Object Oriented Programming TDA547

Krasimir Angelov, tel. 070-234-24-78

2015-03-21

The total number of points is 40. 20 points certainly guarantee a pass. 27p correspond to grade 4 and 32p to grade 5.

No other help materials except an English Dictionary are allowed. Write clean and readable Java code. Trivial syntax errors will be tolerated without affecting the grades. You don't have to comment your code unless if you really want to.

1. Read the following program:

```
public class Question1 {
    public static void decode(int d, int[] a) {
        for (int i = 0; i < a.length; i++) {
            a[i] += d;
            d++;
        }
    }

    public static void main(String[] args) {
        int d = 0;
        int[] a = {2, -1, -1, 2};
        decode(d,a);
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i]-d);
        }
        System.out.println();
    }
}
```

   What will the program print when it is executed? (4p)

2. In this task we do simple array processing:

- A monotonically increasing sequence is a sequence of numbers where each number in the sequence is greater or equal to its predecessor. Implement the static method:

  ```
  public static boolean isIncreasing(int[] a)
  ```

  which returns true if the array a contains an increasing sequence. (4p)
- Implement a class `Question2` which can be used to test whether a sequence is increasing or not. The class should be possible to run like this:

  ```
  > java Question2 1 2 3 3
  true
  ```

  i.e. it takes the sequence of numbers from the command line arguments and prints `true` or `false`. (4p)

3. Every time when we throw a dice we get a random number for 1 to 6. A fair dice should roll to any of its sides with an equal probability.

   - Imagine a program that can be used to check whether a dice is fair. It would need a model class with the following methods:
     - a method to register a new roll of the dice:
       `public void addRoll(int roll)`
       Here `roll` is a number between 1 and 6.
     - a method to compute the frequence of all possible outcomes:
       `public double[] getFrequences()`
       The frequence of an outcome is the number of times the corresponding number rolled up divided by the total number of dice rolls. The returned array should contain one frequency for each possible outcome.
     - a method to estimate whether the dice is fair:
       `public boolean isFair(double epsilon)`
       A dice is fair if the frequences for each outcome is in the range $1/6 \pm$ epsilon.

     Implement a class called `DiceStats` with the above methods. (8p)
   - Write a main method which can be used for testing. The main method should simulate 10000 random dice rolls and after that print the frequence for each outcome. At the end the program should print `true` if the dice is fair with `epsilon=0.01` or false otherwise. (4p)

4. Sample values from an experiment often need to be smoothed out. One simple approach is to replace each value in an array with the average of the value and its two neighbouring values (or only one neightbouring value if it is at either end of the array). Implement a static method:

   ```
   public static void smooth(double[] values)
   ```

   which smoothes the values. For example if values is `{1,4,2,5}` then after smoothing we should get `{2.5, 2.33, 3.66, 3.5}` because:

   ```
   2.5  = (1+4)/2
   2.33 = (1+4+2)/3
   3.66 = (4+2+5)/3
   3.5  = (2+5)/2
   ```

   You should not create another array in your solution. (8p)

   Hint: Every time when you update an element in values, the original number will be needed for computing the next element. Save the number in a temporary variable and use it in the next iteration.

5. Searching for information is the most common use of computers. This question is a simple example for searching. Implement the following method:

   ```
   public static List getNumbers(File phonebook, String name)
             throws FileNotFoundException
   ```

   The method takes as arguments the name of a file containing a phonebook, and a person name. The result is the list of all phone numbers stored for this person. The phonebook is a simple text file which looks like this:

   ```
   Mary 01-22-33
   John 12-11-45
   Carl 77-22-31
   Mary 21-02-33
   ```

   i.e. a sequence of lines with a name followed by a number. In order to make things simpler we assume that both the name and the number are a strings without a spaces. This means that you can parse the file by using a Scanner and the methods next/hasNext (8p)

   (check the cheat sheet for the methods of List and Scanner).