Practices for Lesson 8: Collections Streams, and Filters

Chapter 8

Practices for Lesson 8: Overview

Practice Overview

In these practices, you use lambda expressions to improve an application.

The RoboCall App

The RoboCall app is an application for automating the communication with groups of people. The app can contact individuals by phone, email, or regular mail. In this example, the app will be used to contact three groups of people.

- Drivers: Persons over the age of 16
- Draftees: Male persons between the ages of 18 and 25
- Pilots (specifically commercial pilots): Persons between the ages of 23 and 65

Person

The `Person` class creates the master list of persons you want to contact. The class uses the builder pattern to create new object. The following are some key parts of the class.

First, private fields for each Person are as follows:

Person.java

```
 9 public class Person {
10   private String givenName;
11   private String surName;
12   private int age;
13   private Gender gender;
14   private String eMail;
15   private String phone;
16   private String address;
17   private String city;
18   private String state;
19   private String code;
20
```

So these will be the fields that our application can search.

A static method is used to create a list of sample users. The code looks something like this:

Person.java

```
167   public static List<Person> createShortList(){
168     List<Person> people = new ArrayList<>();
169
170     people.add(
171       new Person.Builder()
172             .givenName("Bob")
173             .surName("Baker")
174             .age(21)
175             .gender(Gender.MALE)
176             .email("bob.baker@example.com")
177             .phoneNumber("201-121-4678")
178             .address("44 4th St")
179             .city("Smallville")
180             .state("KS")
```

```
181                 .code("12333")
182                 .build()
183         );
184
```

forEach

All collections have a new `forEach` method.

RoboCallTest06.java

```
 9 public class RoboCallTest06 {
10
11   public static void main(String[] args){
12
13     List<Person> pl = Person.createShortList();
14
15     System.out.println("\n=== Print List ===");
16     pl.forEach(p -> System.out.println(p));
17
18   }
19 }
```

Notice that the `forEach` takes a method reference or a lambda expression as a parameter. In the example, the `toString` method is called to print out each `Person` object. Some form of expression is needed to specify the output.

Stream and Filter

The following example shows how `stream()` and `filter()` methods are used with a collection in the RoboCall app.

RoboCallTest07.java

```
10 public class RoboCallTest07 {
11
12   public static void main(String[] args){
13
14     List<Person> pl = Person.createShortList();
15     RoboCall05 robo = new RoboCall05();
16
17     System.out.println("\n=== Calling all Drivers Lambda ===");
18     pl.stream()
19         .filter(p -> p.getAge() >= 23 && p.getAge() <= 65)
20         .forEach(p -> robo.roboCall(p));
21
22   }
23 }
```

The `stream` method creates a pipeline of immutable `Person` elements and access to methods that can perform actions on those elements. The `filter` method takes a lambda expression as a parameter and filters on the logical expression provide. This indicates that a `Predicate` is

the target type of the filter. The elements that meet the filter criteria are passed to the `forEach` method, which does a `roboCall` on matching elements.

The following example is functionally equivalent to the last. But in the case, the lambda expression is assigned to a variable, which is then passed to the stream and filter.

RoboCallTest08.java

```
10 public class RoboCallTest08 {
11
12   public static void main(String[] args){
13
14     List<Person> pl = Person.createShortList();
15     RoboCall05 robo = new RoboCall05();
16
17     // Predicates
18     Predicate<Person> allPilots =
19         p -> p.getAge() >= 23 && p.getAge() <= 65;
20
21     System.out.println("\n=== Calling all Drivers Variable ===");
22     pl.stream().filter(allPilots)
23         .forEach(p -> robo.roboCall(p));
24   }
25 }
```

Method References

In cases where a lambda expression just calls an instance method, a method reference can be used instead.

A03aMethodReference.java

```
 9 public class A03aMethodReference {
10
11     public static void main(String[] args) {
12
13         List<SalesTxn> tList = SalesTxn.createTxnList();
14
15         System.out.println("\n== CA Transations Lambda ==");
16         tList.stream()
17             .filter(t -> t.getState().equals(State.CA))
18             .forEach(t -> t.printSummary());
19
20         tList.stream()
21             .filter(t -> t.getState().equals(State.CA))
22             .forEach(SalesTxn::printSummary);
23     }
24 }
```

So lines 18 and 22 are essentially equivalent. Method reference syntax uses the class name followed by "::" and then the method name.

Chaining and Pipelines

The final example compares a compound lambda statement with a chained version using multiple `filter` methods.

A04IterationTest.java

```
 9 public class A04IterationTest {
10
11     public static void main(String[] args) {
12
13         List<SalesTxn> tList = SalesTxn.createTxnList();
14
15         System.out.println("\n== CA Transations for ACME ==");
16         tList.stream()
17             .filter(t -> t.getState().equals(State.CA) &&
18                 t.getBuyer().getName().equals("Acme Electronics"))
19             .forEach(SalesTxn::printSummary);
20
21         tList.stream()
22             .filter(t -> t.getState().equals(State.CA))
23             .filter(t -> t.getBuyerName()
24                 .equals("Acme Electronics"))
25             .forEach(SalesTxn::printSummary);
26     }
27 }
```

The two examples are essentially equivalent. The second example demonstrates how methods can be chained to possibly make the code a little easier to read. Both are examples of pipelines created by the `stream` method.

Practice 8-1: Update RoboCall to use Streams

Overview

In this practice, you have been given an old email mailing list program named `RoboMail`. It is used to send emails or text messages to employees at your company. Refactor `RoboMail` so that it uses lambda expressions instead of anonymous inner classes.

Assumptions

You have completed the lecture and reviewed the overview for this practice.

Tasks

1. Open the `EmployeeSearch08-01Prac` project.
   - Select File > Open Project.
   - Browse to `/home/oracle/labs/08-CollectionsStreamsFilters /practices/practice1`.
   - Select `EmployeeSearch08-01Prac` and click Open Project.
2. Open the `RoboMail01.java` file and remove the `mail` and `text` methods. They are no longer needed since a `stream` will be used to filter the employees and a `forEach` will call the required communication task.
3. Open the `RoboMailTest01.java` file and review the code there.
4. Update `RoboMailTest01.java` to use `stream`, `filter`, and `forEach` to perform the mailing and texting tasks of the previous program.
5. Your program should continue to perform the following tasks to the following groups.
   - Email all sales executives using `stream`, `filter`, and `forEach`.
   - Text all sales executives using `stream`, `filter`, and `forEach`.
   - Email all sales employees older than 50 using `stream`, `filter`, and `forEach`.
   - Text all sales employees older than 50 using `stream`, `filter`, and `forEach`.
6. To mail or text a group in the `forEach` method, use a lambda expression for each task.
   a. Mail example: `p -> robo.roboMail(p)`
   b. Text example: `p -> robo.roboText(p)`

Your output should look similar to the following:

```
==== RoboMail 01

=== Sales Execs
Emailing: Betty Jones age 65 at betty.jones@example.com
Texting: Betty Jones age 65 at 211-33-1234

=== All Sales
Emailing: John Adams age 52 at john.adams@example.com
Emailing: Betty Jones age 65 at betty.jones@example.com
Texting: John Adams age 52 at 112-111-1111
Texting: Betty Jones age 65 at 211-33-1234
```

Practice 8-2: Mail Sales Executives using Method Chaining

Overview

In this practice, continue to work with the RoboMail app from the previous lesson.

Assumptions

You have completed the lecture and completed the previous practice.

Tasks

1. Open the `EmployeeSearch08-02Prac` project.
   - Select File > Open Project.
   - Browse to `/home/oracle/labs/08-CollectionsStreamsFilters /practices/practice2`.
   - Select `EmployeeSearch08-02Prac` and click Open Project.
2. Open the `RoboMailTest01.java` file and review the code there.
3. Update the `RoboMailTest01.java` file to mail all sales executives. Use two filter methods to select the recipients of the mail.

The output from the program should look similar to the following:

```
==== RoboMail 01

=== Sales Execs
Emailing: Betty Jones age 65 at betty.jones@example.com
```

Practice 8-3: Mail Sales Employees over 50 Using Method Chaining

Overview

In this practice, continue to work with the RoboMail app from the previous lesson.

Assumptions

You have completed the lecture and completed the previous practice.

Tasks

1. Open the `EmployeeSearch08-03Prac` project.
   - Select File > Open Project.
   - Browse to `/home/oracle/labs/08-CollectionsStreamsFilters /practices/practice3`.
   - Select `EmployeeSearch08-03Prac` and click Open Project.
2. Open the `RoboMailTest01.java` file and review the code there.
3. Update the `RoboMailTest01.java` file to mail all sales employees over 50. Use two filter methods to select the recipients of the mail.

The output from the program should look similar to the following:

```
==== RoboMail 01

=== All Sales 50+
Emailing: John Adams age 52 at john.adams@example.com
Emailing: Betty Jones age 65 at betty.jones@example.com
```

Practice 8-4: Mail Male Engineering Employees Under 65 Using Method Chaining

Overview

In this practice, continue to work with the RoboMail app from the previous lesson.

Assumptions

You have completed the lecture and completed the previous practice.

Tasks

1. Open the `EmployeeSearch08-04Prac` project.
   - Select File > Open Project.
   - Browse to `/home/oracle/labs/08-CollectionsStreamsFilters /practices/practice4`.
   - Select `EmployeeSearch08-04Prac` and click Open Project.
2. Open the `RoboMailTest01.java` file and review the code there.
3. Update the `RoboMailTest01.java` file to mail all male engineering employees under 65. Use three filter methods to select the recipients of the mail.

The output from the program should look similar to the following:

```
==== RoboMail 01

=== Male Eng Under 65
Emailing: James Johnson age 45 at james.johnson@example.com
Emailing: Joe Bailey age 62 at joebob.bailey@example.com
```