

객체지향, class

OOP, class

객체 지향

- 숫자를 저장할 때 → Int 형
- 실수를 저장할 때 → Float 형, Double 형
- 문자를 저장할 때 → Char형
- 2차원 좌표를 저장할 때 → ???형
- 시간 정보를 저장할 때 → ???형
- 인물 정보를 저장할 때 → ???형

직접 만들어 준다.

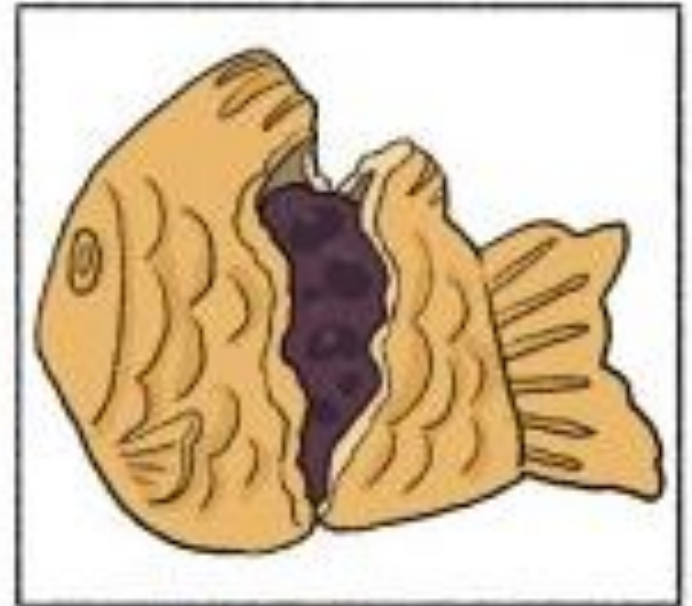
객체 지향

- 하나의 자료형은 class로 만든다.
- class는 멤버 변수와 메서드로 이루어져있다.
- class를 사용해서 제작한게 객체 이다.

클래스



객체



class

```
class 시간(){  
    var 시 : Int? = null  
    var 분 : Int? = null  
    var 초 : Int? = null  
}
```

시간 정보를 저장하는 클래스

클래스의 모든 멤버 변수는 값이 할당 되어있어야 한다.

```
fun main() {  
    val 등교시간 = 시간()  
    등교시간.시 = 8  
    등교시간.분 = 30  
    등교시간.초 = 0  
  
    val 점심시간_시작시간 = 시간()  
    점심시간_시작시간.시 = 11  
    점심시간_시작시간.분 = 30  
    점심시간_시작시간.초 = 0  
}
```

class

class 키워드 사용

```
class 시간() { 클래스 이름
    var 시 : Int? = null
    var 분 : Int? = null
    var 초 : Int? = null
}
```

멤버 변수들

시간 정보를 저장하는 클래스

```
fun main() {
```

```
    val 등교시간 = 시간() 객체 만들기
```

```
    등교시간.시 = 8
```

```
    등교시간.분 = 30
```

```
    등교시간.초 = 0 객체 멤버 변수에 접근 하기
```

```
    val 점심시간_시작시간 = 시간()
```

```
    점심시간_시작시간.시 = 11
```

```
    점심시간_시작시간.분 = 30
```

```
    점심시간_시작시간.초 = 0
```

```
}
```

class

시 : Int
분 : Int
초 : Int

시간 클래스

시 : Int = 8
분 : Int = 30
초 : Int = 0

등교시간


시 : Int = 11
분 : Int = 30
초 : Int = 0

점심시간

시 : Int = 8ull
분 : Int = 40ull
초 : Int = 0ull

1교시 시작 시간

class



```
1 class 시간( ){
2     var 시 : Int? = null,
3     var 초 : Int? = null,
4     var 분 : Int? = null,
5 }
6
7 fun main() {
8     val 기상_시간 = 시간( )
9
10    기상_시간.시 = 6
11    기상_시간.분 = 30
12    기상_시간.초 = 8
13
14    println("${기상_시간.시}")
15 }
```

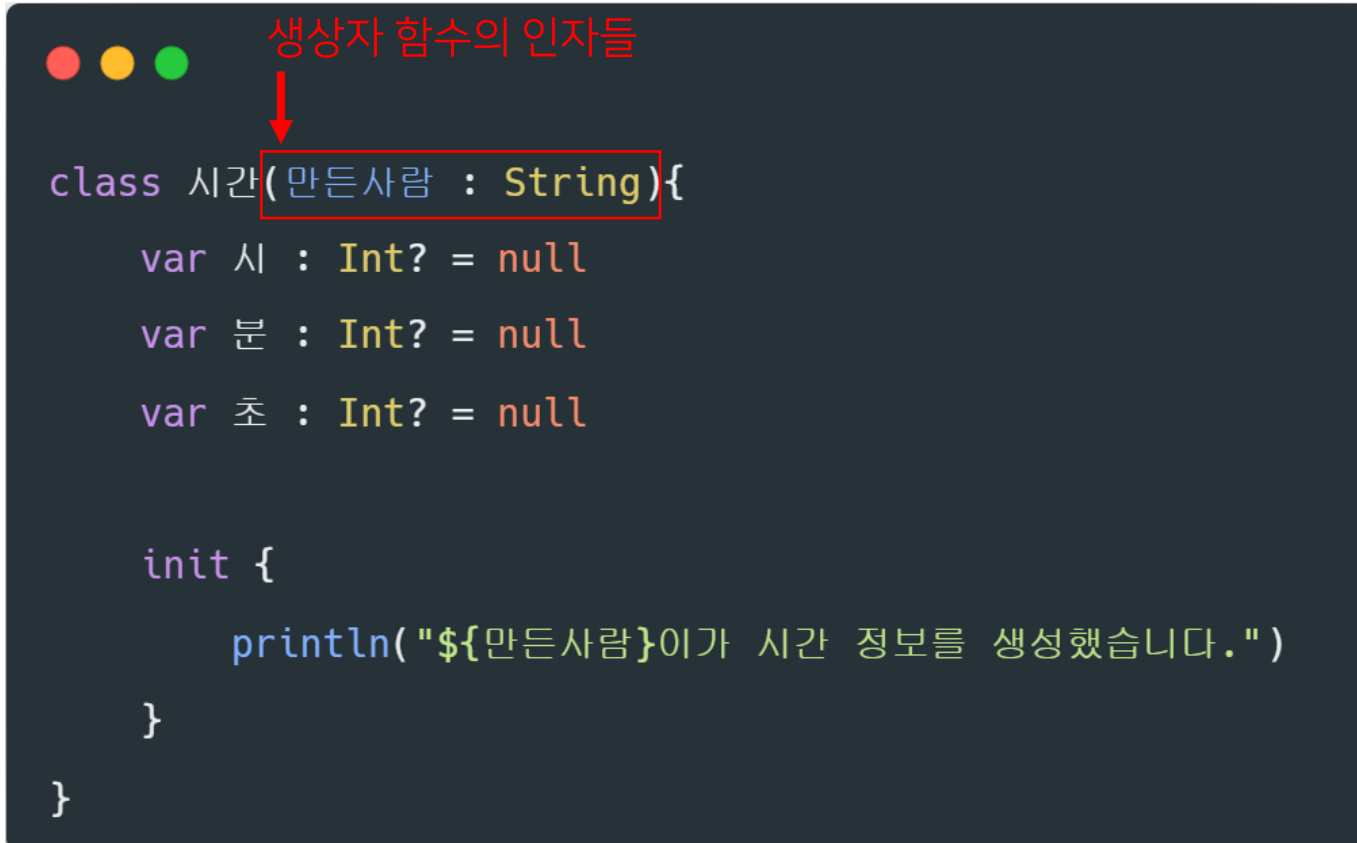
class 생성자

- 생성자 : 객체를 생성할 때마다 실행되는 함수
- init 키워드를 사용해서 만들어준다.
- 객체의 값을 채워 넣을 때 사용한다

```
class 시간(만든사람 : String){  
    var 시 : Int? = null  
    var 분 : Int? = null  
    var 초 : Int? = null  
  
    init {  
        println("${만든사람}이가 시간 정보를 생성했습니다.")  
    }  
}
```


class 생성자

- 생성자 : 객체를 생성할 때마다 실행되는 함수
- init 키워드를 사용해서 만들어준다.
- 객체의 값을 채워 넣을 때 사용한다



```
class 시간(만든사람 : String){  
    var 시 : Int? = null  
    var 분 : Int? = null  
    var 초 : Int? = null  
  
    init {  
        println("${만든사람}이가 시간 정보를 생성했습니다.")  
    }  
}
```

class 생성자




```
fun main() {  
    val 점심시간 = 시간("박희찬")  
    // 박희찬(이)가 시간 정보를 생성했습니다.  
  
    val 등교시간 = 시간("장인수")  
    // 장인수(이)가 시간 정보를 생성했습니다.  
}
```

class 생성자


```
class 시간(val 시 : Int, val 분 : Int, val 초 : Int){  
  
}  
  
fun main() {  
    val 점심시간 = 시간(11, 30, 0)  
    점심시간.시 // 11  
  
    val 등교시간 = 시간(8, 30, 0)  
    등교시간.분 //30  
}
```

class 생성자




```
class 시간(val 시 : Int, val 분 : Int, val 초 : Int){  
    init {  
        println("${시}시 ${분}분 ${초}초인 시간정보를 생성했습니다.")  
    }  
}  
  
fun main() {  
    val 등교시간 = 시간(8, 30, 0)  
    val 점심시간 = 시간(11, 30, 0)  
}
```

객체를 생성하면서 동시에
멤버 변수에 값을 채워 넣어준다.



```
class 시간(val 시 : Int, val 분 : Int, val 초 : Int, 만든사람 : String){  
    init {  
        println("${만든사람}(이)가 ${시}시 ${분}분 ${초}초를 생성했습니다.")  
    }  
}
```

```
fun main() {  
    val 등교시간 = 시간(8, 30, 0, "박희찬")  
    등교시간.시 //8  
    등교시간.분 //30  
    등교시간.초 //0  
    등교시간.만든사람  
}
```



```
class 시간(val 시 : Int, val 분 : Int, val 초 : Int, 만든사람 : String){  
    init {  
        println("${madePerson}(이)가 ${시}시 ${분}분 ${초}초를 생성했습니다.")  
    }  
}
```

멤버변수 0

멤버변수 X

```
fun main() {  
    val 등교시간 = 시간(8, 30, 0, "박희찬")  
    등교시간.시 //8  
    등교시간.분 //30  
    등교시간.초 //0  
    등교시간.만든사람 에러  
}
```

class 메서드

- 메서드 : 클래스가 가지고 있는 함수

```
class 시간(val 시 : Int, val 분 : Int, val 초 : Int){  
    fun 모두_초로_바꿔주는_함수() : Int {  
        return 시 * 60 * 60 + 분 * 60 + 초  
    }  
}  
  
fun main() {  
    val 등교시간 = 시간(8, 30, 0)  
    print(등교시간.모두_초로_바꿔주는_함수()) //30600  
}
```

클래스를 사용 하는 이유

1. 새로운 자료형을 만들어 준다.
2. 변수와 함수를 하나의 변수에 담아줄 수 있다.
3. 현실 세계의 코드로 객체를 표현 할 수 있다.

모든 자료형은 class로 이루어져있다

- 숫자를 저장할 때 → Int 형
- 실수를 저장할 때 → Float 형, Double 형
- 문자를 저장할 때 → Char형

원시 자료형

- 2차원 좌표를 저장할 때 → ???형
- 시간 정보를 저장할 때 → ???형
- 인물 정보를 저장할 때 → ???형

참조 자료형

모든 자료형은 class로 이루어져있다

- 코틀린 내부적으로 모든 자료형이 class로 만들어져있다.

```
508 public class Int private constructor() : Number(), Comparable<Int> {  
509     companion object {  
        | A constant holding the minimum value an instance of Int can have.  
513     public const val MIN_VALUE: Int = -2147483648  
514  
        | A constant holding the maximum value an instance of Int can have.  
518     public const val MAX_VALUE: Int = 2147483647  
519  
        | The number of bytes used to represent an instance of Int in a binary form.  
523     @SinceKotlin(version: "1.3")  
524     public const val SIZE_BYTES: Int = 4  
525  
        | The number of bits used to represent an instance of Int in a binary form.  
529     @SinceKotlin(version: "1.3")  
530     public const val SIZE_BITS: Int = 32  
531 }
```

```
23 public class String : Comparable<String>, CharSequence {  
24     companion object {}  
25  
        | Returns a string obtained by concatenating this string with the string representation of the given  
        | other object.  
29     public operator fun plus(other: Any?): String  
30  
31     public override val length: Int  
32  
        | Returns the character of this string at the specified index.  
        | If the index is out of bounds of this string, throws an IndexOutOfBoundsException except in  
        | Kotlin/JS where the behavior is unspecified.  
39     public override fun get(index: Int): Char
```

변수와 함수를 하나의 변수에 담아 두었다

student1

name : String = "박희찬"

age : Int = 18

school : String = "선린"

club : String = "EDCAN"

score : Int = 0

hi()

study()


goSchool()

walk()


talk()

```
class Student(  
    val name : String,  
    val age : Int,  
    var school : String,  
    val club : String,  
) {  
    var score = 0  
  
    fun hi() { ... }  
    fun study() { ... }  
    fun goSchool() { ... }  
    fun walk() { ... }  
    fun talk() { ... }  
}  
  
fun main() {  
    val student1 = Student("박희찬", 18, "선린인터넷고등학교", "EDCAN")  
}
```

변수와 함수를 하나의 변수에 담아 두었다




```
fun studentInfo(  
    name : String,  
    age : Int,  
    school : String,  
    club : String,  
    score : Int,  
) {  
    ...  
}
```



```
fun studentInfo(  
    studentData : Student  
) {  
    ...  
}
```

class - lateinit


다음 방식의 문제점 : school 변수가 nullable 이다.



```
class Person(public val name : String, public val age : Int){  
    var school : String? = null  
}  
  
fun main() {  
    val 장인수 = Person("장인수", 18)  
    장인수.school = "선린인터넷고등학교"  
}
```

class - lateinit


다음 방식의 문제점 : 클래스의 멤버 변수를 선언 한 뒤 반드시 값이 할당 되어야 한다.



```
class Person(public val name : String, public val age : Int){  
    var school : String 에러 발생  
}  
  
fun main() {  
    val 장인수 = Person("장인수", 18)  
    장인수.school = "선린인터넷고등학교"  
}
```

class - lateinit

멤버 변수를 선언할 때 lateinit을 붙여주면 나중에 값을 할당할 수 있다.



```
class Person(public val name : String, public val age : Int){
    lateinit var school : String
}

fun main() {
    val 장인수 = Person("장인수", 18)
    장인수.school = "선린인터넷고등학교"
}
```

class 접근 제한자

- 접근 제한자 : 클래스의 멤버 변수와 메서드에 접근을 제한한다.

```
class Person(public val name : String, public val age : Int){  
    private val height : Float? = null  
    private val weight : Float? = null  
}  
  
fun main() {  
    val p1 = Person("박희찬", 18)  
    p1.name // "박희찬"  
    p1.age // 18  
  
    p1.height 에러  
    p1.weight 에러  
}
```


class 접근 제한자

- 접근 제한자 : 클래스의 멤버 변수와 메서드에 접근을 제한한다.

접근 제한자	제한 범위
public (기본값)	class 내부 가능 class 외부 가능
private	class 내부 가능 class 외부 불가능

클래스를 사용 하는 이유

1. 새로운 자료형을 만들어 준다.
2. 변수와 함수를 하나의 변수에 담아줄 수 있다.
3. 현실 세계의 코드로 객체를 표현 할 수 있다.

현실 세계의 객체를 코틀린으로 표현

- 객체 : 클래스를 사용해서 만든 것,
우리가 실생활에서 사용하는 모든 것,
- 객체 지향 : 소프트웨어를 객체 들로 이루어서 만든것

현실 세계의 객체를 코틀린으로 표현



학생의 추상화

값

- 이름
- 나이
- 키
- 몸무게
- 학교
- 점수
- 동아리

기능

- 인사 하기
- 공부 하기
- 등교 하기
- 걷기
- 말하기

클래스의 멤버 변수

클래스의 메소드

추상화 : 같은 객체들이 가지는 공통된 값과 기능을 정의한 것

학생을 추상화 한 정보로
클래스를 만든 것

값

- 이름
- 나이
- 키
- 몸무게
- 학교
- 점수
- 동아리

기능

- 인사 하기
- 공부 하기
- 등교 하기
- 걷기
- 말하기

```
class Student(  
    val name : String, 이름  
    val age : Int, 나이  
    var height : Float, 키  
    var weight : Float, 몸무게  
    var school : String, 학교  
    val club : String, 동아리  
) {  
    var score = 0  
  
    fun hi() = println("$name : 안녕하세요.")  
    fun study() {  
        println("$name : 공부 공부")  
        score += 10  
    }  
  
    fun goSchool() = println("$name : 등교 등교")  
    fun walk() = println("$name : 걷기")  
    fun talk() = println("$name : 저는 ${school}에 다니는 ${age}살 ${name}입니다.")  
}
```

현실 세계의 객체를 코틀린으로 표현

- 같은 객체들이 가지는 공통된 값과 기능을 정의한 것



선생님의 추상화

값 (멤버변수)

- 이름
- 나이
- 과목
- 담당 학년
- 교무실

기능 (메서드)

- 자기 소개
- 수업하기
- 문제 내기

추상화

```
class Teacher(  
    val name : String,      이름  
    val age : Int,          나이  
    val subject : String,   과목  
    val chargeOfGrade : Int 담당 학년  
) {  
    lateinit var teacherRoom : String 교무실  
    var quizCount = 0; 문제 수  
  
    fun info(){  
        println("이름 : $name")  
        println("나이 : $age")  
        println("담당 과목 : $subject ${chargeOfGrade}학년")  
        println("교무실 : $teacherRoom")  
    }  
  
    fun lectures(time : Int) = println("${name}선생님(${subject})이 ${time}교시 수업을 합니다.")  
  
    fun quiz(difficulty : String) = println("$subject ${++quizCount}번 문제 (난이도 : ${difficulty})")  
}
```

클래스 예제

다음 중 1개 선택해서 클래스로 구현하기

[자동차, 3차원 좌표, 사람, 마인크래프트 스티브, 시계, 선린 학생]

조건 : - 멤버 변수 5개 이상 사용
- 메서드 3개 이상 사용

클래스 과제

[마인크래프트를 코틀린으로 구현해보기]

조건

- 좀비, 스켈레톤, 크리퍼, 주민, 플레이어 구현
- 클래스에 멤버 변수와 메소드 각각 3개 이상 사용

기한

5월 28일 (토) 자정

클래스 이해 못했다면...?

1. 이 유튜브 영상 참고



<https://youtu.be/cg1xvFy1JQQ>

클래스 이해 못했다면...?

2. Kotlin 강의 수강

— 섹션 2. Kotlin 객체 지향 프로그래밍		13 강의 ⌚ 206 : 37
▶ 12강 객체지향 프로그래밍		⌚ 14 : 26
▶ 13강 생성자		⌚ 18 : 16
▶ 14강 상속		⌚ 13 : 03
▶ 15강 패키지		⌚ 12 : 59
▶ 16강 모듈		⌚ 08 : 44
▶ 17강 접근제한자		⌚ 28 : 32
▶ 18강 Property		⌚ 18 : 43
▶ 19강 지연초기화		⌚ 14 : 16
▶ 20강 Overriding		⌚ 22 : 54
▶ 21강 Any		⌚ 08 : 38
▶ 22강 this와 super		⌚ 17 : 26
▶ 23강 추상클래스		⌚ 12 : 00
▶ 24강 인터페이스		⌚ 16 : 40

들어야 하는 강의들