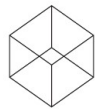
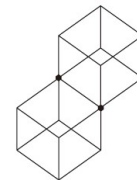


6. 싱글턴패턴



JAVA
개체 지향
디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는



학습목표

학습목표

- 싱글턴 패턴 이해하기
- 다중 스레드와 싱글턴 패턴의 관계 이해하기



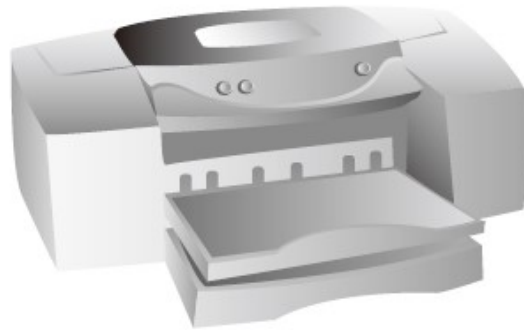
6.1 프린터 관리자 만들기

❖ 문제

- 10명의 직원들이 **프린터 하나만** 공유해서 사용
- 프린터를 관리하는 프로그램을 작성하라



그림 6-1 귀중한 프린터



❖ 코드 6-1 . Printer Class 의 초기 버전

코드 6-1

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

- Printer 클래스를 사용하는 클라이언트프로그램

- New Printer()가 반드시 한번만 호출되도록 주의를 기울여야함
- 해결책... 생성자를 외부에서 호출하지 못하게 함
 - ➔ Printer Class의 생성자를 **Private**로 선언



❖ 코드 6-2. Printer Class의 두번째 버전

코드 6-2

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

- 외부에서는 new Printer()를 사용하지 못함
- 추가보완책. 프린터 인스턴스는 하나 만들어야 하는 상황 필요 --> 코드 6-3



❖ 코드 6-3. Printer Class의 세 번째 버전

- `getPrinter()`

- 프린터 인스턴스가 이미 생성되어 있는지 검사
- 인스턴스가 생성되지 않았으면 생성자를 호출, 생성
- 생성된 인스턴스는 `static` 변수인 `printer`에 의해 참조

- Static method, static variable

- 클래스의 인스턴스 생성 없이도 메소드 수행, 변수 참조
- 어디에서도 참조 가능

코드 6-3

```
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) {  
        ...  
    }  
}
```



코드 6-4 (p. 192) - 5명의 사용자가 프린터를 이용하는 상황

코드 6-4

```
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(); // Printer 인스턴스 생성
        }
        return printer;
    }
}
```

코드 6-4

```
public void print(String str) {
    System.out.println(str);
}

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 인스턴스 생성
            user[i].print();
        }
    }
}
```

Result:

```
1-user print using Printer@2ce07e6b.
2-user print using Printer@2ce07e6b.
3-user print using Printer@2ce07e6b.
4-user print using Printer@2ce07e6b.
5-user print using Printer@2ce07e6b.
```



6.2 문제점 (p. 194)

❖ 다중 스레드에서 인스턴스가 1개 이상 생성되는 문제점

❖ 시나리오

1. Printer 인스턴스가 아직 생성되지 않았을 때 스레드 1이 getPrinter 메서드의 if문을 실행해 이미 인스턴스가 생성되었는지 확인한다. 현재 printer 변수는 null인 상태다.
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 Printer 클래스의 인스턴스가 2개 생성된다.

❖ 경합 조건 -- Race condition

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



코드 6-5. 경합

코드 6-5

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

    public void run() {
        Printer printer = printer.getPrinter();
        printer .print(Thread.currentThread().getName()
+         " print using " + printer.toString() + " *.\n");
    }
}
```

```
2-thread print using Printer@2cfa930d.
4-thread print using Printer@76cc518c.
5-thread print using Printer@5ffdfb42.
3-thread print using Printer@1b7adb4a.
1-thread print using Printer@1ed2e55e.
```

코드 6-5

```
public class UserThread extends Thread {
    private static Printer printer = null;
    private Printer() { }

    public static Printer getPrinter() {
        if ( printer == null ) {
            try {
                Thread.sleep(1);
            }
            catch (InterruptedException e) { }
            printer = new Printer();
        }
        return printer;
    }

    public void print(String str) {
        System.out.println(str);
    }
}

public class Client {

    private static final int THREAD_NUM = 5;

    public static void main(String[] args) {
        UserThread[] user = new UserThread(THREAD_NUM);
        for (int i = 0; i < THREAD_NUM; i++) {
            user[i] = new UserThread((i + 1) + "-thread" );
            user[i].start();
        }
    }
}
```



❖ 코드 6-5. 고의로 instance 생성을 늦춤

- 인스턴스를 생성하는 문장 바로 앞에 Thread.sleep(1)를 이용해 1 ms 시간동안 정지하도록 함
- 현재로서는 큰 문제는 없음.

❖ 그러나, Printer Class가 **상태를 유지해야하는 경우에는 문제가 발생**

- 예) Counter 변수와 같은 값을 인스턴스가 유지해야함



다중 스레드 문제

❖ 코드 6-6

- 인스턴스마다 counter 변수를 각각 만들어 유지하기 때 문

Result

```
1-thread print using Printer@2cfa930d.1  
5-thread print using Printer@76cc518c.1
```

```
3-thread print using Printer@5ffdfb42.1  
2-thread print using Printer@5ffdfb42.2  
3-thread print using Printer@5ffdfb42.3
```

코드 6-6

```
public class Printer {  
    private static Printer printer = null;  
    private int counter = 0;  
    private Printer() { }  
  
    public static Printer getPrinter() {  
        if (printer == null) { // Printer 인스턴스가 생성되었는지 검사  
            try {  
                Thread.sleep(1);  
            }  
            catch (InterruptedException e) { }  
            printer = new Printer(); // Printer 인스턴스 생성  
        }  
        return printer;  
    }  
  
    public void print(String str) {  
        counter++; // 카운터 값 증가  
        System.out.println(str + counter);  
    }  
}
```



6.3 해결책 [p. 197]

❖ 다중 스레드 애플리케이션에서 발생하는 문제를 해결하는 방법 2가지

- 방법 1) 정적 변수에 인스턴스를 만들어 바로 초기화하는 방법
- 방법 2) 인스턴스를 만드는 메서드에 **동기화**하는 방법



방법 1) 정적 변수에 인스턴스를 만들어 바로 초기화하는 방법

❖ 코드 6-7

```
1-user print using Printer@2ce07e6b.  
2-user print using Printer@2ce07e6b.  
3-user print using Printer@2ce07e6b.  
4-user print using Printer@2ce07e6b.  
5-user print using Printer@2ce07e6b.
```

코드 6-7

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



방법 2) 인스턴스를 만드는 메서드에 동기화하는 방법

❖ 코드 6-8

```
3-thread print using Printer@949f69.  
5-thread print using Printer@949f69.  
2-thread print using Printer@949f69.  
4-thread print using Printer@949f69.  
1-thread print using Printer@949f69.
```

코드 6-8

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



문제

❖ 코드 6-9

- Printer 객체가 하나만 생성되었지만, counter 변수의 값이 이상하게 출력
- 여러 개의 스레드가 하나뿐인 counter 변수 값에 동시에 접근하기 때문

```
1-thread print using Printer@5ffcfb42.2  
3-thread print using Printer@5ffcfb42.5  
2-thread print using Printer@5ffcfb42.4  
5-thread print using Printer@5ffcfb42.3  
4-thread print using Printer@5ffcfb42.2
```

코드 6-9

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



해결

❖ 코드 6-10

```
4-thread print using Printer@2cfa30d.1  
2-thread print using Printer@2cfa30d.2  
1-thread print using Printer@2cfa30d.3  
5-thread print using Printer@2cfa30d.4  
3-thread print using Printer@2cfa30d.5
```

코드 6-10

```
public class Printer {  
    private static Printer printer = null;  
    private int counter = 0;  
    private Printer() { }  
  
    public synchronized static Printer getPrinter() {  
        if (printer == null) {  
            Printer = new Printer();  
        }  
        return printer;  
    }  
  
    public void print(String str) {  
        synchronized(this) { // 오직 하나의 스레드만을 접근을 허용함  
            counter++;  
            System.out.println( str+ counter );  
        }  
    }  
}
```

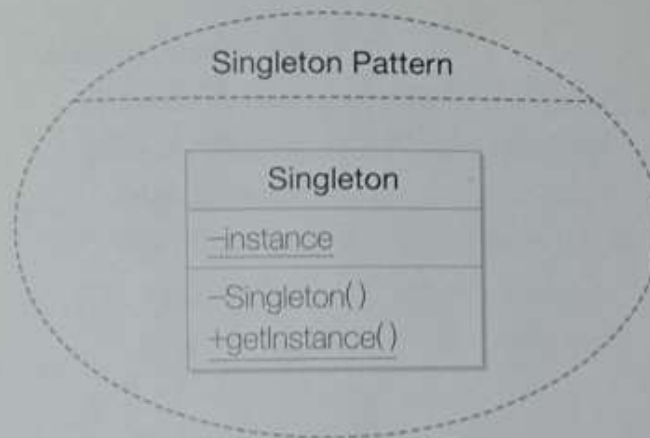


6.4 싱글턴(Singleton) 패턴

❖ 싱글턴 패턴

- singleton 1. (단독) 개체 2. 독신자(결혼을 안 했거나 애인이 없는 사람)
- 인스턴스가 오직 하나만 생성되는 것을 보장
- 어디서든지 이 인스턴스에 접근할수 있도록 함

그림 6-2 싱글턴 패턴 컬레보레이션

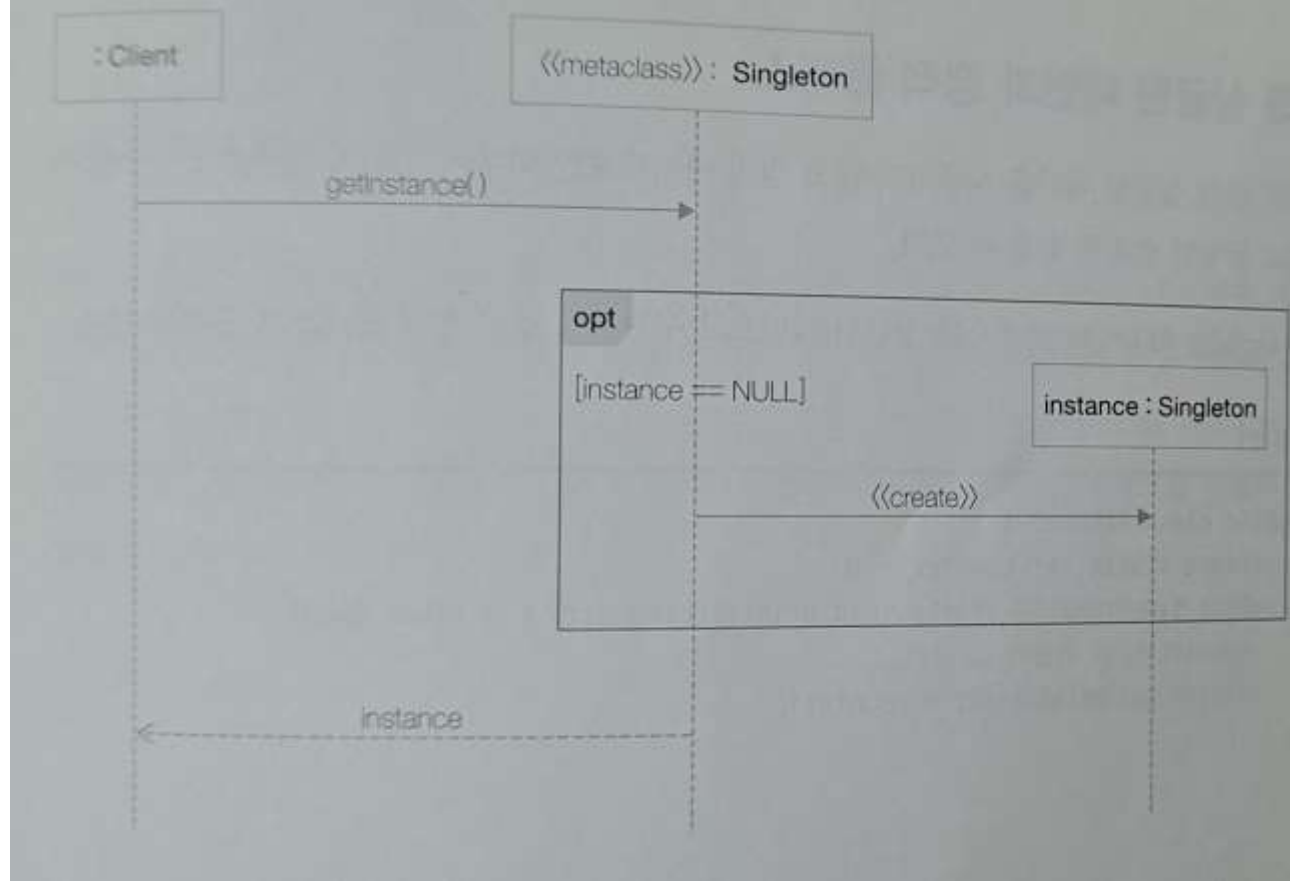


싱글턴 패턴은 매우 단순해 Singleton 요소 하나밖에 없다.

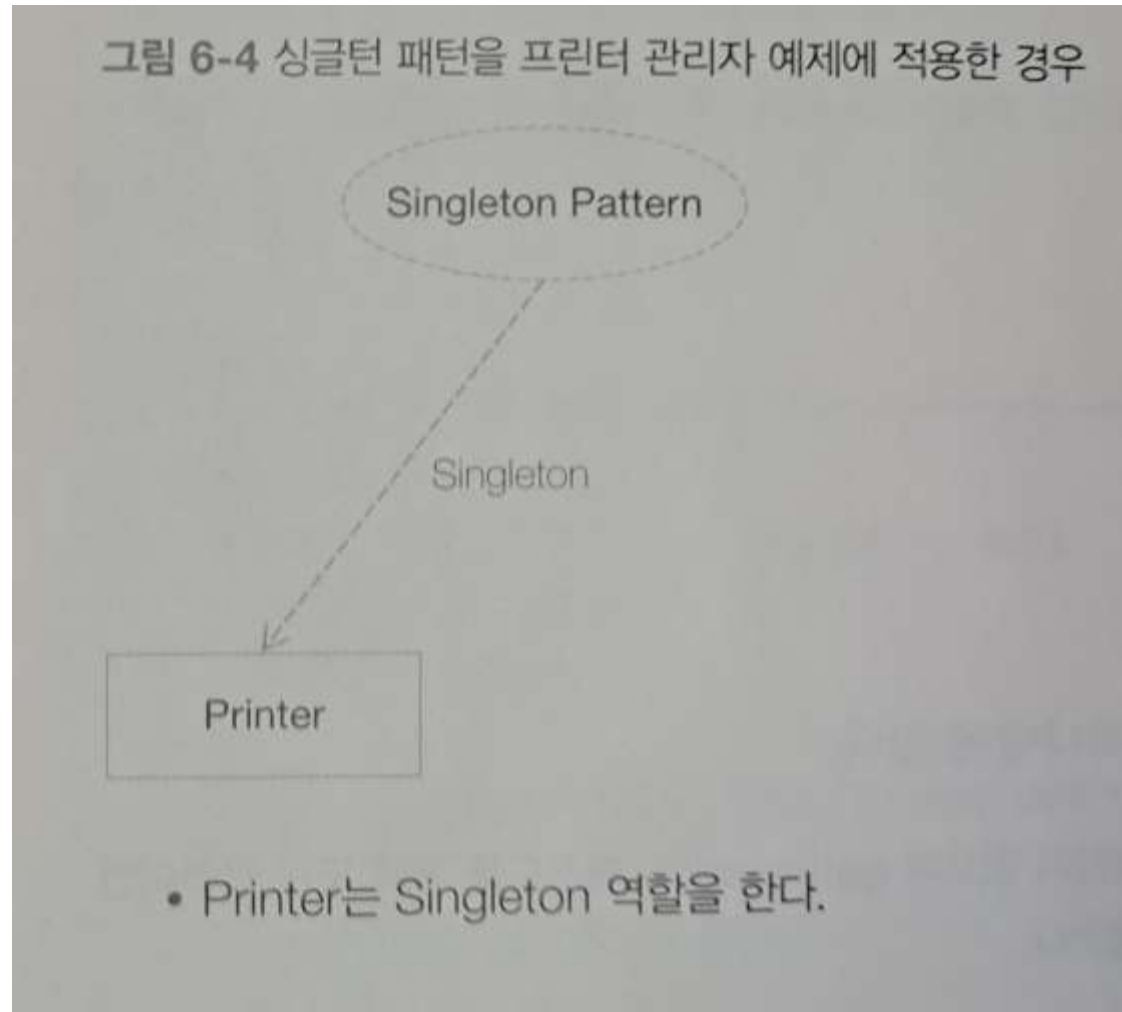
- Singleton: 하나의 인스턴스만을 생성하는 책임이 있으며 getInstance 메서드를 통해 모든 클라이언트에게 동일한 인스턴스를 반환하는 작업을 수행한다.



그림 6-3 싱글턴 패턴의 순차 다이어그램



❖ 그림 6-4



6.5 싱글턴 패턴과 정적 클래스

❖ 코드 6-11

- 정적 메소드로만 이루어진 Static Class를 사용하여 Singleton 패턴과 같은 동일한 효과를 얻을 수 있음

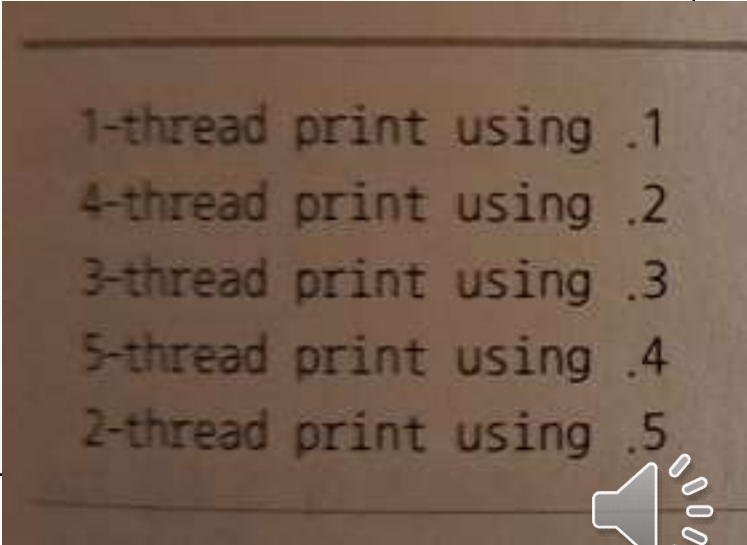
코드 6-11

```
public class Printer {  
    private static int counter = 0;  
  
    public synchronized static void print(String str) { // 메서드 동기화  
        counter++; // 카운터 값 증가  
        System.out.println(str + counter);  
    }  
}
```



코드 6-11

```
public class UserThread extends Thread {  
    public UserThread(String name) { // 스레드 생성  
        super(name);  
    }  
  
    public void run() { // 현재 스레드 이름 출력  
        Printer.print(Thread.currentThread().getName() + " print using " + "." );  
    }  
}  
  
public class Main {  
    private static final int THREAD_NUM = 5;  
    public static void main(String[] args){  
        UserThread[] user = new UserThread[THREAD_NUM];  
        for ( int i = 0; i < THREAD_NUM; i++) {  
            user[i] = new UserThread((i+1) + "-thread" );  
            user[i].start(); // 스레드 실행  
        }  
    }  
}
```



```
1-thread print using .1  
4-thread print using .2  
3-thread print using .3  
5-thread print using .4  
2-thread print using .5
```

❖ 코드 6-11

- 정적 클래스를 사용하는 방법

- 객체를 전혀 생성하지 않고 메소드를 사용함
- 아무 문제 없이 **counter** 변수가 스레드 5개 덕분에 안전하게 공유되어 사용됨
- 바인딩되는 (컴파일 타임에 바인딩되는) 인스턴스 메소드를 사용하는 것보다 성능면에서 우수

- 문제점 : 정적 클래스를 사용할수 없는 경우

- 예) 인터페이스를 구현해야 하는 경우
- 코드 6-12 와 같은 코드는 허용되지 않음



11
E

