**Exercise 1: Creating an Abstract Class Shape.**

1. Create an abstract class named **Shape**.
2. Define an abstract method **calculateArea()** in the **Shape** class.
3. Create two subclasses, **Circle** and **Rectangle**, that inherit from the **Shape** class.
4. Implement the **calculateArea()** method in both the **Circle** and **Rectangle** classes to calculate the area of a circle and a rectangle, respectively.
5. Test your classes by creating instances of both **Circle** and **Rectangle** and calling the **calculateArea()** method on each.

**Exercise 2: Abstract Class Inheritance**

1. Create an abstract class **Animal** with an abstract method **makeSound()**.
2. Create two subclasses, **Dog** and **Cat**, that inherit from the **Animal** class.
3. Implement the **makeSound()** method in both the **Dog** and **Cat** classes to make appropriate sounds for each.
4. Create instances of both **Dog** and **Cat** and call the **makeSound()** method on each.

**Exercise 3: Banking System with Abstract Class**

Create a simple banking system using abstract classes. The system should have the following components:

1. An abstract class **Account** with the following properties and methods:
   * Properties: **accountNumber**, **accountHolder**, and **balance**.
   * Abstract methods: **withdraw(double amount)** and **deposit(double amount)**.
   * A constructor to initialize the account number, account holder's name, and initial balance.
   * A method **getAccountInfo()** to display account information(**toString() method**).
2. Two subclasses of **Account**: **SavingsAccount** and **CheckingAccount**. Implement the **withdraw** and **deposit** methods in both subclasses. For example, in a savings account, there might be a minimum balance requirement, and a checking account might not have such a requirement.
3. Create instances of both a **SavingsAccount** and a **CheckingAccount**. Perform some deposit and withdrawal operations and display the account information.

**Exercise 4: Abstract Shape Hierarchy**

Create an abstract shape hierarchy consisting of **Shape**, **Circle**, and **Square** classes. The classes should have the following properties and methods:

1. **Shape** (abstract class):
   * An abstract method **calculateArea()**.
   * An abstract method **calculatePerimeter()**.
   * A **name** property to store the shape's name.
   * A constructor to initialize the **name**.
2. **Circle** (subclass of **Shape**):
   * A **radius** property.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for a circle.
3. **Square** (subclass of **Shape**):
   * A **sideLength** property.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for a square.
4. Create instances of both a **Circle** and a **Square**. Calculate and display their areas and perimeters.

**Exercise 5: Abstract Employee Hierarchy**

Create an abstract employee hierarchy consisting of an abstract class **Employee** with two subclasses, **HourlyEmployee** and **SalariedEmployee**. The classes should have the following properties and methods:

1. **Employee** (abstract class):
   * Properties: **employeeID**, **firstName**, **lastName**.
   * An abstract method **calculateEarnings()** to calculate the employee's earnings.
   * A constructor to initialize the employee's information.
2. **HourlyEmployee** (subclass of **Employee**):
   * Additional property: **hourlyRate**.
   * Implement the **calculateEarnings()** method to calculate earnings for hourly employees.
3. **SalariedEmployee** (subclass of **Employee**):
   * Additional property: **monthlySalary**.
   * Implement the **calculateEarnings()** method to calculate earnings for salaried employees.
4. Create instances of both an **HourlyEmployee** and a **SalariedEmployee**. Calculate and display their earnings.

**Exercise 6: Abstract Geometric Shapes**

Create an abstract class **Shape** with two subclasses, **Circle** and **Triangle**. The classes should have the following properties and methods:

1. **Shape** (abstract class):
   * An abstract method **calculateArea()** to calculate the area of the shape.
   * An abstract method **calculatePerimeter()** to calculate the perimeter of the shape.
   * A constructor to initialize the name of the shape.
2. **Circle** (subclass of **Shape**):
   * Additional property: **radius**.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for circles.
3. **Triangle** (subclass of **Shape**):
   * Additional properties: **base** and **height**.
   * Implement the **calculateArea()** and **calculatePerimeter()** methods for triangles.
4. Create instances of both a **Circle** and a **Triangle**. Calculate and display their areas and perimeters.

**Exercise 7: Abstract Media Players**

Create an abstract class **MediaPlayer** with two subclasses, **AudioPlayer** and **VideoPlayer**. The classes should have the following properties and methods:

1. **MediaPlayer** (abstract class):
   * Properties: **mediaName** and **volume**.
   * An abstract method **play()** to play media.
   * An abstract method **stop()** to stop media playback.
   * A constructor to initialize the media name and volume.
2. **AudioPlayer** (subclass of **MediaPlayer**):
   * Implement the **play()** method to play audio.
   * Implement the **stop()** method to stop audio playback.
3. **VideoPlayer** (subclass of **MediaPlayer**):
   * Implement the **play()** method to play video.
   * Implement the **stop()** method to stop video playback.
4. Create instances of both an **AudioPlayer** and a **VideoPlayer**. Play and stop media, and adjust the volume.

**Exercise 8: Abstract School Courses**

Create an abstract class **Course** with two subclasses, **MathCourse** and **ScienceCourse**. The classes should have the following properties and methods:

1. **Course** (abstract class):
   * Properties: **courseName**, **courseCode**, and **instructorName**.
   * An abstract method **calculateGrade()** to calculate a student's grade.
   * A constructor to initialize the course name, course code, and instructor name.
2. **MathCourse** (subclass of **Course**):
   * Additional property: **numberOfAssignments**.
   * Implement the **calculateGrade()** method for math courses.
3. **ScienceCourse** (subclass of **Course**):
   * Additional property: **numberOfExperiments**.
   * Implement the **calculateGrade()** method for science courses.
4. Create instances of both a **MathCourse** and a **ScienceCourse**. Calculate and display a student's grade.

**Exercise 9: Abstract Vehicle Hierarchy**

Create an abstract class **Vehicle** with two subclasses, **Car** and **Motorcycle**. The classes should have the following properties and methods:

1. **Vehicle** (abstract class):
   * Properties: **make**, **model**, and **year**.
   * An abstract method **start()**.
   * An abstract method **stop()**.
   * A constructor to initialize the make, model, and year.
2. **Car** (subclass of **Vehicle**):
   * Implement the **start()** method to start the car's engine.
   * Implement the **stop()** method to stop the car's engine.
3. **Motorcycle** (subclass of **Vehicle**):
   * Implement the **start()** method to start the motorcycle's engine.
   * Implement the **stop()** method to stop the motorcycle's engine.
4. Create instances of both a **Car** and a **Motorcycle**. Start and stop their engines.

**Exercise 10: Abstract Payment System**

Create an abstract payment system with an abstract class **Payment** and two subclasses, **CreditCardPayment** and **PaypalPayment**. The classes should have the following properties and methods:

1. **Payment** (abstract class):
   * Properties: **amount**, **recipient**, and **paymentDate**.
   * An abstract method **processPayment()** to process the payment.
   * A constructor to initialize the payment details.
2. **CreditCardPayment** (subclass of **Payment**):
   * Additional properties: **cardNumber**, **cardHolder**, and **expirationDate**.
   * Implement the **processPayment()** method to process credit card payments.
3. **PaypalPayment** (subclass of **Payment**):
   * Additional properties: **email** and **password**.
   * Implement the **processPayment()** method to process PayPal payments.
4. Create instances of both a **CreditCardPayment** and a **PaypalPayment**. Process payments using both methods.