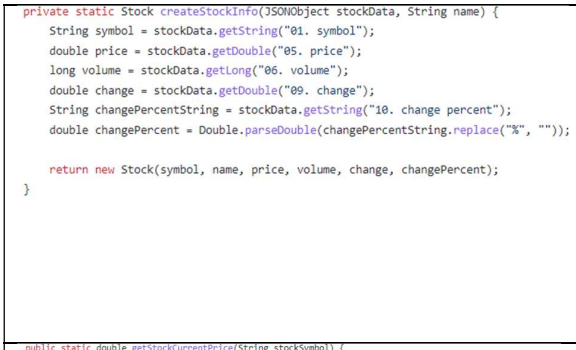
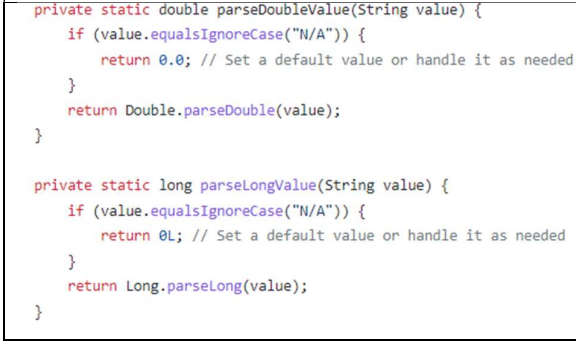
Stock.java

public class Stock implements Comparable<Stock>{	// Defines the class named "Stock" and implements the Comparable interface with the generic type parameter of Stock
Constructor	Diff parameter, initialize variable of Stock class
Getter	Retrieve
Setter	Set
updateStock	// Method to update the stock by subtracting the quantity of the order from the volume of the stock
compareTo	// Method to compare the volume of this stock with another stock for sorting purposes
toString	// Method to provide a string representation of the Stock object

StockUtils.java

`` public class StockUtils { private static final String API_KEY1 = "ZK0GFWAKA82MSEC9"; private static final String API_KEY2 = "FEVX5MOLAQB0H6NH"; ```	These lines define the `StockUtils` class and declare two `API_KEY` constants for accessing stock data. The second API key is specifically for educational purposes with a higher request limit.
`` public static Stock searchStock(String input){ input = input.trim(); try { String[] symbol = getStockSymbol(input.toUpperCase(), API_KEY2); if (symbol == null) { System.out.println("Stock not found."); return null; } JSONObject stockData = getStockData(symbol[0], API_KEY2); if (stockData == null) { System.out.println("Error retrieving stock data. Please try again later. (return null; } return createStockInfo(stockData, symbol[1]); } catch (IOException e) { System.out.println("Error: " + e.getMessage()); } return null; } ```	This method (`searchStock`) takes an `input` parameter, which represents the stock to search for. It trims the input, gets the stock symbol using `getStockSymbol` method, retrieves stock data using `getStockData` method, and creates a `Stock` object using `createStockInfo` method. It returns the created `Stock` object if successful; otherwise, it prints error messages and returns null.
`` public static String[] getStockSymbol(String input, String apikey) throws IOException { String apiUrl = "https://www.alphavantage.co/query?function=SYMBOL_SEARCH&keywords=" + input + "&apikey=" + apikey; URL url = new URL(apiUrl); HttpURLConnection connection = (HttpURLConnection) url.openConnection(); connection.setRequestMethod("GET"); BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream())); StringBuilder response = new StringBuilder(); String line; while ((line = reader.readLine()) != null) { response.append(line); } reader.close(); JSONObject jsonResponse = new JSONObject(response.toString()); if (jsonResponse.has("bestMatches")) { JSONObject bestMatch = jsonResponse.getJSONArray("bestMatches").optJSONObject(0); if (bestMatch != null) { String symbol = bestMatch.getString("1. symbol"); String name = bestMatch.getString("2. name"); return new String[]{symbol, name}; } } return null; } ```	This method (`getStockSymbol`) takes an `input` parameter (stock symbol or name) and an `apikey` parameter. It constructs the API URL, makes an HTTP GET request to the Alpha Vantage API, retrieves the response, and parses the JSON response to extract the stock symbol and name. It returns an array of strings containing the symbol and name if a match is found; otherwise, it returns null.
`` public static JSONObject getStockData(String symbol, String apiKey) throws IOException { String apiUrl = "https://www.alphavantage.co/query?function=GLOBAL_QUOTE&symbol=" + symbol + "&apikey=" + apiKey; URL url = new URL(apiUrl); HttpURLConnection connection = (HttpURLConnection) url.openConnection(); connection.setRequestMethod("GET"); BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream())); StringBuilder response = new StringBuilder(); String line; while ((line = reader.readLine()) != null) { response.append(line); } reader.close(); JSONObject jsonResponse = new JSONObject(response.toString()); if (jsonResponse.has("Global Quote")) { return jsonResponse.getJSONObject("Global Quote"); } return null; } ```	This method (`getStockData`) takes a `symbol` (stock symbol) and `apiKey` as parameters. It constructs the API URL for retrieving global quote data, makes an HTTP GET request to the Alpha Vantage API, retrieves the response, and parses the JSON response to extract the global quote information. It returns a `JSONObject` containing the global quote data if available; otherwise, it returns null.

	This method (`createStockInfo`) takes a `stockData` `JSONObject` and a `name` parameter. It extracts specific data fields (symbol, price, volume, change, change percentage) from the `stockData` JSON object and creates a new `Stock` object with the extracted information. It returns the created `Stock` object.
	This method (`getStockCurrentPrice`) takes a `stockSymbol` parameter and retrieves the current price of the specified stock. It constructs the API URL for retrieving global quote data, makes an HTTP GET request to the Alpha Vantage API, retrieves the response, and parses the JSON response to extract the current stock price. It returns the current stock price as a double value.
	This method (`scrapeStock`) scrapes the stock data from the Yahoo Finance most active stocks page. It sends a GET request to the specified URL, retrieves the HTML document using Jsoup, and then extracts the stock data from the HTML table. It iterates over the table rows, extracts the symbol, name, price, change, change percentage, and volume values from the table cells, creates a `Stock` object with the extracted data, and adds it to the `stockData` list. Finally, it returns the list of `Stock` objects.
	These are helper methods (`parseDoubleValue` and `parseLongValue`) used to parse string values to their respective numerical types. They handle cases where the value is "N/A" by returning a default value (0.0 for `double` and 0L for `long`) or you can handle it as needed in your application.

HomeController.java

public class HomeController implements Initializable {	The `HomeController` class is a controller for the home view in a JavaFX application. It implements the `Initializable` interface, which allows initializing the controller and its components when the associated FXML file is loaded.
`private User user;` `public void setUser(User user){` `this.user = user;` `}`	This code defines a `User` object and a setter method `setUser()` to set the currently logged-in user.
@FXML	These lines declare several labels that are defined in the associated FXML file. The `@FXML` annotation is used to inject the labels defined in the FXML into the controller.
private Timeline timeline;	This declares a `Timeline` object that will be used to update the time and market status labels periodically.
`private TableColumn<Stock, Long> column_volume;` `List<Stock> popularStocks = StockUtils.scrapeStock();`	These lines declare a `TableView` and several `TableColumn` objects used for displaying stock data. The `@FXML` annotation is used to inject them from the FXML file. The `popularStocks` list is populated with stock data using the `scrapeStock()` method from the `StockUtils` class.
`@Override` `public void initialize(URL url, ResourceBundle resourceBundle) {` `// ...` `}`	This is the initialization method that is called when the associated FXML file is loaded. It initializes the controller and sets up the initial state of the components.
`timeline = new Timeline(new KeyFrame(javafx.util.Duration.seconds(1), event -> {` `updateDateTime();` `updateMarketStatus();` `}));` `timeline.setCycleCount(Timeline.INDEFINITE); // Run indefinitely` `timeline.play();`	This code creates a `Timeline` that updates the time and market status labels every second. It uses a `KeyFrame` with a duration of one second and defines an event handler to call the `updateDateTime()` and `updateMarketStatus()` methods.

TradeController.java

public class TradeController implements Initializable {	This line declares the class `TradeController` and specifies that it implements the `Initializable` interface.
`private User user;` `public void setUser(User user){` `this.user = user;` `this.user.displayUserPortfolio();` `System.out.println("load user successful");` `label_balance.setText("$" + String.format("%.2f",user.getBalance()));` `label_profit.setText("$" + String.format("%.2f",user.getPnL()));` `label_loss.setText("" + String.format("%.2f",user.getPoint()));` `}`	These lines declare a private instance variable `user` of type `User` and define a setter method `setUser()` to set the user object. Inside the `setUser()` method, the `user` instance variable is assigned the provided `User` object, and some user-related information is displayed in labels.
`private TradingFunctions tf;` `public void setTf(TradingFunctions tf){` `this.tf = tf;` `}`	These lines declare a private instance variable `tf` of type `TradingFunctions` and define a setter method `setTf()` to set the `TradingFunctions` object.
private Timeline timeline;	This line declares a private instance variable `timeline` of type `Timeline` for updating the time and market status labels.
btn_setting → label_companyName	These lines annotate and declare various JavaFX UI components, such as buttons and labels, with `@FXML` annotations for use in the controller.
label_stockCode → btn_search	These lines declare additional UI components related to stock information.
`@FXML` `private ChoiceBox<String> cBox_action;` `private String[] action = {"BUY", "SELL"};`	These lines declare a choice box UI component and a string array containing the available actions (buy and sell).
tf_symbol → btn_executeTrade	These lines declare UI components related to trade functionality, such as text fields and buttons.