

EDCAN 수업 자료

Kotlin 2

이 수업에서 배우는 것들

함수, 람다, 객체지향,

class, 상속, 접근 제한자

추상 클래스, 인터페이스, 데이터 클래스

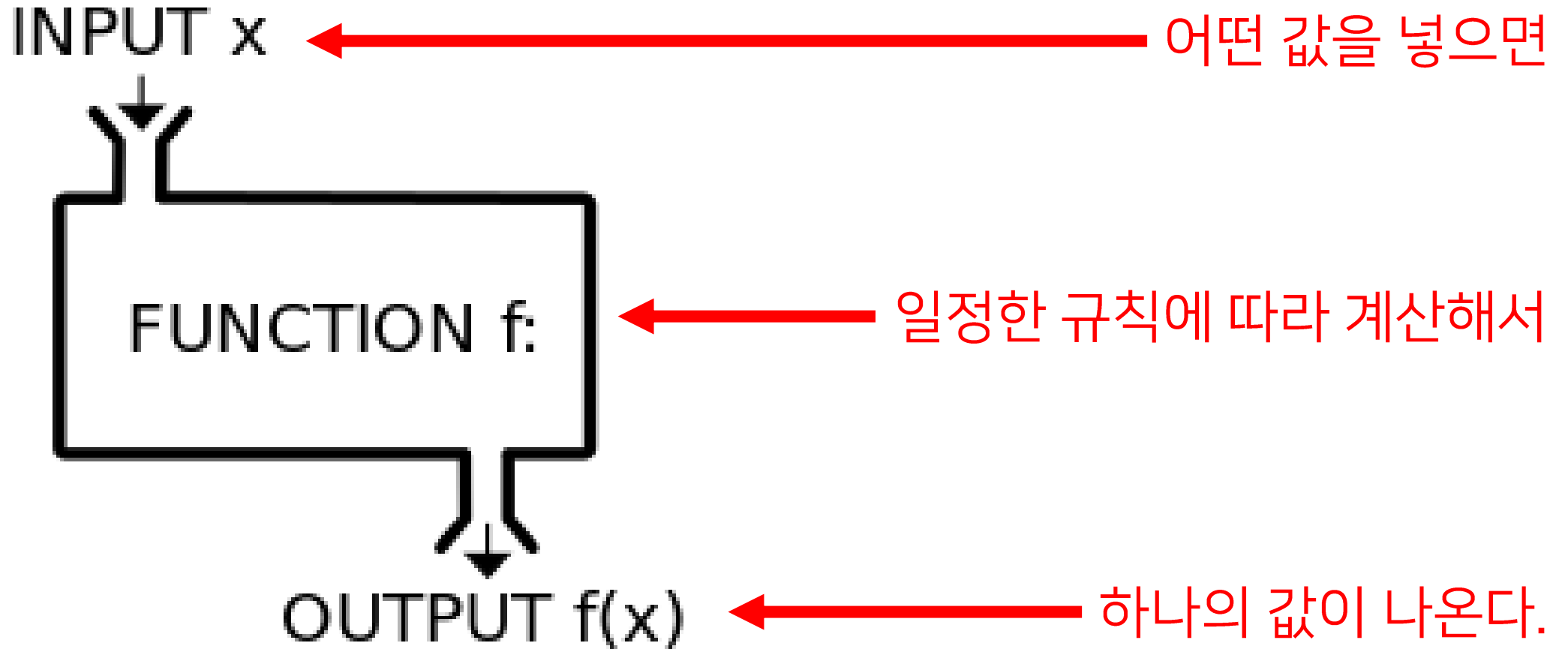
함수

function

함수란?

어떤 값을 넣으면
일정한 규칙에 따라 계산해서
하나의 값이 나온다.

함수란?



함수란?

$$f(x) = x^2 + 2x + 7$$

$$f(1) = 10$$

$$f(0) = 7$$

코틀린에서 함수란?

어떤 값을 넣으면
일정한 규칙에 따라 계산해서
하나의 값이 나온다.

하나의 기능을 만들어 준다.

- 문자열을 출력하는 기능
- 입력을 받는 기능
- 자료형을 변경하는 기능
- etc...

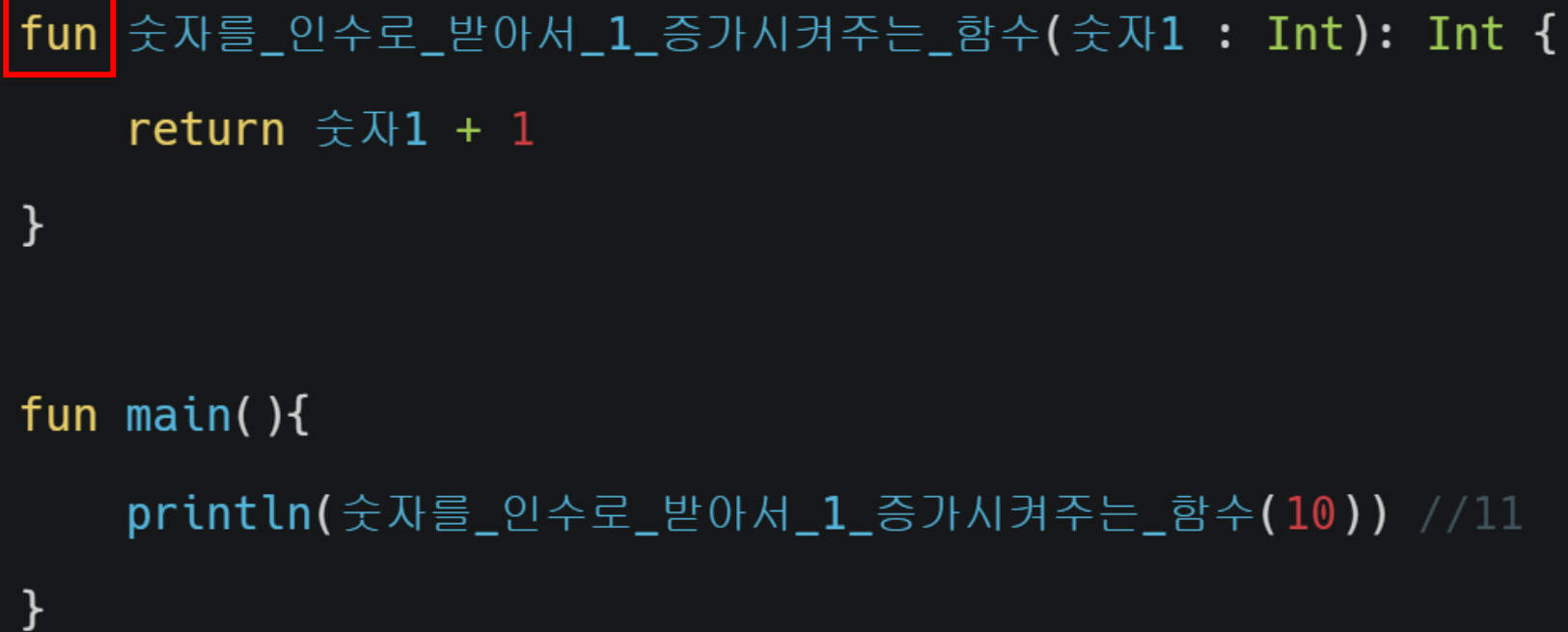
코틀린에서 함수란?



```
fun 숫자를_인수로_받아서_1_증가시켜주는_함수( 숫자1 : Int): Int {  
    return 숫자1 + 1  
}
```

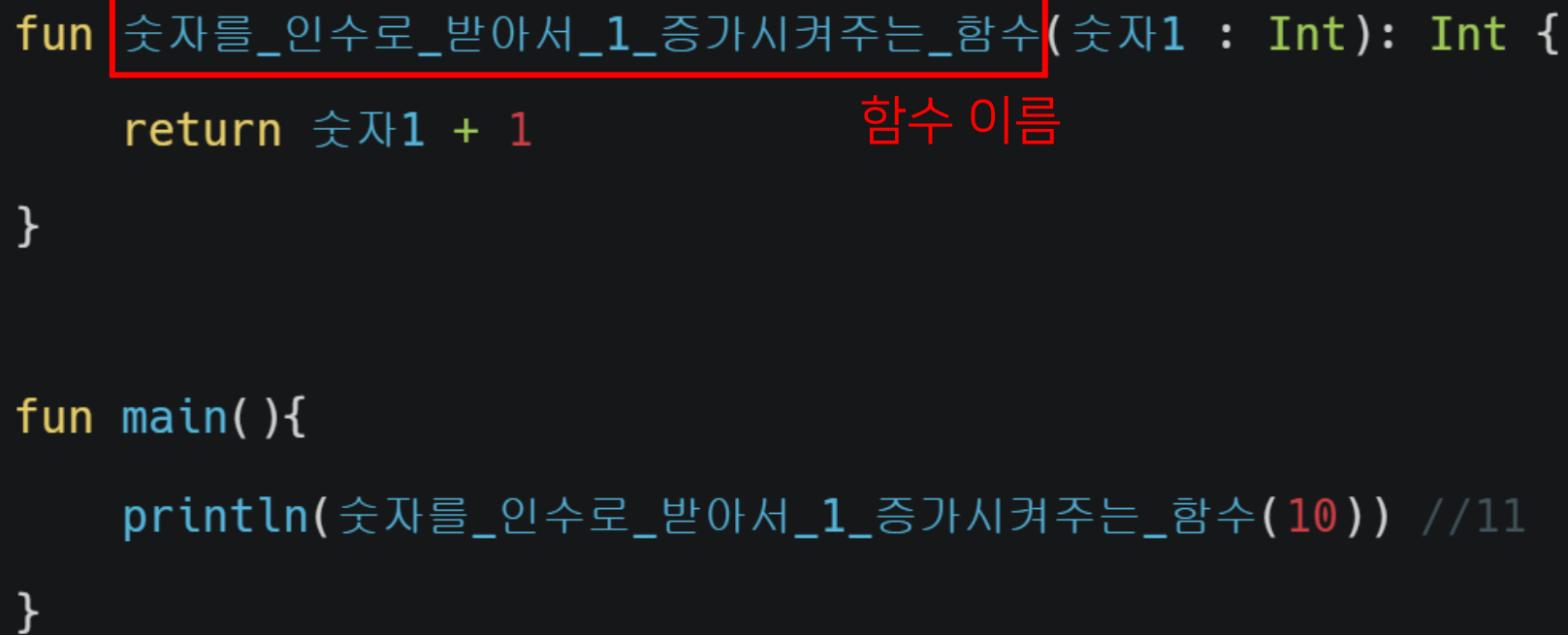
```
fun main(){  
    println( 숫자를_인수로_받아서_1_증가시켜주는_함수(10)) //11  
}
```


코틀린에서 함수란?



```
fun 숫자를_인수로_받아서_1_증가시켜주는_함수(숫자1 : Int): Int {  
    return 숫자1 + 1  
}  
  
fun main(){  
    println(숫자를_인수로_받아서_1_증가시켜주는_함수(10)) //11  
}
```


코틀린에서 함수란?



```
fun 숫자를_인수로_받아서_1_증가시켜주는_함수( 숫자1 : Int): Int {  
    return 숫자1 + 1  
}  
  
fun main(){  
    println(숫자를_인수로_받아서_1_증가시켜주는_함수(10)) //11  
}
```

함수 이름

코틀린에서 함수란?



```
fun 숫자를_인수로_받아서_1_증가시켜주는_함수(숫자1 : Int): Int {  
    return 숫자1 + 1  
}
```

함수에 들어오는 값
(= 파라미터 = 인자 = 인수 = 장인수)

```
fun main(){  
    println(숫자를_인수로_받아서_1_증가시켜주는_함수(10)) //11  
}
```


코틀린에서 함수란?

```
fun 숫자를_인수로_받아서_1_증가시켜주는_함수(숫자1 : Int): Int {  
    return 숫자1 + 1  
}  
  
fun main(){  
    println(숫자를_인수로_받아서_1_증가시켜주는_함수(10)) //11  
}
```

함수가 반환하는 값의 자료형

함수가 반환하는 값

코틀린에서 함수란?



```
fun 숫자를_인수로_받아서_1_증가시켜주는_함수( 숫자1 : Int): Int {  
    return 숫자1 + 1  
}
```

```
fun main(){  
    println(숫자를_인수로_받아서_1_증가시켜주는_함수(10)) //11  
}
```

함수 사용하기 (=함수를 호출한다.)

코틀린에서 함수란?

iceCreamMachine(🍓) // 🍦

iceCreamMachine(🍫) // 🍦

iceCreamMachine(🥛) // 🍦

코틀린에서 함수란?

$$f(x) = x^2 + 2x + 7$$

```
fun f(x : Int): Int {  
    return x * x + 2 * x + 7  
}  
  
fun main(){  
    println(f(1)) //10  
}
```


함수 내부의 변수



```
fun 대충_함수_이름() {  
    val x = 100  
    println(x)  
}
```

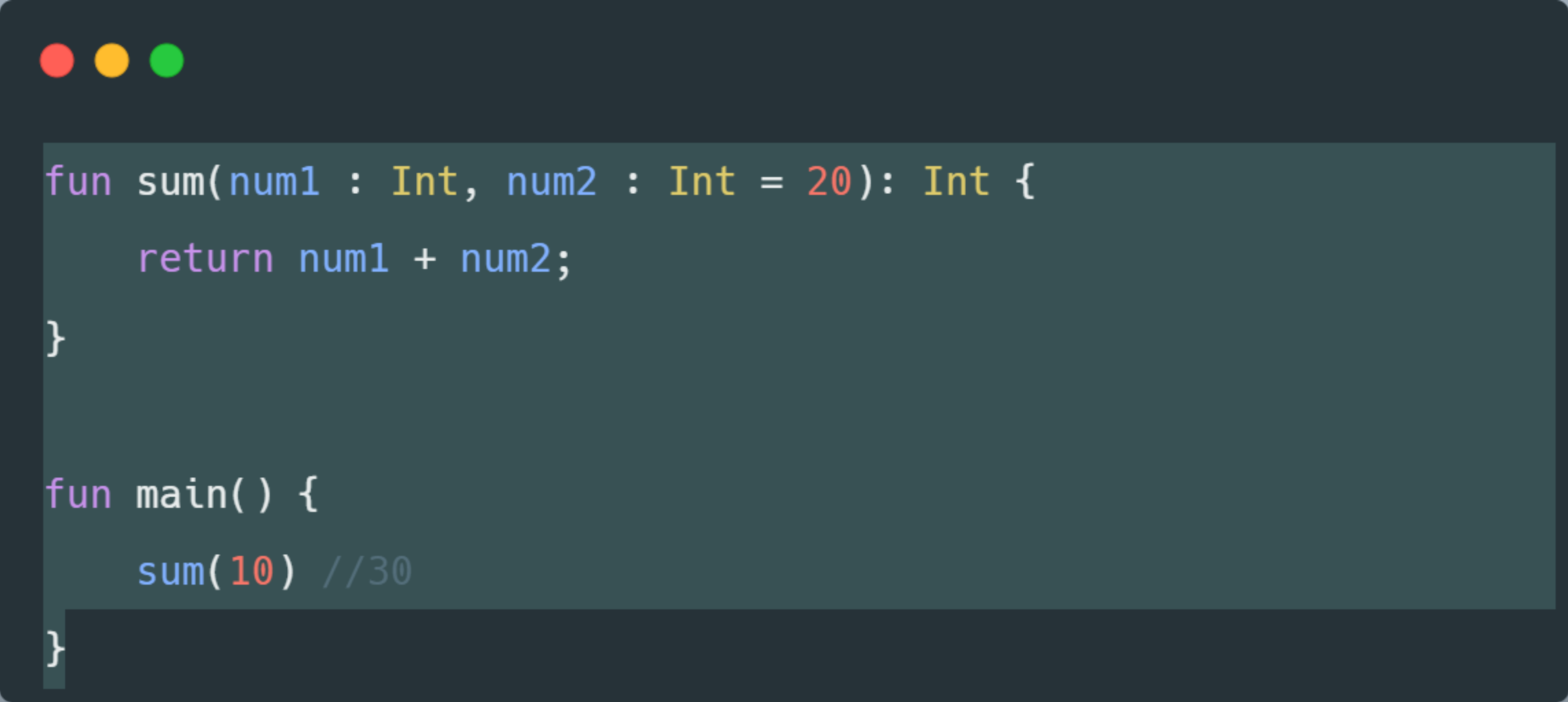
```
fun main(){  
    val x = 20  
  
    println(x)  
    대충_함수_이름()  
    println(x)  
}
```


함수 내부의 변수



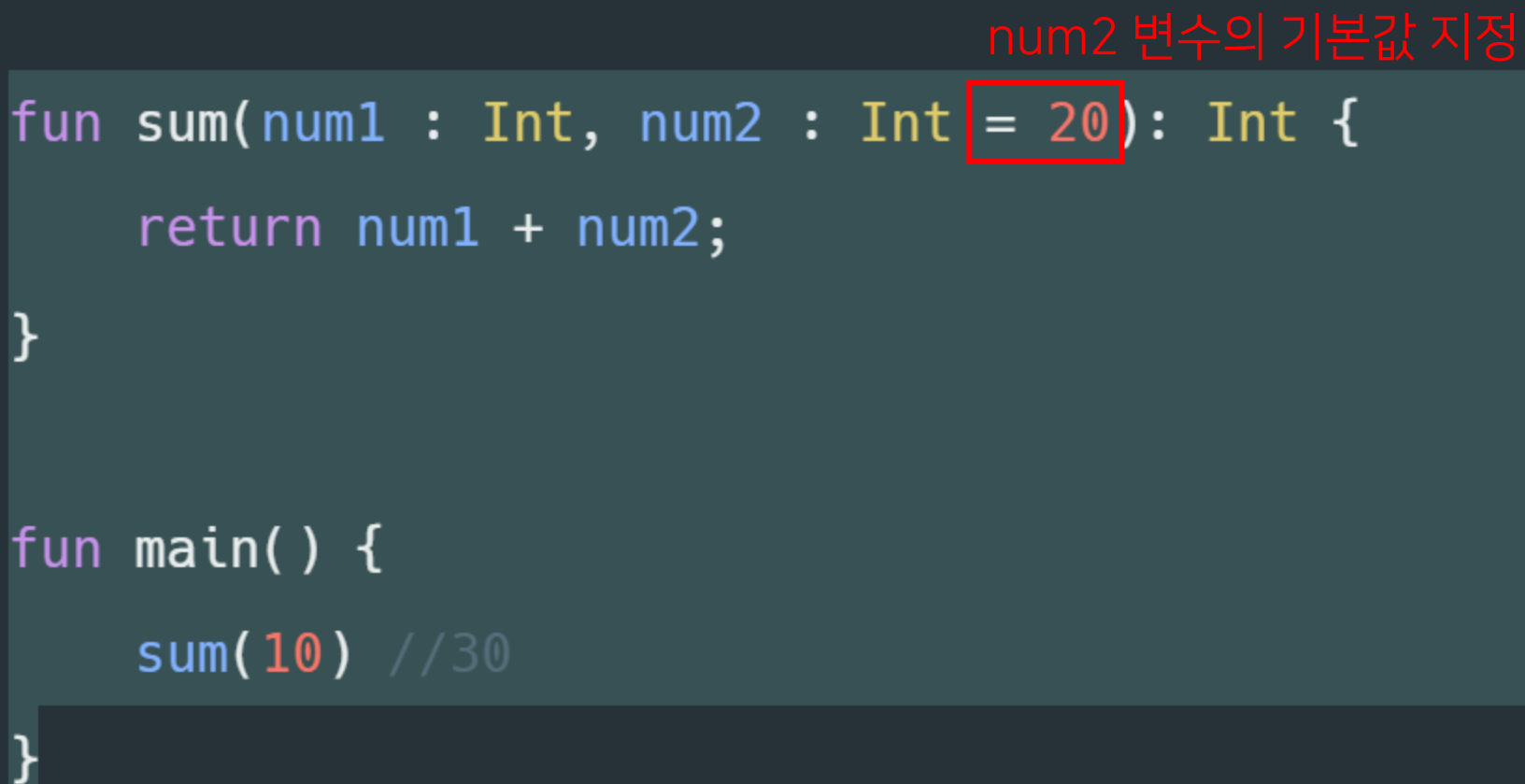
```
fun 대충_함수_이름() {  
    val x = 100  
    println(x) // 100  
}  
  
fun main(){  
    val x = 20  
  
    println(x) // 20  
    대충_함수_이름()  
    println(x) // 20  
}
```

함수 인수의 기본값



```
fun sum(num1 : Int, num2 : Int = 20): Int {  
    return num1 + num2;  
}  
  
fun main() {  
    sum(10) //30  
}
```

함수 인수의 기본값



num2 변수의 기본값 지정

```
fun sum(num1 : Int, num2 : Int = 20): Int {  
    return num1 + num2;  
}  
  
fun main() {  
    sum(10) //30  
}
```

코틀린에서 함수 예제

- 두 수를 인수로 받아서 더 큰 값을 반환하는 함수
- 이름을 인수로 받아서 "안녕하세요. {이름}"을 출력하는 함수
- 두 수를 인수로 받아서 더한 값을 반환하는 함수


람다

lamda

람다

- 람다 : 함수를 더 쉽게 표현해서 변수에 저장한 것


람다



```
fun sum(x: Int, y: Int) {  
    println("$x + $y")  
    return x + y  
}
```

함수로 표현

람다로 표현



```
var sum = { x: Int, y: Int ->  
    println("$x + $y")  
    x + y  
}
```

```
sum(10, 20) // 20
```

람다



```
var sum = { 인수들  
  x: Int, y: Int ->  
    println("$x + $y")  
    x + y 반환 값  
}
```

```
sum(10, 20) // 20 람다의 사용
```


람다를 사용하는 이유

- 편리하게 함수를 만들고, 조작할 수 있다.
- 함수의 인자로 함수를 넘겨줄 수 있다. (= 콜백 함수)

시간이 걸리는 작업이 끝난 뒤 어떤걸 수행해라...

람다를 사용하는 이유



```
fun highOrder(인자로_받은_람다: (Int, Int) -> Int, num1: Int, num2: Int): Int {  
    return 인자로_받은_람다(num1, num2)  
}  
  
fun main() {  
  
    var result : Int = highOrder({ x, y -> x + y}, 10, 20)  
    println(result)  
}
```

람다를 사용하는 이유



람다식을 인자로 받을 때는 자료형을
람다식이 인자로 받는 값, 반환하는 값의 자료형을 적어준다.

```
fun highOrder(인자로_받은_람다: (Int, Int) -> Int, num1: Int, num2: Int): Int {  
    return 인자로_받은_람다(num1, num2) "Int형과 Int형을 받아서 Int형을 반환한다" 라는 뜻  
}
```

```
fun main() {  
  
    var result : Int = highOrder({ x, y -> x + y}, 10, 20)  
    println(result)  
}
```

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.

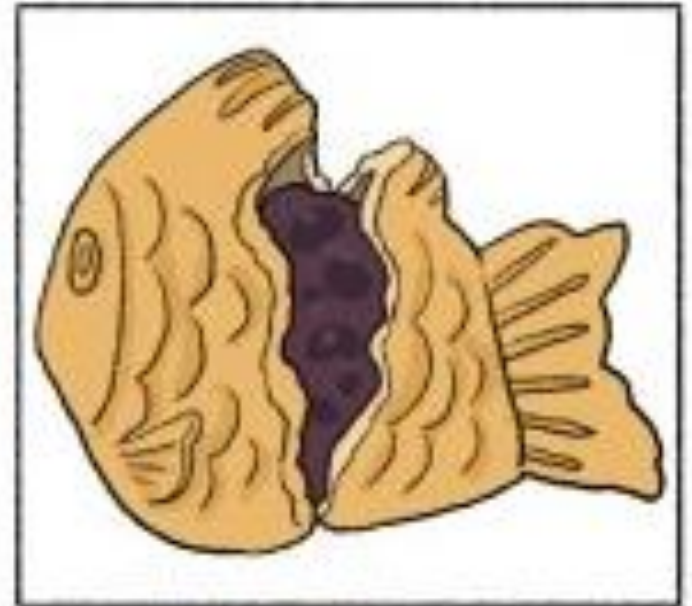
객체 지향

- 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 }
```

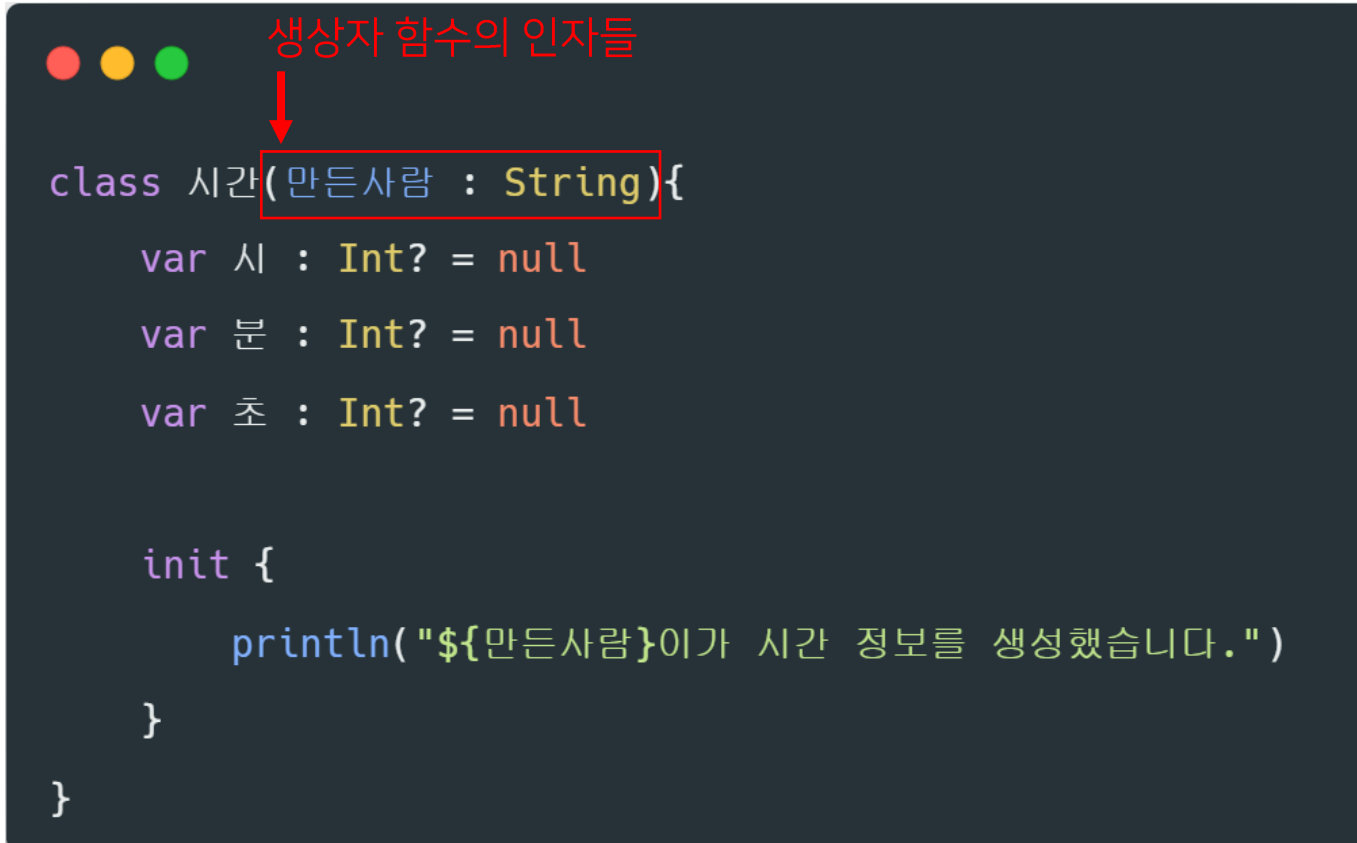
class 생성자

- 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("${만든사람}(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 {
510         | A constant holding the minimum value an instance of Int can have.
513         public const val MIN_VALUE: Int = -2147483648
514
515         | A constant holding the maximum value an instance of Int can have.
518         public const val MAX_VALUE: Int = 2147483647
519
520         | 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
526         | 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
26     | Returns a string obtained by concatenating this string with the string representation of the given
27     | other object.
29     public operator fun plus(other: Any?): String
30
31     public override val length: Int
32
33     | Returns the character of this string at the specified index.
34     | If the index is out of bounds of this string, throws an IndexOutOfBoundsException except in
35     | 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

상속

상속



학생의 추상화

값 (멤버 변수)

- 이름
- 나이
- 키
- 몸무게
- 학교
- 점수
- 동아리

기능 (메소드)

- 인사 하기
- 걷기
- 말하기
- 공부 하기
- 등교 하기

공통된 멤버 변수와 메소드 존재



선생님의 추상화

값 (멤버 변수)

- 이름
- 나이
- 키
- 몸무게
- 학교
- 과목
- 담당 학년
- 교무실

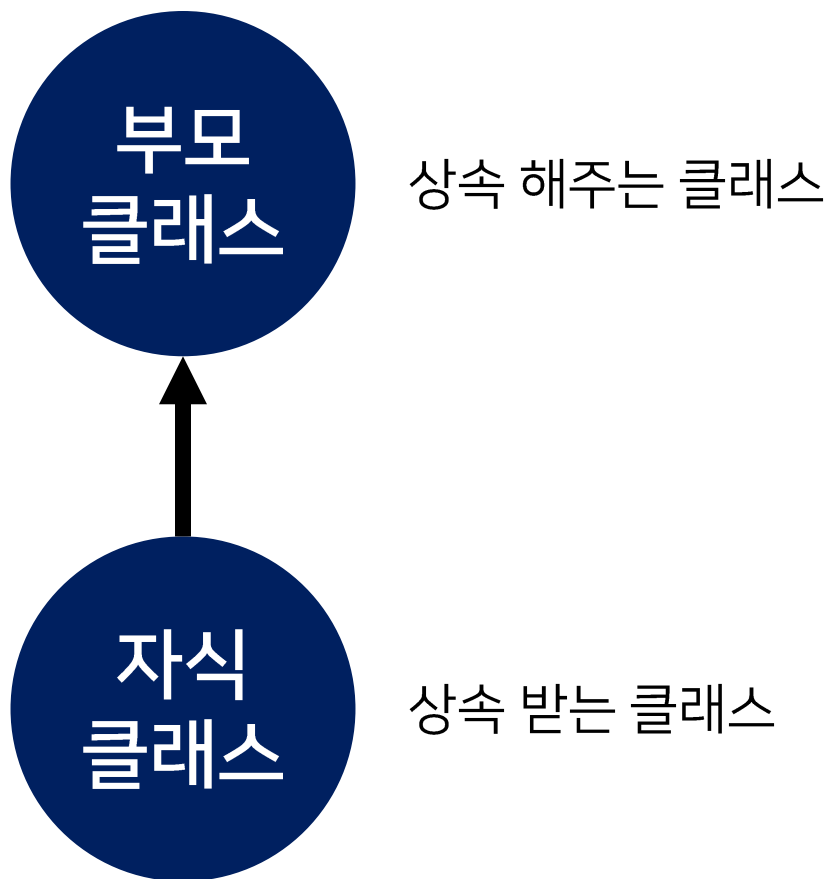
기능 (메소드)

- 인사 하기
- 걷기
- 말하기
- 정보 보여주기
- 수업하기
- 문제 내기

상속

상속 : 다른 클래스들의 멤버 변수와 메소드 들을 사용할 수 있게 한다.

Kotlin에서는 하나의 클래스만 상속 받을 수 있다.



해당 클래스가
상속 할 수 있게 만든다.


```
open class Person(  
    val name : String,  
    val age : Int,  
    val height : Float,  
    val weight : Float,  
    val school : String  
) {  
    fun hello() = println("안녕하세요. ${name}입니다.")  
    fun walk() = println("$name : 걷기")  
    fun talk(message : String) = println("$name : $message")  
}
```

상속 받을 클래스는 : 뒤에 써준다.
이때 넘겨줘야 하는 인자들을
같이 넘겨줘야 한다.

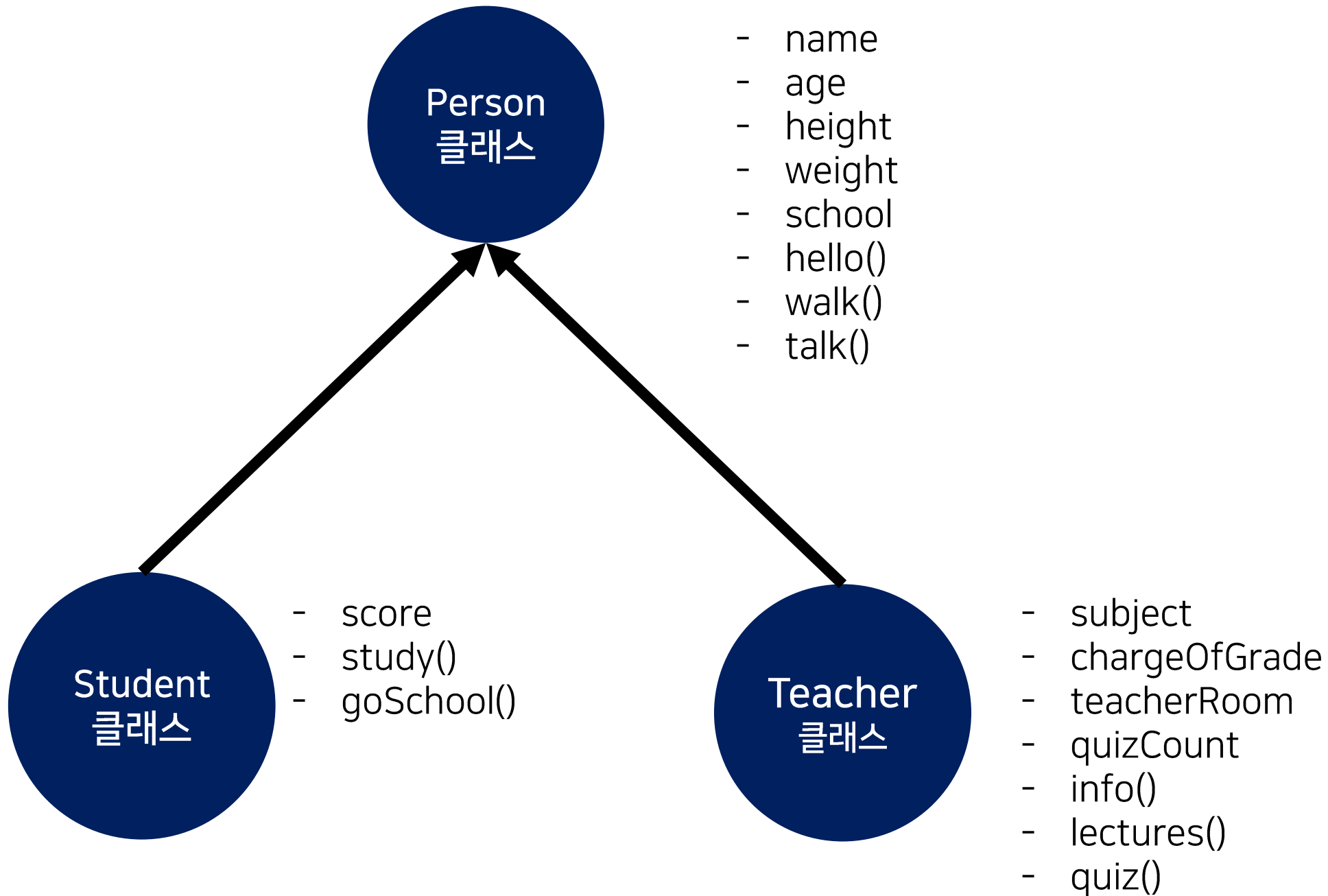
```
class Student(  
    name: String,  
    age: Int,  
    height: Float,  
    weight: Float,  
    school: String,  
) : Person(name, age, height, weight, school){  
    var score: Int = 0  
  
    fun study() {  
        println("$name : 공부 공부")  
        score += 10  
    }  
  
    fun goSchool() = println("$name : 등교 등교")  
}
```

상속 받는 클래스

```
class Teacher(  
    name: String,  
    age: Int,  
    height: Float,  
    weight: Float,  
    school: String,  
    val subject : String,  
    val chargeOfGrade : Int    상속 받는 클래스  
) : Person(name, age, height, weight, school){  
    lateinit var teacherRoom : String  
    var quizCount = 0;  
  
    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})")  
}
```



```
fun main() {  
    val s1 = Student("박희찬", 18, 178f, 70f, "선린인터넷고등학교")  
    val s2 = Student("장인수", 18, 200f, 70f, "선린인터넷고등학교")  
  
    val t1 = Teacher("심희원", 20, 160f, 50f, "선린인터넷고등학교", "프로그래밍", 1)  
    val t2 = Teacher("이왕렬", 40, 170f, 50f, "선린인터넷고등학교", "웹프로그래밍 실무", 2)  
  
    s1.name // 박희찬,  
    s1.age // 18  
    s1.hello()  
    s1.goSchool()  
  
    t1.name // 심희원,  
    t1.age // 20  
    t1.quiz("상")  
}
```



생성자 실행 순서는?

```
open class Parent(){
```

```
    init {
```

```
        println("부모 객체가 생성 되었습니다.")
```

```
    }
```

```
}
```

첫번째 실행

부모 클래스의 생성자 실행



```
class Child() : Parent(){
```

```
    init {
```

```
        println("자식 객체가 생성 되었습니다.")
```

```
    }
```

```
}
```

두번째 실행

```
fun main() {
```

```
    Child()
```

```
}
```

메소드 오버로딩

- 부모 클래스의 메소드의 이름과 받는 인자를 같지만 서로 다른 기능을 만들고 싶을 때

```
open class Person(val name : String, val age : Int){  
    open fun work(time : Int) {  
        println("${time}시간 동안 일을 합니다")  
    }  
}  
  
class Student(name: String, age: Int) : Person(name, age){  
    override fun work(time: Int) {  
        println("${time}시간 동안 공부를 합니다")  
    }  
}
```


메소드 오버로딩

- 부모 클래스의 메소드의 이름과 받는 인자를 같지만 서로 다른 기능을 만들고 싶을 때

```
open class Person(val name : String, val age : Int){  
    open fun work(time : Int) { 이름과 인자는 같다.  
        println("${time}시간 동안 일을 합니다")  
    }  
}  
  
class Student(name: String, age: Int) : Person(name, age){  
    override fun work(time: Int) {  
        println("${time}시간 동안 공부를 합니다")  
    }  
}
```

메소드 오버로딩

- 부모 클래스의 메소드의 이름과 받는 인자를 같지만 서로 다른 기능을 만들고 싶을 때

```
open class Person(val name : String, val age : Int){  
    open fun work(time : Int) {  
        println("${time}시간 동안 일을 합니다")  
    } 오버로딩 해줄 부모 메소드에 open을 붙여준다.  
}  
  
class Student(name: String, age: Int) : Person(name, age){  
    override fun work(time: Int) {  
        println("${time}시간 동안 공부를 합니다")  
    } 자식 메소드에 override을 붙여준다.  
}
```

특수 클래스

특수 클래스

- 특수한 목적을 위해서 사용되는 클래스

데이터
클래스

추상 클래스

인터페이스


데이터 클래스

- 데이터를 저장하기 위해서 사용하는 클래스

예시

- 유저 데이터를 저장하는 클래스,
- 동아리 데이터를 저장하는 클래스,
- 게시물 데이터를 저장하는 클래스

데이터 클래스



```
data class Board(  
    val title : String, //게시물 제목  
    val content : String, //게시물 내용  
    val writer : User, //작성자 정보  
){}  

```

게시물 정보를 저장하는 클래스

데이터 클래스

앞에 data를 붙여준다.

```
data class Board(  
    val title : String, //게시물 제목  
    val content : String, //게시물 내용  
    val writer : User, //작성자 정보  
){}  

```

게시물 정보를 저장하는 클래스

데이터 클래스

사용자 정보를 저장하는
데이터 클래스

```
data class User(  
    var id : String, //유저의 id  
    var name : String, //사용자 이름  
    var email : String, //사용자 이메일  
  
    var userMSG : String, //사용자의 소개 메시지  
    var profileImgUrl : String, //Storage에 저장된 사용 프로필 사진  
  
    var likeBoardList : ArrayList<String>,  
    //사용자가 좋아요 누른 게시물의 id  
  
    var totalLikeCount : Int, //사용자가 지금까지 받은 좋아요 개수  
)
```


데이터 클래스를 사용하는 이유

- toString(), equals(), hashCode() 메소드 등이 자동으로 만들어진다.

```
fun main() {  
    val boardData = Board("안녕하세요.", "대충 내용")  
    println(boardData.toString())  
}
```

Board(title=안녕하세요., content=대충 내용)

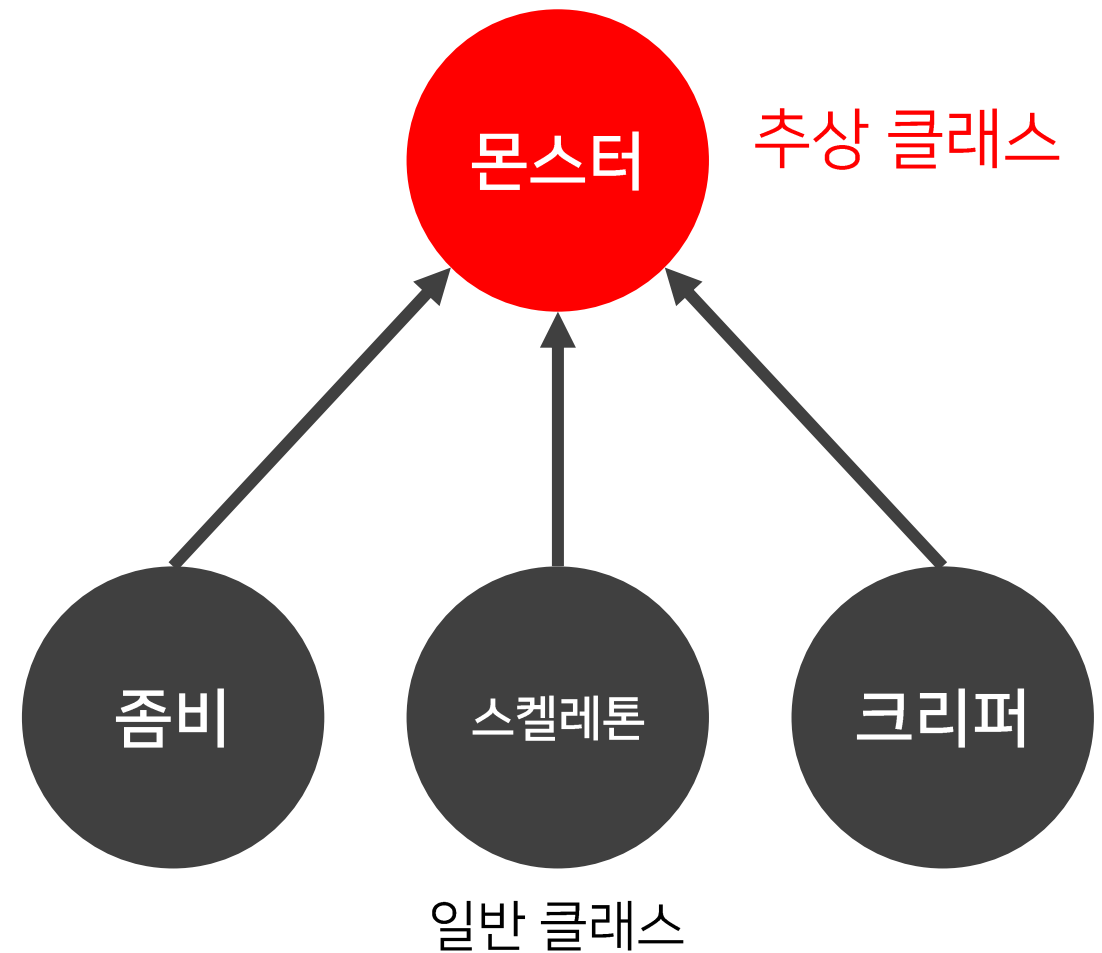
data class로 만들었을 때

Board@5b6f7412

일반 class로 만들었을 때

추상 클래스

- 추상화를 위해서 사용하는 클래스
- 직접 객체로 만들어 주지는 않는다.
- 추상클래스의 멤버 변수와 메소드는 선언만 해주고 상속받은 다음 구현 해준다.



추상 클래스

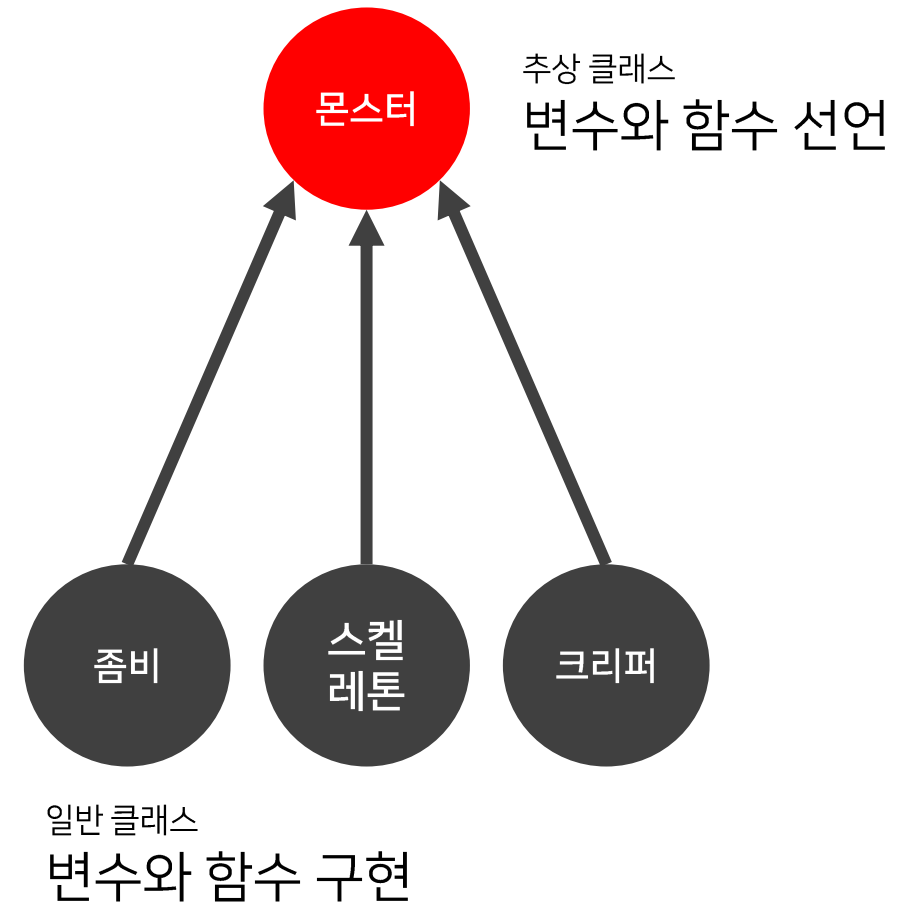
```
abstract class Monster(  
    val hp : Int, // 현재 체력  
    val maxHp : Int, // 최대 체력  
    val damage : Int, // 공격력  
) {  
    abstract val resources : String //몬스터 리소스  
    상속받은 다음 구현 해줄 변수와 함수  
    abstract fun attack()  
}
```

추상 클래스

```
class Zombie hp : Int, maxHp: Int, damage: Int) : Monster hp, maxHp, damage){
    override val resources: String = "res/zombie.png"
    override fun attack() {
        print("때리기")
    }
}

class Skeleton hp : Int, maxHp: Int, damage: Int) : Monster hp, maxHp, damage){
    override val resources: String = "res/skeleton.png"
    override fun attack() {
        print("활쏘기")
    }
}

class Creeper hp : Int, maxHp: Int, damage: Int) : Monster hp, maxHp, damage){
    override val resources: String = "res/creeper.png"
    override fun attack() {
        print("터지기")
    }
}
```

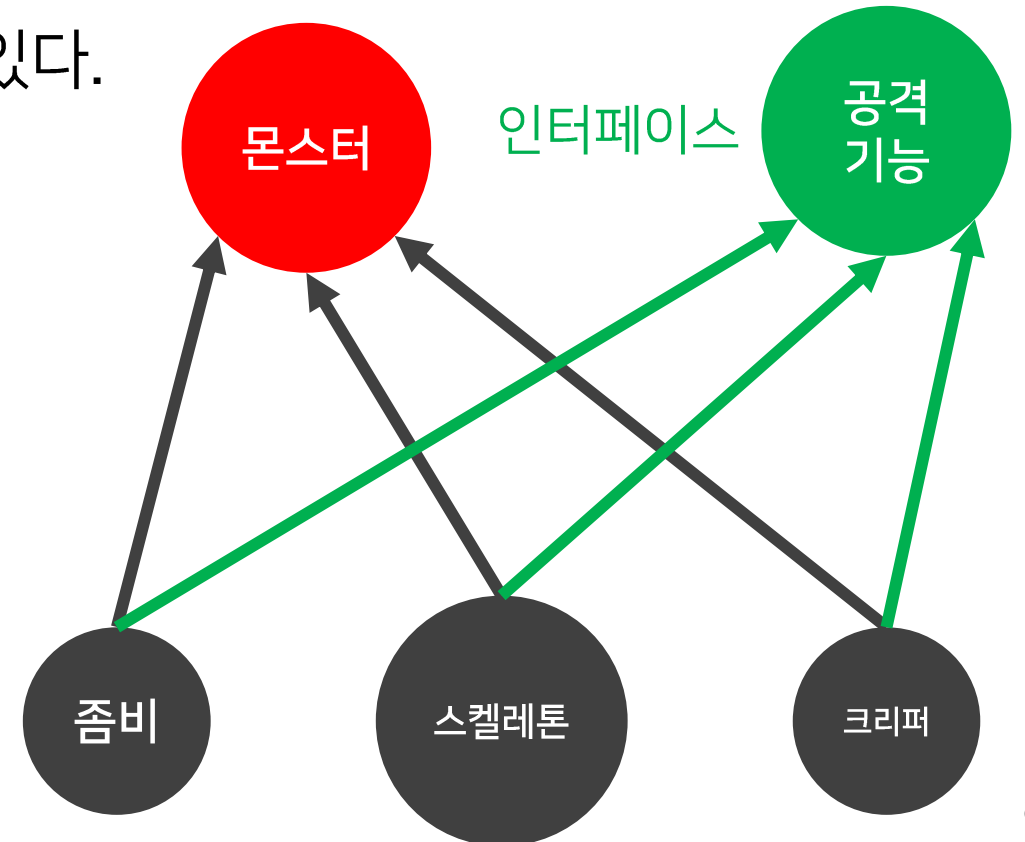


인터페이스

- 추상 클래스 만들기 귀찮을 때 사용하는 것

- 소프트웨어 마에스트로 연수생 **김형진**

- 추상 클래스와 비슷하지만 **메소드만 추상화** 할 수 있다.
- 하나의 클래스는 **여러 인터페이스를 상속** 받을 수 있다.
- 공통된 기능들을 추상화 할 때 사용 한다.



인터페이스



```
interface AttackMonster {  
    fun attack()  
}
```

```
abstract class Monster(  
    val hp : Int, // 현재 체력  
    val maxHp : Int, // 최대 체력  
    val damage : Int, // 공격력  
) {  
    abstract val resources : String //몬스터 리소스  
}
```

인터페이스

```
class Zombie (hp : Int, maxHp: Int, damage: Int) : Monster (hp, maxHp, damage), AttackMonster {  
    override val resources: String = "res/zombie.png"  
    override fun attack() {  
        print("때리기")  
    }  
}
```

```
class Skeleton (hp : Int, maxHp: Int, damage: Int) : Monster (hp, maxHp, damage), AttackMonster {  
    override val resources: String = "res/skeleton.png"  
    override fun attack() {  
        print("활쏘기")  
    }  
}
```

클래스 과제

[마인크래프트를 코틀린으로 구현해보기]

조건

- 좀비, 스켈레톤, 크리퍼, 주민, 플레이어 구현
- 모든 일반 클래스에 멤버 변수와 메소드 각각 3개 이상 사용
- 추상 클래스와 인터페이스 각각 2개 이상 구현

기한

5월 31일 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