

CIS*2430 (Fall 2024) Assignment Three

Instructor: F. Song

Due Time: November 29, 2024 by 11:59 pm

In Assignments One and Two, you created at least four classes (Investment, Stock, MutualFund, and Portfolio) and were able to buy/sell investments, update prices, get the total gain, and search for relevant investments. You were also able to read/write investments from/to files and maintain a HashMap index to speed up keyword search. In Assignment Three, you will add a GUI interface and exception handling so that the system will be more robust and user-friendly.

Please note that you are NOT ALLOWED to use any GUI builder that comes with an IDE (e.g., NetBean and Visual Studio) or any other GUI packages (other than awt and swing) for this assignment since we want you to learn the basic ways of building GUI interfaces rather than taking shortcuts.

Specific Requirements for Assignment Three

- (1) Build a GUI for our “ePortfolio” project. In particular, you need to create the interfaces for the following cases:
 - (a) Create an initial interface that looks like the first figure at the end of this description. The content pane shows a welcome message and a brief instruction of how to use the system, and the “commands” menu has six choices: buy, sell, update, getGain, search, and quit. Choosing “quit” will simply terminate the program, but choosing other commands will change the interface to the ones specified in the following requirements. Note that choosing the X button on the window frame will also terminate your program.
 - (b) Create an interface for buying an investment. When the user selects the “buy” command, the interface will be changed to the figure for “Buying an investment”. Investment “type” can be selected through a combo-box with two choices: stock and mutualfund. The default choice is “stock”, but the user can select “mutualfund” in the drop-down list. The related attributes: symbol, name, quantity, and price all get their values through textfields. There are two buttons on the righthand side panel: clicking the “reset” button will clear all the textfields and clicking the “buy” button will try to create a new investment or add more quantity to an existing investment. However, if there are any violations for the validity of the investment, an error message will be displayed in the text area at the bottom of the interface and the user will be required to try again.
 - (c) Create an interface for selling investments. When the user selects the “sell” command, the current interface will be changed to the figure for “Selling an investment”. The interface looks similar to that for buying an investment but with fewer textfields for the related values. Note that in all the interfaces, you will need to use the scrollable text area so that the user can examine the full feedback if needed.
 - (d) Create an interface for updating investments. When the user selects the “update” command, the interface will be changed to the figure for “Updating investments”. The

interface should show the “Symbol” and “Name” as non-editable textfields so that we know which investment we are updating, but “Price” should be an editable textfield so that we can change its value if needed. The “Prev” and “Next” buttons allow us to move backward or forward in the list of investments so that we can make as many updates as possible. When we move to the first investment on the list, there will be no previous investment; so the “Prev” button should be disabled in such a situation. Similarly, when we reach the last investment on the list, the “Next” button should also be disabled. Once we click the “Save” button, the price for the current investment will be updated and its full content will be shown in the text area at the bottom. Any invalid value will be rejected and an error message will be shown in the text area as well.

- (e) Create an interface for getting the total gain. When the user selects the “getGain” command, the interface will be changed to the figure for “Getting total gain”. The “Total Gain” should be a non-editable textfield since its value is calculated. To examine the details, we will show the individual gains for each investment in the text area at the bottom. There should be a lot of content if there are many investments, but the scroll bars should allow us to see it fully if needed.
 - (f) Create an interface for searching investments. When the user selects the “search” command, the interface will be changed to the figure for “Searching investments”. The interface is almost the same as that for buying an investment except that some or all of the textfields can have empty values. In addition, the label for the text area is changed from “Messages” to “Search Results”. Again, scrollable text area is essential for the user to examine the full content if needed.
- (2) Add exception handling to your program so that any violations to invariant conditions of the related classes can be handled more elegantly. In particular, you need to throw exceptions in the corresponding constructors and mutator methods whenever the class invariants are violated. You should also use exception handling to check for incorrect input so that the user will be allowed to try again for correct values. For example, if your program expects an integer, but the user enters a string value, you should catch the exception and let the user try again.
- (3) Perfect the details: A well-defined class should pay attention to details, including the following specific requirements. Please note that some of them may already be done in your earlier assignments, but here we are checking them collectively for completeness.
- (a) Validity check for creating new objects and getting input values: e.g., for all investments, the values for symbol and name cannot be empty, and the values for quantity and price must be positive. If prices are given for a search request, their values should be non-negative and the low-end price should be equal or smaller than the high-end price.
 - (b) Privacy leak protection for mutable classes: For a mutable class, you should define a copy constructor so that for parameter passing of its references, we can create a duplicate object and assign it to another reference in order to avoid the creation of co-references that

may cause privacy leaks. In addition, for any reference parameters, we should also create duplicates if they belong to mutable classes.

(c) Truly overriding methods for inheritance: An overriding method should be truly overriding instead of overloading. For example, the header for the “equals” method should be “public boolean equals(Object other)”, not something like “public boolean equals(Investment other)” for the Investment class. The latter is an example of overloading since it has a different signature as the “equals” method defined in the Object class. An overriding method must have the same signature, and as a result, can only be defined in a descendant class.


(d) Proper use of abstract class: The Investment class should be declared as an abstract class so that real investments can only be created from the concrete classes, including the Stock and MutualFund classes.

(e) Proper use of abstract methods: both Stock and MutualFund classes have their own methods for computing bookValue, gain, and payment. In order to write general code in the Investment class, you should define the related abstract methods in the Investment class so that we can take advantage of polymorphism when processing real stocks and mutualfunds within the Investment class.

(f) Effective use of polymorphism: polymorphism is done through dynamic binding and if used effectively, we can significantly simplify and create general code for implementation. For example, given “investments” as an ArrayList<Investment>, which may contain both stocks and mutualfunds, we do not need to check its class type for “investments.get(i)” in order to display all the details for stocks and mutualfunds. Instead, we can simply put “System.out.println(investments.get(i).toString())” or even “System.out.println(investments.get(i))” in a for-loop to show the detailed information of all investments in the given list.

Deliverables: Similar to what is required for Assignments One and Two, but for submission, you need to create a new folder named <userid_a3> and include all the related files into this folder. For example, I would name the folder “fsong_a3” if I were to make a submission for Assignment Three. After that, you can compress the folder along with its files into a single file using the allowable compressing utilities before sending your submission to the related dropbox on CourseLink.

ePortfolio	-	[]	x
Commands			
<p>Welcome to ePortfolio.</p> <p>Choose a command from the "Commands" menu to buy or sell an investment, update prices for all investments, get gain for the portfolio, search for relevant investments, or quit the program.</p>			

ePortfolio	-	[]	x
Commands			
Buying an investment			
Type	<input type="text" value="stock"/>		<div>Reset</div> <div>Buy</div>
Symbol	<input type="text"/>		
Name	<input type="text"/>		
Quantity	<input type="text"/>		
Price	<input type="text"/>		
Messages			
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>			

ePortfolio		-	[]	x
Commands				
Selling an investment				
Symbol	<input type="text"/>	<input type="button" value="Reset"/>		
Quantity	<input type="text"/>			
Price	<input type="text"/>	<input type="button" value="Sell"/>		
<hr/>				
Messages				
<div></div>				<div><div>▲</div><div>▼</div></div>
<div>◀</div>	<div></div>			<div>▶</div>

ePortfolio		-	[]	x
Commands				
Updating investments				
Symbol	<input type="text"/>	<input type="button" value="Prev"/>		
Name	<input type="text"/>	<input type="button" value="Next"/>		
Price	<input type="text"/>	<input type="button" value="Save"/>		
<hr/>				
Messages				
<div></div>				<div><div>▲</div><div>▼</div></div>
<div>◀</div>	<div></div>			<div>▶</div>

ePortfolio	-	[]	x
Commands			
Getting total gain			
Total gain	<input type="text"/>		

Individual gains			
<div style="border: 1px solid black; height: 150px; width: 100%;"></div>			<div style="border: 1px solid black; width: 20px; height: 150px; position: relative;"> <div style="position: absolute; top: 0; right: 0; width: 10px; height: 10px; background: white;"></div> <div style="position: absolute; bottom: 0; right: 0; width: 10px; height: 10px; background: white;"></div> </div>
<div style="border: 1px solid black; height: 20px; width: 100%;"></div>			<div style="border: 1px solid black; width: 20px; height: 20px; position: relative;"> <div style="position: absolute; top: 0; right: 0; width: 10px; height: 10px; background: white;"></div> <div style="position: absolute; bottom: 0; right: 0; width: 10px; height: 10px; background: white;"></div> </div>

ePortfolio	-	[]	x
Commands			
Searching investments			
Symbol	<input type="text"/>	<div style="border: 1px solid black; padding: 5px; text-align: center;">Reset</div> <div style="border: 1px solid black; padding: 5px; text-align: center; margin-top: 20px;">Search</div>	
Name	<input type="text"/>		
keywords	<input type="text"/>		
Low price	<input type="text"/>		
High price	<input type="text"/>		

Search results			
<div style="border: 1px solid black; height: 100px; width: 100%;"></div>			<div style="border: 1px solid black; width: 20px; height: 100px; position: relative;"> <div style="position: absolute; top: 0; right: 0; width: 10px; height: 10px; background: white;"></div> <div style="position: absolute; bottom: 0; right: 0; width: 10px; height: 10px; background: white;"></div> </div>
<div style="border: 1px solid black; height: 20px; width: 100%;"></div>			<div style="border: 1px solid black; width: 20px; height: 20px; position: relative;"> <div style="position: absolute; top: 0; right: 0; width: 10px; height: 10px; background: white;"></div> <div style="position: absolute; bottom: 0; right: 0; width: 10px; height: 10px; background: white;"></div> </div>