Chapter 02.코틀린 기초 - 05.널 안정성

널 안정성

1. 널 참조의 위험성

- 자바를 포함한 많은 프로그래밍 언어에서 가장 많이 발생하는 예외 유형이 바로 `NullPointerException`
- 줄여서 `NPE` 라고도 부른다
- `null` 을 발명한 토니호어는 1965년 null을 발명한 후 수십년간 수십억 달러의 시스템 오류와 피해가 발생했기 때문에 `1조원 짜리 실수` 였다고 고백한다
- 자바에선 NPE를 줄이기 위해 JDK8에서 `Optional` 을 지원하기 시작했다
- 자바의 **옵셔널(Optional)**은 값을 래핑하기 때문에 객체 생성에 따른 오버헤드가 발생하고, 컴파일 단계에서 Null 가능성을 검사하지 않음
- 코틀린을 비롯한 최신 언어에선 널 가능성을 컴파일러가 미리 감지해서 NPE 가능성을 줄일 수 있다

2. 코틀린에서 NPE를 해결하는 다양한 방법

- 코틀린은 언어적 차원에서 NPE가 발생할 가능성을 제거한다
- 코틀린의 타입은 기본적으로 `Non-Null` 타입이므로 null을 허용하지 않는다

```
val a : String = null // 컴파일 오류

var b : String = "aabbcc"
b = null // 컴파일 오류
```

- 코틀린은 null을 허용하는 `Nullable` 타입을 제공한다
- Nullable 참조는 컴파일 단계에서 널 안정성을 제공

```
var a : String? = null
a.length // 컴파일 오류
```

- Nullable 참조에 대한 접근은 안전 연산자를 사용한다

```
a?.length // safe-call 연산자
```

- 엘비스 연산자를 사용해 null이 아닌 경우 특정 값을 사용하도록 한다

```
val b : Int = if (a != null) a.length else 0


// 엘비스 연산자를 사용하면 좌변이 null인 경우 우변을 리턴한다
val b = a?.length ?: 0
```

- 이 다음은 실제 자바의 null 처리 코드를 코틀린으로 재작성해본다
- 자바의 null 처리 코드

```
import java.util.Optional;

public class Java_NullSafety {

    public static String getNullStr() {
        return null;
    }

    public static int getLengthIfNotNull(String str) {
        if (str == null || str.length() == 0) {
            return 0;
        }
        return str.length();
    }

    public static void main(String[] args) {
        String nullableStr = getNullStr();
        // Optional사용
        nullableStr = Optional.ofNullable(nullableStr)
                .orElse("null인 경우 반환");

        // Optional을 주석처리하고 null 참조를 사용하더라도 컴파일 오류가 발생하지 않음
        int nullableStrLength = nullableStr.length();
        System.out.println(nullableStrLength);

        // 메서드 내부에서 null을 검사하는 방법도 있다
        int length = getLengthIfNotNull(null);
        System.out.println(length);
    }
}
```

- 코틀린으로 변경한 null 처리 코드

```
fun getNullStr(): String? = null

fun getLengthIfNotNull(str: String?) = str?.length ?: 0


fun main() {
    val nullableStr = getNullStr()

   //val nullableStr = getNullStr() ?: "null인 경우 반환" 아래의 null 가능성이 제거됨
    //println(nullableStr)

    val nullableStrLength = nullableStr.length // 컴파일 오류
    println(nullableStrLength)
```

```
    println(getLengthIfNotNull(null))
}
```

---

3. 코틀린에서도 NPE가 발생할 수 있다

```
// 명시적 NPE 호출
throw NullPointerException()

// !! NotNull임을 단언하는 단언 연산자를 사용
val c: String? = null
val d = c!!.length
```

- 이 외에도 자바와 상호운용하는 경우 자바에서 NPE를 유발하는 코드를 코틀린에서 사용하면 NPE가 발생할 수 있다

```
println(Java_NullSafety.getNullStr().length)

//Exception in thread "main" java.lang.NullPointerException
```

- 코틀린에서 자바 코드를 사용할때는 항상 Nullable 가능성을 염두해 안전 연산자와 엘비스 연산자를 사용

```
println(Java_NullSafety.getNullStr()?.length)
```