SENG301 Assignment 5
Design Patterns

SENG301 Software Engineering II

Neville Churcher Morgan English Fabian Gilson

6th May 2022

Learning objectives

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

- 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;
- the Apache *Log4j2* logging facility;
- UML class diagram, either using a tool like *Plantuml* or by hand.

Next to this handout, a `zip` archive is given where the code base used for *Labs 3* to *5* and for *Assignment 4* has been updated for the purpose of this assignment.

1 Domain, story and acceptance criteria

This Assignment builds upon the *Wordle Clone App* used during term 1 labs and the previous assignment. The code base from *Assignment 4* 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.

A simplified class diagram is reproduced in Figure 1. The diagram depicts the domain model and basic functionality of the application. **Note that several classes are missing or not fully described on purpose.**

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 *"player"*. This list expands on the user stories defined for the

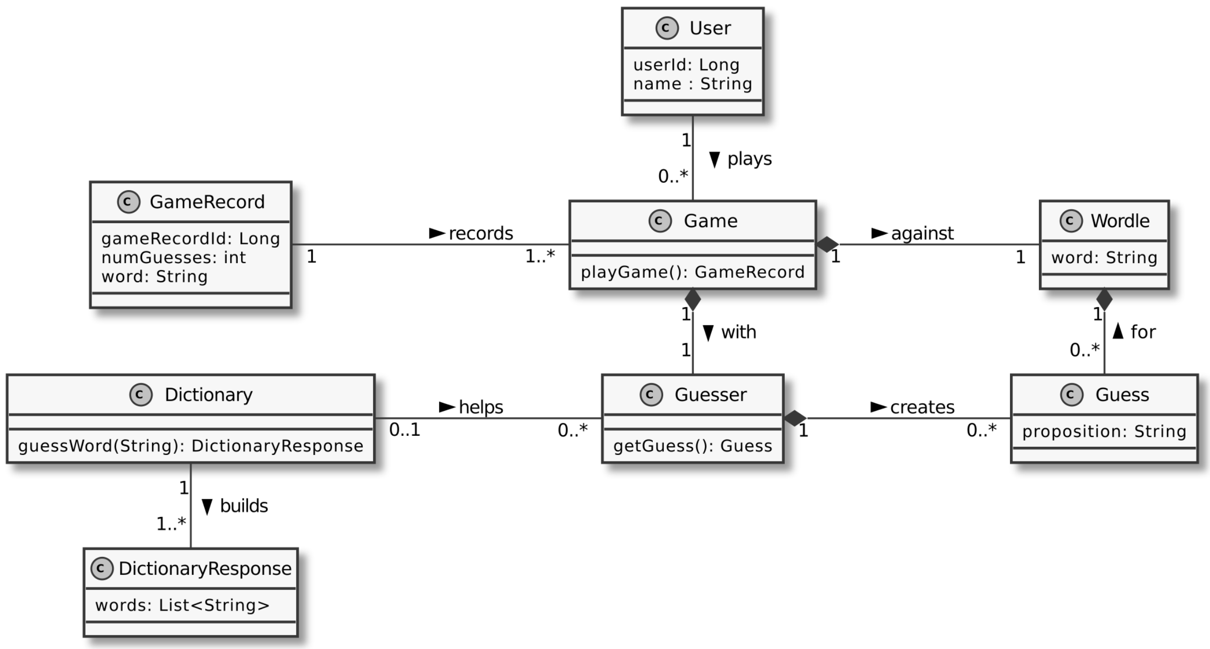

Figure 1: Partial class diagram of the *Wordle Clone App*.

labs and may be slightly modified from those given in Assignment 4. The acceptance tests for these AC's have been implemented already.

- U1 **As a player, I want to record the outcome of my wordle game so that I can keep track of my previous games.**
  - AC.1 I can create a game record with a user, a word, and a number of guesses that are not empty. ***[Implemented in lab 4]***
  - AC.2 Two game records may exist with the same attributes but must have different timestamps. ***[Implemented in lab 4]***
  - AC.3 A game record can not have 0 or fewer guesses. ***[Implemented in lab 4]***

- U2 **As a player, I want to retrieve possible guesses so that I can see possible options.**
  - AC.1 I can get a list of possible words given the letters I know. ***[Implemented in lab 5]***
  - AC.2 I can get a list of all possible words. ***[Implemented in lab 5]***
  - AC.3 If no words match given the letters I know none will be returned. ***[Implemented in lab 5]***

- U3 **As a player, I want to play a game by suggesting words one after the other so that I know how many guesses it took to guess a word.**
  - AC.1 Every time I make a valid guess, the guess count is incremented. ***[Implemented in Assignment 4]***
  - AC.2 I cannot make a guess with strictly less than five letters. ***[Implemented in Assignment 4]***

- U4 **As a player, I want to see a colour coded reply from my guesses so that I know what characters are correctly guessed, placed or neither.**
  - AC.1 A correctly guessed word will have all its letters coloured in green. ***[Implemented in Assignment 4]***
  - AC.2 A correctly guessed and placed letter is coloured in green, a correctly guessed but not placed letter is coloured in yellow and other incorrect letters are coloured in gray. ***[Implemented in Assignment 4]***

In the code base we give to you, all four user stories have been implemented together with their acceptance tests.

1.2 Walkthrough the code base

The code base used for *Labs 3* to *5* and *Assignment 4* has been augmented with more features and acceptance tests have been implemented for you to compare with what you wrote during those labs and assignment 4[1]. You should already be familiar with most of that code.

1.2.1 README

The `zip` archive with the code also contains a `README` and a `LICENSE` 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*.

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*. **You are required to keep this file untouched, failing to do so would prevent us from marking your assignment and you will be awarded 0 marks for the assignment.**

1.2.3 Model layer

In that package, you will find the domain entities (as presented in Figure 1). These two classes have basic documentation.

1.2.4 Accessor layer

Accessors have been implemented for `GameRecord` and `User`. These classes have a more detailed documentation and some implementation details have been changed from *Assignment 4*. You may spot the difference and think why those changes have been made in light of term 2 material.

1.2.5 Dictionary, *a.k.a.* Solver API

From *Lab 5*, you learned how to use an external API to retrieve possible guesses using a simple HTTP server and transparent JSON deserialization. The classes from *Lab 5* have been updated, so take some time to read the *Javadoc*.

1.2.6 Guesser

The `ManualGuesser` class from lab 5 has been generalized, and now new guessers `RandomGuesser` and `SmartGuesser` have been added. These new guessers simulate playing the game and require no input from the player. You may take a look at how the `Knowledge` class works to keep track of guesses but do not spend too much time there[2].

1.2.7 Automated acceptance tests

You can take a look at the four `feature` files under `app/src/test/resources/gradle/cucumber` to see the scenarios covering all acceptance criteria for U1, U2, U3 and U4. The test code is placed under `app/src/test/java/gradle/cucumber` and has been updated to include acceptance tests from Assignment 4. **Pay particular attention to the naming convention we used. We expect you to follow that naming if you plan to achieve the bonus task.**

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

[2]This class is a naive and suboptimal implementation.

1.2.8 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 (i.e. `$ ./gradlew --console=plain run`). The `main` method is placed in the `App` class. For help running different commands, use the command *help* to see a list of all commands accepted by the CLI. When running the `App`, all commands are displayed. Here also, we made some changes on how the `output` is managed.

2 Your tasks

2.1 Task 1 - 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 **in the current code[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; **note that** it is possible that an association in the *GoF* pattern is implemented in a slightly different way, but you need to draw the actual implementation **[5 MARKS]**

Answers to above questions must be provided into the `ANSWERS.md` file under *"Task 1"*. 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).

2.2 Task 2 - Retro-document the design of the existing code base [5 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 1.

The full diagram must be put in the `diagrams` folder that you used already in *Task 1*. You need to add its name in the `ANSWERS.md` file under *"Task 1"*.

Note that **generated diagrams** (e.g., from automatic extraction from *IntelliJ*) **will not be accepted** for this task. 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 non-existent 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.

2.3 Task 3 - Implement U4 using a GoF pattern [25 MARKS]

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

U5 As a manual player, I want to be able to undo and redo my guesses so that I can repeat a game.

AC.1 An "undone" guess is removed from my guesses.
AC.2 The number of guesses is decreased by one when I undo a guess.
AC.3 I can redo a guess only if I have undone a guess.
AC.4 When I redo a guess, the undone guess is re-added to the stack of guesses.
AC.5 I can redo all my guesses in the reverse order I made the guesses, i.e. if I guessed (in order) "fuses", "curls" and "dwarf", then undone (in the reverse order they have been made) "dwarf", "curls" and "fuses", I can redo them in their original order.

In a similar fashion to *Task 1*, you are expected to provide the following under *"Task 3"* in the `ANSWERS.md` file:

- name the pattern you found and give a brief summary of its goal in this code **[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, regardless you implemented the pattern properly).

2.4 Task 4 (BONUS) - Implement acceptance tests for U5 [5 MARKS]

As a bonus, you can implement acceptance tests for above story U5 (award **[1 MARK]** per correctly implemented AC). The name of the files (`feature` and java class) must be specified in the `ANSWERS.md` file under *"Task 4"*.

- create a new `feature` file for that story (remember to respect the naming convention);
- translate the three acceptance criteria of U5 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);
- the tests pass (**a failing test or a test that would not exactly translate the AC into *Cucumber* award 0 marks for that AC**).

3 Submission

You are expected to submit a `.zip` archive of your project named `seng301_asg5_lastname-firstname.zip` on Learn by **Friday 3 June 6PM**[3]. **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 assess your assignment and you will**

[3]There is a 4 day grace period (i.e. extension with no penalty by default). No further extension will be granted, unless special consideration.

be awarded 0 marks;
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 **60 marks**), but is summarised as follow:

Task 1 name **2x2 MARKS**, map **2x8 MARKS** and draw the patterns **2x5 MARKS**
Task 2 draw the full UML diagram **5 MARKS**
Task 3 name **1 MARK**, map **4 MARKS** and implement the feature using a GoF pattern **20 MARKS**
Task 4 correct and functional implementation of U4 ACs **4x1 MARK** (BONUS)

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. If the `gradlew` command has issues (unix), you can recreate the wrapper by running `$ gradle wrapper` (from your local install), as we showed you in *Lab 3*.

We suggest *Typora*[4] or an embedded markdown editor (in *Intellij IDEA* or *VSCode*) to edit your `ANSWERS.md` file.

As UML drawing tools, we recommend the following web- and textual-based drawing *PlantUML* (as used in labs during term 2), `https://plantuml-editor.kkeisuke.com/`. Plugins exist for *IntelliJ IDEA*: `https://plugins.jetbrains.com/plugin/7017-plantuml-integration` and *VSCode*: `https://marketplace.visualstudio.com/items?itemName=jebbs.plantuml`.

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.

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