Functional Programming
(Using lambdas in design patterns)

B.Tech. (IT), Sem-6,
Applied Design Patterns and Application Frameworks (ADPAF)

Dharmsinh Desai University
Prof. H B Prajapati

1

Topics

- Lambda Expression
- Default methods
- Using lambdas in design patterns

2

Using lambdas in design patterns

3

Using lambda with design patterns

- How use of lambda changes the code for using design pattern?
- We study the following patterns
  - Iterator
  - Strategy

4

Example: Iterating without using lambda (Use of iterator design pattern)

```
package lambdaindp.iterator;

import java.util.Arrays;
import java.util.List;
public class IteratorTest {
   public static void main(String[] args) {
      List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
      System.out.println("Access numbers using iterator, i.e., (external iterator)");
      System.out.print("Numbers are ");
      for (int i = 0; i < numbers.size(); i++) {
         System.out.print(numbers.get(i)+" ");
      }
      System.out.println("");
   }
}
```

5

Running the Example: Iterating without using lambda (Use of iterator design pattern)

Notifications | Output - lambdaindp (run) | Search Results

```
run:
Access numbers using iterator, i.e., (external iterator)
Numbers are 1 3 4 6 7 9 10 12 13
```

6

Example: Iterating without using lambda (Use of iterator design pattern)

```
package lambdaindp.iterator;
import java.util.Arrays;
import java.util.List;
public class IteratorTest {
  public static void main(String[] args) {
    List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
    System.out.println("Access numbers using for each loop, \n\tbut still iterator, i.e., (external iterator)");
    System.out.print("Numbers are ");
    for(int n:numbers){
      System.out.print(n+" ");
    }
    System.out.println("");
  }
}
```

7

Running the Example: Iterating without using lambda (Use of iterator design pattern)

Notifications | Output - lambdaindp (run) | Search Results

```
run:
Access numbers using for each loop,
        but still iterator, i.e., (external iterator)
Numbers are 1 3 4 6 7 9 10 12 13
```

8

Example: Iterating with using lambda (Use of iterator design pattern)

```
package lambdaindp.iterator;
import java.util.Arrays;
import java.util.List;

public class LambdaInIterator {
  public static void main(String[] args) {
    List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
    System.out.println("Access numbers using lambda in iterator, i.e., (internal iterator)");
    System.out.print("Numbers are ");
    numbers.forEach(no -> System.out.print(no+" "));
    System.out.println("");
  }
}
```

9

Running the Example: Iterating with using lambda (Use of iterator design pattern)

Notifications | Output - lambdaindp (run) | Search Results

```
run:
Access numbers using lambda in iterator, i.e., (internal iterator)
Numbers are 1 3 4 6 7 9 10 12 13
```

10

External iterator v/s lambda with iterator

- In using external iterator, we focus on how to do iteration
  - Imperative style
- Using lambda, we focus on what to do with each element, rather than how to do.
  - Declarative style
- Understanding of forEach
  - numbers.forEach(no -> System.out.print(no+" "));
  - forEach method is internally calling method whose definition we have passed as an argument to forEach
  - The no (in blue color) is the name of variable of that method's argument
  - System.out.print(no+" ") (in red color) is the method body of that method.

11

Example: Strategy design pattern

- Suppose we have requirement of adding numbers in some project
- We create a function to do this.

```
package lambdaindp.strategy;
import java.util.Arrays;
import java.util.List;
public class StrategyTest {
  public static void main(String[] args) {
    List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
    System.out.print("Total of the numbers is "+totalNumbers(numbers));
    System.out.println("");
  }
```

12

Example: Strategy design pattern

```
  public static int totalNumbers(List<Integer> numbers){
    int result=0;
    for(int no:numbers){
      result +=no;
    }
    return result;
  }
}
```

13

Example: Strategy design pattern

Notifications | Output - lambdaindp (run) | Search R

```
run:
Total of the numbers is 65
```

14

Example: Adding even numbers

```
package lambdaindp.strategy;

import java.util.Arrays;
import java.util.List;
public class StrategyTestEven {
  public static void main(String[] args) {
    List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
    System.out.print("Total of the numbers is "+totalNumbers(numbers));
    System.out.println("");
    System.out.print("Total of the Even numbers is
  "+totalEvenNumbers(numbers));
    System.out.println("");

  }
```

16

Example: Adding even numbers

```
public static int totalNumbers(List<Integer> numbers){
    int result=0;
    for(int no:numbers){
      result +=no;
    }
    return result;
  }
  public static int totalEvenNumbers(List<Integer> numbers){
    int result=0;
    for(int no:numbers){
      if(no % 2==0)
        result +=no;
    }
    return result;
  }
}
```

17

Example: Strategy design pattern

- Suppose we also have requirement of adding even numbers in the project
  - Good part is that we have already written code for adding all numbers (totalNumbers()).
  - We have to just modify it to create another method for adding all even numbers.

15

Running the Example: Adding even numbers

Notifications | Output - lambdaindp (run) | Se

```
run:
Total of the numbers is 65
Total of the Even numbers is 32
```

18

Example: Strategy design pattern

- Suppose we also have requirement of adding odd numbers in the project
  - Again good part is that we have already written code for adding even numbers (totalEvenNumbers()).
  - We have to just modify it to create another method for adding all odd numbers.

19

Example: Adding odd numbers

```
package lambdaindp.strategy;
import java.util.Arrays;
import java.util.List;
public class StrategyTestOdd {
  public static void main(String[] args) {
    List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
    System.out.print("Total of the numbers is "+totalNumbers(numbers));
    System.out.println("");
    System.out.print("Total of the Even numbers is
  "+totalEvenNumbers(numbers));
    System.out.println("");
    System.out.print("Total of the Odd numbers is
  "+totalOddNumbers(numbers));
    System.out.println("");
  }
```

20

Example: Adding odd numbers

```
public static int totalNumbers(List<Integer> numbers){
    int result=0;
    for(int no:numbers){
      result +=no;
    }
    return result;
  }
```

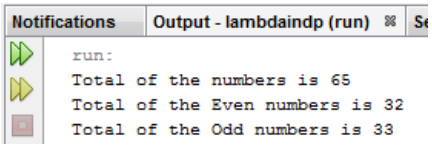
21

Example: Adding odd numbers

```
public static int totalEvenNumbers(List<Integer> numbers){
    int result=0;
    for(int no:numbers){
      if(no % 2==0)
        result +=no;
    }
    return result;
  }
  public static int totalOddNumbers(List<Integer> numbers){
    int result=0;
    for(int no:numbers){
      if(no % 2 != 0)
        result +=no;
    }
    return result;
  }
}
```

22

Running the Example: Adding odd numbers



23

Problems with earlier code and Solution using lambda

- We duplicated code three times.
- If each function's logic is complicated, we need to figure out what are common parts and what are special parts
- We create higher order function that takes another function as parameter
  - Java has Predicate (Generic) functional interface that has method that takes an object and returns true or false.

24

Example: Strategy using lambda

```
package lambdaindp.strategy;
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
public class LambdaInStrategy {
   public static int totalNumbers(List<Integer> numbers, Predicate<Integer>
    selector){
      int result=0;
      for(int no:numbers){
         if(selector.test(no))
            result +=no;
      }
      return result;
   }
```

25

Example: Strategy using lambda

```
   public static void main(String[] args) {
      List<Integer> numbers=Arrays.asList(1,3,4,6,7,9,10,12,13);
      System.out.print("Total of the numbers is
    "+totalNumbers(numbers, no -> true));
      System.out.println("");
      System.out.print("Total of the Even numbers is
    "+totalNumbers(numbers, no -> no%2==0));
      System.out.println("");
      System.out.print("Total of the Odd numbers is
    "+totalNumbers(numbers, no -> no%2!=0));
      System.out.println("");
   }
}
```

26

Running the Example: Strategy using lambda

Notifications | Output - lambdaindp (run) | Se

```
run:
Total of the numbers is 65
Total of the Even numbers is 32
Total of the Odd numbers is 33
```

27

Difference between Strategy implemented without lambda (Prior Java 8) and with lambda

- Earlier, Strategy pattern was implemented by
  - Creating either hierarchy of classes for various strategies

    OR
  - Create an interface indicating signature of strategy operation, and defining concrete classes for various strategies
- Using Lambda
  - We can specify strategy as lambda (No need to write a separate class)

28

A short introduction to stream processing

- Imperative programming
  - Focuses on how to do the work
  - Requires attention while understanding the code
  - Longer code
- Declarative programming
  - Focuses on what to do
  - Code is declarative form. Can be understood easily
  - Less code
  - Examples:
    - SQL, CSS, etc.
  - Functional programming is a subset of declarative programming
- Java 8 Stream processing enables declarative programming

29

Example: Strategy using stream processing

- Can we minimize still the code of totalNumbers() ?

```
public static int totalNumbers(List<Integer> numbers, Predicate<Integer>
   selector) {
return numbers.stream()
        .filter(selector)
        .reduce(0, Integer::sum);
}
```

OR

```
public static int totalNumbers(List<Integer> numbers, Predicate<Integer>
   selector){
        return numbers.stream()
        .filter(selector)
        .mapToInt(no -> no)
        .sum();
  }
```

30

References

- Design Patterns in the Light of Lambda Expressions by Subramaniam (Video)

31