TEST

Software Systems: Programming

course code: 201700117
date: 21 January 2019
time: 8:45 – 11:45

SOLUTIONS

General

- When doing this test, you may use the following (unmarked) materials :
  - the module manual;
  - the slides of the lectures;
  - a Java book of your preference, and
  - a dictionary.

  You may *not* use any of the following:
  - solutions of any exercises published on Canvas (such as recommended exercises or old tests);
  - 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. You do *not* have to add annotations or comments, unless explicitly asked to do so.
- No points will be deducted for minor syntax issues such as semicolons, braces and commas in written code, as long as the intended meaning can be made out from your answer.
- This test consists of 6 exercises for which a total of 100 points can be scored. The minimal number of points is zero. Your final grade of this test will be determined by the sum of points obtained for each exercise.
- Your grade for this test is used to calculate the final grade of the module. The formula used to calculate the final grade of the module can be found in the module manual.

Question 1 (15 points)

In this question, we consider sport games involving two teams, like football games, basketball games, volleyball games, etc. These games have in common that they involve a home and an away team, and have a score, but the structure of the score may vary. For example, in football and basketball, the score is a pair of goals and points, respectively, while in volleyball the score consists of at most 5 sets of pairs of points.

a. *(7 points)* Define a Java interface called `Game` to represent a game that has a home and an away team, a score and a boolean property indicating whether the game has been played or not. The home and away teams are represented as `String` values with their names, while the score is also an interface, to be defined in Question c. The `Game` interface should allow the home and away teams, the score and the 'played' boolean property to be queried. It should be also possible to set the 'played' boolean property to **`true`** through the interface. Define also a JML invariant stating that the home and away teams should always be different, and a postcondition for the method that sets the 'played' boolean property to true.

b. *(3 points)* Define an enumeration called `ScoreResult` with values `HOME` and `AWAY`, to indicate whether the home or away team has won the game, respectively, and `DRAW`, to indicate a draw.

c. *(2 points)* Define the `Score` interface, which has a single method `getResult()` and returns a `ScoreResult`.

Answer to question 1

a. *(4 points) This question is quite straightforward, but students may forget to define some methods; −3 if attributes are defined; −5 if method bodies are defined; −2 for each method that is not declared; −2 for each wrong JML specification.*

```
/** Generic interface for games between two teams (home and away)
 */
public interface Game {

    //@ invariant !getHomeTeam().equals(getAwayTeam());

    /** Returns the name of home team
     */
    public String getHomeTeam();

    /** Returns the name of away team
     */
    public String getAwayTeam();

    /** Returns the score of the game
     */
    public Score getScore();

    /** Returns true if the game was finished (played)
     *  and false otherwise
     */
    public boolean isPlayed();

    /**
     * Sets the game to played
     */
    //@ ensures isPlayed();
    public void setPlayed();
```

```
}
```

b. *(2 points) This question is also straightforward and an error means directly 0 points.*

```
/** Enumeration to define score results
 *
 */
public enum ScoreResult {
    HOME, DRAW, AWAY;
}
```

c. *(2 points) This question is quite simple.* -3 *if attributes are defined;* -3 *if method body is defined but signature is correct.*

```
/** Generic interface for a game score
 *
 */
public interface Score {

    /** Returns the result, i.e., the winning team
     * (HOME, AWAY) or a draw (DRAW)
     */
    public ScoreResult getResult();

}
```

Question 2 (20 points)

a. *(12 points)* Program class `FootballScore`, which implements your `Score` interface and represents the score of a football game. The class `FootballScore` should have the following methods:

- `getHomeGoals`, `getAwayGoals`, to query the number of goals scored by the home and away teams, respectively;
- `incHomeGoals`, `incAwayGoals`, to increment the goals scored by the home and away teams, respectively;
- `getResult()` that implements the `getResult()` of interface `Score` and returns the current result of this score.

Define a public JML invariant stating that the home and away goals should always be bigger or equal to 0, and define the JML postconditions of all the methods above. *The requested JML specifications should be as precise as possible!*

b. *(8 points)* Program class `FootballGame`, which implements your `Game` interface and represents a football game. This class should have a constructor
`public` `FootballGame (String homeTeam, String awayTeam)` that can be used to create an instance of a football game between `homeTeam` and `awayTeam`.

Answer to question 2

a. *(9 points) This class should have the attributes necessary, which are `homeGoals` and `awayGoals`. Method `getResult` determines the game result by comparing `homeGoals` with `awayGoals`. −3 for each missing attribute (`score` can alternatively refer to the `Score` interface instead of the `FootballScore` class, although this requires casting when the class is used); −3 for each missing getter and incrementer method; −1 for each wrong JML specification of the getters and incrementers; −4 if method `getResult()` is wrong; −3 if JML specification of method `getResult()` is wrong. Since there are some acceptable logical variants of the postcondition, this question should be graded carefully.*

```
public class FootballScore implements Score {

    private int homeGoals, awayGoals;

    //@ public invariant getHomeGoals() >= 0 && getAwayGoals() >= 0;

    //@ ensures \result >= 0;
    //@ pure
    public int getHomeGoals() {
        return homeGoals;
    }
    //@ ensures \result >= 0;
    //@ pure
    public int getAwayGoals() {
        return awayGoals;
    }

    /*@ ensures getHomeGoals() == \old(getHomeGoals()) + 1;
      */
    public void incHomeGoals() {
        homeGoals++;
    }

    /*@ ensures getAwayGoals() == \old(getAwayGoals()) + 1;
```

```
         */
        public void incAwayGoals() {
            awayGoals++;
        }

        /*@ ensures \result == HOME && getHomeGoals() > getAwayGoals() ||
                    \result == AWAY && getHomeGoals() < getAwayGoals() ||
                    \result == DRAW && getHomeGoals() == getAwayGoals();
         */
        //@ pure
        public ScoreResult getResult() {
            ScoreResult result = ScoreResult.DRAW;
            if ( homeGoals > awayGoals)
                result = ScoreResult.HOME;
            else if (homeGoals < awayGoals)
                result = ScoreResult.AWAY;
            return result;
        }

    }
```

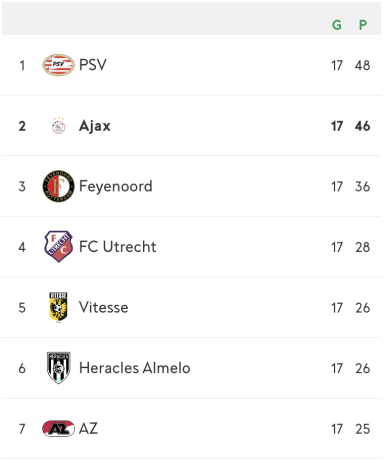
b. *(6 points) This class should also have the necessary attributes, which should be properly set in the constructor.* -2 *for each missing attribute;* -2 *or each missing method;* -4 *if constructor is missing or completely wrong;* -2 *if an instance of the Score interface is created with* `new Score()`*, instead of an instance of the* `FootballScore` *class.*

```
/** Implements the Game interface to define a Football game (match)
 *
 */
public class FootballGame implements Game {
    private String  homeTeam, awayTeam;
    private FootballScore score;
    private boolean played;

    /** Constructor
     *
     */
    public FootballGame (String homeTeam, String awayTeam) {
        this.homeTeam = homeTeam;
        this.awayTeam = awayTeam;
        this.played = false;
        this.score = new FootballScore();
    }

    /** Returns home team
     */
    @Override
    public String getHomeTeam() {
        return homeTeam;
    }

    /** Returns away team
     */
    @Override
    public String getAwayTeam() {
        return awayTeam;
```

```
        }

        /** Returns game score
         */
        @Override
        public FootballScore getScore() {
            return score;
        }

        /** Returns true if game has been played
         *  false otherwise
         */
        @Override
        public boolean isPlayed() {
            return played;
        }

        /** Returns sets the value of played to true
         */
        @Override
        public void setPlayed() {
            played = true;
        }

    }
```

Question 3 (30 points)

a. *(5 points)* Program a class `Standings` that keeps track of the standings of the competition. This class has an attribute `Map<String, int[]>` that stores only the *number of games played* and *number of points* of each team in an array of integer values[1], as illustrated in the figure below.



		G	P
1	PSV	17	48
2	Ajax	17	46
3	Feyenoord	17	36
4	FC Utrecht	17	28
5	Vitesse	17	26
6	Heracles Almelo	17	26
7	AZ	17	25

Define a constructor that creates this attribute to be used afterwards. The other methods of this class will be programmed in the next exercises.

We assume that an object of this class has to be initialised with the competition team somehow before the competition starts. For the purpose of this exercise, we defined the following method, which should be called before the object can be further used:

```
/**
 * Initialises with some values to start
 *
 */
public void initialise() {
    int r1[] = { 0, 0 };
    standings.put("AJX", r1);
    int r2[] = { 0, 0 };
    standings.put("FEY", r2);
    int r3[] = { 0, 0 };
    standings.put("PSV", r3);
    int r4[] = { 0, 0 };
    standings.put("TWE", r4);
}
```

b. *(5 points)* Program a method `getPoints(String team)`, which returns the number of points of `team`. You can assume that `team` is known to the `Standings` object.

c. *(10 points)* Program a method `addGameResult(FootballGame game)`, which gets a game and adds it the standings in case the game is set to 'played'. The winner of the game gets three points and the loser gets no points, while each team gets one point in the case of a draw. You can also assume that the home and away teams are known to the `Standings` object. Give also the part of the JML specification that holds when the game result is a `DRAW`.

[1] In a more realistic program other information would be stored, like number of wins, losses and draws, scored goals, goals against, etc. We limited this information in this exercise to keep it simple.

d. *(10 points)* Program a method `getRanking()`, which returns a `List<String>` with the standings in the competition so far, in which the team leading the competition is in the first position of the list, and so on. In this exercise, the number of points determines the ranking, and in case of equal points, the lexical order of the team names is used as tie-break, such that the names with the lowest letters have precedence (e.g., `"AJX"` $<$ `"FEY"`, so `"AJX"` comes before `"FEY"`). To determine the lexical order of team names you can use the method
`public int` `compareTo(String arg)` of class `String`, which returns value 0 if `arg` is equal to this `String`, a value smaller than 0 if this `String` is lexicographically less than `arg`; and a value greater than 0 if this `String` is lexicographically greater than `arg`.

Answer to question 3

a. *(5 points) This question is about the declaration and constructor of the class.* -3 *if attribute is not properly declared;* -2 *if attribute is not initialised with a* `HashMap`*.*

b. *(5 points) This is quite straightforward.* -2 *if method signature is wrong;* -2 *if* `Map` *attribute is not used; max* -2 *if something is wrong when getting the value.*

c. *(10 points) This question is a bit more intricate and the grading instructions have been defined in a positive way.* 1 *for a postcondition stating that* `game.isPlayed()` *is required or an* **`if`** *statement in the code;* 3 *for the JML postcondition (only the clause concerning a* `DRAW` *has been asked, so only the part starting with* `game.getResult() == DRAW && ...;` 1 *for method signature;* 1 *for using* `getScore()` *to get the* `FootballScore` *object (attribute);* 1 *if results are distinguished;* 1 *if correct points are given;* 1 *if points are taken correctly from the* `Map` *attribute;* 1 *for correctly updating the number of played games.*

d. *(10 points) This question is also intricate and the grading instructions have been defined in a positive way.* 1 *for method signature;* 1 *for properly creating a result list;* 3 *correct loop and insertion mechanism;* 2 *correct points comparison;* 2 *correct names comparison;* 1 *properly returning result.*

The complete class can be found below. The class contains more JML code than what has been asked in the questions.

```
/**
 * Class to keep football competition standings
 *
 */
public class Standings {

    private Map<String, int[]> standings;

    /**
     * Constructor creates an empty standings Map
     *
     */
    public Standings() {
        standings = new HashMap<String, int[]>();
    }
    /* Returns the points of the team in the competition
     *
     *
     */
    //@ requires contains(team);
    public int getPoints(String team) {
        return standings.get(team)[1];
```

```
    }

    /* Computes a game
     *
     */
    /*@ requires contains(game.getScore.getHomeTeam()) &&
            contains(game.getScore.getAwayTeam()) && game.isPlayed();
      @ ensures ( (game.getResult() == HOME) &&
            (getPoints(game.getHomeTeam()) ==
                \old(getPoints(game.getHomeTeam())) + 3) )
            || ( (game.getResult() == AWAY) &&
            (getPoints(game.getAwayTeam()) ==
                \old(getPoints(game.getAwayTeam())) + 3) )
            || ( (game.getResult() == DRAW) &&
            (getPoints(game.getHomeTeam()) ==
                \old(getPoints(game.getHomeTeam())) + 1) &&
            (getPoints(game.getAwayTeam()) ==
                \old(getPoints(game.getAwayTeam())) + 1) );
     */
    public void addGameResult(FootballGame game) {
        FootballScore score = game.getScore();
            if (score.getResult() == ScoreResult.HOME) {
                int[] results = standings.get(game.getHomeTeam());
                results[0]++;
                results[1] = results[1] + 3;
                results = standings.get(game.getAwayTeam());
                results[0]++;
            } else if (score.getResult() == ScoreResult.AWAY) {
                int[] results = standings.get(game.getAwayTeam());
                results[0]++;
                results[1] = results[1] + 3;
                results = standings.get(game.getHomeTeam());
                results[0]++;
            } else {
                int[] results = standings.get(game.getHomeTeam());
                results[1] = results[1] + 1;
                results[0]++;
                results = standings.get(game.getAwayTeam());
                results[1] = results[1] + 1;
                results[0]++;
            }
    }
    /**
     * Gets the competition rankings
     *
     */
    public List<String> getRanking() {
        List<String> result = new ArrayList<String>();

        for (Entry<String, int[]> e : standings.entrySet()) {
            String eName = e.getKey();
            boolean found = false;
            for (int i = 0; i < result.size() && !found; i++) {
                String p = result.get((i));
                if (standings.get(p)[1] < standings.get(eName)[1] ||
                     ((standings.get(p)[1] == standings.get(eName)[1]) &&
```

```
                        p.compareTo(eName) > 0)) {
                    result.add(result.indexOf(p), eName);
                    found = true;
                }
            }
            if (!found)
                result.add(eName);
        }
        return result;
    }
```

Question 4 (15 points)

Method `getPoints()` assumes that the team is known to the `Standings` object, i.e., it has been defined when the `Standings` object has been initialised as shown in Question 3. The same actually holds for method `addGameResult()`, which assumes that the home and away teams are known to the `Standings` object. Furthermore a game should only be added to the standings when it is (completely) played.

a. *(3 points)* Define an exception class `TeamNotFound`, which will be used to indicate that a team was not found, and an exception class `GameNotPlayed`, which will be used to indicate that a game requested to be added is not set to 'played'. Reuse the existing mechanism to construct exceptions with dedicated messages.

b. *(2 points)* Implement a wrapping method `Standings.getPointsX` that throws a `TeamNotFound` if the team given as argument is not known and calls `Standings.getPoints` otherwise.

c. *(5 points)* Implement a wrapping method `Standings.addGameResultX` that throws a `TeamNotFound` exception if the home and away teams of the game to be added are not known, and a `GameNotPlayed` exception if the game to be added is not set to 'played', and calls `Standings.addGameResult` otherwise. The thrown exceptions should have meaningful error messages.

d. *(5 points)* Write a code fragment that calls method `addGameResultX`, handling each exception thrown by this method separately. When an exception is thrown, its message should be printed on the standard output.

Answer to question 4

a. *(3 points. The question is quite simple, so an error will soon give rise to 0 points. Small syntax errors should not be punished.)*

```
public class TeamNotFound extends Exception {
    public TeamNotFound(String message) {
        super(message);
    }
}

public class GameNotPlayed extends Exception {
    public GameNotPlayed(String message) {
        super(message);
    }
}
```

b. *(2 points.)* The question is quite simple, so an error will soon give rise to 0 points.

c. *(5 points.)* The question is quite simple, so an error will soon give rise to 0 points.; -2 for a missing **`throws`** declaration, up to -3 for each thrown exception.

```
public int getPointsX (String team) throws TeamNotFound {
    if (!standings.containsKey(team))
        throw new TeamNotFound("Team␣" + team + "not␣found!");
    return getPoints(team);
}

public void addGameResultX(FootballGame game)
        throws TeamNotFound, GameNotPlayed {
    String homeTeam = game.getHomeTeam();
    String awayTeam = game.getAwayTeam();

    if (!game.isPlayed())
        throw new GameNotPlayed ("Game␣" +
                homeTeam + "␣x␣" + awayTeam + "␣is␣not␣played");
    if (!standings.containsKey(homeTeam))
        throw new TeamNotFound("Team␣" + homeTeam + "␣not␣found!");
    if (!standings.containsKey(awayTeam))
        throw new TeamNotFound("Team␣" + awayTeam + "␣not␣found!");

    addGameResult(game);
}
```

d. *(5 points; up to -3 for errors in the **`try-catch`**-construct.)* Exceptions should be caught in separate catch statements.

```
FootballGame g1 = new FootballGame("TWE", "AJX");
// ... or get the game some other way
try {
    testStand.addGameResultX(g1);
} catch (GameNotPlayed e) {
    System.out.println(e.getMessage());
} catch (TeamNotFound e) {
    System.out.println(e.getMessage());
}
```

Question 5 (15 points)

Consider the class below, which observes and keeps track of the passes between two players.

```
public class Pass extends Thread {
    String p1, p2;
    Object obj;

    public Pass (String p1, String p2, Object obj) {
        this.p1 = p1;
        this.p2 = p2;
        this.obj = obj;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
                System.out.print("␣from␣" + p1);
                System.out.println ("␣to␣" + p2);
        }
    }
}
```

This class keeps a `Object obj` reference that in the beginning will not be used.

Suppose that in the `main` method, two `Pass` objects are created to model two different pairs of players passing the ball to each other, and their corresponding threads are spawned as follows:

```
    public static void main(String[] args) {
        Object obj = new Object();
        Pass obs1 = new Pass ("Messi", "Suarez", obj);
        Pass obs2 = new Pass ("Neymar", "Mbappe", obj);
        obs1.start();
        obs2.start();
    }
```

a. (*5 points*) What possible outcomes can be observed on the console? Which of them could be considered erroneous? Explain your answer.

b. (*5 points*) Suppose now that each `Pass` object created its own `Object` object in the `run` method and uses it in the **`for`** loop of the `run` method in the following way:

```
1    public void run() {
2        Object obj = new Object();
3        for (int i = 0; i < 10; i++) {
4            synchronized(obj) {
5                System.out.print("␣from␣" + p1);
6                System.out.println ("␣to␣" + p2);
7                    }
8        }
9    }
```

What possible outcomes can be observed on the console now? Does this change anything compared with your answer to Question a? Explain your answer.

c. (*5 points*) Now suppose that instead of creating separate `Object` objects, these `Pass` objects simply use the common `Object` object that they get with their constructor, i.e., line 2 in method `run()` of Question b is removed. What possible outcomes can be observed on the console now? Does this change anything compared with your answer to the former questions? Explain your answer.

Answer to question 5

a. The following lines can be printed out:

- "␣from␣Messi␣to␣Suarez"
- "␣from␣Neymar␣to␣Mbappe"
- "␣from␣Neymar␣from␣Messi␣to␣Suarez"
- "␣to␣Mbappe"
- "␣from␣Messi␣from␣Neymar␣to␣Mbappe"
- "␣to␣Suarez"

Also unlikely but in theory possible are the following lines:

- "␣from␣Messi␣from␣Neymar␣to␣Suarez"
- "␣from␣Neymar␣from␣Messi␣to␣Mbappe"

The strings "␣from␣Messi␣to␣Suarez" and "␣from␣Neymar␣to␣Mbappe", possibly interrupted in the ways above, are repeated 10 times each. This happens because a thread can be interrupted between the two `System.out.println()` method calls, so that the execution is passed to the other thread.
Both the expected results as the explanation should be correct!

b. Noting changes here because they use different objects to synchronise, which means that the threads can still be interrupted.
Both the expected results as the explanation should be correct!

c. The following lines will be printed out (10 times each):

- "␣from␣Messi␣to␣Suarez"
- "␣from␣Neymar␣to␣Mbappe"

In this case, the threads use the same object to synchronise, so that the two calls to `System.out.println()` will always happen after each other in a thread.
Both the expected results as the explanation should be correct!

Question 6 (15 points)

Consider an online system for managing all kinds of information surrounding sports teams, including for example schedules for training and matches, but also personal information about the members. Not all of this information should be public, and of course some security measures are put in place to make sure only the right people can access and modify the system.

a. *(3 points)* What are the three main security properties/attributes that, when violated for a (software) system, indicate there has been a security incident?

New users of this (online) team support-system are sent an initial password upon registration. After talking to some other users of the system, you notice that the initial passwords are always of the same length and only contain a very specific set of characters. Some example passwords you find are:

- `pw71761f3757`
- `pwc8acf1a5ba`
- `pw5e6adb15e1`

You wonder how these initial passwords are generated and decide to have a look at the source code. You find the following snippets of code:

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;

// (... More imports not relevant to answer the question ...)

public class InitialPassword {

    public static String generateInitialPassword(String email)
              throws NoSuchAlgorithmException {
        MessageDigest digest;
        digest = MessageDigest.getInstance("SHA-256");
        byte[] hashOutput = digest.digest(email.getBytes());
        byte[] firstBytes = Arrays.copyOfRange(hashOutput, 0, 5);
        return "pw" + Hex.encodeHexString(firstBytes);
    }
    // (... More code not relevant to answer the question ...)

}
```

b. *(4 points)* Please explain why, from a security perspective, such an approach for an initial password is a bad idea.

After you complained about this security-problem to the maintainer of the system, the approach for generating initial passwords was changed to something more suitable. You then move on to have a look at the password-reset functionality. Sometimes users forget their passwords and for this reason this online service has implemented a password-reset functionality. After filling in an e-mail address, users are sent a new password by email. You notice that these reset passwords are 8-9 characters long and have the following pattern:

- First, all passwords start with the text `"play"`, `"ball"`, or `"goal"`.
- After which come 2 characters from the set {`"o"`, `"c"`, `"r"`, `"s"`, `"e"`}.
- This is followed by two to three digits.

c. *(2 points)* If an attacker would try to brute-force access to an account with such a reset password, how many attempts would it at most take? Show the calculation.

Only using passwords for authentication has a number of downsides. Luckily, there are additional methods that can help. Authentication methods are typically subdivided into three categories, also called "authentication factors". The aforementioned password fits into the category, "something the user knows".

d. *(4 points)* What are the other two categories? Also provide at least one example for each of them.

e. *(2 points)* Group the following terms into two groups.

- MD5
- Hexadecimal
- Base64
- SHA-256

Answer to question 6

a. *(3 points)* Confidentiality, Integrity, Availability (one point each). If only "CIA": 1 point. If three properties are provided that seem close enough: 2 points. But you can be pretty strict here.

b. *(4 points)* While the output of a hash function may appear random, it is very predictable. Therefore, when the method for generating initial passwords is known (through guessing, open source code, leaks, etc.), one can simply compute the password for a specific email-address. So an attacker can simply try to login as someone else, and will succeed if that person hasn't changed their password (yet) from the one initially set. Full points if the students say something along those lines. Alternatively, if a students states that the number of possible passwords is too small (8^5), then that is also very true. However, the problem that the initial password can easily be computed by an attacker is a larger issue. Therefore 3 points in this case.

c. *(2 points)* $3 \times 5^2 \times (10^2 + 10^3) = 82500$ The actual answer is not that important (many people will not have brought a calculator). Full points if students produced this formula. No additional explanation is needed. Additional guidelines for grading:

- Swapping exponents and base: 0 points.
- $+$ instead of $\times$: 0 points
- 9 instead of 10: deduct 1 point

d. *(4 points)*

- Something the user *has*. [1 point] Examples: bank card, smart card, token, RSA token, yubikey, FIDO 2FA, ... [1 point]
- Something the user *is* (biometrics). [1 point].Examples: fingerprint scan, voice recognition, iris scan, hand recognition, typing recognition ... [1 point]

e. *(2 points)* The two groups are:

- Base64, Hexadecimal (method of encoding binary data)
- SHA-256, MD5 (cryptographic hash functions)

Only two points if these two groups are made.