SENG301 Assignment 3
Acceptance Testing and Design Patterns

SENG301 Software Engineering II

Neville Churcher Fabian Gilson Julia Harrison Izaak Hoorens Olivia Mackintosh

27th April 2021

Learning objectives

By completing this assignment, students will demonstrate their knowledge about:

- how to write automated user acceptance tests;
- how to identify design patterns from a code base;
- how to justify the usage of particular design patterns to fulfil particular goals;
- how to retro-document a UML Class Diagram from a code base;
- how to implement new features by means of design patterns;
- how to extend a code-base following the identified patterns.

To this end, students will use the following technologies:

- the *Java* programming language with `gradle` dependency and building tool;
- the *Java Persistence API* with *Hibernate*;
- the *Cucumber* acceptance test framework;
- the *Mockito* stubing framework with an external API (i.e. MapQuest[1]);
- the Apache *Log4j2* logging facility;
- UML class diagram, either using a tool or by hand.

Next to this handout, a `zip` archive is given where the code base used for *Labs 3* to *5* has been updated for the purpose of this assignment.

1 Domain, story and acceptance criteria

This Assignment builds upon the *Event Organiser App* used during term 1 labs. The code base from *Lab 5* has been extended and modified to fit the purpose of this summative assignment. The user stories of current code base are listed in Section 1.1. A walkthrough of the code base is given in Section 1.2. Your tasks are described in Section 2 and the submission rules are stated in Section 3.

The domain model is reproduced in Figure 1[2]. Only domain classes (i.e. no implementation classes) are represented.

[1] **http://open.mapquestapi.com/nominatim/v1/search.php"**

[2] Squares next to class attributes denote their `private` visibility. See Section 3.5 of **http://plantuml.com/guide**.

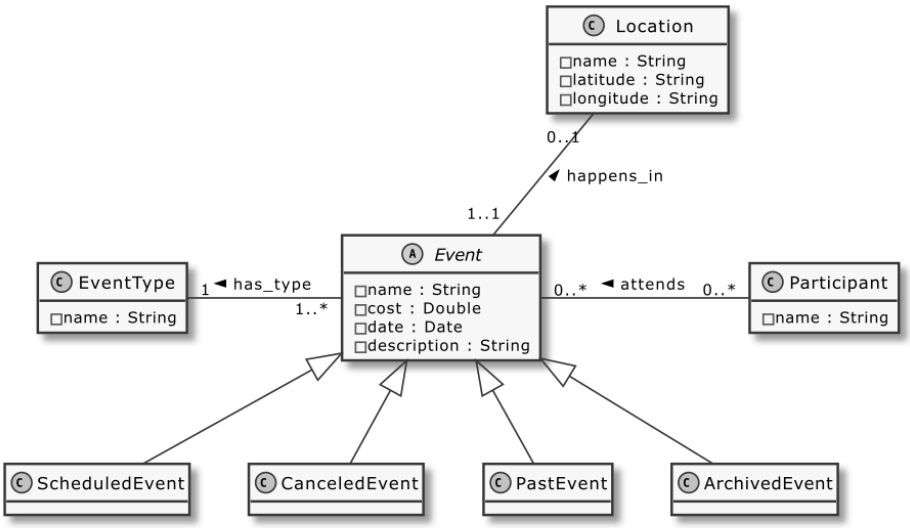

Figure 1: Partial domain model of an *Event Organiser App*

1.1 User stories of current code base

Let's assume your product owner came with the following user stories and acceptance criteria. For the sake of simplicity, we identified only one class of users for the system, called *"individual"*. This list expands on the user stories defined for the labs. Next to each AC below, we note which lab they relate too and if acceptance tests have been implemented already.

U1 As an individual, I want to create an event so that I will be able to invite my friends to it.

- AC.1 An event must have a name, a type, a description and a date that are non empty. Other attributes are optional. ***[Implemented in lab 4]***
- AC.2 I can create events with a name, a type, a description, a date and a cost. ***[Implemented in lab 4]***
- AC.3 Two events cannot have the same name. ***[Implemented in given code base]***
- AC.4 The event date must be in the future, but not later than an a year. ***[Scenario present in feature file]***

U2 As an individual, I want to retrieve the coordinates of the location I set for my created event so that participants can know exactly where to come.

- AC.1 I can add a location to an existing event which consists of a single string value at least. ***[Implemented in lab 5]***
- AC.2 I can retrieve a full description of this location via an external API using a textual value. The retrieved description, if any, also includes the geolocation coordinates of that location and all three fields are persisted with the event. ***[Implemented in lab 5]***

U3 As an individual, I want to add participants to an existing event so that we can have fun together.

- AC.1 I can add one existing participant to an event by its name (non-empty).
- AC.2 I can add one participant that does not exist yet. This participant will be created as a side effect with given non-empty name.
- AC.3 I cannot add empty participants, or participants with names containing invalid characters (e.g., numbers).

As can be seen from above list, automated tests for some ACs have been implemented. In the code base we give to you, all three user stories have been implemented, but U3 is not tested yet.

1.2 Walkthrough the code base

The code base used for *Labs 3* to *5* has been augmented with more features and most acceptance tests from past labs have been implemented for you to compare with what you wrote during those labs[3]. You should already be familiar with most of that code.

1.2.1 README and ANSWERS files

The `zip` archive with the code also contains a `README` and an `ANSWERS` file that you need to consult prior going deeper into the code. This `README` describes the content of the archive and how to run the project. The code is also extensively documented so take some time to read the *Javadoc*.

The `ANSWERS.md` file will contain your answers to the non-coding tasks. You are required to submit your answers in this file using the *Markdown* syntax[4]. The second line of that file asks you to add your name, **so do so straight away. Failing to write your answers in this dedicated file would prevent us to mark your assignment.**

1.2.2 Gradle configuration file

Take some time to review the `build.gradle` file. You should be familiar with its content as it is a complete version of what was needed to complete *Lab 5*. Note that the `test` task will trigger the `cucumber` one each time you build the code, so you do not need to run your *Cucumber* tests separately. **You are required to keep this file untouched, failing to do so would prevent us to mark your assignment.**

1.2.3 Model layer

In that package, you will find the domain model presented in Figure 1. Pay a particular attention to the `Event` class since some important things have changed there. An extensive documentation is provided with pointers to external references, so you should take some time to understand how this works.

1.2.4 Handler layer

The creation of `Event` entities has been extracted into a dedicated class (behind an interface). Some additional verification code has been implemented to satisfy U1 AC4 (that you will need to write acceptance tests for).

1.2.5 Accessor layer

A dedicated accessor has been implemented for `Participant` entities and `Participant`'s `Events` are now eagerly retrieved. See the dedicated parameter attached to the `@ManyToMany` annotation of the list of `Events` held by the `Participant` entity class.

You may also take a look at how the `EventAccessorImpl.getEventAndParticipantsById` method has been updated from Lab 5. i.e. usage of the dedicated `Hibernate.initialize()` method to fetch lazily loaded references.

1.2.6 Fake calendar

The `DateUtil`[5] class has been extended to statically fake a calendar. This manually-triggred calendar has been designed to simplify the overall design, your onboarding into the code and avoid unnecessary complexity for the extent of this assignment by manipulating dates easily.

[3] Be aware that small modifications in the code may imply that the acceptance test you wrote are slightly different to the ones in the supplied code base.

[4] See **`https://www.markdownguide.org/basic-syntax/`**

[5] This class is now a *Singleton* and does not count as one of the two patterns to document from the code.

1.2.7 Location API

From *Lab 5*, you learned how to use an external API to retrieve details about cities using a simple HTTP server and transparent JSON deserialization. The classes from *Lab 5* have been updated a little to abstract the actual implementation from the expected behaviour of the service[6]. **Remember to add your own API key that you created in Lab 5.**

1.2.8 Automated acceptance tests

You can take a look at both `feature` files under `app/src/test/resources/gradle/cucumber` to see the scenarios covering all acceptance criteria for U1 and U2. The test code is placed under `app/src/test/java/gradle/cucumber`.

1.2.9 Common line client (CLI)

After having taken some time to review the code base, you can run the project to get a deeper understanding of its existing features. Check the `README` file for explanations on how to run the CLI code (e.g., `$ ./gradlew --console=PLAIN run`). The `main` method is placed in the `App` class.

2 Your tasks

2.1 Task 1 - Write acceptance tests [35 MARKS]

2.1.1 Task 1.a - Write the acceptance test for U1 AC4 [5 MARKS]

In the `event.feature` file, you will find a scenario for that AC, but no test code has been provided in corresponding `CreateNewEventFeature` class. You are required to implement this cucumber scenario (**without changing its definition**).

When assessing this task, we will verify that:

- the automated acceptance test effectively checks the expected behaviour expressed in the AC;
- the test is self-contained, readable and follow the design practices taught in the course (e.g., Lecture 7 and additional material);
- the test passes (**a failing test award 0 marks**).

2.1.2 Task 1.b - Write *Cucumber* scenarios for *"U3 - Add participants to events"* [3x10 MARKS]

In Section 1.1, acceptance criteria have been given for user story 3 relating to add participants to events. You are required to:

- create a new `feature` file for that story;
- translate the three acceptance criteria of U3 in *Gherkin* syntax (i.e. *Cucumber* scenario);
- implement the acceptance test for each of the scenario into a new test class;

When assessing this task, we will verify that:

- you have adequately translated the acceptance criteria, i.e. the **behavioural semantic** of the *Gherkin* scenario **is identical to the English version** given in Section 1.1;
- the automated acceptance tests effectively **checks the expected behaviour expressed in the AC**;
- the tests are self-contained, readable and follow the design practices taught in the course (e.g., Lecture 7 and additional material);

[6] Do not be mislead, this is no *Façade* pattern since only one service is provided where the *Façade* abstracts a series of complex services. See **https://refactoring.guru/design-patterns/facade** for more details.

- the tests pass (**a failing test or a test that would not exactly translate the AC into *Cucumber* award 0 marks for that AC**).

You are required to fill in the name of the `feature` and Java class in the `ANSWERS.md` file under *"Task 1.b"*.

2.2 Task 2 - Identify two patterns [30 MARKS]

The code base you received contains two design patterns from the famous *"Gang of Four"* [1] (GoF) book that you need to identify in this second task.

For each pattern, you will need to (awards **2x15 MARKS**):

- name the pattern you found and give a brief summary of its goal **[2 MARKS]**
- justify how this pattern is instantiated in the code, *i.e.* by mapping pattern components to Java classes and methods using a table (as done in class), as follows: **[8 MARKS]**
  - map *GoF* patterns components to their implementation Java classes
  - map the relevant pattern methods to their corresponding implementation in the code (note that some non-critical pattern methods may not be present in the code)
- draw the UML diagram of the actual implementation of the pattern (**with the relevant methods only**) by following a similar structure to the pattern, but using the actual class names in the code base **[5 MARKS]**

Answers to above questions must be provided into the `ANSWERS.md` file under *"Task 2"*. UML Class Diagrams [2, chap.11] will need to be referred to by specifying the name of the image file where asked in that `ANSWERS.md` file. These UML diagrams must be placed under the dedicated `diagrams` folder (where you will find a copy of Figure 1).

2.3 Task 3 - Retro-document the design of the existing code base [10 MARKS]

You are asked to retro-document the overall design in the form of a UML Class Diagram [2, chap.11]. **No changes in the source code is required for this task.** To this end, you can build on:

- the domain model given in Figure 1;
- the walkthrough you conducted, helped by Section 1.2;
- the patterns you identified in task 2.

The full diagram must be put in the `diagrams` folder that you used already in *Task 2*. You need to add its name in the `ANSWERS.md` file under *"Task 3"*.

Note that **no generated diagrams** (e.g., from automatic extraction from *IntelliJ*) **will be accepted** for this task. For suggestions of tools, refer to Section 4. When marking your diagram, we will look at:

- the syntactic validity, i.e. is it a valid UML 2.5 class diagram;
- the semantic validity, i.e. are all classes and associations semantically valid (e.g., no wrong types of associations or inexistent ones), are all associations multiplicities present and valid;
- the completeness, i.e. are all classes of the code base present (but you may consider to hide/shorten some methods for readability reasons)
- the readability, e.g., meaningful names on associations, readable layout.

2.4 Task 4 - Implement U4 using a GoF pattern [25 MARKS]

For this task, we supply a fourth user story below:

U4 As an individual, I want to get notified when the status of an event changes so that I am up to date.

AC.1 When the status of an event I am attending changes, I receive a notification containing the event name and the new status of this event.

AC.2 When being notified of a status change, I print a message containing my name, the event name and the new status of the event.

AC.3 When receiving the notification, I can remove myself from the participants list of that event.
AC.4 If an event is "archived", all participants are removed from that event.

In a similar fashion to *Task 2*, you are expected to provide the following under *"Task 4"* in the `ANSWERS.md` file:

- name the pattern you found and give a brief summary of its goal **[1 MARKS]**
- justify how this pattern is instantiated in the code, *i.e.* using a table (as done in class and for *Task 2*) **[4 MARKS]**
  - map *GoF* patterns components to their implementation Java classes
  - map the relevant pattern methods to their corresponding implementation in your new code

When assessing your task, we will verify that **[20 MARKS]**:

- the expected feature is implemented, i.e. **all ACs pass [10 MARKS]**;
- you **respected the pattern** you selected to the letter **[10 MARKS]**;
- your code runs (**unexpected crashes in the code would award 0 marks**, i.e. 0 out of 20 marks).

2.5 Task 5 (BONUS) - Implement acceptance tests for U4 [10 MARKS]

As a bonus, you can implement acceptance tests for above story U4 (award **[2.5 marks]** per correctly implemented AC following the same rubric as *Task 1.b*). The name of the files (`feature` and java class) must be specified in the `ANSWERS.md` file under *"Task 5"*.

3 Submission

You are expected to submit a `.zip` archive of your project named `lastname_seng301_asg3.zip` on Learn by **Friday 4 June 6PM**[7]. **No other format than `.zip` will be accepted**. Your archive must contain:

1. the updated source code (please remove the `.gradle`, `bin`, `build` and `log` folders, but keep the `gradle` - with no dots - folder); **do not remove the `build.gradle` file or we will not be able to assess your assignment**;
2. your answers to the questions in the given `ANSWERS.md` file (you are **required to keep the file format intact**);
3. the UML Class diagrams in `.png`, `.pdf` or `.jpg` format under the `diagrams` folder. All diagrams must be images (you can join the other files, *e.g.,* `.dia` or `.plantuml` if you like). If you use pen & paper photos, please ensure we can read them, it is **your responsibility** to make sure they are legible.

Your code:

1. **may not** import other libraries / dependencies than the ones currently in the `build.gradle` file;
2. **will not be evaluated** if it does not build straight away or fail to comply to 1. (e.g., no or wrong `build.gradle` file supplied);
3. **will be** passed through an advanced clone detection tools (*i.e.* Txl/NiCad) that proved to be performing well on sophistically plagiarised code earlier.

The marking rubric is specified next to each task (overall 100 marks), but is summarised as follow:

Task 1 write acceptance tests from given scenario **5 MARKS**, write scenario and acceptance tests for U3 **3x10 MARKS**
Task 2 name **2x2 MARKS**, map **2x8 MARKS** and draw the patterns **2x5 MARKS**
Task 3 draw the full UML diagram **10 MARKS**
Task 4 name **1 MARK**, map **4 MARKS** and implement the feature using a GoF pattern **20 MARKS**
Task 5 correct and functional implementation of U4 ACs **4x2.5 MARKS** (BONUS)

[7] There is a 4-days no-penalty drop dead date (i.e. extension by default). No further extension will be granted, unless special consideration.

4 Tools

You only need the following tools to run and develop your program if you work on your own computer:

- an IDE, *e.g., IntelliJ IDEA* **https://www.jetbrains.com/idea/download/**
- *OpenJDK Java SDK* **https://openjdk.java.net/install/**
- for *Windows* users, setting up your environment variables for Java to be recognised: **https://stackoverflow.com/a/52531093/5463498**

The code you receive already contains the minimal binaries for `gradle` that will manage the dependencies for you. Please refer to the `README` shipped with the code for more details. You should be familiar with the process as it is the same as for term 1 labs.

We suggest *Typora*[8] or an embedded markdown editor (in *Intellij IDEA* or *VSCode*) to edit your `ANSWERS.md` file.

As UML drawing tools, we recommend the following (in no particular order):

- available in lab machines (and LabBox) and for all operating systems:
  - *dia*, **http://dia-installer.de/download/index.html**
  - *umbrello*, **https://umbrello.kde.org/installation.php**
  - *argoUML*, **https://github.com/argouml-tigris-org/argouml**
- web- and textual-based *PlantUML*, **https://plantuml-editor.kkeisuke.com/**[9]
- pen and paper, and a decent camera or scanner.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Object Management Group, "OMG unified modeling language (OMG UML), version 2.5.1," https://www.omg.org/spec/UML/2.5.1/PDF, 2017.

[8] See **https://typora.io**, still free in its beta version.

[9] Plugins exist for *IntelliJ IDEA*: **https://plugins.jetbrains.com/plugin/7017-plantuml-integration** and *VSCode*: **https://marketplace.visualstudio.com/items?itemName=jebbs.plantuml**