IT Academy 2018 Java stream

Java Collections Framework™

Introduction

All development shall be done using IntelliJ IDEA. You must follow the good coding practices and format the files. Follow the instructions and make sure that all constraints are fulfilled. Good luck!

Project setup:

1. Open Intellij IDEA.
2. Open Java project you've worked on previous lecture (day 3).
3. All tasks should be developed on top of yesterday's code.

Requirements and general constraints:

- You will be provided with some expected output to check, if your solution is correct, against.
- You don't need to write any additional verification code.
- Expected output is generated by test code located in Sharepoint (same place you've found this document). If a task has an expected output specified, you should be able to find a verification class named "TaskXTest.java" in a workspace directory. Copy this file into your Java project. Preferably, put it in a package named "**`lt.swedbank.itacademy`**". Most probably, this class won't compile right out-the-bat. Fix imports and setter methods names (if they are different from ones in your **`Loan`*** classes).
- You are not allowed to change logic inside these test class to make them pass. No cheating!

Warmup: refactoring the code

Task 1. Bye bye arrays:

1. Go through your existing code and find all the places where arrays are used.
2. Replace all arrays, expect ones used in **`DomainInitializer`** and **`LoanService`** constructor, in your services with the most appropriate interface/class from **`List`** hierarchy. Remember: you should always choose the least specific available type possible. This task involves changing:
   a) methods parameters types
   b) methods return types
   c) local variables
   d) array dependent algorithms (initialization, searching, iteration, etc.)

Collections: working with `Set` and `Map`

Task 2. Get vehicle models:

- In **`LoanService`** create a method **`findVehicleModels()`**.
- Method should return a collection of distinct vehicle models.

```
Audi A3
BMW i8
Scout Traveler
Alfa Romeo Spider
```

Task 3. Group loans by risk:

- In **`LoanService`** create a method **`groupLoansByRiskType()`**.
- Method should return a map containing **`LoanRiskType`** (key) – **`Loan`**s (value) pairs.

```
HIGH_RISK -> 4
NORMAL_RISK -> 2 3 6
LOW_RISK -> 1 5
```

Task 4. Make `Loans` compatible with JCF:

- Correctly override both **`equals()`** and **`hashCode()`** methods in **`Loan`** class (and it's subclasses).
- At a minimum, it should be done to **`Loan`**, **`VehicleLoan`** and **`RealEstateLoan`** classes.

```
Testing equals()...
true  true
true  false
true  false
false  false
false  false
false  false
false  false
false  false
false  false
false  false
false  false
true
false

Testing hashCode()...
2
true
false
```

Sorting: natural order and comparators

Task 5. Compare `Car(Loan)`s:

- People like to compare (aka. brag about) their car(s). Price, horse power, discount they got while buying one are the key factors for assessing one's (oh, I mean car's) worth.
- Make **`CarLoan`**s comparable.
- **`CarLoan`**s should come in this natural order:
  - priciest first, then
  - most powerful, then
  - lowest loan interest rate.

```
4 18 16 8 14 15 1 10 11 17 5 13 2 3 6 12 7 9
```

Task 6. Prioritize loans:

- In **`LoanService`** create a method **`prioritizeLoans()`**.
- Method should return a set of **`Loan`** objects.
- Prioritization should be done like this:
  - high risk loans should come first, then
  - highest in total cost, then
  - oldest (creation date).

```
8 7 6 5 4 3 2 1 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9
```

Going deeper: extending the framework

Task 7. Create a custom `Loan` collection for data loading

For this task your job is to create the means for providing data (loans) to **`LoanService`**. These "means" should be implemented as custom collection. **`LoanService`** should access **`Loan`** data using this collection.

1. Create a class called **`LoanIterable`** implementing **`java.lang.Iterable`** interface.
2. Implement **`Iterator<T> iterator();`** method by providing an instance of a custom **`Iterator`** implementation:
   a) **`Iterator`** should be implemented as an inner anonymous class.
   b) **`Iterator`** should iterate through an array of **`Loan`** type/sub-type objects, in other words, it should use an underlying collection in a form of an array. This array should be stored in **`LoanIterable`** and initialized with a constructor.
   c) Implement methods **`hasNext()`** and **`next()`**. Implementation should be straight forward:
      i. If underlying collection has a next element, **`hasNext()`** should return **`true`** (**`false`** otherwise).
      ii. You should not handle situations, when **`next()`** is called even though **`hasNext()`** returns **`false`** (there is no next element).
3. **`LoanIterable`** should replace "all loans pool" used in **`LoanService`** before this point. You should pass an instance of it using a **`LoanService`** constructor.