객체지향과 클래스 1

코틀린 기초강좌

강사: 배정만

학습요약

- 학습목표: 객체지향과 클래스에 대한 기초 학습
- 핵심키워드: Object-oriented, 객체지향, 클래스, 상속, 인터페이스, Abstract, Open
- 학습내용: 객체지향의 기본 개념과 클래스, 상속에 대한 기본 개념 학습과 사용법을 배웁니다.

객체 지향 프로그래밍

OOP, Object-Oriented Programming

객체 지향 프로그래밍에서는 모든 데이터를 객체(object)로 취급하며, 이러한 객체가 바로 프로그래밍의 중심이 됩니다.

클래스(class)

클래스(class)란 객체를 정의하는 틀 또는 설계도와 같은 의미로 사용됩니다.

클래스는 객체의 상태를 나타내는 프로퍼티(properties)와
객체의 행동을 나타내는 함수(function)로 구성됩니다.

인스턴스(instance)

클래스(틀)로 부터 만들어진 객체
메모리에 할당된다.
컨스트럭터(constructor)를 이용해
이니셜라이즈(initialize) 한다

멤버 함수 = 메소드

클래스 멤버 함수(function)란
어떠한 특정 작업을 수행하기 위한
클래스 함수 표현식의 집합

상속 (Inheritance)

Kotlin의 모든 클래스는 공통 수퍼 클래스 Any를 가집니다.

Any는 java.lang.Object가 아닙니다.
특히 equals (), hashCode () 및 toString () 이외의 멤버는 없습니다.

Superclass "Any"

Kotlin의 모든 클래스는 공통 수퍼 클래스 Any를 가집니다.

Any는 java.lang.Object가 아닙니다. 특히 equals (), hashCode () 및 toString () 이외의 멤버는 없습니다.

클래스

```
class Invoice { ... }
```

- 클래스 선언은 클래스 이름, 클래스 헤더 (타입 파라미터, 기본 생성자 등 지정) 및 중괄호로 묶인 클래스 본문으로 구성됩니다.
- 클래스에 본문이 없으면 중괄호를 생략 할 수 있습니다.

생성자(Constructor)

```
class Person constructor(firstName: String) { ... }

class Person(firstName: String) { ... }

class Rectangle(val height: Int, val width: Int) {

    //secondary constructors
    constructor(side: Int) : this(side, side)
}
```

- Kotlin의 클래스는 기본 생성자와 하나 이상의 보조 생성자를 가질 수 있습니다.
- 기본 생성자는 클래스 헤더의 일부입니다.
- 클래스 이름을 따라갑니다.

클래스 프로퍼티

```
class Address {
    var name: String = ...
    var street: String = ...
    var city: String = ...
    var state: String? = ...
    var zip: String = ...
}


val addr = Address()
addr.name = “adsfasfaf”
```

- Kotlin의 클래스는 속성을 가질 수 있습니다.
- var 키워드를 사용해 뮤터블로 선언 가능
- val 키워드를 사용하여 읽기 전용으로 선언
- 프로퍼티를 사용하려면 Java에서 필드 인 것처럼 이름으로 참조

접근 제어 (Visibility Modifiers)

```
// 파일이름: example.kt
package foo

private fun foo() { ... } // example.kt 안에서만 보임

public var bar: Int = 5 // 전제공개 프로퍼티
    private set         // example.kt 안에서만 보임

internal val baz = 6    // 같은 모듈에서만 보임
```

- public이 기본적으로 사용됩니다.
- private은 정의된 파일 내부에서만 볼 수 있습니다.
- internal은 동일한 모듈의 모든 곳에서 볼 수 있습니다.
- protected는 최상위 선언에는 사용할 수 없습니다.

모듈 (Modules)

모듈은 함께 컴파일 된 Kotlin 파일 세트
IntelliJ IDEA module, Maven project,
a Gradle source set, ...

클래스 인스턴스 비교 연산자

```
val set1 = setOf(1, 2, 3)
val set2 = setOf(1, 2, 3)

println(set1 === set2)
println(set1 == set2)
```

상속

```
open class Person {
    open fun validate() {

    }
}

open class Customer: Person {
    final override fun validate() {

    }

    constructor(): super() {


    }
}
```

추상화 클래스

```
abstract class StoredEntity {
    val isActive = true
    abstract fun store()
    fun status(): String {
        return isActive.toString()
    }

}

class Employee : StoredEntity() {
// Error!
}
```

상호 운용성 (Interop)

- Kotlin은 Java 상호 운용성을 염두에두고 설계되었습니다.
- 기존 Java 코드는 Kotlin에서 자연스럽게 호출 할 수 있으며 Java에서 Kotlin 코드를 사용할 수도 있습니다.

Java 에서 Kotlin

```
// class name String.kt
fun merge(a: String, b: String) = "$a $b"

public class Sample {
    @Test
    public void sample() {
      System.out.println("merge " + StringKt.merge("A",
"B"));
    }
}
```

- Java에서 호출할 때는 Kt이라는 이름이 붙은 상태로 호출합니다.

주의 사항!

```
public class Sample {
  private String a = "A";
  private String b = "B";

  @Before
  public void setUp() {
    a = null;
  }

  @Test
  public void sample() {
    System.out.println("merge " + StringKt.merge(a, b));
  }
}
```