Xideral

Java Academy

Week 2 -Day 3

Streams

Presented by:

Edgar Itzak Sánchez Rogers

Monterrey Nuevo León México at 23 august 2024

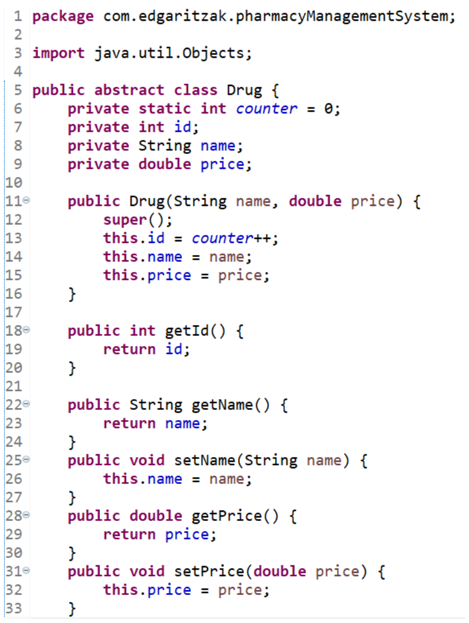
Introduction:

Streams are a sequence of data that can be processed in a declarative and functional style. Streams has the advantage to be easier to read and less verbose. Java streams are more efficient than for loops, streams uses short-circuiting techniques to stop processing as soon as a condition is met. Another advantage of Java Streams is their ability to use lazy evaluation. Stream only performs operations on the elements of the collection as they are needed.

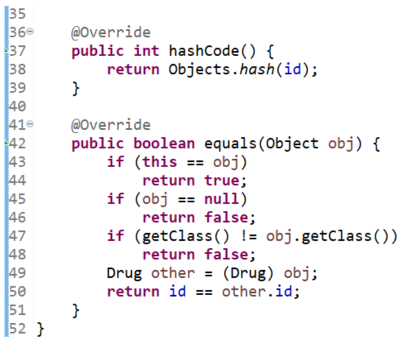
Below are shown 2 cases with the use of streams

Case 1: Pharmacy Inventory Management System

Drug class

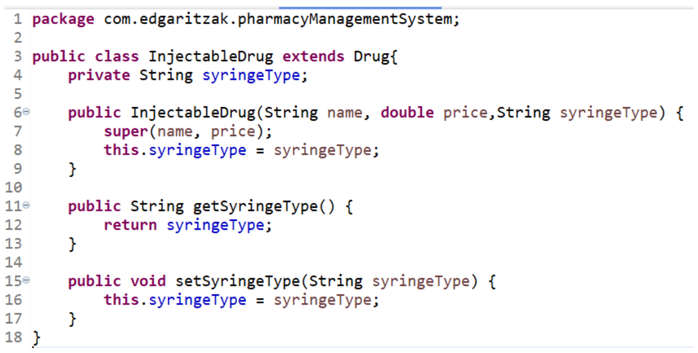


```java
package com.edgaritzak.pharmacyManagementSystem;

import java.util.Objects;

public abstract class Drug {
    private static int counter = 0;
    private int id;
    private String name;
    private double price;

    public Drug(String name, double price) {
        super();
        this.id = counter++;
        this.name = name;
        this.price = price;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
```



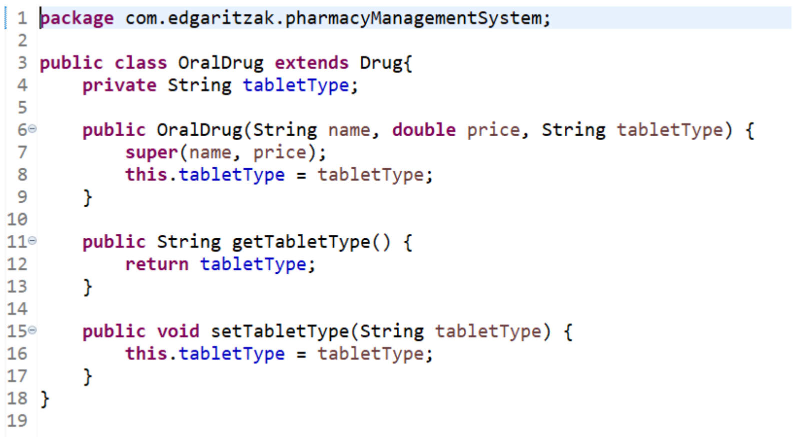
```java

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Drug other = (Drug) obj;
        return id == other.id;
    }
}
```

InjectableDrug class



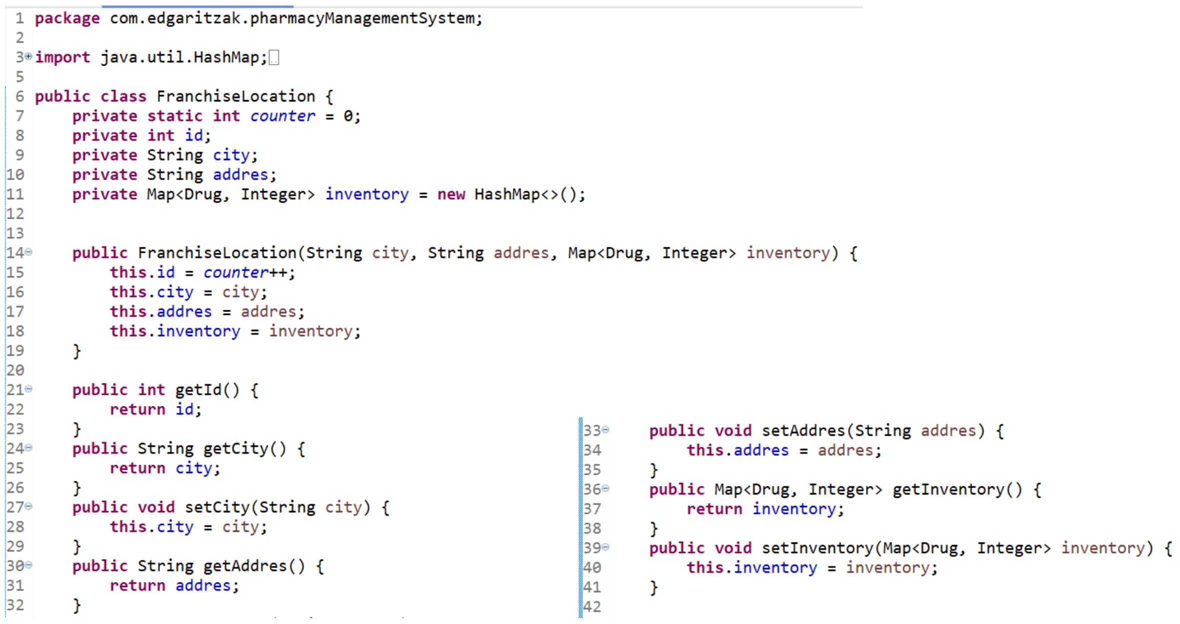
```java
package com.edgaritzak.pharmacyManagementSystem;

public class InjectableDrug extends Drug{
    private String syringeType;

    public InjectableDrug(String name, double price,String syringeType) {
        super(name, price);
        this.syringeType = syringeType;
    }

    public String getSyringeType() {
        return syringeType;
    }

    public void setSyringeType(String syringeType) {
        this.syringeType = syringeType;
    }
}
```

OralDrug class



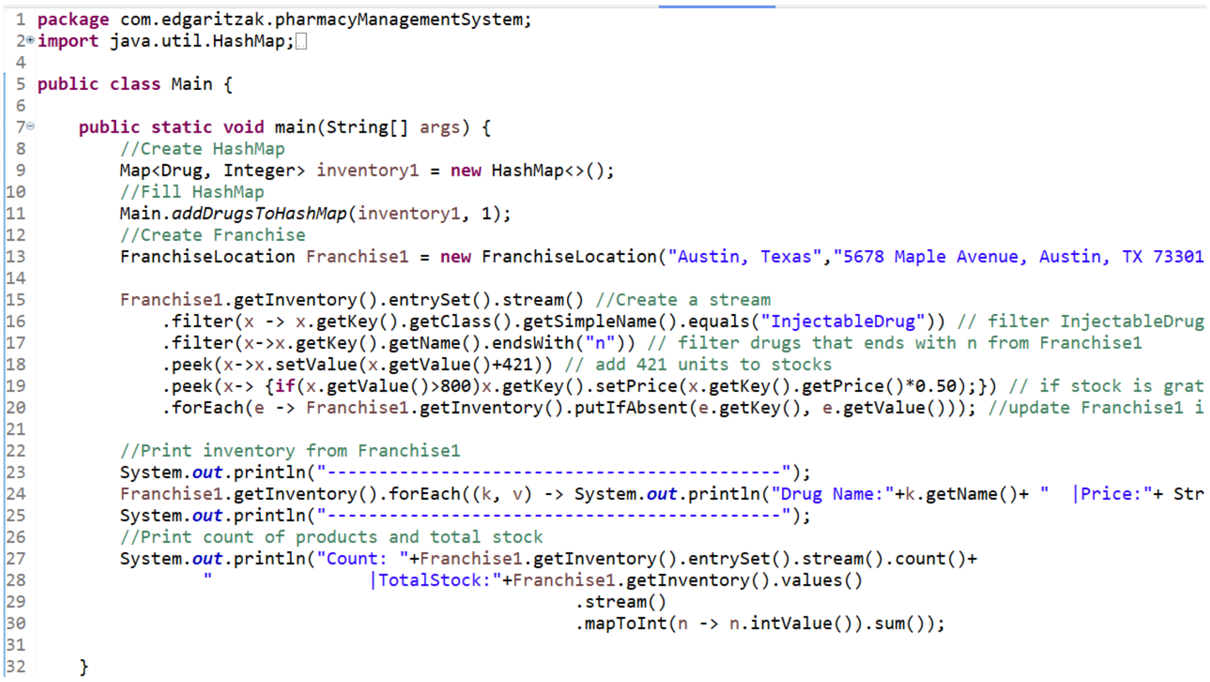
```java
package com.edgaritzak.pharmacyManagementSystem;

public class OralDrug extends Drug{
    private String tabletType;

    public OralDrug(String name, double price, String tabletType) {
        super(name, price);
        this.tabletType = tabletType;
    }

    public String getTabletType() {
        return tabletType;
    }

    public void setTabletType(String tabletType) {
        this.tabletType = tabletType;
    }
}

```

Franchise class



```java
package com.edgaritzak.pharmacyManagementSystem;

import java.util.HashMap;

public class FranchiseLocation {
    private static int counter = 0;
    private int id;
    private String city;
    private String addres;
    private Map<Drug, Integer> inventory = new HashMap<>();


    public FranchiseLocation(String city, String addres, Map<Drug, Integer> inventory) {
        this.id = counter++;
        this.city = city;
        this.addres = addres;
        this.inventory = inventory;
    }

    public int getId() {
        return id;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getAddres() {
        return addres;
    }
    public void setAddres(String addres) {
        this.addres = addres;
    }
    public Map<Drug, Integer> getInventory() {
        return inventory;
    }
    public void setInventory(Map<Drug, Integer> inventory) {
        this.inventory = inventory;
    }

```

Main class



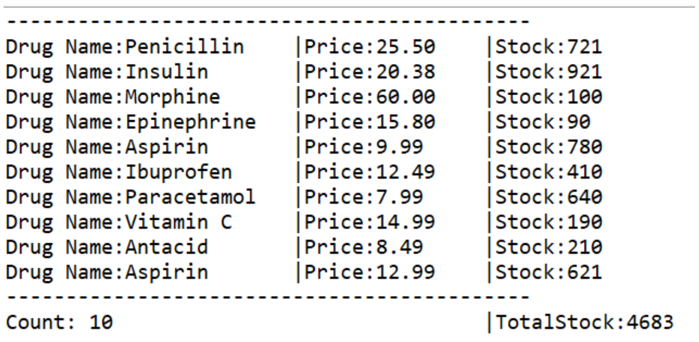
```java
package com.edgaritzak.pharmacyManagementSystem;
import java.util.HashMap;

public class Main {

    public static void main(String[] args) {
        //Create HashMap
        Map<Drug, Integer> inventory1 = new HashMap<>();
        //Fill HashMap
        Main.addDrugsToHashMap(inventory1, 1);
        //Create Franchise
        FranchiseLocation Franchise1 = new FranchiseLocation("Austin, Texas","5678 Maple Avenue, Austin, TX 73301

        Franchise1.getInventory().entrySet().stream() //Create a stream
            .filter(x -> x.getKey().getClass().getSimpleName().equals("InjectableDrug")) // filter InjectableDrug
            .filter(x->x.getKey().getName().endsWith("n")) // filter drugs that ends with n from Franchise1
            .peek(x->x.setValue(x.getValue()+421)) // add 421 units to stocks
            .peek(x-> {if(x.getValue()>800)x.getKey().setPrice(x.getKey().getPrice()*0.50);}) // if stock is grat
            .forEach(e -> Franchise1.getInventory().putIfAbsent(e.getKey(), e.getValue())); //update Franchise1 i

        //Print inventory from Franchise1
        System.out.println("--------------------------------------------");
        Franchise1.getInventory().forEach((k, v) -> System.out.println("Drug Name:"+k.getName()+ "  |Price:"+ Str
        System.out.println("--------------------------------------------");
        //Print count of products and total stock
        System.out.println("Count: "+Franchise1.getInventory().entrySet().stream().count()+
                "               |TotalStock:"+Franchise1.getInventory().values()
                                                    .stream()
                                                    .mapToInt(n -> n.intValue()).sum());

    }
```

Stream explanation:

The purpose of the stream is to filter a drug type, add inventory to the stock, then if some of the updated drugs have more than 800 units, apply a 50% discount, finally update the changes in the franchise hashmap.

- .filter() - keep only “InjectableDrug” type
- .filter() - keep only dugs that ends with ‘n’
- .peeek() - add 421 units to filtered drugs
- .peek() - apply a 50% to drugs that have more than 800 units in stock
- .forEach() -update hashMap

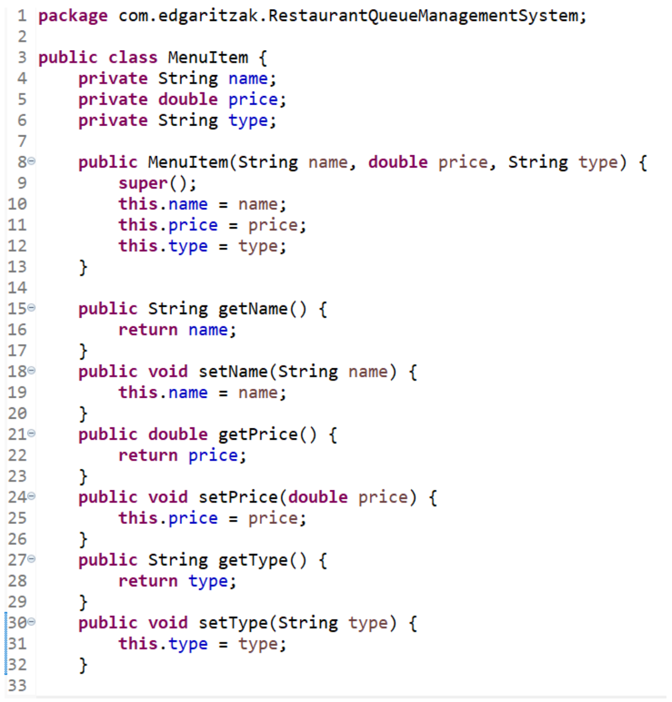
Output:



```
--------------------------------------------
Drug Name:Penicillin    |Price:25.50    |Stock:721
Drug Name:Insulin       |Price:20.38    |Stock:921
Drug Name:Morphine      |Price:60.00    |Stock:100
Drug Name:Epinephrine   |Price:15.80    |Stock:90
Drug Name:Aspirin       |Price:9.99     |Stock:780
Drug Name:Ibuprofen     |Price:12.49    |Stock:410
Drug Name:Paracetamol   |Price:7.99     |Stock:640
Drug Name:Vitamin C     |Price:14.99    |Stock:190
Drug Name:Antacid       |Price:8.49     |Stock:210
Drug Name:Aspirin       |Price:12.99    |Stock:621
--------------------------------------------
Count: 10                               |TotalStock:4683
```

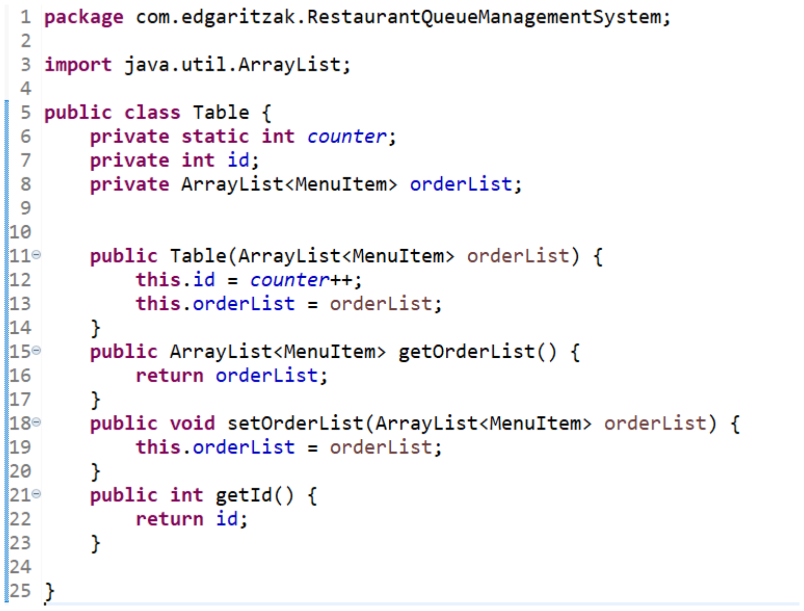
Case 2: Order Queue Management System

MenuItem class



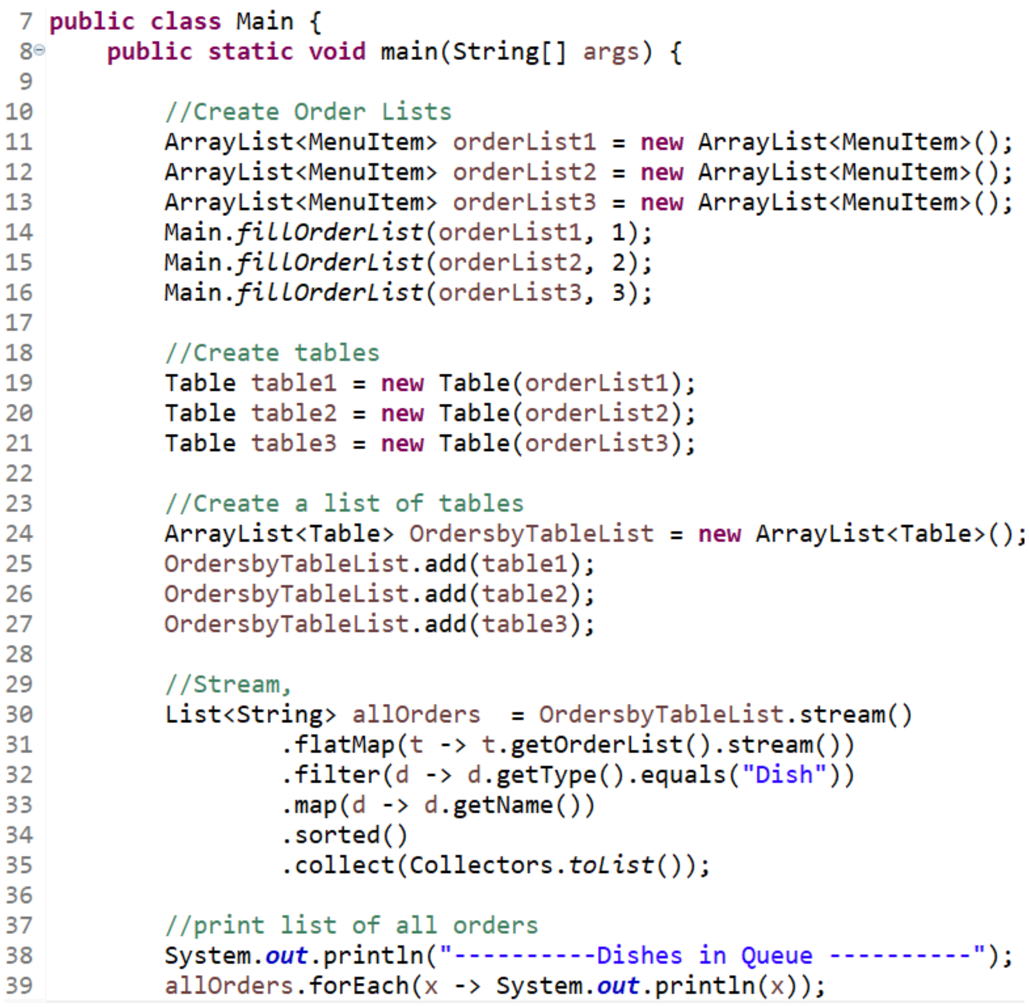
```java
package com.edgaritzak.RestaurantQueueManagementSystem;

public class MenuItem {
    private String name;
    private double price;
    private String type;

    public MenuItem(String name, double price, String type) {
        super();
        this.name = name;
        this.price = price;
        this.type = type;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }

```

Table class



```java
package com.edgaritzak.RestaurantQueueManagementSystem;

import java.util.ArrayList;

public class Table {
    private static int counter;
    private int id;
    private ArrayList<MenuItem> orderList;


    public Table(ArrayList<MenuItem> orderList) {
        this.id = counter++;
        this.orderList = orderList;
    }
    public ArrayList<MenuItem> getOrderList() {
        return orderList;
    }
    public void setOrderList(ArrayList<MenuItem> orderList) {
        this.orderList = orderList;
    }
    public int getId() {
        return id;
    }

}
```

Main class



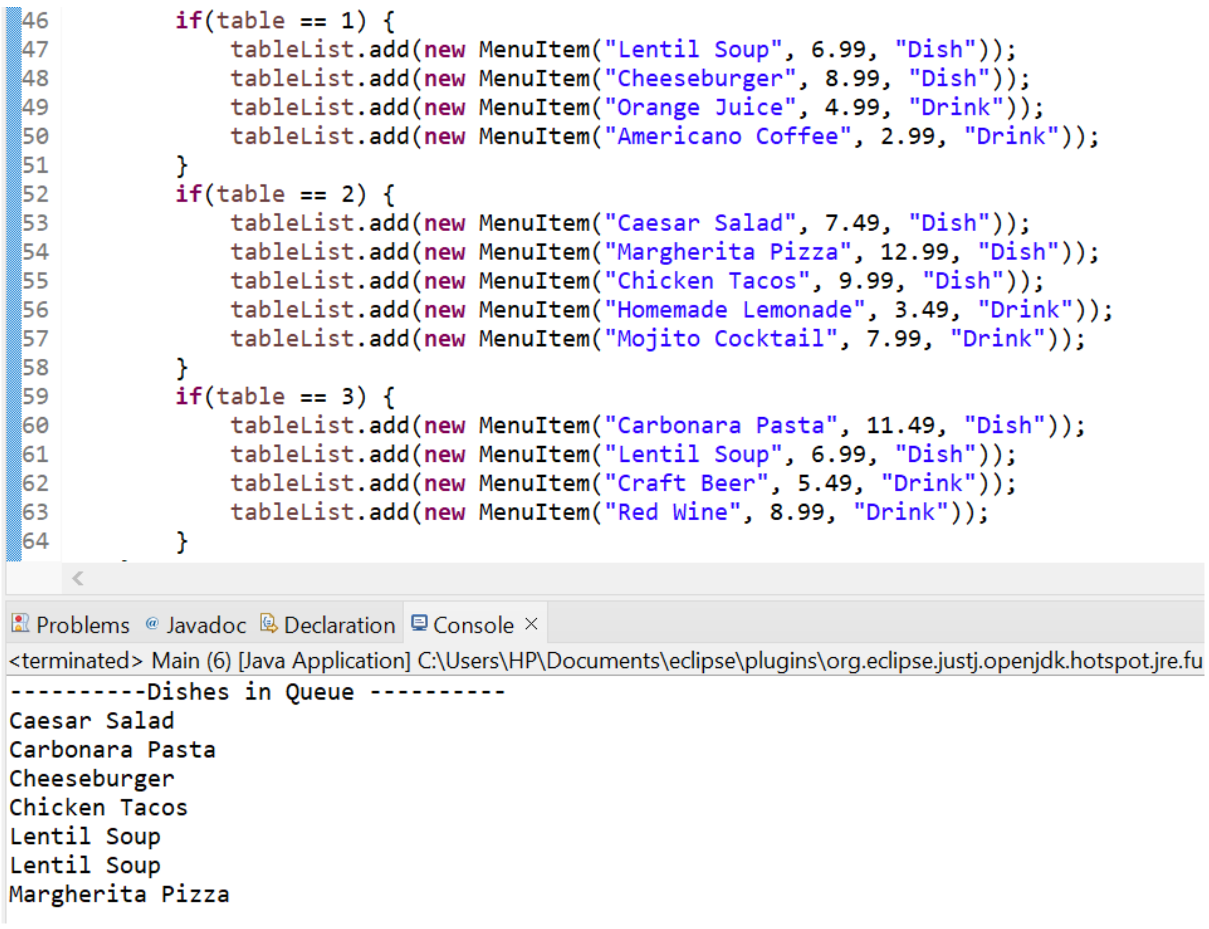
```java
public class Main {
    public static void main(String[] args) {

        //Create Order Lists
        ArrayList<MenuItem> orderList1 = new ArrayList<MenuItem>();
        ArrayList<MenuItem> orderList2 = new ArrayList<MenuItem>();
        ArrayList<MenuItem> orderList3 = new ArrayList<MenuItem>();
        Main.fillOrderList(orderList1, 1);
        Main.fillOrderList(orderList2, 2);
        Main.fillOrderList(orderList3, 3);

        //Create tables
        Table table1 = new Table(orderList1);
        Table table2 = new Table(orderList2);
        Table table3 = new Table(orderList3);

        //Create a list of tables
        ArrayList<Table> OrdersbyTableList = new ArrayList<Table>();
        OrdersbyTableList.add(table1);
        OrdersbyTableList.add(table2);
        OrdersbyTableList.add(table3);

        //Stream,
        List<String> allOrders  = OrdersbyTableList.stream()
                .flatMap(t -> t.getOrderList().stream())
                .filter(d -> d.getType().equals("Dish"))
                .map(d -> d.getName())
                .sorted()
                .collect(Collectors.toList());

        //print list of all orders
        System.out.println("----------Dishes in Queue ----------");
        allOrders.forEach(x -> System.out.println(x));
```

Stream explanation:

The purpose of the stream is to generate a list with names of the items that are “dish” type requested from clients from all tables, order them in alphabetical order and finally show them.

- .flatmap() - transform Table to a stream of MenuItem
- .filter() - keep only dishes
- .map() - transform MenuItem to String, their name
- .sorted() - sort dishes alphabetically
- .collect() - get all items as a list

Output:



```
        if(table == 1) {
            tableList.add(new MenuItem("Lentil Soup", 6.99, "Dish"));
            tableList.add(new MenuItem("Cheeseburger", 8.99, "Dish"));
            tableList.add(new MenuItem("Orange Juice", 4.99, "Drink"));
            tableList.add(new MenuItem("Americano Coffee", 2.99, "Drink"));
        }
        if(table == 2) {
            tableList.add(new MenuItem("Caesar Salad", 7.49, "Dish"));
            tableList.add(new MenuItem("Margherita Pizza", 12.99, "Dish"));
            tableList.add(new MenuItem("Chicken Tacos", 9.99, "Dish"));
            tableList.add(new MenuItem("Homemade Lemonade", 3.49, "Drink"));
            tableList.add(new MenuItem("Mojito Cocktail", 7.99, "Drink"));
        }
        if(table == 3) {
            tableList.add(new MenuItem("Carbonara Pasta", 11.49, "Dish"));
            tableList.add(new MenuItem("Lentil Soup", 6.99, "Dish"));
            tableList.add(new MenuItem("Craft Beer", 5.49, "Drink"));
            tableList.add(new MenuItem("Red Wine", 8.99, "Drink"));
        }
```

```
<terminated> Main (6) [Java Application] C:\Users\HP\Documents\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.fu
----------Dishes in Queue ----------
Caesar Salad
Carbonara Pasta
Cheeseburger
Chicken Tacos
Lentil Soup
Lentil Soup
Margherita Pizza
```

Conclusion:

Once you get used to lambda expressions, streams are a great way to process data. Stream function names are intuitive, so is easy to know what is happening even with nested methods.

References:

[1] How Java Streams Make Your Code More Efficient | LinkedIn

[2] Understanding Java Streams: A Beginner’s Guide | by Techie's Spot | Medium