GTU DEPARTMENT OF COMPUTER ENGINEERING

CSE 222

HOMEWORK #7

EMİR İNCE

200104004025

Code Structure

1. Main Class

This class is the entry point for the program. It handles initial command-line arguments, processes commands from an input file, and orchestrates various operations on stock data.

Functions:

main(String[] args): Processes input file and initializes the stock data simulation including the addition, search, and removal operations on stock data.

processCommand(String line, StockDataManager manager): Processes individual commands like ADD, REMOVE, SEARCH, and UPDATE from the input file.

performFullAnalysis(StockDataManager manager, int size, ArrayList<Long> dataAddY, ArrayList<Long> dataSearchY, ArrayList<Long> dataRemoveY): Performs timed addition, search, and removal operations on stock data to gather performance metrics.

performOperation(StockDataManager manager, int size, String operation): Helper method for performFullAnalysis that times specific stock operations.

warmUpPhase(List<StockDataManager> managers, ArrayList<Integer> sizes): Warms up the program by performing operations on all managers to ensure JIT compilation doesn't impact timing.

2. AVLTree Class

Implements an AVL Tree to manage Stock objects efficiently with balanced tree properties. This class includes methods to insert, delete, and search for stock data.

Functions:

insert(Stock stock), delete(String symbol), search(String symbol): Core AVL operations to modify the tree and search for nodes.

rightRotate(Node y), leftRotate(Node x): Rotations to maintain tree balance.

inOrderTraversal(), preOrderTraversal(), postOrderTraversal(): Various tree traversal methods for debugging or tree data extraction.

3. StockDataManager Class

Acts as a facade to the AVL tree, providing a higher-level interface for stock management operations.

Functions:

addOrUpdateStock(String symbol, double price, long volume, long marketCap): Adds a new stock or updates an existing one.

removeStock(String symbol), searchStock(String symbol): Wrapper methods around AVL tree operations.

updateStock(String oldSymbol, String newSymbol, double newPrice, long newVolume, long newMarketCap): Updates stock information based on given parameters.

4. Stock Class

Represents the stock entity with attributes like symbol, price, volume, and market cap.

Functions:

Contains getter and setter methods for its attributes.

toString(): Overridden to provide a string representation of the stock data.

5. GUIVisualization Class

A graphical user interface class extending JFrame to visualize performance metrics as a scatter or line plot.

Functions:

paint(Graphics g): Core method to draw the graphs based on performance data.

drawGraph(Graphics g): Helper method to handle the actual drawing of data points on the graph.

getMaxYValue(): Calculates the maximum Y-value from the data sets to normalize the graph scaling.

6. RandomInputGenerator Class

Generates random input commands to simulate stock operations, useful for testing and benchmarking.

Functions:

generateRandomInput(String filename, int numNodes, int numAdd, int numRemove, int numSearch, int numUpdate): Generates a specified number of random stock operations and writes them to a file.

Each class is tailored to handle specific parts of the application, from data management with AVL trees to user interface for data visualization, providing a comprehensive system for stock data management and performance analysis.

How To Run

make : Type “make” to compile and run the program.

Problems and Solution

The most significant problem that I encountered is accurate graphs. To make it more accurate, I used “–Xint” flag. But since I used this flag, the program works slow. It takes around 20-30 seconds to open the GUI.

Example Outputs

Red: Remove operation.

Blue: Add operation.

Green: Search Operation