بدون این که بیشتر توی زبون کاتلین غرق بشیم ( ما همه چیز رو توی این دوره یاد میگیریم ) لازمه به چند ویژگی که توی کاتلین هست و در جاوا کمبودش احساس میشه اشاره کنیم:

**یک زبون رسا**

با کاتلین شما میتونین با مقدار کمی از کد تموم مقصودی که میخواین رو برسونین و از خیلی از ویژگی های پیشفرض زبون جاوا راحت بشین. به عنوان مثال توی جاوا اگه میخواستیم یک bean و یا یک کلاس دیتا بسازیم لازم بود همه ی این کد هارو بنویسیم:

public class Artist {  
 private long id;  
 private String name;  
 private String url;  
 private String mbid;  
  
 public long getId() {  
 return id;  
 }  
  
 public void setId(long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getUrl() {  
 return url;  
 }  
  
 public void setUrl(String url) {  
 this.url = url;  
 }  
  
 public String getMbid() {  
 return mbid;  
 }  
  
 public void setMbid(String mbid) {  
 this.mbid = mbid;  
 }  
  
 @Override  
 public String toString() {  
 return "Artist{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", url='" + url + '\'' +  
 ", mbid='" + mbid + '\'' +  
 '}';  
 }  
}

ولی توی کاتلین تنها کدی که باید بزنین که همین مقصود رو برسونه اینه :

data class Artist(var id: Long, var name: String, var url: String, var mbid: String)

تنها یک خط کاملا بیانگر تموم خط کد هایی که اون بالا نوشتیم. در حالت عادی در یک کلاس نیازی به وجود getter و setter نداریم و با اضافه کردن کلیدواژه "data" استفاده از توابعی به مانند toString(),hashCode() و equals() رو برامون ممکن میساره! بدون این که یک خط کد بنویسیم. والبته که میتونیم اون هارو با override کردن به تابع دلخواه خودمون دربیاریم.

**Null Safety**

هنوز معادل فارسی درستی برای این مفهوم پیدا نکردم که بتونه درواژه مفهوم رو بیان کنه. در کاتلین مفهوم null وجود نداره! البته نه به این معنا که واقعا وجود نداره، به این معنی که تایپ ها نمیتونن null باشن، و اگر بخوان مقدار null به خودشون بگیرن باید مشخص بشن که null هستند. خب البته ما زمانی که اندروید رو توسعه میدیم باید هواسمون باشه که همونطور که گفتیم بسیاری از کتابخانه هامون جاوا هستند و جاوا این دفاع رو در خودش نداره. پس زمان کدنویسی با استفاده از عملگر(?) مشخص میکنیم که ایا در زمان اجرا این مقدار null خواهد بود یا نه . با این کد بیشتر متوجه خواهید شد:

// This won't compile. Artist can't be null  
var notNullArtist: Artist = null  
// Artist can be null  
var artist: Artist? = null  
// Won't compile, artist could be null and we need to deal with that  
artist.print()  
  
// Will print only if artist != null  
artist?.print()  
// Smart cast. We don't need to use safe call operator if we previously  
// checked nullity  
if (artist != null) {  
 artist.print()  
}  
// Only use it when we are sure it's not null. Will throw an exception otherwise.  
artist!!.print()  
// Use Elvis operator to give an alternative in case the object is null.  
val name = artist?.name ?: "empty"

**توابع الحاقی**

ما با استفاده از این ویژگی میتونیم توابع جدید رو به هر کلاسی اضافه کنیم. خیلی پیش میومد که دلتون میخواست توابع جدیدی رو به کلاس هایی اضافه کنید که بهشون دسترسی نداشتید. به عنوان مثال خیلی خوبه که بتونین یک toast رو به یک fragment اضافه کنید:

fun Fragment.toast(message: CharSequence, duration: Int = *Toast*.*LENGTH\_SHORT*) {  
 *Toast*.makeText(getActivity(), message, duration).show()  
}

و حالا کافیه این کار رو انجام بدین:

fragment.toast("Hello world!")

**پشتیبانی فانکشنال ( لاندا ها)**

چی می شد به جای استفاده از یک کلاس بی نام، اونم هر دفعه برای یک click listener تنها میگفتیم چه کاری میخوایم انجام بدیم؟ خوشبختانه به لطف لاندا ما حالا میتونیم از خیلی از ویژگی های خسته کننده جاوا راحت بشیم:

view.setOnClickListener **{** toast("Hello world!") **}**

این ها تنها گزینه های کوچیکی بودن از چیز هایی که کاتلین میتونه انجام بده. در قسمت بعدی میخوایم عملی وارد استفاده از کاتلین بشیم.