Assignment 3 – Acceptance Testing and mocking

SENG301 Software Engineering II

Neville Churcher Morgan English Fabian Gilson

2nd April 2024

Learning objectives

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

- translating acceptance criteria into automated acceptance tests;
- using mocking to create independent tests;
- writing automated acceptance tests.

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* and *sqlite*;
- the *Cucumber* and *Mockito* stubbing frameworks;
- the Apache *Log4j* logging facility;

Next to this handout, a `zip` archive is given where the code base used for *Labs 4* to *6* has been updated for the purpose of this assignment.

1 Domain, story, and acceptance criteria

This Assignment builds upon the *Super Auto Pets Clone App* used during term 1 labs. The code base from *Lab 6* has been extended and modified to fit the purpose of this 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[1]. We ignore the `Game`, `CommandLineInterface` and `Accessors` classes for simplicity.

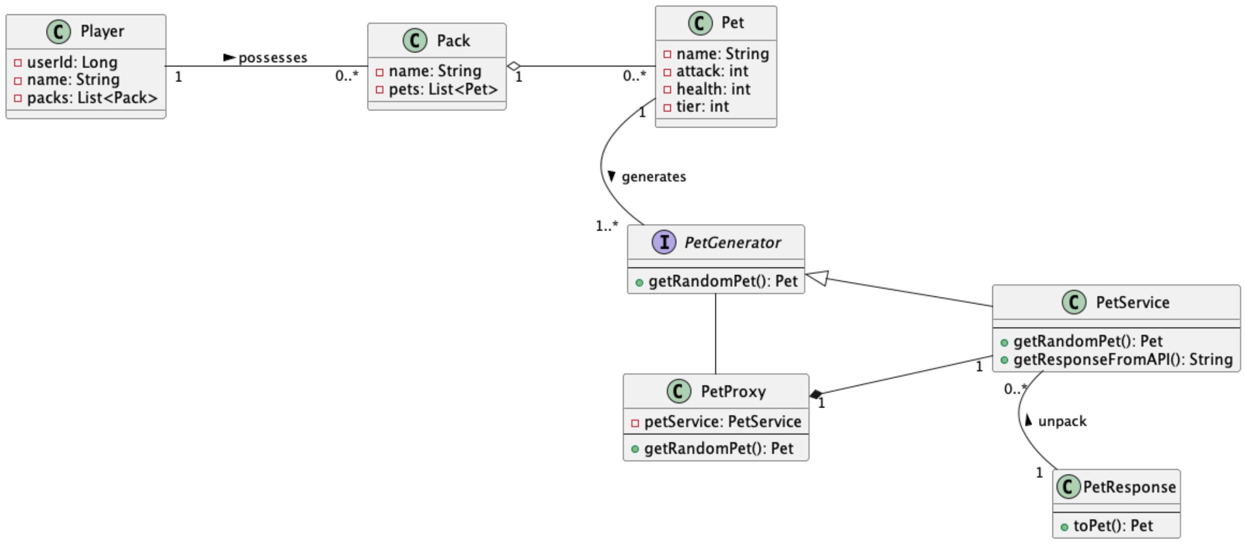

Figure 1: Domain model of *PetBattler App* (same as lab 6)

1.1 User stories of current code base

This list expands on the user stories defined for the labs (with *Alex*, our only persona). Next to each AC below, we note which lab they relate to and if acceptance tests have been implemented already in previous labs. Still, we supply model answers in the attached code. Additionally, for U4, we supply the `features` file with the ACs 1, 2 and 3 in *Gherkin* syntax already, you are required to use the same feature file for U4 ACs 4, 5 and 6.

U1 As *Alex*, I want to create a pet so that I can use it in a pack.
- AC.1 A pet has a unique non-empty name, a strictly positive attack and health stats. ***[Implemented in lab 5]***
- AC.2 A pet name can contain alphabetic characters only. ***[Implemented in lab 5]***
- AC.3 A pet cannot have a negative or zero values for the attack or life stats. ***[Implemented in lab 5]***

U2 As *Alex* I want to create a pack so that I can build a set of pets to use in battle.
- AC.1 A pack has a unique, non-empty alphanumeric name. ***[Implemented in lab 5]***
- AC.2 A pack cannot have a numeric-only name, or special characters in its name. ***[Implemented in lab 5]***
- AC.3 A pack must be able to store zero to many pets. ***[Implemented in lab 5]***

U3 As Alex, I want to draw random pets from an external API so that I can build a pack from them.
- AC.1 I can draw a random pet that has valid name, tier, and attack and health stats. ***[Implemented in lab 6]***
- AC.2 If I find a suitable pet, I can add the pet to my pack. ***[Implemented in lab 6]***
- AC.3 I can decide to ignore the pet and not add it to my pack. ***[Implemented in lab 6]***
- AC.4 A pack cannot contain the same pet twice. ***[Implemented in lab 6]***

U4 As *Alex*, I want to create a team from the pets in one of my packs so that I can use it in a battle.
- AC.1 A pack must contain at least one pet to build a team with. ***[Gherkin Written]***

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

AC.2 When building a team there will be 5 options randomly selected from my pack. ***[Gherkin Written]***
AC.3 When building a team I must select 3 options. ***[Gherkin Written]***
AC.4 When building a team I must select 3 unique options.
AC.5 When building a team and I select 3 unique options then my team is ordered the same way I entered my options.
AC.6 When building a team and I select 3 options of all the same pet type they are discrete copies from those in my pack.

1.2 Walkthrough the code base

The code base used for *Labs 4* to *6* has been augmented with more features and acceptance tests from past labs have been implemented for you to compare with what you wrote during those labs[2]. 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 6*. **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 model presented in Figure 1. An extensive documentation is provided with pointers to external references, so you should take some time to understand how this works.

1.2.4 Accessor layer

Accessors have been implemented for `Pet`, `Pack` and `Player`.

1.2.5 SAP pet API

From *Lab 6*, you learned how to use an external API to retrieve random pets using a simple HTTP server and transparent JSON deserialization. These classes make use of the *Proxy* pattern, see `https://refactoring.guru/design-patterns/proxy` for more details.

1.2.6 Automated acceptance tests

You can take a look at the five `feature` files under `app/src/test/resources/uc/seng301/petbattler/asg3/cucumber` to see the scenarios covering all acceptance criteria for U1, U2 and U3 along with the first three ACs of U4 (and an limited test for player names, called U0). The test code is placed under `app/src/test/java/uc/seng301/petbattler/asg3/cucumber` (for U0, U1, U2, and U3). **Take the time to review these model answers from all labs and pay particular attention to the naming convention we used.** You are required to follow that naming convention.

[2]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.

1.2.7 Command line interface (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 --quiet 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.

2 Your tasks

2.1 Task 1 - Write acceptance tests for U4 AC1-3 [3 x 10 MARKS]

In the `u4-build-team.feature` file, you will find 3 scenarios for U4, but no test code has been provided. You are required to implement these *Cucumber* scenarios in a new (*appropriately named*) step definition file (**without changing their definition**).

The implementation **must** use the `Game.buildTeam()` function, much like what was done in lab 6 for U3.

When assessing this task, we will verify that:

- the automated acceptance tests effectively check the expected behaviour expressed in both ACs (i.e. scenarios);
- the tests are self-contained, readable and follow the design practices taught in the course (e.g., Lecture material on testing, additional material, and labs);
- the test passes i.e. **a failing test awards 0 marks for the failing AC**.

2.2 Task 2 - Write *Cucumber* scenarios in *Gherkin* syntax for U4 AC4-6 [3 x 5 MARKS] and implement these scenarios [3 x 10 MARKS]

In Section 1.1, acceptance criteria have been given for user story 4 relating to building a team from a selected pack. You are required to:

- translate the three acceptance criteria of U4 (AC4-6) into *Gherkin* syntax (i.e. *Cucumber* scenarios);
- implement the acceptance tests for the scenarios into the same test class as you used for Task 1.

When assessing this task, we will verify that:

- you have adequately translated the acceptance criteria, i.e. the **behavioural semantics** of the *Gherkin* scenario **is identical to the English version** given in Section 1.1;
- the automated acceptance tests effectively **check 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 8, additional material and labs);
- the tests pass, i.e. **a failing test awards 0 marks for that AC**.

3 Submission

You are expected to submit a `.zip` archive of your project named `seng301_asg3_lastname-firstname.zip` on Learn by **Friday 5 May 6PM**[3]. **No other format than `.zip` will be accepted**. Your archive must contain 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 and you will be awarded 0 marks**.

Your code:

1. **may not** make any changes to the existing application code;

[3]There is a one week grace period with no penalty. No further extension will be granted, unless special consideration.

2. **may not** import other libraries / dependencies than the ones currently in the `build.gradle` file;
3. **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);
4. **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 **75 MARKS**), but is summarised as follow:

Task 1 write acceptance tests from given scenarios for U4 **3x10 MARKS**.

Task 2 translate ACs 4,5 and 6 from U4 into *Gherkin* syntax (*a.k.a. Cucumber scenarios*) **3x5 MARKS**, and write the tests for these 3 scenarios **3x10 MARKS**.

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.