Tutorial 12 – Java Functional Programming

Description

In this tutorial, you will learn to practice with functional programming in Java with mall grouped series of exercises, including:

(1) Common use of lambda expression

(2) Working with collection using Stream

(3) Using functional interfaces: `Predicate/Consumer/Function`

Instructions

Exercise 1: Common uses of lambda expression

While implementing single method interfaces, we often end up writing an anonymous class, those can be replaced with an equivalent lambda expression. Rewrite these implementations using lambda expressions:

(a) Runnable

```
Runnable r = new Runnable() {
    public void run(){
        System.out.println("In an anonymous class!");
    }
};
```

(b) ActionListener

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("You clicked me!");
    }
});
```

(c) Comparator

```
List<Integer> list = Arrays.asList(1, 9, 7, 10, 8);
Collections.sort(list, new Comparator<Integer>() {
    @Override
    public int compare(Integer i1, Integer i2) {
        return i1.compareTo(i2);
    }
});
```

Exercise 2: Working with collection using Stream

- Intermediate operations: filter, map, distinct, sorted, reversed, limit…
- Terminal operations: forEach, count, collect, sum, reduce…

(a) filter, forEach

What is the output of the following code? (Test with your IDE)

```
ArrayList<Integer> nums = new ArrayList<>();
nums.add(3);
nums.add(5);
nums.add(1);
nums.stream()
        .filter(val -> val > 1)
        .forEach(val -> System.out.println(val));
```

(b) Method references

Rewrite (a) using method references.

(c) Parallel stream

Rewrite (a) using parallel stream and re-run the example to observe the difference.

Exercise 3: Using functional interfaces

- Predicate: single argument function that return a boolean value (test)
- Function: single argument function that return a result of an arbitrary type (apply)
- Consumer: single argument function that return no result - void (accept)

Implement these:

(a) `Predicate<Integer> isOdd` tests whether `Integer x` is odd.

(b) `Function<List<Integer>, List<Integer>> filterOdd` applies on a `List` of `Integer`, filters elements using `isOdd` predicate then returns.

(c) `Consumer<List<Integer>> printOdd` accepts a `List` of `Integer`, filters elements using `isOdd` predicate then prints out on the console.

(d) Make `filterOdd` to be more flexible and reusable (not to be fixed with `isOdd` only), implement method `filterList` which receives a `List` of `Integer` and a `Predicate` as input parameters, filters elements using the `Predicate` then returns.

```
filterList(List<Integer> list, Predicate<Integer> predicate): List<Integer>
```

Build and test with these predicate: `isEven`, `greaterThanTen`, `lowerThanTwenty`… (use `and`, `or`, `negate` for predicate chaining)

Exercise 4: SQL like manipulation

Use the starter source code in `tut12ex4_starter.zip`. Implement all TODOs in `PizzaDemo.java`.

Specifically, you have to:

- Complete the `createPizzas()` method to create a list of at least 5 different pizzas.
- In `main` method, perform these operations:
  a) Query and display description of all tropical pizzas
  b) Select top 2 hamd & cheese pizzas with highest cost
  c) Select all pizzas with total cost > $15 group by type (ham & cheese, pepperoni, tropical)