Effective Java

Item 3. private 생성자나 열거 타입으로 싱글턴임을 보장하라.

싱글턴

싱글턴

인스턴스를 오직 하나만 생성할 수 있는 클래스

```java
public class Singleton {

    private static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

📌 설계상 유일해야하는 시스템 컴포넌트

📌 무상태 객체

싱글턴

인스턴스를 오직 하나만 생성할 수 있는 클래스

```java
// 인터페이스 정의
public interface DatabaseConnection {
    void connect();
}

// 싱글턴 구현
public class RealDatabaseConnection implements DatabaseConnection {

    private static final RealDatabaseConnection INSTANCE = new RealDatabaseConnection();

    private RealDatabaseConnection() {}

    public static RealDatabaseConnection getInstance() {
        return INSTANCE;
    }

    public void connect() {
    }
}
```

- 📌 메모리 절약
- 📌 상태 공유

싱글턴 단점

클라이언트에서 싱글턴 객체를 **테스트하기 어렵다.**

```java
public class Singleton {

    private static Singleton instance;

    private String state; // 전역으로 관리

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }
}
```

private 생성자

```java
public class SingletonTest {

    @Test
    public void test1() {
        Singleton singleton = Singleton.getInstance();
        singleton.setState("A");

        assertEquals("A", singleton.getState());  // 성공
    }

    @Test
    public void test2() {
        Singleton singleton = Singleton.getInstance();
        assertEquals("", singleton.getState());  // 실패! "A"
    }
}
```

private 생성자 접근?

Reflection API : AccessibleObject.setAccessible()

```java
public class SingletonTest {

  @Test
  void 리플렉션_테스트 {
        // Given
        Singleton singleton1 = Singleton.getInstance();

        // When
        // 리플렉션을 사용하여 private 생성자에 접근
        Constructor<Singleton> constructor = Singleton.class.getDeclaredConstructor();
        constructor.setAccessible(true);
        Singleton singleton2 = constructor.newInstance();

        // Then
        assertThat(singleton1).isNotSameAs(singleton2);
    }
}
```

싱글턴 테스트

인터페이스를 구현해 만든 싱글턴일 경우 테스트 ⭕

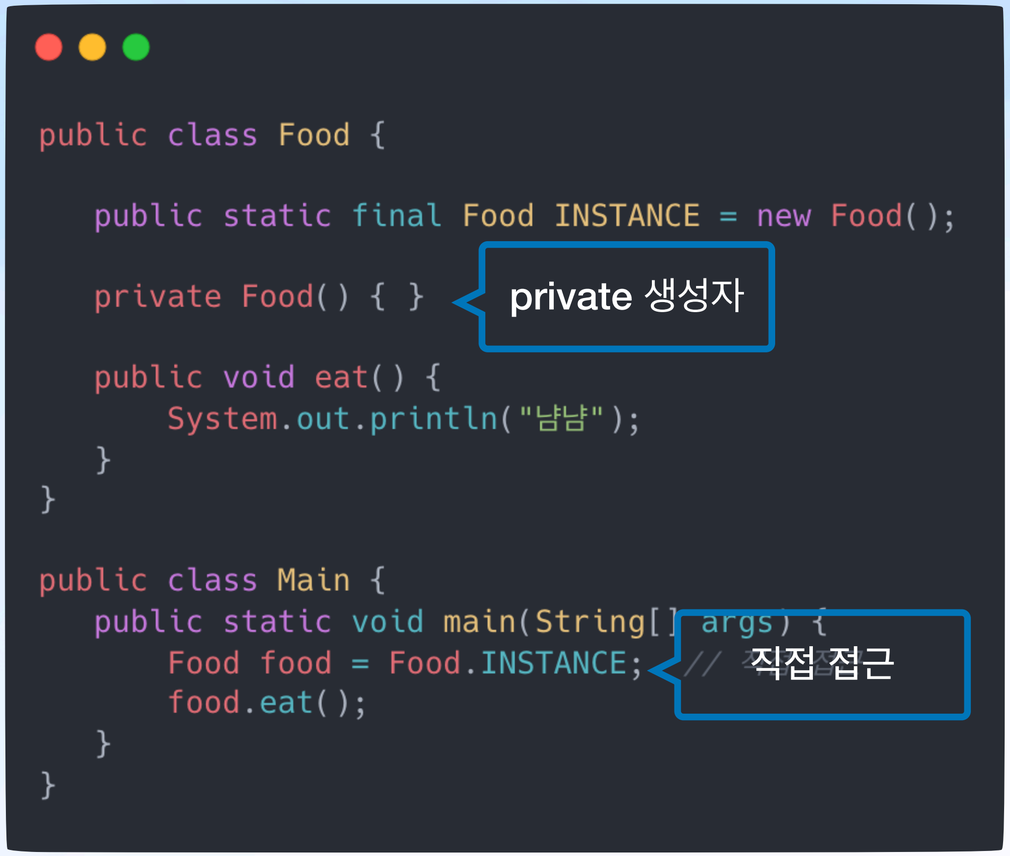
```java
// 인터페이스 정의
public interface DatabaseConnection {
    void connect();
}

// 싱글턴 구현
public class RealDatabaseConnection implements DatabaseConnection {

    private static final RealDatabaseConnection INSTANCE = new RealDatabaseConnection();

    private RealDatabaseConnection() {}

    public static RealDatabaseConnection getInstance() {
        return INSTANCE;
    }

    public void connect() {
    }
}
```

```java
// 가짜 싱글턴 객체
class FakeDatabaseConnection implements DatabaseConnection {
    private boolean connected = false;

    @Override
    public void connect() {
        this.connected = true;
    }

    public boolean isConnected() {
        return this.connected;
    }
}

class 싱글턴_테스트 {

    private FakeDatabaseConnection fakeDatabaseConnection;

    @BeforeEach
    void setUp() {
        fakeDatabaseConnection = new FakeDatabaseConnection();
    }

    @Test
    void 가짜_구현체로_싱글턴을_테스트한다() {
        fakeDatabaseConnection.connect();

        assertThat(fakeDatabaseConnection.isConnected()).isTrue();
    }
}
```

싱글턴 생성 방식

1) public static final 필드

싱글턴을 public static 멤버로 직접 접근한다.



2) 정적 팩터리 방식의 싱글턴

싱글턴 인스턴스를 private으로 만들어 정적 팩토리로 접근한다.


```java
public class Food {

    private static final Food INSTANCE = new Food();

    private Food() { }

    public static Elvis getInstance() { return INSTANCE; }

    public void eat() {
        System.out.println("냠냠");
    }
}

public class Main {
    public static void main(String[] args) {
        Food food = Food.getInstance();
        food.eat();
    }
}
```

3) Enum 타입의 싱글턴 ⭐

원소가 하나인 enum 타입을 선언한다.

```java
// 열거 타입 방식의 싱글턴 - 바람직한 방법 (25쪽)
public enum Food {

    INSTANCE;

    public void eat() {
        System.out.println("냠냠");
    }
}

public class Main {
   public static void main(String[] args) {
        Food food = Food.INSTANCE;
        food.eat();
    }
}
```

싱글톤에서의 직렬화/역직렬화

싱글톤 특성을 유지하기 위해 readResolve을 적절히 구현한다.

```java
public class Singleton implements Serializable {

    private static final Singleton INSTANCE = new Singleton();

    private transient String data;

    private Singleton() {
        data = "Singleton Data";
    }

    public static Singleton getInstance() {
        return INSTANCE;
    }

    // 역직렬화 시 호출
    private Object readResolve() {
        // 싱글턴의 기존 인스턴스를 반환
        return INSTANCE;
    }

    public String getData() {
        return data;
    }
}
```

3) Enum 타입의 싱글턴 ⭐

원소가 하나인 enum 타입을 선언한다.

```java
// 열거 타입 방식의 싱글턴 - 바람직한 방법 (25쪽)
public enum Food {

    INSTANCE;

    public void eat() {
        System.out.println("냠냠");
    }
}

public class Main {
   public static void main(String[] args) {
        Food food = Food.INSTANCE;
        food.eat();
    }
}
```

꿋