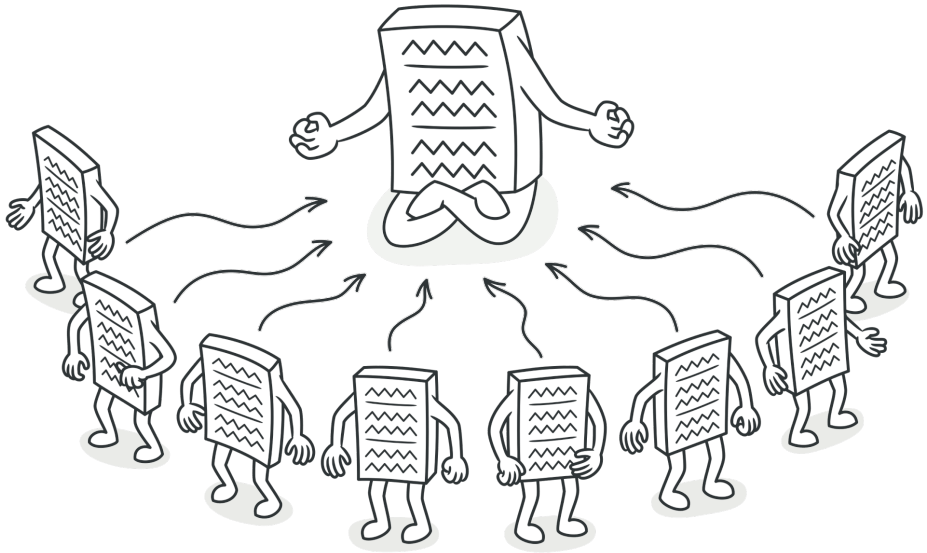
Ch11. Singleton Pattern

Jong-Hyeok Park
akindo19@gmail.com

Singleton Pattern

- 인스턴스가 **오직 1개만 생성**되어야 하는 경우 사용됨.
- 전역 변수를 사용하지 않음.
- 객체의 생성과 조합을 캡슐화해서 특정 객체가 생성/변경되어도 프로그램 구조에 영향을 받지 않도록 함.



Singleton Pattern

- Design
  - Private 생성자
  - Public static 생성 함수 선언 (인스턴스 접근용)
  - 클래스 자체에 속하는 static 변수 선언

Singleton Pattern

```
public class Printer {
    // 외부에 제공할 자기 자신의 인스턴스
    private static Printer printer = null;

    private Printer() { }
    // 자기 자신의 인스턴스를 외부에 제공

    public static Printer getPrinter(){
        if (printer == null) {
            // Printer 인스턴스 생성
            printer = new Printer();
        }
        return printer;
    }

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

Singleton Pattern

```
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());
    }
}
```

문제점

- *Single Responsibility Principle* 위반
- 복잡한 코드 (race condition 처리)
- Test 어려움
  - 격리된 환경 수행 : 매번 인스턴스 상태 초기화 필요
- Multi-Thread race condition 발생

해결책

- Eager Initialization
  - 정적 변수에 인스턴스 바로 초기화
  - 객체 생성 전 메모리 로딩 때 딱 1회 초기화 보장

```
public class Printer {
    // static 변수에 외부에 제공할 자기 자신의 인스턴스를 만들어
    초기화
    private static Printer printer = new Printer();
    private Printer() { }
    // 자기 자신의 인스턴스를 외부에 제공
    public static Printer getPrinter(){
        return printer;
    }
    public void print(String str) {
        System.out.println(str);
    }
}
```

해결책

- Thread-safe Initialization
  - Critical Path 명시 (synchronized 사용)

```
public class Printer {
    // 외부에 제공할 자기 자신의 인스턴스
    private static Printer printer = null;
    private int counter = 0;
    private Printer() { }
    // 인스턴스를 만드는 메서드 동기화 ( 임계 구역)
    public synchronized static Printer getPrinter(){
        if (printer == null) {
            printer = new Printer(); // Printer 인스턴스 생성
        }
        return printer;
    }
    public void print(String str) {
    // 오직 하나의 스레드만 접근을 허용함 ( 임계 구역)
    // 성능을 위해 필요한 부분만을 임계 구역으로 설정한다.
        synchronized(this) {
            counter++;
            System.out.println(str + counter);
        }
    }
    }
```

References

[1] Marc Gregoire, 2018, Professional C++, 4th edition, WILEY

[2] https://refactoring.guru/design-patterns/singleton

[3] https://gmlwjd9405.github.io/2018/07/06/singleton-pattern.html