CSCI 345–Assignment 2
Design

Aran Clauson

Due: February 1, 2019

Introduction

Outcomes

Upon successful completion of this assignment you will

- Understand basic UML Class Diagrams and Sequence Diagrams
- Use at least three of the five class association in a software design:
  - Association, Aggregation, Composition, Implementation, and Inheritance
- Verify that a software design supports the necessary use cases

Problem Statement

Now that you understand what the software should do, you need to figure out how it will do it before you start writing code. Your assignment is to design your software solution to Deadwood™.

Where should you start? Identify the nouns in your system. Rooms, sound-stages, players, scenes, and others all describe parts of the system. Sometimes these nouns represent classes, sometimes they represent interfaces, and sometimes they are just the verbal glue we use to hold a narrative together.

Start putting nouns together. Identify the classes and interfaces. One technique that I find useful is Class-Responsibility-Collaborator or CRC Cards. You can use whatever medium you want, but the original idea was to use note cards. Each card is labeled in the name of a class. The rest of the card is divided vertically into two sections. On the left side list the class's responsibilities. On the right side list the class's collaborators.

If we made a CRC Card for the Minute class in our Alarm Clock problem, it might look something like the following:

Minute	
-Listens for 1Hz ticks -Keeps time for 1 hours -Notifies observers when a minute has elapsed -Notifies observers when an hour has elapsed	Timer Hour Clock

One danger of CRC Cards is over-specifying the solution. The actual Alarm Clock design generalizes a lot of the interactions between these classes with the various Listener interfaces. So CRC Cards are a beginning of a design.

The CRC exercises should give your class names. Start drawing the class diagram and work through your use cases. Make sure that whatever design you have supports the necessary functionality that was described in your analysis. As you design your solution remember the elements of good Object Oriented Design: encapsulation, polymorphism, and inheritance.

Following the association lines in your class diagram while working you uses cases is valuable, but difficult to describe in a document. Imagine reading the following:

> Each second, the Timer object sends a `tick` event to each subscribed Listener. A Minute object, which is a Timer.Listener, receives this message and updates its internal counts. When the Minute object overflows—that is, an hour has passed—it sends an `overflow` message to its subscribed Listeners. ...

Alternatively, we could use the UML Sequence Diagram shown in Figure 1. This is one of several diagram in the sample design document.

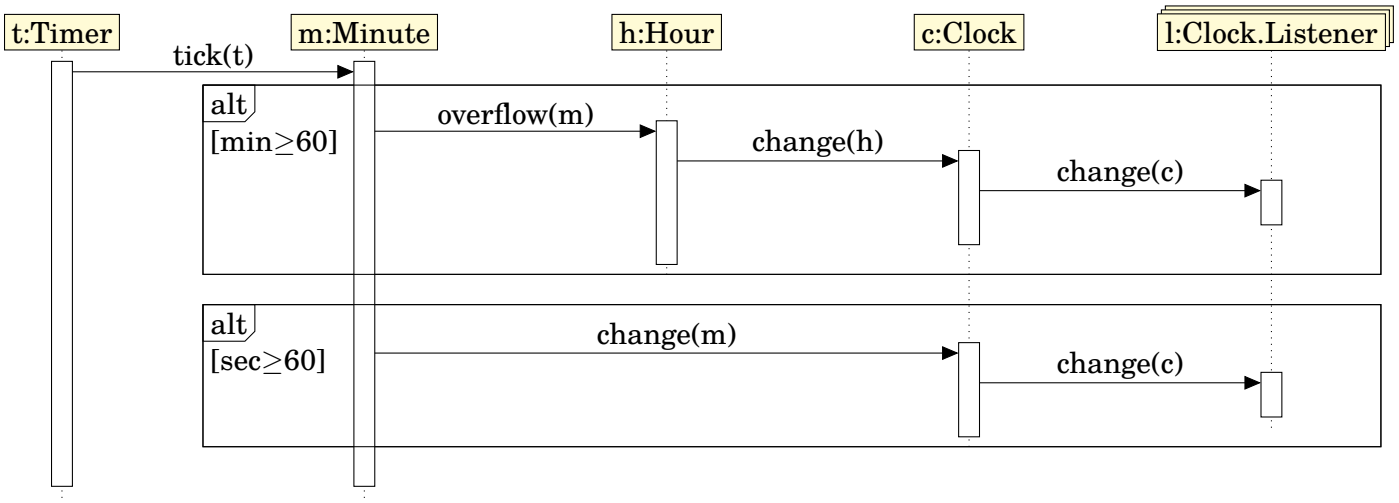

Figure 1: Timekeeping Sequence Diagram

Time flows down the Sequence Diagram. That is to say, events toward the top of the diagram occur before events toward the bottom. Each arrow denotes a method call between classes. For those method calls to be meaningful, the calling class must have some association with the class being called. Looking at Figure 1 again we see that some Timer, t, sends a `tick` message to a Minute, m. We can look at the class diagram (in the sample document) and see that this is through Timer's Listener interface.

All of the illustrated method calls have a void return type. Generally there is no return illustrated in the diagram. If, however, a method returned a value, it would be denoted as a dotted arrow returning to the caller at the end of the activation box. For example, if Minute returned some value, say 1, to the Timer, there would be a dotted arrow toward the bottom of the diagram from Minute to Timer labeled 1.

The goal of all of these diagrams is to describe your design. This ensures that the design actually solves the problem. How do I know if you have designed a system that successfully solves the Deadwood™ problem? You are going to turn in a design document.

Deliverables

Write and turn in a software design document that illustrates your solution. I have posted a sample document for the Alarm Clock on Canvas. Your document should be formatted similarly. The outline for your document is the following:

1. **Introduction:** Your introduction should give the reader context about your design. If you have an introduction that you like from the previous assignment, feel free to use it again.

   (a) **Purpose:** Define the purpose of this document. Again, think about your reader. The introduction tells me what problem is. Now tell your reader that you are going to solve it.

   (b) **Scope:** What is in scope and what is out of scope. You are not going to design the entire software package here. Like the Alarm Clock example, your goal is to design the business logic. We will work on the presentation later in the quarter.

   (c) **Organization:** Give your reader an idea of the document's structure. In the example, I divided the document into structures and behaviors. This may not work for Deadwood™. If you have a better organization, feel free to use it, but tell me what it is. I should not have to guess.

2. **Body:** Here is when where you describe your solution. Please use copious Class and Sequence Diagrams. You should, at the very least, show me that your design will support your use cases. For example, a player shouldn't be allowed to move between non-adjacent rooms. How will your design know which two rooms are adjacent to each other.

Grading

- Complete (e.g., no missing classes)
- Correct (e.g., all uses cases are supported)
- Consistent (e.g., symbol names agree between diagrams)
- Presentation (e.g., clean, easy to read, professional looking document)