Kotlin 시작하기

이준호

Kotlin 소개

- 2011년 7월에 JetBrains에서 처음으로 발표
- 2016년 2월 v1.0 버전으로 시작
- 현재 1.1.3-2버전까지 배포
- Google I/O 2017에서 Android 개발 언어로 정식 채택
- Android Studio 3.0에서 Plugin없이 사용 가능

Kotlin 소개

- Concise(간결하고)
- Safe(안전하고)
- Interoperable(Java와의 100% 호환성을 제공)
- Tool-friendly(다양한 tool에서 개발 가능)

BOOSTCAMP

Kotlin 강의 환경설정

- Android Studio 2.3
- Kotlin 1.1.3-2
- TargetSDK 24(Nougat)
- MinSDK 16(JellBean)

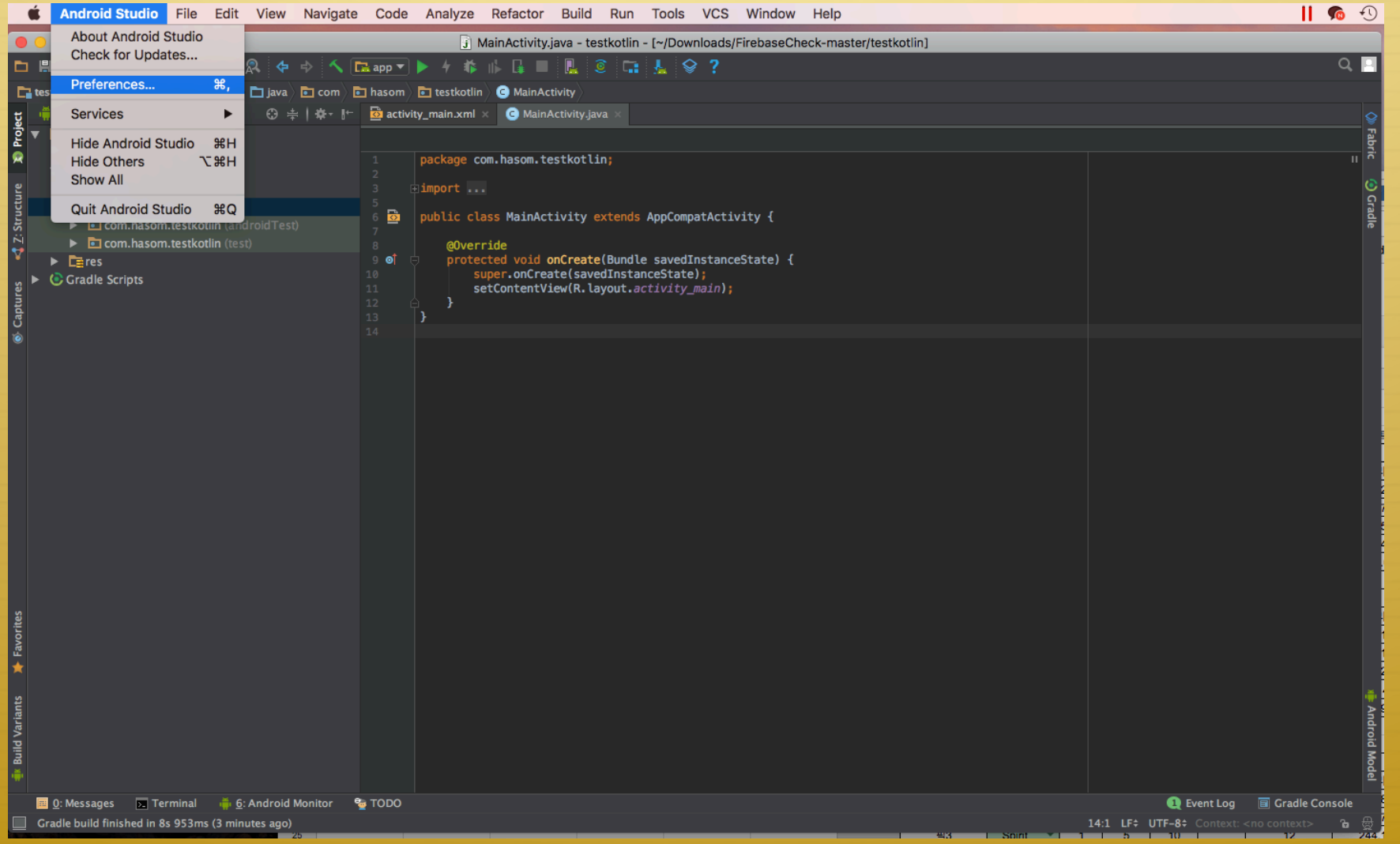
Kotlin 시작하기

BOOSTCAMP

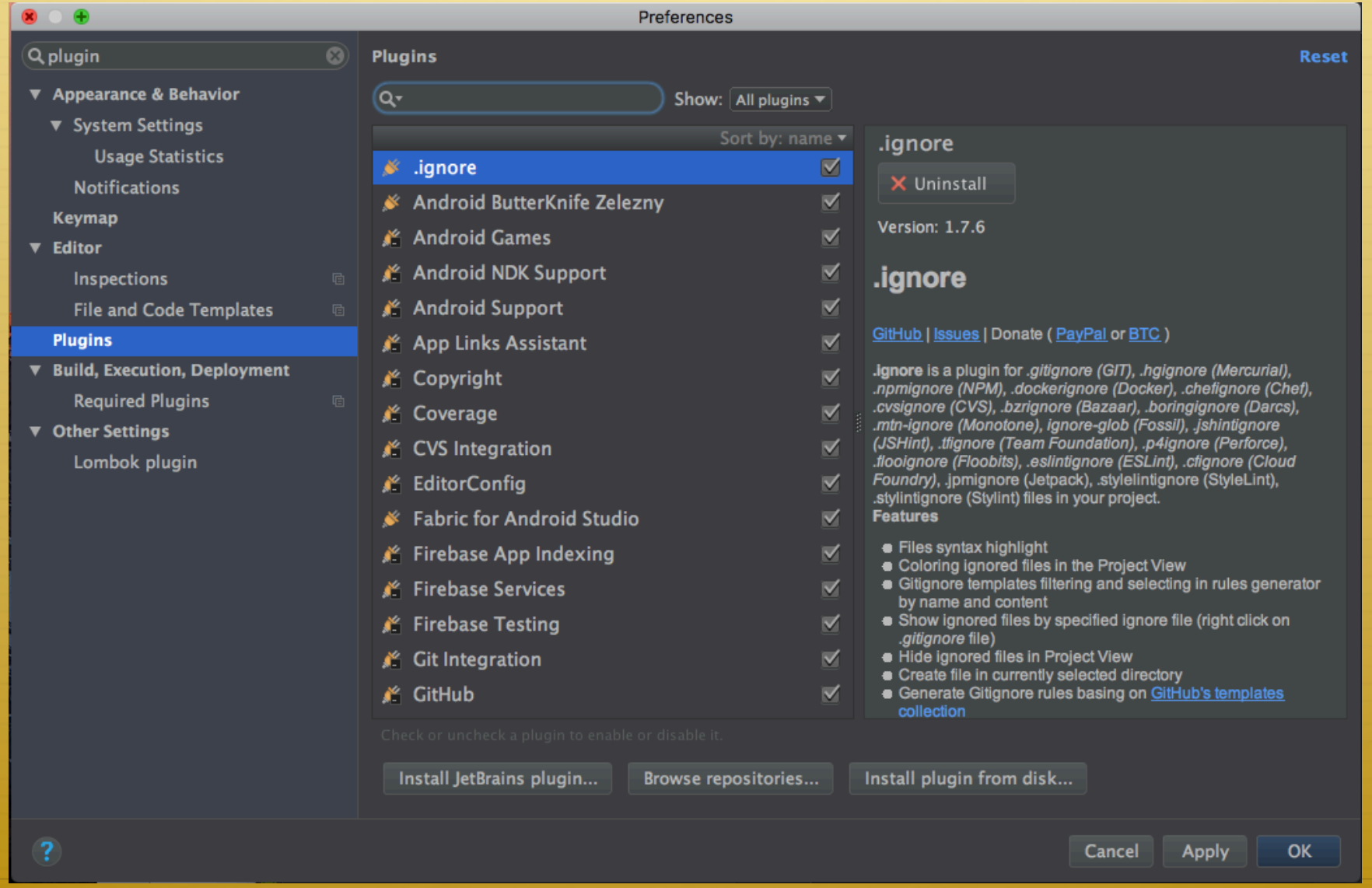
- 안드로이드 스튜디오에서 Kotlin 설치하기
- Java에서 Kotlin으로 Convert하기
- Convert한 코드 살펴보기

BOOSTCAMP

Kotlin 설치하기 - Plugin

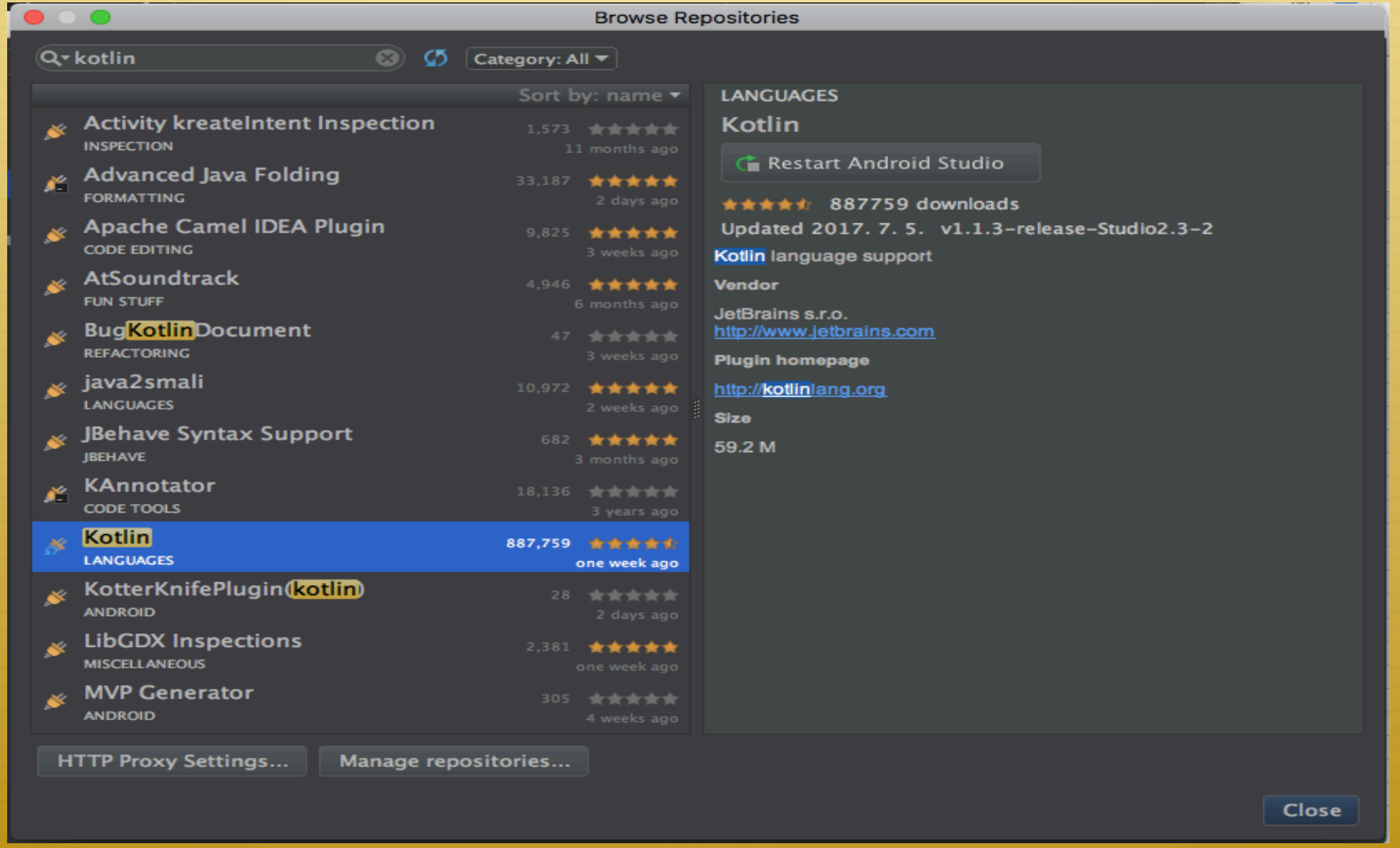


Kotlin 설치하기 - Plugin



BOOSTCAMP

Kotlin 설치하기 - Plugin



Kotlin 설치하기 - dependencies

```
buildscript {
        ext.kotlin_version = ’1.1.3-2'
        dependencies {
        // …
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
        // …
        }
}
```

Kotlin 설치하기 - dependencies

```
apply plugin: 'kotlin-android'


dependencies {
        // ...
        compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version”
}
```

기본 문법

Kotlin 기본타입

- Double 64
- Float 32
- Long 64
- Int 32
- Sort 16
- Byte 8
- String

Kotlin – 변수 선언 방법

- val / var 변수이름 : 변수타입

Kotlin – 변수 선언 방법

- val / var 변수이름 : 변수타입

val a: Int (Read – only)

var a: Int (Read / Write)

Kotlin – 변수 선언 방법

```
val a: Int = 1

a = 111    // ERROR
```

Kotlin – 변수 선언 방법

```
val a: Int = 1

a = 111    // ERROR


var a: Int = 1

a = 111    // OK
```

Java와 비교 - final / val

```
// Java

final int a = 111;
```

Java와 비교 - final / val

```
// Java

final int a = 111;


// Kotlin

val a: Int = 111
```

Java와 비교 - final / val

```
// Java

final int a = 111;



// Kotlin

val a: Int = 111

val a = 111 // Type 생략 가능
```

Java와 비교 - var

```
// Java

int a = 111;
```

Java와 비교 - var

```
// Java

int a = 111;


// Kotlin

var a: Int = 111
```

Java와 비교 - var

```
// Java

int a = 111;


// Kotlin

var a: Int = 111

var a = 111    // Type 생략 가능
```

기본 문법 -findViewById

Kotlin Java 비교

// Java

변수Type 변수명 = (변수Type) findViewById()

// Kotlin

val/var 변수명 : 변수Type = findViewById() as 변수Type

val/var 변수명 = findViewById() as 변수Type

Kotlin Java 비교

```
// Java

Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar)
```

```
// Kotlin

val toolbar: Toolbar = findViewById(R.id.toolbar) as Toolbar

val toolbar = findViewById(R.id.toolbar) as Toolbar
```

기본 문법 - Null 적용하기

Kotlin – Null 적용하기

var a: Int = 123

a = null

Kotlin – Null 적용하기

```
var a: Int = 123

a = null  // 문법상 오류 발생
```

Kotlin - Null 적용하기

```
var a: Int = 123

a = null  // 문법상 오류 발생
```

Kotlin Default NonNull

Kotlin – Null 적용하기

```
var a: Int? = 123
```

Kotlin – Null 적용하기

```
var a: Int? = 123

a = null   // OK
```

기본 문법 - Type 유추하기

Kotlin - Type 유추하기

```
var a: Long = 123

var b: Float = 123.4f
```

Kotlin - Type 유추하기

var a: Long = 123

⇒ var a = 123L // Type 생략 가능하며 Long으로 유추

var b: Float = 123.4f

⇒ var b = 123.4f // Type 생략 가능하며 Float으로 유추

기본 문법 - String templates

Java - String templates

a = 1;

b = 1;

printAB(int a, int b) {

  Log.d(“TAG”, “a = “ + a + “ b = “ + b);

}

결과 : a = 1 b = 1

Kotlin - String templates

a = 1

b = 1

```
fun print(a: Int, b: Int) {
    println(“a = $a b = $b")
}
```

결과 : a = 1 b = 1

Kotlin – String templates

a=1, b=1

```
fun printSum(a: Int, b: Int) {
    println("sum of $a and $b is ${a + b}")
}
```

결과 : sum of 1 and 1 is 2

기본 문법 - String Literals

Kotlin – String Literals

```
val hello = “hello\nworld”

val hello1 =

“””
hello

world

“””
```

Kotlin – String Literals

val hello = “{\”Key\”:\”value\”}”

“””

{“key”:”value”}

“””

Kotlin – String Literals

```
val json = """
        |{"key":"value"},
        |("key1":"value1"}
        """.trimMargin()
```

기본 문법 - 함수생성

Java 함수

```
public int getSum(int a, int b) {

        reutn a+b;

}
```

Kotlin 함수

```
fun getSum(a: Int, b: Int) {
        println(a + b)
}
```

Kotlin 함수

```
fun getSum(a: Int, b: Int) {
        println(a + b)
}
fun sum(a: Int, b: Int): Int {
        return a + b
}
```

Kotlin 함수 - Unit

```
fun 함수명(): Unit {      // Unit 생략가능 (java = void)
        함수 정의
 }
fun 함수명(변수명 : 변수타입) {
        함수 정의
}
```

Kotlin 함수 - return

```
fun 함수명(): return 타입 {
        return 값
}
fun 함수명(변수명 : 변수타입): return 타입 {
        return 값
}
```

Kotlin 함수축약하기

```
fun getSum(a: Int, b: Int): Int {

        return a + b

}
```

Kotlin 함수축약하기

```
fun getSum(a: Int, b: Int): Int {
        return a + b
}
fun getSum(a : Int, b: Int): Int = a + b
```

Kotlin 함수축약하기

```
fun getSum(a: Int, b: Int): Int {
        return a + b
}
fun getSum(a : Int, b: Int): Int = a + b
fun getSum(a : Int, b: Int) = a + b   // Int를 유추 가능
```

Kotlin 함수축약하기

```
fun max(a: Int, b: Int): Int {
        if (a > b) return a
        else return b
}
```

Kotlin 함수축약하기

```
fun max(a: Int, b: Int) = if (a > b) a else b
```

기본 문법 - Infix Notation (중위표기법)

Kotlin 중위표기법

```
fun Int.max(x: Int) : Int = if (this > x) this else x
```

ex) 1.max(15)

result = 15

ex2) 12.max(10)

result = 12

Kotlin 중위표기법

```
// 변수가 1개일때만 가능
infix fun Int.max(x: Int) : Int = if (this > x) this else x
```

ex) 1 max 15

result = 15

ex2) 12 max 10

result = 12

기본 문법 - Any

Java - Object

```
private int getLength(Object obj) {
        if (obj instanceof String) {
                return ((String) obj).length();
        }
        return 0;
}
```

Kotlin – Any

```
fun getLength(obj: Any) : Int {
        if (obj is String) {
                return obj.length
        }
        return 0
}
```

Kotlin – Any

```
fun getLength(obj: Any) : Int {
        if (obj !is String) {
                return 0
        }
        return obj.length
}
```

Kotlin – Any

```
fun cases(obj : Any) {
        when (obj) {
                1 -> println(“1”)
                “Kotlin” -> println(“Kotlin”)
                is Long -> println(“Long”)
                !is String -> println(“Not String”)
                else -> println(“Unknown”)
        }
}
```

기본 문법 - loop

Java - loop

```
ArrayList<String> list = new ArrayList<>();

for (String s : list) {

        Log.d("TAG", s);

}
```

Kotlin - loop

```
val list = new ArrayList<String>();

for (s in list) {

        println(s);

}
```

Java - ranges

```
for(int i = 0; i <= 5; i++) {

        Log.d(“TAG”, i);

}
```

Kotlin – ranges

```
for(i in 0..5) {                    // 5 이하(증가)
        println(i);
}
```

Kotlin – ranges

// 미만 (증가)

for(i in 1 until 5) { println(i) } => 1,2,3,4

// 이하 (감소)

for(i in 5 downTo 1) { println(i) } => 5,4,3,2,1

// 이하 (단위)

for(I in 0.. 5 step 2) { prinln(i)} => 0, 2, 4

기본 문법 - Lambdas(람다식)

Kotlin – Lambdas

```
button.setonClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                view.setAlpha(0.5f);
        }
});
```

Kotlin – Lambdas

```
button.setonClickListener(
        view -> view.alpha = 0.5f
);
```

Kotlin – Lambdas

```
button.setonClickListener(
        it.alpha = 0.5f
);
```

Kotlin – Lambdas

parameter1, parameter2… ->

기본 문법 - for문에 필터걸기

Kotlin – stream

```
for (Integer i: list) {
        if (i > 5) {
                i *= 2;
                Log.d(“TAG”, i);
        }
}
```

Kotlin – stream

```
list.filter { it > 5 }.map { println(it*2) }
```

기본 문법 - Singleton

Java – Singleton

```
public class Singleton {
        private static Singleton singleton;
        private Singleton(){};
        public Singleton getInstance() {
                if(singleton == null) {
                        singleton = new Singleton();
                        return singleton;
        }
}
```

Kotlin – Singleton

```
object Singleton{

        …

}
```

기본 문법 - static

Java – static

```
public class MainFragment extends Fragment {
        public static MainFragment getInstance() {
                return new MainFragment();
        }
        ….
}
```

Kotlin – static

```
class MainFragment : Fragment() {

        companion object {     // Java식의 static instance

                fun getInstance() = MainFragment();

        }

        ….

}
```

안전한 Null 처리

Java – Null 처리 메소드

```
public void set(@NotNull String a, @Nullable String b) {

        …

}
```

Kotlin – Null 처리 메소드

```
fun set(a: String, b: String ?) {

        ….

}
```

Kotlin – Null 처리 메소드

@NotNull String a -> a: string

Kotlin – Null 처리 메소드

@NotNull String a -> a: string

@Nullable String a -> a: string?

Kotlin – Null 문법 오류

```
var temp: String  =“abc”

temp = null          // 문법 오류 발생
```

Kotlin – Null 처리

var temp: String? =“abc”

temp = null

Kotlin – Null 처리 주의사항

var temp = null

temp = “ABC” // Type 오류 발생

⇒ Type을 쓰지 않고 null을 넣게되면 Type이 없는 null의 형태로 만들어지게됨.

⇒ 즉 어떠한 변수로도 초기화 될 수 없음

Kotlin – Null 처리 주의사항

Null을 하기 위한 초기화에는 Type을 써야한다.

var temp: String? = null

temp = “ABC”

Java - Null 체크

```
String tmp = null;

int size = -1;

if (tmp != null) {

        size = tmp.length();

}
```

Java - Null 체크

```
String tmp = null;

int size = -1;

if (!TextUtils.isEmpty(tmp)) {

        size = tmp.length();

}
```

Kotlin – Null 체크

```
var tmp: String? = null

var size = -1

if (tmp != null) {

        size = tmp.length

}
```

Java- Null 체크

간단표현식

```
String tmp = null;

int size = tmp != null ? tmp.length() : 0;
```

Kotlin – Null 체크

간단표현식

```
var tmp: String? = null

val size = if (tmp != null) temp.length else 0
```

Safe Calls

Kotlin – Safe Calls

// null을 포함 할 수 있는 tmp var 변수이며 null로 초기화 합니다.

var tmp: String? = null

Kotlin – Safe Calls

```
// null을 포함 할 수 있는 tmp var 변수이며 null로 초기화 합니다.

var tmp: String? = null

// 실제 사용할 때에는 물음표(?.)를 다시 한번 포함합니다.

val size = tmp?.length // 안전한 null처리
```

Java - Safe Calls

```
int getSize(String temp) {

        return temp != null ? temp.length : null

}
```

Kotlin – Safe Calls

```
fun getSize(temp: String?) : Int? {
        return temp?.length
}
// temp가 null이면 null이 return
// temp가 null이 아니면 length가 return
```

Kotlin -Safe Calls의 장점

```
// 다음과 같이 초기화 되었을 경우
// aaa = AAA()
// bbb = BBB()
// ccc = null
if (aaa != null && bbb != null && ccc != null) {
        return aaa.bbb.ccc.name;
}
return null;
```

Kotlin -Safe Calls의 장점

```
// 다음과 같이 초기화 되었을 경우

// aaa = AAA()

// bbb = BBB()

// ccc = null

return aaa?.bbb?.ccc?.name
```

List에서 Null 제거

Java - List에서 null제외하기

```
List<String> list = new ArrayList<>();
list.add(“A”);
list.add(null);
list.add(“B”);
for (String text : list) {
        if (text != null) {
                Log.d(“TAG”, text);
        }
}
```

Kotlin – List에서 null제외하기

```
val list = List<String?> = listOf(“A”, null, “B”);

for (text in list) {
        if (text != null) {
                println(text)
        }
}
```

Kotlin – List에서 null제외하기

?.let{} 을 이용하면 완전한 null을 배제하고, 즉시 값을 사용할 수 있다.

Kotlin – List에서 null제외하기

```
val list = List<String?> = listOf("A", null, "B");

for (text in list) {
        item?.let{
                println(it)
        }
}
```

Kotlin – List의 null제외하기(FilterNotNull)

```
val list: List<Int?> = listOf(1, 2, null, 4)
for (i in list) {
        println(i)        // => 1, 2, null,  4
}
//null filter
val intList: List<Int> = list.filterNotNull()
for (i in intList) {
        println(i)        // => 1, 2, 4
}
```

if ~ else 대신 사용하기

Java- if / else 대신 사용하기

String temp = “”;

int size = temp != null ? temp.length() : 0;

Kotlin – if / else 대신 사용하기

BOOSTCAMP

```
var temp : String? = “”

val size = temp?.length   // length 또는 null
```

Kotlin – if / else 대신 사용하기

Elvis Operator ?:

Kotlin – if / else 대신 사용하기

Elvis Operator ?:

```
var temp: String? = null

val size = temp?.length ?: 0  // 0 또는 length
```

기본 클래스

Java class

```
public class ClassName {

        ….

}
```

Kotlin class

```
class ClassName {

        …

}
```

Kotlin class

```
class ClassName

// 별도 구현 내용이 없다면 중괄호 생략 가능
```

생성자

Java 생성자

```
public class ClassName {

        public ClassName(String name) {

                //,..,.

        }

}
```

Kotlin 생성자

```
class ClassName(val name: String) {

        //...

}
```

Kotlin 생성자

```
class 클래스이름(val / var 변수이름: 변수 Type) {}


class ClassName(val name: String) {
        //....
}
```

Kotlin 생성자

생성자의 함수 원형은 constructor를 써야한다.

constructor는 생략 가능

class ClassName constructor(val name: String) {}

다중 생성자

Java 다중 생성자

```
public class ClassName {
        public ClassName(String name) {
        //…
        }
        public ClassName(String name, int age) {
                this(name);
                //….
        }
}
```

Kotlin 다중 생성자

```
class ClassName(val name: String) {

        constructor(name: String, age: Int) : this(name) {

                //,...

        }

}  //  => age에는 접근이 불가능
```

Kotlin 다중 생성자

```
class ClassName(val name: String) {

        var age : Int = 0

        constructor(name: String, age: Int) : this(name) {

                this.age = age

        }

}
```

생성자 초기화

Java 생성자 초기화

```
public class ClassName {
        private String name;

        // 생성자 초기화
        public ClassName() {
                name = “ClassName”;
        }
}
```

Kotlin 생성자 초기화

```
class ClassName (name: String) {
        // 생성자 초기화 블록
        init {
                name =“ClassName”
        }
}
```

Kotlin 생성자 초기화

```
class ClassName (name: String){

        val upperName = name.toUpperCase()

}
```

생성자 private

Java 생성자 private

```
public class ClassName {

        private ClassName() {

                //...

        }

}
```

Kotlin 생성자 private

```
public ClassName private constructor()

        //....

}
```

클래스 사용하기

Java 클래스 사용

```
public class AAA{

        //....

}

AAA classAAA = new AAA();
```

Kotlin 클래스 사용

```
class AAA{

        //....

}

val aaa: AAA = AAA()

val bbb: AAA()
```

data class

BOOSTCAMP

Kotlin – data class

data class User(val id: Long, val name: String)

- constructor
- getter / setter
- hashcode / equals
- toString
- componentN() functions
- copy() function

Overriding(함수 재정의)

Java Overriding(함수 재정의)

Java는 기본적으로 함수 재정의를 허용

```
public /* final */ class Test {          // final 여부에 따라 재정의가능/불가
        public void methodA() {          // 함수 재정의 가능
                //…
        }
        public final void methodB() {// 함수 재정의 불가
                //..
        }
}
```

Kotlin Overriding(함수 재정의)

Kotlin은 기본적으로 함수 재정의 불가

-> 재정의를 허용하기 위해서는 open 키워드를 사용

Kotlin Overriding(함수 재정의)

Kotlin은 기본적으로 함수 재정의 불가

-> 재정의를 허용하기 위해서는 open 키워드를 사용

```
open class Test {                          // 재정의 가능
        open fun methodA() { // ... }      // 재정의 가능
        fun methodB() { // ...}            // 재정의 불가
}
```

abstract

Java abstract class

```
public abstract class Test {
        public String attributeA;
        public String attributeB = “AttributeB”;
        public abstract void methodA();
        public String methodB() {
                return attributeA;
        }
}
```

Kotlin abstract class

```
abstract class Test {
        abstract var attributeA: String
        var attributeB: String = “AttributeB”
        abstract fun methodA()
        fun methodB(): String {
                return attributeA
        }
}
```

interface

Java interface

```
public interface Test {
        final String attributeA = null; // final 이 생략 가능함
        void methodA();
        String methodB();
}
```

Kotlin interface

```
interface Test {
        // interface에서 변수 선언시 final이 아니며 구현해야됨
        var attributeA: String
        var attributeB: String
        fun methodA()
        fun methodB(): String
}
```

Java8 virtual extension methods

```
interface A {
        void a();                          // Must also implement
        void b() default { //... };        // May implement, overide b()
        void c() final {//...};            // Cannot override c()
}
```

Kotlin interface

```
interface Test2 {

        fun methodA() { //..... }      // interface에서 함수 정의 가능

        fun methodB(): String

}
```

다중 상속

Java 다중 상속

```
class MultiInheritance extends Parent implements Inter1, Inter2 {
        //..
}
```

Kotlin 다중 상속

```
class MultiInheritance : Parent(), Inter1, Inter2 {

        //..

}
```

Kotlin 다중 상속

open class의 open method와 interface의 같은 클래스명을 다중상속 받는다면 super<Base>를 통해서 각각의 클래스를 호출 가능

Kotlin 다중 상속

```
open class OpenA {
        open fun methodA() {println("openA methodA") }
}
interface InterfaceB {
        fun methodA() {println("InterfaceB methodA") }
}
class Test : OpenA(), InterfaceB {
        overide fun methodA() {
                super<OpenA>.methodA();
                super<InterfaceB>.methodA();
        }
}
```

실습

https://codelabs.developers.google.com/codelabs/build-your-first-android-app-kotlin/index.html?index=..%2F..%2Findex#0

감사합니다