Chapter 03.코틀린 고급 - 10.고급 예외처리

고급 예외처리

1. use를 사용한 리소스 해제

- Java7 부터 제공하는 `try-with-resources` 구문을 사용하면 자동으로 리소스를 close 처리 해준다

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class Java_TryWithResources {

    public static void main(String[] args) {

        try (PrintWriter writer = new PrintWriter("test.txt")) {
            writer.println("Hello World");
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        }

    }
}
```

- 정확히는 `Closeable` 또는 상위 개념인 `Autoloseable` 인터페이스의 구현체에 대해서 자동으로 close 메서드를 호출한다
- Closeable 인터페이스 구현하는 Writer

```
public abstract class Writer implements Appendable, Closeable
```

- Writer를 상속 받은 PrintWriter는 close 메서드를 재정의하고있다

```
public class PrintWriter extends Writer {
 ...

  public void close() {
        try {
            synchronized (lock) {
                if (out == null)
                    return;
                out.close();
                out = null;
            }
        }
        catch (IOException x) {
            trouble = true;
        }
    }

}
```

- 코틀린은 try-catch-resources 구문을 제공하지 않지만 `use` 라는 확장 함수를 제공한다

```
FileWriter("test.txt")
        .use { it.write("Hello Kotlin") }
```

2. runCatching을 사용해 우아하게 예외처리하기

- 코틀린은 try-catch를 통한 예외처리외에도 함수형 스타일의 `Result 패턴`을 구현한 `runCatching`을 제공한다
- Result 패턴이란 함수가 성공하면 캡슐화된 결과를 반환하거나 예외가 발생하면 지정한 작업을 수행하는 패턴이다

- try-catch 사용

```
fun getStr(): Nothing = throw Exception("예외 발생 기본 값으로 초기화")

fun main() {

    val result = try {
        getStr()
    } catch (e: Exception) {
        println(e.message)
        "기본값"
    }

    println(result)
}
// 예외 발생 기본 값으로 초기화
// 기본값
```

- runCatching 사용

```
val result2 = runCatching { getStr() }
        .getOrElse {
            println(it.message)
            "기본값"
        }
    println(result2)

// 예외 발생 기본 값으로 초기화
// 기본값
```

- runCatching 내부

```
public inline fun <R> runCatching(block: () -> R): Result<R> {
    return try {
        Result.success(block())
    } catch (e: Throwable) {
        Result.failure(e)
    }
}
```

- Result 내부 동반 객체에서 성공 상태와 실패 상태에 따라 Result를 생성하고 있음

```
public companion object {

      public inline fun <T> success(value: T): Result<T> =
          Result(value)

      public inline fun <T> failure(exception: Throwable): Result<T> =
          Result(createFailure(exception))
    }
```

- Result의 선언부

```
public value class Result<out T> internal constructor(

    internal val value: Any?
) : Serializable {

    public val isSuccess: Boolean get() = value !is Failure

    public val isFailure: Boolean get() = value is Failure

    ...
}
```

- `isSuccess`는 성공적인 결과를 나타내는 경우 true 반환
- `isFailure`는 실패한 결과를 나타내는 경우 true 반환

Result의 다양한 기능들

- `getOrNull` : 실패인 경우 null을 반환

```
val result3 = runCatching { getStr() }
        .getOrNull()
    println(result3)
// null반환
```

- `exceptionOrNull` : 성공인 경우 null을 반환하고 실패인 경우 Throwable을 반환

```
val result3: Throwable? = runCatching { getStr() }
        .exceptionOrNull()


result3?.let {
    println(it.message)
    throw it
}
```

- `getOrDefault` : 성공시엔 성공 값을 반환하고 실패시엔 지정한 기본 값을 반환

```
val result3 = runCatching { getStr() }
        .getOrDefault("기본 값")

println(result3)
// 기본 값
```

- `getOrElse` : 실패시 수신자 객체로 Throwable을 전달 받고 let, run과 같이 함수의 결과를 반환한다

```
val result3 = runCatching { getStr() }
        .getOrElse {
            println(it.message)
```

```
            "기본 값"
        }

println(result3)
// 예외 발생 기본 값으로 초기화
// 기본 값
```

- `getOrThrow` : 성공시엔 값을 반환 실패시엔 예외를 발생시킨다

```
val result3 = runCatching { getStr() }
        .getOrThrow()

// Exception in thread "main" java.lang.Exception: 예외 발생 기본 값으로 초기화
```

- `map` : 성공인 경우 원하는 값으로 변경할 수 있다

```
val result3 = runCatching { "안녕" }
        .map {
            it + "하세요"
        }.getOrThrow()

println(result3)
// 안녕하세요
```

- `mapCatching` : map 처럼 성공인 경우 원하는 값으로 변경할 수 있다 예외가 발생하면 재처리가능
    - map에서 예외가 발생한 경우

```
val result3 = runCatching { "안녕" }
        .map {
            getStr()
        }.getOrDefault("기본값")

    println(result3)
// Exception in thread "main" java.lang.Exception: 예외 발생 기본 값으로 초기화
```

- mapCatching에서 예외가 발생한 경우

```
val result3 = runCatching { "안녕" }
        .mapCatching {
            getStr()
        }.getOrDefault("기본값")

println(result3)
// 기본값
```

- `recover` : map은 성공시에 원하는 값으로 변경하지만 recover는 실패시에 원하는 값으로 변경할 수 있다

```
val result3 = runCatching { "정상" }
        .recover {
            "복구"
        }.getOrNull()

    println(result3)
// 정상


// 실패시
val result3 = runCatching { getStr() }
        .recover {
            "복구"
        }.getOrNull()

    println(result3)
// 복구
```

- `recoverCatching` : recoverCatching내에서 예외가 발생할 경우 재처리 가능
  - recover에서 예외가 발생한 경우

```
val result3 = runCatching { getStr() }
        .recover {
            getStr()
        }.getOrDefault("복구")
```

```
    println(result3)
// Exception in thread "main" java.lang.Exception: 예외 발생 기본 값으로 초기화
```

- recoverCatching에서 예외가 발생한 경우

```
val result3 = runCatching { getStr() }
        .recoverCatching {
            getStr()
        }.getOrDefault("복구")

    println(result3)
// 복구
```