Q1

import java.util.Scanner;

import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {

        Scanner intInput = new Scanner(System.in);

        Scanner strInput = new Scanner(System.in);

        Queue qHouse = new Queue();

        Queue qSemi\_D = new Queue();

        Queue qTerrace = new Queue();

        Queue temporary = new Queue();

        // b) Input ten (10) objects of houses and store them into qHouse.

        for (int i = 0; i < 10; i++) {

            System.out.print("1. Semi-D\n2. Terrace\nEnter house type: ");

            int typeInt = intInput.nextInt();

            String type = "";

            if (typeInt == 1)

                type = "Semi-D";

            else if (typeInt == 2)

                type = "Terrace";

            else

                System.out.println("Invalid input.");

            System.out.print("Enter location: ");

            String location = strInput.nextLine();

            System.out.print("Enter size (Metre): ");

            double size = intInput.nextDouble();

            System.out.print("Enter price per unit (RM): ");

            double price = intInput.nextDouble();

            qHouse.enqueue(new House(type, location, size, price));

            System.out.println();

        }

        // c) Get all houses from qHouse and store all type of semi-D houses into a

        // queue called qSemi\_D and all terrace houses into a queue called qTerrace.

        while (!qHouse.isEmpty()) {

            House house = (House) qHouse.dequeue();

            if (house.getType().equals("Semi-D")) {

                qSemi\_D.enqueue(house);

            } else if (house.getType().equals("Terrace")) {

                qTerrace.enqueue(house);

            }

            temporary.enqueue(house);

        }

        // restore the house back to qHouse

        while (!temporary.isEmpty()) {

            qHouse.enqueue(temporary.dequeue());

        }

        // d) Display the information of house from qTerrace that the price is less than

        // RM150,000.

        int countTerrace = 0;

        while (!qTerrace.isEmpty()) {

            House house = (House) qTerrace.dequeue();

            if (house.getPrice() < 150000) {

                countTerrace++;

                if (countTerrace == 1)

                    System.out.println("Houses with price less than RM 150,000.00: ");

                System.out.println(house);

            }

        }

        if (countTerrace == 0)

            System.out.println("No houses with price less than RM 150,000.00.");

        // restore the house back to qTerrace

        while (!temporary.isEmpty()) {

            qTerrace.enqueue(temporary.dequeue());

        }

        // e) Count the number of houses that the price is more than RM 300,000.00 and

        // display all information for that houses from qHouse.

        int count = 0;

        while (!qHouse.isEmpty()) {

            House house = (House) qHouse.dequeue();

            if (house.getPrice() > 300000) {

                count++;

                if (count == 1)

                    System.out.println("Houses with price more than RM 300,000.00: ");

                System.out.println(house);

            }

        }

        if (count == 0)

            System.out.println("No houses with price more than RM 300,000.00.");

        // restore the house back to qHouse

        while (!temporary.isEmpty()) {

            qHouse.enqueue(temporary.dequeue());

        }

        System.out.println("Number of houses with price more than RM 300,000.00: " + count);

        strInput.close();

        intInput.close();

    }

}

class Queue extends LinkedList<Object> {

    protected LinkedList<Object> list;

    public Queue() {

        list = new LinkedList<Object>();

    }

    public void enqueue(Object element) {

        list.addFirst(element);

    }

    public Object dequeue() {

        return list.removeLast();

    }

    public boolean isEmpty() {

        return list.isEmpty();

    }

}

class House {

    private String type;

    private String location;

    private double size;

    private double price;

    public House(String type, String location, double size, double price) {

        this.type = type;

        this.location = location;

        this.size = size;

        this.price = price;

    }

    public String getType() {

        return type;

    }

    public String getLocation() {

        return location;

    }

    public double getSize() {

        return size;

    }

    public double getPrice() {

        return price;

    }

    @Override

    public String toString() {

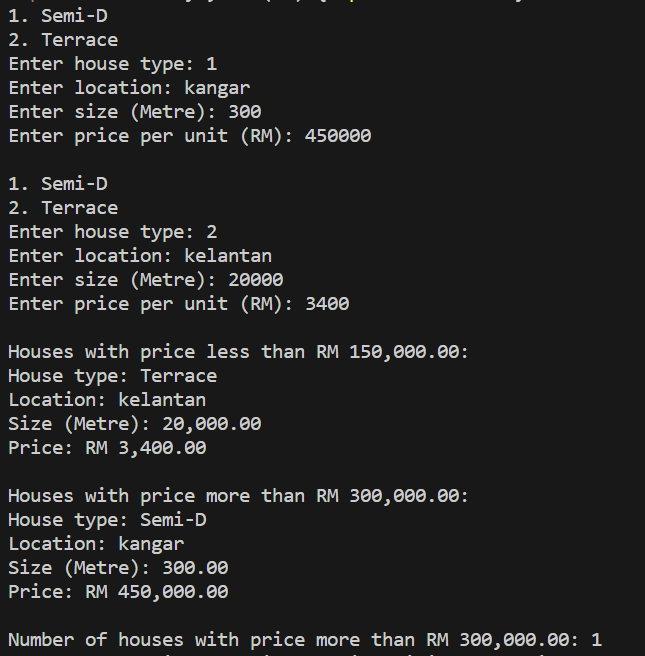
        return "House type: " + type + "\nLocation: " + location + "\nSize (Metre): " + String.format("%,.2f", size)

                + "\nPrice: RM " + String.format("%,.2f", price) + "\n";

    }

}

Sample Input/Output



Q2

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner strInput = new Scanner(System.in);

        Scanner intInput = new Scanner(System.in);

        QUEUE qCustomer = new QUEUE();

        QUEUE qQualify = new QUEUE();

        System.out.print("Please enter number of records: ");

        int rec = intInput.nextInt();

        System.out.println();

        for (int i = 0; i < rec; i++) {

            System.out.print("Enter customer name: ");

            String name = strInput.nextLine();

            System.out.print("Enter account number: ");

            int accountNo = intInput.nextInt();

            System.out.print("Enter saving (RM): ");

            double saving = intInput.nextDouble();

            System.out.print("Enter total transaction (RM): ");

            double totalTransaction = intInput.nextDouble();

            Customer customer = new Customer(name, accountNo, saving, totalTransaction);

            qCustomer.enqueue(customer);

            if (customer.process()) {

                qQualify.enqueue(customer);

            }

            System.out.println();

        }

        System.out.println("List of customers that has more than RM 1000 saving after transaction:\n");

        while (!qQualify.isEmpty()) {

            System.out.println(qQualify.dequeue() + "\n");

        }

        intInput.close();

        strInput.close();

    }

}

class Customer {

    private String name;

    private int accountNo;

    private double saving;

    private double totalTransaction;

    public Customer(String name, int accountNo, double saving, double totalTransaction) {

        this.name = name;

        this.accountNo = accountNo;

        this.saving = saving;

        this.totalTransaction = totalTransaction;

    }

    public String getName() {

        return name;

    }

    public int getAccountNo() {

        return accountNo;

    }

    public double getSaving() {

        return saving;

    }

    public double getTotalTransaction() {

        return totalTransaction;

    }

    @Override

    public String toString() {

        return "Customer name: " + name + "\nAccount No: " + accountNo + "\nSaving: RM "

                + String.format("%,.2f", saving)

                + "\nTotal Transaction: RM " + String.format("%,.2f", totalTransaction);

    }

    public boolean process() {

        return saving - totalTransaction > 1000;

    }

}

class Node {

    Object data;

    Node link;

    public Node(Object elem) {

        this.data = elem;

        this.link = null;

    }

    public Node(Object elem, Node nextElem) {

        this.data = elem;

        this.link = nextElem;

    }

    public Object getData() {

        return data;

    }

    public Node getLink() {

        return link;

    }

}

class ListNode {

    Node first;

    Node last;

    public ListNode() {

        this.first = null;

        this.last = null;

    }

}

class QUEUE extends ListNode {

    public QUEUE() {

        super();

    }

    public void enqueue(Object elem) {

        Node newNode = new Node(elem);

        if (this.first == null) {

            this.first = newNode;

            this.last = newNode;

        } else {

            this.last.link = newNode;

            this.last = newNode;

        }

    }

    public Object dequeue() {

        if (this.first != null) {

            Object data = this.first.data;

            this.first = this.first.link;

            return data;

        }

        return null;

    }

    public boolean isEmpty() {

        return this.first == null;

    }

    public Object getFirst() {

        if (this.first != null) {

            return this.first.data;

        } else {

            return null;

        }

    }

    public Object getNext() {

        if (this.first != null && this.first.link != null) {

            return this.first.link.data;

        } else {

            return null;

        }

    }

    public Object getLast() {

        if (this.last != null) {

            return this.last.data;

        } else {

            return null;

        }

    }

}

Sample Input/Output

A screenshot of a computer

Description automatically generated