*Luke's Language로 파싱된 이펙티브 자바*

# 클린 자바

제작. 홍성혁

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

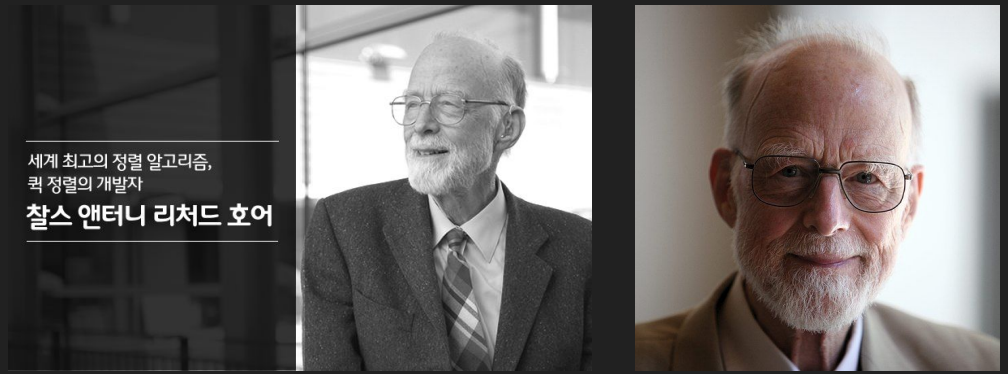
옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

## Null

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

- Sir Charles Antony Richard Hoare (A.K.A. Tony Hoare), 2009 Software Conference



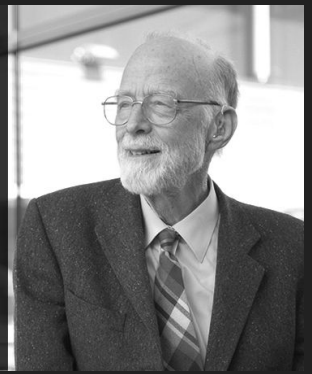
옵셔널 반환은 신중히 하라

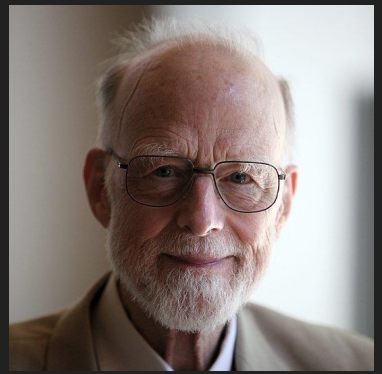
# Item 55 - 10억 달러 실수의 보완책, Optional

## Null

나는 그것을 나의 "10억 달러짜리 실수"라고 부릅니다. 그것은 1965년에 null 참조를 발명한 것이었습니다. 그 당시 나는 객체 지향 언어인 ALGOL W에서 참조를 위한 최초의 종합적인 타입 시스템을 설계하고 있었습니다. 나의 목표는 모든 참조의 사용이 컴파일러에 의해 자동으로 검사되어 절대적으로 안전하도록 하는 것이었습니다. 그러나 구현이 너무 쉬웠기 때문에 null 참조를 넣고 싶은 유혹을 이길 수 없었습니다. 이것은 수많은 오류, 취약점, 시스템 크래시를 일으켰으며, 지난 40년 동안 아마도 10억 달러의 고통과 피해를 초래했을 것입니다.

- Sir Charles Antony Richard Hoare (A.K.A. Tony Hoare), 2009 Software Conference



옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

**Null**

- 컴퓨팅 에서 널 포인터 또는 널 참조는 포인터 또는 참조가 유효한 객체 를 참조하지 않는다는 것을 나타내기 위해 저장된 값

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

## Null

- 컴퓨팅 에서 널 포인터 또는 널 참조는 포인터 또는 참조가 유효한 객체 를 참조하지 않는다는 것을 나타내기 위해 저장된 값
- 문제는 Null Pointer를 ***dereferencing*** 역참조하는데서 발생

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

## Null

- 컴퓨팅 에서 널 포인터 또는 널 참조는 포인터 또는 참조가 유효한 객체 를 참조하지 않는다는 것을 나타내기 위해 저장된 값
- 문제는 Null Pointer를 ***dereferencing*** 역참조하는데서 발생
  - C 에서 널 포인터의 역참조는 정의되지 않은 동작 입니다 . 많은 구현에서 이러한 코드로 인해 프로그램이 액세스 위반 으로 중단됩니다 . 이는 널 포인터 표현이 시스템에서 객체를 저장하기 위해 할당되지 않은 주소로 선택되기 때문입니다. 이러한 동작은 보편적이지 않습니다. 컴파일러는 정의되지 않은 동작이 없다는 가정 하에 프로그램을 최적화하기 때문에, 컴파일러가 안전을 보장하지도 못합니다.

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

## Null

- 컴퓨팅 에서 널 포인터 또는 널 참조는 포인터 또는 참조가 유효한 객체 를 참조하지 않는다는 것을 나타내기 위해 저장된 값
- 문제는 Null Pointer를 ***dereferencing*** 역참조하는데서 발생
  - C 에서 널 포인터의 역참조는 정의되지 않은 동작 입니다 . 많은 구현에서 이러한 코드로 인해 프로그램이 액세스 위반 으로 중단됩니다 . 이는 널 포인터 표현이 시스템에서 객체를 저장하기 위해 할당되지 않은 주소로 선택되기 때문입니다. 이러한 동작은 보편적이지 않습니다. 컴파일러는 정의되지 않은 동작이 없다는 가정 하에 프로그램을 최적화하기 때문에, 컴파일러가 안전을 보장하지도 못합니다.
- Java 에서는 null 참조에 액세스하면 NullPointerException(NPE)가 발생하고 이는 오류 처리 코드에서 처리할 수 있지만, 이러한 예외가 발생하지 않도록 하는 것이 가장 좋습니다.

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional<T>

- 자바 8에서 등장
- 불변 컬렉션 (엄밀히 말해, Collection<T>를 구현하진 않았음)

```
@jdk.internal.ValueBased
public final class Optional<T> {
    /** Common instance for {@code empty()}. */
    private static final Optional<?> EMPTY = new Optional<>( value: null);

    /** If non-null, the value; if null, indicates no value is present */
    private final T value;

    /** Returns an empty {@code Optional} instance.  No value is present for this ...*/
    public static <T> Optional<T> empty() {...}

    /** Constructs an instance with the described value. ...*/
    private Optional(T value) { this.value = value; }

    /** Returns an {@code Optional} describing the given non-{@code null} ...*/
    public static <T> Optional<T> of( @Flow(targetIsContainer = true) T value) { return new Optio

    /** Returns an {@code Optional} describing the given value, if ...*/
    /unchecked/
    public static <T> Optional<T> ofNullable( @Flow(targetIsContainer = true) T value) {...}

    /** If a value is present, returns the value, otherwise throws ...*/
     @NotNull  @Contract(pure = true)  @Flow(sourceIsContainer = true)
    public T get() {...}

    /** If a value is present, returns {@code true}, otherwise {@code false}. ...*/
    public boolean isPresent() { return value != null; }

    /** If a value is  not present, returns {@code true}, otherwise ...*/
    public boolean isEmpty() { return value == null; }

    /** If a value is present, performs the given action with the value, ...*/
    public void ifPresent(Consumer<? super T> action) {...}
```

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional<T>

- 자바 8에서 등장
- 불변 컬렉션 (엄밀히 말해, Collection<T>를 구현하진 않았음)

코드 55-1 컬렉션에서 최댓값을 구한다(컬렉션이 비었으면 예외를 던진다).

```
public static <E extends Comparable<E>> E max(Collection<E> c) {
    if (c.isEmpty())
        throw new IllegalArgumentException("빈 컬렉션");

    E result = null;
    for (E e : c)
        if (result == null || e.compareTo(result) > 0)
            result = Objects.requireNonNull(e);

    return result;
}
```

```
@jdk.internal.ValueBased
public final class Optional<T> {
    /** Common instance for {@code empty()}. */
    private static final Optional<?> EMPTY = new Optional<>( value: null);

    /** If non-null, the value; if null, indicates no value is present */
    private final T value;

    /** Returns an empty {@code Optional} instance.  No value is present for this ...*/
    public static <T> Optional<T> empty() {...}

    /** Constructs an instance with the described value. ...*/
    private Optional(T value) { this.value = value; }

    /** Returns an {@code Optional} describing the given non-{@code null} ...*/
    public static <T> Optional<T> of( @Flow(targetIsContainer = true) T value) { return new Optio

    /** Returns an {@code Optional} describing the given value, if ...*/
    /unchecked/
    public static <T> Optional<T> ofNullable( @Flow(targetIsContainer = true) T value) {...}

    /** If a value is present, returns the value, otherwise throws ...*/
     @NotNull  @Contract(pure = true)  @Flow(sourceIsContainer = true)
    public T get() {...}

    /** If a value is present, returns {@code true}, otherwise {@code false}. ...*/
    public boolean isPresent() { return value != null; }

    /** If a value is  not present, returns {@code true}, otherwise ...*/
    public boolean isEmpty() { return value == null; }

    /** If a value is present, performs the given action with the value, ...*/
    public void ifPresent(Consumer<? super T> action) {...}
```

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional<T>

- 자바 8에서 등장
- 불변 컬렉션 (엄밀히 말해, Collection<T>를 구현하진 않았음)

코드 55-1 컬렉션에서 최댓값을 구한다(컬렉션이 비었으면 예외를 던진다).

```
public static <E extends Comparable<E>> E max(Collection<E> c) {
    if (c.isEmpty())
        throw new IllegalArgumentException("빈 컬렉션");

    E result = null;
    for (E e : c)
        if (result == null || e.compareTo(result) > 0)
            result = Objects.requireNonNull(e);

    return result;
}
```

```
@jdk.internal.ValueBased
public final class Optional<T> {
    /** Common instance for {@code empty()}. */
    private static final Optional<?> EMPTY = new Optional<>( value: null);

    /** If non-null, the value; if null, indicates no value is present */
    private final T value;

    /** Returns an empty {@code Optional} instance.  No value is present for this ...*/
    public static <T> Optional<T> empty() {...}

    /** Constructs an instance with the described value. ...*/
    private Optional(T value) { this.value = value; }

    /** Returns an {@code Optional} describing the given non-{@code null} ...*/
    public static <T> Optional<T> of( @Flow(targetIsContainer = true) T value) { return new Optio

    /** Returns an {@code Optional} describing the given value, if ...*/
    /unchecked/
    public static <T> Optional<T> ofNullable( @Flow(targetIsContainer = true) T value) {...}

    /** If a value is present, returns the value, otherwise throws ...*/
     @NotNull  @Contract(pure = true)  @Flow(sourceIsContainer = true)
    public T get() {...}

    /** If a value is present, returns {@code true}, otherwise {@code false}. ...*/
    public boolean isPresent() { return value != null; }

    /** If a value is  not present, returns {@code true}, otherwise ...*/
    public boolean isEmpty() { return value == null; }

    /** If a value is present, performs the given action with the value, ...*/
    public void ifPresent(Consumer<? super T> action) {...}
```

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional<T>

- 자바 8에서 등장
- 불변 컬렉션 (엄밀히 말해, Collection<T>를 구현하진 않았음)

코드 55-2 컬렉션에서 최댓값을 구해 Optional<E>로 반환한다.

```
public static <E extends Comparable<E>>
        Optional<E> max(Collection<E> c) {
    if (c.isEmpty())
        return Optional.empty();

    E result = null;
    for (E e : c)
        if (result == null || e.compareTo(result) > 0)
            result = Objects.requireNonNull(e);

    return Optional.of(result);
}
```

```
@jdk.internal.ValueBased
public final class Optional<T> {
    /** Common instance for {@code empty()}. */
    private static final Optional<?> EMPTY = new Optional<>( value: null);

    /** If non-null, the value; if null, indicates no value is present */
    private final T value;

    /** Returns an empty {@code Optional} instance.  No value is present for this ...*/
    public static <T> Optional<T> empty() {...}

    /** Constructs an instance with the described value. ...*/
    private Optional(T value) { this.value = value; }

    /** Returns an {@code Optional} describing the given non-{@code null} ...*/
    public static <T> Optional<T> of( @Flow(targetIsContainer = true) T value) { return new Optio

    /** Returns an {@code Optional} describing the given value, if ...*/
    /unchecked/
    public static <T> Optional<T> ofNullable( @Flow(targetIsContainer = true) T value) {...}

    /** If a value is present, returns the value, otherwise throws ...*/
     @NotNull  @Contract(pure = true)  @Flow(sourceIsContainer = true)
    public T get() {...}

    /** If a value is present, returns {@code true}, otherwise {@code false}. ...*/
    public boolean isPresent() { return value != null; }

    /** If a value is  not present, returns {@code true}, otherwise ...*/
    public boolean isEmpty() { return value == null; }

    /** If a value is present, performs the given action with the value, ...*/
    public void ifPresent(Consumer<? super T> action) {...}
```

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 옵셔널을 반환하는 메서드는 null 을 반환하지 말자

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 옵셔널을 반환하는 메서드는 null 을 반환하지 말자

코드 55-4 옵셔널 활용 1 - 기본값을 정해둘 수 있다.

```
String lastWordInLexicon = max(words).orElse("단어 없음...");
```

코드 55-5 옵셔널 활용 2 - 원하는 예외를 던질 수 있다.

```
Toy myToy = max(toys).orElseThrow(TemperTantrumException::new);
```

코드 55-6 옵셔널 활용 3 - 항상 값이 채워져 있다고 가정한다.

```
Element lastNobleGas = max(Elements.NOBLE_GASES).get();
```

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 옵셔널을 반환하는 메서드는 null 을 반환하지 말자

```
Optional<ProcessHandle> parentProcess = ph.parent();
System.out.println("부모 PID: " + (parentProcess.isPresent() ?
    String.valueOf(parentProcess.get().pid()) : "N/A"));
```

```
System.out.println("부모 PID: " +
    ph.parent().map(h -> String.valueOf(h.pid())).orElse("N/A"));
```

```
streamOfOptionals
    .filter(Optional::isPresent)
    .map(Optional::get)
```

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 옵셔널을 반환하는 메서드는 null 을 반환하지 말자
- 컬렉션, 스트림, 배열, 옵셔널 같은 컨테이너 타입은 옵셔널로 감싸지 말자 (Item54)

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 옵셔널을 반환하는 메서드는 null 을 반환하지 말자
- 컬렉션, 스트림, 배열, 옵셔널 같은 컨테이너 타입은 옵셔널로 감싸지 말자 (Item54)
- 메서드 반환 타입에 Null이 가능할 때, Optional<T>를 사용하자

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 옵셔널을 반환하는 메서드는 null 을 반환하지 말자
- 컬렉션, 스트림, 배열, 옵셔널 같은 컨테이너 타입은 옵셔널로 감싸지 말자 (Item54)
- 메서드 반환 타입에 Null이 가능할 때, Optional<T>를 사용하자
- 박싱된 기본 타입을 담은 옵셔널을 사용하지 말자
  - 대신 OptionalInt, OptionalLong, OptionalDouble을 사용하자
- 옵셔널을 컬렉션이나 배열의 키, 값, 원소로 사용하는 것은 보통 옳지 않을 것이다.

옵셔널 반환은 신중히 하라

# Item 55 - 10억 달러 실수의 보완책, Optional

Optional 사용시 주의사항

- 결론: 옵셔널은 반환 용도 외에 쓰면 별로 안좋음

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술
  - How가 아닌 **What**

```
 * @param  index index of element to return; must be
 *         non-negative and less than the size of this list
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException if the index is out of range
 *         ({@code index < 0 || index >= this.size()})
 */
E get(int index);
```

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술

```
/**
 * 이 리스트에서 지정한 위치의 원소를 반환한다.
 *
 * <p>이 메서드는 상수 시간에 수행됨을 보장하지 <i>않는다</i>. 구현에 따라
 * 원소의 위치에 비례해 시간이 걸릴 수도 있다.
 *
 * @param  index 반환할 원소의 인덱스; 0 이상이고 리스트 크기보다 작아야 한다.
 * @return 이 리스트에서 지정한 위치의 원소
 * @throws IndexOutOfBoundsException index가 범위를 벗어나면,
 *         즉, ({@code index < 0 || index >= this.size()})이면 발생한다.
 */
E get(int index);
```

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술
  => @param 매개변수 뜻 설명
  - How가 아닌 What

```
 * @param   index index of element to return; must be
 *          non-negative and less than the size of this list
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException if the index is out of range
 *         ({@code index < 0 || index >= this.size()})
 */
E get(int index);
```

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술
  - How가 아닌 **What**

```
 * @param
 *
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException if the index is out of range
 *         ({@code index < 0 || index >= this.size()})
 */
E get(int index);
```

=> @return 반환값 뜻 설명

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술
  - How가 아닌 **What**

```
 * @param  index index of element to return; must be
 *
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException if the index is out of range
 *         ({@code index < 0 || index >= this.size()})
 */
E get(int index);
```

=> @throw 예외와 예외를 던지는 조건 설명

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술

```
/**
 * Retu
 *
 * @implSpec
 * This implementation returns {@code this.size() == 0}.
 *
 * @return true if this collection is empty
 */
public boolean isEmpty() { ... }
```

=> @implSpec 해당 메서드의 하위 클래스에서 재정의 해야 할 것을 알려줌

공개된 API 요소에는 항상 문서화 주석을 작성하라

# Item 56 - 문서화

API를 공개하려거든,
공개된 모든 클래스, 인터페이스, 메서드, 필드 선언에 문서화 주석을 달라

- 메서드용 문서화 주석에는 해당 메서드와 클라이언트 사이의 규약을 명료하게 기술
  - How가 아닌 **What**
- @param, @return, @throw
- @implSpec
- {@literal}
  - html 메타문자는 특별하게 취급되므로
- {@index}
  - API 문서 주요 키워드를 색인
- 제네릭 값에도 @param으로 문서화할 것
- 애너테이션 타입에도 문서화를 꼼꼼히 할 것
- 클래스, 또는 정적 메서드에 스레드 안전 수준을 명시할 것

지역변수의 범위를 최소화하라

# Item 57 - Local Variable Tip

가장 처음 쓰일 때 선언하기
선언과 동시에 초기화 하기
메서드를 작게 유지하고 한 가지 기능에 집중하기

지역변수의 범위를 최소화하라

# Item 57 - Local Variable Tip

가장 처음 쓰일 때 선언하기
선언과 동시에 초기화 하기
메서드를 작게 유지하고 한 가지 기능에 집중하기

=> 다음과 같은 예외가 있기도 함

```
// 생성자를 얻는다.
Constructor<? extends Set<String>> cons = null;
try {
    cons = cl.getDeclaredConstructor();
} catch (NoSuchMethodException e) {
    fatalError("매개변수 없는 생성자를 찾을 수 없습니다.");
}
```

지역변수의 범위를 최소화하라

# Item 57 - Local Variable Tip

가장 처음 쓰일 때 선언하기
선언과 동시에 초기화 하기
메서드를 작게 유지하고 한 가지 기능에 집중하기

**=>** 반복문을 이용한 변수 범위 최소화

```
for (Element e : c) {
    ... // e로 무언가를 한다.
}
```

```
Iterator<Element> i = c.iterator();
while (i.hasNext()) {
    doSomething(i.next());
}
```

```
for (Iterator<Element> i = c.iterator(); i.hasNext(); ) {
    Element e = i.next();
    ... // e와 i로 무언가를 한다.
}
```

E.O.D.