

## 28. 싱글턴 패턴

- 싱글턴 패턴(singleton pattern) : 하나의 클래스는 **하나의 객체만** 갖도록 제한한다.
- 싱글턴 패턴은 하나의 객체 대한 **전역적인 접근점**을 제공한다.(GoF의 디자인 패턴)

### 〈싱글턴 패턴〉

Singleton
- singleton : Singleton
- Singleton()
+ getInstance() : Singleton

UML에서 밑줄은 자료형 **static**을 나타낸다.(속성 또는 메서드)

↓ 싱글턴 패턴을 자바로 구현하면

### 〈싱글턴 패턴에 대한 자바 코드〉

```
public class Singleton                //싱글턴 클래스
{
    private static Singleton singleton; //싱글턴 클래스 객체 선언 - 정적속성
    private Singleton(){               //생성자를 private으로 선언하면, 외부에서 접근 불가
    public synchronized static Singleton getInstance() //싱글턴 객체를 반환하기 위한 메서드
    {
        if(singleton == null)         //싱글턴 클래스 객체가 없으면
            singleton = new Singleton(); //싱글턴 클래스 객체 생성
        return singleton;             //싱글턴 클래스 객체 반환
    }
}
```

- 어떤 클래스에 대한 객체(instance)는 **오직 하나임을 보장**하는 패턴이다. - 정적(static) 필드
- 싱글턴 패턴은 프로그램 내에서 **오직 하나의 객체만 생성**되는 것을 보장하고,
- 최초 생성된 객체는 프로그램 어디에서나 접근할 수 있도록 하는 패턴이다.
- 싱글턴 패턴은 생성된 객체에 접근할 수 있는 **전역적인 접근점**을 제공하는 패턴이다.
- 외부에서 생성자를 통해 객체를 생성할 수 없도록 **생성자의 접근범위를 private**로 제한한다.
- 고정된 메모리 영역을 사용하고(데이터 공유), 메모리 낭비를 방지한다. - 정적속성
- 싱글턴 패턴은 **동시성** 문제 해결을 위해 키워드 **synchronized**를 사용한다.
- 싱글턴 패턴은 객체가 사용될 때마다 새로운 객체를 생성하지 않고, 최초 생성된 객체를 다시 사용하도록 구현해야 한다.
- 싱글턴 패턴 사용자 : 캐시, 대화상자, 스레드풀, 사용자 설정, 레지스트리 설정 등
- 싱글턴 패턴 문제점 : 싱글턴 객체가 많은 데이터를 공유하는 경우(높은 결합도 발생)

[예제] 싱글턴 패턴 - 클래스 싱글턴에서 하나의 객체만을 생성하는 프로그램

---

〈싱글턴 패턴 응용〉

Apple	Singleton	Test
	- singleton : Singleton	
+ Apple()	- Singleton()	
	+ getInstance() : Singleton	+ main(String[]) : void

---

↓자바로 구현하면

---

〈싱글턴 패턴 응용프로그램〉

```
class Apple //싱글턴 클래스와 비교하기 위한 클래스 정의
{
    public Apple(){ System.out.println("사과 객체 생성!"); }
}
class Singleton //싱글턴 클래스
{
    private static Singleton singleton; //싱글턴 클래스 객체 선언
    private Singleton(){ System.out.println("싱글턴 객체 생성!"); }
    public synchronized static Singleton getInstance()
    {
        if(singleton == null) //오직 하나의 객체만 생성 가능
            singleton = new Singleton(); //싱글턴 클래스 객체 생성
        return singleton; //싱글턴 클래스 객체 반환
    }
}
public class Test //싱글턴 클래스를 테스트 하기 위한 것
{
    public static void main(String[] args)
    {
        Apple apple1 = new Apple(); //클래스 Apple 객체 apple1 생성
        Apple apple2 = new Apple(); //클래스 Apple 객체 apple2 생성
        Singleton singleton1 = Singleton.getInstance(); //클래스 Singleton 객체 최초 생성
        Singleton singleton2 = Singleton.getInstance(); //최초 생성된 Singleton 객체 사용
    }
}
```

---

