Assignment1

StudentID: 1098648 StudentName: Xinyu Wang (Wendell) Section: W01

DQ Question: What+how/why? What program would you write to solve the following problems and why does it work? Please also comment on other students' code at least three times.

1) Create an ArrayListReview class with one data field of ArrayList and one with LinkedList with the generic type passed to the class.

```java
public class ArrayListReview<E> {
    ArrayList<E> al;
    LinkedList<E> ll;

    ArrayListReview() {
        this.al = new ArrayList<>();
        this.ll = new LinkedList<>();
    }
}
```

2) Create a constructor that populate the ArrayList and the LinkedList filled with the generic type through inserting new elements into the specified location index-i in the list.

```java
public class ArrayListReview<E> {
    ArrayList<E> al;
    LinkedList<E> ll;

    // init an empty object
    ArrayListReview() {
        this.al = new ArrayList<>();
        this.ll = new LinkedList<>();
    }

    // init with a given list
    ArrayListReview(E[] list) {
        this.al = new ArrayList<>(Arrays.asList(list));
        this.ll = new LinkedList<>(Arrays.asList(list));
    }

    // insert new element to the list
    E insert(E element, int i) {
        this.al.add(i, element);
        this.ll.add(i, element);

        return element;
    }
}
```

Code for Test:

```java
ArrayListReview<Integer> arrayListReview = new ArrayListReview<>();
arrayListReview.insert(120, 0);
arrayListReview.insert(110, 1);
arrayListReview.insert(100, 1);
arrayListReview.insert(90, 1);
System.out.println(arrayListReview.ll.toString());
System.out.println(arrayListReview.al.toString());
```

3) Write the code to print the No. 50 Fibonacci number.

```java
public class AS01Q03 {
    public static void main(String[] args) {
        System.out.println(factorial(50));
    }

    private static int factorial(int value) {
        if (value < 1)  return 0;
        if (value == 1) return 1;

        return value * factorial(value - 1);
    }
}
```

4) Create an order processing system that reads pricing information for fruits, and then processes invoices from customers, determining the total amount for each invoice based on the type and quantity of fruit for each line item in the invoice. The program input starts with the pricing information. Each fruit price (single quantity) is specified on a single line, with the fruit name followed by the price. You can assume that each fruit name is a single word consisting of alphabetic characters (A–Z and a–z). You can also assume that prices will have exactly two decimal places after the decimal point. Fruit names and prices are separated by a single space character. The list of fruit prices is terminated by a single line consisting of the text END_PRICES. After the fruit prices, there will be one or more invoices. Each invoice consists of a series of line items. A line item is a fruit name followed by an integer quantity, with the fruit name and quantity separated by a single space. You can assume that no line item will specify a fruit name that is not specified in the fruit prices. Each invoice is terminated by a line consisting of the text END_INVOICE. As a special case, if a line with the text QUIT appears instead of the beginning of an invoice, the program should exit immediately. The overall input will always be terminated by a QUIT line.

Example input:

orange 0.80
pomegranate 2.50
plum 1.20
peach 1.00
persimmon 1.75
lime 0.60
END_PRICES
persimmon 2
orange 3
peach 1
plum 10
pomegranate 5
END_INVOICE
peach 11
plum 5
orange 1
lime 9
END_INVOICE
QUIT

For each invoice, the program should print a single line of the form Total: X.YY where X.YY is the total cost of the invoice, which is the sum of the costs of all of the line items in the invoice. The cost should be printed with exactly two digits after the decimal point.

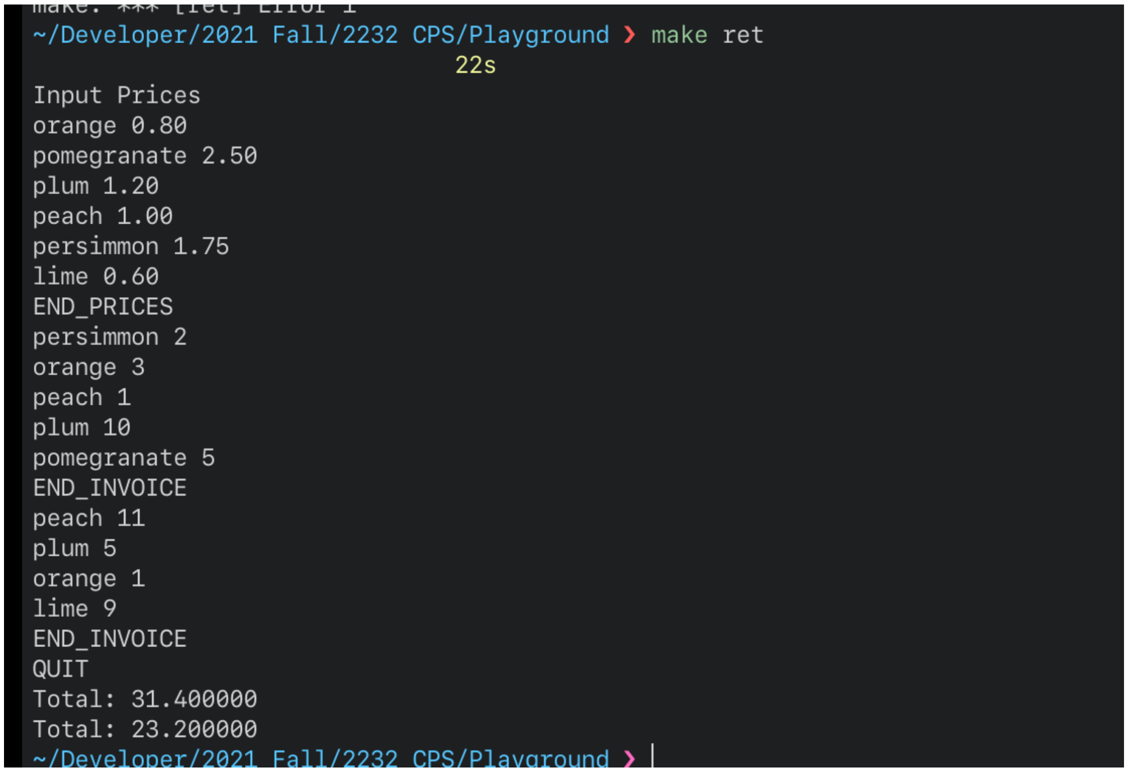
Example output (corresponding to the input shown above):

Total: 31.40

Total: 23.20

```java
class Order {
    LinkedList<Integer> IDs;
    LinkedList<Integer> counts;

    Order() {
        this.IDs = new LinkedList<Integer>();
        this.counts = new LinkedList<Integer>();
    }

    void add(int ID, int count) {
        this.IDs.add(ID);
        this.counts.add(count);
    }

    void add(String buffer, OrderProcessingSystem system) {
        final String[] tempStrings = buffer.split(" ");
        // System.out.println(Arrays.asList(tempStrings).toString());
        final int id = system.orderNameToID(tempStrings[0]);
        final int count = Integer.parseInt(tempStrings[1]);

        this.add(id, count);
    }
}

class OrderProcessingSystem {
    LinkedList<String> names;
    LinkedList<Double> prices;

    int length;

    OrderProcessingSystem() {
        this.names = new LinkedList<String>();
        this.prices = new LinkedList<Double>();
        this.length = 0;
    }

    void addNewGood(String name, double price) {
        names.add(name);
        prices.add(price);
        length += 1;
    }

    void addNewGood(String buffer) {
        final String[] tempStrings = buffer.split(" ");
        // System.out.println(Arrays.asList(tempStrings).toString());
        final double price = Double.parseDouble(tempStrings[1]);
        this.addNewGood(tempStrings[0], price);
    }

    double calculateTotalPrice(LinkedList<Integer> IDs, LinkedList<Integer>
 counts) {
        double total = 0.0;
        for (int i = 0; i < counts.size(); i++) {
            final int id = IDs.get(i);
            final int count = counts.get(i);
            total += prices.get(id) * count;
        }
        return total;
    }

    double calculateTotalPrice(Order order) {
        return calculateTotalPrice(order.IDs, order.counts);
    }

    int orderNameToID(String name) {
        for (int i = 0; i < this.length; i++)
            if (names.get(i).equals(name))
                return i;

        return -1;
    }
}

public class Playground {
    public static void main(String[] args) {
        OrderProcessingSystem system = new OrderProcessingSystem();

        LinkedList<Order> orders = new LinkedList<>();

        Order tmpOrder = new Order();

        final String quitCommand = "QUIT";
        final String endPriceCommand = "END_PRICES";
        final String endInvoiceCommand = "END_INVOICE";

        boolean enteringPrice = true;

        System.out.println("Input Prices");

        Scanner scanner = new Scanner(System.in);

        while (true) {
            String buffer = scanner.nextLine();

            if (buffer.equals(quitCommand)) {
                break;
            };

            if (buffer.equals(endPriceCommand)) {
                enteringPrice = false;
                continue;
            };

            if (enteringPrice) {
                system.addNewGood(buffer);
            } else {
                if (buffer.equals(endInvoiceCommand)) {
                    orders.add(tmpOrder);
                    tmpOrder = new Order();
                } else {
                    tmpOrder.add(buffer, system);
                }
            }
        }
        scanner.close();

        // Show result
        for (int i = 0; i < orders.size(); i++) {
            final double ret = system.calculateTotalPrice(orders.get(i));
            System.out.printf("Total: %f\n", ret);
        }
    }
}
```

Console Running Result below:



```
make: *** [ret] Error 1
~/Developer/2021 Fall/2232 CPS/Playground ❯ make ret
                                 22s
Input Prices
orange 0.80
pomegranate 2.50
plum 1.20
peach 1.00
persimmon 1.75
lime 0.60
END_PRICES
persimmon 2
orange 3
peach 1
plum 10
pomegranate 5
END_INVOICE
peach 11
plum 5
orange 1
lime 9
END_INVOICE
QUIT
Total: 31.400000
Total: 23.200000
~/Developer/2021 Fall/2232 CPS/Playground ❯
```