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

古い Singleton 実装の問題点を知り、問題点を補った Singleton 実装を行えるようにする。また、何故そのような問題が起きるのかを考察する。

2 課題設定

今回のレポートでは、課題 2.1 を選択する。古い Singleton 実装である OldSingleton.java では、Singleton のインスタンスが複数生成されてしまう場合が存在する。そのため、何故そのような事が起きるのか、それを防ぐにはどうしたら良いのかを考察する。考察には、OldMain.java の実行結果と、OldSingleton.java に修正を加えた NewSingleton.java を用いた NewMain.java の実行結果を用いる。

また、今回使用する OldSingleton.java と、OldMain.java は既に用意されているものを用いる。

3 ソースコード

3.1 OldSingleton

リスト 1 OldSingleton.java

```
1  package kg07;
2
3  public class OldSingleton {
4    private static OldSingleton singleton = null;
5
6    private OldSingleton() {
7      System.out.println("インスタンスを生成しました");
8      waitfor();
9    }
10
11   public static OldSingleton getInstance() {
12     if (singleton == null) {
13       singleton = new OldSingleton();
14     }
15     return singleton;
16   }
17
18   private void waitfor() {
19     try {
20       Thread.sleep(1000);
21     }catch (InterruptedException e) {
22       e.printStackTrace();
23     }
24   }
25 }
```

3.2 OldMain

リスト 2　OldMain.java

```
1  package kg07;
2
3  public class OldMain extends Thread {
4    public static void main(String[] args) {
5      System.out.println("START!");
6
7      Thread t1 = new OldMain("A");
8      Thread t2 = new OldMain("B");
9      t1.start();
10     t2.start();
11
12     System.out.println("END!");
13   }
14
15   @Override
16   public void run() {
17     OldSingleton obj = OldSingleton.getInstance();
18     System.out.println(this.getName() + ":obj=" + obj);
19   }
20
21   public OldMain(String name) {
22     super(name);
23   }
24 }
```

3.3 NewSingleton

リスト 3　NewSingleton.java

```
1  package kg07;
2
3  public class NewSingleton {
4    private static NewSingleton singleton = null;
5
6    private NewSingleton() {
7      System.out.println("インスタンスを生成しました");
8      waitfor();
9    }
10
11   public static synchronized NewSingleton getInstance() {
12     if (singleton == null) {
13       singleton = new NewSingleton();
14     }
15     return singleton;
16   }
17
18   private void waitfor() {
19     try {
20       Thread.sleep(1000);
21     }catch (InterruptedException e) {
22       e.printStackTrace();
23     }
24   }
25 }
```

3.4 NewMain

リスト 4　NewMain.java

```
1  package kg07;
2
3  public class NewMain extends Thread{
4    public static void main(String[] args) {
5      System.out.println("START!");
6
7      Thread t1 = new NewMain("A");
8      Thread t2 = new NewMain("B");
9      t1.start();
10     t2.start();
11
12     System.out.println("END!");
13   }
14
15   @Override
16   public void run() {
17     NewSingleton obj = NewSingleton.getInstance();
18     System.out.println(this.getName() + ":obj=" + obj);
19   }
20
21   public NewMain(String name) {
22     super(name);
23   }
24 }
```

4 実行結果

リスト 5　OldMain.java の実行結果

```
START!
インスタンスを生成しました
END!
インスタンスを生成しました
A:obj=kg07.OldSingleton@6678e022
B:obj=kg07.OldSingleton@d25dd6b
```

リスト 6　NewMain.java の実行結果

```
START!
インスタンスを生成しました
END!
A:obj=kg07.NewSingleton@6678e022
B:obj=kg07.NewSingleton@6678e022
```

5 解説

挙動説明を簡易化するために、Singleton のインスタンス生成時にかかる時間を延ばす。そのため、コンストラクタ内で一秒停止している。また、インスタンスが生成された場合、「インスタンスを作成しました」のメッセージが出るようにしている。

リスト 1 では、古い Singleton 実装が定義されている。インスタンスの取得は getInstance メソッドを用いる。getInstance では、自身のフィールドである singleton が null だった場合、新しく OldSingleton インスタンスを生成し、singleton に代入する。また、その singleton を戻り値とする。

リスト 2 では、古い Singleton を用いて、マルチスレッドによる Singleton のインスタンス参照を行う。2 つのサブスレッドを作成して、順次スレッドを実行する。スレッド内では、OldSingleton の getInstance を呼び出し、取得したイ

ンスタンスの ObjectID を確認する。それぞれのスレッドで異なった ObjectID が検出された場合、同一のインスタンスが取得出来ていない事が分かる。

リスト 3 では、改良した Singleton 実装を定義した。インスタンスの取得は、OldSingleton の場合と同様の getInstance メソッドを用いるが、修飾子に synchronized を付与して、並列実行が行われないようにする。他の実装は、OldSingleton と変わらない。

リスト 4 では、OldSingleton の代わりに NewSingleton を用いて、OldMain と同様の処理を行う。

6 考察・まとめ

リスト 5 では、「インスタンスを生成しました」のメッセージが 2 度出ている。また、ObjectID がそれぞれ異なっている。以上の事から、それぞれのスレッドで参照している Singleton インスタンスは別物である事が分かる。これは、それぞれのスレッドから、getInstance メソッドを同時に呼び出している事が原因である。getInstance メソッドでは、既存のインスタンスが作成されていない場合、インスタンスを生成する。この時、インスタンス生成途中で getInstance メソッドが呼び出されると、既存のインスタンスは未だ無いため、追加でインスタンスを生成し始める。そのため、OldSingleton では複数のインスタンスが存在することになる。

リスト 6 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。また、ObjectID がそれぞれ一致している。以上の事から、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。

NewSingleton では、OldSingleton で複数のインスタンスが生成される問題に対して、synchronized 修飾子を getInstance メソッドに付与することで対策している。synchronized 修飾子は、そのブロックに対して処理が始まった際にロックを行い、別の場所からのアクセス要求があった場合、現在行っている処理が完了するまで待機させる。そのため、今回の場合、インスタンス生成時に getInstance メソッドが呼び出されることが無くなり、インスタンスが 1 つに定まる。