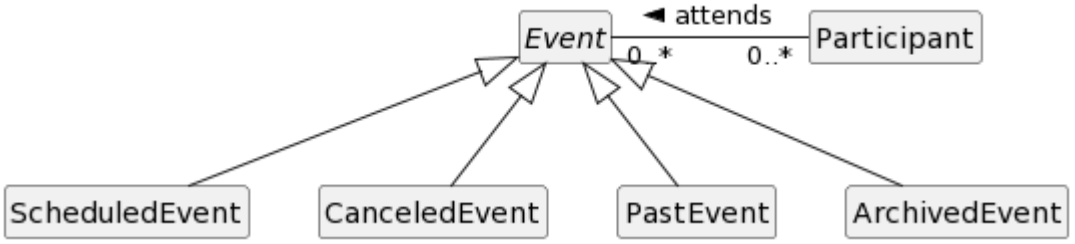
SENG301: Practical Work

Lab 4 2022

Design patterns

1. The 2021 SENG301 assignment involved an application for managing events. Part of the domain model is shown below in simplified form. We will explore some ways in which GoF design patterns can be used in extending the application.



Design notes: Individual "one off" events each have a cost which is made up of the ticket price (which varies according to the event) and a fixed booking fee (which is the same for all events. For example, the cost of *The Waggles on Tour* is $80 — $75 plus the standard $5 booking fee. Some of our products are actually a series of events. For example, *New Zealand Symphony Orchestra Summer Season* comprises *Bach is Back ($50)*, *Beethoven Bangers ($75)* and *Operatic Overtures ($60)*. A series may also include other series. For example, New Zealand Symphony Orchestra 2022 Season comprises *New Zealand Symphony Orchestra Summer Season, New Zealand Symphony Orchestra Autumn Season, New Zealand Symphony Orchestra Winter Season,* and *New Zealand Symphony Orchestra Spring. Season*.

   a. Consider the above design notes and identify the key elements. What GoF design pattern(s) seems the most appropriate for modelling this domain? Use plantuml to sketch a UML class diagram to help you. Include stereotypes to indicate the rôles of the pattern participants. Discuss your ideas with your neighbours or tutor and resolve any issues that arise.
   b. Implement your design in Java. Use javadoc comments to document the rôles key elements, such as classes or methods, play in the pattern(s). Write some JUnit and/or Cucumber tests to verify that your implementation works as intended.

2. Consider an application for managing events. Didn't we just do that? Yes, we did. For the purposes of this exercise, you can consider starting again from scratch. However, if you're feeling adventurous then read the next exercise before deciding how to proceed.

   Design notes: There are several types of events. Some are online events (e.g. webinars) and others take place in "meat space" — the real world. People who book events are charged a booking fee, which is different for each type. Each event has a

description — a simple string is enough for this exercise, but in reality we would probably want information about the name, description, and location — and a base cost. Clients can book various add-ons such as a merchandise package (T shirt, cap and backpack), catering option (buffet and drink vouchers), or premium parking option. Each add-on has its own additional cost. Some add-ons may not be appropriate for particular events (e.g. a webinar probably shouldn't be sold with a parking package add-on). Some clients may purchase several add-ons for the same event.

   a. Consider the above design notes and identify the key elements. What GoF design pattern(s) seems the most appropriate for modelling this domain? Use plantuml to sketch a UML class diagram to help you. Include stereotypes to indicate the rôles of the pattern participants. Discuss your ideas with your neighbours or tutor and resolve any issues that arise.
   b. Implement your design in Java. Use javadoc comments to document the rôles key elements, such as classes or methods, play in the pattern. Write some JUnit and/or Cucumber tests to verify that your implementation works as intended.

3. Consider an application for managing events. Didn't we just do that? Yes, we did. Twice? Yes.

   Design notes: Now we want to combine the features you've developed in the previous two exercises. Clients will be able to purchase a series of webinars, add a catering option to their *New Zealand Symphony Orchestra Autumn Season* and so on.

   a. Consider the above design notes and identify the key elements. Can you combine the GoF design patterns you used in the previous exercises? Sketch, using plantuml, a UML class diagram to help you. Include stereotypes to indicate the rôles of the pattern participants — remember that a particular class might play a different rôle in each pattern. Discuss your ideas with your neighbours or tutor and resolve any issues that arise.
   b. Implement your design in Java. Use javadoc comments to document the rôles key elements, such as classes or methods, play in the pattern(s). Write some JUnit and/or Cucumber tests to verify that your implementation works as intended.

4. The Alien Abduction scenario is well known around CSSE — primarily as an example of GoF design patterns implemented in Java. One of the patterns it contains is Observer.

   a. Obtain a copy of the source. Import it into your IDE of choice, then build and run it to see what it does. You will have access to the code as a gradle project. The program expects command line arguments so you'll need to run it with a command like: `./gradlew run --args="dave bert ernie"`
   b. Use your knowledge of the GoF Observer pattern to identify classes, interfaces and methods involved in occurrences of the pattern. Document these using the format presented in class.

c. Design a “home brew” version of the GoF Observer pattern. Take into account whether participating elements will be abstract classes, interfaces, etc. Will it be necessary to include data when notifying observers? Record the design decisions you make.
d. Refactor the code to implement your design.

5. How many other patterns can you identify in the alien abduction code? How should you go about looking for them? For each pattern you find:
    a. Document it in the format discussed in class.
    b. Add javadoc comments in the corresponding classes.
    c. Use plantuml to draw a class diagram including information about the patterns.