

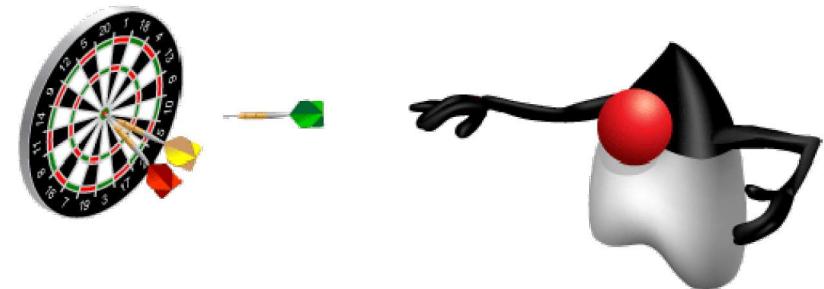
Collections, Streams, and Filters



Objectives

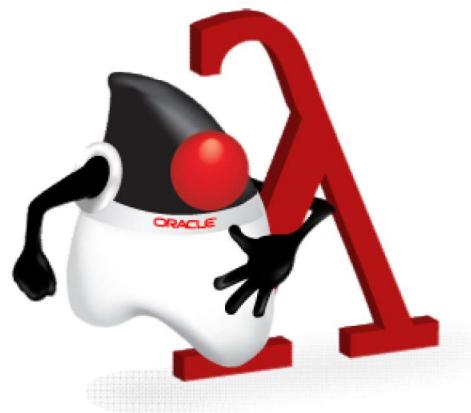
After completing this lesson, you should be able to:

- Describe the Builder pattern
- Iterate through a collection by using lambda syntax
- Describe the Stream interface
- Filter a collection by using lambda expressions
- Call an existing method by using a method reference
- Chain multiple methods
- Define pipelines in terms of lambdas and collections



Collections, Streams, and Filters

- Iterate through collections using forEach
- Streams and Filters



The Person Class

- Person class
 - Attributes like name, age, address, etc.
- Class created by using the Builder pattern
 - Generates a collection persons for examples
- RoboCall Example
 - An app for contacting people via mail, phone, email
 - Given a list of people query for certain groups
 - Used for test and demo
- Groups queried for
 - Drivers: Persons over the age of 16
 - Draftees: Male persons between 18 and 25 years old
 - Pilots: Persons between 23 and 65 years old

Person Properties

- A Person has the following properties:

```
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;
```

Builder Pattern

- Allows object creation by using method chaining
 - Easier-to-read code
 - More flexible object creation
 - Object returns itself
 - A fluent approach
- Example

```
260     people.add(  
261         new Person.Builder()  
262             .givenName("Betty")  
263             .surName("Jones")  
264             .age(85)  
265             .gender(Gender.FEMALE)  
266             .email("betty.jones@example.com")  
267             .phoneNumber("211-33-1234")  
272             .build()  
273     );
```

Collection Iteration and Lambdas

- RoboCall06 **Iterating with** `forEach`

```
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 }
```

RoboCallTest07: Stream and Filter

```
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 }
```

RoboCallTest08: Stream and Filter Again

```
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     }
```

SalesTxn Class

- Class used in examples and practices to follow
- Stores information about sales transactions
 - Seller and buyer
 - Product quantity and price
- Implemented with a Builder class
- Buyer class
 - Simple class to represent buyers and their volume discount level
- Helper enums
 - BuyerClass: Defines volume discount levels
 - State: Lists the states where transactions take place
 - TaxRate: Lists the sales tax rates for different states

- Streams
 - `java.util.stream`
 - A sequence of elements on which various methods can be chained
- Method chaining
 - Multiple methods can be called in one statement
- Stream characteristics
 - They are immutable.
 - After the elements are consumed, they are no longer available from the stream.
 - A chain of operations can occur only once on a particular stream (a pipeline).
 - They can be serial (default) or parallel.

The Filter Method

- The Stream class converts collection to a pipeline
 - Immutable data
 - Can only be used once and then tossed
- Filter method uses Predicate lambdas to select items.
- Syntax:

```
15      System.out.println("\n== CA Transactions Lambda ==");  
16      tList.stream()  
17          .filter(t -> t.getState().equals("CA"))  
18          .forEach(SalesTxn::printSummary);
```

In some cases, the lambda expression merely calls a class method.

- `.forEach(t -> t.printSummary())`
- Alternatively, you can use a method reference
 - `.forEach(SalesTxn::printSummary) ;`
- You can use a method reference in the following situations:
 - Reference to a static method
 - `ContainingClass::staticMethodName`
 - Reference to an instance method
 - Reference to an instance method of an arbitrary object of a particular type (for example,
`String::compareToIgnoreCase`)
 - Reference to a constructor
 - `ClassName::new`

Method Chaining

- Pipelines allow method chaining (like a builder).
- Methods include filter and many others.
- For example:

```
21      tList.stream()
22          .filter(t -> t.getState().equals("CA"))
23          .filter(t -> t.getBuyer().getName()
24              .equals("Acme Electronics"))
25          .forEach(SalesTxn::printSummary);
```

Method Chaining

- You can use compound logical statements.
- You select what is best for the situation.

```
15     System.out.println("\n== CA Transactions for ACME ==");  
16     tList.stream()  
17         .filter(t -> t.getState().equals("CA") &&  
18             t.getBuyer().getName().equals("Acme Electronics"))  
19         .forEach(SalesTxn::printSummary);  
20  
21     tList.stream()  
22         .filter(t -> t.getState().equals("CA"))  
23         .filter(t -> t.getBuyer().getName()  
24             .equals("Acme Electronics"))  
25         .forEach(SalesTxn::printSummary);
```

Pipeline Defined

- A stream pipeline consists of:
 - A source
 - Zero or more intermediate operations
 - One terminal operation
- Examples
 - Source: A Collection (could be a file, a stream, and so on)
 - Intermediate: Filter, Map
 - Terminal: forEach

After completing this lesson, you should be able to:

- Describe the Builder pattern
- Iterate through a collection by using lambda syntax
- Describe the Stream interface
- Filter a collection by using lambda expressions
- Call an existing method by using a method reference
- Chain multiple methods together
- Define pipelines in terms of lambdas and collections

Practice Overview

- Practice 3-1: Update RoboCall to use Streams
- Practice 3-2: Mail Sales Executives using Method Chaining
- Practice 3-3: Mail Sales Employees over 50 using Method Chaining
- Practice 3-4: Mail Male Engineering Employees Under 65 Using Method Chaining