Soit la classe Dish

**public** **class** Dish {

**private** **final** String name;

**private** **final** **boolean** vegetarian;

**private** **final** **int** calories;

**private** **final** Type type;

**public** Dish(String name, **boolean** vegetarian, **int** calories, Type type) {

**this**.name = name;

**this**.vegetarian = vegetarian;

**this**.calories = calories;

**this**.type = type;

}

**public** String getName() {

**return** name;

}

**public** **boolean** isVegetarian() {

**return** vegetarian;

}

**public** **int** getCalories() {

**return** calories;

}

**public** Type getType() {

**return** type;

}

**public** **enum** Type {

***MEAT***,

***FISH***,

***OTHER***

}

@Override

**public** String toString() {

**return** name;

}

**public** **static** **final** List<Dish> ***menu*** = Arrays.*asList*(

**new** Dish("pork", **false**, 800, Dish.Type.***MEAT***),

**new** Dish("beef", **false**, 700, Dish.Type.***MEAT***),

**new** Dish("chicken", **false**, 400, Dish.Type.***MEAT***),

**new** Dish("french fries", **true**, 530, Dish.Type.***OTHER***),

**new** Dish("rice", **true**, 350, Dish.Type.***OTHER***),

**new** Dish("season fruit", **true**, 120, Dish.Type.***OTHER***),

**new** Dish("pizza", **true**, 550, Dish.Type.***OTHER***),

**new** Dish("prawns", **false**, 400, Dish.Type.***FISH***),

**new** Dish("salmon", **false**, 450, Dish.Type.***FISH***)

);

}

1. Filtrer les objets Dish de type « Vegetarian » ;
2. Trier la liste des dishes, puis filtrer les objets Dish suivant Calories < 320 ;
3. Utiliser la méthode takeWhile() pour filtrer la liste (Calories> 300);
4. Utiliser la méthode dropWhile() pour filtrer la liste (Calories> 300);
5. Filter la liste des objets Dish, puis utiliser la fonction skip;
6. Utiliser la fonction anyMatch pour vérifier l’existence d’un objet Dish de type Vegetarian ;
7. Utiliser la fonction allMatch pour s’assurer que tous les objets Dish ont des calories<1000 ;
8. Utiliser la fonction noneMatch pour s’assurer qu’aucun objet Dish n’a pas de calories>=1000 ;
9. Soit la liste : List<Integer> numbers = Arrays.asList(3, 4, 5, 1, 2); et à l’aide de la fonction reduce déclarée dans les Streams,
   1. Calculer la somme de numbers ;
   2. Renvoyer la valeur maximale de cette liste ;
   3. Renvoyer la valeur minimale de cette liste ;
10. à l’aide de la fonction reduce, calculer la somme des calories de la liste menu.

Exercice 2 :

Soit la classe Trader :

**public** **class** Trader {

**private** String name;

**private** String city;

**public** Trader(String n, String c) {

name = n;

city = c;

}

**public** String getName() {

**return** name;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String newCity) {

city = newCity;

}

@Override

**public** **int** hashCode() {

**return** (name == **null** ? 0 : name.hashCode())+ (city == **null** ? 0 : city.hashCode());

}

@Override

**public** **boolean** equals(Object other) {

**if** (other == **this**) {

**return** **true**;

}

**if** (!(other **instanceof** Trader)) {

**return** **false**;

}

Trader o = (Trader) other;

**boolean** eq = Objects.equals(name, o.getName());

eq = eq && Objects.equals(city, o.getCity());

**return** eq;

}

@Override

**public** String toString() {

**return** String.*format*("Trader:%s in %s", name, city);

}

}

**public** **class** Transaction {

**private** Trader trader;

**private** **int** year;

**private** **int** value;

**public** Transaction(Trader trader, **int** year, **int** value) {

**this**.trader = trader;

**this**.year = year;

**this**.value = value;

}

**public** Trader getTrader() {

**return** trader;

}

**public** **int** getYear() {

**return** year;

}

**public** **int** getValue() {

**return** value;

}

@Override

**public** **int** hashCode() {

**return** (trader == **null** ? 0 : trader.hashCode()) + year + value;

}

@Override

**public** **boolean** equals(Object other) {

**if** (other **instanceof** Transaction) {

Transaction o = (Transaction) other;

**return** **this**.trader.equals(o.trader) && **this**.year == o.year && **this**.value == o.value;

}

**return** **false**;

}

@Override

**public** String toString() {

**return** trader + ", year : " + year + ", value: " + value;

}

}

Soient les objets :

Trader raoul = **new** Trader("Raoul", "Cambridge");

Trader mario = **new** Trader("Mario", "Milan");

Trader alan = **new** Trader("Alan", "Cambridge");

Trader brian = **new** Trader("Brian", "Cambridge");

List<Transaction> transactions = Arrays.*asList*(

**new** Transaction(brian, 2011, 300),

**new** Transaction(raoul, 2012, 1000),

**new** Transaction(raoul, 2011, 400),

**new** Transaction(mario, 2012, 710),

**new** Transaction(mario, 2012, 700),

**new** Transaction(alan, 2012, 950)

);

1. Récupérer les transactions effectuées en 2011, trier le résultat en fonction de leur valeur(value) ;
2. Récupérer toutes les villes (sans les doublons) dont les «traders » travaillent ;
3. Récupérer tous les « traders » résidant dans Cambridge, trier les en fonction du name ;
4. Récupérer les noms des « traders » triés par l’ordre alphabétique ;
5. Y’a-t-il un trader résidant à Milan ?
6. Renvoyer le maximum des transactions ;
7. Trouver la valeur minimale des transactions.