Chapter 03.코틀린 고급 - 07.지연초기화

지연초기화

1. 지연초기화란

- `지연초기화`는 대상에 대한 초기화를 미뤘다가 실제 사용시점에 초기화하는 기법을 말한다
- 초기화 과정에서 자원을 많이 쓰거나 오버헤드가 발생할 경우 지연초기화를 사용하는게 유리할 수 있다
- 지연초기화는 많은 상황에서 쓰이고 있다
  - 웹페이지에서 특정 스크롤에 도달했을때 컨텐츠를 보여주는 무한 스크롤
  - 싱글톤 패턴의 지연초기화

```
public class Java_Singleton {

    private Java_Singleton() {
        /* do nothing */
    }

    public Java_Singleton getInstance() {
        return LazyHolder.INSTANCE;
    }

    private static class LazyHolder {
        private static final Java_Singleton INSTANCE = new Java_Singleton();
    }
```

```
}
```

  - JPA의 엔티티 LazyLoading 기능

```
@OneToMany(fetch = FetchType.LAZY)
```

  - 이외에도 지연초기화는 많은 상황에서 사용된다

- 코틀린은 두가지 다른 방식의 지연초기화를 제공한다

---

2. by lazy

- 예시와 같이 변수를 선언한 시점엔 초기화 하지 않다가 특정 시점 이후 초기화가 필요한 때

```
class HelloBot {

    var greeting: String? = null

    fun sayHello() = println(greeting)
}

fun getHello() = "안녕하세요"

fun main() {
    val helloBot = HelloBot()

    //...
    helloBot.greeting = getHello()
    helloBot.sayHello()
}
```

- 나중에 값을 수정할 수 있게 `var`로 선언하여 가변으로 만들어야한다
- 이전 `데이터클래스` 시간에도 알아봤듯이 var로 선언하는 것은 몇 가지 위험성을 가지고 있으므로 될수 있으면 불변으로 유지하는 것이 좋다

- 이때 코틀린에서 제공하는 `by lazy` 를 사용하면 불변성을 유지하면서 지연초기화가 가능하다
- lazy 함수 내부에 초기화 로직을 구현한다

```
class HelloBot {

    val greeting: String by lazy { getHello() }

    fun sayHello() = println(greeting)
}

fun getHello() = "안녕하세요"

fun main() {
    val helloBot = HelloBot()

    // ...
    helloBot.sayHello()
}
// 안녕하세요
```

- by lazy를 사용하면 사용 시점에 1회만 초기화 로직이 동작한다

```
class HelloBot {

    val greeting: String by lazy {
        println("초기화 로직 수행")
        getHello()
    }

    fun sayHello() = println(greeting)
}

fun getHello() = "안녕하세요"

fun main() {
    val helloBot = HelloBot()

    // ...
    helloBot.sayHello()
    helloBot.sayHello()
}
```

```
// 초기화 로직 수행
// 안녕하세요
// 안녕하세요
```

- by lazy는 기본적으로 멀티-스레드 환경에서도 안전하게 동작하도록 설계되었다

```
fun main() {
    val helloBot = HelloBot()

    // ...
    for (i in 0..5) {
        Thread{
            helloBot.sayHello()
        }.start()
    }
}
// 초기화 로직 수행
// 안녕하세요
// 안녕하세요
// 안녕하세요
// 안녕하세요
// 안녕하세요
// 안녕하세요
```

- by lazy 내부에서 스레드에 대한 동기화처리를 해주기 때문이다 (내부코드확인)
- 기본 모드는 `LazyThreadSafetyMode.SYNCHRONIZED` 과 동일하다
- 만약 `LazyThreadSafetyMode.NONE` 모드로 변경한 뒤 멀티-스레드 환경에서 실행하면?

```
class HelloBot {

    val greeting: String by lazy(LazyThreadSafetyMode.NONE) {
        println("초기화 로직 수행")
        getHello()
    }

    fun sayHello() = println(greeting)
}
```

- 실행할때마다 결과가 계속 달라지는 것을 볼 수 있다

- 멀티-스레드 환경이 아닌 경우에는 동기화 작업이 오버헤드가 될 수 있으므로 `NONE` 모드를 사용하고
- 멀티-스레드 환경이어도 동기화가 필요하지 않은 경우라면 `PUBLICATION` 모드를 사용

---

3. lateinit

- 가변 프로퍼티에 대한 지연초기화가 필요한 경우가 있다
- 특정 프레임워크나 라이브러리에서 지연초기화가 필요한 경우
- 예를 들어 테스트 코드를 작성할때 특정 애노테이션에 초기화 코드를 작성해야하는 경우가 있다

```
class `7_LateExample` {

    @Autowired
    lateinit var service: TestService

    lateinit var subject: TestTarget

    @SetUp
    fun setup() {
        subject = TestTarget()
    }

    @Test
    fun test() {
        subject.doSomething()
    }
}
```

- 이와 같은 경우엔 `lateinit`을 사용해 해결한다
- lateinit을 사용할땐 가변 변수여야한다

```
lateinit val text: String // 컴파일에러
```

- val인 경우 컴파일 오류가 발생하며 lateinit을 이용한 경우엔 항상 non-null이다

```
lateinit var text: String? // 컴파일에러
```

- 즉 lateinit은 가변이어야하며 non-null이기에 예시와 같이 사용한다

```
class `7_LateInit` {
    lateinit var text: String

    fun printText() {
        text = "안녕하세요"
        println(text)
    }
}

fun main() {
    val test = `7_LateInit`()
    test.printText()
}
// 안녕하세요
```

- 초기화 전에 사용하면 어떻게 될까?

```
class `7_LateInit` {
    lateinit var text: String

    fun printText() {
        println(text)
        text = "안녕하세요"
    }
}

fun main() {
    val test = `7_LateInit`()
    test.printText()
}
// UninitializedPropertyAccessException
```

- 앞서 언급했던 것처럼 특정 DI와 같이 외부에서 초기화를 해주는 경우를 염두해두고 만든 기능이기 때문에 초기화 전에 사용하더라도 컴파일 오류가 발생하지 않는다

- 초기화 여부를 파악하고 사용하려면 `isInitialized` 프로퍼티를 사용하라

```
class `7_LateInit` {
    lateinit var text: String

    fun printText() {
        if (this::text.isInitialized) {
            println("초기화됨")
            println(text)
        } else {
            text = "안녕하세요"
            println(text)
        }
    }
}

fun main() {
    val test = `7_LateInit`()
    test.printText()
}
```

- isInitialized는 내부에선 사용할 수 있지만 외부에선 사용할 수 없다

```
fun main() {
    val test = `7_LateInit`()
    test.printText()
    test.isInitialized // 컴파일 오류
}
```