

Spring 2022: Programming Project

Due by 7:00pm on Friday 21st October 2022

Assessment Weight: 40%

A. Requirements

- a) ALL instructions given in this document **MUST** be followed to be **eligible** for full marks for the project. This document has six (6) pages.
- b) This project is **NOT** a group project; collusion, plagiarism, cheating of any kind is not acceptable. As part of your submission, you **MUST** certify that all work submitted is your own. If you cannot honestly certify that the work is your own, then do not submit the project. Breaches of the Misconduct Rule will be dealt with according to the university Rule (see the learning guide for more information).
- c) All project submissions will be checked for academic misconduct by the use of the MOSS program from Stanford University.
- d) Aspects of the project for COMP2016 that are different to COMP2015 are identified in this document in blue text. These aspects are to be implemented by students enrolled in COMP2016 only.

For the problem definition described in section B you must

- e) include your student id at the end of all filenames for all java code files. Two classes have been identified in section B as being required as part of your solution. Do not modify the names of these classes except for adding your student id to the end of the filename. Other Java files will be needed as part of your solution. All Java code files that are used in this project **MUST** have your student id appended to the filename. For example, Driver_#####.java;
- f) include your authorship details at the top of **each** file in code comments (see item 3.1 in Section C of this document for details);
- g) adhere to the coding standard as identified in the Google Java Style Guide (see Section C of this document for details);
- h) ensure that standard console Input/Output are used in all code segments, **do not** use Swing;
- i) ensure that your java code is appropriately modularised for the given problem definition. That is, you need to write appropriate classes and methods to solve the problem;
- j) **reference** all sources that you used for inspiration of your solution as per Section D of this document;
- k) Ensure that your java code compiles and runs in Eclipse installed in the SCDMS labs.

B. Project Details

B(i) - Background information and description

By definition, a shareholder owns one or more shares of stock in a public or private corporation. A shareholder may own shares in one or multiple corporations, this is known as a portfolio. Shares of stock for a corporation are traded (bought and sold) on the stock market where the price of the shares fluctuates due to various influences including supply and demand. A shareholder may engage the services of an agent, commonly known as a broker or stockbroker, to trade shares on their behalf. The stockbroker maintains records of their customers, their portfolios, shares traded, and charges the shareholder a fee for each of the trades they transact on their behalf.

A Sydney stockbroker, Ms Dee Zaster, wants you to create an object-oriented Java program that can be used to maintain records of her customers, their portfolios, shares traded, and to generate portfolio reports.

The specific functional requirements of the Java program required by Dee Zaster are described in section B(ii) of this document. The customer, portfolio and share records are stored in secondary storage in text files. These text files are described in section B(iii) of this document. The Java classes that **must** be implemented in your Java program are described in section B(iv) of this document, however, other classes may also be needed to solve the program requirements.

B(ii) - Program Requirements/Functionality

The Java program must be menu driven. When the program loads it must first display the following main menu which is used to control program execution:

1. Load Files
2. Update
3. Trade Shares

4. Portfolio Report

5. Save Files

6. Exit Program

Each menu item performs a specific set of tasks and/or displays a sub-menu of options that can be chosen by the user.

Main Menu Item Functionality

The required functionality for each menu item is described as follows:

1. Load Files – when this menu option is selected the following actions should be performed by the program:

- read the data from the `shareholder.txt` file into either an array or arraylist of Shareholder objects.
- read the data from the `portfolios.txt` file into either an array or arraylist of Portfolio objects.
- read the data from the `shares.txt` file into either an array or arraylist of Share objects.

If any of the above files do not exist when attempting to read them the user should be informed of this and given the opportunity to provide an alternate filename that contains the relevant data. This alternate file should then be read into the appropriate array or arraylist.

After successfully reading the three files into memory program control should return to the main menu.

See section B(iii) for a description of each text file and section B(iv) for a description of each class.

2. Update – when this menu option is selected a sub-menu is to be displayed that controls which data values can be changed by the user. The sub-menu items are:

1. Update Share Price
2. Update Customer Phone
3. Return to main menu

The sub-menu items perform the following tasks:

1. Update Share Price – Find the share stock for which the price needs to be changed by searching for the share code within the array or arraylist of Share objects. Once found, enter the new share price and update the share price in the array or arraylist. Share prices must be positive numeric values.

2. Update Customer Phone – Find the customer whose phone number needs changing by searching for the surname and first name within the array or arraylist of shareholder objects. Once found, enter the new phone number and update the phone number in the array or arraylist. Phone numbers must be a 10-digit number starting with 04. Phone numbers are not mandatory.

3. Return to main menu - Return program control to the main menu (do not exit the program)

3. Trade Shares - when this menu option is selected the following menu should be displayed:

1. Trades by Keyboard
2. Trades by File

When **Trades by Keyboard** is chosen, the following actions should be performed by the program:

- Find the customer for whom shares are to be traded by searching for the surname and first name within the array or arraylist of shareholder objects.
- Identify which share stock they wish to trade by searching for the share code within the array or arraylist of Share objects. If the customer already owns shares in the chosen corporation, then shares may be either bought or sold. However, if they wish to trade in shares they don't own then shares may only be bought.
- determine if the shares are to be bought or sold for the customer, and how many shares
- transact the trade (ensure your program takes into account all logical validation conditions when doing so).
- generate an on-screen summary of the trade. The summary of the trade must show the customer name and address, portfolio ID, current date, share code, company name, number of shares bought or sold, share price, total of the trade. Make sure that this is displayed in a clear and logical format on the screen.

When **Trades by File** is chosen, the program will need to read the data from the `Trades.txt` file and transact the trades that are identified for each of the portfolios therein. After all of the trades from the file have been transacted the program must generate a report to screen and external file (`TradeReport.txt`) which shows the details of each successful trade for each customer. The report must show trades grouped by customer as in the following example report:

CUSTOMER TRADE REPORT

Firstname Surname

Address
Portfolio ID, Report Date

Share Code	Shares Held	Shares Sold	Shares Bought	Shares Held
BHP	11000	-	1000	12000
SUN	0	-	1000	1000
SUN	1000	-	1000	2000
SUN	2000	1900	-	100

In the situation where a trade would fail, do not generate the above report but rather write an error statement to the **FailedTrades.txt** file. The error statement must contain Portfolio ID, Date, Share Code, Number of Shares that failed, reason for failure, on one line. For example, a line in the FailedTrades.txt file may look like

pf3700,17/09/2017,BEN,-3000,Insufficient shares to sell

4. Portfolio Report - when this menu option is selected the following actions should be performed:

- Determine if the report is for all customers or for a specific customer
- generate the Portfolio Report for either all customers or the specified customer. The Portfolio Report should have a similar layout to the following example:

PORTFOLIO REPORT for Firstname Surname
Address
Portfolio #, Report Date

Share Code	Company Name	Number of Shares	Share Price (\$)	Shares Total (\$)
BHP	BHP Billiton	12000	10.50	126000.00
SUN	Suncorp Group	100	12.00	1200.00
XXX	XXXXXX	XXXXX	XXX.XX	XXXXXXXX.XX
TOTAL		YYYYY		YYYYYYY.YY

5. Save Files – when this menu option is selected the program must write the portfolios data to the Portfolios text file, the customer data to the shareholders text file, and the share data to the shares text file. Ensure that when writing to each file you use the same output format as indicated in section B(iii) so that the files can be used as input files by the program on next execution.

6. Exit Program – the program must terminate when this menu item is selected. The program should not terminate until this option is chosen. If the portfolio, share, or customer data has changed since the last save operation then do not exit the program. Instead, warn the user that changes have been made to the file(s) and that they should choose the Save Files option, then return program control to the main menu.

B(iii) - Text files to be processed

The data that is to be manipulated by your Java program for this project is contained in the text files **shareholders.txt**, **portfolios.txt**, and **shares.txt**. Examples of these text files are found in the zip file for the project. As explained in Load Files of section B(ii) the data within these text files will need to be read into memory by your program so that it may be manipulated to solve many aspects of the required functionality of the project. The text files have been created to conform to a particular format. The format for each file is described below:

File: shareholders.txt

This file contains a full record of all Dee Zaster customers (shareholders). Each line within the file represents an individual customer, and has the following format:

Customer ID, Customer Name, Customer Address, Customer Phone, Portfolio ID

where each data item is separated by a comma (,). A brief explanation of each of these data items:

Customer ID: a unique numeric identifier for a customer
Customer Name: the customer name in the format: firstname surname
Customer Address: the customer address
Customer Phone: the customer mobile number
Portfolio ID: a unique identifier for the customer share portfolio

Two (2) shareholders.txt file has been provided in the project zip file.

File: portfolios.txt

This file contains a full record of all portfolios for each customer. Each line within this file represents an individual portfolio of shares for a shareholder, and has the following format:

Portfolio ID,[Sharecode,Number of shares]{n}

where each data item is separated by a comma (,). Note: A portfolio can have many different share stocks hence the Share Code and Number of shares may be repeated up to 'n' times. A brief explanation of each of these data items:

Portfolio ID: a unique identifier for the customer share portfolio
Sharecode: code used to identify the share stock
Number of Shares: the number of shares for the share held

Two (2) portfolios.txt file has been provided in the project zip file.

File: shares.txt

This file contains a full record of all shares that can be traded by the stockbroker. Each line within the file represents a share stock, and has the following format:

Sharecode,Company Name,price

where each data item is separated by a comma (,). A brief explanation of each of these data items:

Sharecode: code used to identify the share stock
Company Name: the name of the company of the share code
price: the closing price of the share stock

One (1) shares.txt file has been provided in the project zip file.

File: trades.txt

This file contains a full record of all trades that are to be transacted on portfolios for customers. Each line within this file represents an individual trade of shares for a shareholder, and has the following format:

Portfolio ID,Sharecode,Number of shares

where each data item is separated by a comma (,). A brief explanation of each of these data items:

Portfolio ID: a unique identifier for the customer share portfolio
Sharecode: code used to identify the share stock
Number of Shares: the number of shares for the share to be traded. Note: the value will be positive when the shares are to be bought, and will be negative when the shares are to be sold.

One (1) trades.txt file has been provided in the project zip file.

Note: for the purpose of marking the project the number of lines of data and the data values in the text files will be replaced with different data by the marker. This is to ensure that your solution has not relied upon specific data values or the number of lines in the text files to work. You should therefore test your program with different data files before submission.

B(iv) - Required Classes

To write your solution for this project it is a **requirement** that you write appropriate code for **at least** the following java Classes:

- Shareholder
- Portfolio
- Share

These classes are described in general terms as follows:

- **Shareholder class:** The Shareholder class represents *an individual* shareholder (customer). The Shareholder class needs data fields for the Customer ID, first name, surname, address, customer phone, portfolio id. Implement appropriate constructors, accessors, and mutators **where necessary** and other **appropriate** methods for this class based upon the general requirements of the project specification – that is, you will need to identify if the shareholder class is required to perform any other actions and implement the identified methods in the class.

- **Portfolio class:** The Portfolio class represents a collection of *one or more* shares that *an individual* shareholder owns. The Portfolio class needs data fields for the Portfolio ID, Share Code, the number of shares. Implement appropriate constructors, accessors, and mutators where necessary and other **appropriate** methods for this class based upon the general requirements of the project specification – that is, you will need to identify if the portfolio class is required to perform any other actions and implement the identified methods in the class. **Note:** A portfolio can have **up to ‘n’** different share stocks, hence the Share Code and number of shares may need to be stored up to ‘n’ times.
- **Share class:** The Share class represents *an individual* share stock that is traded on the stock market by the broker. The Share class needs data fields for the Share Code, the Company name, the Share price. Implement appropriate constructors, accessors, and mutators where necessary and other **appropriate** methods for this class based upon the general requirements of the project specification – that is, you will need to identify if the share class is required to perform any other actions and implement the identified methods in the class.

These classes **must** be incorporated into your solution. It is likely that you **may also need** to write other classes depending upon your solution method.

COMP2016 Programming Techniques (Advanced) additional classes:

- Determine if *Trades* is a separate class or if they are a specialised extension of one of the classes identified above. Implement the most appropriate solution for Trades. Also, there may be opportunities for class aggregation; if so, implement the identified aggregation(s).
- Implement appropriate utility classes to reduce the amount of duplicate code and to increase the level of code reusability.

C. Google Java Style Guide

The submission in this project must adhere to the following listed coding standards as defined in the Google Java Style Guide that is found at <https://google.github.io/styleguide/javaguide.html>

Style Guide Item Number	Changes to	Modification
2.1 to 2.3	2.1 modified	2.1 - File Name: The source file name consists of the case-sensitive name of the top-level class it contains as identified in the question, plus an underscore, plus the student ID, plus the .java extension Example: Share_12345678.java
3.1 to 3.4.1	3.1 modified	3.1 – License Info is replaced by Authorship information All java source files must contain the authorship information as follows: Student ID: Name: Campus: Tutor Name: Class Day: Class Time:
4.1 to 4.7, 4.8.2.1 to 4.8.2.3, 4.8.4 to 4.8.4.3, 4.8.6	NIL	
5, 5.1 to 5.3	NIL	

D. Referencing

Referencing must follow the guidelines given in Section 2.5.1 of the unit Learning Guide. An example implementation of this referencing style can be found in the FAQ in the Programming Techniques vUWS site.

E. Project Submission Procedure

To submit your **project** you must do the following by the due date and time specified on page 1 of this document:

1. Create a zip file which contains
 - a. your **complete Java project** including the Java source code file(s).

Note: The zip file must be named according to the naming convention

studentid_StudentName_COMP2015_Project.zip

where *studentid* is your student id, and *StudentName* is your full name,

2. Upload the above zip file in vUWS in the **Project Submission** link provided.

F. Marking Criteria and Standards

The marking criteria and standards for the project are published in **section 2.5.2 in the Learning Guide** and will be used to assess your project submission according to the specific weightings identified in the table below

Code Functionality/Correctness:	55%	Code Documentation:	5%
Class Construction	20%	Identifier Use:	5%
Algorithm Selection:	10%	Code Readability:	5%