Fida Matin a1798239 - Matthew Parsons a1801482 - Kainalu Peters a180413
OOP Semester 1, 2021 - The University of Adelaide

Project Specification - Major Practical

Introduction

This project utilizes the concept of a stock market portfolio. Users have a starting budget and can buy/sell stocks weekly as they see fit. Each week there will be a new event occurring in the world which will affect different sectors and some individual stocks, users must decide how the event will affect their stocks and act accordingly. Users have a year to maximise their profits.

Design Description

Assessment Concepts

Memory allocation from stack and the heap

- **Arrays:** Use of both dynamic and static arrays will allow for storage of stocks and current stock prices. On top of this there will be arrays to store which stocks the user has purchased and at which price.
- **Strings:** Storing names of stocks, name of player, buy/sell options and user inputs
- **Objects:** Events, Stock market, portfolio.

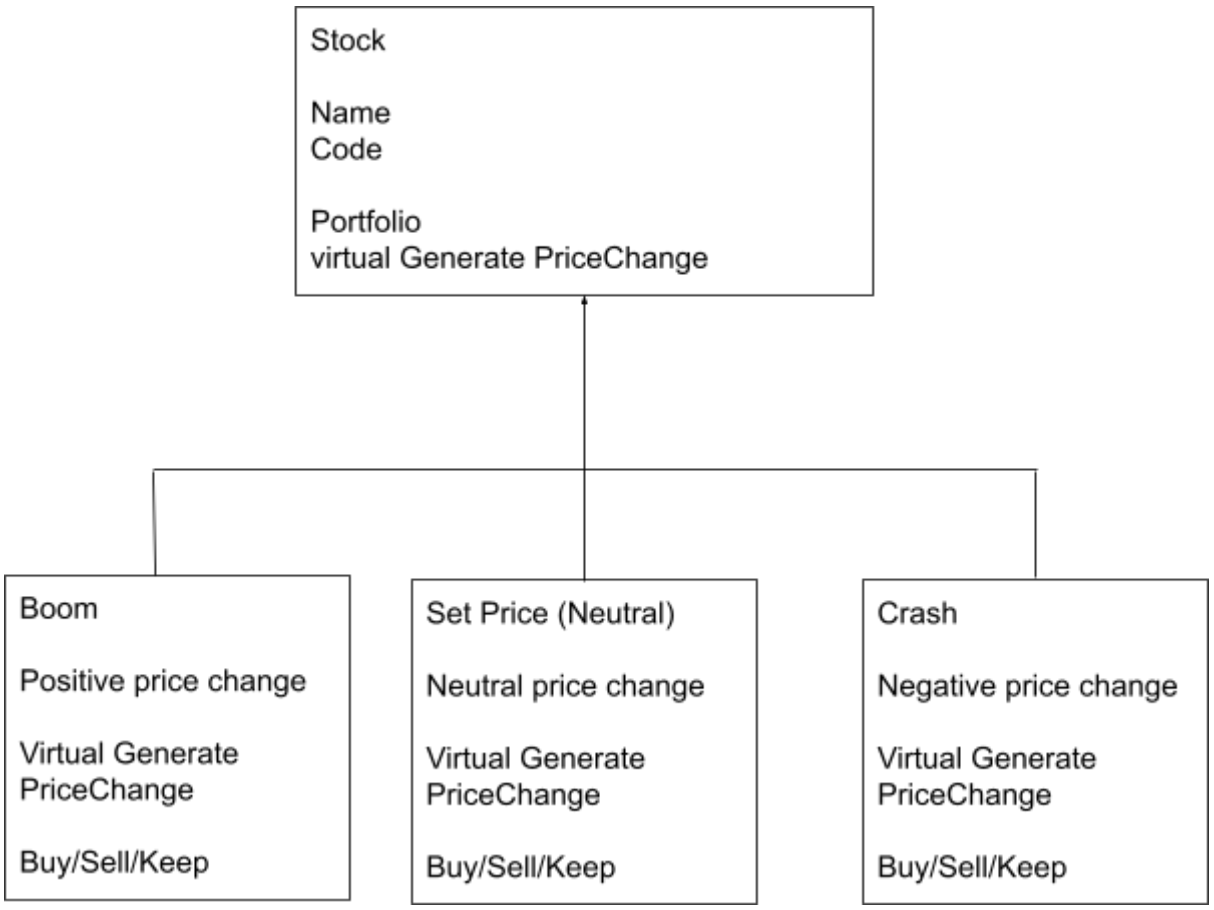
User Input and Output

- **I/O of different data types:** Command-line, users enter decision to buy/sell using given stock codes, users also enter how many stocks they wish to buy.

Object-oriented programming and design

- **Inheritance:** Events have different types which alter the stock price in unique ways. Things such as the Suez Canal being blocked up, which in turn causes the oil prices to decrease. Stock prices also alter the profit of the user.

Class Diagram



Class Descriptions

Neutral (set price)

Abstract base class. Price remains constant, stock does not change. It checks the previous or set price and maintains the price. All stocks begin at this state. Checks whether the stock is already bought and how many stocks are owned. Behaviours in Neutral allow for events to affect selected stocks/sectors by a given percentage. The states will be placed in accordance with the context of the event while the events maintain a sense of ambiguity.

Boom

Boom stocks receive an increase of a given percentage in the stock price. It checks the previous price and increases it by the given percentage. This boom status can be given to both specific stocks and given sectors, i.e. Technology, Medical, Retail, etc.

Crash

Crash stocks receive a decrease of a given percentage in the stock price. It checks the previous price and decreases it by the given percentage. This crash status can be given to both specific stocks and given sectors.

User Interface

A user of this program is playing a “choose-your-adventure” style stock market game. Users are presented with a list of options to choose from:

```
Welcome to the Stock market!
You have $1M, choose your stocks wisely and see how much profit
you can make in a year!

Select a sector:
   1. Energy
   2. Materials
   3. Industrials
   4. Utilities
   5. Healthcare
   6. Financials
   7. Consumer
   8. Information Technology
   9. Communication Services
   10.  Real Estate
```

User enters input (1-10)

```
>> Select stock based on code:


   - Apple (AAPL)
   - Twitter (TWTR)
   - Uber (UBER)
   - Alphabet (GOOG)
```

User enters stock code - checks if user has any stocks, if so ask whether they want to buy/sell otherwise asks if they want to buy.

```
>> BREAKING NEWS! New Privacy laws introduced in the US

Would you like to change your portfolio? (Y/N)

>> Y
```

Starts at select sector as a directory to get specific stocks.

Testing Plan

Unit Testing

Our strategy for testing is to initially get the correct output. This would be testing without inputs from the user to ensure simple units and values will be carried through the code. The main function would purchase stocks under a budget from two classes which are sectors with a set price in the stock market. After the amount and value of each sector purchased and the total value would be displayed as a portfolio. From here the events will occur and be displayed to the user which either increases or decreases the value of a sector. After the events occur the new value of the portfolio will be displayed with their profit or loss. For these tests we will be able to predict the output and clearly see if the code is working correctly in each area as they would be displaying values.

The main function used in this will be an automated testing function. It will test multiple set inputs and allow us to see if our expected output matches the real output. This will be used for the majority of testing so that the calculations and code can be tested separately to testing user inputs.

Input/Output Testing

To begin including user inputs we will add user input to select sectors and purchase stocks from them under a budget in the main function. After each purchase the user will receive information on their new budget and purchase, acting as their receipt.
The user inputs will be conditioned to make sure that the user inputs something the code will be able to receive. This will also need to be rigorously tested as errors can easily occur due to mistyping.

A new aspect of the portfolio would be added, selling. This will be tested by allowing the user to buy or sell their shares in sectors of their choice after events occur after question prompts. A receipt would be generated after purchasing and selling.
The receipt would be an important assistance when testing the code as it would clearly show where values and calculation are incorrect in the code.

Inheritance and Class testing

There will be two inheritance groups, events and sectors, these would be tested one at a time and separate from the main code. It will be tested by a separate function aiming to build different objects with the inheritance so they will have the values required for the code to run. Once the sectors are built they would be displayed for the user with their price afterwards the ability to update and change the prices of the sector would need to be tested.
The events will not change and simply would need to occur and affect the price of the sectors. So to test the events they could be integrated into the testing of the sectors.

Schedule Plan

Stretch Goals

Our goal is to complete project specification, ready for week 9. Once completed we will construct stock objects and events objects and begin creating the inheritance between them. We will expand our program from there by creating a user interface and more sectors and events.

Week 8 - Preparing for assessment in Week 9

Project Specification:

- Makefile (Matt)
- Testing (Kai)
- General Idea - How to implement it (Fida)
- Start writing .h files (Group)

Week 9 - Begin writing code

- Write .h files
- Write object .cpp files
- Begin creating interface

Week 10 - Demonstrate Code Progress

- Integrate .h files and .ccp files to be run by a main function
- Have user input and output throughout the code
- Have adequate interface guidance throughout the code
-

Week 11 - Finalise code

- Finish interface with the set amount of sectors and events running for the given time (1 year)
- Finish portfolio report display of stock prices and profits
- Ensure code displays price changes from price bought