

# Project Part 3: Refactoring

**Project Title:** Stock Score



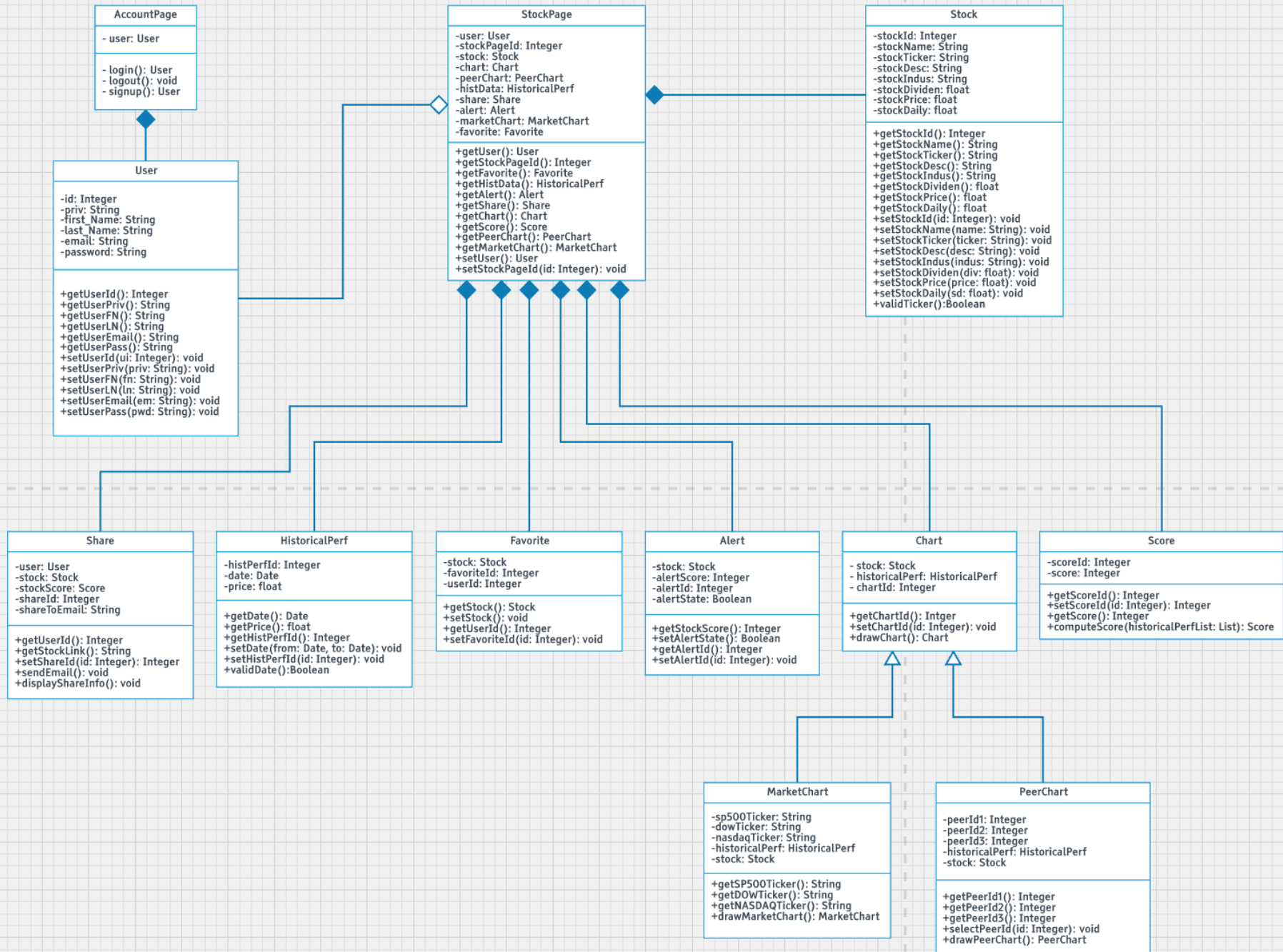
**Team 10:** Douglas Allen

Heesuk Jang

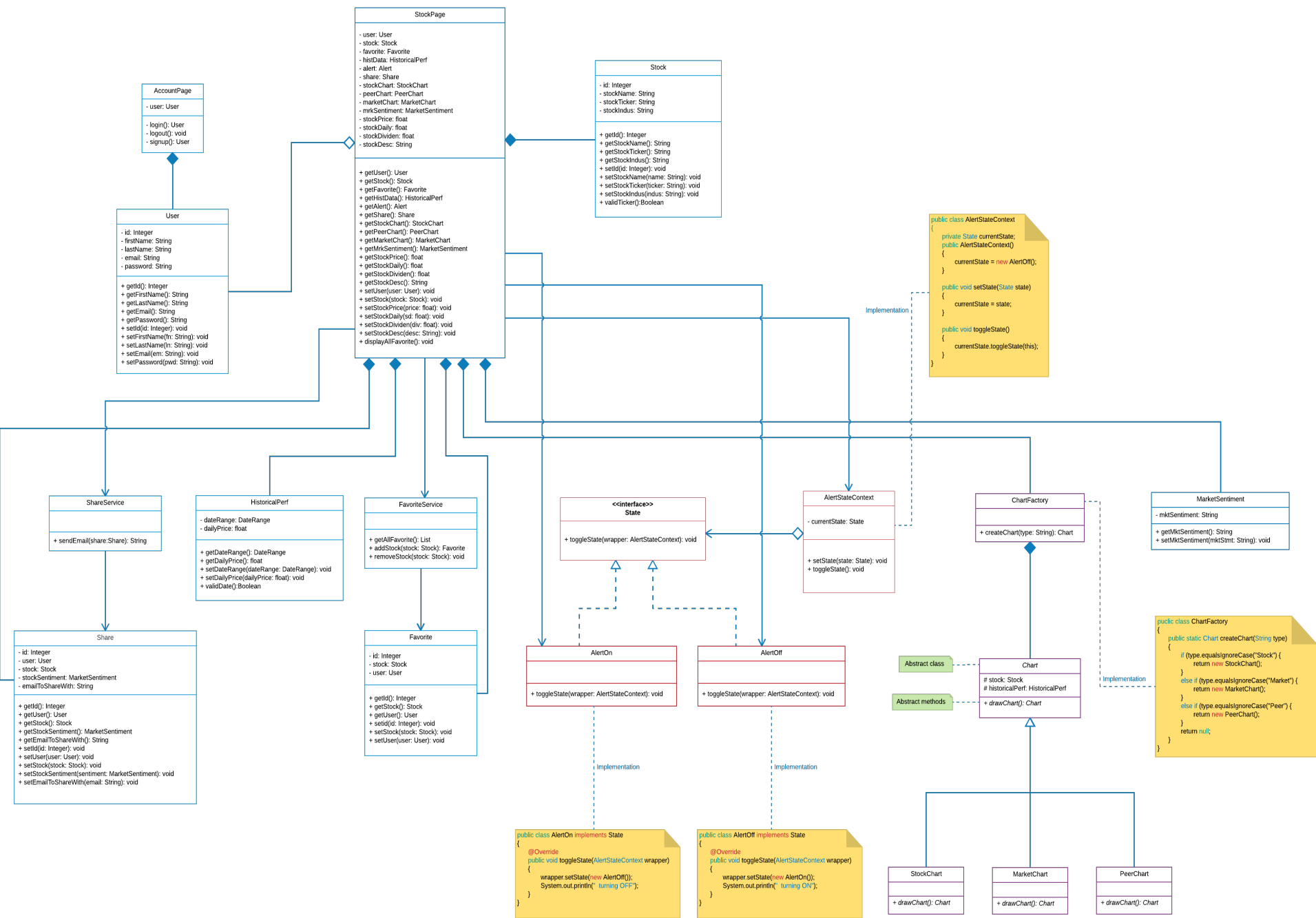
David Zhuzhunashvili

## Old Class Diagram from Part 2:

10\_StockScore: Class Diagram



New Class Diagram:



## Applied Design Patterns

### Factory Method Pattern:

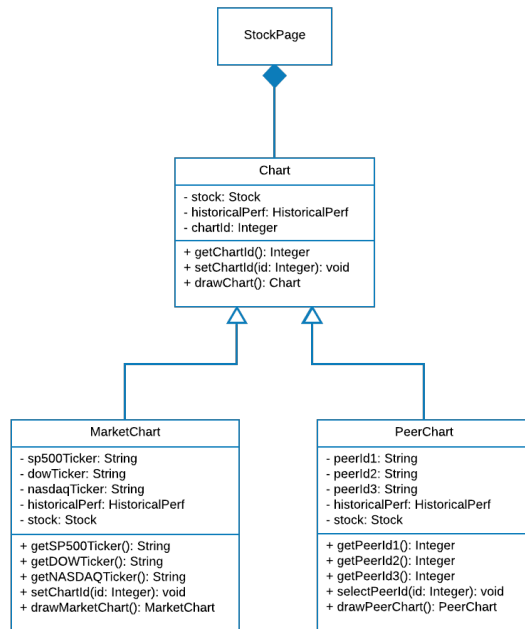
Due to the existence of an inheritance hierarchy among the base class **Chart** and its subclasses **MarketChart** and **PeerChart**, we decided to add a polymorphic creation capability by defining a static factory method *createChart()* in the factory class, **ChartFactory**. The factory method returns three different instances of Chart: **StockChart**, **MarketChart**, and **PeerChart** and allows the subclasses of the abstract class, **Chart**, to decide which type of concrete Chart to create by overriding the abstract method, *drawChart()*, declared in the class **Chart**.

In this way, **StockPage** (as the client) only needs to call the generic function of *createChart()* in the factory class as a following example when new charts need to be added without affecting **StockPage** or **ChartFactory**.

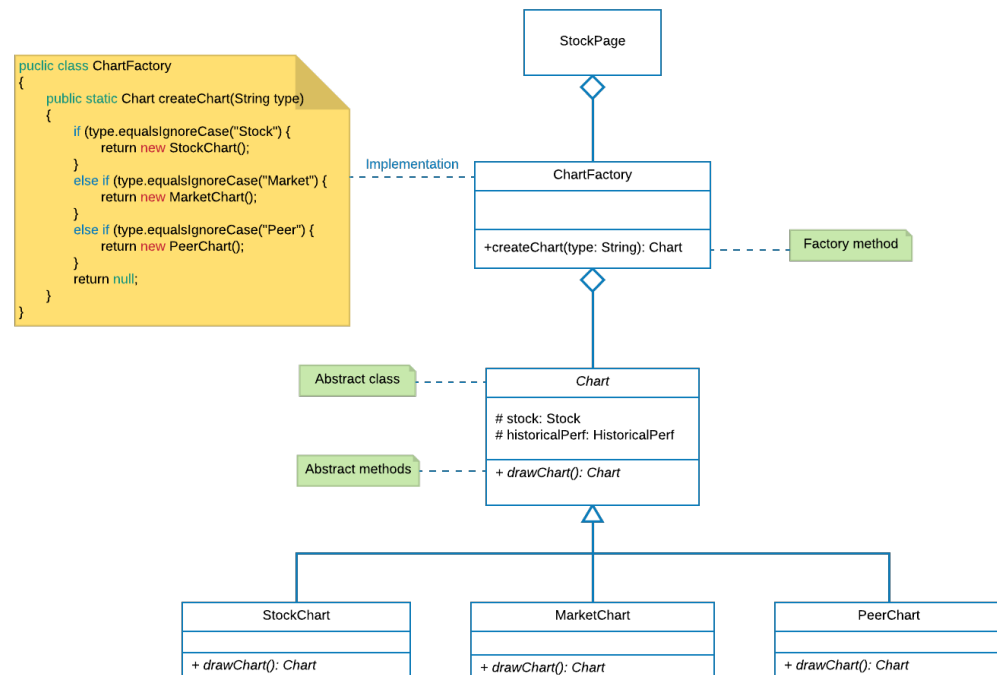
```
Public class StockPage
{
    public static void main (String[] args)
    {
        Chart sp500Chart = ChartFactory.createChart("Market");
    }
}
```

For more detailed implementation of the changes, please look into the following diagrams.

#### [ Original Class Diagram ]



#### [ After Applying Factory Method Pattern ]



## State Pattern:

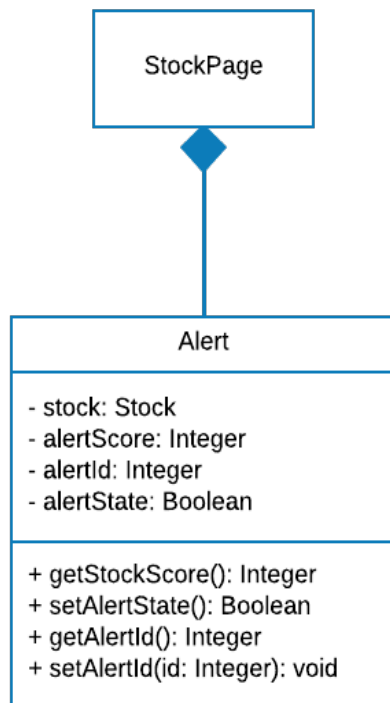
To allow the object **Alert** to alter its behavior when its internal state changes from **OFF** and **ON** and vice versa, we applied State Pattern to create two new classes, **AlertOn** and **AlertOff** of the context object, **AlertStateContext** and to extract the state-related behaviors into these classes.

**AlertStateContext** contains an instance of a concrete State subclass, **AlertOff**, that defines the *currentState* and the **AlertStateContext** will delegate the execution to each state object, **AlertOn** and **AlertOff**. The class **AlertStateContext** defines the interface **State** of interests to **StockPage** and **State** encapsulates *toggleState(wrapper: AlertStateContext)* behavior associated with a particular state of the **AlertStateContext**.

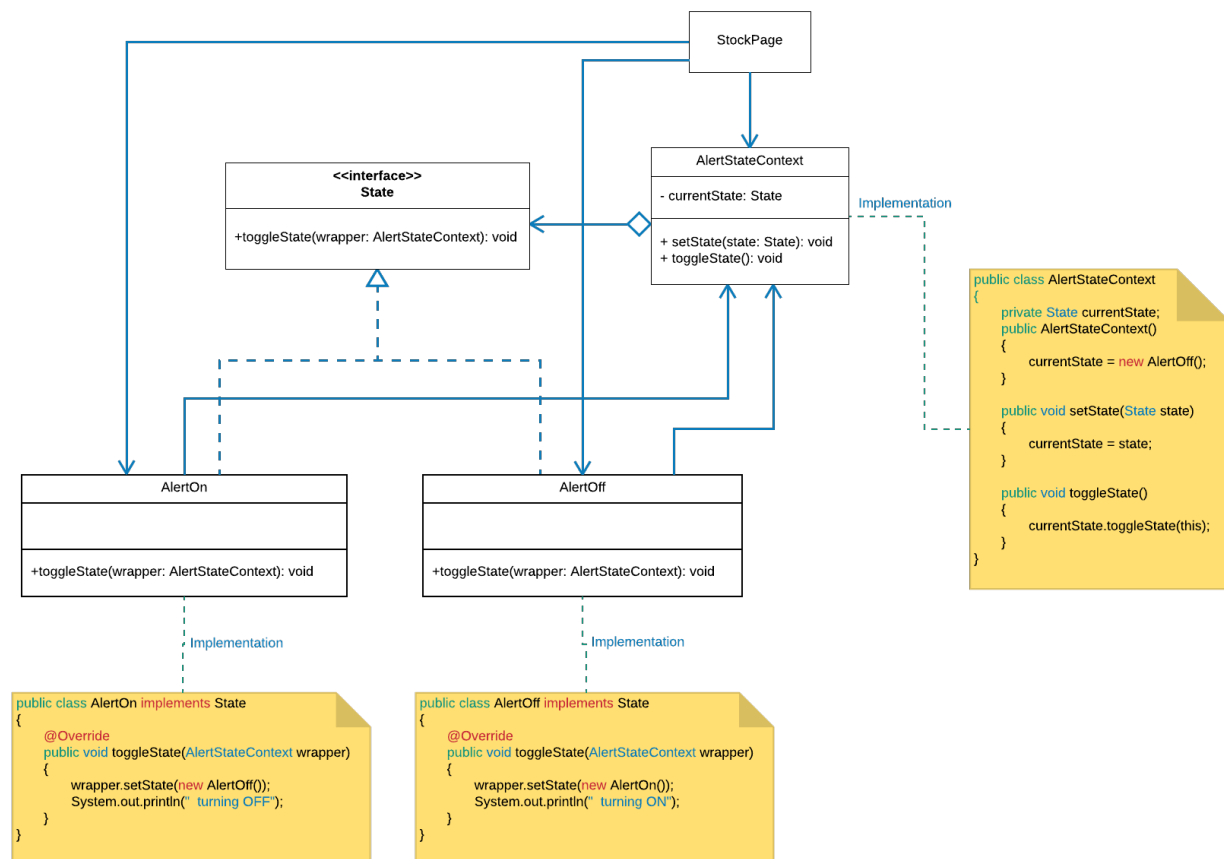
Both **AlertOn** and **AlertOff** subclasses implement a behavior associated with On or Off state of the **AlertStateContext** by overriding *toggleState(wrapper: AlertStateContext)* method, which follow the common interface **State**.

For more detailed implementation of the changes, please look into the following diagrams.

[ Original Class Diagram ]

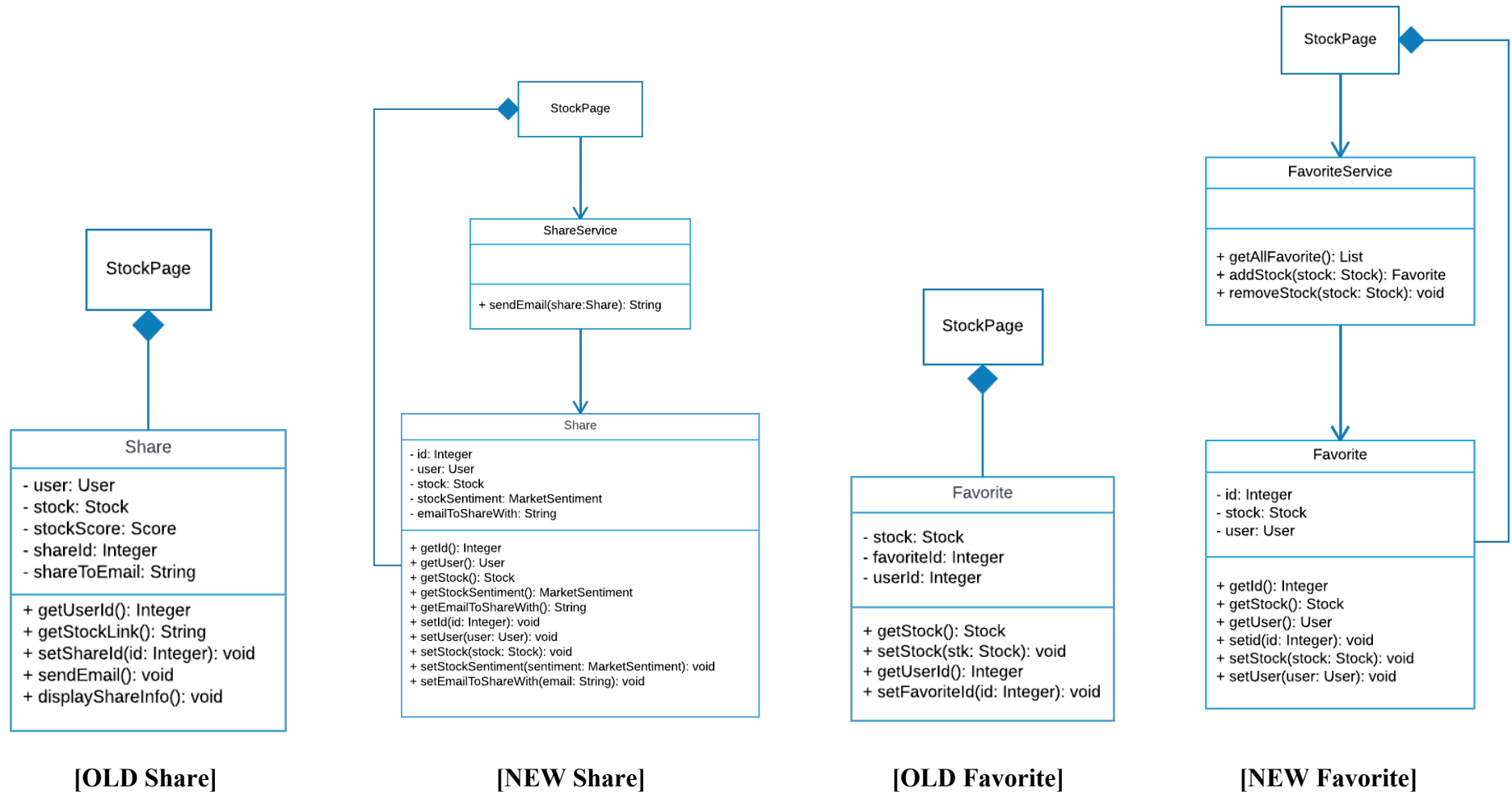


[ After Applying State Pattern ]



## Other Refactoring Changes

- ✓ We separated the entity and the actions on the entity into two separate classes. The entity for Share or Favorite only contain the getters and setters. Actions such as `getAllFavorite()` was moved to the `FavoriteService` class. This follows the single responsibility principle.



- ✓ We removed storing information for stocks that can easily be retrieved via APIs and may change so that we avoid stale data. For example, we moved `current stockPrice` from the class `Stock` to `StockPage` due to its frequent changes. `StockPage` will get the `stockPrice` from an API.