Hold It Detailed Design Document

Team: RKJ

Written By:
Kegan, Ryan, Joseph

Table of Contents

1 Introduction

1.1 Object design trade-offs

1.2 Interface documentation guidelines

- All classes should have to the point noun names.
- All code should be commented in great detail, even simple acts.
- Function names should be descriptive of their purpose.
- All variables should be descriptive of their purpose. All code should be indented using 4 space tabs.
- All variables will have no capitals unless used as an abbreviation
- All functions will have headers
- The closure of all {} pairings will end with a comment stating the end of what they represent.
- The start of every {} pairing should proceed on the line after the line that starts it.
- Comments will be put above the sections of code they pertain to
- Each section of related code in a function will not be separated by whitespace while it is enacting a very specific goal.

1.3 Definitions, acronyms, and abbreviations

Holdee: A client who is holding a spot in line.

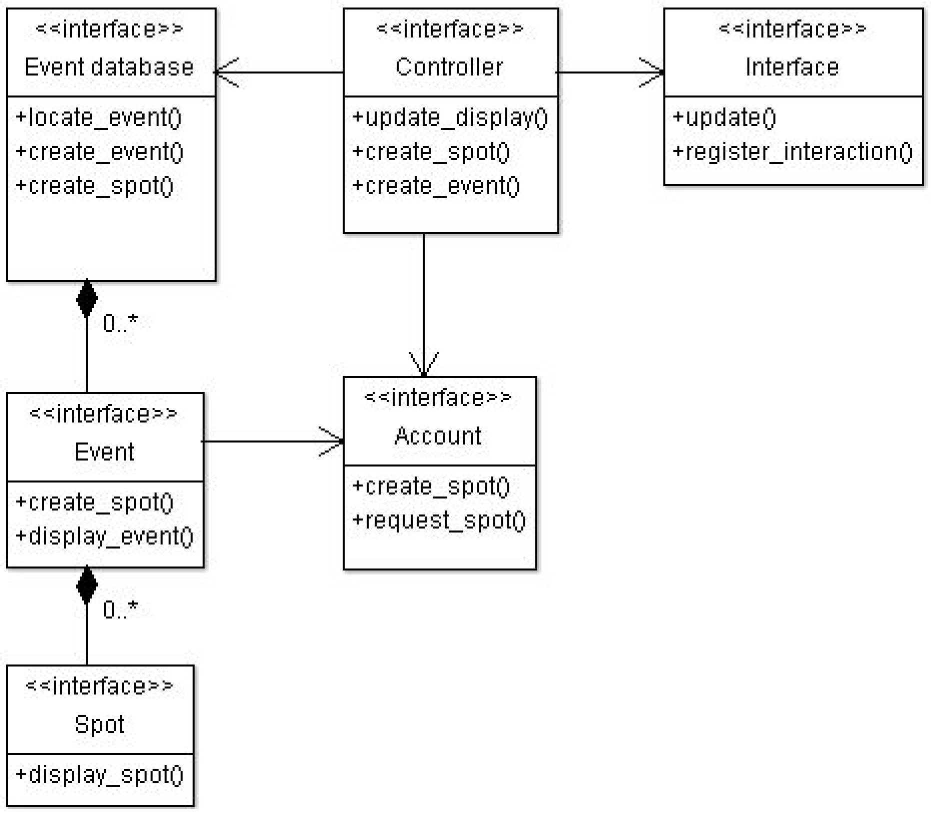
Customer: A client who is purchasing a spot in line

Tag: Label attached to an event to help with identification.

Spot: An offer to hold a spot at an event.

2 Packages

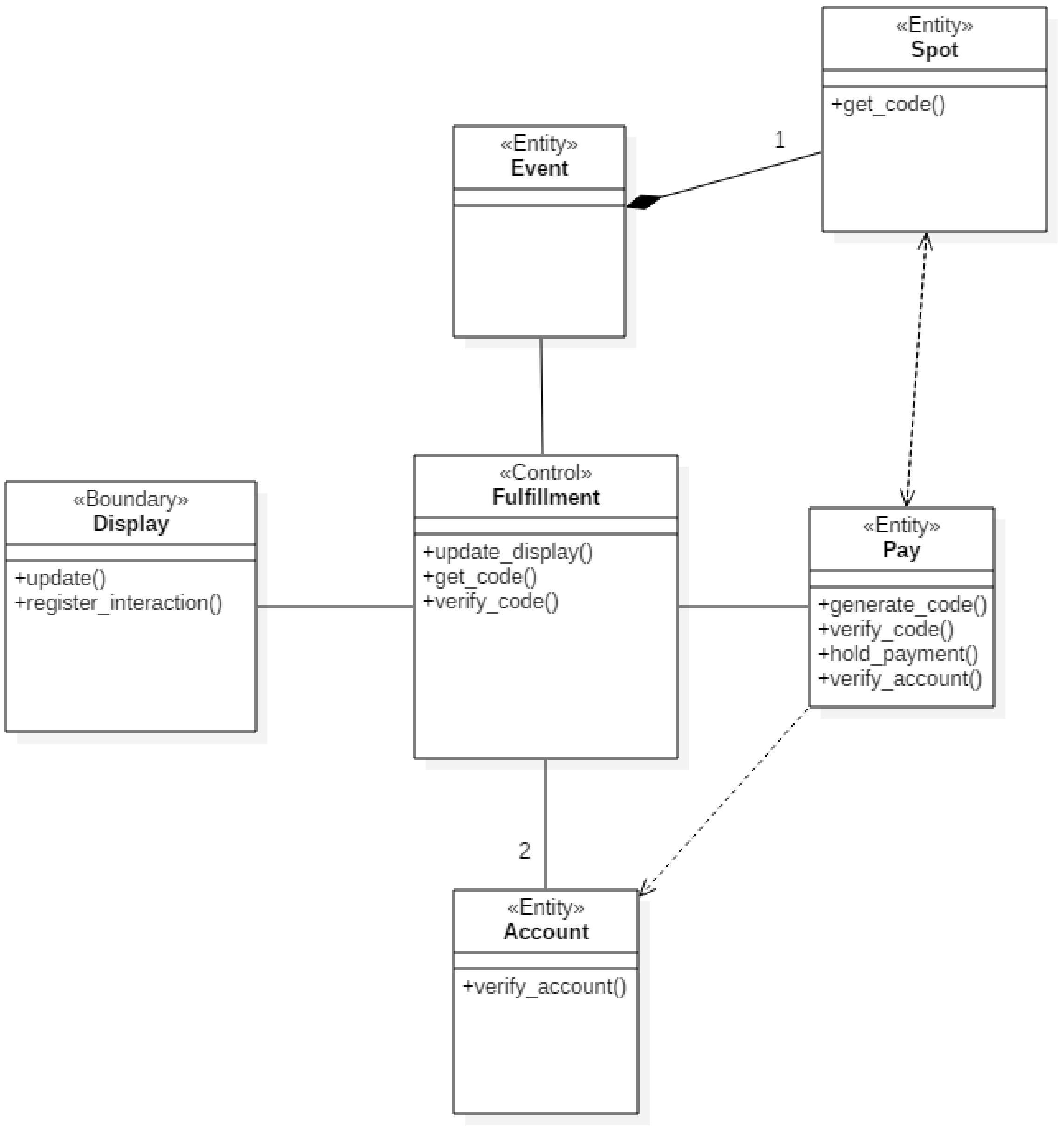
This is the event creation class diagram.



This diagram shows the interaction of the classes that are creating the Event class instance. The user requests a spot creation via the interface, then Hold It checks to see if the event exists, then if it does not it creates one for the Holdee accounts to see if they would like to find a spot to post online.

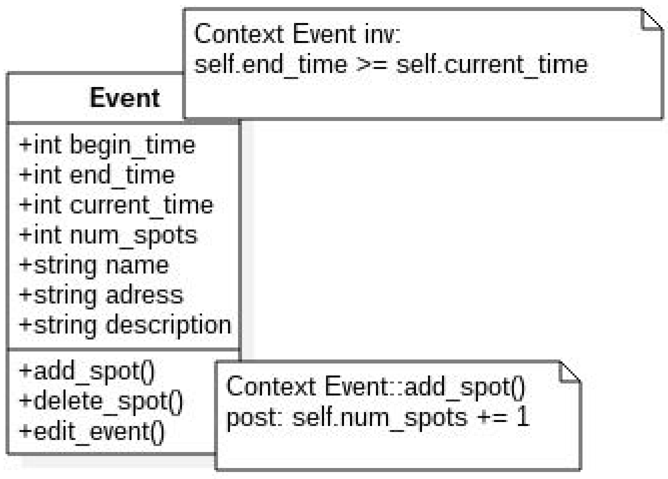
The following diagram shows the classes necessary for fulfilling the holding of spots and for processing payment. Once the Holdee has purchased a spot from a Holder the Holdee's interface will display a six digit code that the Holder will need to enter into their interface to complete the transaction.

Fulfillment of Holdings Class Diagram:



<u>3 Class Interfaces</u>

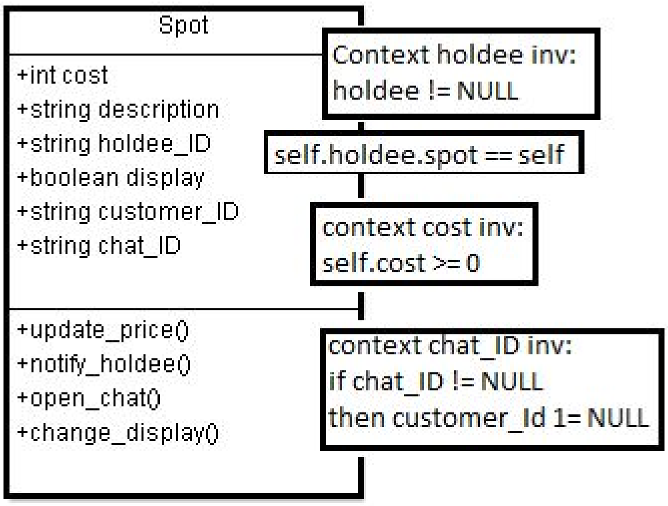
The event class holds information on events and allows for manipulation and creation of events. Events have spots, has direct interaction with the search class, knows about the account class.



The invariant in the event class makes sure the event exists only whilst it is still ongoing. The purpose of this is to not display events that can no longer be accessed in the Hold It application. When the current time passes the inputted ending time of the event it will be deleted.

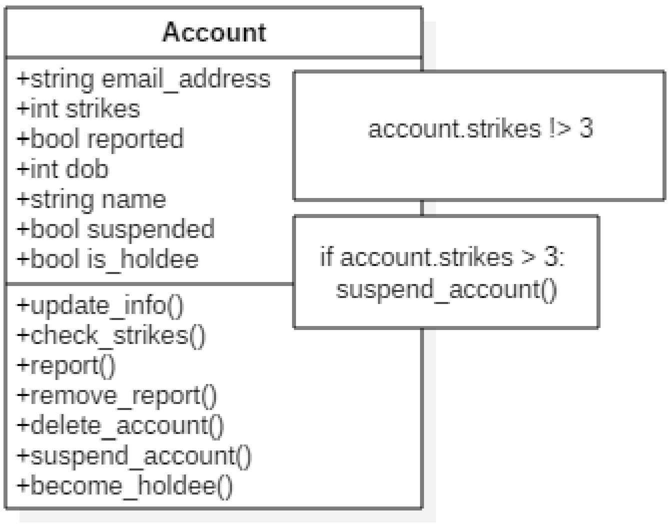
The postcondition for the add_spot function of the event class simply increments the spot counter that the event class already has. This is done so that evens can accurately display the number of spots they currently have. The delete spot would have a similar decrement post condition.

The spot ties the Holdee to their spot in line. This is exclusively held in the event class, and is referenced in the holdee class. The spot class can only exist inside of an Event class, and can only exist while attached to a Holdee class.



The invariants in this class make it so that there can never not be a holdee attached to this class instance. It makes sure the cost is never a negative number. Finally the it assures that if a customer retrieves the spot a chat is opened between the customer and holdee.

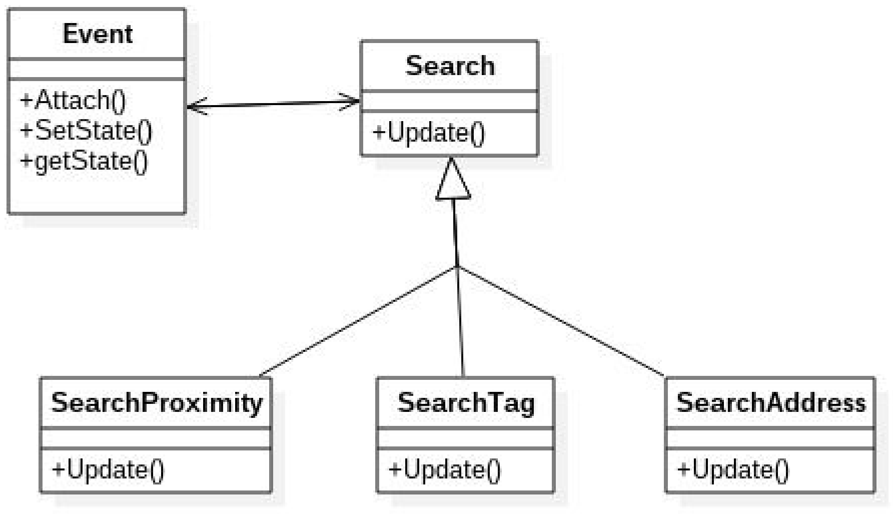
The account class holds all of the information stored by our system for a user's account.



The invariant in this class makes sure no account gets above three strikes. Once an account receives three strikes the account is suspended. Suspended accounts cannot use the services our system provides. The precondition for the become_holdee() function is that the account must not have any strikes.

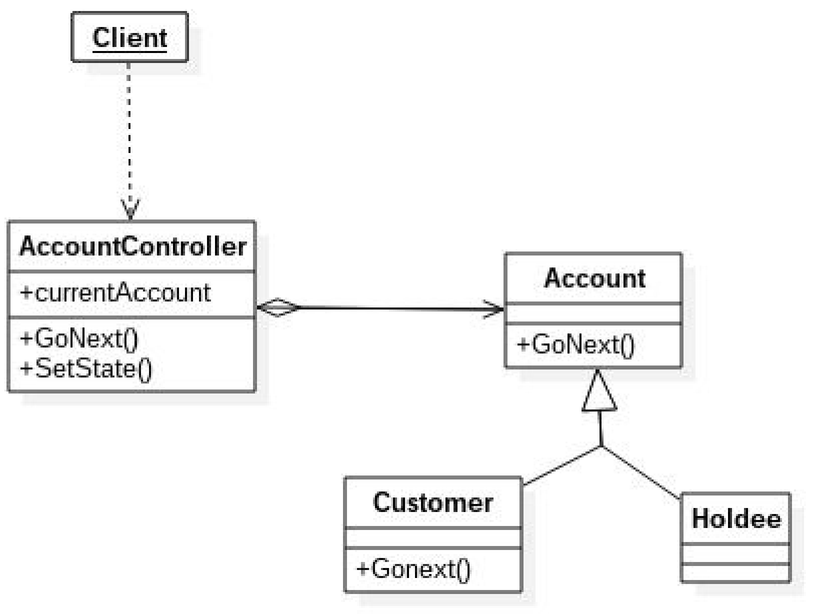
4 Design Patterns

Observer design pattern - Search function



The Hold It application will include three different methods of searching for events. All the different methods for searching will still sort through the same data, but display it differently to the user. This is why the observer design pattern was chosen. When the list of events is changed by having an event added or removed the observing search class will be notified, allowing it to update each of the different views or search methods connected to the search class.

State Design Pattern - Switching customer to holdee



Hold It has two different types of users, customers and holdees. To become a holdee a user has to first be a customer. Once a user is a holdee the state of their application will change giving them access to places they previously did not have access to, such as spot creation. The state design pattern was chosen here because a holdee account will still hold the same information as a customer account, it just needs to change its state in the application.