Effective Java

Item 4. 인스턴스화를 막으려거든 private 생성자를 사용하라.

개요

1. 정적 멤버만을 가진 클래스란 무엇인가?
2. 유틸리티 클래스 사용처
3. 인스턴스화를 막는 방법
4. 객체지향 관점 한계

정적 메서드와 정적 필드만을 담은 클래스

1. 유틸리티 클래스

java.lang.Math

```java
public final class Math {

    /**
     * Don't let anyone instantiate this class.
     */
    private Math() {}

    public static final double E = 2.718281828459045;
    public static final double PI =
3.141592653589793;

    @IntrinsicCandidate
    public static double log(double a) {
        return StrictMath.log(a);
    }

    @IntrinsicCandidate
    public static double sqrt(double a) {
        return StrictMath.sqrt(a);
    }
}
```

java.util.Arrays

```java
public final class Arrays {

    // Suppresses default constructor, ensuring non-instantiability.
    private Arrays() {}

    public static void sort(int[] a) {
        DualPivotQuicksort.sort(a, 0, 0, a.length);
    }

    public static void sort(int[] a, int fromIndex, int toIndex) {
        rangeCheck(a.length, fromIndex, toIndex);
        DualPivotQuicksort.sort(a, 0, fromIndex, toIndex);
    }

}
```

사용처

2. 팩토리 메서드 패턴

```java
public class Collections {
    // Suppresses default constructor, ensuring non-instantiability.
    private Collections() {
    }

  public static final <T> List<T> emptyList() {
        return (List<T>) EMPTY_LIST;
    }

     public static <T> List<T> unmodifiableList(List<? extends T> list) {
        if (list.getClass() == UnmodifiableList.class || list.getClass() ==
UnmodifiableRandomAccessList.class) {
           return (List<T>) list;
        }

        return (list instanceof RandomAccess ?
                new UnmodifiableRandomAccessList<>(list) :
                new UnmodifiableList<>(list));
    }
}
```

```java
public class Application {

    public static void main(String[] args) {
    List<String> immutableList =
Collections.unmodifiableList(originalList);
    // 비어있는 List를 만들어주는 정적 팩토리 메서드
    List<String> emptyList = Collections.emptyList();
    }
}
```

3. final 클래스 - 기능 추가

```java
public final class Money {

    private final int amount;

    public Money(int amount) {
        this.amount = amount;
    }

    public int getAmount() {
        return amount;
    }
}
```

```java
public class MoneyUtils {
    private MoneyUtils() {
    }

    public static Money add(Money money1, Money money2) {
        return new Money(money1.getAmount() + money2.getAmount(), money1.getCurrency());
    }

    public static Money subtract(Money money1, Money money2) {
        return new Money(money1.getAmount() - money2.getAmount(), money1.getCurrency());
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Money dollars1 = new Money(100, "USD");
        Money dollars2 = new Money(50, "USD");

        Money sum = MoneyUtils.add(dollars1, dollars2);
    }
}
```

기본 생성자 자동 생성

```
public class StringParser {

    public static int parseToInteger(final String input, final ErrorMessage message)
{       try {
            return Integer.parseInt(input.strip());
        } catch (NumberFormatException exception) {
            throw new CustomIllegalArgumentException(message);
        }
    }
}
```

public 생성자

```
public class Application {

    public static void main(String[] args) {
        StringParser stringParser = new
StringParser();
    }
}
```

인스턴스화 막는 방법 - 추상 클래스 ❌

직접적인 인스턴스화는 불가능 하더라도 **하위 클래스를 만들어 인스턴스화**가 가능하다.

```
abstract class StringParser {

    public static int parseToInteger(final String input, final ErrorMessage message)
{       try {
            return Integer.parseInt(input.strip());
        } catch (NumberFormatException exception) {
            throw new CustomIllegalArgumentException(message);
        }
    }
}
```

- 상속하라는 의미로 받아들여질 수 있음

인스턴스화 막는 방법

private 생성자를 명시하여 인스턴스화를 막는다.

```java
public final class Math {

    /**
     * Don't let anyone instantiate this class.
     */
    private Math() {}

    public static final double E = 2.718281828459045;
    public static final double PI =
3.141592653589793;

    @IntrinsicCandidate
    public static double log(double a) {
        return StrictMath.log(a);
    }

    @IntrinsicCandidate
    public static double sqrt(double a) {
        return StrictMath.sqrt(a);
    }
}
```

```java
public class UtilityClass {
    // 기본 생성자가 만들어지는 것을 막는다(인스턴스화 방지용).
    private UtilityClass() {
        throw new AssertionError();
    }
}
```

- 생성자 호출시 예외 throw
- **상속 불가능 ➡ 다형성 X**

static 메서드 - 다형성 ❌

static 메서드는 상속시 **오버라이드**가 되지 않는다.

static 메서드

```java
public class Parent {
    public static void printMessage() {
        System.out.println("Parent");
    }
}

public class Child extends Parent {
    public static void printMessage() {
        System.out.println("Child");
    }
}

public class Main {
    public static void main(String[] args) {
        Parent p = new Child();
        p.printMessage();        // Parent
    }
}
```

인스턴스 메서드

```java
public class Parent {
    public void instanceMethod() {
        System.out.println("Parent");
    }
}

public class Child extends Parent {
    @Override
    public void instanceMethod() {
        System.out.println("Child");
    }
}

Parent p = new Child();
p.instanceMethod();   // Child
```

static 메서드 - 캡슐화 위배

```java
public class OrderProcessor {
    public static void processOrder(Order order, Stock stock) {
        if (stock.getRemaining() >= order.getAmount()) {
            stock.subtract(order.getAmount());
        }
    }
}
```

- 객체의 데이터를 **파라미터**로 받아 처리한다.
- **데이터**와 **행위**가 분리된다.

절차지향 프로그래밍

Summary

- 정적 멤버만을 담은 클래스는 private 생성자를 통해 인스턴스화를 막을 수 있다.

```java
public class UtilityClass {
    // 기본 생성자가 만들어지는 것을 막는다(인스턴스화 방지용).
    private UtilityClass() {
        throw new AssertionError();
    }
}
```

- 특징

1. **상속**이 불가능하다. -> **private 생성자**

2. **객체지향 특성**을 위배한다 -> **다형성, 캡슐화 위배**

꿋