Chapter 03.코틀린 고급 - 03.싱글톤과 동반객체

싱글톤과 동반객체

1. 싱글톤

- 싱글톤 패턴은 클래스의 인스턴스를 하나의 `단일` 인스턴스로 제한하는 디자인 패턴이다
- 싱글톤 패턴을 구현할때는 몇가지 제약사항을 통해 구현한다
  - 직접 인스턴스화 하지 못하도록 생성자를 `private` 으로 숨긴다
  - `getInstance()` 라는 클래스의 단일 인스턴스를 반환하는 static 메서드를 제공한다
  - 멀티-스레드 환경에서도 `안전하게` 유일한 인스턴스를 반환해야한다
- 다양한 구현 방법들
  - DCL(Double Check Locking)
    - JVM 환경에선 거의 사용 안함
  - Enum 싱글톤
    - 이펙티브 자바에서 소개
  - 이른 초기화(Eager)
  - 지연 초기화(Lazy)
- 자바에서 많이 쓰이는 구현 방식

- 이른 초기화

```
public class Java_Singleton {

    private static final Java_Singleton INSTANCE = new Java_Singleton();

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

    public Java_Singleton getInstance() {
        return INSTANCE;
    }

}
```

- 지연 초기화

```
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();
    }

}
```

---

2. 코틀린의 싱글톤

- 코틀린은 언어에서 `객체 선언`을 통해 싱글톤을 기본 지원한다
- 객체 선언은 `object` 키워드를 사용한다

```
object Singleton {
}
```

- 함수나 변수를 사용할때는 `클래스 한정자` 를 사용한다 (클래스명.함수명)

```
object Singleton {
    val a = 1234

    fun printA() = println(a)
}

fun main() {
    println(Singleton.a)
    Singleton.printA()
}
```

- 객체 선언을 사용하면 자바의 static 유틸리티를 대신해 쉽게 싱글톤 기반의 유틸리티를 만들 수 있다
- object로 만든 DatetimeUtils

```
import java.time.LocalDateTime

object DatetimeUtils {

    val now: LocalDateTime
        get() = LocalDateTime.now()

    const val DEFAULT_FORMAT = "YYYY-MM-DD"


    fun same(a: LocalDateTime, b: LocalDateTime) :Boolean {
        return a == b
    }
}

fun main() {
  println(DatetimeUtils.now) // 2022-05-23T22:42:52.186117
  println(DatetimeUtils.now) // 2022-05-23T22:42:52.189968
  println(DatetimeUtils.now) // 2022-05-23T22:42:52.190034

  println(DatetimeUtils.DEFAULT_FORMAT) // YYYY-MM-DD

  val now = LocalDateTime.now()
```

```
  println(DatetimeUtils.same(now, now)) // true
}
```

---

3. 동반객체

- `companion` 키워드를 사용해 클래스 내부에 객체 선언을 사용할 수 있다

```
class MyClass {

  companion object {

  }
}
```

- 동반객체의 멤버는 object로 선언한 객체와 마찬가지로 클래스 한정자를 사용해 호출할 수 있다

```
class MyClass {

    private constructor()

    companion object  {
        val a = 1234

        fun newInstance() = MyClass()
    }
}

fun main() {
    println(MyClass.a)
    println(MyClass.newInstance())

    // 이렇게도 가능하나 생략 가능
    println(MyClass.Companion.a)
    println(MyClass.Companion.newInstance())
}
```

- 생성자를 private으로 숨기고 newInstance 함수를 통해서만 객체 생성을 가능하게 함

- 동반객체는 이름을 가질 수 있다

```
class MyClass {

    companion object MyCompanion {
        val a = 1234

        fun newInstance() = MyClass()
    }
}

fun main() {
    // MyCompanion은 생략 가능
    println(MyClass.MyCompanion.a)
    println(MyClass.MyCompanion.newInstance())
}
```