CSCI 145 – Lab Final

Make sure to read all instructions before attempting this assignment. You cannot work with another student or communicate with anyone for this assignment. If you submitted an online solution, a solution on site such as Discord, or someone else solution, 0 will be given. If you posted/shared a solution and someone else uses it, you might get a 0 as well. You will only need to submit 2 out of 3 problems.

Pick 2 out of 3 problems and revise them as specified. Submit only 2 problems or the last problem will not be graded. Submit one PDF file with code and any input/output. It is your responsibility to confirm that you submitted your file correctly, so it is best to view your submission to make sure it was submitted correctly before you leave.

1. One interesting application of two-dimensional arrays is special squares. **A special square is a square matrix in which values in each row and in each column are in non-descending order** (smallest to largest in all rows and columns but can be like 5 5 7 8). In this exercise you will write code to determine whether a square is a special square. Your program reads input for squares from a file named *specialData.txt* (need to create the data file) and tells whether each is a special square by printing yes or no. Note that the -1 at the bottom tells the test program to stop reading. You can write your program from scratch, but it is recommended that you use "Magic Squares" exercise from one PA as a guide.

   Sample input data for specialData.txt:

```
3
0 1 3
1 2 3
2 5 8
4
0 2 4 6
1 3 6 7
2 4 5 8
5 7 8 9
3
0 1 2
1 2 3
2 5 4
-1
```

   Sample output data for specialData.txt:

```
1 yes
2 no
3 no
```

   New requirements:

   - Need to confirm that each row has the right number elements (can be fewer or too many values). Output "invalid" for invalid amount of data.
   - Output the size of each square

   Sample input data for specialData1.txt:

```
3
0 1 3
1 2 3
2 5 8
4
0 2 4 6
1 3 6 7 9
2 4 5 8
5 7 8 9
3
0 1 2
1 2 3
2 5 4
3
0 1 2
1 2 3
5 4
-1
```

Sample output data for specialData1.txt:

```
1 3x3 yes
2 4x4 invalid
3 3x3 no
4 3x3 invalid
```

Copy/paste source code and input/output including your input file below:

2. Given the following recursive definition, implement a recursive method. You can assume that n and x are integers as well as n >= 0 and x != 0.

```
p(x, n) = 1 when n is 0
p(x, n) = p(x, n/2) ^ 2 when n is even
p(x, n) = x * p(x, n-1) when n is odd
```

Try this recursive method with the following two test cases:

```
result = p(2, 9);
// use System.out.println to output result
// result: 512

result = p(2, 4);
// use System.out.println to output result
// result: 16
```

New requirements:

- `p(x, n) = x * p(x, n/2)^2 when n is odd`
- Use a sentinel loop (0 and 0 as sentinel values) to run all test cases
- Keep track number of recursive calls and output number of calls for each case

Sample input and output:

```
Enter base and exponent: 2 9<Enter>
result: 512
calls: 5

Enter base and exponent: 3 4<Enter>
result: 81
calls: 4

Enter base and exponent: 5 0<Enter>
result: 1
calls: 1

Enter base and exponent: 0 0<Enter>
Done.
```

Copy/paste source code and input/output below:

source code:

```
package final_lab;

import java.util.Scanner;

public class Recursive {
	static int c;

    public static void main(String[] args) {
	int x;
	int n;
```

```
            Scanner scan = new Scanner(System.in);

            System.out.print("Enter base and exponent: ");
            x = scan.nextInt();
            n = scan.nextInt();

            while (x != 0 || n != 0) {
                  c = 0;
                  System.out.println("result: " + recursive(x, n));
                  System.out.println("calls: " + c + "\n");

                  System.out.print("Enter base and exponent: ");
                  x = scan.nextInt();
                  n = scan.nextInt();
            }
            scan.close();
      }

      public static int recursive(int x, int n) {
            ++c;
            if (n == 0)
                  return 1;
            return (n % 2 == 0) ? (int) Math.pow(recursive(x, n / 2),
2) : x * (int) Math.pow(recursive(x, n / 2), 2);
      }
}
```

input/output below:

```
Enter base and exponent: 2 9
result: 512
calls: 5

Enter base and exponent: 3 4
result: 81
calls: 4

Enter base and exponent: 5 0
result: 1
calls: 1

Enter base and exponent: 0 0
```

3. Define a Java class called **Changer** with the following methods (set up proper attributes and imagine this is for a money changer for quarters, dimes, and nickels). Pennies are not used, and we only deal with multiples of 5 cents.
   - a default constructor: takes no parameter; sets id to "Changer 1" and sets quarters, dimes, and nickels to 4.
   - an overload constructor: takes 4 parameters for id, quarters, dimes, and nickels; assume valid data.
   - getId: returns the id of this changer.
   - setId: takes a new id and replaces the current id.
   - balance: no parameter; returns the total amount in the changer as dollars and cents like 5.25.
   - makeChanges: takes an int parameter representing an amount (assume multiple of 5 and between 5 and 95); returns true if there are available coins to perform the transaction (maximize largest denomination and subtract each denomination as applicable); returns false otherwise (e.g., need 2 quarters and there are not enough quarters regardless availability of dimes and nickels) and do not modify coins in the changer.
   - toString: overrides ***toString*** method and returns applicable information.

   Set up a simple test driver to test your class. Run the following test cases:

```
Changer myChanger = new Changer();
System.out.println(myChanger);   // Changer 1, q = 4, d = 4, n = 4
System.out.println("amount: " + myChanger.balance());
                                 // amount: 1.60

boolean valid = myChanger.makeChanges(60);   // valid = true
System.out.println("valid: " + valid);       // valid: true
System.out.println(myChanger);   // Changer 1, q = 2, d = 3, n = 4

myChanger.setId("C1");
valid = myChanger.makeChanges(90);           // valid = false
System.out.println("valid: " + valid);       // valid: false
System.out.println(myChanger);   // C1, q = 2, d = 3, n = 4

Changer changer2 = new Changer("C2", 2, 1, 1);
System.out.println(changer2);    // output: C2, q = 2, d = 1, n = 1
valid = changer2.makeChanges(45);            // valid = false
System.out.println("valid: " + valid);       // valid: false
System.out.println(changer2);    // output: C2, q = 2, d = 1, n = 1
```

   New requirements:

   - Set up a derived class **NewChanger** from Changer class that has:
     - 2 constructors like Changer class
     - Override method makeChanges() to utilizes lower denominations if needed (e.g., need 2 quarters and there are not enough quarters so use available dimes and nickels if available). It also outputs the number of dispensed coins if applicable.

Run the following test cases:

```
// same test cases as before
Changer myChanger = new Changer();
System.out.println(myChanger);   // Changer 1, q = 4, d = 4, n = 4
System.out.println("amount: " + myChanger.balance());
                                 // amount: 1.60

boolean valid = myChanger.makeChanges(60);   // valid = true
System.out.println("valid: " + valid);       // valid: true
System.out.println(myChanger);   // Changer 1, q = 2, d = 3, n = 4

myChanger.setId("C1");
valid = myChanger.makeChanges(90);           // valid = false
System.out.println("valid: " + valid);       // valid: false
System.out.println(myChanger);   // C1, q = 2, d = 3, n = 4

Changer changer2 = new Changer("C2", 2, 1, 1);
System.out.println(changer2);    // output: C2, q = 2, d = 1, n = 1
valid = changer2.makeChanges(45);            // valid = false
System.out.println("valid: " + valid);       // valid: false
System.out.println(changer2);    // output: C2, q = 2, d = 1, n = 1

// add the following new test cases
Changer myNewChanger = new NewChanger();
System.out.println(myNewChanger);   // Changer 1, q = 4, d = 4, n = 4
System.out.println("amount: " + myNewChanger.balance());
                                 // amount: 1.60

boolean valid1 = myNewChanger.makeChanges(60);      // valid1 = true
// output: Dispense q = 2, d = 1, n = 0
System.out.println("valid1: " + valid1);            // valid1: true
System.out.println(myNewChanger);   // Changer 1, q = 2, d = 3, n = 4

myNewChanger.setId("C1");
valid1 = myNewChanger.makeChanges(90);              // valid1 = true
System.out.println("valid1: " + valid1);            // valid1: true
// output: Dispense q = 2, d = 3, n = 2
System.out.println(myNewChanger);   // C1, q = 0, d = 0, n = 2

Changer newChanger2 = new NewChanger("C2", 2, 1, 1);
System.out.println(newChanger2);    // output: C2, q = 2, d = 1, n = 1
valid1 = newChanger2.makeChanges(45);               // valid1 = false
// output: Unable to dispense coins.
System.out.println("valid1: " + valid1);            // valid1: false
System.out.println(newChanger2);    // output: C2, q = 2, d = 1, n = 1
```

Copy/paste source code and input/output below:

source code:

```
package final_lab;

public class NewChanger extends Changer {
	public NewChanger() {
		super();
	}

	public NewChanger(String newId, int quarterQty, int dimeQty,
int nickelQty) {
```

```
            super(newId, quarterQty, dimeQty, nickelQty);
      }

      public boolean makeChanges(int amount) {
            int quartersNeeded = 0;
            int dimesNeeded = 0;
            int nickelsNeeded = 0;

            while (quartersNeeded < quarters && amount >= 25) {
                  amount -= 25;
                  ++quartersNeeded;
            }
            while (dimesNeeded < dimes && amount >= 10) {
                  amount -= 10;
                  ++dimesNeeded;
            }
            while (nickelsNeeded < nickels && amount >= 5) {
                  amount -= 5;
                  ++nickelsNeeded;
            }

            if (amount > 0)
                  return false;

            quarters -= quartersNeeded;
            dimes -= dimesNeeded;
            nickels -= nickelsNeeded;
            System.out.println("Dispense q = " + quartersNeeded + ",
d = " + dimesNeeded + ", n = " + nickelsNeeded);
            return true;
      }
}

package final_lab;

import java.text.DecimalFormat;

public class Changer {
      String id;
      int quarters;
      int dimes;
      int nickels;

      public Changer() {
            id = "Changer 1";
            quarters = 4;
            dimes = 4;
            nickels = 4;
```

```
	}

	public Changer(String newId, int quarterQty, int dimeQty, int
nickelQty) {
		id = newId;
		quarters = quarterQty;
		dimes = dimeQty;
		nickels = nickelQty;
	}

	public String getId() {
		return id;
	}

	public void setId(String newId) {
		id = newId;
	}

	public String balance() {
		DecimalFormat decimal = new DecimalFormat("#.00");
		return decimal.format(0.25 * quarters + 0.10 * dimes +
0.05 * nickels);
	}

	public boolean makeChanges(int amount) {
		int quartersNeeded = 0;
		int dimesNeeded = 0;
		int nickelsNeeded = 0;

		quartersNeeded = amount / 25;
		if (quartersNeeded > quarters)
			return false;
		amount %= 25;

		dimesNeeded = amount / 10;
		if (dimesNeeded > dimes)
			return false;
		amount %= 10;

		nickelsNeeded = amount / 5;
		if (nickelsNeeded > nickels)
			return false;

		quarters -= quartersNeeded;
		dimes -= dimesNeeded;
		nickels -= nickelsNeeded;

		return true;
```

```
	}

	@Override
	public String toString() {
		return id + ", q = " + quarters + ", d = " + dimes + ", n
= " + nickels;
	}

}

package final_lab;

public class ChangerTest {

	public static void main(String[] args) {
		// same test cases as before
		Changer myChanger = new Changer();
		System.out.println(myChanger);   // Changer 1, q = 4, d =
4, n = 4
		System.out.println("amount: " + myChanger.balance());
                                    // amount: 1.60

		boolean valid = myChanger.makeChanges(60);   // valid =
true
		System.out.println("valid: " + valid);        // valid:
true
		System.out.println(myChanger);   // Changer 1, q = 2, d =
3, n = 4

		myChanger.setId("C1");
		valid = myChanger.makeChanges(90);            // valid =
false
		System.out.println("valid: " + valid);        // valid:
false
		System.out.println(myChanger);   // C1, q = 2, d = 3, n =
4
		System.out.println();

		Changer changer2 = new Changer("C2", 2, 1, 1);
		System.out.println(changer2);    // output: C2, q = 2, d
= 1, n = 1
		valid = changer2.makeChanges(45);             // valid =
false
		System.out.println("valid: " + valid);        // valid:
false
		System.out.println(changer2);    // output: C2, q = 2, d
= 1, n = 1
		System.out.println();
```

```
            // add the following new test cases
            Changer myNewChanger = new NewChanger();
            System.out.println(myNewChanger);   // Changer 1, q = 4,
d = 4, n = 4
            System.out.println("amount: " + myNewChanger.balance());
                                               // amount: 1.60

            boolean valid1 = myNewChanger.makeChanges(60);     //
valid1 = true
            // output: Dispense q = 2, d = 1, n = 0
            System.out.println("valid1: " + valid1);           //
valid1: true
            System.out.println(myNewChanger);   // Changer 1, q = 2,
d = 3, n = 4

            myNewChanger.setId("C1");
            valid1 = myNewChanger.makeChanges(90);        // valid1 =
true
            System.out.println("valid1: " + valid1);           //
valid1: true
            // output: Dispense q = 2, d = 3, n = 2
            System.out.println(myNewChanger);   // C1, q = 0, d = 0,
n = 2
            System.out.println();

            Changer newChanger2 = new NewChanger("C2", 2, 1, 1);
            System.out.println(newChanger2);    // output: C2, q = 2,
d = 1, n = 1
            valid1 = newChanger2.makeChanges(45);         // valid1 =
false
            // output: Unable to dispense coins.
            System.out.println("valid1: " + valid1);           //
valid1: false
            System.out.println(newChanger2);    // output: C2, q = 2,
d = 1, n = 1

      }

}
```

input/output below:

```
Changer 1, q = 4, d = 4, n = 4
```

```
amount: 1.60
valid: true
Changer 1, q = 2, d = 3, n = 4
valid: false
C1, q = 2, d = 3, n = 4

C2, q = 2, d = 1, n = 1
valid: false
C2, q = 2, d = 1, n = 1

Changer 1, q = 4, d = 4, n = 4
amount: 1.60
Dispense q = 2, d = 1, n = 0
valid1: true
Changer 1, q = 2, d = 3, n = 4
Dispense q = 2, d = 3, n = 2
valid1: true
C1, q = 0, d = 0, n = 2

C2, q = 2, d = 1, n = 1
valid1: false
C2, q = 2, d = 1, n = 1
```