Singleton Pattern

2015004239 정성운

Singleton_Pattern

SeongWoon Jeong edited this page now · 2 revisions

Software Engineering - CSE4006

Singleton Pattern

professor. 정소희

---

1. 개요

- Singleton Pattern에 대해서 이해한다.
  - Singleton Pattern : 인스턴스가 오직 하나만 생성되는 것을 보장하고, 어디에서든 이 인스턴스에 접근할 수 있도록 하는 디자인 패턴

2. 과제 설명

- 다음 TicketMaker 클래스는 getNextTicketNumber메소드를 호출할 때마다 1000, 1001, 1002, ... 라는 수를 순서대로 반환하는 것으로 Ticket의 번호나 일련 번호를 생성할 때 사용한다. 현재 TicketMaker 클래스는 인스턴스를 몇 개라도 만들 수 있다. Singleton Pattern을 적용해서 인스턴스가 하나만 만들어지도록 하시오.

3. 사용언어 / 환경

- Java / Mac OS

[주어진 소스코드]

```java
package exercise01.singleton;

public class TicketMaker {
    private int ticket = 1000;

    public int getNextTicketNumber() {
        return ticket++;
    }
}
```

```java
package exercise01.singleton;

public class UserThread extends Thread{
    public UserThread(String name) {
        super(name);
    }

    public void run() {
        TicketMaker ticketMaker =  TicketMaker 객체 생성
        System.out.println(Thread.currentThread().getName() +
            "is making ticket: " + ticketMaker.getNextTicketNumber());
    }
}
```

```java
package exercise01.singleton;

public class Client {
    public static void main(String[] args) {
        UserThread[] users = new UserThread[10];

        for(int i = 0; i < users.length; i++) {
            users[i] = new UserThread((i + 1) + "-thread");
            users[i].start();
        }
    }
}
```

[주어진 소스코드의 취약점 분석]

- 현재 주어진 TicketMaker 클래스는 생성자가 public이기 때문에, 여러개의 티켓 메이커 객체를 만들어낼 수 있다. 이렇게 되면 여러 티켓메이커 객체에서 같은 넘버를 가진 티켓들이 만들어지게 되는데, 이는 우리가 의도한 바와 다르다.

```
public class TicketMaker {
        private int ticket = 1000;

        // TicketMaker(){} 디폴트 생성자가 생략된 상태.

        public synchronized static int getNextTicketNumber
                return ticket++;
        }
}
```

[취약점 개선 - Singleton Pattern]

- 우선 먼저, TicketMaker()생성자를 private으로 변경한다.
- 이후, private한 생성자를 내부 메서드에서 불러 사용하기 위해 getNextTicketNumber()메서드를 static으로 선언한다.
- 이는 단일 쓰레드 방식에서는 1개의 객체가 보장되지만, 멀티 쓰레드 방식에서는 1개의 객체가 보장되지 않는다
    - 객체를 만드는 도중에 다른 쓰레드가 접근해 새로운 객체를 만들수 있음.

```
public class TicketMaker {
        public static TicketMaker ticketmaker = null;
        private int ticket = 1000;

        // private으로 만든다.
        private TicketMaker() {
        }

        // 티켓메이커 객체가 없으면 객체를 생성후 ticketnumber를 준다
        public static int getNextTicketNumber() {
                if(ticketmaker == null) {
                        ticketmaker = new TicketMaker();
                }
                return ticketmaker.ticket++;
        }
}
```

- 멀티쓰레딩시 싱글톤 패턴을 유지하는 방법

[early loading]

```
public class TicketMaker {
        // static한 객체를 컴파일시 만들어버린다.
        public static TicketMaker ticketmaker = new Ticket
        private int ticket = 1000;

        // private으로 만든다.
        private TicketMaker() {
        }

        // 티켓메이커 객체가 없으면 객체를 생성후 ticketnumber를 준다
        public static int getNextTicketNumber() {
                return ticketmaker.ticket++;
        }
}
```

[lazy loading]

```
public class TicketMaker {
        private static TicketMaker ticketmaker = null;
        private int ticket = 1000;

        // 쓰레드가 접근하는 순서는 JVM이 운영체제에게 위임해서 운영체제
        private TicketMaker() {
        }

        // synchronized를 통해 여러 쓰레드의 접근을 막는다.
        public synchronized static int getNextTicketNumber
                if(ticketmaker == null) {
                        ticketmaker = new TicketMaker();
                }
                return ticketmaker.ticket++;
        }
}
```

[전체 소스코드 - early loading] : Static객체를 선언해 컴파일시 객체를 하나만 만드는 기법

[UserThread.java] :

```
public class UserThread extends Thread {
        public UserThread(String name) {
                super(name);
        }

        public void run() {
                System.out.println(Thread.currentThread().
        }
}
```

[TicketMaker.java] :

```
public class TicketMaker {
        private static TicketMaker ticketmaker = new Ticke
        private int ticket = 1000;

        // 쓰레드가 접근하는 순서는 JVM이 운영체제에게 위임해서 운영체제
        private TicketMaker() {
        }

        public static int getNextTicketNumber() {
                return ticketmaker.ticket++;
        }
}
```

TicketMaker() 생성자는 private선언후, static 객체를 만든다. 여러개 쓰레드에서의 동시접속을 막기 위해 synchronized를 이용한다.

[Client.java] :

```
public class Client {
        public static void main(String[] args) {
                UserThread[] users = new UserThread[10];
                for (int i = 0; i < users.length; i++) {
                        users[i] = new UserThread((i+1) +
                        users[i].start();
                }
        }
}
```

[전체 소스코드 - lazy loading] : synchronized를 이용해 런타임시 여러 쓰레드 접근을 방지한다.

[UserThread.java] :

```
public class UserThread extends Thread {
        public UserThread(String name) {
                super(name);
        }

        public void run() {
                System.out.println(Thread.currentThread().
        }
}
```

[TicketMaker.java] :

```
public class TicketMaker {
        private static TicketMaker ticketmaker = null;
        private int ticket = 1000;

        // 쓰레드가 접근하는 순서는 JVM이 운영체제에게 위임해서 운영체제
        private TicketMaker() {
        }

        public synchronized static int getNextTicketNumber
                if(ticketmaker == null) {
                        ticketmaker = new TicketMaker();
                }
                return ticketmaker.ticket++;
        }
}
```

TicketMaker() 생성자는 private선언후, 만약 ticketmaker객체가 없으면 하나 만들고 있으면 해당 객체의 티켓을 반환한다. synchronized를 이용해 여러 쓰레드의 접근을 막아 ticket객체가 하나만 생성될 수 있도록 한다.

[Client.java] :

```
public class Client {
        public static void main(String[] args) {
                UserThread[] users = new UserThread[10];
                for (int i = 0; i < users.length; i++) {
                        users[i] = new UserThread((i+1) +
                        users[i].start();
                }
        }
}
```

[예상되는 클라이언트 실행 결과와 실제 실행 결과]

- 예상되는 클라이언트 실행 결과

```
5-threadis making ticket: 1001
8-threadis making ticket: 1000
2-threadis making ticket: 1005
3-threadis making ticket: 1004
6-threadis making ticket: 1003
4-threadis making ticket: 1002
9-threadis making ticket: 1006
1-threadis making ticket: 1007
7-threadis making ticket: 1008
10-threadis making ticket: 1009
```

- 실제 실행 결과

```
5-thread is making ticket: 1001
8-thread is making ticket: 1007
3-thread is making ticket: 1004
4-thread is making ticket: 1003
10-thread is making ticket: 1009
2-thread is making ticket: 1005
1-thread is making ticket: 1000
7-thread is making ticket: 1006
6-thread is making ticket: 1002
9-thread is making ticket: 1008
```