CSE2DES/CSE5DES - 2019 - Assignment - Part 2

Updated on 09-10-2019

To supersede the previous handout

Due Date: 10.00 am, Wednesday 23rd October, 2019

The original handout was posted on the 25th of September. Unfortunately, the solution for Task 2 and Task 3 was also accidentally posted among the files provided – except for use case 6, Assign a Judge to a Session.

It is necessary, therefore, to make some changes. This new handout will replace the previous one

---

The changes to the requirements are outlined below:

- **Task 1**

  An additional use case is to be specified. This is the use case is to enter the scores that all nine judges award to a performance. We will refer to this as Use Case 8.

- **Task 2**

  The complete solution for use cases listed in Task 1, except for Use Case 8, has been provided. For this Task 2, effectively, you are only required to implement the postcondition of Use Case 8.

- **Task 3**

  The complete solution for use cases listed in Task 1, except for Use Case 6 and Use Case 8, has been provided. For this Task 3, effectively, you are only required to implement Use Case 6 and Use Case 8.

- **Task 4 and task 5**

  The requirements are unchanged.

The marks are also re-distributed:

- **Task 1: 45 marks**
- **Task 2: 12 marks**
- **Task 3: 26 marks**
- **Task 4: 17 marks**

Assessment: This part is worth 50% of the total assignment mark, i.e. it is worth 15% of the final marks.

Assignment Submission: An electronic copy of work, including those that are provided, is to be submitted electronically via latcs8 using the submit command:

```
submit DES <file name>
```

This is an individual assignment. You are not permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Computer Engineering treats plagiarism very seriously. When it is detected, penalties are strictly imposed. Students are referred to the Department of Computer Science and Computer Engineering's Handbook and policy documents with regard to plagiarism.

No extensions will be given: Penalties are applied to late assignments: 5 marks are deducted per day, accepted up to 5 working days after the due date only. If there are circumstances that prevent the assignment being submitted on time, an application for special consideration may be made. See the departmental Student Handbook for details. Note that delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime.

Return of Assignments: Students are referred to the departmental Student Handbook for details. It is planned that the assignments will be returned within three weeks of the due date.

Objectives: To specify system's behaviors in terms of atomic use cases, to implement the atomic use cases and test them.

For Part 2 of the assignment, you are to continue with the Figure Skating System described in Part 1. Whereas Part 1 is concerned with the analysis phase, Part 2 will be concerned with design, prototyping and testing.

As the starting point for Part 2, a structural design model, consisting of three class diagrams, is given in Figures 1-3.

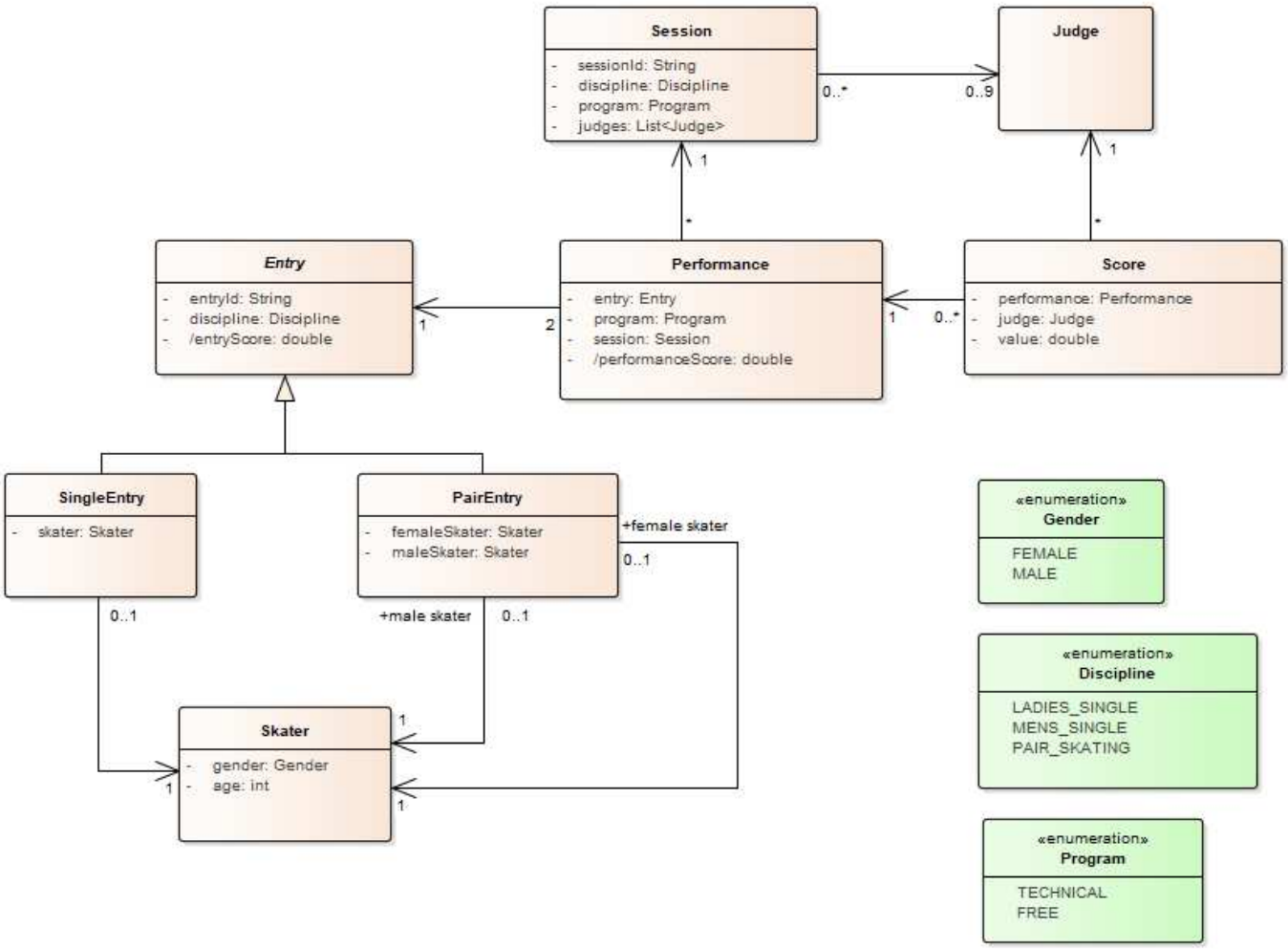

Figure 1 - Structural Design Model - Diagram 1

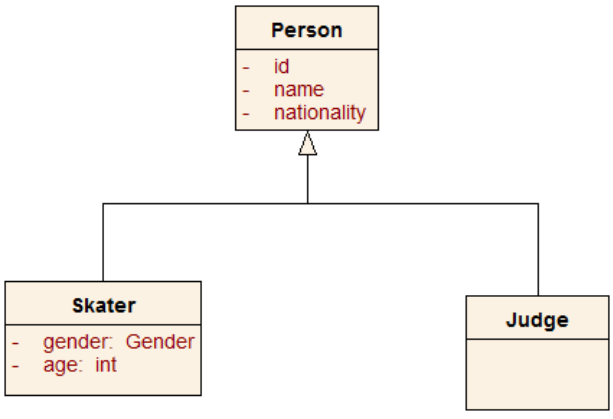

Figure 2 - Structural Design Model - Diagram 2

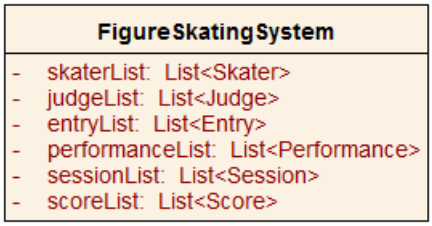

Figure 3 - Structural Design Model - Diagram 3

Regarding the requirements for the system to be prototyped,

- Unless otherwise specified, you must follow the description given in Part 1.
- If some features have been explicitly changed or added, they will override those in Part 1. For example, as shown in Figure 1,
  - A skater now has attribute `age` instead of date of birth.
  - Competition entries are now identified by their entry id's.
  - Sessions no longer have attributes date, start time and end time.
- Clearly state any additional assumptions you make. Make sure they are consistent with what are given in the handout.

* * *

A number of Java files have been provided, including the classes representing the domain objects. The two provided test programs must be able to run without changes.

Task 1 – Atomic Use Case Specifications (45 marks)

Formally specify the following atomic use cases:

- Use case 1: Add a skater
- Use case 2: Add a judge
- Use case 3: Add a single entry

  This use case registers a female skater for a Ladies' Single entry, or a male skater in a Mens' Single entry.

  This use case must also generate and save the two performances (technical and free programs) for the entry. Their session and entry score are initially recorded as missing values. See more about missing values below.
- Use case 4: Add a session

  This use case, as for part 1, does not involve entering details about the judges and performances assigned to it.
- Use case 5: Assign a performance to a session
- Use case 6: Assign a judge to a session
- Use case 7: Add a score for a performance by a judge
- Use case 8: Add the scores the nine judges awarded to a performance

  Note carefully, that one atomic use case specification *cannot call* another one. Each specifies a system behavior by itself.

Additional language features available:

- We can use enumerated types, as shown in the design class model, in the specifications of the atomic use cases. For example:

```
gender? : Gender
gender = Gender.FEMALE
```

- The keyword `null` can be used to denote a missing object and data value. However, in the prototypes, missing numerical values are to be represented by `-999`.
- There is available the function `choice` that takes a condition `C` and two values `V1` and `V2`

```
choice(C, V1, V2)
```

  and returns `V1` if `C` is true and `V2` otherwise.
- There is a function `isOfType` that takes an object and a class and return true if the object is of the type of the class, e.g.

```
isOfType(entry, SingleEntry)
```

- For convenience, the construct `let ... = element in ...|... then <boolean expression>` (i.e. item B4 in the "Syntax Summary") can be used to specify conditions about *more than one element*. In other words, we can have an expression like the one below:

```
let  x = element in ... | ... and
     y = element in ... | ...
then
     ...
```

Task 2 – Build Prototype Version 1 (12 marks)

- Build version 1 of the prototype for the system. For this version, you **only need to implement the postconditions** of the use cases listed in Task 1. The system class for this version is to be named `FigureSkatingSystemV1`.
- The code for this task, except for Use Case 8, can be copied from the posted class `FigureSkatingSystem`.
- Hence, effectively, you only need implement the postcondition for Use Case 8. Your implementation must be such that we can run the provided test program

  `FigureSkatingSystemV1Tester`

  which include a test for Use Case 8, without the need to make any changes.

Task 3 – Build Prototype Version 2 (26 marks)

- Copy all classes of Version 1 to a separate directory. Rename the system class to `FigureSkatingSystemV2`. Now, modify the system class to build version 2 of the prototype.
- For this version, in addition to the implementation of the postconditions, you are required to implement **all the preconditions** as well.
- The code for this task, except for Use Case 6 and Use Case 8, is available in class `FigureSkatingSystem`.
- Hence, effectively, you only need implement Use Case 6 and Use Case 8. Your implementation must be such that we can run the provided test program

  `FigureSkatingSystemV2Tester`

  without the need to make any changes.

  Hint: Though an atomic use case specification cannot call another one. A method can call another method. Hence, for Use case 8, you may want to call the method you already have for Use Case 7. However, you need to consider this point: Will your method preserve the system's state if a precondition is not satisfied? For example, if the second judge does *not* exist, will the system's state remain the same (as it should be)?

Task 4 – Test the Assign Performance to Session Use Case (17 marks)

Write a test program, called `Task4.java`, to test the *Assign a Performance to a Session* atomic use case. You must include both valid request test cases and invalid request test cases. You must use one method for the valid request test cases, and one method for each of the invalid request test cases. These methods must be independent of each other, i.e. each can be called by itself. You must also *include comments to explain the purpose of each test*.

Task 5 - For CSE5DES Only (10 marks)

Add to your class `FigureSkatingSystemV1` a method `calculatePerformanceScore` to calculate the performance score of a performance. You do not need to test it.

This task 5 is worth 10 marks. The total marks for CSE5DES will be 110, whereas the total mark for CSE2DES will be 100.

How to submit your work

Prepare three files as described below and submit them.

1. A pdf file containing the specifications of atomic use cases
2. A zip file that contains all the java files, including those provided, to compile and test version 1 of the prototype.
3. A zip file that contains all the java files, including those provided, to compile and test version 2 of the prototype.

The Java classes must not be in any package. In other words, we should be able to compile them from the system command prompt with the simple command

```
javac *.java
```

and run the test program using a simple command, for example,

```
java FigureSkatingSystemV1Tester
```

Your pdf file and each of your Java file must have your id, username, first name and last name and the subject code (CSE2DES or CSE5DES).

■