03-1. 클래스와 생성자

Object Oriented Programming

클래스 선언 및 생성

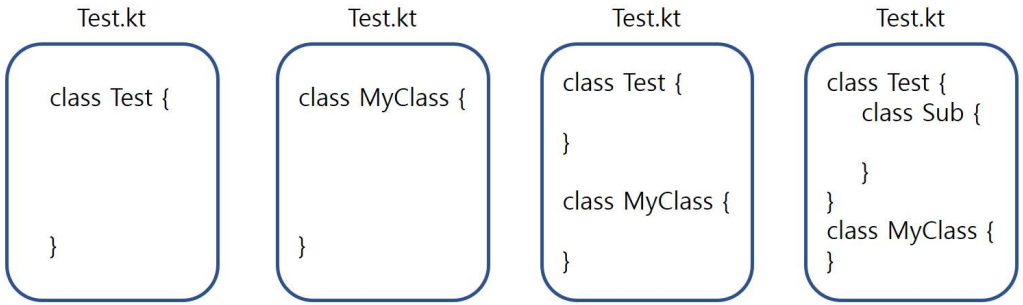
클래스 선언

- class 예약어로 선언
- 클래스 몸체가 없다면 { }는 생략 가능

```
class MyClass { }
```

```
class MyClass
```

- 코틀린 파일에 파일명과 같은 이름의 클래스가 없어도 됩니다.



클래스 선언 및 생성

클래스 선언

- 클래스에는 여러 가지 구성요소를 선언할 수 있는데 프로퍼티, 메서드(함수), 생성자, 클래스 등을 선언할 수 있습니다.

```
class MyClass {
    val myVariable=10

    constructor(){ }

    fun myFun(){ }

    class Inner {

    }
}
```

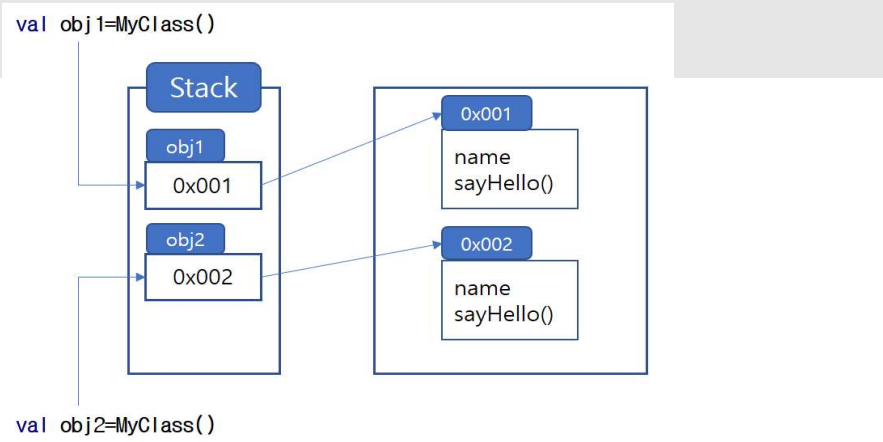
클래스 선언 및 생성

객체 생성

```
class MyClass {
    var name: String="world"

    fun sayHello(){
        println("hello $name")
    }
}

fun main(args: Array<String>) {
    val obj1=MyClass()
    val obj2=MyClass()
}
```



생성자

주 생성자

- 코틀린에서는 생성자를 주 생성자와 보조 생성자로 구분합니다.
- 주 생성자(Primary Constructor)는 클래스 선언부분에 작성
- 하나의 클래스에 하나의 주 생성자만 정의 가능
- 필수 작성 대상은 아니며 보조 생성자가 있다면 작성 안될수도 있음.

```
class MyClass1 { }

class MyClass2() { }

class MyClass3 constructor() { }

fun main(args: Array<String>) {
    val obj1=MyClass1()
    val obj2=MyClass2()
    val obj3=MyClass3()
}
```

생성자

주 생성자

- 매개변수를 가지는 주생성자

```
class User1 constructor(name: String, age: Int){ }
class User2(name: String, age: Int){ }
//.................
val user1=User1()//error
val user2=User1("kkang", 33)
val user3=User2("kim", 28)
```

- 생성자 매개변수 기본값 명시

```
class User3(name: String, age: Int = 0) { }
//.................
val user4=User3("kkang", 33)
val user5=User3("kkang")
```

생성자

주 생성자

- 생성자 초기화 블록(init)
- 주 생성자는 클래스 헤더에 명시하다 보니 실행문을 포함할 수 없다.
- 즉, 주 생성자는 { }를 가질 수 없습니다

```
class User4(name: String, age: Int) { }{ //error

}
```

- 주 생성자는 클래스 선언 영역에 작성하다 보니 실행 영역 init 예약어로 따로 명시하는 기법을 사용합니다.

```
class User4(name: String, age: Int) {
    init {
        println("i am init...")
    }
}
//.....................
val user6=User4("kkang", 33)
```

생성자

생성자 매개변수 값 이용

- 클래스의 초기화 블록, 클래스 프로퍼티에서는 접근이 되지만 클래스에 정의된 함수에서는 사용이 불가능

```
class User5(name: String, age: Int){
    init {
        println("i am init... constructor argument : $name .. $age")
    }
    val upperName=name.toUpperCase()

    fun sayHello(){
        println("hello $name")//error
    }
}
```

생성자

생성자 매개변수 값 이용

- 생성자 내에서 val, var 을 이용해 매개변수를 선언
- 생성자의 매개변수를 선언할 때 val 또는 var를 추가하면 해당 매개변수는 클래스의 프로퍼티가 된다.

```
class User6(val name: String, val age: Int){
    val myName=name
    init {
        println("i am init... constructor argument : $name .. ${age}")
    }
    fun sayHello() {
        println("hello $name")
    }
}
```

생성자

보조 생성자

- 보조 생성자는 클래스 바디 영역에 constructor 예약어로 선언하는 생성자.
- 만약 보조 생성자를 선언했으면 주생성자는 선언하지 않아도 됩니다.
- 주 생성자든 보조 생성자든 개발자가 생성자를 추가하면 컴파일러는 주 생성자를 자동으로 추가하지 않습니다.
- 생성자는 클래스 선언시 최소한 하나 이상 정의되어 있어야 한다.

```
//컴파일러에 의해 매개변수 없는 주 생성자 자동 추가
class User1 { }

//주생성자만 선언
class User2(name: String) { }

//보조 생성자만 선언
class User3 {
    constructor(name: String){ }
}

fun main(args: Array<String>) {
    val user1=User1()
    val user2=User2("kkang")
//    val user3=User3()//error
    val user4=User3("kkang")
}
```

생성자

생성자 오버로딩

- 생성자 오버로딩(constructor overloading)은 하나의 클래스 내에 여러 개의 생성자를 선언한 상황을 말합니다.
- 객체지향 프로그램에서 흔히 오버로딩은 같은 이름의 함수를 여러 개 선언하는데 단지 매개변수 부분(개수, 타입)만 다르게 선언하는 기법을 말합니다.

```
class User4 {
    constructor(){}
    constructor(name: String){}
    constructor(name: String, age: Int){}
}//.....val user5=User4()
val user6=User4("kkang")
val user7=User4("kkang", 10)
```

생성자

보조 생성자 초기화 블록

- init로 정의한 초기화 블록은 주 생성자든 보조 생성자든 객체 생성 때 가장 먼저 실행됩니다.

```
class User5 {
    init {
        println("i am init block....")
    }
    constructor(){
        println("i am constructor....")
    }
}
//.....................
val user8=User5()
```

생성자

보조생성자의 매개변수 사용

- 보조 생성자의 매개변수를 선언할 때 주 생성자처럼 var나 val을 추가해서 선언할 수 없다.

```
class User6 {
    init {
        println("i am init block....$name")//error
    }
    constructor(name: String){
        println("i am constructor....$name")
    }
    fun sayHello(){
        println("hello $name")//error
    }
}
```

```
class User6 {
    constructor(val name: String){//error
        println("i am constructor....$name")
    }
}
```

생성자

this() 에 의한 생성자 연결

- 주생성자가 선언되어 있다면 보조생성자는 무조건 주 생성자를 같이 호출해 주어야 한다.

```
class User1(name: String){

    constructor(name: String, age: Int){//error

    }
}
```

```
//주생성자와 보조생성자 같이 선언
class User1(name: String){
    init {
        println("init block... $name")
    }
    constructor(name: String, age: Int): this(name){
        println("constructor ... $name ... $age")
    }
}

fun main(args: Array<String>) {
    println("----- 주생성자로 생성한 경우 -------")
    val user1=User1("kkang")
    println("----- 보조생성자로 생성한 경우 -------")
    val user2=User1("kkang", 33)
}
```

생성자

---

this() 에 의한 생성자 연결

- 보조 생성자를 여러 개 선언하면 this ( )로 다른 보조 생성자를 호출할 수는 있지만, 최종 주 생성자
- 는 호출해 주어야 합니다.

```
//보조생성자 여러개의 경우
class User2(name: String){
    constructor(name: String, age: Int): this(name){
    }
    constructor(name: String, age: Int, email: String): this(name, age) {
    }
}

fun main(args: Array<String>) {
    //보조생성자 여러개선언된 경우
   val user3=User2("kkang")
    val user4=User2("kkang", 30)
    val user5=User2("kkang", 30, "a@a.com")
}
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare