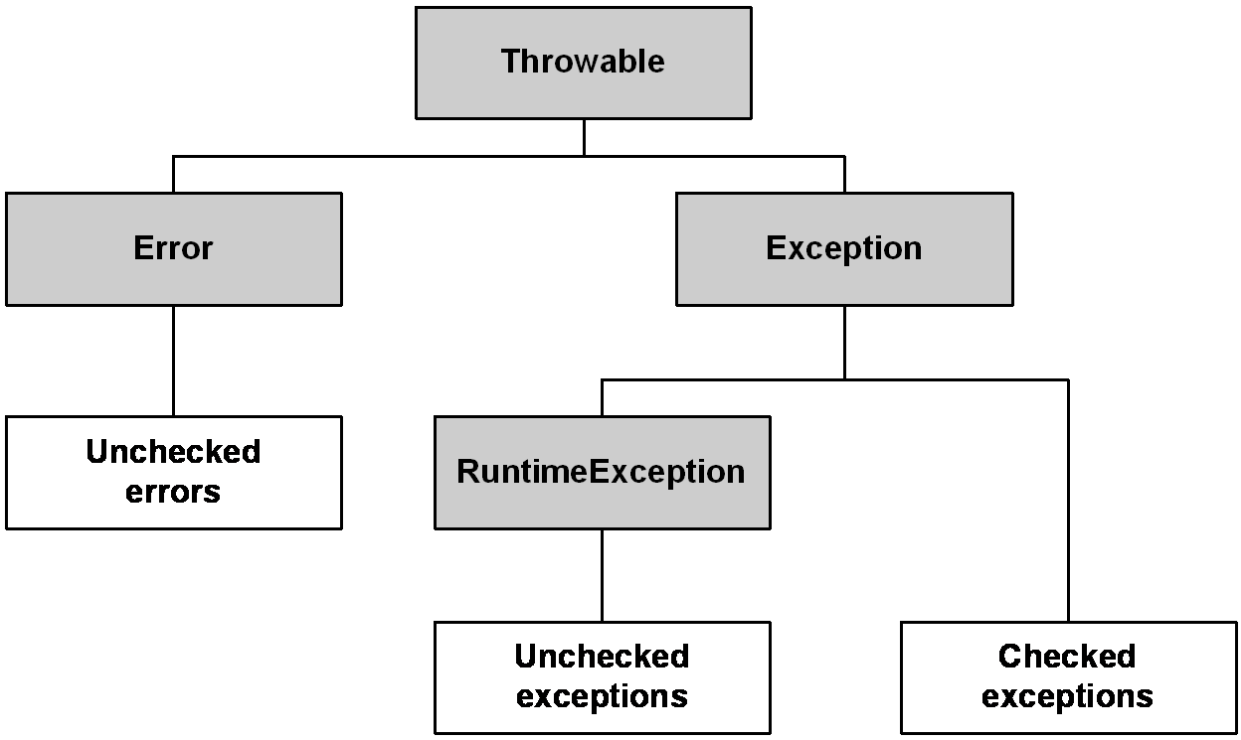
Chapter 01.코틀린 소개 - 03.자바와 코틀린의 차이점

자바와 코틀린의 차이점

1. 자바에는 있지만 코틀린에는 없는 기능

체크드 익셉션(Checked Exception)

- 자바의 익셉션 계층
  - Throwable : 예외 계층의 최상위 클래스
  - Error : 시스템에 비정상적인 상황이 발생. 예측이 어렵고 기본적으로 복구가 불가능 함
    - e.g. OutOfMemoryError, StackOverflowError, etc
  - Exception : 시스템에서 포착 가능하여(try-catch) 복구 가능. 예외 처리 강제
    - IOException, FileNotFoundException, etc
    - `@Transactional` 에서 해당 예외가 발생하면 기본적으론 롤백이 동작하지 않음
      - rollbackFor를 사용해야함
  - RuntimeException : 런타임시에 발생하는 예외. 예외 처리를 강제하지 않음
    - e.g. NullPointerException, ArrayIndexOutOfBoundsException, etc



- 자바에서 체크드 익셉션은 컴파일 에러가 발생하기 때문에 무조건 try-catch로 감싸거나 throws로 예외를 전파해야한다

```
try {
  Thread.sleep(1);
} catch (InterruptedException e) {
    // 예외 처리
}
```

  - 대부분의 개발자들이 자바에서 체크드 익셉션을 처리할때 의미 없는 처리를 반복한다
  - 체크드 익셉션이 발생할 경우 catch안에서 에러를 해결하는 일은 생각보다 흔하지 않고 오히려 생산성을 감소시킨다

```
try {
  log.append(message)
} catch(IOException e) {
  // Do nothing 흔하게 볼 수 있는 코드
```

```
}

try {
  File file = FileUtils.get(filename);
  // ...
} catch (FileNotFoundException e) {
  // 파일이 없는데 어떤 처리를 하지?
}


try {
  return objectMapper.readValue(json, clazz);
} catch (IOException e) {
  // 단순 에러 로그 출력
  logger.error(e.getMessage(), e);
}
```

- 코틀린은 체크드 익셉션를 강제하지 않는다

```
Thread.sleep(1);
```

- 원하는 경우에는 try-catch를 쓸 수 있다

```
try {
  Thread.sleep(1)
} catch (e: Exception) {
  // 예외 처리
}
```

기본 자료형

- 자바는 원시 자료형을 지원하며 객체로된 레퍼런스 타입도 지원한다

```
int i = 0;
Integer ii = 0;
String str = ii.toString();
```

- 코틀린은 레퍼런스 타입만 지원한다

```
val i: Int = 0;
val str: String = i.toString()
```

- 코틀린의 레퍼런스 타입은 최적화된 방식으로 컴파일된다

```
int i = 0;
String str = String.valueOf(i);
```

정적 멤버

- 자바는 static 키워드로 정적 멤버를 선언한다

```
pubic class JavaClass {

  static int i = 0;

  public static void staticMethod() {
    // ...
  }
}
```

- 코틀린은 `companion object`로 대체

```
class KotlinClass {

    companion object {
        val i: Int = 0

        fun function() {
            // ...
        }
    }
}
```

삼항 연산자

- 자바는 삼항 연산자가 존재한다

```
String animalSound =  "호랑이".equals(animal) ? "어흥" : "야옹";
```

- 코틀린은 if-else로 대신한다

```
val animalSound: String = if ("호랑이" == animal) "어흥" else "야옹"
```

세미콜론(;)

- 자바에서 라인의 끝은 무조건 세미콜론으로 끝나야한다

```
Boolean isAdmin = userService.isAdmin(userId);
```

- 코틀린은 세미콜론이 필수가 아니다

```
val isAdmin: Boolean = userService.isAdmin(userId)
```

---

2. 코틀린에는 있지만 자바에는 없는 기능

확장

- 개발자가 임의로 객체의 함수나 프로퍼티를 확장해서 사용할 수 있다

```
MyStringExtensions.kt


fun String.first(): Char {
    return this[0]
}

fun String.addFirst(char: Char): String {
    return char + this.substring(0)
}

fun main() {
    println("ABCD".first())   // 출력 : A

    println("ABCD".addFirst('Z'))   // 출력 : ZABCD
}
```

데이터 클래스

- 데이터를 보관하거나 전달하는 목적을 가진 **불변 객체**로 사용 `e.g. DTO`

```
data class Person(val name: String, val age: Int)
// equals(), hashCode(), toString()등 유용한 함수를 자동 생성
```

- 기존 자바에선 주로 Lombok을 사용

```
@Data
public class Person {
  private final String name;
  private final int age;
}
```

- JDK 15에서 record 라는 이름으로 추가됨

```
public record Person(String name, int age) {
}
```

문자열 템플릿

- 문자열에 변수를 사용하거나 여러 행으로 된 텍스트 블록을 만들 수 있다

```
val text = "World"
val greeting = "Hello, ${text}"

println(greeting) // Hello, World


// 문자열 템플릿 기반의 다이나믹 쿼리
fun sql(nameIncluded: Boolean) =
    """
        SELECT id, name, email, age
        FROM users
        WHERE id = :id
        ${
            if (nameIncluded) {
                """
                AND name = :name
```

```
                """
            } else ""
        }
    """
```

- 자바는 21년 9월 초안이 만들어졌고 현재는 프리뷰 상태

널 안정성

- 자바에서 가장 많이 발생하는 예외는 `NullPointerException` 줄여서 NPE
- 자바의 **옵셔널(Optional)**은 값을 래핑하기 때문에 객체 생성에 따른 오버헤드가 발생하고, 컴파일 단계에서 Null 가능성을 검사하지 않음
- 코틀린은 언어적 차원에서 NPE가 발생할 가능성을 제거한다

```
val a : String = null // 컴파일 오류

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

- Nullable 참조는 컴파일 단계에서 널 안정성을 제공

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

a?.length // safe-call 정상

a!!.length // Null이 아니라고 확신하는 경우
```

기타

- 스마트 캐스트
- 실드 클래스 (Jdk15 추가)
- 위임

- 중위 표현식
- 연산자 오버로딩
- 코루틴
- etc