© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham
22 October 2019

Worksheet 3

MSc/ICY Software Workshop

Assessed Worksheet: 3% of the module mark (5% for the 20cr version).
Submission Deadline is Tuesday, 5 Nov 2019, at 12:00 noon via Canvas.
Follow the submissions guidelines on Canvas. JavaDoc comments are mandatory.
All submissions must pass the tests provided on 29 Oct. For Exercises 1 – 3 submit own tests.

Exercise 1: (Basic, 30%) Assume you have a non-empty array of double values `double[] temperatures = new double[365];` that stores for the 365 days of a year the temperatures measured at a weather station. Write a class `Temperature` with a method `public static int coldest(double[] temperatures)` that finds the first day of the year with the lowest temperature.
Hints:

- You may want to first find the lowest temperature.
- The day is to be given as an integer with `1` standing for the first day of the year, the 1st of January, and `365` for the last day, 31st of December.
- We assume that the year is not a leap year.

Example: If we have an array with the entries {`-10,0,0,0,0,...,0`}, that is, except for the first entry, `-10`, all other entries are `0`, then the method should return `1`, since the lowest temperature is `-10`, and the first day when it is measured is the first day of the year. For {`0,-10,-10,0,0,...,0`}, the method should return `2`, since again the lowest temperature is `-10`, and the second day of the year is the first day with the lowest temperature.

Exercise 2: (Medium, 30%) A company stores the salaries of its employees in an `ArrayList<double[]> allSalaries`, an `ArrayList` of arrays of type `double`. Each entry in `allSalaries` is an array of the 12 salaries of an employee for the 12 months of the year.
Write a class `Salaries` with the constructor

- `public Salaries()` having no parameters to create an initially empty `ArrayList`.

And write the following methods:

- `public void add(double[] employeeSalaries)` to add the salaries of one employee to the field variable `allSalaries`.
- The method `public static double average(double[] employeeSalaries)` computes the average salary for an employee. Note that any 0 entry should be disregarded, since a 0 means that the employee was not employed in that particular month. For instance, the average of {`1000,1000,2000,2000,0,0,0,0,0,0,0,0`} should be `1500.0` as the sum of the four non-zero values divided by 4.
If all values in the `employeeSalaries` array are zero, the method should throw an `IllegalArgumentException`.
- The method `public ArrayList<Double> averageSalaries()` generates an ArrayList storing the average salaries of all employees that have at least one non-zero monthly salary. Make use of the method `average`.
Hint: You need to catch possible exceptions thrown by the method `average`.
- The method `public boolean not3TimesHigher()` checks whether for each employee with at least one non-zero monthly salary their average salary is not higher than three times the overall average salary of the other employees. That is, you need here the average of the averages.

Exercise 3: (Advanced, 30%) Write a class `GenerateClass` to generate from the field variables a constructor, the setters, and the getters automatically just as `Eclipse` does. Assume three field variables:

- `String classname`, `String[] variableNames`, and `String[] variableTypes`, representing the class name of the class and the names and types of the field variables. The two arrays are of the same length.
- Write a constructor.
- Write a method `public String makeFields()` that returns a String with a declaration of the corresponding field variables.
- Write a method `public String makeConstructor()` that returns a String with a corresponding constructor.
- Write a method `public String makeGetters()` that returns a String with all getters in the order given by `variableNames/variableTypes`.
- Write a method `public String makeSetters()` that returns a String with all setters in the order given by `variableNames/variableTypes`.
- Write a method `public void writeFile()` that writes a rudimentary class, starting with the classname and the opening {, the field variables, the constructor, the getters, the setters, and the closing } to a file with the name given by the classname extended by `.java`. If the file cannot be written, the corresponding `IOException` is to be caught and an error message to be printed to the user. Indent always by two empty spaces.

Example: Let the variables be given as

- `String classname = "Person";`
- `String[] variableNames = {"name", "dob"};` // Date of Birth
- `String[] variableTypes = {"String", "Date"};`

Then `makeConstructor()` should return the String

```
"  private String name;
  private Date dob;
"
```

`makeConstructor()` should return the String

```
"  public Person(String name, Date dob) {
    this.name = name;
    this.dob = dob;
  }
"
```

`makeGetters()` should return the String

```
"  public String getName(){
    return name;
  }
  public Date getDob(){
    return dob;
  }
"
```

`makeSetters()` should return the String

```
"  public void setName(String name){
    this.name = name;
  }
  public void setDob(Date dob){
    this.dob = dob;
  }
"
```

A call to `writeFile()` should – if possible – write a file `Person.java` with all the output above together with the class header and the closing }. After copying over the `Date.java` class from the lecture in week 2, the `Person.java` class should compile.

Exercise 4: (Debugging, 10%) In order to keep track on the number of items in a storage facility somebody tries to build a robust interactive counting system as given in the class `Counting.java` given below. A user has 5 choices:

1 `add` a number to the counter,

2 `subtract` a number from the counter,

3 `show` the number stored in the counter,

4 `set` the counter to a new number, and

5 `exit` the program.

Unfortunately, the system does not behave as it should and crashes frequently. Submit the improved code with an assessment of the original code as a comment at the start. Your improved code should not crash under any input and behave as expected.

```
import java.util.*;
import java.util.regex.Pattern;
/**
 *  The class is used in order to interactively count a number of
 *  persons/items in a room, storeroom, etc. A user is able to select
 *  between five options:
 *  <pre>
 *  1. add to the counter,
 *  2. subtract from the counter,
 *  3. show the counter,
 *  4. set the counter to a new value, and
 *  5. exit the program.
 *  </pre>
 *  @author Manfred Kerber and Alexandros Evangelidis
 *  @version 2019-10-21
 */
public class Counter {
    /**
     * Pattern for the choice of input: either 1, 2, 3, 4, or 5
     */
    public static final Pattern p12345 = Pattern.compile("[12345]");
    /**
     *  Field variable to store the value of the counter.
     */
    private int counter;
    /**
     *  The constructor initializes the counter as 0. Starts a scanner
     *  to read from the command line, offering 5 choices for (add,
     *  subtract, show counter, set counter, and exit). It stays in a
     *  loop until the program is explicitly exited by entering 5.
     *  It has no parameters.
     */
    public Counter(){
        this.counter = 0;
        //System.in is like System.out, however, for input and not for output.
        Scanner s = new Scanner(System.in);
        /* can take value 1 (add)
         *                2 (subtract)
         *                3 (show counter)
         *                4 (set counter)
         *                5 (exit)
         */
        byte topChoice = 1; // Can be anything but 5 to enter the loop
        while (topChoice != 5) {
            System.out.println("Please enter:\n"+
                               "1 to add to the total\n" +
                               "2 to subtract from the total\n"+
                               "3 to show the total\n"+
                               "4 to set the total\n"+
                               "5 to exit the program");
            topChoice = (byte) Integer.parseInt(s.next(p12345));
            switch(topChoice) {
```

```
            case 1:  add(s);
            case 2:  subtract(s);
            case 3:  show();
            case 4:  set(s);
            case 5:  System.out.println("Finally there are "
                                        + counter + " items available.");
            default: throw new IllegalArgumentException();
            }
        }
    }
    /**
     *  The method reads in a number from the input and adds it to the counter.
     *  @param s The scanner from which the input is read.
     */
    public void add(Scanner s) {
        System.out.println("How much to add?");
        try {
            counter += Integer.parseInt(s.next());
        }
        catch (NumberFormatException e) {
            System.out.println("You need to enter an integer.");
            s = new Scanner(System.in);
        }
    }
    /**
     *  The method reads in a number from the input and subtracts it
     *  from the counter.
     *  @param s The scanner from which the input is read.
     */
    public void subtract(Scanner s) {
        System.out.println("How much to subtract?");
        try {
            counter -= Integer.parseInt(s.next());
        }
        catch (NumberFormatException e) {
            System.out.println("You need to enter an integer.");
            s = new Scanner(System.in);
        }
    }
    /**
     *  The method prints the current value of the counter.
     */
    public void show() {
        System.out.println("Currently there are " + counter +
                           " items available.");
    }
    /**
     *  The method reads in a number from the input and sets the
     *  counter to this value
     *  @param s The scanner from which the input is read.
     */
    public void set(Scanner s) {
        System.out.println("To which value do you want to set the counter?");
        try {
            counter = Integer.parseInt(s.next());
        }
        catch (NumberFormatException e) {
            System.out.println("You need to enter an integer.");
            s = new Scanner(System.in);
        }
    }
    public static void main(String[] args) {
        Counter parcels = new Counter();
    }
}
```