SWEN30006 Software Modelling and Design
Workshop 1: Use Cases and Java/Eclipse

School of Computing and Information Systems
University of Melbourne
Semester 2, 2020

Assessment: You *should* complete the exercises ***as a team***. Check instructions carefully as some exercises should be completed individually. To receive a workshop mark, you **need** to demonstrate your active participation during the workshop. Your tutor will observe your participation for you to be eligible to receive a mark for this workshop. The active participation include (but not limit to) present your solutions, help your teammates, and actively engage with the team. Attending the workshop with only passive participation will *not* be given a mark. See the Workshop Overview under Subject Information on the LMS for more details.

Requisite Knowledge and Tools

To complete this workshop you will need to be familiar with the following terms and concepts:

- Use cases
- Use case diagrams
- Object-oriented programming in Java

Introduction

This workshop looks at the elements that bracket our modelling and design content. At the start point is a representation of user requirements in the form of use cases. At the end point is engineered software developed in Java using an Interactive Development Environment (IDE).

This workshop will cover some interpretation and construction of use cases and use case diagrams. By the end of the workshop you should be comfortable with creating simple use cases and use case diagrams.

The workshop also aims to familiarise you with Eclipse, the IDE we will be using through this semester and provide some revision in basic Java and Object-Oriented programming concepts. By the end of the workshop you should be familiar and comfortable with basic actions in the development environment that we will be using throughout the semester.

Part 1 Use Cases

These tasks will take you through the process of creating, modifying, and interpreting some use cases. When you are selecting a task for your use case, remember to *keep it simple* – use cases can very quickly grow in size as you add alternative paths to the main success scenario.

The System Under Discussion

You recently joined the startup company *Laze Dev* as a software developer. The project you have been assigned to is *LazyFair*, a delivery service. *LazyFair* was originally going to be a competitor to the popular *Uber Eats* service, but the company directors have decided that it will be expanded to offer delivery of any product from any participating store, not just food. There are three distinct areas of functionality that need to be designed and implemented:

1. A way for stores to advertise their goods
2. A way for purchasers to browse available goods, select items for purchase, and pay for their selected items
3. A way for deliverers to see what deliveries need to be completed, the items to be delivered (so they can decide whether the items will fit into their vehicle), and a mapping system which directs them to the pick-up and drop-off locations for delivery jobs they choose to accept

Each area is only for a subset of the user base (e.g., stores don't need to access the section of the software intended for deliverers, and vice versa). Your job is to help scope and design this application.

Q1.1 Write a Main Success Scenario

This task is done individually: Write a main success scenario. Pick any of the three areas of the application, and write a main success scenario for a task in that area. Possible tasks include:

- A store manager adding, removing, or updating an item in their catalogue
- A new store adding themselves to the application
- A deliverer selecting an open delivery
- A purchaser adding some items to their cart and paying for them

Select one of these tasks, or come up with your own. Write the main success scenario as a series of numbered steps. This will be important for a later task!

Q1.2 Group Discussion

Present the main success scenario to the team – it's fine if you've written scenarios for different areas of the application.
Discussion Topics:

- Does the scenario make sense?
- Are there any missing steps?
- Is the level of detail appropriate?
- Any other questions you come up with.

Keep some brief notes about your discussion.

Note: Your tutor may ask the summary of your discussion to assess your active participation.

Q1.3 Add Some Alternate Scenarios

This task is done as a team: Pick one of the main success scenarios that everyone made earlier to add at least three alternate paths. When writing these paths reference the steps in the main success scenario by their number.

Tip: To work together, you may use a collaborative tool (e.g. Google Doc) so that everyone will have the opportunity of contributing actively. You should also share your screen via Zoom so that your tutor can observe your participation.

Q1.4 Draw a Use Case Diagram

Collaboratively draw a use case diagram that includes the use cases that the team produces earlier. Remember to think about who the actors are, their type (i.e., primary/secondary/offstage), and the system boundaries.

Tip: To work together, one of you may use an online tool (e.g. draw.io) and share screen via Zoom. The others should actively contribute by offering suggestions and ideas about how the diagram should be drawn.

Q1.5 Show Your Understanding

Write **brief** responses to the following questions. You might want to refer to the tasks you have done in this workshop, or come up with other examples.

1. What are the qualities of a well written use case?
2. When designing software, how many use cases should you write? Justify your answer!
3. How does use case text differ from a use case diagram? Do we need both?
4. There are three kinds of actors that can participate in a use case: primary, supporting, and offstage. What are the differences between them? Use an example.
5. What do you think is the biggest challenge to writing a quality use case?

Note: You can present your work of Q1.4 and Q1.5 to your tutor to demonstrate your active participation (only for those who present the diagram and answer the questions)

Part 2 The Development Environment

In SWEN30006, we will be supporting the use of the Eclipse IDE for Java development.

If you want to use your own laptop for this workshop please ensure you visit The Eclipse Project, download the latest version of the *Eclipse IDE for Java EE Developers* for your operating system, and proceed with installation.

Feel free to use any development environment you would like. However, please be aware that if you choose to use something other than Eclipse, the tutors may not be able to help you with other IDEs.

Note: Part 2 exercises should be completed *individually*. However, you are welcome to help your teammates to demonstrate your active participation!

Q2.1 Creating an Eclipse Project with Existing Code

This task will walk you through importing an existing Java program into Eclipse. To begin, either open Eclipse from the start menu (if you are using one of the lab computers), or from where you installed it (if you are using your own computer). You will be asked to select a directory, which Eclipse will use to store your code workspace. Select any directory you like.

You may now see the Eclipse welcome screen. If you do, close it by pressing the small 'X' next to the tab labelled 'Welcome'. You should now be presented with a screen similar to Figure 1.

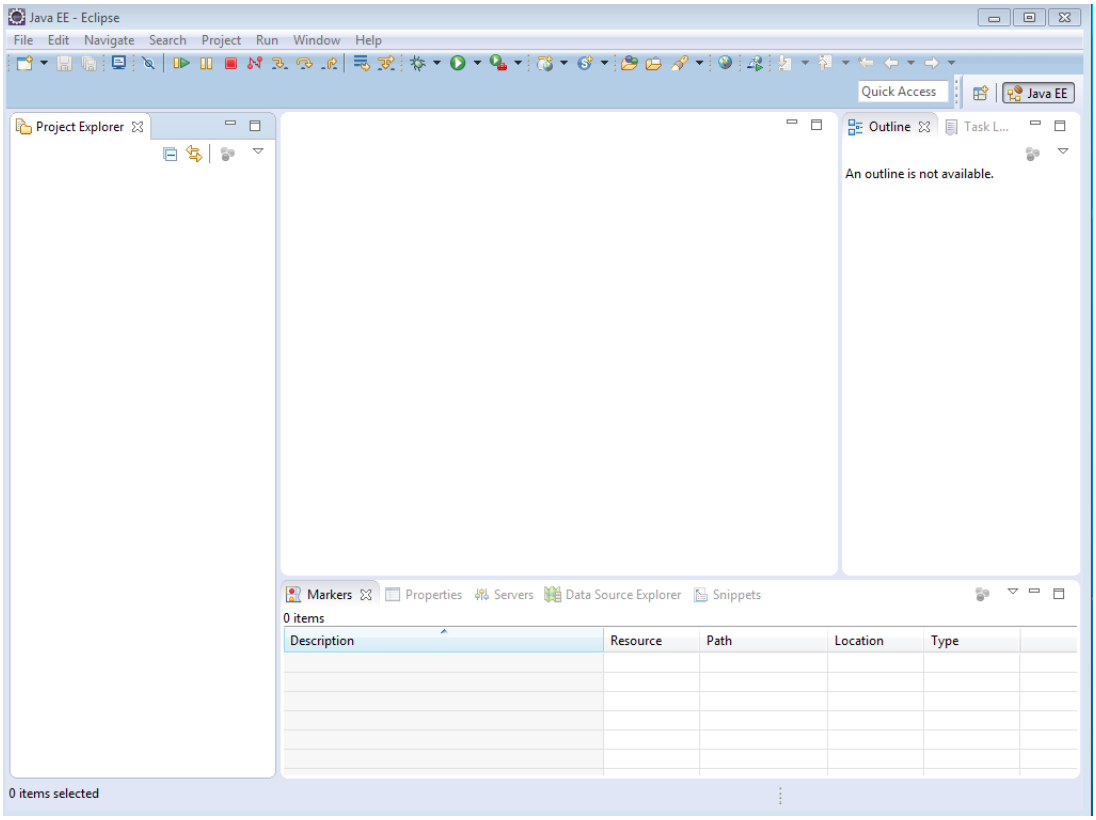

Figure 1: Eclipse workbench

This is the default view for everything that you do in Eclipse. Depending on whether you have used Eclipse or not before, you may need to change window layout to the **Java Perspective**. To do this open

the **Window** menu, select **Perspective**, **Open Perspective**, then **Java**. If the 'Java' option is not listed, you are already in the java perspective.

Now we need some code! Go to the LMS for this unit, click on the **Larman Case Study Code** link on the left, and download the archive file **Monopoly_It1.zip**.

Right click in the Project Explorer (or Package Explorer) window and select **Import...**. Then select **Projects from Folder or Archive** (you can use the search bar to find it) and click **Next >**. To the right of 'Import source:' click **Archive...**, then select the zip file you just downloaded and click **Finish**.

You should now see a directory containing the imported code files, similar to the one in Figure 2. Run the program by pressing the green play button in the Eclipse toolbar.

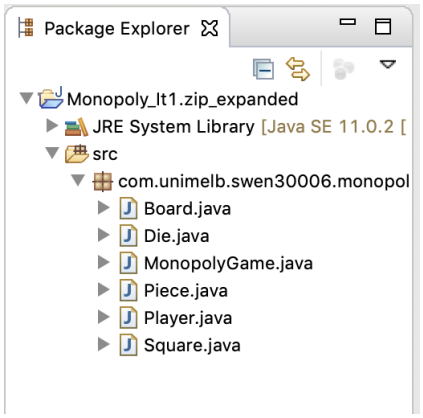

Figure 2: A project imported into Eclipse

Q2.2 Changing the Code

Now that you've set up your Eclipse project, we're going to make a small change to the code. Look through the code, then modify it so that at the end of the simulation it prints out the number of rounds that were simulated.

Note: You should present this change working to your tutor.

Q2.3 Setting up a New Eclipse Project

From the java perspective in Eclipse, click **File**, then **New**, then **Java Project**. This should create a popup window. In this window enter a name for your project. The default location should be the workspace you set up earlier. Feel free to leave it, or create a new workspace. Once you have entered a name, click **Finish**. Again, you do not need to create a *module-info.java* file.

In the Project Explorer (or Package Explorer) you should now see a directory for your new project, similar to the one in Figure 3. In this case the new project has been called *WordFrequency*. This name relates to the next task.

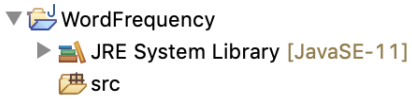

Figure 3: A new project created in Eclipse with the name *WordFrequency*

To add java class files to your project, right click on the **src** folder, select **new**, then select the kind of

file you would like to create (e.g., class, interface etc.). You will then be invited to enter a name for the new file and its package.

Q2.4 Writing a Small Program

For this task you need to write a small program in java. This program should be able to:

- Read a basic text file, the name of which is passed in as a command line argument,
- Print out an alphabetical list of all the unique words in the file, and
- Print out how many times each word occurred in the file.

The output of the program should be a series of lines printed to the terminal in the format `<number of times word appeared in text file> <word>`.

For this program, words are defined as any sequence of alphabetical characters. Thus, words can be separated by any non-alphabetical character (e.g., a space, a number, a symbol). When counting how many times each word occurs in the file, the program should ignore case (e.g., 'Hello' and 'hello' are the same word).

The file *test_words_file.txt* on the LMS has been provided for you to test your code. Using this file, your program should output:

```
1  a
1  be
1  brown
1  but
1  complete
1  for
1  fox
1  hopefully
1  is
1  less
1  maybe
3  quick
2  task
12 the
1  this
1  to
1  will
1  you
```