`, and `Truck` should have `loadCapacity`.
4. Implement a `main` method to create instances of each vehicle type and demonstrate starting and stopping them.