TEST

Software Systems: Programming

course code: 202001024
date: 20 January 2023
time: 8:45 - 11:45

General

- You may use the following (unmarked) materials when making this test:
  - Module manual.
  - Slides of the Programming topics.
  - The book
    David J. Eck. *Introduction to Programming Using Java*. Version 9.0, May 2022.
  - A dictionary of your choice.
  - IntelliJ and/or Eclipse.
  - Javadoc documentation of Java 11
    `https://docs.oracle.com/en/java/javase/11/docs/api/`

  You are *not* allowed to use any of the following:

  - Solutions of any exercises or old tests published on Canvas;
  - Your own materials (copies of (your) code, solutions of lab assignments, notes of any kind, etc.).
- When you are asked to write Java code, follow code conventions where they are applicable. Failure to do so may result in point deductions. It is recommended that you use IntelliJ and/or Eclipse to write code.
- You do *not* have to add Javadoc or comments, unless explicitly asked to do so. Invariants, preconditions and postconditions should be given only when they are explicitly asked.
- You are not allowed to leave the room during the first 30 minutes or the last 15 minutes of the exam.
- Place your student ID card on the table as well as documentation that grants extra time (if applicable).

Question 1 (20 points)

The questions in the rest of this exam are related to the scenario presented here.

A software company recently discovered that not all their code is great. Some of the code is good, some of it is bad, and some code is just ugly. You are given a task to implement software to record code quality scores of each programmer. An AI is used to classify code into the three categories: the good, the bad and the ugly. After classifying a piece of code of a programmer, the software will invoke the appropriate method of a class that you are going to develop.

Write an interface `CodeScores` that has three commands `good(String name)`, `bad(String name)`, and `ugly(String name)`. These commands are given as the parameter the name of a programmer who wrote some good, bad, or ugly code. Furthermore, add a query `getScore(String name)` that returns the current score of a programmer, or zero if the programmer has not been scored yet, and a query `getCount(String name)` that returns the number of times a programmer was scored, or zero if the programmer has not been scored yet. Finally, a query `getProgrammers()` should return all programmers in the system.

The rules for updating the scores are as follows:
- Good: add 4 points to the score
- Bad: divide the score by 2
- Ugly: subtract 7 points from the score

All implementations of `CodeScores` have to follow these rules.

a. *(5 pts)* Define the interface `CodeScores`, using appropriate keywords and return types.

b. *(6 pts)* Include appropriate Javadoc for the interface and all methods.

c. *(9 pts)* Write JML postconditions for all commands. Write postconditions that describe what changes and postconditions that describe what stays the same.

Question 2 (15 points)

a. *(10 pts)* Write a class `CodeScoresImpl` that implements the `CodeScores` interface. Give the full implementation as your answer. Javadoc and JML are not needed.
Hint: first think about what data your class needs to store, i.e., what fields will you need, and then implement the methods.

b. *(5 pts)* Sometimes the keywords **final** and/or **static** are used on fields of a class. Is it appropriate to use one or both of these keywords on the fields you defined in your implementation? Explain your answer for both keywords.

Question 3 (10 points)

It is important that the data collected by your `CodeScoresImpl` class can be stored in a file. Implement a method `void readFromFile(String filename)` that reads a file in the following file format:

```
50,5,Clint Eastwood
-5,13,Lee Van Cleef
30,20,Eli Wallach
```

Here, Lee has a score of -5 and has been scored 13 times.

You do not need to provide a method for writing to a file. You may assume files are correctly written following the file format. Your method should pass exceptions on to the caller. You may assume no scores exist yet in the `CodeScoresImpl` object.
Hint: you can use the `split` method of the `String` class.

Question 4 (15 points)

The company wants to fire the worst programmer, i.e., the programmer with the lowest score.

a. *(5 pts)* Write a method `getWorstProgrammer()` that returns the name of the worst programmer, or `null` if there is no programmer scored yet.

b. *(5 pts)* Add a postcondition to the `getWorstProgrammer` method that describes that the returned name (if not `null`) is the name of the worst programmer.

c. *(5 pts)* One of your friends bribes you to change the software so their name is never returned as the worst programmer. Instead, if your friend is the worst programmer, then the name of the second worst programmer should be returned. Modify the method `getWorstProgrammer` to accommodate their wishes. Store the name of the friend as a constant in the class. Finally, also modify the postcondition.

Question 5 (10 points)

a. *(5 pts)* If the person does not exist when invoking the `getScore` or `getCount` methods, an exception `ProgrammerNotFound` should be thrown. Create this exception and modify `getScore` and `getCount` to correctly throw the exception. Give the exception and the new implementations of those two methods as your answer.

b. *(5 pts)* Exceptions in Java can be "checked" and "unchecked". Explain what the difference is between checked and unchecked, explain whether the exception you made is checked or unchecked, and explain your design decision of a checked or unchecked exception.

Question 6 (10 points)

The company wants to use multiple AIs to classify code simultaneously, i.e., using multiple threads.

a. *(5 pts)* Consider that there are multiple threads of AIs scoring programmers. They will invoke methods of the same `CodeScores` object from different threads. Give a concrete example of what can go wrong, and in your answer, explicitly mention the invoked methods, the expected result and a possible wrong result, explaining how this result can occur.

b. *(5 pts)* Make your `CodeScoresImpl` implementation thread safe such that no race conditions can occur. Explain why the example in the previous question is no longer possible.

Question 7 (10 points)

The company wants to automatically fire anyone when they get a negative score. To do this, they run a separate thread that waits until a programmer has a negative score and then immediately fires them.

Add a method `String waitForNegative()` that returns the name of the next programmer that gets a negative score. This method will be called from a separate thread repeatedly. The method should not return if there is no programmer with a negative score (yet).

Give all changed methods and the new `waitForNegative` method as your answer.

Question 8 (10 points)

The company wants a website where the programmers of the company can see their current scores. For this, user authentication is required.

a. *(4 pts)* Describe two approaches that can be used for user authentication; one sentence per approach is enough.

b. Below is some Java code that has a vulnerability.

```
@RequestMapping("/register")
public void registerUser(@RequestBody UserInfo user) throws DatabaseException {
    try {
        // open a connection with the database
        database.openConnection();
        // store the given information of the user
        database.saveUser(user.email, user.password);
    } catch (DatabaseException ex) {
        // just throw it again, but ensure the connection is closed
        throw ex;
    } finally {
        // ensure the connection is closed even after exceptions
        database.closeConnection();
    }
}
```

The above code is an endpoint that directly receives an email address and a password from the request body, i.e., from the browser of the user, via a secure connection.

(6 pts) Identify a vulnerability and give a method to overcome this vulnerability.