Lambda 道場 問題編

1. はじめに

Java SE 8 で Project Lambda が導入されます。Project Lambda の Lambda 式や Stream API を使用することで、関数型言語のアイデアを導入することができ、内部イテレータによって処理を行うことが可能になります。

今までの外部イテレータを使用した処理から、内部イテレータでの処理への移行により、Java のプログラミングスタイルが大きく変化しようとしています。

すでに、Java SE 8 の Developer Preview も公開されているので、今すぐにでも Project Lambda を試すことができます。

そこで、JJUG CCC のハンズオンでは Project Lambda の Lambda 式や Stream を書いていただきます。習うより慣れろで、繰り返し書くことにより内部イテレータの書き方も身についていくはずです。

本ハンズオンでは、プログラムの断片を提示していきます。それを Lambda 式や Stream で書き換えていってください。PC はなくても大丈夫。鉛筆で直接書き換えていってください。

もし、余裕があるのであれば、家に帰ってから、ぜひ PC で実行し直してみてください。使用した問題、解答は GitHub の LambdaDojo プロジェクトに置いておきます。

LambdaDojo https://github.com/skrb/LambdaDojo (https://github.com/skrb/LambdaDojo)

2. Lambda 式

Project Lambda で最も重要なのは内部イテレータを実現する Stream API です。しかし、Lambda 式が書けないと、Stream API も冗長な匿名クラスで書かざるをえません。やはり、Lambda 式で記述することによって、Stream API も簡潔になります。

Lambda 式は関数型インタフェースを実装した匿名クラスの簡易的な記述法です。関数型インタフェースは実装すべきメソッドが単一の関数で @FunctionalInterface でインタフェースが修飾されています。

Lambda 式は (引数) -> { 処理 } の形式で表します。引数は関数型インタフェースの実装すべきメソッドの引数を表し、処理はそのメソッドの実体を表しています。

そこで、まず匿名クラスを Lambda 式を書き換えてみてましょう。

2-1. Lambda 式で書き換えてみましょう

```
Comparator<Integer> comparator1 = new Comparator<Integer>() {
    @Override
    public int compare(Integer x, Integer y) {
        return x - y;
    }
};
```

2-2. Lambda 式で書き換えてみましょう

```
Callable<Date> callable1 = new Callable<Date>() {
    @Override
    public Date call() throws Exception {
        return new Date();
    }
};
```

2-3. Lambda 式で書き換えてみましょう

```
        Runnable runnable1 = new Runnable() {
            @Override
            public void run() {
                doSomething();
            }
        };
```

2-4. Lambda 式で書き換えてみましょう

```
    @FunctionalInterface
    public interface Doubler<T extends Number> {
        T doDouble(T x);
    }

    ...

        Doubler doubler = new Doubler() {
            @Override
            public Double doDouble(Double x) {
                return 2.0 * x;
            }
        };
```


2-5. Lambda 式で書き換えてみましょう

次のプログラムは Swing のプログラムです。ボタンをクリックするとカウンタが増加するというだけの単純なプログラムですが、ここで使われている匿名クラスを Lambda 式で書き換えてみましょう。

```
public class LambdaForSwing {
    private int count;

    public LambdaForSwing() {
        JFrame frame = new JFrame("Swing Lambda");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton button = new JButton("Count");
        frame.add(button, BorderLayout.NORTH);

        final JLabel counter = new JLabel(String.valueOf(count));
        frame.add(counter, BorderLayout.CENTER);

        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                count++;
                counter.setText(String.valueOf(count));
            }
        });

        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String... args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new SwingLambda();
            }
        });

    }
}
```

3. for 文の変換 (Iterable)

Lambda 文を書くことに慣れたら、外部イテレータを内部イテレータで書き直してみましょう。

最も単純な方法は、Iterable インタフェースに追加された forEach メソッドです。forEach メソッドを使用することで、拡張 for 文を内部イテレータに変換することが可能です。

3-1. forEach メソッドで書き換えてみましょう

```
List<String> strings = Arrays.asList("a", "b", "c", "d", "e");

StringBuilder builder = new StringBuilder();
for (String s: strings) {
    builder.append(s);
}
System.out.println(builder.toString());
```

3-2. forEach メソッドで書き換えてみましょう

```
List<Integer> numbers = Arrays.asList(10, 5, 2, 20, 12, 15);

int sum = 0;
for (Integer number: numbers) {
    sum += number;
}
System.out.println(sum);
```


3-3. forEach メソッドで書き換えてみましょう

拡張 for 文でない、単純な for 文も forEach メソッドで書き換えることができます。ただし、この場合ループインデックスを作成しなければなりません。この場合、Iterable の forEach メソッドではなく、Stream の forEach メソッドが使われます。

ここでは単純なループインデックスを使用して、書き換えを行ってみましょう。

```
for (int i = 0; i < 10; i++) {
    System.out.print(i);
}
System.out.println();
```

4. for 文の変換 (Stream)

最後は、Stream です。Stream にはオブジェクトを対象とした Stream インタフェース、プリミティブが対象の IntStream インタフェース、LongStream インタフェース、DoubleStream インタフェースの 4 種類があります。

基本的なメソッドは同じですが、sum メソッド、max メソッドなどプリミティブが対象の Stream にしかないメソッドもあります。

はじめから様々な処理を行っている for 文を変換するのは難しので、簡単な例から徐々に慣れていきましょう。

4-1. Stream で書き換えてみましょう

```
List<Integer> numbers = Arrays.asList(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
for (Integer x: numbers) {
    if (x % 2 == 0) {
        System.out.print(x);
    }
}
System.out.println();
```

4-2. Stream で書き換えてみましょう

```
Random random = new Random();
List numbers = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    numbers.add(random.nextDouble());
}

double ave = 0.0;
for (Double x: numbers) {
    ave += x;
}
ave /= numbers.size();

double div = 0.0;
for (Double x: numbers) {
    div += (x - ave) * (x -ave);
}
div /= numbers.size();
```

4-3. Stream で書き換えてみましょう

次に行うのはファイルを読み込んで単語数を数えるプログラムの断片です。これも Stream で書き換えてみましょう。

ヒント: BufferReader クラスに Java SE 8 で追加されたメソッドがありますよ。

```
try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
    int wordCount = 0;

    for (;;) {
        String line = reader.readLine();
        if (line == null) {
            break;
        }

        String[] words = line.split(" ");
        wordCount += words.length;
    }
    System.out.println(wordCount);
} catch (IOException ex) {
    // 例外処理
}
```


いかがでしたでしょうか。

ちなみに、個々の問題は正解が 1 つとは限りません。いろいろな書き方があり、それに応じて使用するメソッドも変化します。いろいろな書き方をできるようにしておくと、応用範囲がひろがると思います。