

Spring Study - 스프링 핵심 원리 이해

Singleton Container

Singleton

Singleton class : 인스턴스를 오직 하나만 생성할 수 있는 클래스

장점 - 한 번의 객체 생성으로 재사용이 가능해져 메모리 낭비를 방지할 수 있다.

- 무상태 설계를 해야한다.

단점 - 잘못하고 공유 값을 설정? 버그 출몰 가능성 ↑

- 테스트 하기 힘들다.

Singleton

쉽게 말해 “절약정신”

Spring은 짤돌아

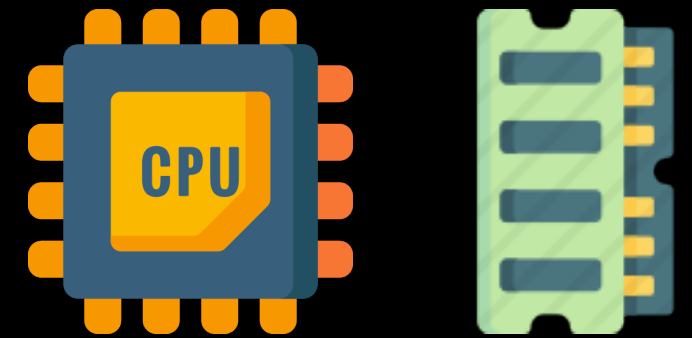


나 이거 사줘
저것도 사줘



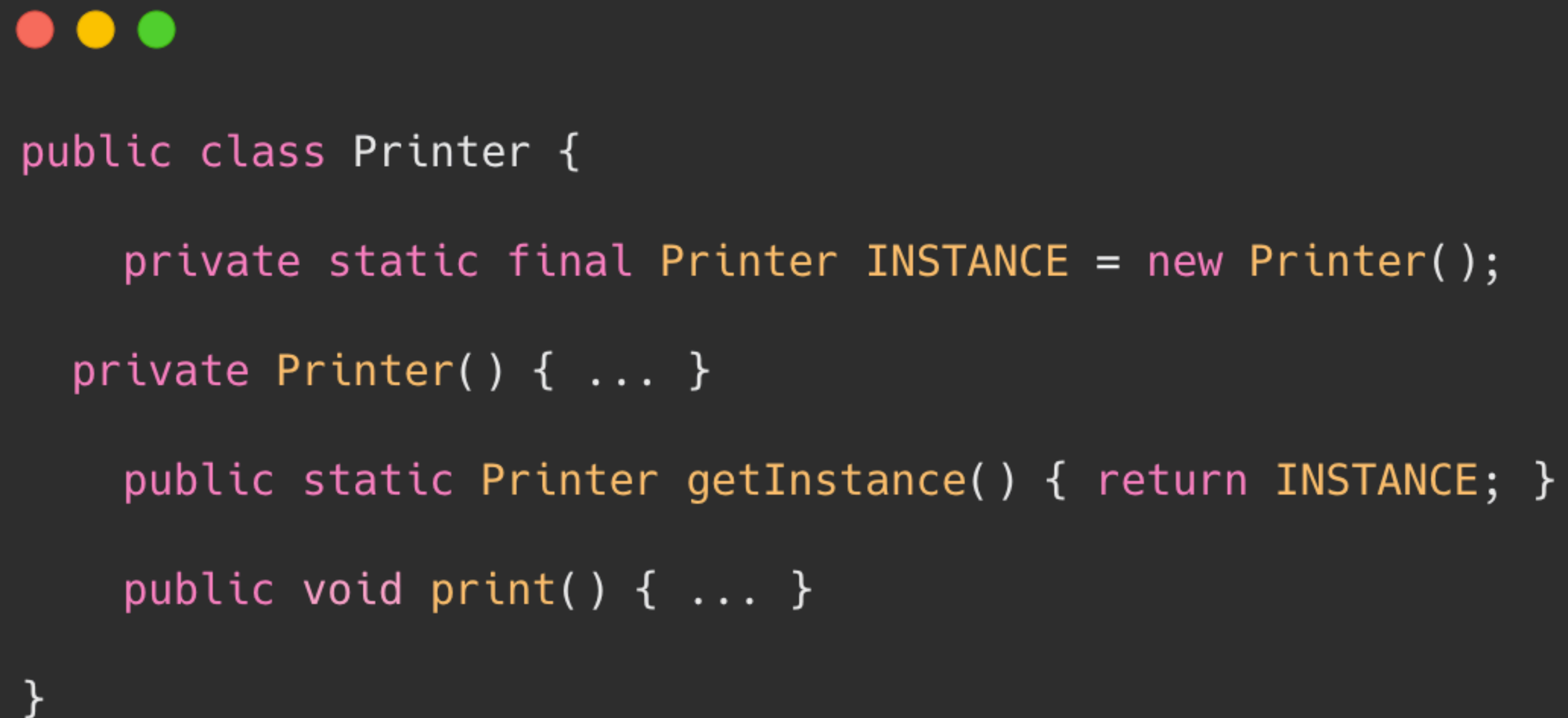
있는데 또 뭐냐 사~
야~ 있느거 그냥 쓰자~

Computer Money



다양한 Singleton Class 구현 방법

정적 팩터리 방식

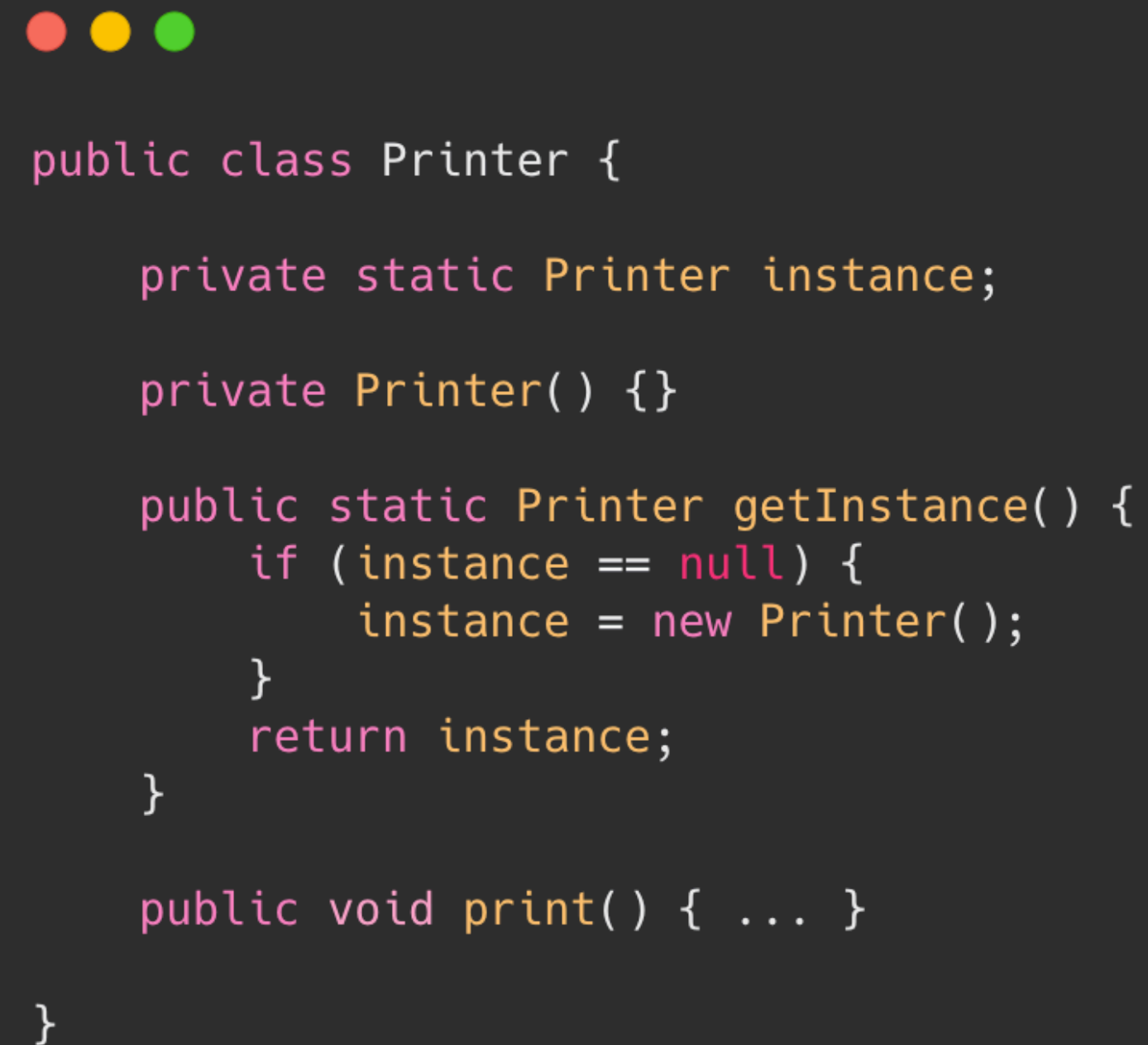


```
public class Printer {  
    private static final Printer INSTANCE = new Printer();  
    private Printer() { ... }  
    public static Printer getInstance() { return INSTANCE; }  
    public void print() { ... }  
}
```

정적 바인딩(컴파일 시점), Eager Initialization, Thread-safe -> Hmmmmmmmm....

다양한 Singleton Class 구현 방법

Lazy Initialization

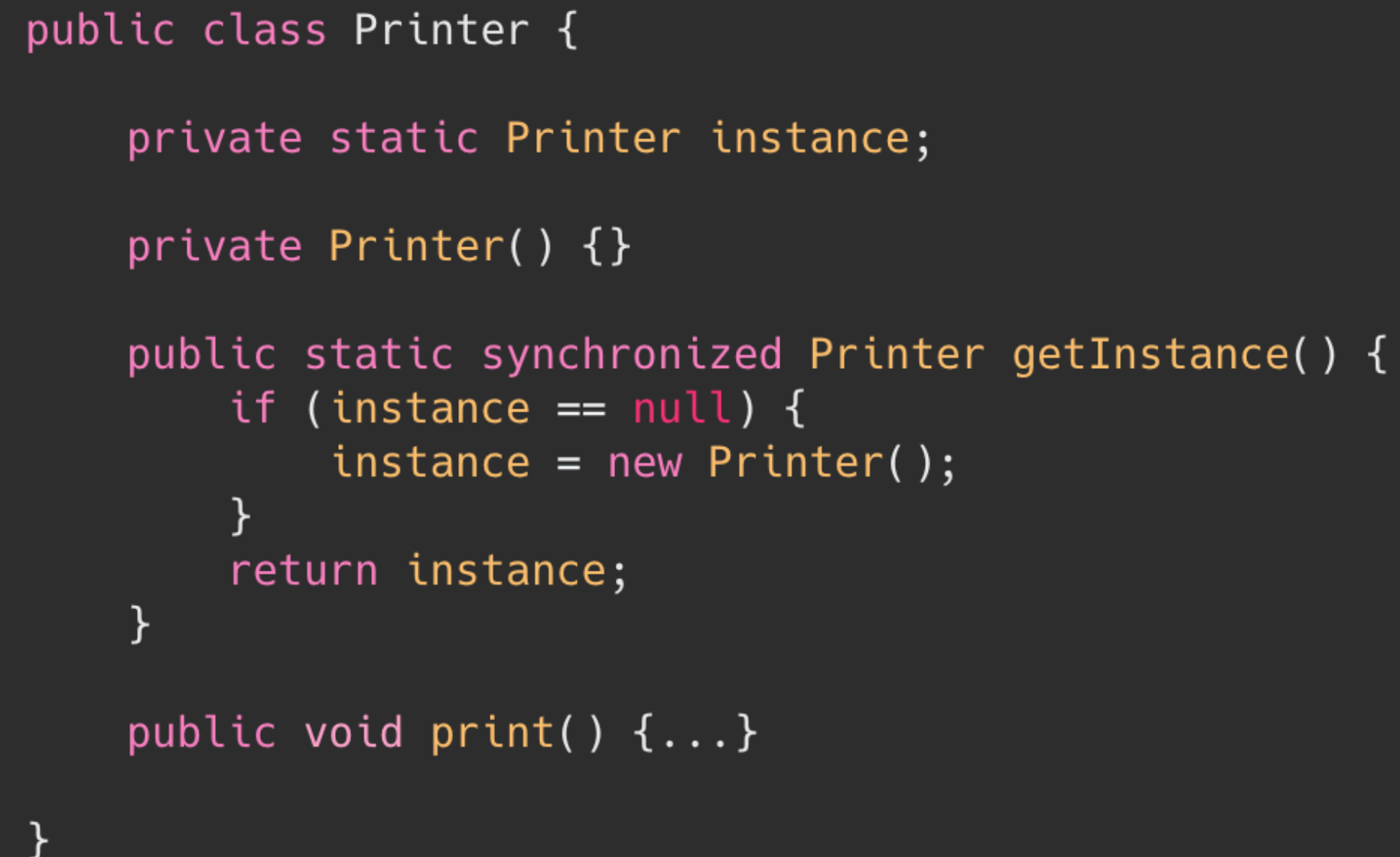


```
public class Printer {  
    private static Printer instance;  
  
    private Printer() {}  
  
    public static Printer getInstance() {  
        if (instance == null) {  
            instance = new Printer();  
        }  
        return instance;  
    }  
  
    public void print() { ... }  
}
```

동적 바인딩(런타임), but Multi thread ? -> 취약

다양한 Singleton Class 구현 방법

Lazy Initialization - synchronized



```
public class Printer {  
  
    private static Printer instance;  
  
    private Printer() {}  
  
    public static synchronized Printer getInstance() {  
        if (instance == null) {  
            instance = new Printer();  
        }  
        return instance;  
    }  
  
    public void print() {...}  
  
}
```

Thread-safe, but 무조건 거치는 synchronized 사용으로 인한 성능 저하

다양한 Singleton Class 구현 방법

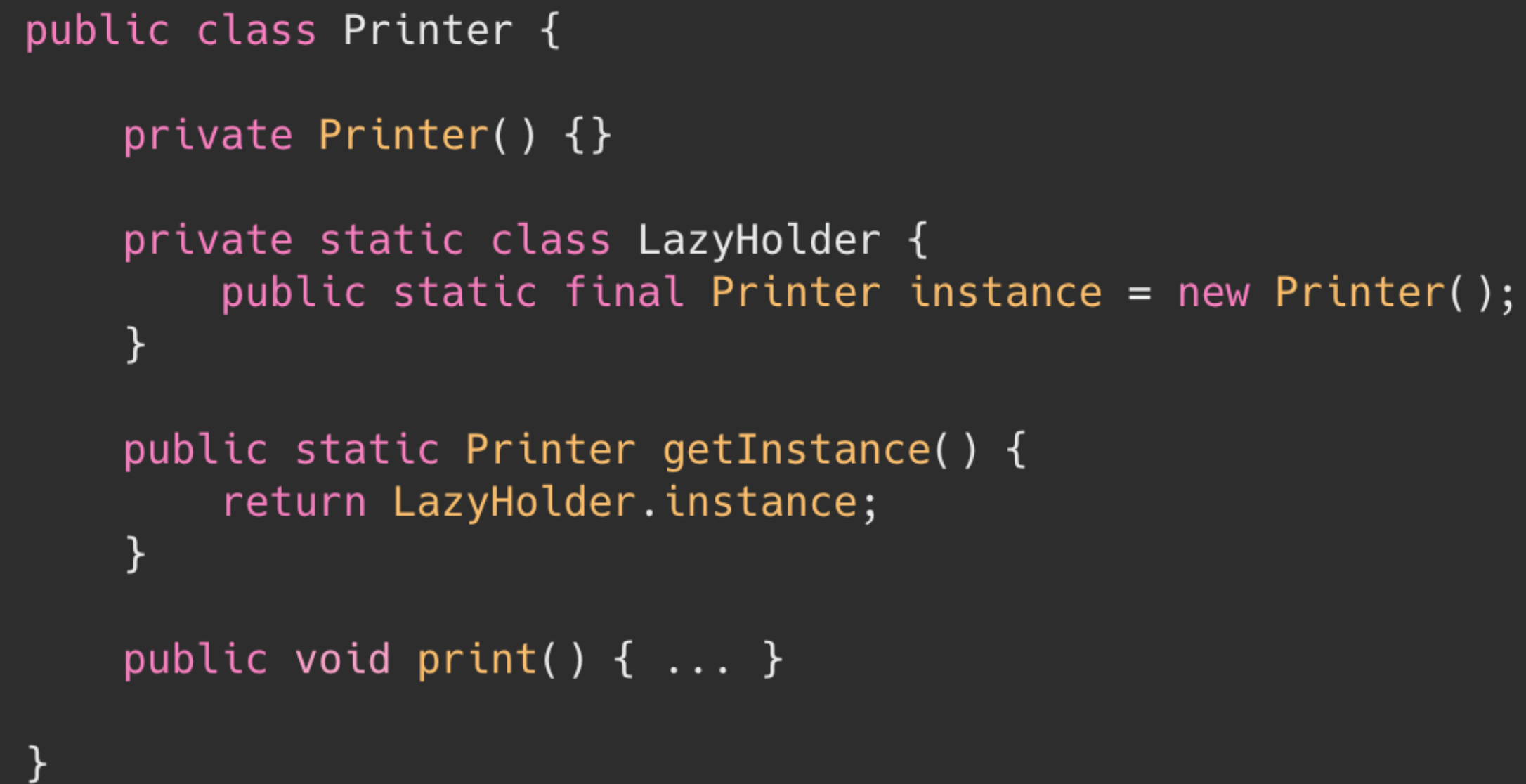
Lazy Initialization - Double Checking Locking

```
public class Printer {  
  
    private volatile static Printer instance;  
  
    private Printer() {}  
  
    public static Printer getInstance() {  
        if (instance == null) {  
            synchronized(Printer.class) {  
                if(instance == null) {  
                    instance = new Printer();  
                }  
            }  
        }  
        return instance;  
    }  
  
    public void print() {...}  
  
}
```

무조건적인 Synchronized 회피 -> 조금의 성능 향상

다양한 Singleton Class 구현 방법

Holder

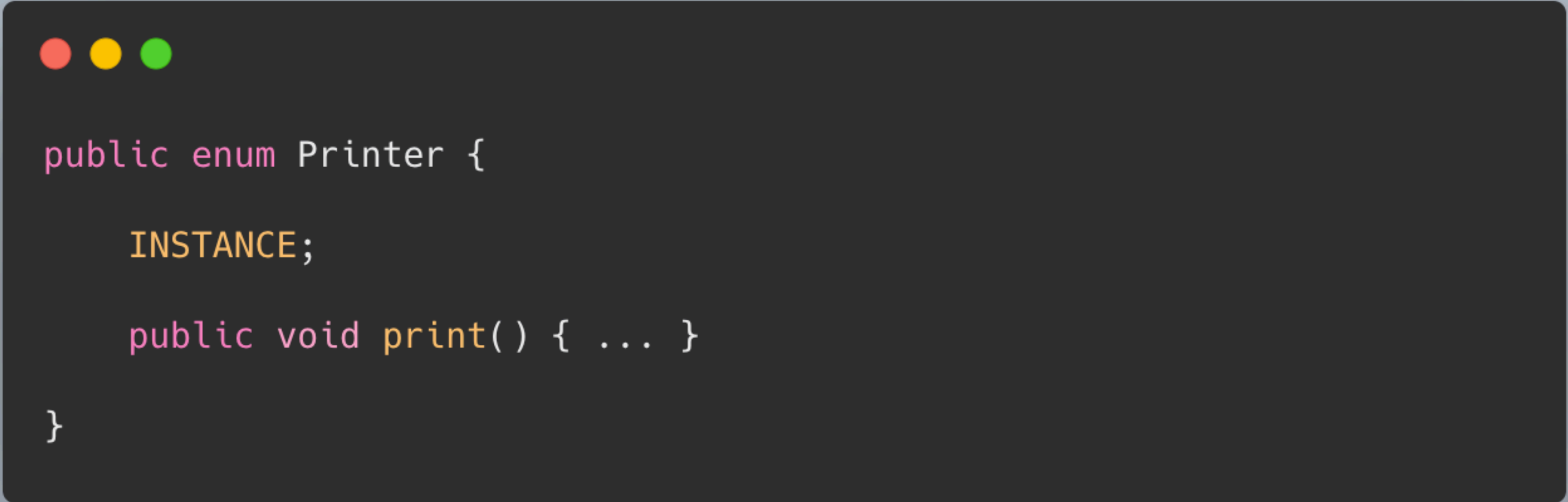


```
public class Printer {  
    private Printer() {}  
  
    private static class LazyHolder {  
        public static final Printer instance = new Printer();  
    }  
  
    public static Printer getInstance() {  
        return LazyHolder.instance;  
    }  
  
    public void print() { ... }  
}
```

Inner static class 로딩 시점 사용

다양한 Singleton Class 구현 방법

Enum

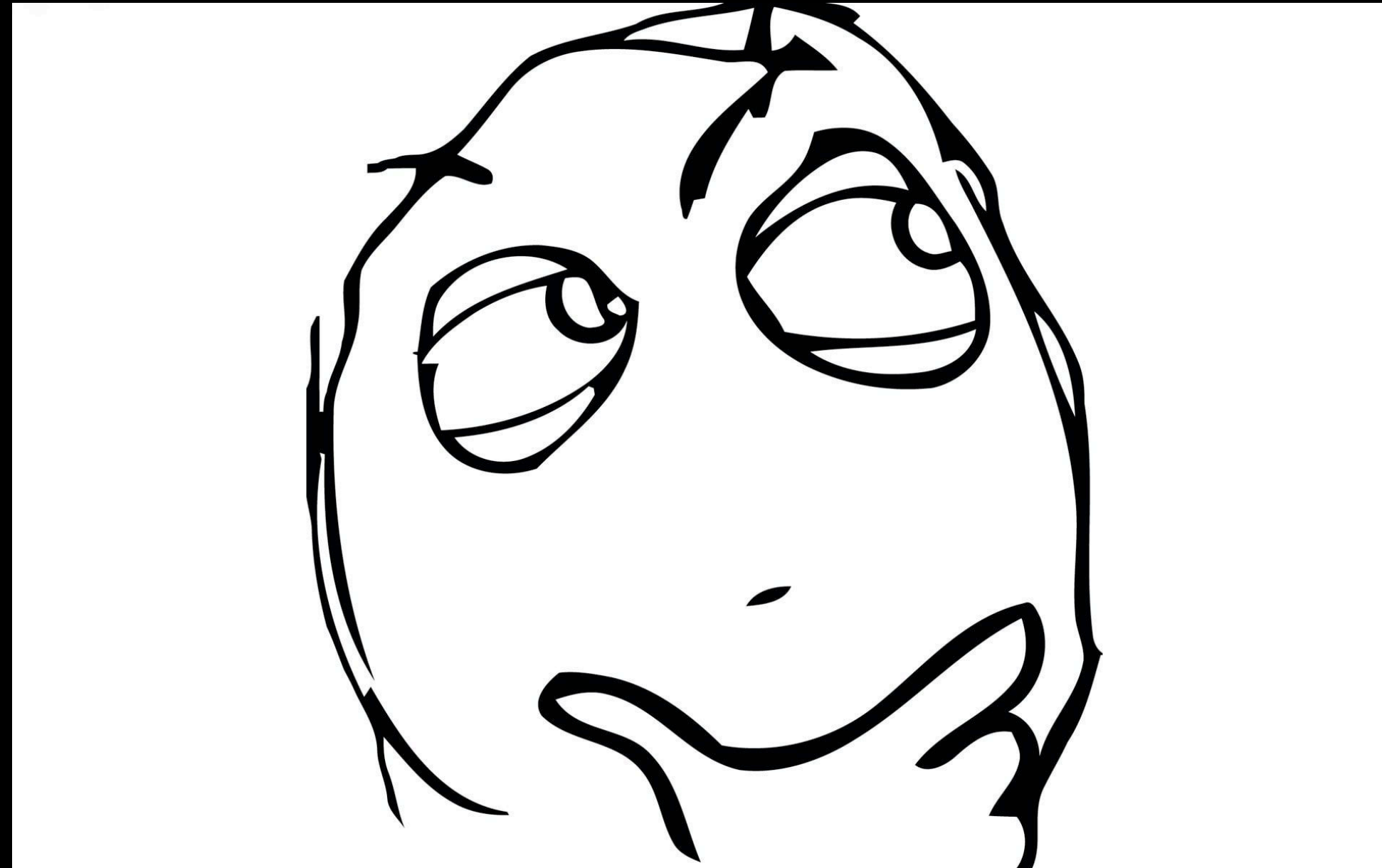


```
public enum Printer {  
    INSTANCE;  
  
    public void print() { ... }  
}
```

복잡하지 않음, 직렬화 가능, 리플렉션 공격 방지 -> 가장 좋은 방법 (Effective java)

Bean Scope

그럼 Spring은 Bean을 Singleton으로만 관리할까 ... ?



정답은 NO

다양한 Bean Scope가 존재한다 -> 다음 자료에서 ...