توی کاتلین همه چیز شی­هه! اینجا مثل جاوا تایپ های اولیه نداریم و این خیلی خوبه، چون در اینصورت ما یک روش درست حسابی برای برخورد با تایپ های مختلف داریم و اینجوری نیست که بگیم خب الان اینه باید اینکار رو بکنیم و اگر اون بود فلان کار رو . همیشه و همیشه یک روش برخورد با تایپ ها داریم.

**تایپ های پایه ای :**

البته که تایپ های پایه ای مثل integers, float, characters و یا Boolean هنوز وجود دارن ولی همه­ی اینا تو کاتلین یک شی­ان.نام تایپ­های ابتدایی و نحوه برخورد با اونها هنوز مثل جاوایه ولی یک سری تفاوت هایی هست که باید اونهارو درنظر بگیرین:

* هیچ تبدیل اتوماتی بین تایپ­های شمارشی وجود نداره.به عنوان مثال شما نمیتونین یک Int رو به یک متغیر Float بدین .اگه بخواین چنین اعمالی انجام بدین باید با توابع مربوطه به صورت صریح بیان کنین:

val i: Int = 7  
val d: Double = i.toDouble()

* کارکترها (Char) نمیتونن به صورت مستقیم به عنوان تایپ شمارشی استفاده بشن.هرچند هر موقع خواستیم میتونیم با توابع مربوطه اون کارهارو انجام بدیم.

val c: Char = 'c'  
val i: Int = c.toInt()

* عملگرهای محاسباتی بیتی یکم متفاوتن.توی اندروید باید از “and” و یا “or” استفاده کنیم.

// Java  
int bitwiseOr = FLAG1 | FLAG2;  
int bitwiseAnd = FLAG1 & FLAG2;

// Kotlin  
val bitwiseOr = FLAG1 or FLAG2  
val bitwiseAnd = FLAG1 and FLAG2

* همینطور که میدونین توی کاتلین نیازی نیست که مشخص کنین این دقیقا چه تایپی داره، کاتلین خودش میفهمه ولی بعضی اوقات تشخیص دشوار میشه.به عنوان مثال وقتی مقدار 10 یه یک متغیر داده میشه، ایا منظور توسعه دهنده Int بوده یا Double ؟ در اینجا از یک سری حروف استفاده میشه تا کامپایلر متجه تایپ مورد نظر بشه.

val i = 12 // An Int  
val iHex = 0x0f // An Int from hexadecimal literal  
val l = 3L // A Long  
val d = 3.5 // A Double  
val f = 3.5F // A Float

* به یک رشته میشه به مانند یک آرایه دسترسی پیدا کرد و میشه روش حرکت کرد:

val s = "Example"  
val c = s[2] // This is the Char 'a'  
  
// Iterate over String  
val s = "Example"  
for (c in s) {  
 *print*(c)  
}

**متغیرها:**

متغیر ها در کاتلین دوصورت دارن، یا مصون اند(val) یا میتونن تغییر کنن(var) به مانند final توی جاوا. ولی مصون بودن توی کاتلین یک مفهوم خیلی مهمیه. اگه یک متغیر مصون باشه به این معنیه که بعد از مقدارگیری نمیتونه مقدارش تغییر کنه. اگه شما مقدار تغییر یافته از این شی رو میخواین، باید شی جدید ازش بسازین! توی جاوا اکثر متغیرها قابل تغییر بودن، به این معنی که هر قسمت از کد که دسترسی به این متغیر داشت میتونست مقدارش رو عوض کنه.

متغیر های مصون همچنین thread-safe اند. چون نمیتونن تغییر کنن پس هیچ کنترل دسترسی لازم نیست تعریف بشه چون هر نخ در واقع یک شی رو دراختیار میگیره که دیگری گرفته.

**نکته کلید: تاجای ممکن از val استفاده کن.**

>ناقص< نیازی به کامل شدن نداره!

**خصیصه ها:**

بذارین اینطور بگم، خصیصه ها مثل متغیر های field توی جاوا میمونن. خصیصه ها کار field رو انجام میدن.ولی یک تفاوت هایی با هم دارن.به عنوان مصال بیاین تفاوت های این دوتا رو مقایسه کنیم:

public class Person {  
 private String name;  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
}

…

Person person = new Person();  
person.setName("name");  
String name = person.getName();

و توی کاتلین ما تنها یک خصیصه نیازداریم!

public class Person {  
 var name: String = ""  
}  
...  
val *person* = Person()  
person.name = "name"  
val *name* = *person*.name

بدون این که شما یک خط کد بنویسین خصیصه هم getter رو داره و هم setter . البته که شما میتونین این توابع رو شخصی سازی کنین و به صورت دلخواهتون در بیارین.

public class Person {  
 var name: String = ""  
 get() = **field**.*toUpperCase*()  
 set(value) {  
 **field** = "Name: $value"  
 }  
}

اگه یک خصیصه در توابع دلخواه getter و setter خودش، نیاز به دسترسی به مقدار خودش داشت از کلیدواژه­ای به نام field استفاده میشه و به این روش هم backing field میگن.

همینطور که قبلا گفتیم، کاتلین دسترسی به java داره و میتونه باهاش همکاری کنه، همینطور گفتیم که توی یک فایل کاتلین وقتی میخوایم به یک متغیر field دسترسی داشته باشیم نیازی به استفاده از getter و setter نداریم و میتونیم مستقیم به اون field به عنوان یک خصیصه دسترسی داشته باشیم.ولی یادتون باشه که این روش عملکرد رو ضعیف نمیکنه و خود کامپایلر در زمان کامپایل از تابع جاوای درست استفاده میکنه!