THIS IS JUST A PRACTICE EXAM. DO NOT SUBMIT TO SUBMIT SERVER

Specifications

For this assignment you will implement methods for the classes **MultipleChoice**, **ArrayUtilities**, and **Cashier**. You will find the classes in the **sysImplementation** package. A description of each method is provided below. A driver (that you can ignore if you know what to implement) and associated output is provided at the end. This driver is not part of the code distribution. Regarding the code you need to implement:

- You don't need to add comments to your code, but you must have good variable names, indentation and you should avoid code duplication.
- At this point you may want to take a look at the classes you will find in sysImplementation. We have provided a shell for each method you need to implement. You can also see the instance variables associated with each class.
- During the implementation of the above classes, you can add instance variables, constants (static final) and private methods.
- You can assume parameters are valid, unless we indicate otherwise (e.g., if the parameter is null throw …).
- You can create a StringBuffer out of another (e.g., new StringBuffer(anotherStringBuffer)) or by using a string (e.g., new StringBuffer(aString)).
- You can add toString() methods to the classes, but you are not required to do so. **We have provided a toString() method for the Cashier class; do not modify it, otherwise you will not pass release/secret tests.**
- You can use the compareTo method of the String class.
- You must provide an implementation for every method, otherwise your code will not work in the submit server. If you don't know what to do for a method, you can leave the following statement in the provided method shell:

throw new UnsupportedOperationException("Not Implemented");

MultipleChoice Class Specification

This class provides multiple choice questions, where each method represents a question. To answer a question, replace the statement

throw new UnsupportedOperationException("Not Implemented");

with a return statement that returns a character. The possible choices for characters are 'a', 'b', 'c', and 'd'. The following is an example of the kind of question we will provide, and the answer you need to provide:

/* What we will provide */

```
public static char question1() {
        /* Question #1

           This course is:

           a. cmsc106
           b. cmsc131
           c. cmsc200
           d. None of the above


        */
        throw new UnsupportedOperationException("Not Implemented");

}
```

/* You will replace the throw statement with a return and a character */

```
public static char question1() {
        /* Question #1

           This course is:

           a. cmsc106
           b. cmsc131
           c. cmsc200
           d. None of the above


        */
        return 'b';
}
```

There is a release test for the multiple choice problem. If you pass the release test it does not mean you got the right answers; it means you have provided an answer for each question 😊.

ArrayUtilities Class Specification

This class defines two methods: **getDigits** and **getSubArray**.

1. **getDigits** - The method's prototype is provided below. The method initializes the **digits** output parameter with digits (if any) found in the **data** string. The method returns the number of digits found. You can use the method **Character.isDigit()** that takes a character and returns true if it represents a digit and false otherwise. If the string parameter is null or it is the empty string, the method will return 0 and perform no further processing. You can assume the **digits** parameter is large enough to fit the answer. This method does not throw any exception.

   ```
   public static int getDigits(String data, char[] digits)
   ```

2. **getSubArray** - The method's prototype is provided below. The method returns a new array with elements of the **src** array that exists between the index **startIndex** (inclusive) and **endIndex** (inclusive). Remember that the index of the first array entry is 0, the index of the second entry is 1, etc. The method will throw an IllegalArgumentException (with any message) if **src** is null, if **src** has no elements, or if **endIndex** is less than **startIndex**.

   ```
   public static int[] getSubArray(int[] src, int startIndex, int endIndex)
   ```

Cashier Class Specification

The **Cashier** class represents a supermarket cashier. A cashier has a name (**name** instance variable), a total amount of money collected (**total** instance variable) after processing a number of items, a number of items processed (**numberOfItems** instance variable), and a description of each item that was processed (**allItemsScanned** instance variable). The declaration of each variable follows.

```
private String name;
private int total, numberOfItems;
private StringBuffer allItemsScanned;
```

The class methods are:

1. **Constructor** - Takes a string as parameter. The parameter represents the cashier's name. It will initialize instance variables accordingly. The method assigns a StringBuffer object to the **allItemsScanned** instance variable. The method will throw an IllegalArgumentException (with any message) if the parameter is null or its length is less than 3.
2. **Copy Constructor** - Creates a copy where changes to the copy will not affect the original.
3. **processItem** - Takes as parameters a string representing an item, and an integer representing its price. It will append the string to the **allItemsScanned** StringBuffer and will increase the number of items. No separator will be added (e.g., a comma). The **total** instance variable must be adjusted accordingly. The method will not modify the instance variables if the string parameter is null, if the string parameter is the empty string or if the price is negative. The method always returns a reference to the current object. No exception will be thrown by this method.
4. **getName** - get method for **name**.
5. **getTotal** - get method for **total**.
6. **getNumberOfItems** - get method for **numberOfItems**.
7. **getAllItemsScanned** - get method for **allItemsScanned**. **You must avoid privacy leaks.**
8. **compareTo** - It has a Cashier as parameter and returns an integer. This method will return:
   a. A negative value if the **total** value of the current object is less than the parameter.
   b. A positive value if the **total** value of the current object is greater than the parameter.
   c. If the current object and the parameter have the same value for **total**, the method will return:
      i. A negative value if the number of items of the current object is greater than the parameter.
      ii. A positive value if the number of items of the current object is less than the parameter.
      iii. 0 otherwise.

Driver / Expected Output (Feel free to ignore)

Driver

```
public static void main(String[] args) {
   Cashier cashier = new Cashier("Lindsay");
   cashier.processItem("coffee", 10);
   cashier.processItem("milk", 7);

   String answer = cashier.getName();
   answer += "\nTotal: " + cashier.getTotal();
   answer += "\nNumberOfItems: " + cashier.getNumberOfItems();
   answer += "\nAllItemsAScanned: " + cashier.getAllItemsScanned();
   answer += "\n===============================";

   String data = "a7b4";
   char[] digits = new char[data.length()];
   int found = ArrayUtilities.getDigits(data, digits);
   answer += "\nNumber of digits: " + found;
   answer += "\nDigits: " + digits[0] + ", " + digits[1];
   answer += "\n===============================";

   int[] src = { 6, 3, 19, 7 };
   int startIndex = 1, endIndex = 2;
   int[] result = ArrayUtilities.getSubArray(src, startIndex, endIndex);
   /* Arrays.toString() --> returns a string for the argument */
   answer += "\nSubArray: " + Arrays.toString(result);

   System.out.println(answer);
}
```

Output

```
Lindsay
Total: 17
NumberOfItems: 2
AllItemsAScanned: coffeemilk
===============================
Number of digits: 2
Digits: 7, 4
===============================
SubArray: [3, 19]
```