応用プログラミングIII [Q2] 中間課題

c0115114 菅野 路哉

2016 年 06 月 19 日

1 目的

最新の Singleton 実装を知り、古い Singleton よりも優れている点を調査する。

2 課題設定

今回のレポートでは、課題 2.2 を選択する。講義内で実装した Singleton は、クラス変数にインスタンスを持たせ、コンストラクタを private にしてアクセス制限をかけることで、インスタンスの唯一性を保った。しかし、これは 1 世代前の実装方法であるため、最新の Singleton 実装を調査して実装する。また、最新の Singleton 実装では、何が以前の Singleton 実装よりも優れているのかを考察する。

3 ソースコード

3.1 Singleton

リスト 1　Singleton.java

```
1 enum Singleton{
2   INSTANCE;
3   Singleton(){
4     System.out.println("インスタンスが生成されました");
5   }
6 }
```

3.2 Main

リスト 2　Main.java

```
1  public class Main extends Thread {
2    public static void main(String[] args) {
3      System.out.println("START!");
4
5      Thread t1 = new Main("A");
6      Thread t2 = new Main("B");
7      t1.start();
8      t2.start();
9
10     System.out.println("END!");
11   }
12
13   @Override
14   public void run() {
15     Singleton obj = Singleton.INSTANCE;
16     System.out.println(this.getName() + ":obj=" + obj.hashCode());
17   }
18
19   public Main(String name) {
20     super(name);
21   }
22 }
```

4 実行結果

リスト 3　Main.java の実行結果

```
START!
インスタンスが生成されました
A:obj=1498557318
END!
B:obj=1498557318
```

5 解説

リスト 1 では、最新の Singleton 実装が定義されている。enum を用いて、クラスの定義を行う。シングルトンの実体であるインスタンスは、Singleton クラスの INSTANCE フィールドとなっている。enum で宣言したクラスのインスタンスは、enum 内で列挙されたもの以外作成出来ないことが保証されている。また、列挙されたインスタンスは、それぞれ定数であるために書き換えも出来ない。そのため、シングルトンである事が分かる。

リスト 2 では、実装したシングルトンを使用する。また、使用するコードは、課題 2.1 に使用されている OldMain.java とほぼ同じであり、使用するシングルトンクラスの変更、ObjectID の表示部分の変更を行った。使用するシングルトンクラスは、今回作成した Singleton である。ObjectID 部分の変更は、enum によって列挙されたインスタンスを表示する際に、インスタンスの変数名が表示されるためである。そのため、インスタンスのハッシュ値を表示して同一の Object であるかを確認するために、hashCode メソッドを用いた。

6 考察・まとめ

リスト 3 では、「インスタンスを生成しました」のメッセージが 1 度出ている。ここで、「START」の表示よりも「インスタンスを生成しました」のメッセージが後ろに表示されていることから、クラスを読み込んだタイミングでインスタンスが生成されていない事が分かる。enum で列挙されたインスタンスは、列挙されたインスタンスが参照された際に、同クラス内の列挙されたインスタンス全てが初期化される。そのため、任意のタイミングでインスタンスを生成することができる。

また、表示されたハッシュ値がそれぞれ一致している。そのため、それぞれのスレッドで参照している Singleton インスタンスは同一であることが分かる。

enum を用いてシングルトン実装を行うと、書くべき事が簡潔になる。これが、他のシングルトン実装と比べて最も大きなメリットであると考える。class を用いたシングルトン実装では、どのようなシングルトン実装でも行える。しかし、より厳密なシングルトン実装を行おうとすると処理すべき事項が多くなる。また、意図していないシングルトン実装が行われていても、エラーが出ないためそれに気づくことが難しくなる。enum を用いたシングルトン実装のように、書くべき事項が簡潔であれば、そういった意図していないバグを防ぐことが出来る。

また、enum を用いたクラスは、JVM によってシングルトンであることが保証されているため、実装したものがシングルトン実装であることを明記できる。

しかし、enum によるシングルトン実装にはデメリットも存在する。処理によって生成するインスタンスを変更できない点である。生成するインスタンスは、予め列挙しておく必要があるため、生成されるインスタンスはソースコード記述の際に定まってしまう。そのため処理途中にインスタンスを動的に決定したい場合は、この方法を用いることは出来ない。