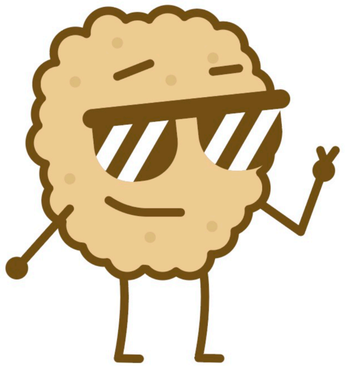
Item 4

인스턴스화를 막으려거든 private 생성자를 사용하라

강철원

인스턴스화를 막으려거든



인스턴스를 만들 필요가 없는 경우

🙌 자바에서 인스턴스가 필요하지 않는 경우

1. 정적 메서드 (static methods)만 사용시
2. 정적 변수 (static variables)만 사용 시
3. 유틸리티 클래스 사용시

📌 Class의 유틸리티(utility)성

특정 작업을 수행하는 데 도움이 되는

정적 메서드(static methods) 와 정적 변수(static variables)를 모아놓은 것

💎 유틸리티 클래스의 특징

1. 정적 메서드와 변수

2. 인스턴스 생성 방지

3. 일반적으로 final 클래스로 선언

4. java.lang.Math, java.util.Arrays, java.util.Collections 등

인스턴스 생성 방지? 어떻게?

? 추상 클래스

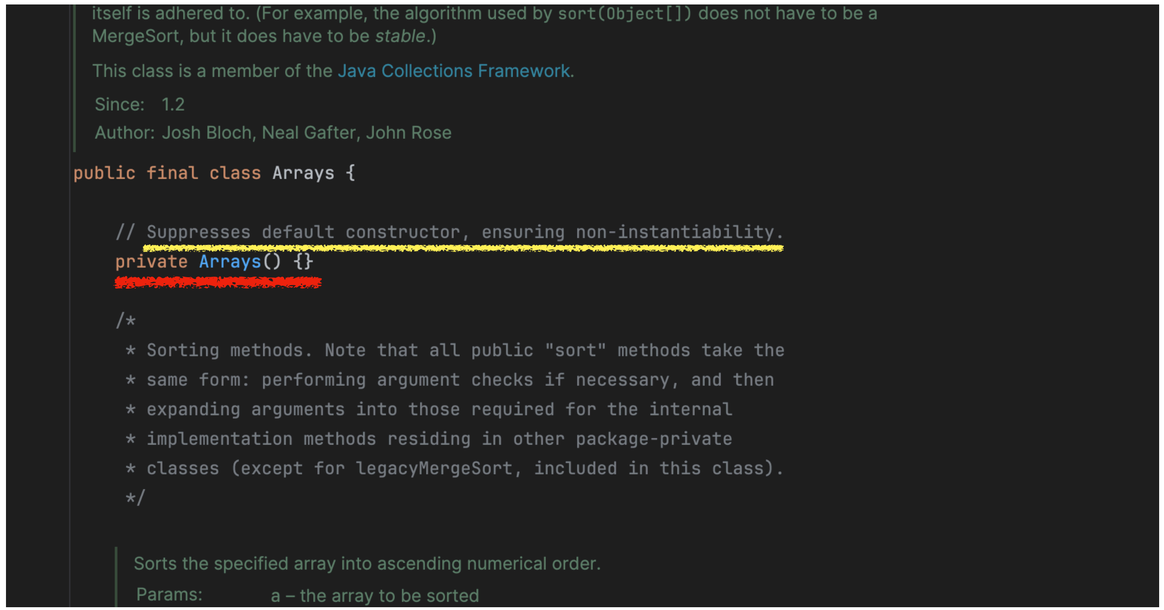
```
1 inheritor
public abstract class AnnotationConfigUtils {
    public static final String CONFIGURATION_ANNOTATION_PROCESSOR_BEAN_NAME = "org.springframework.context.a
    public static final String CONFIGURATION_BEAN_NAME_GENERATOR = "org.springframework.context.annotation.i
    public static final String AUTOWIRED_ANNOTATION_PROCESSOR_BEAN_NAME = "org.springframework.context.annot

    @Deprecated
    public static final String REQUIRED_ANNOTATION_PROCESSOR_BEAN_NAME = "org.springframework.context.annota
    public static final String COMMON_ANNOTATION_PROCESSOR_BEAN_NAME = "org.springframework.context.annotati
    public static final String PERSISTENCE_ANNOTATION_PROCESSOR_BEAN_NAME = "org.springframework.context.ann
    private static final String PERSISTENCE_ANNOTATION_PROCESSOR_CLASS_NAME = "org.springframework.orm.jpa.s
    public static final String EVENT_LISTENER_PROCESSOR_BEAN_NAME = "org.springframework.context.event.inter
    public static final String EVENT_LISTENER_FACTORY_BEAN_NAME = "org.springframework.context.event.interna
    private static final boolean jsr250Present;
    private static final boolean jpaPresent;

    public AnnotationConfigUtils() {
    }
```

```java
import org.springframework.context.annotation.AnnotationConfigUtils;

public class DefaultUtilityClass extends AnnotationConfigUtils {

    public static void main(String[] args) {
        DefaultUtilityClass utilityClass = new DefaultUtilityClass();
        utilityClass.processCommonDefinitionAnnotations(null);
    }
}
```

추상 클래스로 만드는 것으로는 인스턴스화를 막을 수 없습니다.

인스턴스화를 막으려거든 private 생성자를 사용하라

📁 java.util.Arrays / java.util.Collections / java.util.Math



itself is adhered to. (For example, the algorithm used by `sort(Object[])` does not have to be a MergeSort, but it does have to be *stable*.)

This class is a member of the Java Collections Framework.

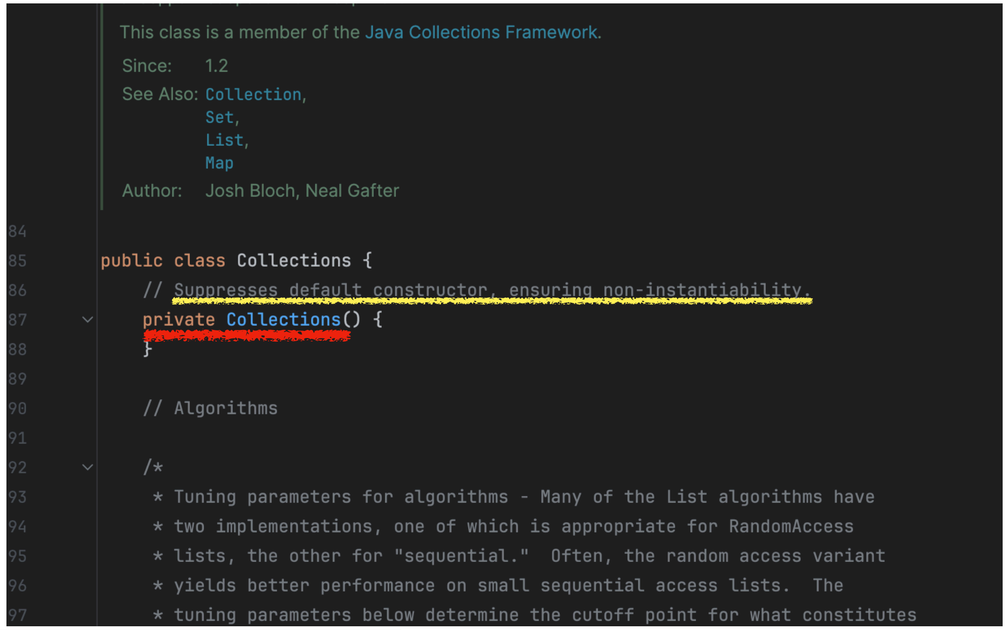
Since: 1.2

Author: Josh Bloch, Neal Gafter, John Rose

```
public final class Arrays {

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

    /*
     * Sorting methods. Note that all public "sort" methods take the
     * same form: performing argument checks if necessary, and then
     * expanding arguments into those required for the internal
     * implementation methods residing in other package-private
     * classes (except for legacyMergeSort, included in this class).
     */
```

Sorts the specified array into ascending numerical order.

Params: a – the array to be sorted



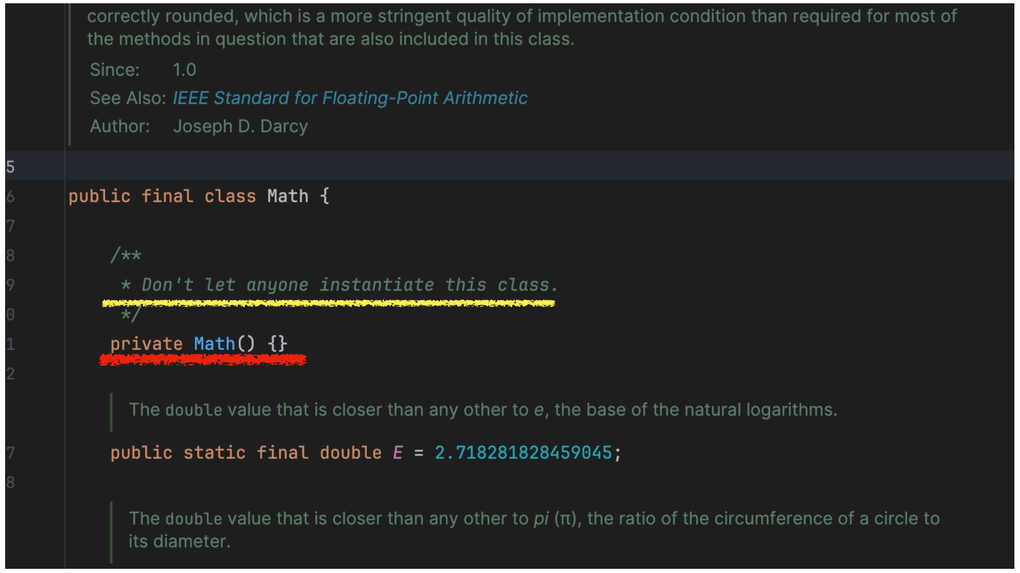
This class is a member of the Java Collections Framework.

Since: 1.2

See Also: Collection, Set, List, Map

Author: Josh Bloch, Neal Gafter

```
84
85  public class Collections {
86      // Suppresses default constructor, ensuring non-instantiability.
87      private Collections() {
88      }
89
90      // Algorithms
91
92      /*
93       * Tuning parameters for algorithms - Many of the List algorithms have
94       * two implementations, one of which is appropriate for RandomAccess
95       * lists, the other for "sequential."  Often, the random access variant
96       * yields better performance on small sequential access lists.  The
97       * tuning parameters below determine the cutoff point for what constitutes
```



correctly rounded, which is a more stringent quality of implementation condition than required for most of the methods in question that are also included in this class.

Since: 1.0

See Also: *IEEE Standard for Floating-Point Arithmetic*

Author: Joseph D. Darcy

```
5
6   public final class Math {
7
8       /**
9        * Don't let anyone instantiate this class.
0        */
1       private Math() {}
2
```

The `double` value that is closer than any other to *e*, the base of the natural logarithms.

```
7       public static final double E = 2.718281828459045;
8
```

The `double` value that is closer than any other to *pi* (π), the ratio of the circumference of a circle to its diameter.

기본 생성자를 억제하여

인스턴스화를 방지합니다.

😎 private

책 예시에서는

```java
public class UtilityClass {

    /**
     * 이 클래스는 인스턴스를 만들 수 없습니다.
     */
    private UtilityClass() {
        throw new AssertionError();
    }

    ... // 나머지 코드는 생략
}
```

🤔 왜 인스턴스화를 막을려고 할까?

유틸리티 클래스가 설계상 상태를 가지지 않고 (static 변수를 제외하고)

오직 메서드만을 제공하기 때문입니다.

이러한 클래스는 정적 메서드와 변수로만 구성되어 있어 객체의 인스턴스를 생성할 필요가 없습니다.

♟️ 인스턴스화를 막는다면

1️⃣ 명확한 의도

인스턴스화를 막음으로써 개발자는 이 클래스가 인스턴스화될 필요가 없다는 명확한 의도를 전달할 수 있습니다.

2️⃣ 리소스 절약

인스턴스 자체가 불필요하기 때문에 리소스 낭비를 방지할 수 있습니다.

3️⃣ 사용 용이성

객체를 생성하고 관리할 필요가 없게 만들어 사용의 용이성을 높입니다

4️⃣ 오류방지

개발자가 실수로 유틸리티 클래스의 인스턴스를 생성하려고 할 때, 컴파일 타임이나 런타임에 이를 방지함으로써 잠재적인 오류를 줄일 수 있습니다.

Item 4

인스턴스화를 막으려거든 private 생성자를 사용하라

강철원

끗