Kotlin

آموزش صفر تا صد زبان برنامه نویسی کاتلین +pdf
(قسمت اول)

کاتلین یک زبان برنامه نویسی متن باز مثل جاوا ، جاوا اسکریپت و غیره برای طراحی اپلیکیشن موبایل است. آموزش کاتلین که یک زبان نگارشی سطح بالا و به شدت آماری است که بخش های تکنیکی و عملی را در یک مکان ارائه می دهد. در حال حاضر هدف کاتلین جاوا و جاوا اسکریپت است. کاتلین روی JVM اجرا می شود.

آموزش کاتلین از سایر زبان های برنامه نویسی مثل جاوا، , Scala, Groovy, Gosu و غیره تاثیر پذیرفته است. نحوه ی نوشتن کاتلین ممکن است دقیقا مثل جاوا نباشد، ولی، کاتلین از درون به کتابخانه ی رده ی جاوا متکی است تا نتایج خارق العاده ای را برای برنامه نویس به ارمغان بیاورد. کاتلین قابلیت انتقال اطلاعات، امنیت کدنویسی، و شفافیت را برای توسعه دهنده ها بوجود می آورد.

نکات مثبت و منفی برنامه نویسی کاتلین

در ادامه بخشی از نکات مثبت استفاده از کاتلین برای توسعه ی نرم افزار خود را می بینید.

زبان ساده – کاتلین یک زبان کاربردی و بسیار راحت برای یادگیری است. نحوه ی نوشتن آن به شدت شبیه جاواست، در نتیجه بسیار راحت به یاد آورده می شود. کاتلین بیان بیشتری دارد و از این جهت کد شما قابل فهم تر و قابل خواندن می شود.

مختصر – کاتلین بر اساس JVM است و زبانی کاربردی است. در نتیجه تعداد کد های بی کاربرد که در زبان های برنامه نویسی دیگر استفاده می شوند، کم می شود.

زمان اجرا و عملکرد – عملکرد بهتر و زمان اجرای کم

قابلیت انتقال اطلاعات – کاتلین به حدی پیشرفته است که بتوان با آن یک نرم افزار انتقال اطلاعات با راهی نه چندان پیچیده ساخت.

جدید – کاتلین یک زبان جدید است که به توسعه دهنده ها شروعی تازه می دهد. جایگزینی برای جاوا نیست، از آنجا که روی JVM توسعه داده شده است. این زبان به عنوان اولین زبان رسمی توسعه ی اندروید مورد قبول واقع شده است. کاتلین را می شود به شکل زیر تعریف کرد :

آموزش کاتلین = جاوا + ویژگی های جدید و آپدیت شده.

در ادامه تعدادی از معایب کاتلین را می بینیم.

فضای نام – کاتلین به توسعه دهنده ها اجازه می دهد که عملکرد ها را در سطح بالا اعلام کنند.

اما، هر وقت همان عملکرد در بخش های مختلفی از نرم افزار درخواست شود، آنوقت فهمیدن این که کدام درخواست خواسته شده است سخت می شود.
عدم اعلام ثابت – تغییر دهنده ی بررسی کننده ی ثابتی مثل جاوا ندارد، که می تواند برای کسی که توسعه دهنده ی جاوا است مشکل ساز شود.

نصب محیطی

به هر طریق، اگر هنوز می خواهید از کاتلین آفلاین رو سیستم محلی تان استفاده کنید، باید مراحل زیر را طی کنید تا سیستم کحل کارتان را پیکر بندی کنید.
مرحله ی 1 : نصب جاوا 8
کاتلین روی JVM کار می کند، در نتیجه بسیار ضروری است که از JDK8 برای توسعه کاتلین تان استفاده کنید. برای دانلود و نصب JDK8 به وبسایت رسمی oracle مراجعه کنید.احتمالا متغیر محیط را برای جاوا به گونه ای تنظیم کنید که به خوبی کار کند.برای تایید نصب در سیستم عامل ویندوزتان عبارت : java-version را در command prompt وارد کنید که در خروجی به شما نسخه ی جاوایی که روی سیستمتان نصب است را می دهد.
مرحله ی 2 : نصب IDE
تعداد زیادی IDE روی اینترنت وجود دارد. می توانید از هر کدام که می خواهید استفاده کنید. لینک دانلود IDE های مختلف در جدول زیر موجود است. توصیه می شود که همیشه از آخرین نسخه ی نرم افزار استفاده شود تا بیشترین روانی را داشته باشد.
مرحله ی 3
Eclipse مشخص کردن :
Eclipse را باز کنید و به Eclipse Market Place بروید. صفحه ی زیر را می بینید.
Kotlin را در نوار جستجو جستجو کنید و همین را روی سیستم محلیتان نصب کنید. با توجه به سرعت اینترنتتان ممکن است کمی طول بکشد. احتمالا باید بعد از نصب Eclipse خود را ری استات کنید.
مرحله ی 4: پروژه ی Kotlin
زمانی که Eclipse با موفقیت ری استارت شد و کاتلین نصب شده است، می توانید پروژه ی کاتلین بسازید. به file سپس new سپس others بروید و Kotlin project را از لیست انتخاب کنید.
زمانی که نصب پروژه تمام شد می توانید یک پوشه ی کاتلین زیر فولدر SRC ایجاد کنید. روی فولدر SRC کلیک چپ بزنید و new را انتخاب کنید. یک انتخاب برا فایل کاتلین به شما داده می شود، در غیر این صورت باید از others جستجو کنید. وقتی فایل جدید ساخته شد، راهنمای پروژه ی شما به شکل زیر خواهد بود.

اکنون محیط توسعه ی شما آماده است. کد زیر را در فایل Hello.kt وارد کنید.

```
fun main(args: Array<String>)
{

("!println("Hello, World

}
```

آن را به عنوان یک Kotlin application اجرا کنید و نتیجه را در میز فرمان همانطور که در تصویر زیر نشان داده شده است ببینید. ما برای فهم و دسترسی بهتر، از ابزار های پایه ی کد نویسی خودمان استفاده می کنیم.

ساختار

کاتلین یک زبان برنامه نویسی موبایل است و برای تخصیص دادن حافظه و تهیه کردن خروجی کیفیت برای کاربر نهایی ساختار خود را داراست. در ادامه سناریو های مختلفی را شاهد هستیم که در آن برنامه ی مترجم کاتلین هر زمان که زبان های دیگری مثل جاوا و جاوا اسکریپ را هدف خود قرار می دهد ، به شکل های مختلف کار می کند.

برنامه ی مترجم کاتلین یک کد بایت می سازد و آن کد بایت می تواند روی JVM اجرا شود. که دقیقا همان چیزی است در کد بایت جاوا .class file انجام می دهد. هر وقت دو فایل کد شده ی بایت روی JVM اجرا می شوند، می توانند با یکدیگر ارتباط برقرار کنند و این گونه است که یک ویژگی انتقال اطلاعات در کاتلین برای جاوا ایجاد می شود.

هر وقت کاتلین جاوا اسکریپت را هدف می گیرد، برنامه ی ترجمه ی کاتلین فایل .kt را به ES5.1 تبدیل می کند و کدی سازگار برای جاوا اسکریپت تولید می کند. برنامه ی ترجمه ی کاتلین توانایی ساخت کد های سازگار بر پایه پایگاه با LLVM را داراست.

نمونه های اولیه

در این بخش، درباره ی نمونه های داده ی اولیه در زبان برنامه نویسی کاتلین حرف می زنیم.

اعداد

نمایش اعداد در کاتلین به شدت شبیه جاواست. اما آموزش کاتلین اجازه ی تبدیل درونی نمونه های داده ی مختلف را نمی دهد. جدول پیش رو درازای متغیر مختلف برای اعداد مختلف را نشان می دهد.

در مثال زیر، می بینیم که چطور کاتلین با نمون داده های مختلف کار می کند. لطفا کد های زیر را در محل کد نویسی ما وارد کنید.

```
fun main(args: Array<String>) { val a: Int = ۱۰۰۰۰ val d: Double = ۱۰۰/۰۰ val f: Float = ۱۰۰/۰۰f val l: Long = ۱۰۰۰۰۰۰۰۰۴ val s: Short = ۱۰ val b: Byte = ۱ println("Your Int Value is "+a); println("Your Double Value is "+d); println("Your Float Value is "+f); println("Your Long Value is "+l); println("Your Short Value is "+s); println("Your Byte Value is "+b;}
```

وقتی کد های بالا را در محل کد نویسی اجرا کنید، خروجی زیر را در میز فرمان وب وارد تولید می کند.

Your Int Value is 10000Your Double Value is 100.0Your Float Value is 100.0Your Long Value is 1000000004Your Short Value is 10Your Byte Value is 1

علامت ها

کاتلین با استفاده از char. علامت ها را نشان می دهد. علامت باید داخل داخل یک کاما مثل 'c' گفته شود. لطفا کد زیر را در محل کد نویسی وارد کنید و ببینید که کاتلین چطور متغیر علامت را تفسیر می کند. متغیر علامت نباید مانند متغیر اعداد نوشته شود. متغیر کاتلین به دو شکل می تواند نوشته شود. یکی با استفاده از "var" و دیگری با استفاده از "val" .

```
fun main(args: Array<String>) {  val letter: Char   // defining a variable   letter = 'A'     // Assigning a value to it   println("$letter")}
```

کد های بالا نتیجه ی زیر را در پنجره ی خروجی مرورگر نشان می دهد.

A

Boolean

بولی خیلی ساده است مثل بقیه ی زبان های برنامه نویسی. تنها دو مقدار برای بولی داریم. یا درست یا غلط. در مثال زیر، می بینیم که چطور کاتلین بولی را تفسیر می کند.

```
fun main(args: Array<String>) {  val letter: Boolean   // defining a variable   letter = true       // Assinging a value to it   println("Your character value is "+"$letter")}
```

کد های بالا نتیجه ی زیر را در مرورگر تولید می کند.

Your character value is true

Strings

رشته ها آرایه های علامت ها هستند. مثل جاوا، ذاتا تغییر ناپذیر هستند. ما نوع رشته در

کاتلین داریم: یکی رشته ی خام raw String و دیگری رشته ی گریخته escaped String . در مثال زیر، از این رشته ها استفاده می کنیم.

```
fun main(args: Array<String>) {   var rawString :String  = "I am Raw String!"
   val escapedString : String = "I am escaped String!\n"
 println("Hello!"+escapedString)  println("Hey!!"+rawString)  }
```

مثال بالا از ریشه های گریخته اجازه می دهد تا فاصله ی خط بیشتری پس از اولین دستور چاپ ایجاد شود. نتیجه ی زیر در مرورگر ایجاد می شود.

```
Hello!I am escaped String! Hey!!I am Raw String!
```

Arrays

آرایه ها مجموعه از داده ی مشابه هستند. مثل جاوا، کاتلین از آرایه های نمونه داده های مختلف پشتیبانی می کند. در مثال زیر، از آرایه های مختلفی استفاده می کنیم.

```
fun main(args: Array<String>) {   val numbers: IntArray = intArrayOf(1, 2, 3, 4,
5)   println("Hey!! I am array Example"+numbers[2])}
```

کد های بالا خروجی زیر را تولید می کنند. فهرست کردن آرایه ها مثل بقیه ی زبان های برنامه نویسی است. در اینجا به دنبال یک فهرست دوم هستیم، که مقدار آن ۳ است.

```
Hey!! I am array Example3
```

مجموعه ها

مجموعه بخش بسیار مهمی از ساختار داده است، که توسعه ی نرم افزار را برای مهندسین ساده می کند. کاتلین دو نوع مجموعه دارد: یکی مجموعه ی تغییر ناپذیر (که به معنی لیست ها، نقشه ها و دستگاه هایی اند که قابل ویرایش دادن نیستند) و دیگر مجموعه ی تغییر پذیر(این نوع مجموعه قابل ویرایش است). بسیار مهم است که نوع مجموعه ای که در نرم افزارتان استفاده می شود را در ذهن داشته باشید، چرا که سیستم کاتلین تفاوتی بین آن ها را نشان نمی دهد.

```
fun main(args: Array<String>) {   val numbers: MutableList<Int> =
mutableListOf(1, 2, 3) //mutable List   val readOnlyView: List<Int> = numbers
// immutable list   println("my mutable list--"+numbers)      // prints "[1, 2, 3]"
 numbers.add(4)   println("my mutable list after addition --"+numbers)      //
prints "[1, 2, 3, 4]"   println(readOnlyView)      readOnlyView.clear()   // ⇒
does not compile  // gives error  }
```

کد های بالا نتیجه ی زیر را در مرورگر به نمایش خواهند گذاشت. وقتی می خواهیم لیست مجموعه ی قابل تغییر را پاک کنیم خطا می دهد.

```
main.kt:9:18: error: unresolved reference: clear  readOnlyView.clear()  // ->
                                                  ^
does not compile
```

در مجموعه، کاتلین متد های بدرد بخوری مثل filter ,last(), first(), و غیره ارائه می دهد.
همه ی این متد ها خود شرح دهنده اند و اجرا کردنشان راحت است. علاوه بر این، کاتلین از همان ساختار جاوا حین اجرا کردن مجموعه استفاده می کند. می توانید هر نوع مجموعه ای که می خواهید را اجرا کنید مانند نقشه و دستگاه.
در مثال زیر نقشه و دستگاه را با استفاده از روش های غیر قابل انتقال اجرا کرده ایم.

```
fun main(args: Array<String>) {  val items = listOf(1, 2, 3, 4)  println("First Element of our list----"+items.first())  println("Last Element of our list----"+items.last())  println("Even Numbers of our List----"+items.  filter { it % 2 = = 0 })  // returns [2, 4]  val readWriteMap = hashMapOf("foo" to 1, "bar" to 2)  println(readWriteMap["foo"])  // prints "1"  val strings = hashSetOf("a", "b", "c", "c")  println("My Set Values are"+strings)}
```

کد های بالا خروجی زیر را در مرورگر نشان می دهد.

```
First Element of our list—-1Last Element of our list—-4Even Numbers of our List—-[2, 4]1My Set Values are[a, b, c]
```

دامنه ها

دامنه ها یکی دیگر از ویژگی های خاص کاتلین است. مثل Haskell ، اپراتوری ایجاد می کند که به شما کمک می کند داخل یک دامنه تکرار کنید. ذاتا برای اجرا از rangeTo() استفاده کنید و شکل اپراتور آن نیز (..) است.
در مثال زیر، می بینیم که کاتلین چگونه این اپراتور دامنه را اجرا می کند.

```
fun main(args: Array<String>) {  val i:Int  = 2  for (j in 1..4)  print(j) // prints "1234"  if (i in 1..10) { // equivalent of 1 <= i && i <= 10  println("we found your number --"+i)  }}
```

کد های بالا خروجی زیر را در مرورگر به نمایش می گذارد.

روند کنترل

در بخش قبل از آموزش کاتلین در مورد نمونه داده های مختلف موجود در سیستم کاتلین یاد گرفتیم. در این بخش، در مورد نمونه های مختلف مکانیزم روند کنترل موجود در کاتلین بحث می کنیم.

If – Else

کاتلین یک زبان برنامه نویسی کاربردیست زیرا مانند همه ی زبان های کاربردی در آموزش کاتلین دستور "if" یک عبارت است نه فقط یک لغت. عبارت If مقداری را هر زمان که نیاز باشد باز می گرداند. مانند زبان های برنامه نویسی دیگر،بلوک "if-else" به عنوان اپراتور چک کردن شرطی اولیه کار می کند. در مثال زیر دو متغیر را مقایسه کرده و به ترتیب خروجی مرود نیاز را نمایش داده شده است.

```
fun main(args: Array<String>) {  val a:Int = 5  val b:Int = 2  var max: Int    if (a > b) {    max = a  } else {    max = b  }  print("Maximum of a or b is " +max)   // As expression   // val max = if (a > b) a else b}
```

کد های بالا خروجی زیر را به عنوان نتیجه در مرورگر نشان می دهند. مثال ما شامل خط کدی دیگری نیز می باشد که نحوه ی استفاده از دستور if به عنوان یک عبارت را نشان می دهد.

```
Maximum of a or b is 5
```

When

اگر با زبان های دیگر برنامه نویسی آشنا باشید، احتمالا با عبارت دستور راه گزیدن آشنا هستید، که به طور کل وقتی که چند شرط به یک متغیر خاص اعمال می شوند به عنوان یک اپراتور شرطی است. اپراتوط When مقدار متغیر را با شرط های شاخه ای تطبیق می دهد. اگر با شرط شاخه مطابقت داشت، عبارت را داخل آن حدود اجرا می کند. در مثال زیر چگونگی استفاده از When در کاتلین را می آموزیم.

```
fun main(args: Array<String>) {  val x:Int = 5  when (x) {    1 -> print("x = = 1")    2 -> print("x = = 2")        else -> { // Note the block      print("x is neither 1 nor 2")    }  }}
```

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

```
x is neither 1 nor 2
```

در مثال بالا برنامه ی مترجم کاتلین مقدار X را با شاخه های داده شده تطبیق می دهد. اگر با هیچ کدام از شاخه ها تطبیق داده نشد، آن گاه بخش ELSE را اجرا می کند. در واقع، When با چند بلوک IF برابری می کند. آموزش کاتلین یک ویژگی دیگر برای توسعه دهنده ها مهیا کرده است که در آن توسعه دهنده می تواند تعدادی چک در یک خط با استفاده از "," داخل چک ها بوجود بیاورد. مثال بالا را به شکل زیر توضیح می دهیم.

```
fun main(args: Array<String>) {   val x:Int = 5   when (x) {      1,2 -> print("
Value of X either 1,2")          else -> { // Note the block        print("x is neither 1
nor 2")     }   }}
```

همین را در مرورگر اجرا کنید. که خروجی زیر را در مرورگر نشان می دهد.

```
x is neither 1 nor 2
```

حلقه For

حلقه اختراعی است که قابلیت انعطاف پذیری برای تکرار کردن داخل هر نوع ساختار داده را یوجود می آورد. مانند دیگر زبان های برنامه نویسی، کاتلین هم روش های مختلفی برای ایجاد حلقه ارائه می دهد. اما در بین آن ها For از همه موفق تر است. اجرا و استفاده از حلقه ی For از لحاظ مفهومی شبیه به جاوا است. مثال زیر نشان می دهد چگونه می توانیم از همین در مثال های زندگی واقعی استفاده کنیم.

```
fun main(args: Array<String>) {   val items = listOf(1, 2, 3, 4)   for (i in items)
println("values of the array"+i)}
```

در کد های بالا یک لیست به نام ITEMS معرفی کرده ایم. و برای استفاده در حلقه در حال اجرا میان همان لیست تعریف شده هستیم و مقدارش را در مرورگر نشان می دهیم. خروجی به شکل زیر است.

```
values of the array1values of the array2values of the array3values of the
array4
```

در زیر مثال دیگری از کد ، در حالی که از کاربرد کتابخانه ای استفاده می کنیم تا کار توسعه ی مان راحت تر از قبل باشد نشان داده شده است.

```
fun main(args: Array<String>) {   val items = listOf(1, 22, 83, 4)     for ((index,
value) in items.withIndex()) {      println("the element at $index is $value")   }}
```

وقتی کد های بالا در محل کد نویسی را جمع آوری و اجرا کنیم، حروجی زیر در مرورگر نشان داده خواهد شد.

```
the element at 0 is 1the element at 1 is 22the element at 2 is 83the element at 3
is 4
```

حلقه WHILE و DO-WHILE

WHILE و DO-WHILE دقیقا مثل سایر زبان ها برنامه نویسی مثل یکدیگر کار می کنند. تنها تفاوت میان این دو حلقه این است که در صورتی که DO-WHILE در حال اجرا باشد، شرایط در پایان حلقه آزمایش می شود. مثال زیر نحوه ی استفاده از حلقه ی WHILE را نشان می دهد.

```
fun main(args: Array<String>) {  var x:Int = 0  println("Example of While Loop--")    while(x< = 10) {    println(x)    x++  } }
```

کد های بالا خروجی زیر را در مرورگر نشان می دهد.

Example of While Loop–012345678910

کاتلین همچنین یک حلقه ی دیگر به نام DO-WHILE دارد که بدنه ی حلقه یک بار اجرا می شود، تنها آن زمان است که شرط چک می شود. مثال زیر نجوه ی استفاده از حلقه ی DO-WHILE را نشان داده است.

```
fun main(args: Array<String>) {  var x:Int = 0  do {    x = x + 10    println("I am inside Do block---"+x)  } while(x <= 50)}
```

کد های بالا خروجی زیر را در مرورگر نشان می دهند. در کد بالا، برنامه ی مترجم کاتلین بلوک DO را اجرا می کند، سپس می رود سراغ چک کرده شرط ها در بلوک WHILE .

I am inside Do block—10I am inside Do block—20I am inside Do block—30I am inside Do block—40I am inside Do block—50I am inside Do block—60

استفاده از Return, Break, Continue

اگر با هر کدام از زبان های برنامه نویسی آشنا باشید، احتمالا با کلیدواژه های مختلفی که به اجرا خوب روند کنترل در نرم افزار کمک می کنند آشنایی دارید. در ادامه تعدادی از این کلیدواژه ها که برای کنترل حلقه ها یا هر نوع روند کنترل هستند معرفی شده اند.

Return – یک کلیدواژه است که مقداری از کاربرد خواسته شده به کاربرد در حال درخواست باز میگرداند. در مثال زیر این سناریو را با استفاده از محل کدنویسی کاتلین اجرا می کنیم.

```
fun main(args: Array<String>) {  var x:Int = 10  println("The value of X is--"+doubleMe(x))}fun doubleMe(x:Int):Int {  return 2*x;}
```

در کد های بالا، ما در خواست کار دیگری را می دهیم و ورودی را با ۲ دو برابر می کنیم، و برگرداندن مقدار برایند به کار خواسته شده که هدف اصلی مان است.کاتلین کار را به شکل دیگری تعریف می کند که در بخش بعدی به آن می پردازیم. فعلا، کافی است درک کنیم که کد های بالا خروجی زیر را در مرورگر می دهند.

The value of X is–20

Continue & Break

Continue & Break حیاتی ترین بخش یک مسئله ی منطقی هستند. کلیدواژه ی break روند کنترل را در صورتی که شرطی بد کار کند متوقف می کند و Continue عکس این کار را انجام میدهد. همه ی این عملیات با قابلیت رویت در لحظه همراه اند. از آن جایی که توسعه دهنده

می تواند تعداد بیشتر ی مطلب به عنوان با قابلیت رویت ارائه دهد، کاتلین از بقیه ی زبان های برنامه نویسی باهوش تر عمل می کند. کد های زیر نشان می دهند چگونه مطلب زیر را در کاتلین اجرا کنیم.

```
fun main(args: Array<String>) {   println("Example of Break and Continue")
 myLabel@ for(x in 1..10) { // appling the custom label     if(x = = 5) {
 println("I am inside if block with value"+x+"\n-- hence it will close the
operation")        break@myLabel //specifing the label     } else {        println("I
am inside else block with value"+x)        continue@myLabel     }   }}
```

کد های بالا خروجی زیر را در مرورگر نشان خواهند داد.

```
Example of Break and ContinueI am inside else block with value1I am inside
else block with value2I am inside else block with value3I am inside else block
with value4I am inside if block with value5– hence it will close the operation
```

همانطور که می بینید، کنترل کننده حلقه را ادامه میدهد، تا و مگر آن که مقدار x 5 باشد. به محض این که مقدار x به 5 برسد، بولک های if را اجرا می کند و وقتی به عبارت break رسید، کل روند کنترل اجرای برنامه را متوقف می کند.

Class & Object

در این بخش ، اصول اولیه ی برنامه نویسی مقصود گرا با استفاده از کاتلین را یاد میگیریم. در مورد طبقه و مقصودش و این که چطور با آن مقصود کار کنیم می آموزیم. طبق تعریف OOP ، طبقه طرح اولیه ی بخش ران تایم است و مقصود وضعیت آن است، که شامل رفتار و هم وضعیت آن می شود. در کاتلین، اعلام طبقه از سرپیام طبقه و یک بدنه ی طبقه تسکیل شده که است که توسط جفتی مجعد احاطه شده است. شبیه به جاوا.

```
Class myClass { // class Header     // class Body}
```

مثل جاوا، کاتلین اجازه ی ساخت مقصود های مختلفی از یک طبقه را می دهد، و شما مختارید تا اعضای طبقه و کارکرد هایشان را اضافه کنید. می توانیم قابلیت رویت متغیر های اعضای طبقه را با استفاده از کلیدواژه های مختلفی که در بخش ۱۰ (کنترل قابلیت رویت) یاد خواهیم گرفت کنترل کنیم.در مثال زیر، طبقه ای همراه با مقصود هایش می سازیم که به وسیله ی آن ها به اعضای داده ی آن طبقه دسترسی پیدا می کنیم.

```
class myClass {   // property (data member)   private var name: String =
"Tutorials.point"     // member function   fun printMe() {     print("You are at the
best Learning website Named-"+name)   }}fun main(args: Array<String>) {
 val obj = myClass() // create obj object of myClass class   obj.printMe()}
```

کد های بالا خروجی زیر را در مرورگر به نمایش می گذارند، که در حال فرا خواندن printMe() از myClass با استفاده از مقصود هایش هستیم.

You are at the best Learning website Named- Tutorials.point

طبقه آشیانه ای

طبق تعریف، وقتی که طبقه ای درون طبقه ی دیگری ساخته شده باشد به آن طبقه ی آشیانه ای می گویند. در آموزش کاتلین، طبقه ی آشیانه ای به صورت خودکارثابت است، در نتیجه، بدون ساختن مقصودی برای آن طبقه قابل دسترسی است. در مثال زیر، می بینیم که چطور کاتلین طبقه ی آشیانه ای ما را تفسیر می کند.

```
fun main(args: Array<String>) {   val demo = Outer.Nested().foo() // calling nested class method   print(demo)}class Outer {   class Nested {     fun foo() = "Welcome to The TutorialsPoint.com"   }}
```

کد های بالا خروجی زیر را در مرورگر نمایش می دهند.

Welcome to The TutorialsPoint.com

طبقه درونی

وقتی یک طبقه ی آشیانه ای به عنوان inner علامت گذاری شده است به آن طبقه ی درونی می گویند. یک طبقه ی درونی می تواند از طریق اعضای داده ی طبقه ی خارجی قابل دسترس باشد. در مثال زیر، به اعضای داده ی طبقه ی خارجی دسترسی پیدا می کنیم.

```
fun main(args: Array<String>) {   val demo = Outer().Nested().foo() // calling nested class method   print(demo)}class Outer {   private val welcomeMessage: String = "Welcome to the TutorialsPoint.com"   inner class Nested {     fun foo() = welcomeMessage   }}
```

کد های بالا خروجی زیر را در مرورگر به نمایش خواهند گذاشت، که در آن طبقه ی آشیانه ای را با استفاده از سازنده ی پیش فرض کاتلین در زمان اجرا فرا می خوانیم.

Welcome to the TutorialsPoint.com

طبقه داخلی بدون نام

طبقه ی داخلی بدون نام مفهوم بسیار خوبی است که زندگی یک برنامه نویس را به شدت آسان می کند. هر گاه در حال اجرای یک واسط باشیم، مفهوم بلوک داخلی بدون نام به میان می آید. مفهوم ساخت مقصودی واسط با استفاده از مرجع مقصود رانتایم به عنوان طبقه ی بدون نام شناخته می شود. در مثال زیر، یک واسط می سازیم و مقصودی برای آن واسط با استفاده از مکانیزم طبقه ی درونی بدون نام می سازیم.

```
fun main(args: Array<String>) {   var programmer :Human = object:Human //
creating an instance of the interface {     override fun think() { // overriding the
think method       print("I am an example of Anonymous Inner Class ")     }  }
 programmer.think()}interface Human {   fun think()}
```

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

I am an example of Anonymous Inner Class

نام نمونه

نام های نمونه بخشی از برنامه ی مترجم کاتلین هستند. انعطافی برای تولید نامی جدید از نمونه ای موجود ایجاد می کنند، نمونه ای جدید ایجاد نمی کنند. اگر نام نمونه خیلی بلند باشد، به راحتی می توانید نامی کوتاه تر معرفی کرده و از آن برای استفاده های آینده استفاده کنید. نام های نمونه برای نمونه های پیچیده بسیار مفید اند. در آخرین نسخه، کاتلین پشتیبانی از نام های نمونه را لغو کرده است اما اگر از نسخه ها قدیمی استفاده می کنید می توانید به شکل زیر از آن استفاده کنید.

```
typealias NodeSet = Set<Network.Node>typealias FileTable<K> =
<<MutableMap<K, MutableList<File
```

سازنده ها

در این بخش ، در مورد سازنده ها در کاتلین اطلاعاتی کسب می کنیم. کاتلین دو نوع سازنده دارد. یکی سازنده ی اولیه و دیگری سازنده ی ثانویه. یک طبقه ی کاتلین می تواند یک سازنده ی اولیه و یک یا چند سازنده ی ثانویه داشته باشد. سازنده ی جاوا به متغیر های اعضا مقدار می دهد ولی در کاتلین سازنده ی اولیه به طبقه مقدار می دهد، در حالی که سازنده ی ثانویه سعی در منطق دادن در حین همان مقدار دادن می باشد. سازنده ی اولیه می تواند در سطح سرپیام طبقه مانند مثال زیر نشان داده شود.

```
class Person(val firstName: String, var age: Int) {

// class body

}
```

در مثال بالا، به سازنده ی اولیه داخل پرانتز دستور داده ایم. بین دو بخش، نام اول فقط قابل خواندن است همانطور که با "val" اعلام شده است، در حالی که بخش زمینه قابل ویرایش است.

در مثال زیر از سازنده ی اولی استفاده می کنیم.

```
ffun main(args: Array<String>) {

val person1 = Person("TutorialsPoint.com", 15)

println("First Name = ${person1.firstName}")

println("Age = ${person1.age}")

}

class Person(val firstName: String, var age: Int) {

}

}
```

کد های بالا به طور خودکار به دو متغیر مقدار دهی می کنند و خروجی زیر را در مرورگر به نمایش می گذارند.

```
. First Name = TutorialsPoint.com
Age = 15
```

همانطور که قبلا به آن اشاره شده بود، کاتلین اجازه می دهد تا یک یا تعداد بیشتری سازنده ی ثانویه برا طبقه ی خود بسازیم. این سازنده ی ثانویه با استفاده از کلیدواژه ی "constructor" ساخته شده است.لازم است هر وقت می خواهید بیش از یک سازنده در کاتلین داشته باشید یا هر وقت که می خواهد منطق بیشتری به سازنده ی اولیه ی خود اضافه کنیدو به خاطر این که سازنده ی اولیه ممکن است توسط طبقه ی دیگری فراخوانده شده باشد، نمی توانید این کار را کنید. به مثال زیر توجه کنید، که در آن یک سازنده ی ثانویه ساخته ایم و از مثال بالا برای همان اجرا استفاده کرده ایم.

```
fun main(args: Array<String>) {
val HUman = HUman("TutorialsPoint.com", 25)
print("${HUman.message}"+"${HUman.firstName}"+
"Welcome to the example of Secondary constructor, Your Age
is-${HUman.age}")
}
```

```
class HUman(val firstName: String, var age: Int) {
val message:String = “Hey!!!”
constructor(name : String , age :Int ,message :String):this(name,age) {
}
}
```

توجه: هر تعدادی از سازنده های ثانویه قابل ساخت است، اما، همه ی آن سازنده ها باید به سازنده ی اولیه پاسخ گو باشند چه به صورت مستقیم چه غیر مستقیم.

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

```
Hey!!! TutorialsPoint.comWelcome to the example of Secondary constructor,
Your Age is- 25
```

توارث

در این بخش، در مورد توارث می آموزیم. طبق تعریف، می دانیم که توارث یعنی دادن بخش هایی از طبقه ی مادر به طبقه ی فرزند. در کاتلین، طبقه ی مبنا “Any” نام گذاری شده است، که طبقه ی فوق طبقه ی پیش فرض “Any” در آموزش کاتلین است. مانند بقیه ی برنامه نویسی های مقصودگرا، کاتلین هم این کاربرد را با استفاده از کلیدواژه ی “:” نشان میدهد.

همه چیز در کاتلین به طور پیش فرض نهایی است، در نتیجه، احتیاج است از کلیدواژه ی open در جلوی اعلام طبقه استفاده کنیم تا به آن قابلیت وراثت دهد. به مثال زیر توجه کنید.

```
import java.util.Arrays
open class ABC {
fun think ()
{
print("Hey!! i am thiking ")
}
}
class BCD: ABC(){ // inheritence happend using default constructor
}
fun main(args: Array<String>) {
var  a = BCD()
a.think()
}
```

کد های بالا خروجی زیر را در مرورگر به نمایش خواهند گذاشت.

```
Hey!! i am thiking
```

را در طبقه ی ()think حالا در این قسمت آموزش کاتلین، چه می شد اگر می خواستیم روش فرزند غیر فعال کنیم. باید به مثال زیر توجه کنیم که در آن دو طبقه ساخته ایم و یکی از کارکرد های طبقه ی فرزند را غیرفعال کرده ایم.

```
import java.util.Arrays

open class ABC
{
open fun think () {
print("Hey!! i am thinking ")
}
}
class BCD: ABC() { // inheritance happens using default constructor
override fun think() {
print("I Am from Child")
}
}
fun main(args: Array<String>) {
var  a = BCD()
a.think()
}
```

کد های بالا متد توارث طبقه ی فرزند را فرا می خوانند و خروجی زیر را در مرورگر به نمایش می گذارند. مثل جاوا، کاتلین اجازه چند توارث را نمی دهد.

```
I Am from Child
```