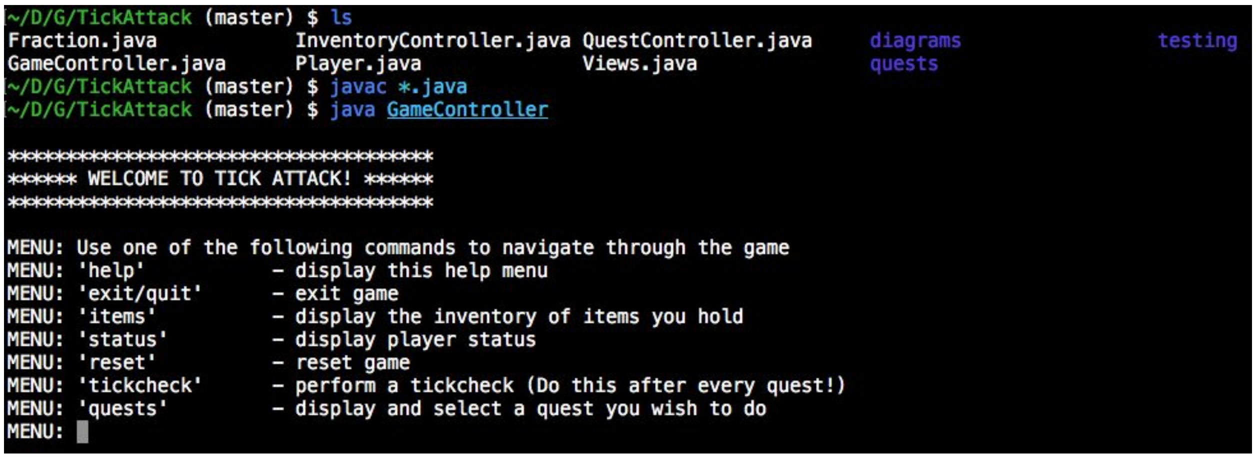
Drew Chaboyer
William Fiset
COMP 3721

Milestone 4 - Tick Attack Game

How to run game

You can begin running the game by compiling all the java files and then executing the *GameController* class which contains the main method. The game contains a help menu with further instructions that can help eliminate any confusion. Here is a picture of what you will see after starting the game:


```
~/D/G/TickAttack (master) $ ls
Fraction.java           InventoryController.java QuestController.java     diagrams                testing
GameController.java     Player.java              Views.java               quests
~/D/G/TickAttack (master) $ javac *.java
~/D/G/TickAttack (master) $ java GameController

************************************
****** WELCOME TO TICK ATTACK! ******
************************************

MENU: Use one of the following commands to navigate through the game
MENU: 'help'          - display this help menu
MENU: 'exit/quit'     - exit game
MENU: 'items'         - display the inventory of items you hold
MENU: 'status'        - display player status
MENU: 'reset'         - reset game
MENU: 'tickcheck'     - perform a tickcheck (Do this after every quest!)
MENU: 'quests'        - display and select a quest you wish to do
MENU:
```

Decision for selecting TAC

When selecting which project we were going to choose, we had two options to go with, either we selected the Tick Attack (TA) project of Olivier and Drew (TAOD: TickAttack Ovlivier & Drew) or the TA project of William and Jonathan (TAWJ: TickAttack William & Jonathan). Both projects had their strengths and weaknesses:

The primary advantage of the project that Drew and Olivier created was its strong use of the Model View Controller paradigm. However, it have the disadvantage of requiring more work to implement a quest. This is because the QuestController in TAOD handles each quest differently, specifically you need to add functionality to the controller each time the programmer wishes to implement a new quest. TAWJ on the other hand used a streamlined standardized procedure to handle all quests via .quest files thus requiring little fuss to add a new quest.

One weakness of selecting TAWJ was that we had difficulty finding a structural design pattern we could easily incorporate into the game. This difficulty actually also reflects a strength however: the code in TAWJ in general reflects a high level of sophistication and already adheres to many coding principles. This quality standard in the end is why we chose his project, and with some extra thinking we were indeed able to discern an appropriate place to add the decorator structural design pattern.

Incorporation of new quest into TAC

TAWJ elegantly abstracts the idea of a quest. Each quest is represented in the form of a graph, with each quest receiving its own .quest file storing all relevant information about how a quest will unravel itself. The data stored in these files adheres to a standardized structure, that Will taught Drew. After learning the proper syntax, it was a relatively easy affair for Drew to transfer a quest from his TickAttack project. No modification of Will's java files was necessary.

We chose to adapt the smuggling quest from TAOD, as that is the more interesting of the two in Drew's old project. Adapting this quest required some modifications. First of all, TAWJ runs in a fundamentally different way: its decisions are decided by random number generators instead of user input. This means in the smuggling quest the computer now chooses which item the user buys and whether the user bribes the border guard or not. It is important to note that this is a difference in how the quests functions between the two games, but within TAWJ there is no implementation difference between the smuggling quest and the other quests. There was one other significant modification Drew had to make: players in Will's game do not have a lyme disease status, thus Drew replaced the lyme disease detector item with one that provides a bonus to the player's health (an arbitrary choice for a different item). Drew also modified the money costs associated with all actions to bring them more in line with the flow of money in Will's game.

Implementing the quest began with drawing on paper a graph that corresponded to the smuggling choices and all possible outcomes. Once this was done it was a matter of incorporating the quest's text from Drew's TickAttack files and carefully creating the nodes and edges in the .quest file.

Structural Design Pattern

We decided to adopt the decorator design pattern into TAWJ to add extra flexibility when creating multiple different kinds of views. The decorator pattern (also sometimes called the Wrapper pattern) allows you to take an already existing class and extend its functionality by 'wrapping' it in a subclass which contains the additional functionality you desire.

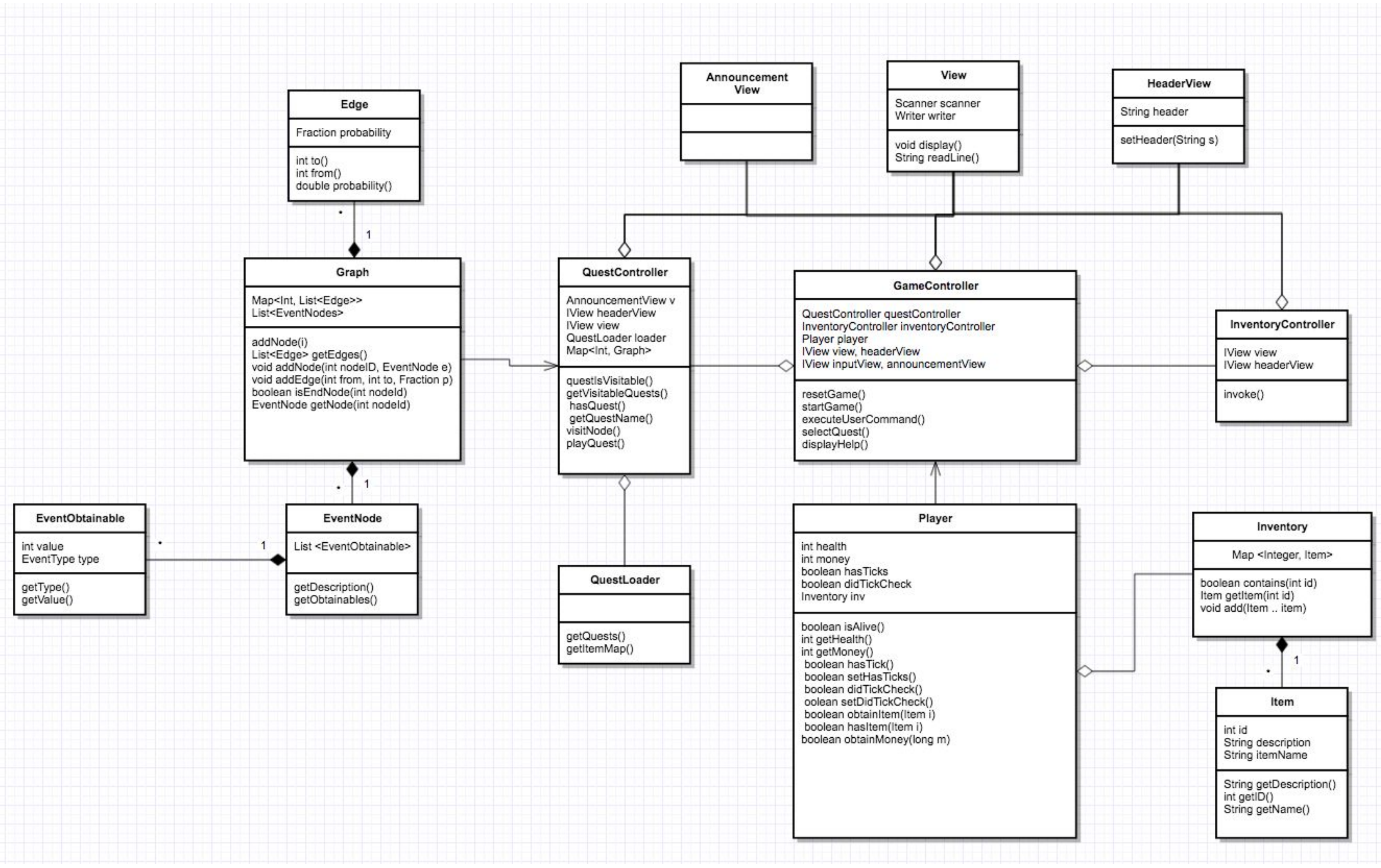
In the case of the *View* class and its two subclasses *HeaderView* and *AnnoucementView* we decided that it would be appropriate to instead decorate the *HeaderView* and the *AnnouncementView* for the following reasons:

1) Both of these classes only differ from their superclass by one distinct feature.
2) There is a lot of repeated code between the *View* class and the *HeaderView* and *AnnouncementView* class that could be simplified by using the decorator pattern.

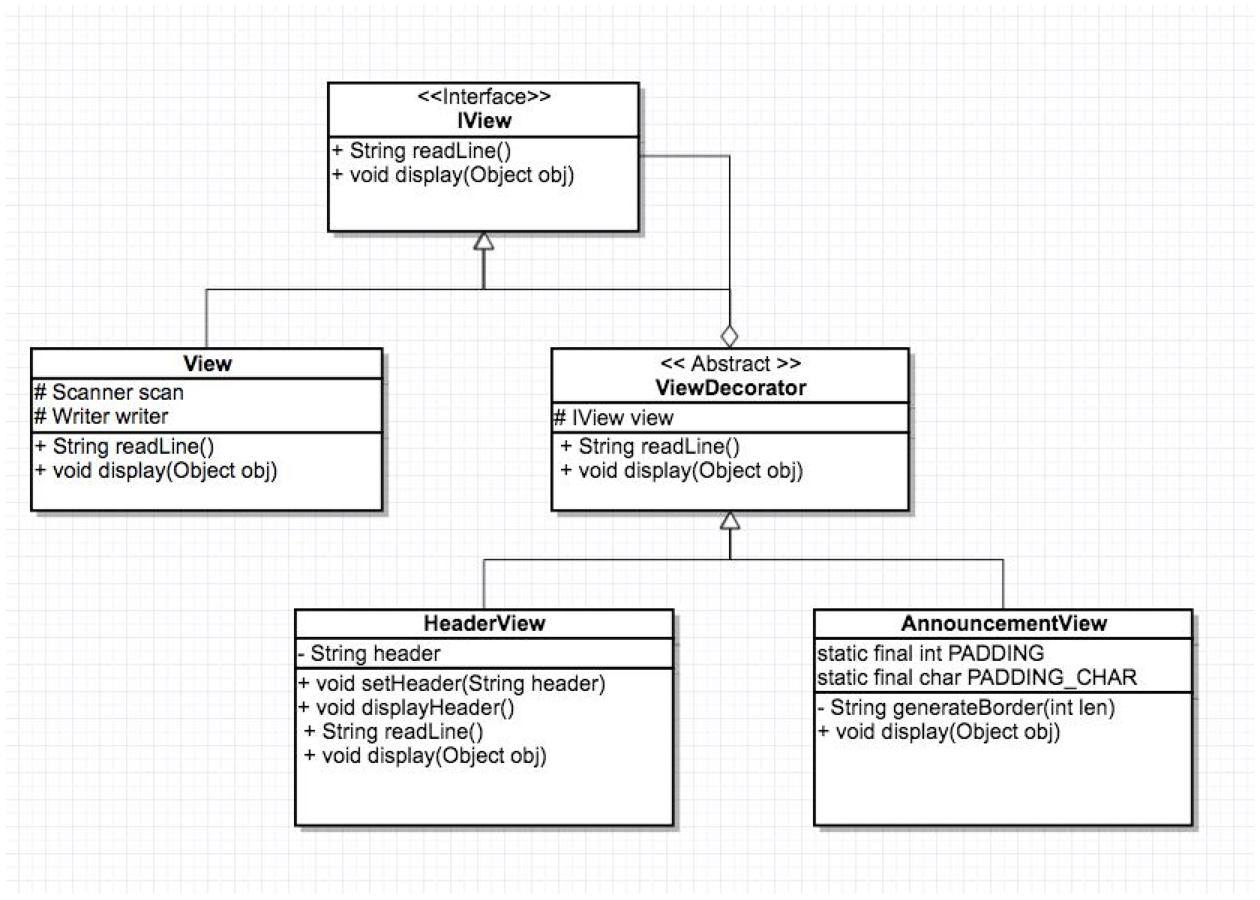
The end outcome of using the decorator pattern with the *HeaderView* and the *AnnouncementView* was that we were able to reduce the number of lines of code used for both the *HeaderView* class and the *AnnouncementView* class. Furthermore, the code looks a lot simpler and is now very extendible, especially if the requirements for the view change in the future.

Updated UML Diagram

Old UML diagram:



New component in UML diagram (the rest did not change):



<<Interface>>
IView
+ String readLine()
+ void display(Object obj)
View
# Scanner scan
# Writer writer
+ String readLine()
+ void display(Object obj)
<< Abstract >>
ViewDecorator
# IView view
+ String readLine()
+ void display(Object obj)
HeaderView
- String header
+ void setHeader(String header)
+ void displayHeader()
+ String readLine()
+ void display(Object obj)
AnnouncementView
static final int PADDING
static final char PADDING_CHAR
- String generateBorder(int len)
+ void display(Object obj)