Method Overriding Problem Statements

Problem 1: Animal Sound Simulator

Scenario: You are tasked with creating an application for simulating animal sounds. Each animal has a different sound. Use method overriding to simulate the sounds of different animals.

Classes:

- `Animal`: Base class with a virtual `makeSound()` method.
- `Dog` and `Cat`: Derived classes that override `makeSound()` to produce dog and cat sounds.

Sample Input:

```
Dog dog;
Cat cat;

dog.makeSound();  // Dog sound
cat.makeSound();  // Cat sound
```

Sample Output:

```
Woof!
Meow!
```

Problem 2: Vehicle Speed Calculation

Scenario: Create an application to calculate the speed of different types of vehicles. Each vehicle has its method for calculating speed, but they all inherit from a `Vehicle` base class.

Classes:

- `Vehicle`: Base class with a virtual `calculateSpeed()` method.
- `Car` and `Bike`: Derived classes that override `calculateSpeed()` based on their unique speed calculation methods.

Sample Input:

```
Car car(150, 3); // 150 km/h, 3 hours
Bike bike(50, 2); // 50 km/h, 2 hours

car.calculateSpeed();  // Speed calculation for car
bike.calculateSpeed(); // Speed calculation for bike
```

Sample Output:

```
Car Speed: 450 km
Bike Speed: 100 km
```

Problem 3: Bank Account Interest

Scenario: Design a system for calculating the interest on a bank account. There are different types of accounts, each with its interest calculation method.

Classes:

- `BankAccount`: Base class with a virtual `calculateInterest()` method.
- `SavingsAccount` and `CurrentAccount`: Derived classes that override `calculateInterest()` with their respective formulas.

Sample Input:

```
SavingsAccount savings(10000);  // Principal = 10,000
CurrentAccount current(10000);  // Principal = 10,000

savings.calculateInterest();  // Savings account interest
current.calculateInterest();  // Current account interest
```

Sample Output:

```
Savings Account Interest: 500
Current Account Interest: 250
```

Problem 4: Employee Salary Calculation

Scenario: Design a system where different types of employees calculate their salary based on different rules. Use method overriding to implement custom salary calculation methods.

Classes:

- `Employee`: Base class with a virtual `calculateSalary()` method.
- `Manager` and `Developer`: Derived classes that override `calculateSalary()` with their own salary calculation logic.

Sample Input:

```
Manager manager(5000, 2000);  // Base salary = 5000, Bonus = 2000
Developer developer(4000, 1000); // Base salary = 4000, Bonus = 1000

manager.calculateSalary();  // Salary for Manager
developer.calculateSalary(); // Salary for Developer
```

Sample Output:

```
Manager Salary: 7000
Developer Salary: 5000
```

Problem 5: Shape Area Calculation

Scenario: You need to calculate the area of different shapes. Use method overriding to provide specific formulas for each shape.

Classes:

- `Shape`: Base class with a virtual `calculateArea()` method.
- `Circle` and `Rectangle`: Derived classes that override `calculateArea()`.

Sample Input:

```
Circle circle(5);      // Radius = 5
Rectangle rectangle(4, 6); // Length = 4, Width = 6

circle.calculateArea();    // Area for Circle
rectangle.calculateArea(); // Area for Rectangle
```

Sample Output:

```
Circle Area: 78.5398
Rectangle Area: 24
```

Problem 6: Employee Leave Tracker

Scenario: You need to manage employee leave requests in a company. Different employee types (e.g., regular employees, managers) have different leave policies. Use method overriding to handle leave calculation.

Classes:

- `Employee`: Base class with a virtual `calculateLeave()` method.
- `Manager` and `RegularEmployee`: Derived classes that override `calculateLeave()`.

Sample Input:

```
Manager manager(20);  // Manager has 20 days leave
RegularEmployee employee(15);  // Regular employee has 15 days leave

manager.calculateLeave();  // Leave for Manager
employee.calculateLeave(); // Leave for Regular Employee
```

Sample Output:

```
Manager Leave: 20 days
Regular Employee Leave: 15 days
```

Problem 7: Ticket Pricing System

Scenario: You are designing a ticket pricing system for a cinema. Different movie types have different ticket prices. Use method overriding to apply specific pricing logic for each movie type.

Classes:

- `Movie`: Base class with a virtual `calculatePrice()` method.
- `3DMovie` and `RegularMovie`: Derived classes that override `calculatePrice()` with different pricing schemes.

Sample Input:

```
3DMovie movie3D(15);    // Base price = 15, 3D surcharge
RegularMovie movieRegular(15);  // Base price = 15, no surcharge

movie3D.calculatePrice();    // Price for 3D Movie
movieRegular.calculatePrice(); // Price for Regular Movie
```

Sample Output:

```
3D Movie Price: 22.5
Regular Movie Price: 15
```

Problem 8: Shape Perimeter Calculation

Scenario: You need to calculate the perimeter of different shapes. Each shape has its own method for calculating the perimeter.

Classes:

- `Shape`: Base class with a virtual `calculatePerimeter()` method.
- `Circle` and `Square`: Derived classes that override `calculatePerimeter()`.

Sample Input:

```
Circle circle(7);  // Radius = 7
Square square(4);  // Side = 4

circle.calculatePerimeter();  // Perimeter for Circle
square.calculatePerimeter();  // Perimeter for Square
```

Sample Output:

```
Circle Perimeter: 43.9823
Square Perimeter: 16
```

Problem 9: Employee Performance Review

Scenario: You need to create a system that evaluates employee performance. Different types of employees have different performance evaluation criteria.

Classes:

- `Employee`: Base class with a virtual `evaluatePerformance()` method.
- `Manager` and `Developer`: Derived classes that override `evaluatePerformance()` with their own evaluation logic.

Sample Input:

```
Manager manager(85);  // Manager performance score = 85
Developer developer(90);  // Developer performance score = 90

manager.evaluatePerformance();  // Performance for Manager
developer.evaluatePerformance(); // Performance for Developer
```

Sample Output:

```
Manager Performance: Good
Developer Performance: Excellent
```

Problem 10: Payment System for Different Payment Methods

Scenario: You need to create a payment system where different payment methods (CreditCard, PayPal, etc.) process payments differently. Use method overriding to define each payment method's unique behavior.

Classes:

- `Payment`: Base class with a virtual `processPayment()` method.
- `CreditCardPayment` and `PayPalPayment`: Derived classes that override `processPayment()` with specific logic for each payment method.

Sample Input:

```
CreditCardPayment creditCard(500);  // Amount = 500
PayPalPayment paypal(300);  // Amount = 300

creditCard.processPayment();  // Credit card payment processing
paypal.processPayment();  // PayPal payment processing
```

Sample Output:

```
Processing Credit Card Payment of $500
Processing PayPal Payment of $300
```
