WIA1002 DATA STRUCTURE
WEEK 3&4 LAB

1. The GeometricObject, Circle, Rectangle classes

Design a class named GeometricObject. The class contains:

- A private String data field named color
- A private boolean data field named filled
- A no-arg constructor that creates a default GeometricObject.
- A constructor that creates a GeometricObject with the specified color, and filled condition
- Getter and setter for both data fields
- A toString method that returns a string description of the GeometricObject

Design a class named Circle that extends GeometricObject. The class contains:

- A private double data field named radius containing the default value of 1
- A no-arg constructor that creates a default Circle.
  - public Circle()
- A constructor that creates a GeometricObject with the specified radius
  - public Circle(double radius)
- A constructor that creates a GeometricObject with the specified radius, color, and filled condition
  - public Circle(double radius, String color, boolean filled)
- Getter and setter for the data field radius
- A toString method that overrides the superclass and returns a string description of the Circle mentioning all the data fields of the Circle as a GeometricObject

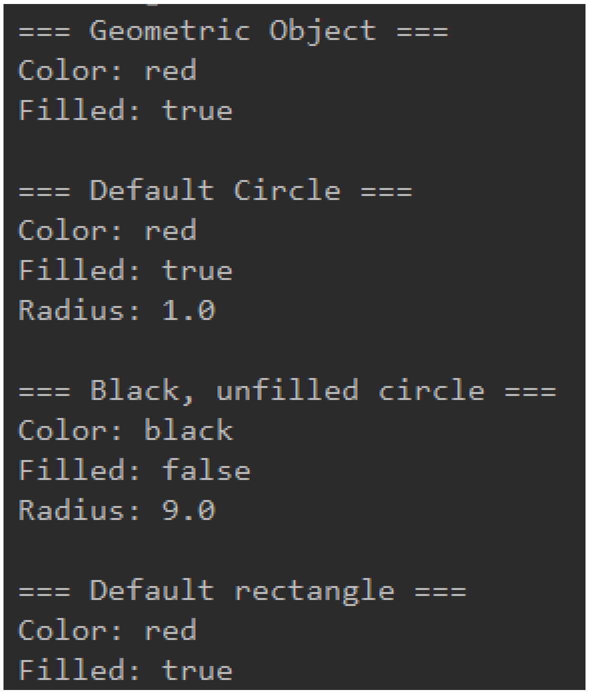
Design a class named Rectangle that extends GeometricObject. The class contains:

- A private data field named width containing the default value of 1
- A private data field named height containing the default value of 1
- A no-arg constructor that creates a default Rectangle.
- A constructor that creates a GeometricObject with the specified width and height
  - public Rectangle(double width, double height)
- A constructor that creates a GeometricObject with the specified radius, height, color, and filled condition
  - public Rectangle(double width, double height, String color, boolean filled)
- Getter and setter for the data fields

Write a test program that:

- Creates a GeometricObject
- Creates a Circle
- Creates a Circle, set the color as black, filled as false, set radius as 9
- Creates a Rectangle

Print this out:



```
=== Geometric Object ===
Color: red
Filled: true

=== Default Circle ===
Color: red
Filled: true
Radius: 1.0

=== Black, unfilled circle ===
Color: black
Filled: false
Radius: 9.0

=== Default rectangle ===
Color: red
Filled: true
```

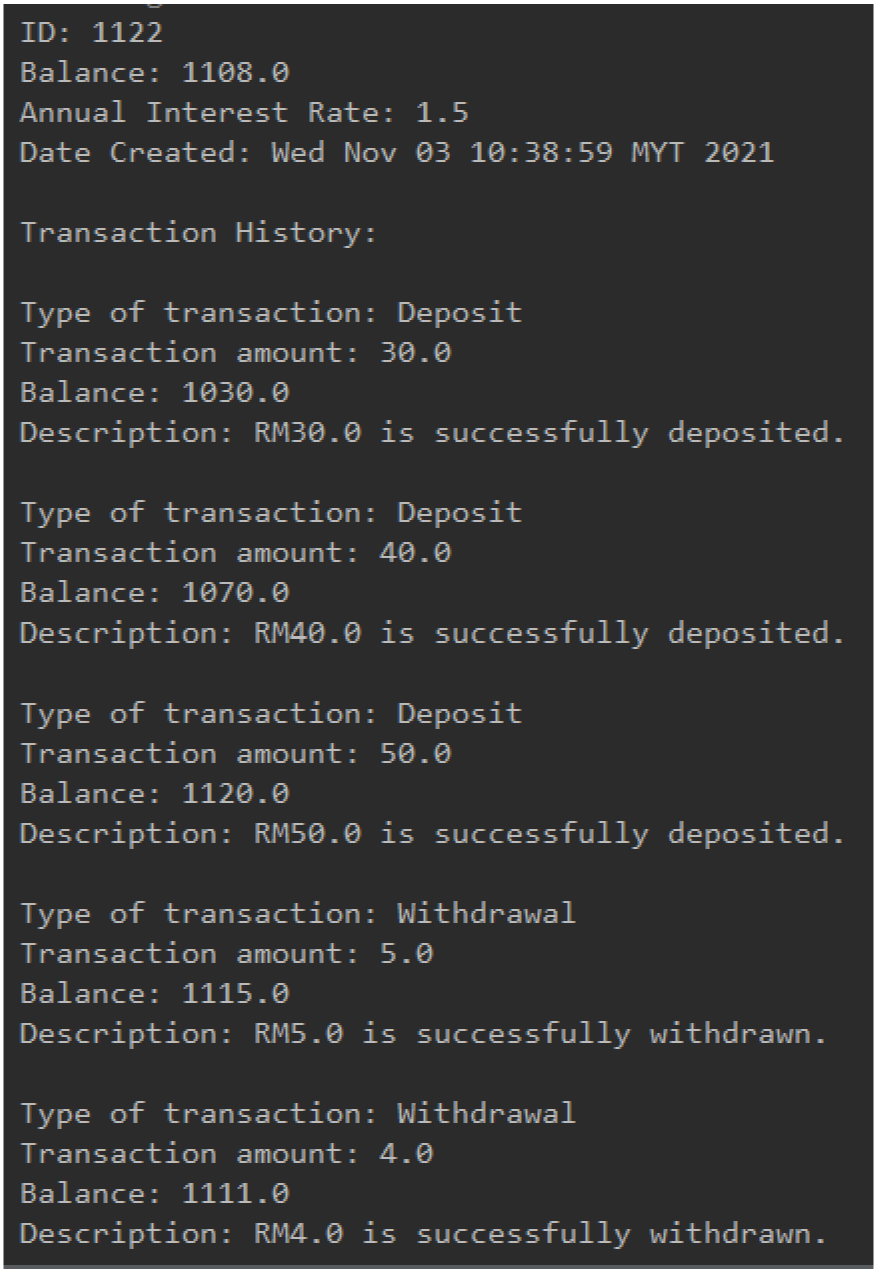
Notice that even though there is no toString method in the Rectangle class, the toString method is still working for Rectangle object, but the width and height will not be printed.

2. (Account class) An Account class was created in the previous lab exercise. Modify it and ensure that it has the following:

- Add a new data field named transactions whose type is **ArrayList** that stores the transaction for the accounts. You are allowed to use the ArrayList library. Each transaction is an instance of the Transaction class, which will be defined as:
    - (Transaction class)
    - A private char type data field named type to identify the transaction type, in which during the instantiation of the Transaction object, passing in ‘w’ stands for withdrawing while ‘d’ stands for depositing.
    - A private double type named amount
    - A private double type named balance
    - A private string type named description
    - A constructor that takes in type, amount, balance, and description
    - A toString method that return the String describing all details
- Modify the withdraw and deposit methods to add a *Transaction* object to the transactions array list.
- A toString method that returns all details about the account
- All other properties and methods are the same as in the previous exercise.

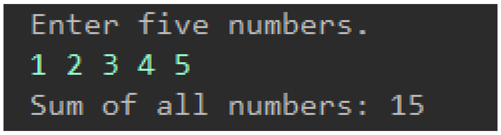
Write a test program that creates an Account with an annual interest rate of 1.5%, the balance of 1000, and id 1122. Deposit $30, $40, and $50 to the account and withdraw $5, $4, and $2 from the account.

Call the toString method and displays this:



```
ID: 1122
Balance: 1108.0
Annual Interest Rate: 1.5
Date Created: Wed Nov 03 10:38:59 MYT 2021

Transaction History:

Type of transaction: Deposit
Transaction amount: 30.0
Balance: 1030.0
Description: RM30.0 is successfully deposited.

Type of transaction: Deposit
Transaction amount: 40.0
Balance: 1070.0
Description: RM40.0 is successfully deposited.

Type of transaction: Deposit
Transaction amount: 50.0
Balance: 1120.0
Description: RM50.0 is successfully deposited.

Type of transaction: Withdrawal
Transaction amount: 5.0
Balance: 1115.0
Description: RM5.0 is successfully withdrawn.

Type of transaction: Withdrawal
Transaction amount: 4.0
Balance: 1111.0
Description: RM4.0 is successfully withdrawn.
```

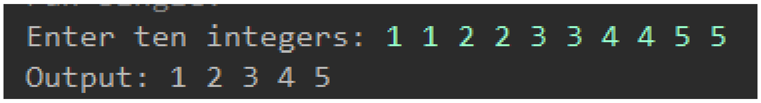
3. Write a test program that prompts the user to enter five numbers, stores them in an **ArrayList**, and displays their sum. You are allowed to use the ArrayList library.



```
Enter five numbers.
1 2 3 4 5
Sum of all numbers: 15
```

4. Write a test program that prompts the user to enter 10 integers into an **ArrayList**, removes the duplicate elements, and displays the distinct integers in their input order and separated by exactly one space. You are allowed to use the ArrayList library.

Here is a sample run:



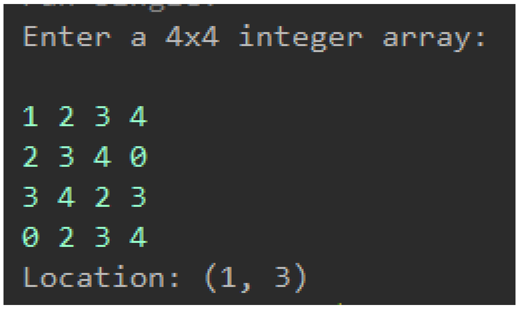
```
Enter ten integers: 1 1 2 2 3 3 4 4 5 5
Output: 1 2 3 4 5
```

5. (Locate the smallest element) Write a method to return the x and y indices of the smallest element in an array.

   public static int[] locateSmallest(int[][] a)

   The return value is a 1D array with two elements. These two elements indicate the row and column indices of the smallest element in the two-dimensional array. Write a test program that prompts the user to enter a 4x4 two-dimensional array and displays the location of the smallest element in the array.

   Here is a sample run:



```
Enter a 4x4 integer array:

1 2 3 4
2 3 4 0
3 4 2 3
0 2 3 4
Location: (1, 3)
```

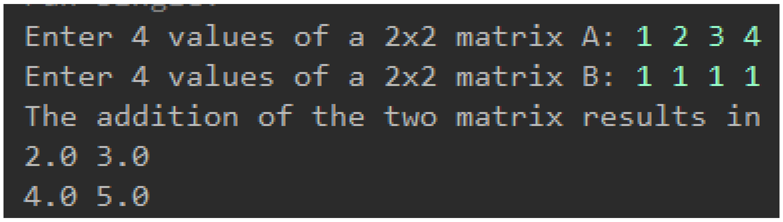
6. (Algebra: add two matrices) Write a method to add two **Array** matrices. The header of the method is as follows:

   public static double[][] addMatrix(double[][] a, double[][] b)

   In order to be added, the two matrices must have the same dimensions and the same or compatible types of elements. Let c be the resulting matrix. Each element cij is aij + bij. For example, for two 2 * 2 matrices a and b, c is

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

   Write a test program that prompts the user to enter two 2 * 2 matrices and displays their sum. Here is a sample run:

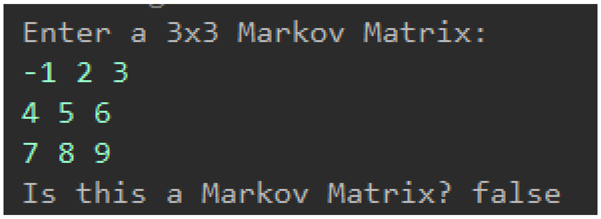


```
Enter 4 values of a 2x2 matrix A: 1 2 3 4
Enter 4 values of a 2x2 matrix B: 1 1 1 1
The addition of the two matrix results in
2.0 3.0
4.0 5.0
```

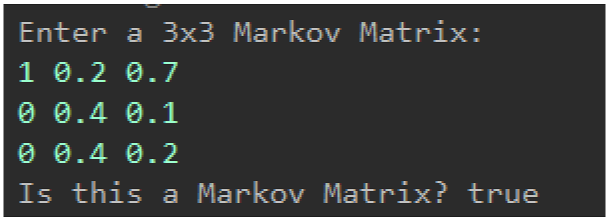
7. A Markov matrix is a matrix with all positive elements and the summation of each column is 1. Define a method as below to check if a 3x3 matrix is a Markov matrix or not.

   public static boolean isMarkovMatrix(double[][] m)

   Write a test program that prompts the user to enter a 3 * 3 matrix of double values and tests whether it is a Markov matrix. Here are sample runs:



```
Enter a 3x3 Markov Matrix:
-1 2 3
4 5 6
7 8 9
Is this a Markov Matrix? false
```

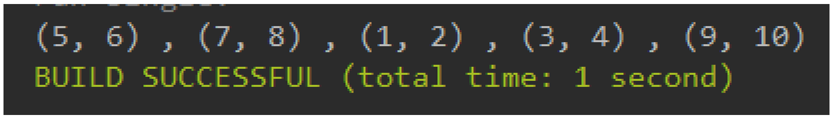


```
Enter a 3x3 Markov Matrix:
1 0.2 0.7
0 0.4 0.1
0 0.4 0.2
Is this a Markov Matrix? true
```

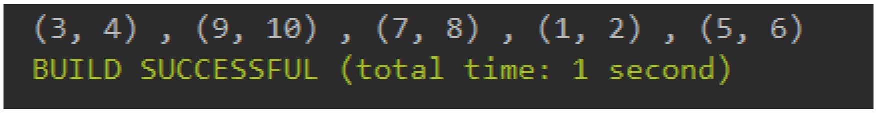
8. (Shuffle rows) Write a method to shuffle the row of an array as below:

   public static void shuffle(int[][] m)

   Write a test program that shuffles the following matrix:
   int[][] m = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};



```
(5, 6) , (7, 8) , (1, 2) , (3, 4) , (9, 10)
BUILD SUCCESSFUL (total time: 1 second)
```



```
(3, 4) , (9, 10) , (7, 8) , (1, 2) , (5, 6)
BUILD SUCCESSFUL (total time: 1 second)
```