**CM1602**

**DATA STRUCTURES**

**AND**

**ALGORITHMS**

**COURSEWORK**

EXECUTIVE SUMMARY

CONTENTS

QUESTION 1

import java.util.NoSuchElementException;  
import java.util.Scanner;  
  
public class CapitalGainCalculator {  
  
 public static void main(String[] args) {  
 Deque<Transaction> buyQueue = new Deque<Transaction>(); //initialize Dequeue with max size 100  
 Deque<Transaction> sellQueue = new Deque<Transaction>(); //initialize Dequeue with max size 100  
 Scanner = new Scanner(System.*in*);  
 String input;  
 double totalGain = 0.0;  
  
 while (true) {  
 System.*out*.println("Enter B for buying transaction, S for selling transaction or Q to exit:");  
 input = scanner.nextLine().trim().toUpperCase();  
  
 if (input.equals("B")) {  
 System.*out*.println("Enter number of shares bought and unit price, separated by a space:");  
 String[] tokens = scanner.nextLine().trim().split(" ");  
 if (tokens.length != 2) {  
 System.*out*.println("Invalid input, please try again.");  
 continue;  
 }  
 int shares = 0;  
 double price = 0.0;  
 try {  
 shares = Integer.*parseInt*(tokens[0]);  
 price = Double.*parseDouble*(tokens[1]);  
 } catch (NumberFormatException e) {  
 System.*out*.println("Invalid input, please try again.");  
 continue;  
 }  
 if (shares <= 0 || price <= 0.0) {  
 System.*out*.println("Invalid input, please try again.");  
 continue;  
 }  
 buyQueue.addLast(new Transaction(shares, price));  
 System.*out*.println("Transaction recorded.");  
 } else if (input.equals("S")) {  
 System.*out*.println("Enter number of shares sold and unit price, separated by a space:");  
 String[] tokens = scanner.nextLine().trim().split(" ");  
 if (tokens.length != 2) {  
 System.*out*.println("Invalid input, please try again.");  
 continue;  
 }  
 int shares = 0;  
 double price = 0.0;  
 try {  
 shares = Integer.*parseInt*(tokens[0]);  
 price = Double.*parseDouble*(tokens[1]);  
 } catch (NumberFormatException e) {  
 System.*out*.println("Invalid input, please try again.");  
 continue;  
 }  
 if (shares <= 0 || price <= 0.0) {  
 System.*out*.println("Invalid input, please try again.");  
 continue;  
 }  
 int remainingShares = shares;  
 double gain = 0.0;  
 while (remainingShares > 0 && !buyQueue.isEmpty()) {  
 Transaction oldestTransaction = buyQueue.removeFirst();  
 if (remainingShares >= oldestTransaction.shares) {  
 // sell all shares in the oldest transaction  
 remainingShares -= oldestTransaction.shares;  
 gain += (price - oldestTransaction.price) \* oldestTransaction.shares;  
 } else {  
 // sell only some shares in the oldest transaction  
 oldestTransaction.shares -= remainingShares;  
 gain += (price - oldestTransaction.price) \* remainingShares;  
 remainingShares = 0;  
 buyQueue.addFirst(oldestTransaction);  
 }  
 }  
 if (remainingShares > 0) {  
 // not enough shares to fulfill the sell order  
 System.*out*.println("Not enough shares to sell, transaction cancelled.");  
 } else {  
 sellQueue.addLast(new Transaction(shares, price));  
 System.*out*.printf("Transaction recorded, capital gain: $%.2f%n", gain);  
 totalGain += gain;  
 }  
 } else if (input.equals("Q")) {  
 double gain = 0.0;  
 while (!buyQueue.isEmpty()) {  
 Transaction = buyQueue.removeLast();  
 gain += transaction.price \* transaction.shares;  
 }  
 System.*out*.printf("Total capital gain: $%.2f%n", totalGain);  
 System.*out*.printf("Remaining items for : $%.2f%n", gain);  
 break;  
 } else {  
 System.*out*.println("Invalid input, please try again.");  
 }  
 System.*out*.println();  
 }  
 }  
  
 private static class Transaction {  
 private int shares;  
 private double price;  
  
 public Transaction(int shares, double price) {  
 this.shares = shares;  
 this.price = price;  
 }  
 }  
  
 public static class Deque<T> {  
 private Node head; // pointer to the head of the deque  
 private Node tail; // pointer to the tail of the deque  
 private int size; // size of the deque  
  
 // Node class to represent each item in the deque  
 private class Node {  
 private T data;  
 private Node prev;  
 private Node next;  
  
 public Node(T data) {  
 this.data = data;  
 this.prev = null;  
 this.next = null;  
 }  
 }  
  
 // constructor to create an empty deque  
 public Deque() {  
 head = null;  
 tail = null;  
 size = 0;  
 }  
  
 // method to add an item to the front of the deque  
 public void addFirst(T item) {  
 Node newNode = new Node(item);  
 if (head == null) {  
 head = newNode;  
 tail = newNode;  
 } else {  
 newNode.next = head;  
 head.prev = newNode;  
 head = newNode;  
 }  
 size++;  
 }  
  
 // method to add an item to the back of the deque  
 public void addLast(T item) {  
 Node newNode = new Node(item);  
 if (tail == null) {  
 head = newNode;  
 tail = newNode;  
 } else {  
 newNode.prev = tail;  
 tail.next = newNode;  
 tail = newNode;  
 }  
 size++;  
 }  
  
 // method to remove and return the item at the front of the deque  
 public T removeFirst() {  
 if (head == null) {  
 throw new NoSuchElementException();  
 }  
 T data = head.data;  
 head = head.next;  
 if (head == null) {  
 tail = null;  
 } else {  
 head.prev = null;  
 }  
 size--;  
 return data;  
 }  
  
 // method to remove and return the item at the back of the deque  
 public T removeLast() {  
 if (tail == null) {  
 throw new NoSuchElementException();  
 }  
 T data = tail.data;  
 tail = tail.prev;  
 if (tail == null) {  
 head = null;  
 } else {  
 tail.next = null;  
 }  
 size--;  
 return data;  
 }  
  
 // method to check if the deque is empty  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 // method to get the size of the deque  
 public int size() {  
 return size;  
 }  
 }  
}