Inleiding

Deze opdracht bestaat uit twee delen.
In deel 1 van deze opdracht toon je aan dat je het gebruik van Sets en Maps van de Java Collections Framework beheerst. Je mag in dit deel **geen** gebruik maken van Streams.
In deel 2 van de opdracht toon je aan hoe je Streams en Functional Interfaces kunt gebruiken. Je maakt dan juist **alleen maar** gebruik van Streams.

Beschrijving

Een nationale supermarkt franchise ***Jambi*** wil informatie over het aankoopgedrag van de klanten. Je moet de nodige statistieken produceren over de aankopen van de klanten.

Jambi heeft data aangeleverd met aankopen van 250 klanten tijdens hun bezoek aan een supermarkt tussen 12:00 uur en 15:00 uur op een zaterdagmiddag. De data staat in een bestand ***JambiBigJson.txt*** en bevat json-objecten. Daarin staan de producten met een omschrijving en een prijs per eenheid. Van de klanten weten we de postcode, de aankomsttijd bij de kassa's en per klant is een collectie met aangekochte producten aanwezig.

We gaan er van uit dat de aankomsttijd van een klant uniek is. Een product kan uiteraard meer dan 1 keer in de winkelwagen van een klant liggen. Maar de collectie van aankopen (itemsCart) van een klant bevat een product maar één keer samen met het aantal van dat product, zie ook het klassendiagram. Een product mag vaker worden toegevoegd, maar dan moet de collectie het product nog steeds maar één keer registreren. Het aantal zal dan aangepast moeten worden.

Er is een startproject gemaakt met databestanden, met unittests en met handige code om de databestanden in te lezen. Het startproject moet aangevuld en uitgebreid worden, zodat de statistieken gegenereerd kunnen worden. Bestudeer eerst de aanwezig code, met name ook de unittests.
Let op: verderop in dit document behandelen we de aanpak om de gevraagde statistieken te genereren.
Let op: het startproject draait pas goed als je de methode `initializeCollections()` geïmplementeerd hebt.

Statistieken

De ***Supermarket*** klasse in het startproject bevat methodes `printProductStatistics()`, `printCustomerStatistics()` en `printRevenueStatistics()` die gebruikt worden om een overzicht te geven van de aankoop statistieken na het laden van de data uit het bestand. Deze methodes moet het volgende tonen, zie ook het printvoorbeeld.

`printProductStatistics()` toont:

1. De naam van de supermarket en de openings- en sluitingstijden van de periode waarin de data gemeten is.
2. Het aantal klanten dat in de periode aan de kassa's is verschenen, het totaal aantal items dat ze gekocht hebben en het aantal beschikbare producten.
3. Per product de omschrijving van het product met het aantal aankopen van dit product. De lijst producten moet gesorteerd zijn op omschrijving.
4. Per product een opsomming van alle postcodes van klanten die het product hebben gekocht. Een postcode wordt maar 1 keer getoond per product. Deze lijst hoeft niet gesorteerd te zijn.
5. Het product dat door de meeste klanten gekocht is. Dit kunnen eventueel meerdere producten zijn.
6. Per postcode het product dat door klanten met die postcode het meest is gekocht. (Mochten meerdere producten in een postcode op hetzelfde hoogste aantal uitkomen, dan laat je van die producten er willekeurig eentje zien)

`printCustomerStatistics()` toont:

1. De klant met de hoogste rekening en het bedrag van de hoogste rekening.
2. Het aantal klanten per tijdsinterval van 15 minuten.

`printRevenueStatistics()` toont:

1. De totale omzet van alle aangekochte producten bij elkaar en de gemiddelde omzet per klant.
2. De totale omzet van alle klanten per postcode, gesorteerd naar postcode. (De omzet is het totaalbedrag van alle aangekochte producten.)
3. De totale omzet van alle klanten per tijdsinterval van 15 minuten.

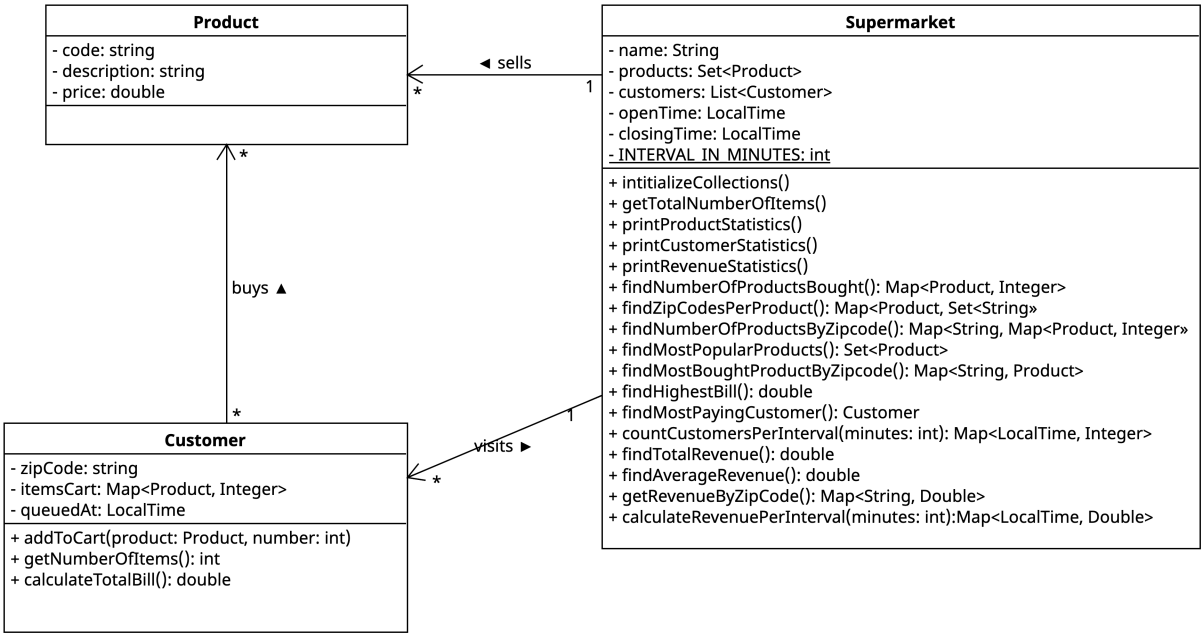
Testing

Het startproject bevat een aantal unittests die je helpen om correcte code te schrijven. De namen van de tests starten met een nummer: ***tXXX***. Hoe lager het nummer hoe fundamenteler de test is voor het programmeren van je oplossing. De nummers verwijzen ook naar de stappen, test tx1x hoort bij stap 1. In de stappen die hieronder beschreven staan schrijf je code die de laag genummerde testen als eerste laten slagen.

Sommige tests gebruiken data in kleinere databestanden genaamd ***jambi1Json.txt***, ***jambi2Json.txt*** en ***jambi5Json.txt***. Je moet zorgen dat alle tests slagen en je moet ook drie tests zelf schrijven. Uiteraard kun je zelf nog meer tests toevoegen. De bestaande tests mag je niet aanpassen. Als een test niet slaagt, dan is de gekozen oplossing niet goed of er ontbreekt wellicht code.

Aanpak

Het startproject bevat de klassen uit het klassendiagram. In de klassen die in het klassendiagram staan ontbreekt nog code, dit is aangegeven met TODO-comments.



Deel 1: Collections

Stap 1

1. Start met de klassen ***Product*** en ***Customer***.
2. Omdat ***Product*** en ***Customer*** in een Set en/of een Map moeten komen, zul je daar methodes aan toe moeten voegen.
3. Voor gebruik in een ***HashMap*** of een ***HashSet*** moet je altijd `equals()` en `hashcode()` overriden (overriden omdat die methodes al wel in de ***Object*** klasse zitten net als `toString()`)
4. **Let wel**: in de opdracht staat iets over uniciteit van de verschillende objecten en je `equals()` en `hashcode()` methodes moeten daar dus bij aansluiten.
5. Voor het gebruik in een ***TreeSet*** of ***TreeMap*** moet je een klasse ***Comparable*** maken en de `compareTo()` methode schrijven.
6. Schrijf nu de gevraagde code voor de `TODO Stap 1` in de twee genoemde klassen.

Stap 2

1. Implementeer ook de methodes `getNumberOfItems()` en `addToCart()` van de ***Customer*** klasse (zie `TODO Stap 2` in de code).
2. Hiervoor moet je goed kijken welke informatie waar staat. Let bijvoorbeeld op de Map ***itemsCart*** die informatie van ***Product*** en aantal bevat.
3. Met *foreach loops* en *getters* kun je deze methodes maken.
4. Gebruik, zoals eerder beschreven, de unittests om te controleren of je code werkt.

Stap 3

1. We beginnen met implementeren van de ***Supermarket*** klasse.
2. Schrijf de code voor de methodes `initializeCollections()` en `getTotalNumberOfItems()`.
3. Start met implementeren van de methode `printProductStatistics()`. Let wel: in dit deel doe je alleen de `TODO`'s van stap 3.
4. Voor de `ToDo`'s in `printProductStatistics()` moet je eerst de methodes `findNumberOfProductsBought()`en `findZipCodesPerProduct()` geschreven hebben. Let wel: je gebruikt in dit deel nog geen ***Streams*** en ***Functional Interfaces***!

Hulp bij `findNumberOfProductsBought()`

5. Hier moet je gezien de signature een ***Map*** maken en retourneren. Bedenk eerst of je een ***TreeMap*** of een ***HashMap*** wilt gebruiken. Gezien de vraag moet er een ***Product*** als *key* gebruikt worden en een aantal als *value* (per product het totaal aantal keer dat dit product gekocht is).
6. Onderzoek hoe je een map vult. Zie vooral de oefening index maken in de zelfstudiewijzer.
7. Merk op dat je gegevens van Customers nodig hebt. Je hebt in de Supermarkt klasse een ***Set*** met ***Customers***! Je kunt dus in een loop elke ***Customer*** langs gaan en dan per ***Customer*** de aankopen van die customer doorlopen. Daarin staat steeds een product met een aantal.
    a. Als het aangekochte product van de ***Customer*** nog niet in de map zit, dan voeg je dat product toe met het bijbehorende aantal.
    b. Als het aangekochte product van de ***Customer*** wel in de map zit, dan pas je het aanwezige '***totaal aantal***' aan door het aantal van deze aankoop erbij op te tellen.
8. Ga nu verder met `findZipCodesPerProducts()`. De aanpak is hetzelfde als hierboven. Let op dat de *value* nu een ***Set*** is.
9. Schrijf de code voor de methode `findNumberOfProductsByZipcode()`. Deze methode ga je gebruiken in de methode `findMostBoughtProductByZipcode()`, die in stap 5 gemaakt wordt.

Hulp bij `findNumberOfProductsByZipcode()`

10. Je moet per zipcode een map maken met alle producten die door klanten met die zipcode gekocht zijn met als *value* het totaal aantal van de aankopen van alle klanten met dezelfde zipcode.
11. Loop alle customers af en controller of de zipcode al in de map voorkomt.
12. Als de zipcode van een customer nieuw is, voeg deze dan toe met als *value* een nieuw `Map<Product, Integer>`. Zet alle producten van de de aankopen van de desbetreffende customer in de map met bijbehorende aantallen.
13. Als de zipcode van een customer al bestaat, dan moet je van alle aankopen van de customer controleren of het product al in de map van deze zipcode zit. Als het product nieuw is dan voeg je het toe met het juiste aantal. Als het product al bestaat, van verhoog je het totaal aantal met het aantal van de betreffende aankoop.
14. Als je de bovenstaande methodes hebt uitgewerkt, dan kun je de TODO stap 3 van de methode `printProductStatistics()` afmaken. Let wel: ***Most popular products*** en ***Most bought products*** doe je nu nog niet.

Deel 2: Streams en Functional Interfaces

Stap 4

Vanaf deze stap moet je gebruik maken van ***Functional Interfaces*** en ***Streams***

1. Implementeer de methode `calculateTotalBill()` in ***Customer*** met behulp van een ***Stream*** en een ***lambda expressie***.
2. Implementeer de methode `printCustomerStatistics()`.
3. In `printCustomerStatistics()` zie je dat de methodes `findHighestBill()`, `findMostPayingCustomer()` en `countCustomersPerInterval()` worden aangeroepen. Je moet dus eerst deze methodes schrijven voordat je de ToDo's in `printCustomerStatistics()` kunt schrijven. Let wel: gebruik in deze stap alleen ***Functional Interfaces*** en ***Streams***.

Hulp bij `countCustomersPerInterval()`

4. Je moet een map maken met als keys de starttijden van de intervallen en als waarde het aantal customers in het interval vanaf de betreffende starttijd. Zet met behulp van een for-loop eerst alle begintijden van de tijdsintervallen in een map met een waarde 0. Loop dan alle customers na en bepaal per customer in welk tijdsinterval de customer zit en verhoog dan de juiste waarde in de map.

Stap 5

In deze stap doe je eerst basale omzet statistieken en daarna complexe product en omzet statistieken. Let wel: gebruik in deze stap alleen ***Functional Interfaces*** en ***Streams***.

1. Implementeer de methode `printRevenueStatistics()`.

2. Je ziet dat de methode `findTotalRevenue()`, `findAverageRevenue()` en `getRevenueByZipCode()` worden aangeroepen. Schrijf deze methodes eerst.
3. Schrijf nu eerst de methode `findMostPopularProducts()`, die nog hoort bij de product statistieken. Tip: Maak eerst een map die per product het aantal winkelwagens telt waar het product in ligt. Gebruik een stream en een collector. Bepaal dan de hoogste waarde met behulp van een stream en daarmee het product met die hoogste waarde.
4. Schrijf de methode `findMostBoughtProductByZipcode()`. Gebruik de methode `findNumberOfProductsByZipcode()` om per zipcode alle producten met aantallen te vinden. Bepaal per zipcode wat het grootste aantal is en zoek dan het product dat bij dit grootste aantal hoort.
5. Implementeer de methode `calculateRevenuePerInterval(int minutes)`. Dit is een uitdagende opdracht. Voor deze opdracht moet je zelf de oplossing bedenken. Gebruik alleen ***forEach***, ***Functional Interfaces*** en ***Streams***.

Stap 6

In deze stap ga je na dat alle aanwezige tests slagen en je schrijft de drie tests in de ***SupermarketTest*** klasse.

1. Controleer dat alle tests in de ProductTest klasse slagen.
2. Controleer dat alle tests in de ***CustomerTest*** klasse slagen.
3. Schrijf de code voor t051_`highestBillIsCorrect()`, `t052_mostPayingCustomerIsCorrect()` en `t061_totalRevenueIsCorrect()` en controleer vervolgens dat alle tests in de ***SupermarketTest*** klasse slagen.

Uitvoer op de console

Uiteindelijk moet je onderstaande uitvoer kunnen tonen.

```
>>> Customer Statistics of 'Jambi' between 12:00 and 15:00

250 customers have shopped 2064 items out of 25 different products

>>>>> Product Statistics of all purchases <<<<<

>>> Products and total number bought:
Aardappelen vastkokend 2.5Kg     72
Bloemkool                       108
Bonne Maman aardbeienjam         67
Calve Pindakaas 650g             74
Campina halfvolle melk 1L        58
Campina magere yoghurt 1.5L      78
Cashew noten 300g                81
Coca Cola Zero 1.5L              49
Croissant                       101
Douwe Egberts snelfilter 500g    74
Eendeborst 500g                  79
Filetlapjes 700g                 73
Gourmet tonijn 100g             122
Hertog Jan 6-pack                89
Kaiser broodje                   70
Kip kilo knaller                103
Multivruchtensap 2L              46
Old Amsterdam stuk 1Kg           97
Paprika                          66
Purina kip adult 1.5Kg           53
Robijn kleur en fijn             93
Robijn stralend wit              75
Studentenhaver 300g              99
Verse scharreleieren 4 stuks     99
Zaanse snijder heel             138
```

```
>>> Products and zipcodes
Campina magere yoghurt 1.5L:
    1013JG, 1014KH, 1017EH, 1013BK, 1015DM, 1017FO, 1015LI, 1016MJ,
    1017NK, 1013AD, 1013KN, 1014LO, 1014BE, 1015CF, 1016NQ, 1016DG
Studentenhaver 300g:
    1013JG, 1014KH, 1017OR, 1013BK, 1015DM, 1016EN, 1017FO, 1015LI,
    1017NK, 1013KN, 1014BE, 1014LO, 1015MP, 1015CF, 1016DG
Cashew noten 300g:
    1017OR, 1017EH, 1013BK, 1014CL, 1016EN, 1017FO, 1015LI, 1016MJ,
    1017NK, 1013KN, 1014BE, 1015MP, 1015CF, 1016NQ, 1016DG
Campina halfvolle melk 1L:
    1017EH, 1014CL, 1015DM, 1015LI, 1016MJ, 1017NK, 1013KN, 1013AD,
    1015MP, 1015CF, 1016DG
Old Amsterdam stuk 1Kg:
    1013JG, 1017EH, 1014CL, 1015DM, 1016EN, 1017FO, 1015LI, 1016MJ,
    1017NK, 1013KN, 1013AD, 1014LO, 1015MP, 1016DG, 1016NQ
Verse scharreleieren 4 stuks:
    1017OR, 1017EH, 1013BK, 1014CL, 1015DM, 1016EN, 1017FO, 1017NK,
    1013KN, 1013AD, 1014LO, 1014BE, 1015CF, 1015MP, 1016NQ
Bonne Maman aardbeienjam:
    1017EH, 1013BK, 1014CL, 1013JG, 1014KH, 1015LI, 1017NK, 1013AD,
    1013KN, 1014BE, 1016NQ
Coca Cola Zero 1.5L:
    1013JG, 1014KH, 1017OR, 1017EH, 1014CL, 1015DM, 1016EN, 1017NK,
    1013KN, 1014BE, 1015CF, 1015MP, 1016NQ, 1016DG
Hertog Jan 6-pack:
    1014KH, 1013BK, 1015DM, 1016EN, 1017FO, 1016MJ, 1017NK, 1013KN,
    1013AD, 1014BE, 1014LO, 1015CF, 1015MP, 1016DG, 1016NQ
Calve Pindakaas 650g:
    1017EH, 1013BK, 1015DM, 1013JG, 1017FO, 1015LI, 1017NK, 1013KN,
    1014BE, 1015CF, 1015MP
.....
.....
```

```
>>> Most popular products

Product(s) bought by most customers:
     Studentenhaver 300g
```

```
>>> Most bought products per zipcode

1013AD   Gourmet tonijn 100g
1013BK   Aardappelen vastkokend 2.5Kg
1013JG   Zaanse snijder heel
1013KN   Croissant
1014BE   Filetlapjes 700g
1014CL   Zaanse snijder heel
1014KH   Studentenhaver 300g
1014LO   Verse scharreleieren 4 stuks
1015CF   Calve Pindakaas 650g
1015DM   Kip kilo knaller
1015LI   Campina halfvolle melk 1L
1015MP   Gourmet tonijn 100g
1016DG   Old Amsterdam stuk 1Kg
1016EN   Croissant
1016MJ   Cashew noten 300g
1016NQ   Cashew noten 300g
1017EH   Old Amsterdam stuk 1Kg
1017FO   Kip kilo knaller
1017NK   Eendeborst 500g
1017OR   Studentenhaver 300g
```

```
>>>>> Customer Statistics of all purchases <<<<<

Customer that has the highest bill of 302.70 euro:
queuedAt: 14:16:12
zipCode: 1015LI
Purchases:
     Eendeborst 500g: 14
     Purina kip adult 1.5Kg: 1
     Campina magere yoghurt 1.5L: 2
     Robijn kleur en fijn: 1

>>> Time intervals with number of customers

Between 12:00 and 12:15 the number of customers was 16
Between 12:15 and 12:30 the number of customers was 18
Between 12:30 and 12:45 the number of customers was 19
Between 12:45 and 13:00 the number of customers was 29
Between 13:00 and 13:15 the number of customers was 21
Between 13:15 and 13:30 the number of customers was 28
Between 13:30 and 13:45 the number of customers was 13
Between 13:45 and 14:00 the number of customers was 22
Between 14:00 and 14:15 the number of customers was 27
Between 14:15 and 14:30 the number of customers was 22
Between 14:30 and 14:45 the number of customers was 16
Between 14:45 and 15:00 the number of customers was 19
```

```
>>>>> Revenue Statistics of all purchases <<<<<

Total revenue = 8480.86
Average revenue per customer = 33.92

>>> Revenues per zip-code:

1013AD   216.98
1013BK   368.19
1013JG   297.74
1013KN   737.30
1014BE   473.67
1014CL   270.57
1014KH   434.12
1014LO   323.63
1015CF   529.38
1015DM   388.08
1015LI   757.07
1015MP   507.89
1016DG   455.67
1016EN   214.09
1016MJ   316.55
1016NQ   322.92
1017EH   461.75
1017FO   257.71
1017NK   982.31
1017OR   165.24

>>> Revenues per interval of 15 minutes

Between 12:00 and 12:15 the revenue was 577.33
Between 12:15 and 12:30 the revenue was 561.73
Between 12:30 and 12:45 the revenue was 559.09
Between 12:45 and 13:00 the revenue was 931.52
Between 13:00 and 13:15 the revenue was 1063.15
Between 13:15 and 13:30 the revenue was 760.56
Between 13:30 and 13:45 the revenue was 231.54
Between 13:45 and 14:00 the revenue was 682.01
Between 14:00 and 14:15 the revenue was 1088.64
Between 14:15 and 14:30 the revenue was 1005.24
Between 14:30 and 14:45 the revenue was 362.37
Between 14:45 and 15:00 the revenue was 657.68
```