JAVA 객체지향 디자인 패턴

6장 싱글턴 패턴

목차

- 프린터 관리자 만들기
- 문제점
- 해결책
- 싱글턴 패턴
- 싱글턴 패턴과 정적 클래스

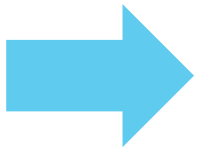
6장 싱글턴 패턴

프린터 관리자 만들기

프린터 관리자 만들기

- 프린터 관리자 만들기
  - 10명의 직원들이 프린터 하나만 공유해서 사용

```
public class Printer{
    public Printer(){}
    public void print(Resource r){
        ....
    }
}
```



```
public class Printer{
    private Printer(){}
    public void print(Resource r){
        ....
    }
}
```

프린터 관리자 만들기

- 프린터 관리자 만들기
  - 외부에 제공해줄 메서드 필요
  - 단 하나의 객체만을 생성하여 어디에서 든지 참조 가능하도록 메서드 선언

```
public class Printer{
    private static Printer printer = null;
    private Printer(){}
    public static Printer getPrinter(){
        if(printer == null){
            printer = new Printer();
        }
        return printer;
    }
    public void print(Resource r){
        ....
    }
}
```

프린터 관리자 만들기

- 프린터 관리자 만들기
  - 5명의 사용자가 프린터를 이용하는 상황

프린터 관리자 만들기

```
public class User{
    private String name;
    public User(String name){
        this.name = name;
    }
    public void print(){
        Printer printer = Printer.getPrinter();
        printer.print(this.name + "print using "
+ printer.toString() + ".");
    }
}
```

```
public class Printer{
    private static Printer printer = null;
    private Printer(){}
    public static Printer getPrinter(){
        if(printer == null){
            printer = new Printer();
        }
        return printer;
    }
    public void print(Resource r){
        ....
    }
}
```

프린터 관리자 만들기

```
public class Main{
    private static final int User_NUM = 5;
    public static void main(String[] args){
        User[] user = new User[User_NUM];
        for(int i=0; i<User_NUM; i++){
            user[i] = new User((i+1) + "-user");
            user[i].print();
        }
    }
}
```

문제점

- 다중 스레드에서 Printer 클래스를 이용할 때 인스턴스가 1개 이상 생성되는 경우가 발생할 수 있다.
  1. Printer 인스턴스가 아직 생성되지 않았을 때 스레드 1이 getPrinter 메서드의 if문을 실행해 이미 인스턴스가 생성되었는지 확인한다. 현재 printer 변수는 null인 상태이다.
  2. 만약 스레드 1이 생성자를 호출해 인스턴스를 만들기 전 스레드 2가 if문을 실행해 printer 변수가 null인지 확인한다. 현재 null이므로 인스턴스를 생성하는 코드, 즉 생성자를 호출하는 코드를 실행하게 된다.
  3. 스레드 1도 스레드 2와 마찬가지로 인스턴스를 생성하는 코드를 실행하게 되면 결과적으로 Printer클래스의 인스턴스가 2개 생성된다.

* 위 시나리오는 경합 조건(race condition)을 발생시킨다. 경합 조건이란 메모리와 같은 동일한 자원을 2개 이상의 스레드가 이용하려고 경합하는 현상을 말한다.

해결책

- 프린터 관리자는 다중 스레드 애플리케이션이 아닌 경우에는 아무런 문제가 되지 않는다. 따라서 다중 스레드 애플리케이션에서 발생하는 문제를 해결하는 방법 2가지를 설명한다.
  1. 정적 변수에 인스턴스를 만들어 바로 초기화하는 방법
  2. 인스턴스를 만드는 메서드에 동기화하는 방법

해결책

- 정적 변수에 인스턴스를 만들어 바로 초기화하는 방법
  - if(printer == null)라는 조건 검사 구문을 원천적으로 제거하기 위한 방법

```
public class Printer{
    private static Printer printer = new Printer();
    private int counter = 0;
    private Printer(){}
    public static Printer getPrinter(){
        printer = new Printer();
    }
    public void print(String str){
        counter++;
        System.out.println(str);
    }
}
```

해결책

- 인스턴스를 만드는 메서드에 동기화하는 방법
  - 동시에 여러 스레드가 **getPrinter**메서드를 소유하는 객체에 접근하는 것을 방지한다.

```
public class Printer{
    private static Printer printer = null;
    private Printer(){}
    public synchronized static Printer getPrinter(){
        if(printer == null){
            printer = new Printer();
        }
        return printer;
    }
    public void print(String str){
        System.out.println(str);
    }
}
```

해결책

- 인스턴스를 만드는 메서드에 동기화하는 방법
  - Counter 변수를 변경하는 부분도 동기화할 필요가 있다.

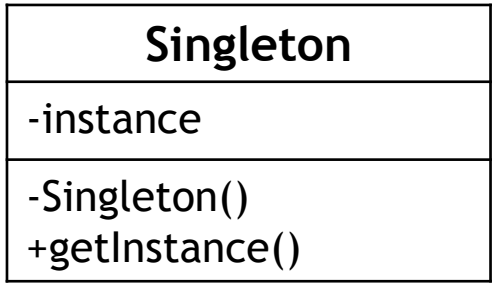
```
public class Printer{
    private static Printer printer = null;
    private int counter = 0;
    private Printer(){}
    public synchronized static Printer getPrinter(){
        ...
    }
    public void print(String str){
        synchronized(this){
            counter++;
            System.out.println(str);
        }
    }
}
```

6장 싱글턴 패턴

싱글턴 패턴

싱글턴 패턴

- 인스턴스가 오직 하나만 생성되는 것을 보장하고 어디에서든 이 인스턴스에 접근할 수 있도록 하는 디자인 패턴



싱글턴 패턴과 정적 클래스

- 굳이 싱글턴 패턴을 사용하지 않고 정적 메서드로만 이루어진 정적 클래스를 사용해도 동일한 효과를 얻을 수 있다.
  (코드 6-11 참조)

정적 클래스

- 장점
  1. 정적 클래스와 싱글턴 패턴을 이용하는 방법 중에 가장 차이가 나는 점은 객체를 전혀 생성하지 않고 메서드를 사용한다는 점이다.
  2. 정적 클래스는 바인딩되는(컴파일 타임에 바인딩되는) 인스턴스 메서드를 사용하는 것보다 성능 면에서 우수하다.
- 단점
  1. 정적 메소드는 인터페이스에서 사용할 수 없다.