Pre at Software Design Pattern, Singleton

Group 5

2019-10-10

Singleton Pattern

Goal of the Singleton Pattern

- To eliminate the option of instantiating more than one object

Design of Boiler

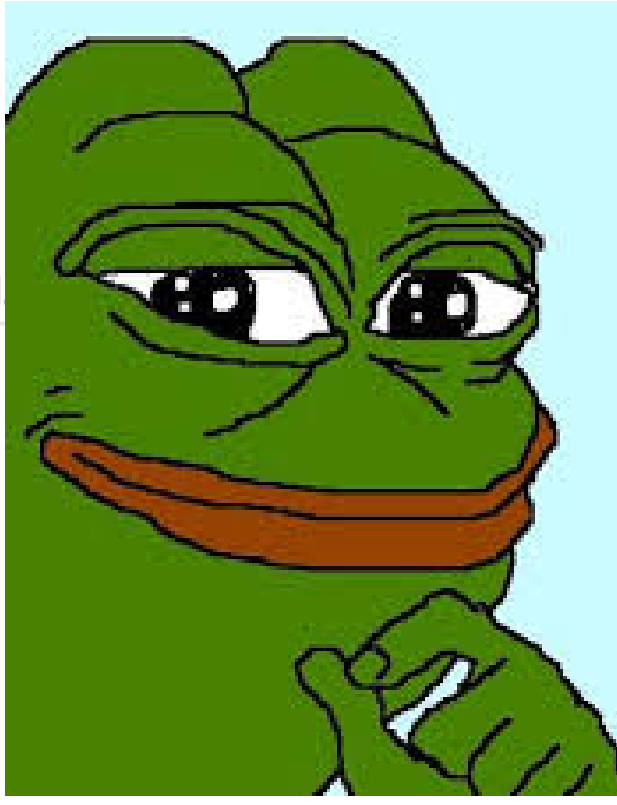
```
class Boiler {
    private static Boiler uniq = new Boiler();
    private Boiler() {}
    public static Boiler getInstance() {
        return uniq;
    }
}
```

Execution

```
public class Main {
    public static void main(String[] args) {
        Boiler blrFirst = Boiler.getInstance();
        Boiler blrSecond = Boiler.getInstance();
        System.out.println("Boiler the First, ID "
            + System.identityHashCode(blrFirst));
        System.out.println("Boiler the Second, ID "
            + System.identityHashCode(blrSecond));
    }
}
```

Demonstration

```
<terminated> Main (3) [Java Application] /Library/Java/J
Boiler the First, ID 1259475182
Boiler the Second, ID 1259475182
```



- That's great, but not enough

K Pattern Eager

```
import java.util.LinkedList;

class Boiler {
    private static int numOfBoilers = 3;
    private static LinkedList<Boiler> blrs
        = new LinkedList<>();
    static { for (int i = 0; i < numOfBoilers; i++)
        { blrs.add(new Boiler()); } }

    private Boiler() { }
    public static Boiler getInstance(int n) {
        return blrs.get(n - 1);
    }
}
```

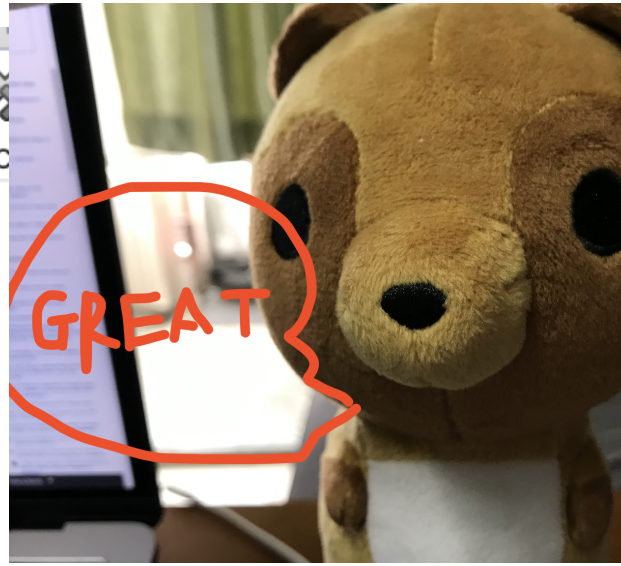
Execution

```
public class Main {
    public static void main(String[] args) {
        Boiler blrFirst = Boiler.getInstance(1);
        Boiler blrSecond = Boiler.getInstance(1);
        Boiler blrThird = Boiler.getInstance(2);
        Boiler blrFourth = Boiler.getInstance(3);
        Boiler blrFifth = Boiler.getInstance(2);

        /* Print the identityHashCode of the five
         * objects */
    }
}
```

Demonstration

```
<terminated> Main (3) [Java Application] /Library/Java/JavaVirtualMac
Boiler the First, ID 1259475182
Boiler the Second, ID 1259475182
Boiler the Third, ID 1300109446
Boiler the Fourth, ID 1020371697
Boiler the Fifth, ID 1300109446
```



- We avoid a concurrenet demo with EAGER K Pattern
- If you are implementing LAZY pattern, you should take **synchronization** into consideration

Cons of Eager Version

- Preoccupation of the space, may cause waste
- Trade-off of the runtime changeablility

But it is more friendly to us, right? Avoid dynamic allocation and release

More Thinkings

- Dynamically distribute objects, with SEMAPHORE or MUTEX
-

```
3
4  /* Please don't access any members of this structure directly */
5  struct semaphore {
6          raw_spinlock_t          lock;
7          unsigned int            count;
8          struct list_head        wait_list;
9  };
0
1  #define __SEMAPHORE_INITIALIZER(name, n)                                \
2  {                                                                       \
3          .lock           = __RAW_SPIN_LOCK_UNLOCKED((name).lock),        \
4          .count          = n,                                            \
5          .wait_list      = LIST_HEAD_INIT((name).wait_list),             \
6  }
7
```

Just like the RES in OS, in a big picture

Thanks