Exercise Question:

Problem Statement:

Create a Java class called `Circle` to represent a circle. The `Circle` class should have the following attributes:

- `radius` (a double): the radius of the circle.

The `Circle` class should also have the following methods:

1. `getRadius()`: Returns the value of the radius.
2. `setRadius(double radius)`: Sets the value of the radius to the given value.
3. `getArea()`: Returns the area of the circle (π * radius * radius).
4. `getCircumference()`: Returns the circumference of the circle (2 * π * radius).

Create another class called `CircleTester` to test the `Circle` class. In the `CircleTester` class, do the following:

1. Create two `Circle` objects, `circle1` and `circle2`, with initial radii of 5.0 and 7.5 respectively.
2. Use the `getRadius()`, `getArea()`, and `getCircumference()` methods to display the radius, area, and circumference of both circles.
3. Change the radius of `circle1` to 10.0 using the `setRadius()` method.
4. Display the updated radius, area, and circumference of `circle1`.
5. Calculate and display the difference in area between `circle2` and the updated `circle1`.

Solution:

```java
// Circle.java
public class Circle {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Getter for radius
    public double getRadius() {
        return radius;
    }

    // Setter for radius
    public void setRadius(double radius) {
        this.radius = radius;
    }

    // Method to calculate and return the area of the circle
    public double getArea() {
        return Math.PI * radius * radius;
    }

    // Method to calculate and return the circumference of the circle
    public double getCircumference() {
        return 2 * Math.PI * radius;
    }
}

// CircleTester.java
public class CircleTester {
    public static void main(String[] args) {
        // Create two Circle objects
        Circle circle1 = new Circle(5.0);
        Circle circle2 = new Circle(7.5);

        // Display information for circle1
        System.out.println("Circle 1 - Radius: " + circle1.getRadius());
        System.out.println("Circle 1 - Area: " + circle1.getArea());
        System.out.println("Circle 1 - Circumference: " + circle1.getCircumference());

        // Change the radius of circle1
        circle1.setRadius(10.0);

        // Display updated information for circle1
        System.out.println("\nUpdated Circle 1 - Radius: " + circle1.getRadius());
        System.out.println("Updated Circle 1 - Area: " + circle1.getArea());
        System.out.println("Updated Circle 1 - Circumference: " + circle1.getCircumference());

        // Calculate and display the difference in area between circle2 and updated circle1
        double areaDifference = circle2.getArea() - circle1.getArea();
        System.out.println("\nDifference in Area (Circle 2 - Updated Circle 1): " + areaDifference);
    }
}
```

This Java program defines a `Circle` class and a `CircleTester` class to demonstrate its functionality. It creates two circles, displays their properties, updates one of the circles, and calculates the difference in area between the two circles.

Inheritance Exercise Question:

Problem Statement:

Imagine you are building a system to manage a library's collection of books. Design a class hierarchy using inheritance to represent different types of books. Start with a base class called `Book` and then create two subclasses: `Fiction` and `NonFiction`.

1. Create a `Book` class with the following attributes and methods:
   - Attributes:
     - `title` (String): the title of the book.
     - `author` (String): the author of the book.
     - `yearPublished` (int): the year the book was published.
   - Methods:
     - `getTitle()`: Returns the title of the book.
     - `getAuthor()`: Returns the author of the book.
     - `getYearPublished()`: Returns the year the book was published.

2. Create a `Fiction` class that inherits from `Book` and includes an additional attribute:
   - Attributes:
     - `genre` (String): the genre of the fiction book.
   - Methods:
     - `getGenre()`: Returns the genre of the fiction book.

3. Create a `NonFiction` class that also inherits from `Book` and includes an additional attribute:
   - Attributes:
     - `subject` (String): the subject of the non-fiction book.
   - Methods:

  - `getSubject()`: Returns the subject of the non-fiction book.

Now, create instances of both `Fiction` and `NonFiction` books in a test program to demonstrate the inheritance and functionality of the classes. Display the details of these books, including their titles, authors, publication years, genres (for fiction), and subjects (for non-fiction).

Solution:

```java
// Book.java
public class Book {
    private String title;
    private String author;
    private int yearPublished;

    public Book(String title, String author, int yearPublished) {
        this.title = title;
        this.author = author;
        this.yearPublished = yearPublished;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public int getYearPublished() {
        return yearPublished;
    }
}

// Fiction.java
public class Fiction extends Book {
    private String genre;

    public Fiction(String title, String author, int yearPublished, String genre) {
        super(title, author, yearPublished);
        this.genre = genre;
    }

    public String getGenre() {
        return genre;
    }
}

// NonFiction.java
public class NonFiction extends Book {
    private String subject;

    public NonFiction(String title, String author, int yearPublished, String subject) {
        super(title, author, yearPublished);
        this.subject = subject;
    }

    public String getSubject() {
        return subject;
    }
}

// BookTester.java
public class BookTester {
    public static void main(String[] args) {
        Fiction fictionBook = new Fiction("The Great Gatsby", "F. Scott Fitzgerald", 1925, "Drama");
        NonFiction nonFictionBook = new NonFiction("Sapiens: A Brief History of Humankind", "Yuval Noah Harari", 2011, "Anthropology");

        // Display details of the fiction book
        System.out.println("Fiction Book Details:");
        System.out.println("Title: " + fictionBook.getTitle());
        System.out.println("Author: " + fictionBook.getAuthor());
        System.out.println("Year Published: " + fictionBook.getYearPublished());
        System.out.println("Genre: " + fictionBook.getGenre());

        // Display details of the non-fiction book
        System.out.println("\nNon-Fiction Book Details:");
        System.out.println("Title: " + nonFictionBook.getTitle());
        System.out.println("Author: " + nonFictionBook.getAuthor());
        System.out.println("Year Published: " + nonFictionBook.getYearPublished());
        System.out.println("Subject: " + nonFictionBook.getSubject());
    }
}
```

This Java program demonstrates the use of inheritance to create `Fiction` and `NonFiction` subclasses derived from the `Book` base class. Instances of these classes are created and their details are displayed in the `BookTester` class.

Wrapper Class Exercise Question:

Problem Statement:

You are working on a project that involves processing various types of data, including integers, floating-point numbers, and characters. To make the data processing more flexible, you decide to create a set of wrapper classes to encapsulate these data types and provide useful methods.

Create the following wrapper classes:

1. **IntWrapper:** A wrapper class for integers. It should have the following methods:
   - `IntWrapper(int value)`: Constructor that initializes the wrapped integer.
   - `getValue()`: Returns the wrapped integer value.
   - `setValue(int value)`: Sets a new value for the wrapped integer.
   - `isEven()`: Returns true if the wrapped integer is even, false otherwise.
   - `isOdd()`: Returns true if the wrapped integer is odd, false otherwise.

2. **FloatWrapper:** A wrapper class for floating-point numbers. It should have the following methods:
   - `FloatWrapper(float value)`: Constructor that initializes the wrapped floating-point number.
   - `getValue()`: Returns the wrapped floating-point value.
   - `setValue(float value)`: Sets a new value for the wrapped floating-point number.
   - `isPositive()`: Returns true if the wrapped number is positive, false otherwise.
   - `isNegative()`: Returns true if the wrapped number is negative, false otherwise.

3. **CharWrapper:** A wrapper class for characters. It should have the following methods:
   - `CharWrapper(char value)`: Constructor that initializes the wrapped character.
   - `getValue()`: Returns the wrapped character value.
   - `setValue(char value)`: Sets a new value for the wrapped character.

  - `isVowel()`: Returns true if the wrapped character is a vowel (a, e, i, o, u), false otherwise.

  - `isConsonant()`: Returns true if the wrapped character is a consonant (not a vowel), false otherwise.

Create a test program called `WrapperTester` that demonstrates the use of these wrapper classes. Create instances of each wrapper class, perform various operations, and display the results.

Solution:

```java
// IntWrapper.java
public class IntWrapper {
    private int value;

    public IntWrapper(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public boolean isEven() {
        return value % 2 == 0;
    }

    public boolean isOdd() {
        return value % 2 != 0;
    }
}

// FloatWrapper.java
public class FloatWrapper {
    private float value;
    public FloatWrapper(float value) {
        this.value = value;
    }

    public float getValue() {
        return value;
    }

    public void setValue(float value) {
        this.value = value;
    }

    public boolean isPositive() {
        return value > 0;
    }

    public boolean isNegative() {
        return value < 0;
    }
}

// CharWrapper.java
public class CharWrapper {
    private char value;

    public CharWrapper(char value) {
        this.value = value;
    }

    public char getValue() {
        return value;
    }

    public void setValue(char value) {
        this.value = value;
    }
    public boolean isVowel() {
        char lowerValue = Character.toLowerCase(value);
        return lowerValue == 'a' || lowerValue == 'e' || lowerValue == 'i' || lowerValue == 'o' || lowerValue == 'u';
    }

    public boolean isConsonant() {
        return !isVowel();
    }
}

// WrapperTester.java
public class WrapperTester {
    public static void main(String[] args) {
        // Test IntWrapper
        IntWrapper intWrapper = new IntWrapper(10);
        System.out.println("IntWrapper Value: " + intWrapper.getValue());
        System.out.println("Is Even: " + intWrapper.isEven());
        System.out.println("Is Odd: " + intWrapper.isOdd());

        // Test FloatWrapper
        FloatWrapper floatWrapper = new FloatWrapper(-3.5f);
        System.out.println("\nFloatWrapper Value: " + floatWrapper.getValue());
        System.out.println("Is Positive: " + floatWrapper.isPositive());
        System.out.println("Is Negative: " + floatWrapper.isNegative());

        // Test CharWrapper
        CharWrapper charWrapper = new CharWrapper('E');
        System.out.println("\nCharWrapper Value: " + charWrapper.getValue());
        System.out.println("Is Vowel: " + charWrapper.isVowel());
        System.out.println("Is Consonant: " + charWrapper.isConsonant());
    }
}
```

This Java program defines three wrapper classes (`IntWrapper`, `FloatWrapper`, and `CharWrapper`) and a test program (`WrapperTester`) to demonstrate their functionality. It allows you to create instances of each wrapper class, set values, and perform various operations on them.

Real-Life Problem in Object-Oriented Programming (Java): Bank Account Management

Problem Statement:

You are tasked with creating a Java program to manage bank accounts for a financial institution. Each bank account should have the following attributes and methods:

BankAccount Class:

- Attributes:
  - `accountNumber` (String): a unique identifier for the bank account.
  - `accountHolder` (String): the name of the account holder.
  - `balance` (double): the current balance in the account.

- Methods:
  - `deposit(double amount)`: Adds the specified amount to the account balance.
  - `withdraw(double amount)`: Subtracts the specified amount from the account balance.
  - `getAccountNumber()`: Returns the account number.
  - `getAccountHolder()`: Returns the account holder's name.
  - `getBalance()`: Returns the current balance.
  - `toString()`: Returns a string representation of the account.

SavingsAccount Class (Inherits from BankAccount):

- Attributes:
  - `interestRate` (double): the annual interest rate for the savings account.

- Methods:
  - `calculateInterest()`: Calculates and returns the interest earned on the current balance based on the interest rate.

CheckingAccount Class (Inherits from BankAccount):

- Attributes:
  - `overdraftLimit` (double): the maximum negative balance allowed (overdraft limit).

- Methods:
  - `withdraw(double amount)`: Overrides the `withdraw` method to allow overdrafts within the specified limit. If the withdrawal exceeds the balance + overdraft limit, it should be denied.

Create a test program (`BankAccountManager`) that demonstrates the use of these classes. Create instances of both `SavingsAccount` and `CheckingAccount`, perform various transactions, and display the account details.

Solution:

```java
// BankAccount.java
public class BankAccount {
    private String accountNumber;
    private String accountHolder;
    private double balance;

    public BankAccount(String accountNumber, String accountHolder, double balance) {
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public boolean withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            return true;
        }
        return false;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public String getAccountHolder() {
        return accountHolder;
    }
    public double getBalance() {
        return balance;
    }

    @Override
    public String toString() {
        return "Account Number: " + accountNumber + "\nAccount Holder: " + accountHolder + "\nBalance: $" + balance;
    }
}

// SavingsAccount.java
public class SavingsAccount extends BankAccount {
    private double interestRate;

    public SavingsAccount(String accountNumber, String accountHolder, double balance, double interestRate) {
        super(accountNumber, accountHolder, balance);
        this.interestRate = interestRate;
    }

    public double calculateInterest() {
        return getBalance() * (interestRate / 100);
    }
}

// CheckingAccount.java
public class CheckingAccount extends BankAccount {
    private double overdraftLimit;

    public CheckingAccount(String accountNumber, String accountHolder, double balance, double overdraftLimit) {
        super(accountNumber, accountHolder, balance);
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    public boolean withdraw(double amount) {
        if (amount <= (getBalance() + overdraftLimit)) {
            balance -= amount;
            return true;
        }
        return false;
    }
}

// BankAccountManager.java
public class BankAccountManager {
    public static void main(String[] args) {
        // Create a savings account
        SavingsAccount savingsAccount = new SavingsAccount("SA123456", "Alice", 1000.0, 3.5);

        // Create a checking account
        CheckingAccount checkingAccount = new CheckingAccount("CA789012", "Bob", 500.0, 200.0);

        // Perform transactions
        savingsAccount.deposit(200.0);
        checkingAccount.withdraw(700.0);

        // Display account details
        System.out.println("Savings Account Details:\n" + savingsAccount);
        System.out.println("\nChecking Account Details:\n" + checkingAccount);

        // Calculate and display interest for the savings account
        double interest = savingsAccount.calculateInterest();
        System.out.println("\nInterest Earned: $" + interest);
    }
}
```

This Java program models a bank account management system with `BankAccount`, `SavingsAccount`, and `CheckingAccount` classes. The `BankAccountManager` test program demonstrates transactions and interest calculations for the account types.