33장. 코틀린 OOP 프로그램

33.1 클래스 정의

33.1.1. 클래스 선언 및 생성

```
class MyClass { }
```

```
val obj = MyClass()
```

33.1.2. 생성자

- 생성자는 constructor라는 예약어로 선언
- 주 생성자(Primary Constructor)와 보조 생성자(Secondary Constructor)로 구분
- 주 생성자는 클래스 몸체가 아닌 클래스 헤더에 선언

```
class MyClass constructor() //class MyClass와 동일
```

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

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

- 주 생성자의 실행 부분을 클래스 몸체 영역에 init라는 키워드로 따로 명시 가능

```
class Person(firstName: String) {
init {
println("primary constructor... $firstName")
}
}
```

33.1 클래스 정의

- 생성자의 매개변수를 선언하면서 var, val 예약어로 선언하면 그 자체가 클래스의 프로퍼티가 된다.

```
class Person(val firstName: String) {
init {
println("primary constructor... $firstName")
}
val upperName=firstName.toUpperCase() // ok~~~~~~~
fun someFun(){
println("someFun().... $firstName") //ok~~~~
}
}
```

- 클래스 몸체 영역에 constructor 예약어로 보조 생성자를 정의

```
class User {
init {
println("init block.....")
}
constructor(name: String,email: String){
printLog("constructor(name: $name ,email: $email)")
}
}
```

33.1 클래스 정의

- 매개변수 부분을 다르게 하여 constructor 예약어로 여러 개의 생성자를 정의

```
class User {
init {
printLog("init block.....")
}
constructor(name: String,email: String){
printLog("constructor(name: $name ,email: $email)")
}
constructor(name: String, email: String, age: Int){
printLog("constructor(name: $name ,email: $email , age: $age)")
}
}
```

- 생성자를 선언하는 부분에 다른 생성자와의 연결을 명시

```
class User {
init {
printLog("init block.....")
}
constructor(name: String, email: String){
printLog("constructor(name: $name ,email: $email)")
}
constructor(name: String, email: String, age: Int): this(name, email){
printLog("constructor(name: $name ,email: $email , age: $age)")
}
}
```

33.2 상속

33.2.1. 상속 관계 명시

- 상위 클래스를 명시할 때는 클래스 헤더 영역에 콜론(:)을 사용
- 클래스는 기본으로 final로 선언되므로 다른 클래스에서 상속 불가
- 클래스를 상속받아 하위 클래스를 만들도록 허용하려면 open 예약어를 생성자 선언 부분에 명시 해야함.

```
open class User
class Customer: User()
```

- 상속받는 하위 클래스를 선언할 때는 상위 생성자 정보에 맞게 호출 구문을 작성
- 상위 클래스에 주 생성자와 보조 생성자가 함께 선언되어 있을 때는 하위 클래스에서 상위 클래스의 생성자 중 하나와 일치하도록 생성자 호출 구문을 작성

```
open class User(name:String) {
constructor(name: String, email: String):this(name)
}
class Customer: User("kkang", "a@a.com")
class Customer2: User("kkang")
```

33.2 상속

33.2.2. 함수 오버라이딩

- 상위 클래스에서 open 예약어를 함수에 선언
- 하위 클래스에서는 override 예약어를 사용

```
open class User(name:String) {
constructor(name: String, email: String):this(name)
open fun someFun(){}
}
class Customer: User("kkang"){
override fun someFun() {}
}
```

33.2.3. 변수 오버라이딩

- 상위 클래스에서 변수의 오버라이드를 허용하고자 한다면, open을 명시적으로 선언
- 하위 클래스에서는 오버라이드 변수 앞에 override를 명시적으로 이용
- val로 선언된 상위 클래스의 변수를 하위 클래스에서 var로 선언해서 사용하는 건 가능
- var로 선언된 상위 클래스의 변수를 하위 클래스에서 val로 선언해서 사용하는 건 불가

33.2 상속

```
open class User(name:String) {
constructor(name: String, email: String):this(name)
open val x: Int = 10
open fun someFun(){}
}
open class Customer: User("kkang") {
override val x: Int = 20
override fun someFun() {}
}
```

33.3 추상 클래스와 인터페이스

33.3.1. 추상 클래스

- 함수에 구현 부분 { }이 없거나 선언과 동시에 초기화되지 않는 변수는 abstract로 선언
- abstract로 선언된 함수나 변수를 가지는 클래스도 abstract로 선언
- 하위 클래스는 상위 클래스에 abstract로 선언된 함수나 변수를 꼭 재정의

```
abstract class Base {
abstract val name: String
abstract fun someFun()
}
class SubBase : Base(){
override val name="kkang"
override fun someFun() { }
}
```

33.3.2. 인터페이스

```
interface MyInterface {
fun bar()
fun foo() {
//...
}
}
```

33.3 추상 클래스와 인터페이스

```
class Child : MyInterface {
override fun bar() { }
}
```

33.4 프로퍼티

- var, val로 선언되는 변수들이 프로퍼티
- 변수 데이터를 이용하기 위한 getter/setter 함수가 선언된 변수

var <propertyName>[: <PropertyType>] [= <property_initializer>]
[<getter>]
[<setter>]

```
class User {
var greeting: String="Hello"
set(value) {
field=value
}
get() = field
val name:String ="Kim"
get() = field
}
```

- set ( ), get ( ) 내부에서 사용한 field 예약어는 프로퍼티에 대입된 데이터를 지칭

33.5 접근제한자

- private, protected, internal, public 4개를 지원

• 33.5.1. 패키지 접근 제한

- 접근 제한자를 선언하지 않으면 기본값은 public이며 어느 곳에서나 접근 가능
- private으로 선언하면 같은 파일 내에서만 접근 가능
- internal로 선언하면 모듈 내에 어디서나 접근 가능
- protected로 선언하면 최상위 레벨로는 선언 불가

```
private fun foo() {}
public var bar: Int = 5
private set
internal var baz = 6
```

33.5.2 _ 클래스 접근 제한

- **private**: 클래스 내부에서만 사용 가능
- **protected**: private + 서브 클래스에서 사용 가능
- **internal**: 같은 모듈에 선언된 클래스에서 사용 가능
- **public**: 어디서든 사용 가능

Stey by Step 33-1 – 코틀린 OOP 문법 테스트

코틀린으로 개발하기

1. 모듈 생성
2. 파일 복사
3. InheritanceTest 파일 작성
4. PropertyTest 파일 작성
5. MainActivity.kt 작성
6. 모듈 실행