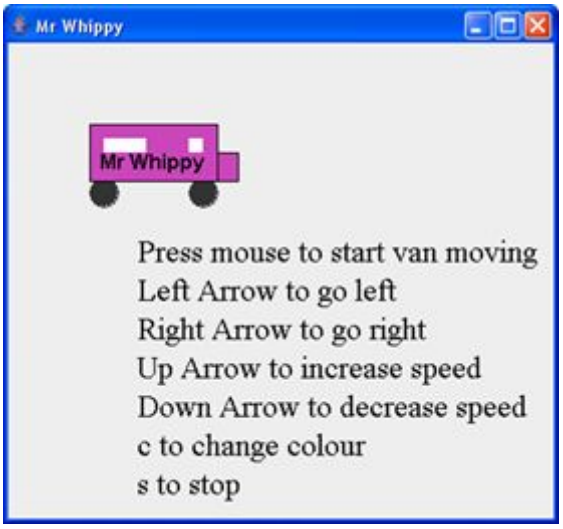
PGCert IT: Programming for Industry

Homework Task 04

Question One - A Simple Swing App

We would like to build a simple event driven GUI program that animates the movement of a van. Currently, the program displays a simple drawing of a van and some instructions at the foot of the window as shown in the following screenshot.



Complete the instructions listed in steps a–f below, such that the user can instruct the van to:

- Move automatically. This is to work by starting a timer (in response to a mouse click) that periodically causes the van to move. To stop periodic movement the users press the ‘s’ key.
- Move left or right. Regardless of whether the van is moving automatically or not, the user can instruct the van to move left or right (using the left and right arrow keys respectively).
- Change the vehicle’s speed. This determines how many pixels the van moves in response to a periodic movement or an explicit move request. The user uses the up and down arrow keys to change the van’s speed.
- Change the vehicle’s colour, by pressing the ‘c’ key.

You should modify only the Q1Panel and the Van class.

a. Add a Timer and MouseListener to the JPanel, and modify the actionPerformed() mousePressed() method so that the van starts to move as soon as the mouse is pressed. You should set the timer to tick every 30 milliseconds.

   Hint: Note that Q1Panel includes MouseListener and ActionListener methods, but you need to add necessary implements clauses.
   Hint: Move the van by calling the move() method on the instance of Van.

b. Add a KeyListener to the JPanel and modify the keyPressed() method in the JPanel class and the move() method in the Van class so that when the user presses the left arrow key or right arrow key, the van's direction changes to the direction of the arrow key that was pressed.

   Note: The constants in the Van class (Van.LEFT and Van.RIGHT) should be used in the keyPressed() method when passing the direction to the setDirection() method in the Van class.

c. Change the colour of the van to a random colour when the user presses the 'c' key.

   Hint: Change the colour of the van by calling the changeColour() method on the instance of Van. The method has already been written for you.

d. Stop the van when the user presses the 's' key.

e. Increase the speed of the van by 2 when the UP arrow key is pressed.

f. Decrease the speed of the van by 2 when the DOWN arrow key is pressed. If the speed goes below 2, set it back to 2.

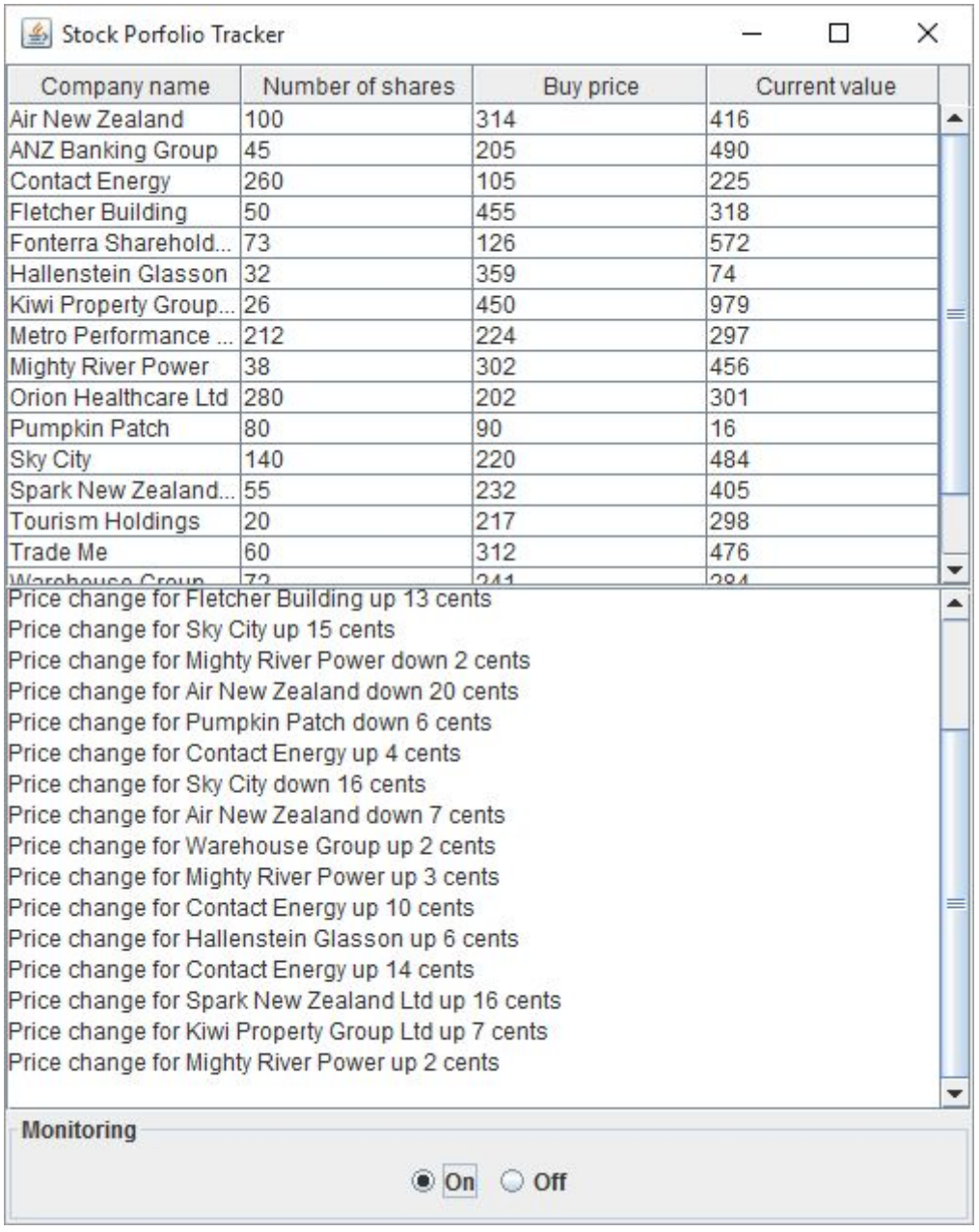
Marking Guide

Question one is marked according to the following criteria:

Timer added to JPanel, listener hooked up	*(6 marks)*
MouseListener added which starts the timer appropriately	*(6 marks)*
KeyListener added which appropriately changes the van's direction	*(7 marks)*
Van's color changes correctly on key press	*(6 marks)*
Van stops correctly on key press	*(5 marks)*
Van's speed increases correctly	*(5 marks)*
Van's speed decreases correctly	*(5 marks)*

Question Two - A Stock Tracking App

In this exercise, you are to complete an application that has been designed according to the Observer design pattern.

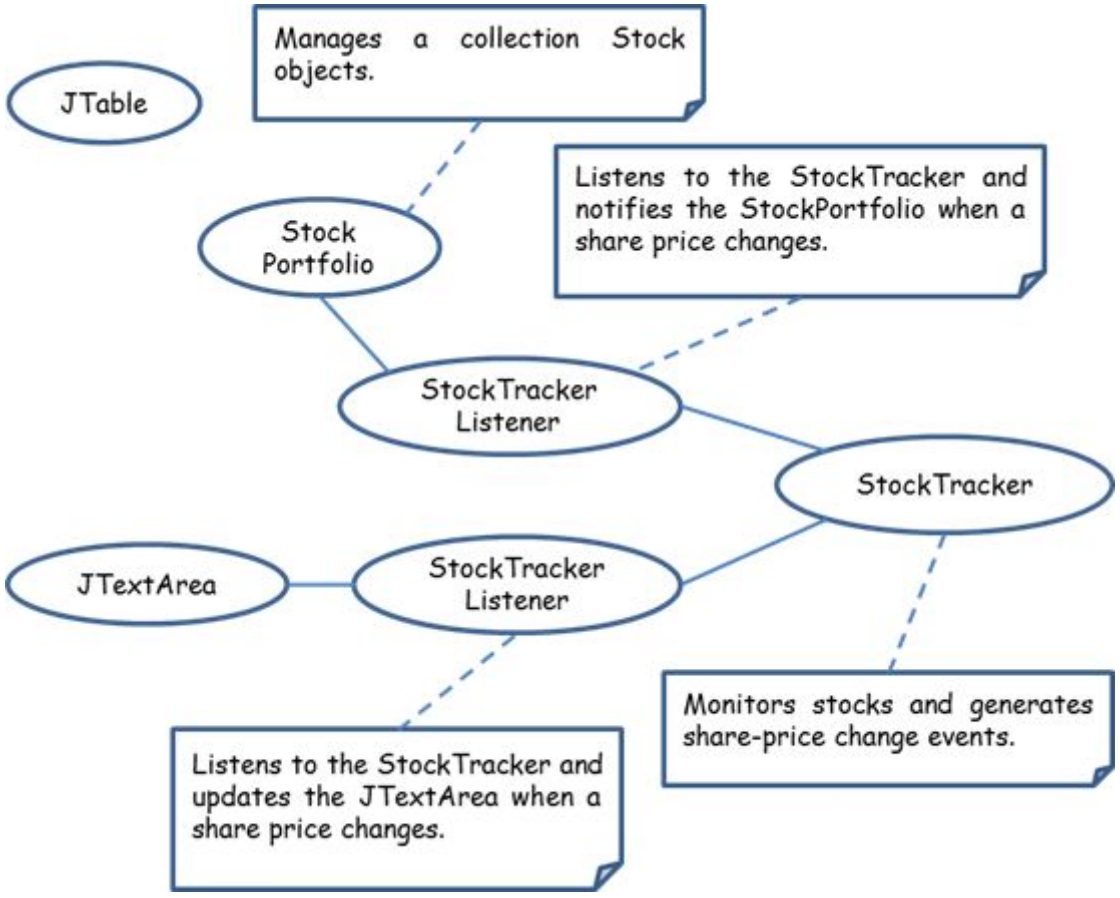


The application is a share-price tracker, and manages a portfolio that comprises shares in company stocks. When in monitoring mode, the application tracks changes in share prices. When a share price changes, the application is supposed to operate as follows:

- The JTable component should update itself to show the new share price in column Current value above.
- The JTextArea component should display how the share price has changed – either rising or falling by a given amount.

In the source code provided, the JTextArea component does get updated – but the JTable fails to be populated or updated.

The following diagram shows informally the key objects and how they currently interact.



Responsibilities of the key entities in the application are as follows:

- **StockPortfolio.** A StockPortfolio object represents a user's portfolio – shares in one or more stocks. A StockPortfolio instance is essentially a collection of Stock objects, with each Stock being described by a company name, the number of shares in the stock (company), the share price when the shares were bought, and the current share price. A StockPortfolio allows its Stocks to be queried and their current share prices to be updated. Finally, a StockPortfolio allows StockPortfolioListeners to be registered and notified when a StockPortfolio changes (e.g the share price of a particular Stock within the portfolio changes).

- **StockPortfolioListener.** This is an interface which should be implemented by classes that are interested in StockPortfolio changes.
- **StockTracker.** A StockTracker object simulates a network feed for share price changes. When set to monitoring mode, the StockTracker is able to generate share price changes. To be made aware of changes in share prices, StockTrackerListeners can be registered with the StockTracker.
- **StockTrackerListener.** This interface is intended to be implemented by classes that should be notified of changes to share prices.
- **StockPortfolioTrackingApp.** This is the main application class that presents the GUI and which creates necessary application objects.

To complete this exercise, you need to complete the application so that the JTable:

1. Displays a representation of the StockPortfolio.
2. Is updated when the StockPortfolio changes.

Constraints:

- You are not permitted to edit any supplied source files except StockPortfolioTrackingApp.
- You may write new classes.

Hints:

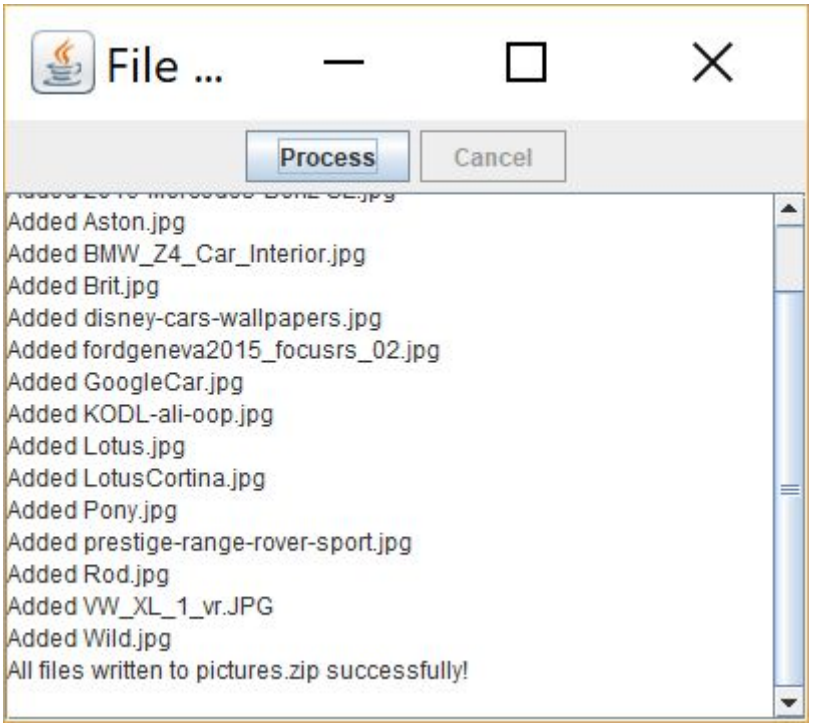
- The only method that you need to change in StockPortfolioTrackingApp is the constructor. Within the source code for this method, there is a comment that suggests where your edits should be contained.
- You need to write one Adapter class.
- Note that in addition to creating a JTable object with a TableModel using JTable's constructor, JTable also provides a method named setModel() that allows you to set the model on a previously created JTable.

Marking Guide

Question two is marked according to the following criteria:

Adapter correctly allows JTable to display stock info	*(18 marks)*
JTable has correct column names	*(6 marks)*
Adapter correctly listens for changes in the stock portfolio	*(8 marks)*
Adapter correctly notifies its own listeners when the portfolio changes	*(8 marks)*

Question Three - A File Zipper App



Class `FileZipperApp` is a complete Swing application that generates a Zip archive containing all the files in a selected directory. In response to pressing the Process button, the application allows the user to select a directory. After selecting the directory, the application generates a Zip file with the same name as the directory, but with the ".zip" file extension. It then loops through all files in the selected directory and adds them to the Zip archive.

The program correctly performs the zip generation – you can go ahead and run it from within IntelliJ to see what it does. However, for large files or large numbers of files in particular, the processing can be expensive and `FileZipperApp` handles all processing on the Event Dispatch thread. Your task is to modify the program so that it more appropriately uses a `SwingWorker` to handle zip processing.

Specifically, modify `FileZipperApp` such that it meets the following requirements:

1. All file loading and creation of the Zip archive must be performed in the background.
2. Whenever an individual file has been added to the Zip archive, the GUI should report (as an intermediate String result) that that particular file has been added to the archive.
3. By clicking on the Cancel button, the user must be able to abort the processing.

Hints

Introduce a new class that is a subclass of `SwingWorker`, and nest this class inside `FileZipperApp`. Define an instance variable in class `FileZipperApp` to hold a reference to an instance of your `SwingWorker` subclass. This way, you'll be able to access the `SwingWorker` object from any method in `FileZipperApp`.

Other than defining the `SwingWorker` class and associated instance variable, you only need to modify the constructor for `FileZipperApp`, and do so in two places:

1. Implement the `ActionListener` for the Cancel button in `FileZipperApp`'s constructor.
2. Move the code on lines 81 to 116 (marked with a TODO comment) into your `SwingWorker` implementation appropriately.

You should not have to do much coding, several lines at most – make sure you understand the exercise before editing code.

To test your program, you may use the supplied pictures folder containing car images.

Marking Guide

Question two is marked according to the following criteria:

Nested SwingWorker class definition	*(4 marks)*
Appropriately starts SW running	*(4 marks)*
Necessary code moved into various SW methods	*(6 marks)*
No incorrect code moved into SW methods	*(4 marks)*
Button event handler	*(4 marks)*
Appropriate background processing	*(4 marks)*
Appropriate intermediate results processing	*(6 marks)*
Appropriate cancellation processing	*(5 marks)*
GUI "niceties" (enabling / disabling buttons, displaying alternative mouse cursor, etc)	*(3 marks)*