Southern University Bangladesh

Department of Computer Science and Engineering

Faculty of Science and Engineering

Object Oriented Programming Lab

Lab Report: 07

Course code: CSE 0613-108

Submitted by:

Name : Sharmistha Chowdhury

ID: 666-61-60

Submitted to :

Mohammed Arif Hasan Chowdhury, Assistant Professor

Dept. of Computer Science and Engineering

Date: 20/05/2024

Problem 01: Write a Java program to create an interface Shape with the getArea() method.Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getArea() method for each of the three classes.

Objective:

The objective of this Java program is to demonstrate the use of interfaces in Java by calculating and printing the area of three different shapes: a rectangle, a circle, and a triangle.

Algorithm:

1. Define the Shape interface: This interface contains one abstract method, getArea(), which will be implemented in the classes that implement this interface.

2. Define the Rectangle class that implements Shape: This class has two instance variables length and width and a constructor to initialize them. It also provides the implementation for the abstract method:
- getArea(): Returns the area of the rectangle, calculated as length * width.

3. Define the Circle class that implements Shape: This class has an instance variable radius and a constructor to initialize it. It also provides the implementation for the abstract method:
- getArea(): Returns the area of the circle, calculated as Math.PI * Math.pow(radius, 2).

4. Define the Triangle class that implements Shape: This class has two instance variables base and height and a constructor to initialize them. It also provides the implementation for the abstract method:
- getArea(): Returns the area of the triangle, calculated as 0.5 * base * height.

5. In the main method:
- Create a Rectangle object and call its getArea() method. Print the area of the rectangle.
- Create a Circle object and call its getArea() method. Print the area of the circle.
- Create a Triangle object and call its getArea() method. Print the area of the triangle..

Code:

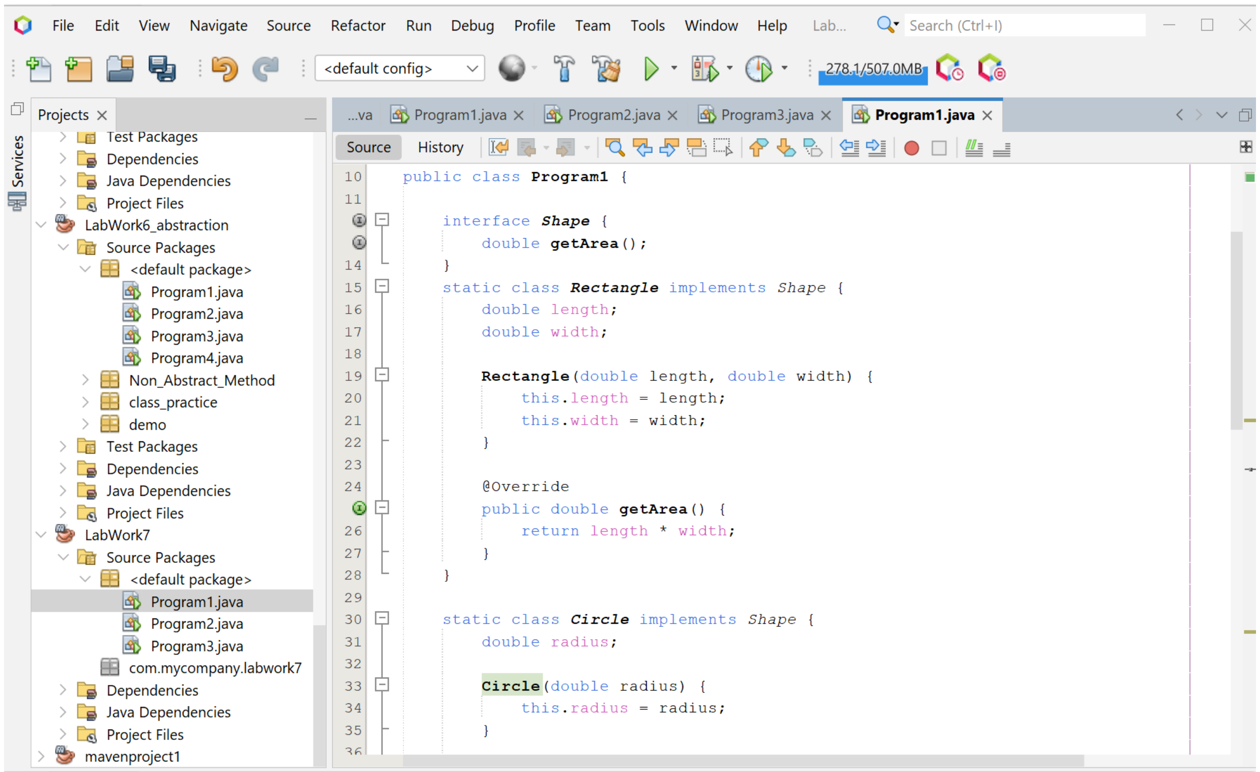
```
public class Program1 {

   interface Shape {
      double getArea();
   }
   static class Rectangle implements Shape {
      double length;
      double width;

      Rectangle(double length, double width) {
         this.length = length;
         this.width = width;
      }

      @Override
      public double getArea() {
         return length * width;
      }
   }

   static class Circle implements Shape {
      double radius;

      Circle(double radius) {
         this.radius = radius;
      }

      @Override
      public double getArea() {
         return Math.PI * Math.pow(radius, 2);
      }
```

```
    }

    static class Triangle implements Shape {
        double base;
        double height;

        Triangle(double base, double height) {
            this.base = base;
            this.height = height;
        }

        @Override
        public double getArea() {
            return 0.5 * base * height;
        }
    }

    public static void main(String[] args) {

        Rectangle rectangle = new Rectangle(5, 7);
        System.out.println("Area of rectangle: " + rectangle.getArea());

        Circle circle = new Circle(3);
        System.out.println("Area of circle: " + circle.getArea());

        Triangle triangle = new Triangle(4, 6);
        System.out.println("Area of triangle: " + triangle.getArea());
    }
}
```

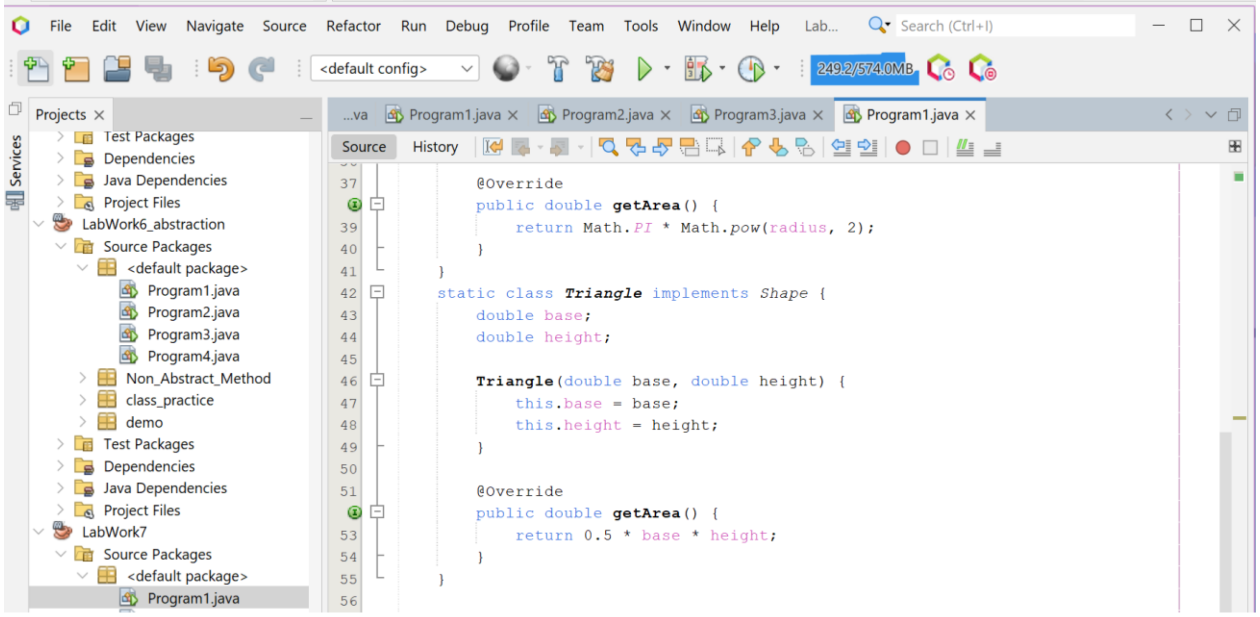
Input/ Expected Output:

```
Area of rectangle: 35.0
Area of circle: 28.274333882308138
Area of triangle: 12.0
```

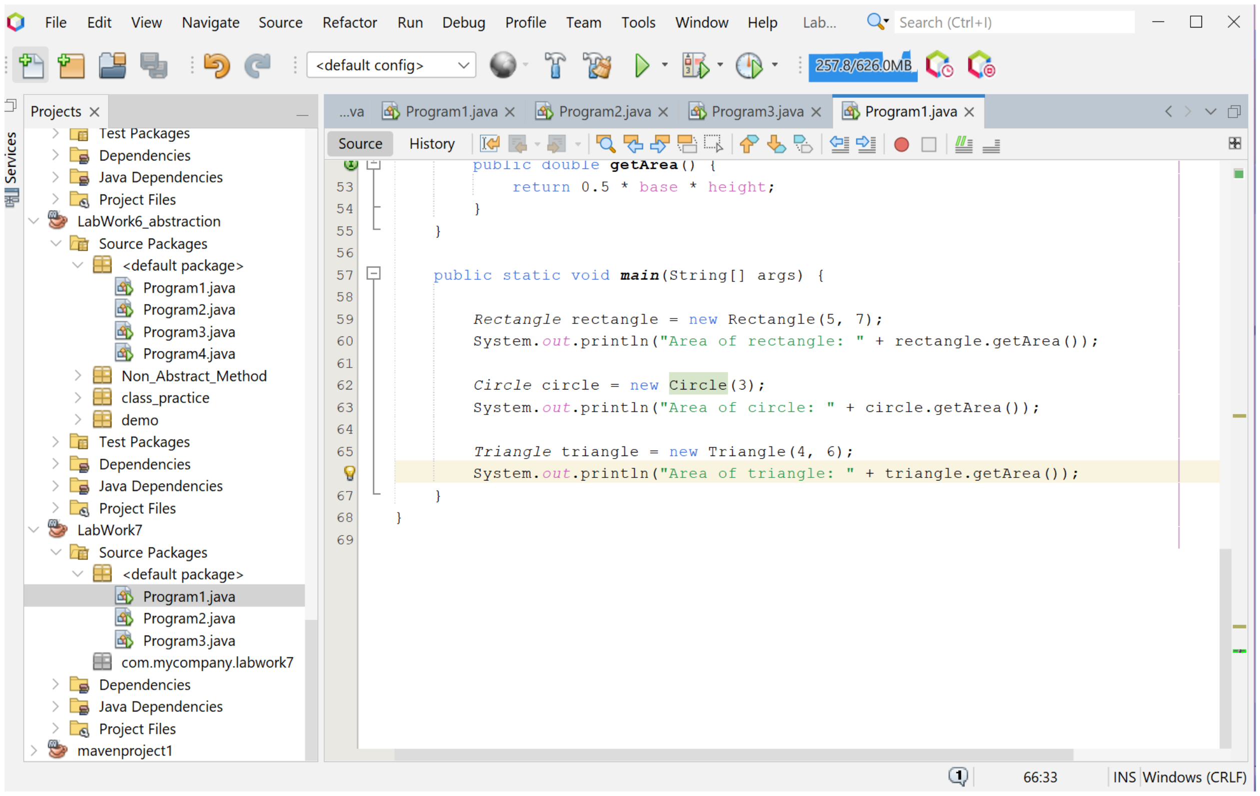
Screenshot of code edition window:



```java
public class Program1 {

    interface Shape {
        double getArea();
    }
    static class Rectangle implements Shape {
        double length;
        double width;

        Rectangle(double length, double width) {
            this.length = length;
            this.width = width;
        }

        @Override
        public double getArea() {
            return length * width;
        }
    }

    static class Circle implements Shape {
        double radius;

        Circle(double radius) {
            this.radius = radius;
        }
```

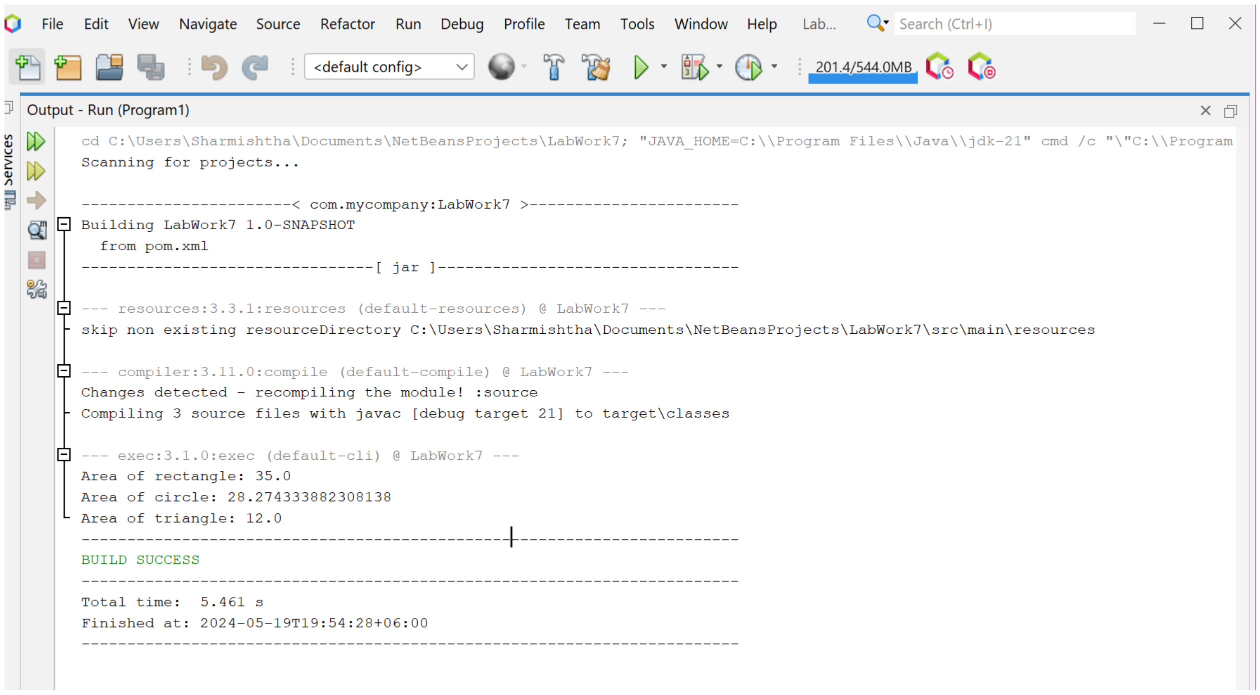


```java
        @Override
        public double getArea() {
            return Math.PI * Math.pow(radius, 2);
        }
    }
    static class Triangle implements Shape {
        double base;
        double height;

        Triangle(double base, double height) {
            this.base = base;
            this.height = height;
        }

        @Override
        public double getArea() {
            return 0.5 * base * height;
        }
    }
```



```java
    public double getArea() {
        return 0.5 * base * height;
    }
}

public static void main(String[] args) {

    Rectangle rectangle = new Rectangle(5, 7);
    System.out.println("Area of rectangle: " + rectangle.getArea());

    Circle circle = new Circle(3);
    System.out.println("Area of circle: " + circle.getArea());

    Triangle triangle = new Triangle(4, 6);
    System.out.println("Area of triangle: " + triangle.getArea());
    }
}
```

Screenshot of Output screen/Run screen window:



```
cd C:\Users\Sharmishtha\Documents\NetBeansProjects\LabWork7; "JAVA_HOME=C:\\Program Files\\Java\\jdk-21" cmd /c "\"C:\\Program
Scanning for projects...

----------------------< com.mycompany:LabWork7 >----------------------
Building LabWork7 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]--------------------------------

--- resources:3.3.1:resources (default-resources) @ LabWork7 ---
skip non existing resourceDirectory C:\Users\Sharmishtha\Documents\NetBeansProjects\LabWork7\src\main\resources

--- compiler:3.11.0:compile (default-compile) @ LabWork7 ---
Changes detected - recompiling the module! :source
Compiling 3 source files with javac [debug target 21] to target\classes

--- exec:3.1.0:exec (default-cli) @ LabWork7 ---
Area of rectangle: 35.0
Area of circle: 28.274333882308138
Area of triangle: 12.0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  5.461 s
Finished at: 2024-05-19T19:54:28+06:00
------------------------------------------------------------------------
```

Explanation:

This Java code snippet begins the definition of a program that uses interfaces to model the behavior of shapes. An interface `Shape` is defined with an abstract method: `getArea()`. This method is intended to be implemented in the classes that implement the `Shape` interface, each representing a specific type of shape. The `getArea()` method, when implemented, should return the area of the shape.

Discussion

The use of interfaces in this context is a demonstration of the principles of abstraction and polymorphism in object-oriented programming. The `Shape` interface serves as a contract for creating more specific shape classes, and each specific shape class is expected to provide its own implementation of the `getArea()` method. This design allows for code that is organized and easy to understand, as each shape class will be responsible for its own specific behaviors. Furthermore, it allows for flexibility, as more shape types can be easily added in the future without having to modify the existing code, making the code more maintainable and adaptable to changes. This program is a great starting point for a more complex system that models various shapes and their behaviors.

Problem 02: Write a Java programming to create a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw, calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the Account interface and have their own unique methods.

Objective:

The objective of this Java program is to simulate a simple banking system with two types of accounts (Savings and Current), allowing deposits and withdrawals, demonstrating the use of interfaces in Java.

Algorithm:

1. Define the Account interface: This interface contains four abstract methods: `deposit(double amount)`, `withdraw(double amount)`, `calculateInterest()`, and `getBalance()`.

2. **Define the SavingsAccount class that implements Account**: This class has two instance variables `balance` and `interestRate`, and a constructor to initialize `interestRate`. It also provides the implementation for the abstract methods:
   - `deposit(double amount)`: Increases the balance by the amount.
   - `withdraw(double amount)`: Checks if the balance is greater than or equal to the amount. If it is, decreases the balance by the amount. If not, prints a message about insufficient balance.
   - `calculateInterest()`: Returns the interest calculated as `balance * (interestRate / 100)`.
   - `getBalance()`: Returns the balance.

3. **Define the CurrentAccount class that implements Account:** This class has an instance variable `balance` and provides the implementation for the abstract methods. The implementations are similar to those in the SavingsAccount class, except `calculateInterest()` always returns 0.

4. **Define the Bank class**: This class has an instance variable `account` and a method `addAccount(Account account)` to set the account.

5. **In the main method**:
   - Create a Scanner object to read user input.
   - Prompt the user to enter the interest rate for the savings account and create a SavingsAccount object with the entered interest rate. Add the savings account to the bank.
   - Prompt the user to enter an amount to deposit into the savings account, perform the deposit, and print the new balance.
   - Add interest to the savings account and print the new balance.
   - Prompt the user to enter an amount to withdraw from the savings account, perform the withdrawal, and print the new balance.

   - Create a CurrentAccount object and add it to the bank.
   - Prompt the user to enter an amount to deposit into the current account, perform the deposit, and print the new balance.
   - Prompt the user to enter an amount to withdraw from the current account, perform the withdrawal, and print the new balance.
   - Close the Scanner object.

Code:

```
public class Program2 {


  interface Account {
    void deposit(double amount);
    void withdraw(double amount);
    double calculateInterest();
    double getBalance();
  }


  static class SavingsAccount implements Account {
    private double balance;
    private double interestRate; // Interest rate


    SavingsAccount(double interestRate) {
      this.interestRate = interestRate;
    }


    void addInterest() {
      balance += calculateInterest();
    }


    @Override
    public void deposit(double amount) {
      balance += amount;
    }
    @Override
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance");
        }
    }

    @Override
    public double calculateInterest() {
        return balance * (interestRate / 100);
    }

    @Override
    public double getBalance() {
        return balance;
    }
}
static class CurrentAccount implements Account {
    private double balance;

    void checkMinimumBalance() {
        if (balance < 1000) {
            System.out.println("Balance is below the minimum threshold");
        }
    }

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

```
        @Override
        public void withdraw(double amount) {
            if (balance >= amount) {
                balance -= amount;
            } else {
                System.out.println("Insufficient balance");
            }
        }
        @Override
        public double calculateInterest() {
            return 0; // No interest for current accounts
        }
    @Override
        public double getBalance() {
            return balance;
        }
    }
    static class Bank {
        private Account account;

        void addAccount(Account account) {
            this.account = account;
        }
    }
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the interest rate for the savings account:");
        double interestRate = scanner.nextDouble();



        SavingsAccount savingsAccount = new SavingsAccount(interestRate);
        Bank bank = new Bank();
        bank.addAccount(savingsAccount);
```

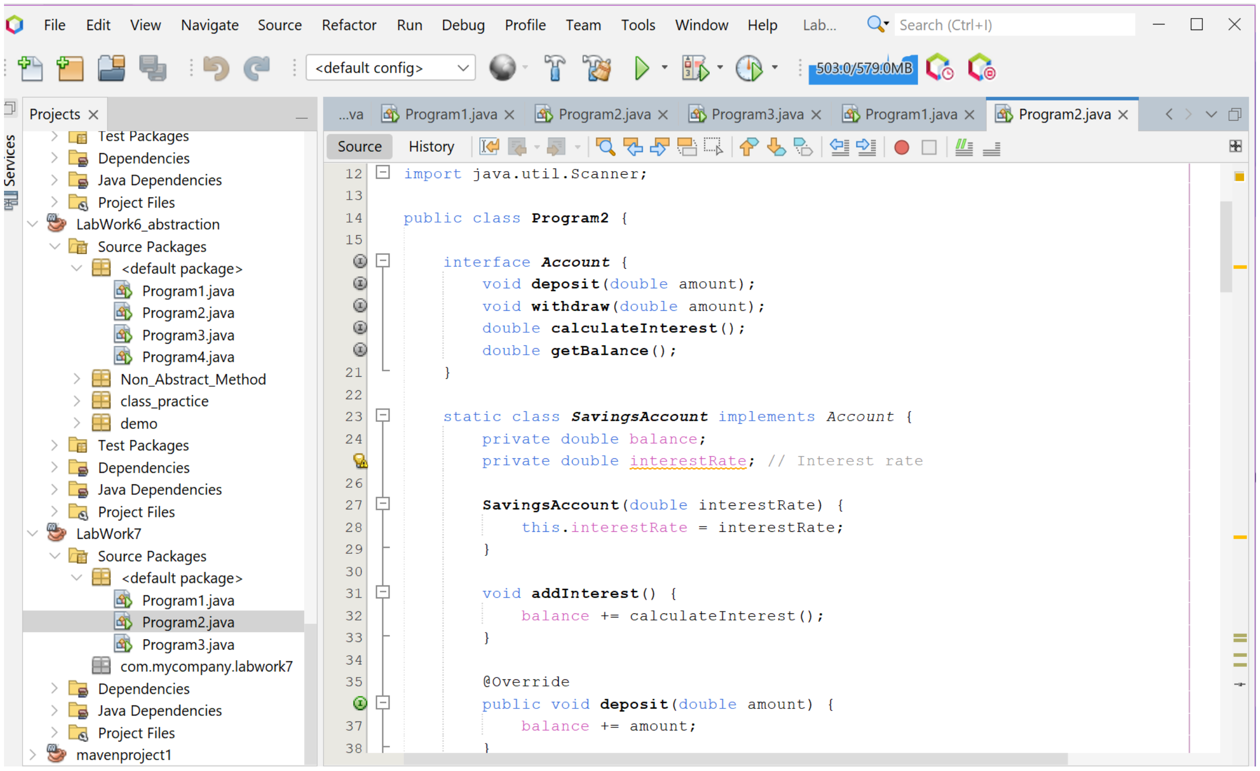
```
System.out.println("Enter amount to deposit into savings account:");
double amount = scanner.nextDouble();
savingsAccount.deposit(amount);

System.out.println("New balance of savings account: " + savingsAccount.getBalance());

savingsAccount.addInterest();
System.out.println("Balance after adding interest: " + savingsAccount.getBalance());

System.out.println("Enter amount to withdraw from savings account:");
amount = scanner.nextDouble();
savingsAccount.withdraw(amount);
System.out.println("Balance after withdrawal: " + savingsAccount.getBalance());


CurrentAccount currentAccount = new CurrentAccount();
bank.addAccount(currentAccount);


System.out.println("Enter amount to deposit into current account:");
amount = scanner.nextDouble();
currentAccount.deposit(amount);

System.out.println("New balance of current account: " + currentAccount.getBalance());


System.out.println("Enter amount to withdraw from current account:");
amount = scanner.nextDouble();
currentAccount.withdraw(amount);
System.out.println("Balance after withdrawal: " + currentAccount.getBalance());
```

```
        scanner.close();
    }
}
```

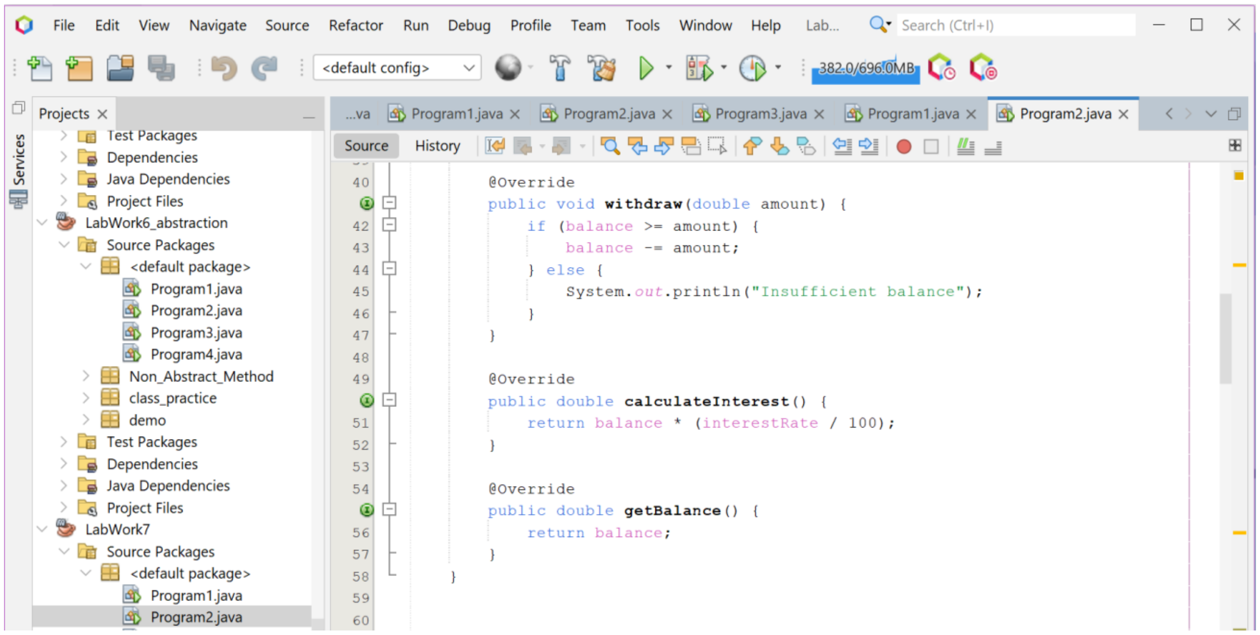
Input/ Expected Output:

```
Enter the interest rate for the savings account:
 0.05
Enter amount to deposit into savings account:
100000
New balance of savings account: 100000.0
Balance after adding interest: 100050.0
Enter amount to withdraw from savings account: 400000
Insufficient balance
Balance after withdrawal: 100050.0
Enter amount to deposit into current account: 60000
New balance of current account: 60000.0
Enter amount to withdraw from current account: 6000
Balance after withdrawal: 54000.0
```

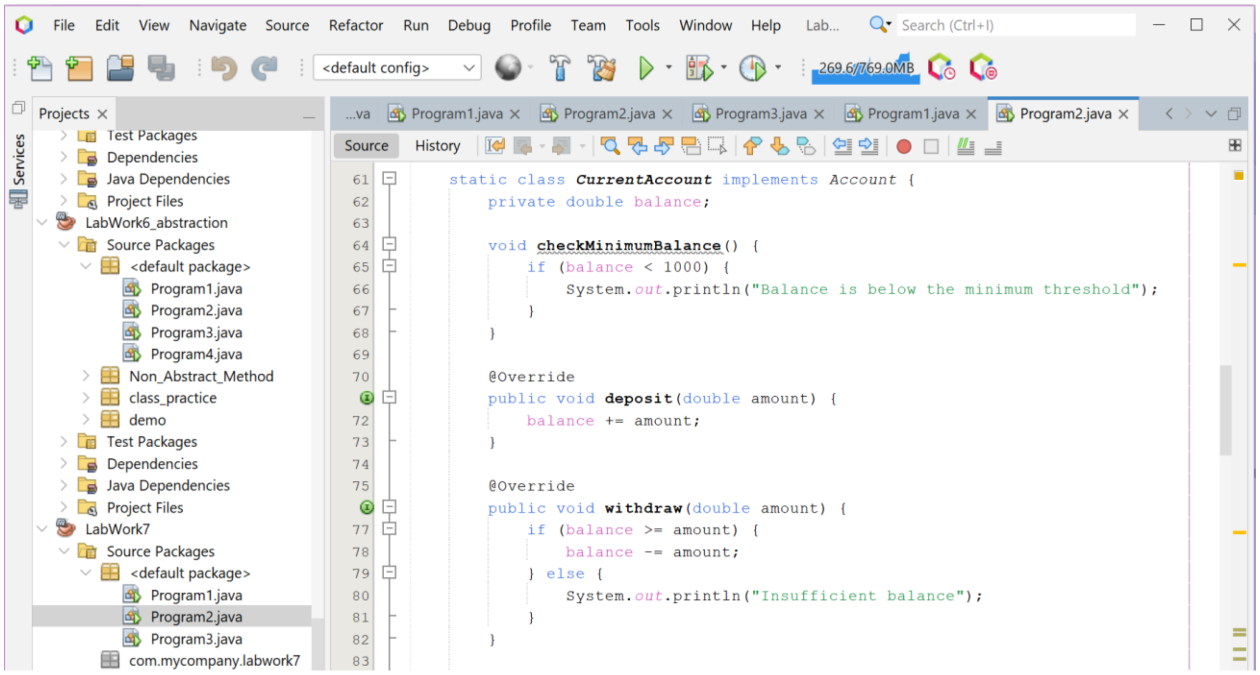
Screenshot of code edition window:



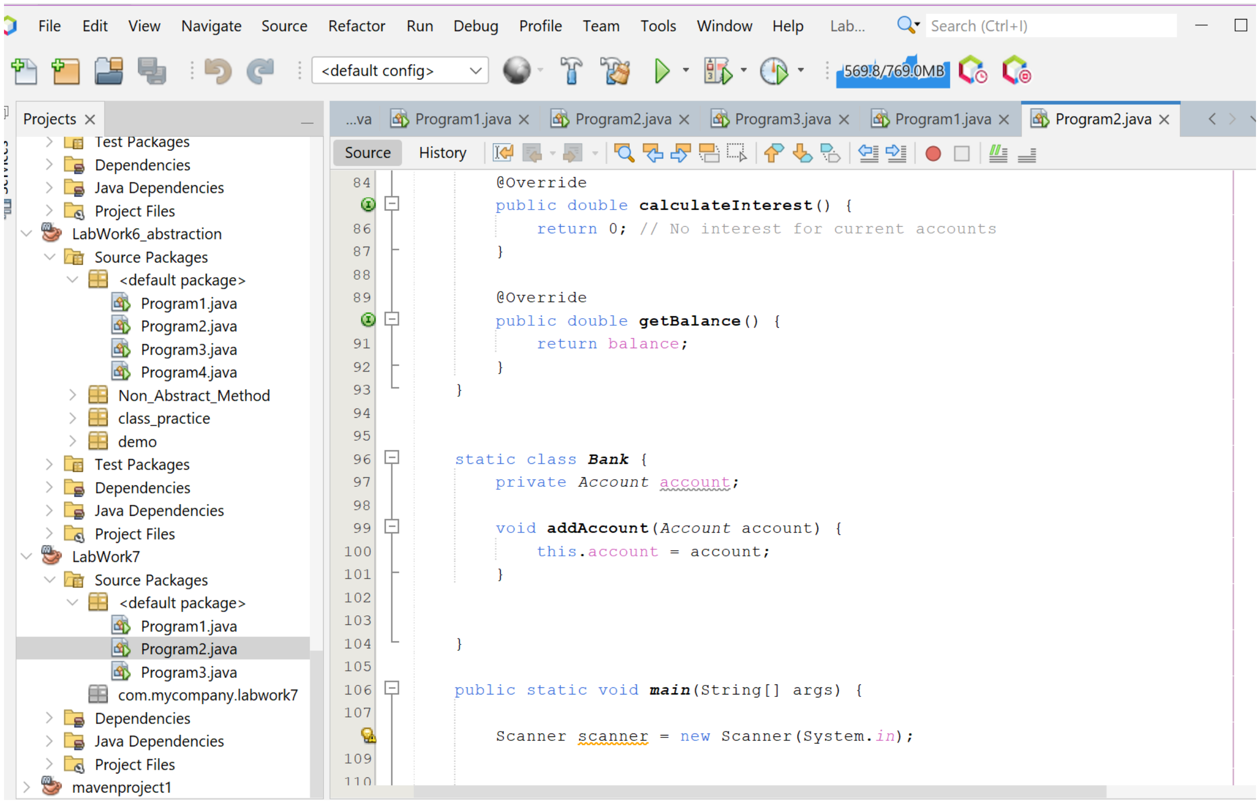
```java
import java.util.Scanner;

public class Program2 {

    interface Account {
        void deposit(double amount);
        void withdraw(double amount);
        double calculateInterest();
        double getBalance();
    }

    static class SavingsAccount implements Account {
        private double balance;
        private double interestRate; // Interest rate

        SavingsAccount(double interestRate) {
            this.interestRate = interestRate;
        }

        void addInterest() {
            balance += calculateInterest();
        }

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



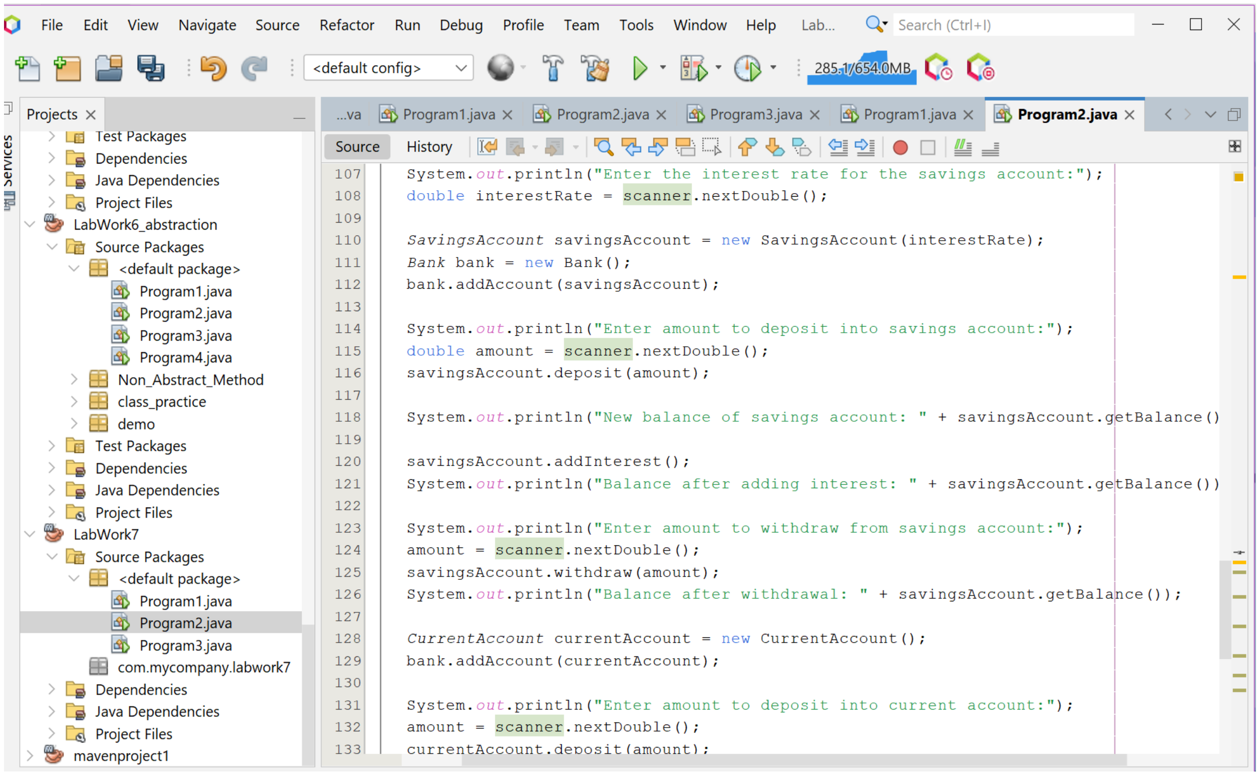
```java
        @Override
        public void withdraw(double amount) {
            if (balance >= amount) {
                balance -= amount;
            } else {
                System.out.println("Insufficient balance");
            }
        }

        @Override
        public double calculateInterest() {
            return balance * (interestRate / 100);
        }

        @Override
        public double getBalance() {
            return balance;
        }
    }
```



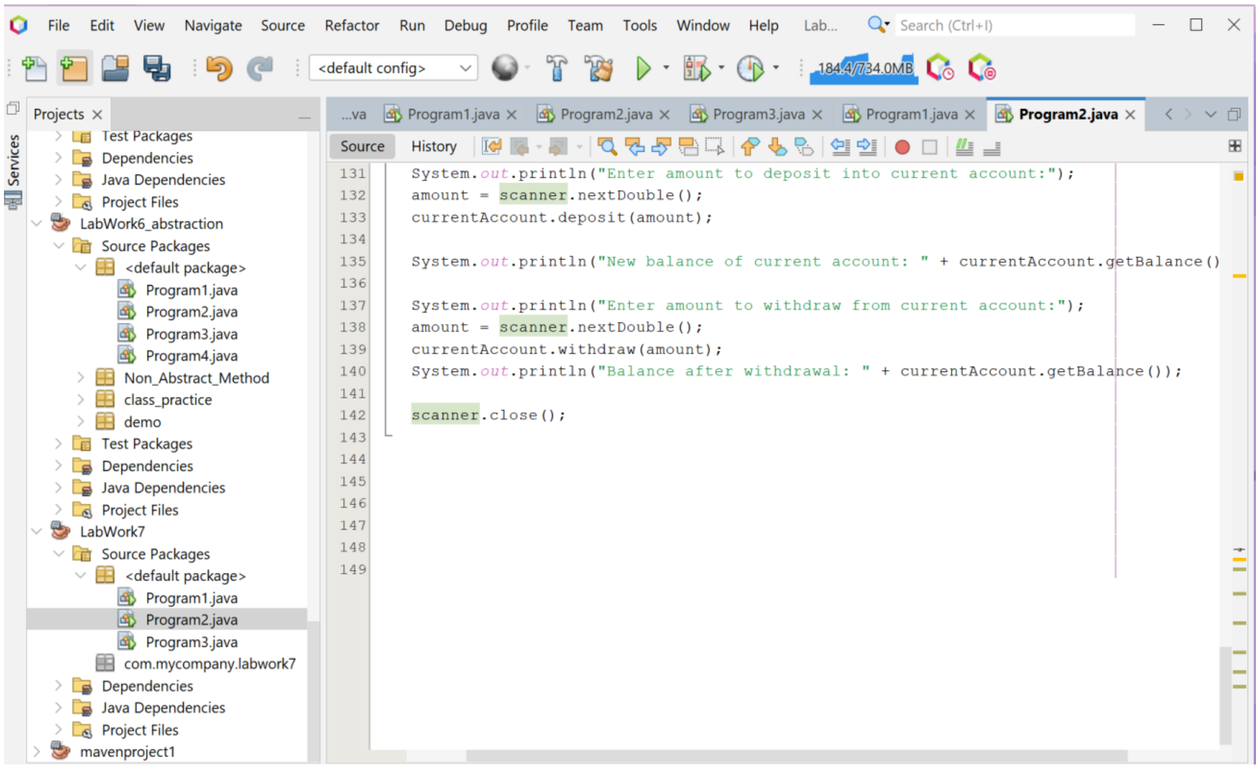
```
    static class CurrentAccount implements Account {
        private double balance;

        void checkMinimumBalance() {
            if (balance < 1000) {
                System.out.println("Balance is below the minimum threshold");
            }
        }

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

        @Override
        public void withdraw(double amount) {
            if (balance >= amount) {
                balance -= amount;
            } else {
                System.out.println("Insufficient balance");
            }
        }

```



```
        @Override
        public double calculateInterest() {
            return 0; // No interest for current accounts
        }

        @Override
        public double getBalance() {
            return balance;
        }
    }


    static class Bank {
        private Account account;

        void addAccount(Account account) {
            this.account = account;
        }


    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

```

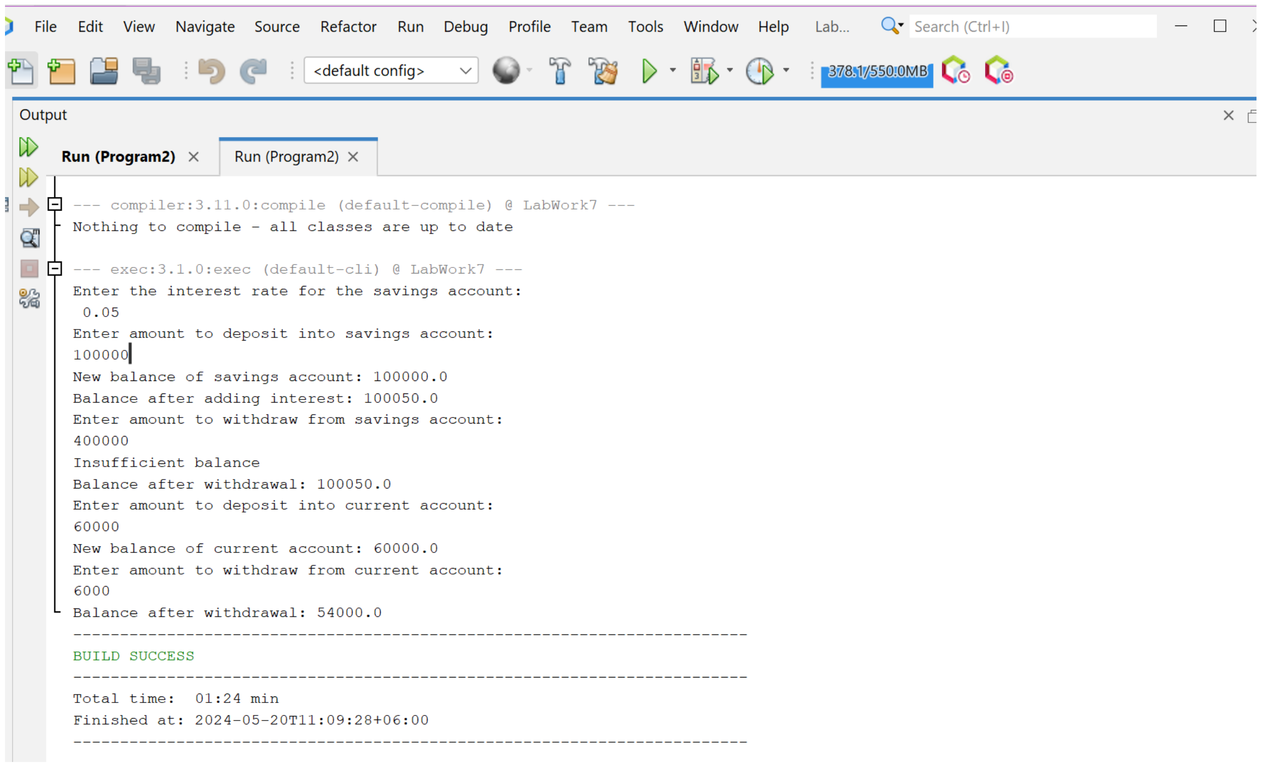


```
System.out.println("Enter the interest rate for the savings account:");
double interestRate = scanner.nextDouble();

SavingsAccount savingsAccount = new SavingsAccount(interestRate);
Bank bank = new Bank();
bank.addAccount(savingsAccount);

System.out.println("Enter amount to deposit into savings account:");
double amount = scanner.nextDouble();
savingsAccount.deposit(amount);

System.out.println("New balance of savings account: " + savingsAccount.getBalance()

savingsAccount.addInterest();
System.out.println("Balance after adding interest: " + savingsAccount.getBalance())

System.out.println("Enter amount to withdraw from savings account:");
amount = scanner.nextDouble();
savingsAccount.withdraw(amount);
System.out.println("Balance after withdrawal: " + savingsAccount.getBalance());

CurrentAccount currentAccount = new CurrentAccount();
bank.addAccount(currentAccount);

System.out.println("Enter amount to deposit into current account:");
amount = scanner.nextDouble();
currentAccount.deposit(amount);
```



```
System.out.println("Enter amount to deposit into current account:");
amount = scanner.nextDouble();
currentAccount.deposit(amount);

System.out.println("New balance of current account: " + currentAccount.getBalance()

System.out.println("Enter amount to withdraw from current account:");
amount = scanner.nextDouble();
currentAccount.withdraw(amount);
System.out.println("Balance after withdrawal: " + currentAccount.getBalance());

scanner.close();
```

Screenshot of Output screen/Run screen window:



```
--- compiler:3.11.0:compile (default-compile) @ LabWork7 ---
Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ LabWork7 ---
Enter the interest rate for the savings account:
 0.05
Enter amount to deposit into savings account:
100000
New balance of savings account: 100000.0
Balance after adding interest: 100050.0
Enter amount to withdraw from savings account:
400000
Insufficient balance
Balance after withdrawal: 100050.0
Enter amount to deposit into current account:
60000
New balance of current account: 60000.0
Enter amount to withdraw from current account:
6000
Balance after withdrawal: 54000.0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  01:24 min
Finished at: 2024-05-20T11:09:28+06:00
------------------------------------------------------------------------
```

Explanation :

The Java code implements a basic banking system with two types of accounts: Savings and Current, using an Account interface. The SavingsAccount class calculates interest based on the balance and interest rate, while the CurrentAccount class does not earn interest. The Bank class manages these accounts. In the main method, user input is used to perform operations like deposit, withdrawal, and interest calculation on these accounts.

Discussion:

While this code demonstrates the use of interfaces and class implementation in Java, it is a simplified version of a banking system. It lacks features like error checking for negative amounts, balance protection, data persistence, and handling multiple accounts. However, it serves as a good starting point for understanding object-oriented programming in Java and can be extended to more accurately represent a real banking system.

Problem 03: Write a Java program to create an interface Playable with a method play() that takes no arguments and returns void. Create three classes Football, Volleyball, and Basketball that implement the Playable interface and override the play() method to play the respective sports.

Objective:

The objective of the Program3 class is to demonstrate the use of interfaces in Java by defining a Playable interface and three classes (Football, Volleyball, Basketball) that implement this interface, each representing a different sport.

Algorithm:

1. Define an interface Playable with a single method play().
2. Define a class Football that implements the Playable interface. In the play() method, print "Playing football".
3. Define a class Volleyball that implements the Playable interface. In the play() method, print "Playing volleyball".
4. Define a class Basketball that implements the Playable interface. In the play() method, print "Playing basketball".
5. In the main method, create instances of Football, Volleyball, and Basketball, all referenced by Playable variables.
6. Call the play() method on each of the Playable instances.

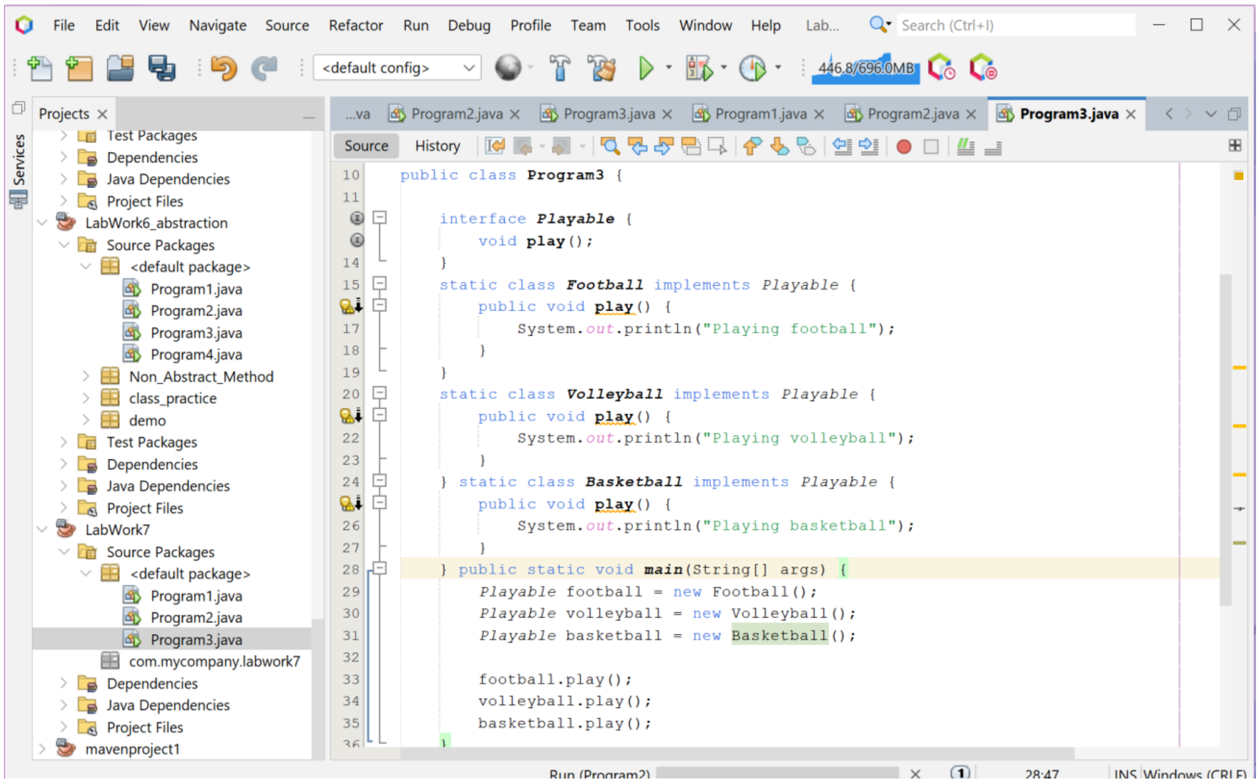
Code:

```
public class Program3 {
  interface Playable {
    void play();
  }
static class Football implements Playable {
    public void play() {
      System.out.println("Playing football");
    }
  }
  static class Volleyball implements Playable {
    public void play() {
      System.out.println("Playing volleyball");
    }
  }
  static class Basketball implements Playable {
    public void play() {
            System.out.println("Playing basketball");
        }
    }
    public static void main(String[] args) {
        Playable football = new Football();
        Playable volleyball = new Volleyball();
        Playable basketball = new Basketball();
        football.play();
        volleyball.play();
        basketball.play();
    }
}
```

Input/ Expected Output:

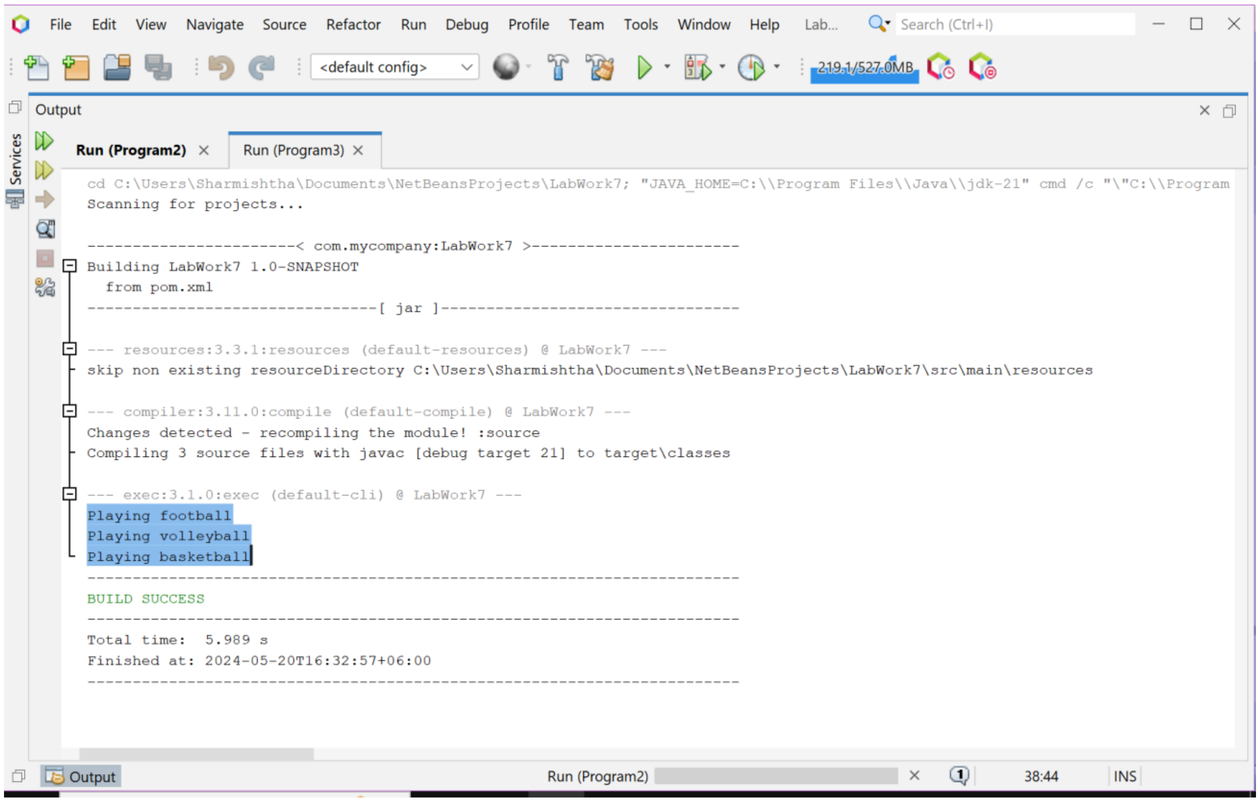
```
Playing football
Playing volleyball
Playing basketbal
```

Screenshot of code edition window:



```
public class Program3 {

    interface Playable {
        void play();
    }
    static class Football implements Playable {
        public void play() {
            System.out.println("Playing football");
        }
    }
    static class Volleyball implements Playable {
        public void play() {
            System.out.println("Playing volleyball");
        }
    } static class Basketball implements Playable {
        public void play() {
            System.out.println("Playing basketball");
        }
    } public static void main(String[] args) {
        Playable football = new Football();
        Playable volleyball = new Volleyball();
        Playable basketball = new Basketball();

        football.play();
        volleyball.play();
        basketball.play();
```

Screenshot of Output screen/Run screen window:



```
cd C:\Users\Sharmishtha\Documents\NetBeansProjects\LabWork7; "JAVA_HOME=C:\\Program Files\\Java\\jdk-21" cmd /c "\"C:\\Program
Scanning for projects...

-----------------------< com.mycompany:LabWork7 >-----------------------
Building LabWork7 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.1:resources (default-resources) @ LabWork7 ---
skip non existing resourceDirectory C:\Users\Sharmishtha\Documents\NetBeansProjects\LabWork7\src\main\resources

--- compiler:3.11.0:compile (default-compile) @ LabWork7 ---
Changes detected - recompiling the module! :source
Compiling 3 source files with javac [debug target 21] to target\classes

--- exec:3.1.0:exec (default-cli) @ LabWork7 ---
Playing football
Playing volleyball
Playing basketball
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  5.989 s
Finished at: 2024-05-20T16:32:57+06:00
------------------------------------------------------------------------
```

Explanation :

The Program3 class is a simple demonstration of polymorphism in Java using interfaces. An interface Playable is defined with a single method play (). Three classes Football, Volleyball, and Basketball are defined, each implementing the Playable interface and providing their own implementation of the play () method. This allows each class to define its own behavior for the play () method, demonstrating the concept of polymorphism where a single interface is used to represent common behavior across different classes.

Discussion:

While this code is a simple demonstration of interfaces and polymorphism in Java, it's a powerful concept that forms the basis of much of Java's flexibility. By defining a common interface, we can write code that works with any class that implements that interface, regardless of what the class actually does. This allows us to write more flexible and reusable code. In this case, we could easily add more classes that implement the Playable interface, and the main method would be able to work with them without any changes. This demonstrates the power of interfaces and polymorphism in object-oriented programming.