Casino Simulator Refactor

Paige Johnson, Derek Gorthy, Michael Condon

April 6, 2017

Original Class Diagram

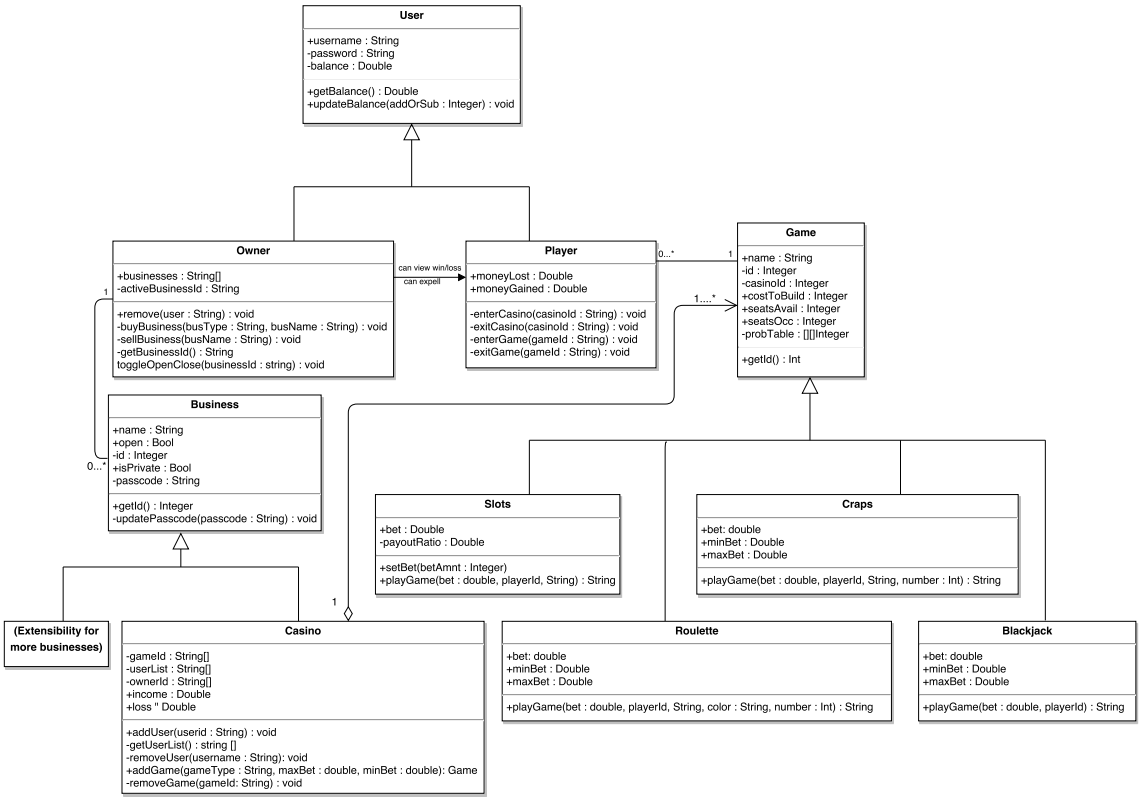

Figure 1: Class Diagram

Refactored Class Diagram

2.1 Modifications

2.1.1 Model View Controller

The feedback that we had on our original class diagram was that we did not have a controller. In order to fix that we created both a controller class, UI classes, and an input handler. These classes, along with User and Business as the "Model Classes" use the MVC design pattern. As the functionality of our project grows, it is likely that additional "View" classes will be added to accurately represent the website.

2.1.2 Decorator Pattern

We realized that having a different subclass for every single game would get extremely cluttered if we wanted to add more games. We decided to fix this by using the Decorator Design Pattern.

This way we can dynamically assign behaviors to each game depending on what we want to load. This design pattern was not an obvious choice considering the limited number of games that our project currently supports.

The Decorator Pattern accommodates the fluidity of existing casinos. Games with the exact same betting and payout specifications can physically be different on a casino floor. For example, a slot machine can have it's own theme, dedicated chair, payout mechanism and maintainance schedule. Our current design oversimplifies the complexities of real-life; however, refactoring our design with the Decorator Pattern incurs the development cost up-front, but will be much easier to add new attributes.

2.2 Diagram

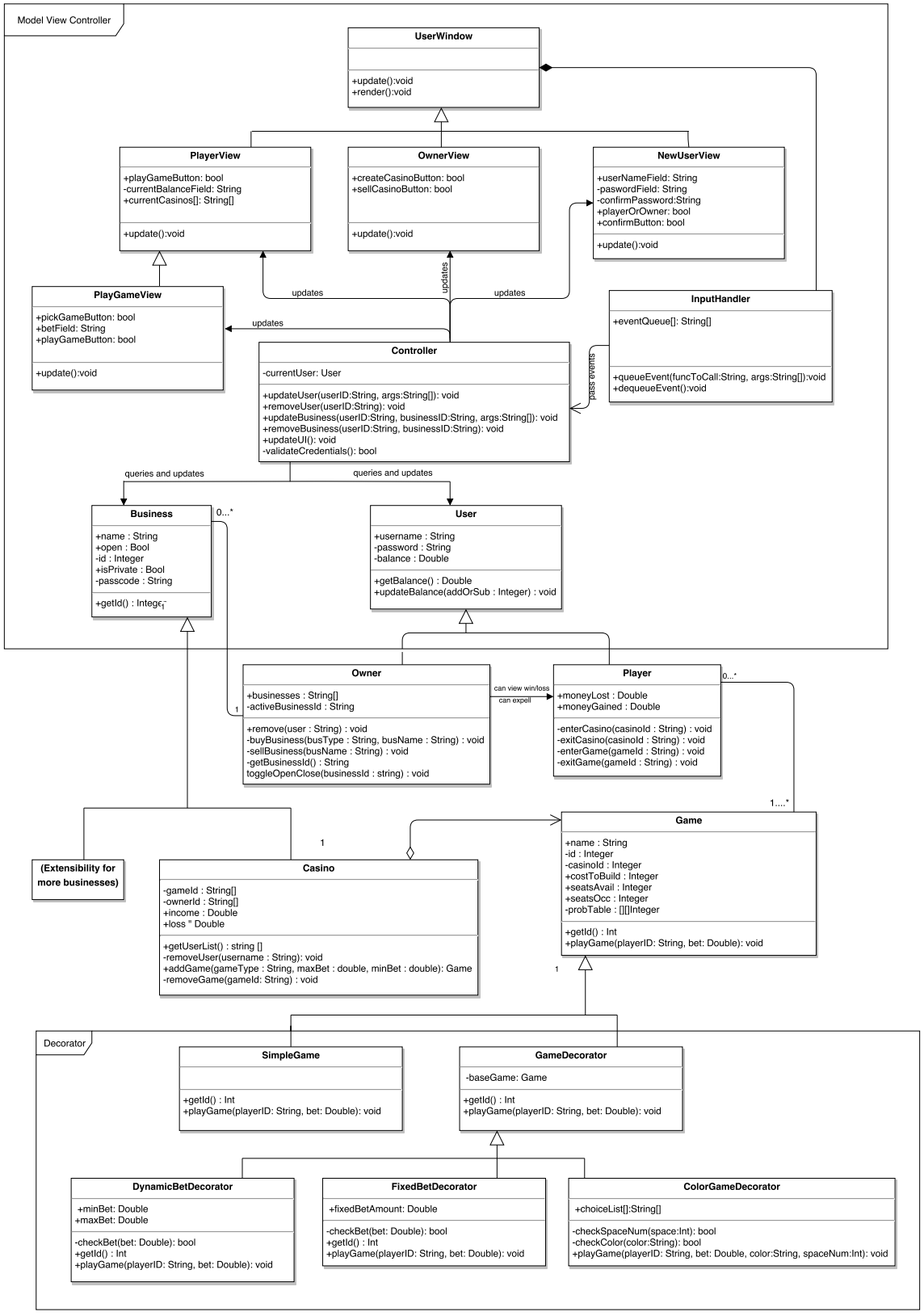

Figure 2: Class Diagram