راهنمای آموزش برنامه‌نویسی با زبان کوتلین

کوتلین در مقایسه با زبان‌های برنامه‌نویسی امروزی یک زبان جدید محسوب می‌شود که البته قرار است جایگزین زبان جاوا در ساخت برنامه‌های اندرویدی شود. برای بسیاری از کاربران این سوال مطرح می‌شود که اساسا اولین گام در یادگیری زبان برنامه‌نویسی کوتلین چیست و چگونه باید با این زبان برنامه‌نویسی کنیم. در این مقاله قصد داریم به شکل مختصر اصول اولیه برنامه‌نویسی با زبان کوتلین را به شما آموزش دهیم.

کوتلین چیست؟

کوتلین زبانی چند منظوره و متن‌باز است که قابلیت‌های برنامه‌نویسی شی‌گرا و تابعی را با یکدیگر ترکیب کرده تا قدرت برنامه‌نویسان در ساخت برنامه‌های مبتنی بر این زبان را افزایش دهد. فلسفه ساخت زبان کوتلین، خلق یک زبان برنامه‌نویسی بود که همانند زبان جاوا به سرعت کامپایل شود. در فوریه سال 2012 میلادی، شرکت جت‌برینز پروژه فوق را تحت مجوز آپاچی 2 را به شکل متن‌باز منتشر کرد.

چرا باید زبان کوتلین را یاد بگیریم؟

بیشتر توسعه‌دهندگان برنامه‌های اندرویدی که از جاوا برای ساخت برنامه‌های خود استفاده می‌کنند، این پرسش را مطرح می‌کنند که چرا باید از کوتلین به جای جاوا در ساخت برنامه‌های اندرویدی استفاده کرد؟ اولین دلیل است که گوگل همزمان با انتشار اندروید استودیو نگارش 3 که در سال 2017 میلادی از آن رونمایی کرد اعلام کرد که کوتلین به عنوان یک جایگزین برای کامپایلر جاوا در نظر گرفته شده است. کامپایلر کوتلین اندروید به توسعه‌دهندگان اجازه می‌دهد میان بایت‌کدهای منطبق با جاوا 6 یا 8 گزینه مدنظر خود را انتخاب کنند. تقریبا از اوایل سال جاری میلادی بود که کوتلین به تدریج از سوی توسعه‌دهندگان اندرویدی به جای جاوا انتخاب شد و حتا گوگل نیز تلویحا اعلام کرده است که توسعه‌دهندگان بهتر است برای ساخت برنامه‌های اندرویدی خود از این زبان استفاده کنند. از مهم‌ترین ویژگی‌های این زبان برنامه‌نویسی می‌توان به سازگاری (سازگاری با JDK6) به منظور پشتیبانی از دستگاه‌های قدیمی، عملکرد (یکسان بودن عملکرد با جاوا)، زمان کامپایل و سادگی یادگیری به ویژه افرادی که از زبان‌های برنامه‌نویسی دیگر استفاده می‌کنند اشاره کرد. خوشبختانه مبدل جاوا به کاتلین در اندروید استودیو قرار گرفته تا روند یادگیری بیش از پیش ساده شود.

ساختار برنامه‌نویسی با کوتلین چگونه است؟

کوتلین یک زبان مبتنی بر کامپایلر است و در نتیجه شبیه به زبان جاوا است. به عبارت دقیق‌تر، پیش از آن‌که بتوانید کدهای کوتلین را اجرا کنید، کدها ابتدا باید کامپایل شوند. اما کدها چگونه کامپایل می‌شود؟ سورس کدهای کوتلین در فایل‌هایی با فرمت فایلی .kt ذخیره می‌شوند. کامپایلر کوتلین به تحلیل کدها می‌پردازد و فایل‌های .class را ایجاد می‌کند، درست مشابه با کاری که کامپایلر جاوا انجام می‌دهد. در مرحله بعد فایل‌های .class ایجاد و بسته‌بندی می‌شوند از طریق یک روال استاندارد برای نوع برنامه‌ای که در حال ساخت آن هستید آماده اجرا می‌شوند. شکل

زیر این موضوع را نشان می‌دهد.

کوتلین چه تفاوتی با جاوا دارد؟

اصلی‌ترین پرسش پیرامون انتخاب کوتلین به عنوان یک زبان برنامه‌نویسی قدرتمند تفاوت آن با جاوا است. از مهم‌ترین تفاوت‌های این دو زبان به موارد زیر می‌توان اشاره کرد:

چاپ کردن در کنسول

دستورات زیر برای چاپ یک محتوای متنی در جاوا استفاده می‌شوند:

```
System.out.print("Hamid Reza taebi");

System.out.println("Hamid Reza taebi ");
```

در کوتلین فرمان چاپ به شرح زیر استفاده می‌شود:

```
print("Hamid Reza taebi ")

println("Hamid Reza taebi ")
```

ثابت‌ها و متغیر

*var (Mutable variable)

*val (Immutable variable)

جاوا

```
String name = "Hamid Reza taebi";

final String name = "hamid reza taebi";
```

معادل این دستورات در کوتلین به شرح زیر است:

```
var name = "Hamid Reza Taebi"

val name = "Hamid Reza Taebi"
```

نوع‌های داده‌ای

نوع‌های داده‌ای اشاره به نوع و اندازه داده‌های مرتبط با متغیرها و توابع دارند. نوع داده‌ای برای تعریف مکانی که یک متغیر در حافظه ذخیره شده و همچنین مشخص کردن ویژگی‌های داده‌ها استفاده می‌شود. در کولتین هر چیزی یک شی است، به این معنا که می‌توانیم یک تابع عضو و خاصیت‌های روی هر متغیر را فراخوانی کنیم.

- Number
- Character
- Boolean
- Array
- String

انتساب مقدار تهی

در زبان جاوا

```
String otherName;

otherName = null;
```

معادل دستورات در کوتلین به شرح زیر است

```
var otherName: String?

otherName = null
```

بررسی تهی بودن یک مقدار

در جاوا

```
 if (text != null) {

   int length = text.length();

 }
```

در کوتلین

```
 text?.let

{

    val length = text.length

 }
```

یا به شکل ساده‌تر

```
 val length = text?.length
```

چسباندن رشته‌ها

در جاوا

```
 String firstName = "Hamid";

 String lastName = "Reza";

 String message = "My name is: " + firstName + " " + lastName;
```

در کوتلین

```
 var firstName = "Hamid"
```

```
var lastName = "Reza"

var message = "My name is: $firstName $lastName"
```

ساخت خط جدید در یک رشته

در جاوا

```
String text = "First Line\n" +
              "Second Line\n" +
              "Third Line";
```

در کوتلین

```
val text = """
        |First Line
        |Second Line
        |Third Line
        """.trimMargin()
```

عملگر سه‌تایی

در جاوا

```
String text = x > 5 ? "x > 5" : "x <= 5";


String message = null;
log(message != null ? Message : "");
```

معادل آن در کوتلین

```
val text = if (x > 5) "x > 5" else "x <= 5"


val message: String? = null
log(message ?: "")
```

عملگرهای بیتی

در جاوا

```
final int andResult  = a & b;
final int orResult   = a | b;
final int xorResult  = a ^ b;
final int rightShift = a >> 2;
final int leftShift  = a << 2;
final int unsignedRightShift = a >>> 2;
```

در کوتلین

```
val andResult  = a and b
val orResult   = a or b
val xorResult  = a xor b
val rightShift = a shr 2
val leftShift  = a shl 2
val unsignedRightShift = a ushr 2
```

بررسی نوع و تبدیل

در جاوا

```
if (object instanceof Car) {
}
Car car = (Car) object;
```

در کوتلین

```
if (object is Car) {
}
var car = object as Car
// if object is null
var car = object as? Car
// var car = object as Car?
```

شرایط چندگانه (حالتSwitch)

جاوا

```
 int score = // some score;
 String grade;
 switch (score)
}
 case 10:
 case 9:
 grade = "Excellent";
 break;
 case 8:
 case 7:
 case 6:
 grade = "Good";
 break;
 case 5:
 case 4:
 grade = "OK";
 break;
 case 3:
 case 2:
 case 1:
 grade = "Fail";
 break;
 default:
     grade = "Fail";
 {
```

در کوتلین

```
 var score = // some score
 var grade = when (score(
}
 9, 10 -> "Excellent"
 in 6..8 -> "Good"
 4, 5 -> "OK"
 in 1..3 -> "Fail"
 else -> "Fail"
 {
```

حلقه for

جاوا

```
for (int i = 1; i <= 10 ; i++) { }

for (int i = 1; i < 10 ; i++) { }

for (int i = 10; i >= 0 ; i--) { }

for (int i = 1; i <= 10 ; i+=2) { }

for (int i = 10; i >= 0 ; i-=2) { }

for (String item : collection) { }

for (Map.Entry<String, String> entry: map.entrySet()) { }
```

در کوتلین

```
for (i in 1..10) { }
```

```
for (i in 1 until 10) { }
```

```
for (i in 10 downTo 0) { }
```

```
for (i in 1..10 step 2) { }
```

```
for (i in 10 downTo 0 step 2) { }
```

```
for (item in collection) { }
```

```
for ((key, value) in map) { }
```

مجموعه‌ها

جاوا

```
final List<Integer> listOfNumber = Arrays.asList(1, 2, 3, 4);
final Map<Integer, String> keyValue = new HashMap<Integer, String>();
map.put(1, "Amit");
map.put(2, "Ali");
map.put(3, "Mindorks");
// Java 9
final List<Integer> listOfNumber = List.of(1, 2, 3, 4);
final Map<Integer, String> keyValue = Map.of(1, "Amit",
                                             2, "Ali",
                                             3, "Mindorks");
```

کوتلین

```
val listOfNumber = listOf(1, 2, 3, 4)
```

```
val keyValue = mapOf(1 to "Amit",
                     2 to "Ali",
                     3 to "Mindorks")
```

نحوه تعریف متدها

جاوا

```
void doSomething() {
   // logic here
}
```

کوتلین

```
fun doSomething() {
   // logic here
}
```

تعداد آرگومان‌های متغیر

جاوا

```
void doSomething(int... numbers) {
   // logic here
}
```

کوتلین

```
fun doSomething(vararg numbers: Int) {
   // logic here
}
```

تعریف متدها با مقدار بازگشتی (return)

جاوا

```
int getScore() {
   // logic here
   return score;
```

```
}
```

کوتلین

```
fun getScore(): Int {
    // logic here
    return score
}
// as a single-expression function
fun getScore(): Int = score
// even simpler (type will be determined automatically)
fun getScore() = score // return-type is Int
```

ویژگی‌های در دسترس در زبان کاتلین

همان‌گونه که مشاهده کردید، کوتلین در بیشتر بخش‌ها کدها را کوچک‌تر و خواندن دستورات را ساده‌تر کرده است. با این وجود یکسری ویژگی‌های مختص کوتلین هستند که از آن جمله به موارد زیر می‌توان اشاره کرد.

مقداردهی اولیه با تاخیر

مقداردهی همراه با تاخیر (by lazy) زمانی کاربر پیدا می‌کند که در حال کدنویسی خاصیت‌هایی هستیم که ویژگی فقط خواندنی دارند و باید به شکل با تاخیر در کوتلین اجرا شوند. قالب by lazy{...} مقداردهی اولیه را به جای آن‌که در زمان اعلان کردن انجام دهد، در مکانی که خاصیت اولین بار استفاده می‌شود انجام می‌دهد.

```
class Demo { val myName: String by lazy { "John" } }
```

مقداردهی با کمی تاخیر

مقداردهی با کمی تاخیر به حالتی اشاره دارد که توسعه‌دهندگان متغیرها را پیش از آن‌که دسترسی به آن‌ها انجام شود مقداردهی اولیه خواهند کرد.

```
class Demo { val myName: String by lazy { "John" } }
```

کلاس داده

ما همواره در حال ساخت کلاس‌هایی هستم که تنها برای نگه‌داری داده‌ها استفاده می‌شوند. در این مدل کلاس‌ها برخی عملکرهای استاندارد از داده‌ها مشتق می‌شوند. در کوتلین، کلاس داده مسئولیت انجام این‌کار را داشته و به صورت داده‌ای علامت‌گذاری می‌شود.

```
data class Developer(val name: String, val age: Int)
```

هر زمان کلاسی به صورت کلاس داده علامت‌گذاری شود، ضرورتی ندارد که توابع را در آن پیاده‌سازی یا ایجاد کنیم. (شبیه به کاری که در زبان جاو انجام می‌دهیم.)

- ()hashCode
- ()equals

- ()toString
- ()copy

کامپایلر به شکل خودکار توابع فوق را ایجاد می‌کند.

کلاس‌های Sealed

کلاس‌های Sealed با هدف نشان دادن سلسله مراتب کلاس محدودشده استفاده می‌شوند. کلاس‌هایی که در آن‌ها شی یا مقدار می‌توانند تنها یک نوع داشته باشند.

```
sealed class Operation {

    class Add(val value: Int) : Operation()

    class Substract(val value: Int) : Operation()

    class Multiply(val value: Int) : Operation()

    class Divide(val value: Int) : Operation()

}
```

توابع گسترشی

توابع گسترشی اجازه می‌دهند تا عملکرد یک کلاس را با اضافه کردن قابلیت‌های جدید گسترش دهیم. کلاس همیشه ماهیتی نیست که ما ایجاد کرده‌ایم، بلکه می‌تواند کتابخانه شخص ثالث باشد و نیازی ندارد از کلاسی ارث‌بری داشته باشد.

```
fun Int.triple(): Int {

  return this * 3

}
```

در این مقاله سعی کردیم به شکل کاملا فشرده و مختصر شما را با کم و کیف و ساختار کلی کوتلین آشنا کنیم.

تاریخ انتشار:

22 مهر 1398

نشانی منبع:
https://www.shabakeh-mag.com/workshop/programming/16144/%D8%B1%D8%A7%D9%87%D9%86%D9%85%D8%A7%DB%8C-%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87%E2%80%8C%D9%86%D9%88%DB%8C%D8%B3%DB%8C-%D8%A8%D8%A7-%D8%B2%D8%A8%D8%A7%D9%86-%DA%A9%D9%88%D8%AA%D9%84%DB%8C%D9%86