SWEN30006 Assignment 2 - Report

Workshop 09, Team 02
Lucas Fern & Cameron Maddern

May 26, 2021

This project required changes to be made to an existing Cribbage card game trainer system to add scoring and logging functionality. These additions were made with the use of a variety of design patterns in order to have minimum impact on the existing code, and therefore reduce coupling between existing classes and classes added for the new functionality.

This report will discuss the changes which have been made, provide justification for the design patterns used, and argue in favour of these design patterns over the use of others where appropriate. The additional classes will be covered first, then how these additional classes were integrated into the existing system.

A class diagram of the updated design is included at the end of the report and will be relevant to visualise all changes to the system.

1 Cribbage Observers

In an effort to minimise coupling between added classes and maximise cohesion within them, a `CribbageObserver` interface was implemented. This allowed classes to subscribe to various events of the Cribbage game, and respond with unique actions. This required the addition of a list of subscribers in the main `Cribbage` class, as well as a method to register new subscribers.

1.1 Cribbage Events

Now that a publish-subscribe pattern is implemented it must be decided when events are broadcast to the subscribers. There are a variety of events in the game of Cribbage, and it was decided that all events which relate to the logging and scoring functionality will be broadcast to the subscribers. This allows - for example - the logging functionality for the `deal` to be implemented by an observer responding to a `Deal` event. The various subscribers to Cribbage Events are discussed further in Section 1.2.

The complete list of `CribbageEvents` implemented appears on the right side of the class diagram, and their functionality is briefly summarised below:

- `SetSeed`

  Subscribers are notified of this event when the game's seed is set from the `cribbage.properties` file. The event contains the random seed which was chosen.

- `InitPlayer`

  This event is broadcast when the players of the game are initialised and assigned a number. This provides subscribers with information about the player type which has been initialised (*eg.* `cribbage.RandomPlayer`) and their number.

- `Discard`

  This event is created once for each player each game, when the player has selected which cards to discard. It provides observers with the players number, and a `jcardgame.Hand` object containing the cards they selected for discarding.

- `PlayStarter`

  The `PlayStarter` event is published once per game when the starter card is selected, and provides the starter card as a `jcardgame.Card`.

- `Play`

  This event is raised on every turn of the game, when a player selects the card they wish to play. It contains the player's number, the card they played, and the total face value of the current board after adding this card.

- `Show`

  The `Show` event occurs at the end of the game when play has ceased and players are showing card combinations from their hands to be scored. This event contains information about the player's number, the starter card, and the cards that they are showing to be scored.

- `Score`

  The `Score` event is a special type of event raised by the Cribbage Scorer in response to other events, it occurs whenever a player's score is incremented, and contains data on the amount of points scored, their total score, and the type of score achieved. This will be discussed further in 1.2.1.

Each of these events also overrides the default `toString()` method. Their implementation of the method returns a `String`, formatted as required for the logging functionality. This will be discussed further in Section 1.2.2

1.2 Cribbage Subscribers

Subscribers are classes which implement the `CribbageObserver` interface and register with the `Cribbage` class to be notified of game events. Their creation and registration is done inside the `Cribbage` class since currently the only subscribers are the Logger and Scorer, which was not complex enough to justify their creation in an external factory.

1.2.1 Cribbage Scorer

1.2.2 Cribbage Logger

The `CribbageLogger` class [singleton?], which appears in the bottom left of the class diagram, is cohesively responsible for all of the logging functionality of the game. As a subscriber to the `Cribbage` class, the logger's `update()` method is called whenever an event (defined in Section 1.1) occurs. The event is passed through to the class in this method call.

On notification of any event, the cribbage logger calls the event's `toString()` method, yielding the information required for logging, and uses a `BufferedWriter` to write this to `cribbage.log`. Taking a `Play` event for example, the contents and respective string representation might be:

```
examplePlay = {                                 |
    eventId: "play",   [String]                 |
    playerId: "P0",    [String]                 |      String representation:
    totalPoints: 23,   [int]                    |      "play,P0,23,KH"
    card: KH           [jcardgame.Card]         |
}                                               |
```

This highly cohesive design, enabled by the observer pattern, is all that is required for the logging functionality.

2 Changes to Existing Classes

Changing existing classes was avoided as much as possible when adding the new functionality to reduce the possibility of bloated classes, and reduce coupling with new classes. This section will provide justification for the few changes which were made to the existing classes despite these considerations.

2.1 Cribbage

The `Cribbage` class is where most of the changes to existing classes were made. The changes made here are:

- **Adding the Subscriber Functionality**

  Since the majority of additions to the system were made by implementing the observer pattern, an attribute had to be added to the `Cribbage` class to store the list of classes which subscribe to the events. A method to register subscribers was also added so that the `CribbageScorer` and `CribbageLogger` could register themselves to be notified.

- **The Registration of Subscribers**

  Since they are separate classes, the `CribbageScorer` and `CribbageLogger` must be instantiated at the beginning of the game, and registered as subscribers to the `Cribbage` class. It was decided that this was an appropriate amount of code (2 lines) to add to the cribbage class, as opposed to creating a factory to instantiate these classes (which would itself need to be initialised on start-up.)

- **Privacy / Visibility Changes**

  To support the functionality of the new system, it was sensible to change the privacy of certain methods.

The most significant change was to each of the 4 `canonical()` methods, which were `private` instance methods in the original design, and have been changed to `public static` methods. They were able to be converted to `static` since the conversion of `jcardgame` objects to `String`s did not strictly require any attributes of the `Cribbage` instance.

This change was made to support the logging functionality, since the log file contains canonical representations of the cards and hands, and the `String`ification of the `jcardgame` objects occurs in the `toString()` methods of each of the `CribbageEvent`s. Considering the alternative would be to pass string representations as well as the original `jcardgame` objects around with each event; and the fact that there is no clear downside to this change, this was an obvious choice.