Enforce the singleton property with a private constructor or an enum type

정의

single-ton이란 정확히 한 번 인스턴스화되는 class를 의미합니다.

예시

- stateless object (function)
- unique한 system componenet

특징

test하기 어렵습니다.

- single-ton의 instance를 mock instance로 대체하기 어렵기 때문입니다.
- 또한 보통 test frame-work는 객체를 생성하여 테스트하기 때문이기도 하죠
- single-ton을 interface를 구현하여 만든다면 테스트할 수는 있습니다.

구현

1. public static final field

```
// Singleton with public final field  (Page 17)
public class Elvis {
    public static final Elvis INSTANCE = new Elvis();

    private Elvis() { }

    public void leaveTheBuilding() {
        System.out.println("Whoa baby, I'm outta here!");
    }

    // This code would normally appear outside the class!
    public static void main(String[] args) {
        Elvis elvis = Elvis.INSTANCE;
```

```
        elvis.leaveTheBuilding();
    }
}
```

장점

private constructor가 오직 한 번만 호출되고, final로 지정함으로써 오직 하나의 객체만 있음을 보장해줍니다.

side case - reflection

reflection을 이용해서 객체를 생성할 가능성이 있습니다.

이런 경우를 대비해서 private constructor에 exception을 던지도록 방어로직을 삽입할 수 있습니다.

2. static factory method

```
public class Elvis {
    private static final Elvis INSTANCE = new Elvis();
    private Elvis() { }
    public static Elvis getInstance() { return INSTANCE; }

    public void leaveTheBuilding() {
        System.out.println("Whoa baby, I'm outta here!");
    }

    // This code would normally appear outside the class!
    public static void main(String[] args) {
        Elvis elvis = Elvis.getInstance();
        elvis.leaveTheBuilding();
    }
}
```

장점

1. class가 single-ton임이 명확합니다. 1번 방법과 마찬가지로 static final이기 때문이지요.
2. static Factory method로 구현되어 있기 때문에 item 1설명의 장점을 모두 가집니다.
    - 추후 변경 가능성을 두도록 유연성을 제공합니다.
    - generic singleton factory로 만들 수 있습니다.
3. method reference가 supplier로 사용될 수 있습니다.

    `Elvise::instance` == `Supplier<Elvis>` 와 동일합니다.

Serializable 처리

serializable을 구현하려면 Serializable을 추가하는 것만으로는 안됩니다. single-ton임을 보장하려면

1. 모든 instance field를 transient로 바꾸고 readResolve method를 제공합니다. (transient 선언시 직렬화 과정에서 제외됨)
2. 또는 serialized instance가 deserialized될 때마다 instance가 생성될텐데, 이를 막기 위해서 아래의 readResolve emthod를 제공합니다.

```
//readResolve method to preserve singleton property
private Object readResolve() {
  return INSTANCE;
}
```

3. Enum

```
public enum Elvis {
    INSTANCE;

    public void leaveTheBuilding() {
        System.out.println("Whoa baby, I'm outta here!");
    }

    // This code would normally appear outside the class!
    public static void main(String[] args) {
        Elvis elvis = Elvis.INSTANCE;
        elvis.leaveTheBuilding();
    }
}
```

어색해 보일지라도 Enum 특성상 single-ton임을 보장할 수 있습니다.

장점

1. serialization을 쉽게 적용할 수 있습니다.
2. reflection attack, 정교한 serialization으로부터 single-ton임을 보장할 수 있습니다.

단점

1. enum은 상속받을 수 없습니다.