Step 14 – Optional, Lambda, Validate Input

Think about registering a bank loan.

Task 1:

Create a very simple Customer class. It has two attributes: String name; Integer age. Create Getters/Setters and Constructor.

Create LoanInfo class. It is a simplified version of bank loan in real life. It has several attributes:

Double loanAmount; String startDate; Integer loanId; Customer customer

@Override toString(), @Override equals(), @Override hashCode() for both Customer and LoanInfo class.

Hint: use Objects.hash() when override hashCode(), and use Objects.equals() when override equals()

Task 2:

Create LoanOperation class, which handles different loan operations.

First, create a HashMap that contains all registered loans. The (key, value) pair will be loanId and LoanInfo.

Second step is to create a method registerLoan(LoanInfo loan), which register a new loan by adding it to the HashMap. Note that we can set loanId as a static integer, and the value will be automatically incremented each time when we register a new loan.

The third step is to create a validateLoan(LoanInfo loan) method. In this method, we will practice validating input. You can check the following five conditions:

1. Check if loanInfo is not null
2. Check if loan customer is not null
3. Check if loanAmount > 0
4. Check if customer age is valid. (here we assume that a loan customer needs to be between 15-60)
5. Check if loan startDate is valid (here we assume that loan startDate needs to be later than today)

If the five conditions are fulfilled, then the loan is valid and can be registered. Otherwise you will need to throw IllegalArgumentException. Remember the error message need to reflect the error.

You will need to create two methods: boolean dateIsValid(String startDate) and boolean ageIsValid(Integer age). Hint: use SimpleDateFormat and compareTo to find out if the startDate is later than today.

Task 3:
Still in LoanOperation class, create below methods:

LoanInfo getLoanByLoanId(Integer loanId) – this will return a single loan
ArrayList<LoanInfo> getLoanByName(String name) – this might return multiple loans
ArrayList<LoanInfo> getLoanByStartDate(String startDate) – this might return multiple loans

Hint: if no loan that fulfills the criteria is found, return null object or empty arrayList.
Try to use forEach, or enhanced for loop.

Task 4:

Let's write below methods using Optional
Optional<LoanInfo> getLoanByLoanIdOptional(Integer loanId) – this will return a single loan
Optional<ArrayList<LoanInfo>> getLoanByNameOptional(String name) – this might return multiple loans
Optional<ArrayList<LoanInfo>> getLoanByStartDateOptional(String startDate) – this might return multiple loans

Hint: use Optional.of(), Optional.empty(), or Optional.ofNullable()

Task 5:
Let's write below methods using Lambda

ArrayList<LoanInfo> getLoanByNameLambda(String name)

ArrayList<LoanInfo> getLoanByStartDateLambda(String startDate)

Hint:

You can use

Stream<LoanInfo> filtered_data = loanHashMap.values().stream().filter(loan->{/*your lambda function body*/})

Filtered_data.forEach(loan->{/*your lambda function body*/})

Task 6:

Let's practice method reference. Create two functionalInterface

```
@FunctionalInterface
interface ValidateAge {
    public boolean validate(Integer age);
}

@FunctionalInterface
interface ValidateDate {
    public boolean validate(String startDate);
}
```

Rewrite validateDate and validateAge using lambda expressions.

Task 7:

Create a main() class

1. Call registerLoans. Try to register several valid loans and several invalid loans (age is not valid, loan amount is not valid, or startDate is not valid). Use try/catch to catch the exceptions
2. Call getLoanbyLoanId and getLoanbyLoanIdOptional. Try to get non-existing loan and existing loan. Observe how Optional works.
3. Call getLoansByName and getLoansByNameOptional. Try non-existing customer name and existing customer name.
4. Call getLoansByStartDate and getLoansBySTartDateOptional. Try valid startDate (later than today) and invalid startDate.
5. Use lambda functions you have defined and see if they work
6. Use method reference ValidateAge and ValidateDate to see if they work.

Have fun!