```
/* Lucas Carpenter
 * C243 - Data Structures
 * March 4, 2025
 * Program 2 - QueueApp
 *
 * Description: This program implements a queue ADT using a linked list of floats.
 * The UserQueue class implements the MyQueue interface, which defines the core queue operations: enqueue, dequeue, viewFront,
 * and isEmpty.
 * The queue is built using a inner class PurchaseNode, which is a node class that is used to create the linked list of floats
 * that each represent stocks (10) that the user has purchased.
 *
 * The program includes a stand-alone driver class QueueApp that uses the UserQueue class to facilitate the user's stock
 * purchases and sales. A class, UserQueue(outer), that simulates the queue ADT with use of a Linked List that is
 * characterized by the PurchaseNode class(inner). The UserQueue front and rear are tracked with PurchaseNodes
 * that point to the respective front and rear of the queue.
 *
 * Overall the program is designed for the purpose of simulating a linked list structure to implement a queue ADT.
 * By means of making a media for users to abstractly buy and sell stocks with a few other features in the menu system in the
 * driver class(QueueApp).
 */
```

```java
import java.util.Scanner;                       // import the Scanner class to get user input.(specifically floats)



/*
 * UserQueue class Description: This object simulates the queue ADT with use of a Linked List.
 * UserQueue implements interface MyQueue, provided by the instructor, which requires the implementation of methods enqueue,
 * dequeue, viewFront, and isEmpty. The User Queue is a object that facilitates the user's stock purchases and sales.
 * The QueueApp class makes use of the UserQueue class by implementing a menu which calls the methods and inner class of the
 * UserQueue class. The inner class PurchaseNode is a node class that is used to create the linked list of floats that each
 * represent stocks(10) that the user has purchased. There are 2 instance variables front and rear that represent the front
 * and rear of the queue respectively. these nodes help to keep track of the oldest stock purchased by the user and the
 * newest stock purchased by the user. which lets the program manipulate the linked list by the medium that
 * is the QueueApp driver.
 */
class UserQueue implements MyQueue {

    //instance variables
    PurchaseNode front;                         // front pointer of the queue.
    PurchaseNode rear;                          // rear pointer of the queue.

    // constructor for UserQueue
    public UserQueue() {
        /*
         * UserQueue constructor Description: This constructor initializes the front and rear pointers to null.
         * The purpose of the constructor is to ensure a new queue is empty when it is created.
         * It is to be used in the QueueApp class to create a new queue object(UserQueue).
         */
        front = null;                           // set the front pointer to null.
        rear = null;                            // set the rear pointer to null.
    }
```

```
public void enqueue(float price) {
      PurchaseNode newNode = new PurchaseNode(price);// create a new node with the value of the stocks.
      if (isEmpty()) {                       // IF the queue is empty THEN set the front and rear pointers to the new node.
          front = newNode;                   // set the front pointer to the new node.
          rear = newNode;                    // set the rear pointer to the new node.
      } else {                               // IF the queue is not empty THEN add the new node to the rear of the queue.
          rear.link = newNode;               // set the link of the rear node to the new node.
          rear = newNode;                    // set the rear pointer to the new node.
      }
  }

  public float dequeue() {
      /*
       * dequeue method Description: removes the front node from the queue and returns the value of the node
       * the front node represents the longest(oldest) standing stock owned by the
       * user.
       * IMPORTANT: the user/program should check if the queue is empty before calling
       * dequeue.
       */

      float price = front.price;              // store the value of the front node in a temp variable
      front = front.link;                     // move the front pointer to the next node.
      if (front == null) {                    // IF the front pointer is null THEN the queue is empty
          rear = null;                        // THEN, set the rear pointer to null.
      }
      return price;                           // in the end return the value of the front node.
  }
```

```
public float viewFront() {
     /*
      * viewFront method Description: returns the value of the front node of the queue
      * the front node represents the longest(oldest) standing stock owned by the user.
      * IMPORTANT: the user/program should check if the queue is empty before calling
      * viewFront to avoid null pointer exceptions
      */
     return front.price;                         // return the value of the front node
 }
```

```
 public boolean isEmpty() {
     /*
      * isEmpty method Description: Checks if the queue is empty by checking if the front and rear pointers are null.
      * returns true if the queue is empty and false if the queue is not empty. This method is especially useful for
      * avoiding errors in the driver class QueueApp when removing head from the LinkedList and peeking the front element.
      */
     return ((front == null) && (rear == null)); // if both the front AND rear pointers are null THEN the queue is
                                                 // certainly empty(return true) else if front is not null and rear
                                                 // is not null the the queue is not empty(return false).
 }
```

```
    /*
     * PurchaseNode class Description: This is a inner class to the UserQueue class. This class is a node class that is
     * used to create the linked list of floats that each represent stocks(10) that the user has purchased. The class has 2
     * instance variables price(float) and link(PurchaseNode). float(price) represents the purchased value of the stock
     * by the user
     */
    class PurchaseNode {

        // instance variables
        private float price;                                    // the value of the stock(price, float) initialized.
        private PurchaseNode link;                              // the next node in the linked list(link, PurchaseNode)
                                                                // initialized.


        // start constructors for Node
        public PurchaseNode(float initPrice) {
            /*
             * PurchaseNode constructor Description: This constructor creates a new node with the value of the stock and
             * sets the link to null. The constructor is used to create a new node with the value of the stock and sets
             * the link to null. This constructor is used when the user is adding a new stock to the queue. and only price
             * is given to the PurchaseNode constructor.
             */
            this(initPrice, null);                             // use the constructor that handles price and link. set link
                                                               // to null and use the parameter initPrice in place of the
                                                               // twin formal parameter.
        }

        public PurchaseNode(float initPrice, PurchaseNode initLink) {
            /*
             * PurchaseNode constructor Description: This constructor creates a new node with the value of the stock and sets
             * the link to the next node in the linked list.
             */
            price = initPrice;                                // set the price of the stock to the parameter initPrice.
            link = initLink;                                  // set the link of the stock to the parameter initLink.
        }

    }
}
```

```java
/*
 * QueueApp description: This class houses the main method and serves as a user interface for interacting with the
 * UserQueue class.(a menu system)The class has a main method that creates a UserQueue object and a variable to
 * store the total gain/loss of the user that is to be displayed at the end of the program.The class utilizes
 * a while loop that quits when the user inputs 3 by means of the getUserFloat method. The getUserFloat
 * method gets the users input with a prompt message or without a prompt. The method serves 2 purposes
 * it may display the menu of options to the user THEN accept a response or the getUserFloat method
 * may accept the users floating point number with no menu prompt at the discretion of the needs
 * of the program e.g. when asking for price of stock OR a menu choice.
 */
public class QueueApp {

    public static void main(String[] args) {

        //instance variables
        final int STOCK_QUANTITY = 10;                                  // define the quantity of stocks that the user can
                                                                        // buy/sell at a time.

        UserQueue queue = new UserQueue();                              // create a new queue object.

        System.out.println("Welcome to the Queue App");                 // print message to user that the program has started.

        float gainLossTotal = 0;                                        // create a variable to store the total gain/loss
                                                                        // of the user.

        float choice = -1;                                              // create a variable to store the user's choice
                                                                        // for the menu options.
// while user input is NOT 3 THEN ask the user for their choice of operation on the queue.
while (choice != 3) {


    choice = getUserFloat(false);                                 // get the user's choice for the menu options and store it in the

    // menu options
    if (choice == 0) {                                            // opt1: enqueue[item to the queue]
        float temp;
        System.out.print("The stock costs: $");                   // print message to user.

        temp = getUserFloat(true);                                // getting user input for what float they wish to add to the queue

        queue.enqueue(temp);                                      // push the user input to the queue


                                                                  // print message to user. that the user has added the
                                                                  // stock to the queue
        System.out.print("You just purchaced 10 stocks at $" + temp + " totaling $" + temp * STOCK_QUANTITY + "\n\n");

        continue;                                                 // continue the loop from the top.
    }
```

```
if (choice == 1) {                                              // opt2: dequeue[remove item from the queue]
    // if the queue as an element pop the element, if there are no items in the queue print message to user.
    if (queue.isEmpty() == false) {

        float temp;                                             // create a variable to store the current
                                                                // value of stock before sell(temp)

                                                                // ask user for the current value of the stock
        System.out.print("The stock is currently valued at: $");
                                                                // subtract the value of the stock purchaced from the same
                                                                // stock's current value.

                                                                // get the users responce with getUserFloat(true) because
                                                                // no menu screen is needed for price input

        // get the gain/loss of the stock with current price(getUserFloat) and old price(viewFront)
        // multiply by 10 because the user can only buy/sell 10 stocks at a time.
        temp = (getUserFloat(true) - queue.viewFront()) * STOCK_QUANTITY; //do ascribed actions on this line

        gainLossTotal += temp;                                  // add the value of the stock to the total gain/loss

        System.out.println("Gained: $" + temp + "\n");          // print the gain/loss to the user

        queue.dequeue();                                        // remove the stock from the queue head

    } else {
        System.out.println("NO STOCKS TO SELL!\n");             // if the queue is empty print message to user. that the
                                                                // queue is empty here.
    }
    continue;                                                   // continue the loop from the top.
}
```

```
        if (choice == 2) {                                             // opt3: view front [look at the front of the queue]

            if (queue.isEmpty() == false) {                            // if the queue has a node
                                                                       // THEN print the value of the node (front node)
                System.out.println(
                        "The item you have currently owned the longest was bought at: $" + queue.viewFront() + "\n");

            } else {
                System.out.println("NO STOCKS OWNED TO CHECK!\n");     // print message to user. if queue is empty here
            }
            continue;                                                  // continue the loop from the top.
        }




        if (choice == 3) {                                             // opt4: quit[exit the program]
            break;                                                     // break the loop
        }
        System.out.println("NOT VALID MENU INPUT!\n");                 // print message to user. That the input is not valid
    }

    // end menu
    System.out.println("\nTotal Capital gain/loss: $" + gainLossTotal);// print the total gain/loss to the user before
                                                                       // ending main() and after final G&L is calculated.
}
```

```
  public static float getUserFloat(boolean forPriceInput) {
        /*
         * This method gets the users input with a prompt message or without a prompt.
         * If the method is called with the intent that the user input will be used for
         * a menu choice
         * THEN the menu prompt will display, otherwise the method will accept the users
         * floating point
         * number with no menu prompt.
         */
        Scanner sc = new Scanner(System.in);           // create a scanner object to get user input
        if (!forPriceInput) {                          // if the user input is not for the price of the stock THEN
                                                       // print the menu options

            System.out.println("0\tPurchase 10 shares of stock.");
            System.out.println("1\tSell 10 shares of stock.");
            System.out.println("2\tCheck the purchace price of the oldest 10 owned shares of stock.");
            System.out.println("3\tQuit");
            System.out.print("[As a floating point value] Enter your choice: ");
        }


        return sc.nextFloat();                         // THEN return and call for the user to input a float
                                                       // regardless of the boolean value of for PriceInput.


    }


}
```