مطمئنم تا الان اسم خطای میلیارد دلاری رو شنیدین که مبحث Null هارو پیش میاره. مقالات خیلی زیادی در این مورد ارائه شده که به زبان هایی اشاره دارن که از Null ها ساپورت میکنن و در زمان اجرا به NullPointerException میخورن.

به صورت پیشفرض کاتلین Null Safe است. این به چه معنیه؟ بیاین یکم کد بزنیم تا متوجه بشیم

fun main**(**args: Array<String>**)** {  
 val message = "A String"  
  
 println**(**message.length**)**}

یک رشته ساختیم به نام message و داخلش یک مقدار ریختیم. حالا طول message رو پرینت میکنیم. مطمئنم هیچ وقت به مشکل برنمیخوریم! چرا که هیچ وقت message تهی یا Null نخواهد بود.چرا ؟ چون خب مقدار دهی کردیمش. بیاین یکم واضح درمورد message صحبت کنیم و دقیق تایپش رو بنویسیم. بعدش مقدار جلوش رو پاک کنیم و مقدار null رو جایگزینش کنیم.

fun main**(**args: Array<String>**)** {  
 val message :String = null

println**(**message.length**)**}

متوجه میشیم که کامپایلر بهمون ارور میده! چراکه در کاتلین به صورت پیشفرض تایپ ها امکان null بودن نخواهند داشت برای اینه که میگیم کاتلین یک زبون Null Safe هست! پس زمانی که یک متغیر رو معرفی میکنی Null نخواهد بود و این اطمینان رو بهمون میده که وقتی از message.length استفاده میکنیم هیچ وقت به NullRefrence نخواهیم خورد.

ولی زمانی که ما خودمون بخوایم از Null استفاده کنیم چی؟ به هرحال دلیلی ممکنه پیش بیاد که ما بخوایم با null ها کار کنیم مثلا یک چیزی که ما خیلی ذکر کردیم اینه که کاتلین میتونه به راحتی با جاوا ارتباط داشته باشه و همکاری کنه! خب توی جاوا هم null ها هستن، اگر کلاس جاوامون به کلاس کاتلینمون null داد چی؟

در واقع ما میتونیم متغیر هایی تعریف کنیم که مقدار null رو بپذیرن و تعریفشون هم با استفاده از "?" انجام میگیره

fun main**(**args: Array<String>**)** {  
// val message :String = null

val nullMessage :String? = null

// println**(**message.length**)**}

همچنین میتونیم اصلا تایپش رو ذکر نکنیم و کامپایلر خودش فرض کنه این متغیر میتونه null باشه

fun main**(**args: Array<String>**)** {  
// val message :String = null

// val nullMessage :String? = null

val inferredNullMessage = null

// println**(**message.length**)**}

خب حالا ایا ما خاصیت NullSafety رو از کاتلین از بین بردیم؟ بیاین این کد رو اجرا کنین:

fun main**(**args: Array<String>**)** {  
// val message :String = null

val nullMessage :String? = null

// val inferredNullMessage = null

// println**(**message.length**)**

println**(**nullMessage.length**)**

}

متوجه میشیم که کامپایلر بازم ارور میده، کامپایلر میگه ممکنه این متغییر به Null بخوره، پس من اجازه نمیدم اجراشه!

خب درواقع کاری که قبلا میکردیم این بود که یک شرط قبل اجرا میذاشتیم که اگه اون مقدار null بود فلان کار رو انجام نده.الانم همین کارو میتونیم انجام بدیم

fun main**(**args: Array<String>**)** {  
// val message :String = null

val nullMessage :String? = null

// val inferredNullMessage = null

// println**(**message.length**)**

if**(**nullMessage != null**)** println**(**nullMessage.length**)**}

و کامپایلر متوجه میشه که شما قبلش دارین چک میکنین که null هست یا نه و اجازه میده که این برنامه اجرا شه. ولی خب برگشتیم سر خونه اول! الان یک متغیر داریم که میتونه null باشه و هرجا میخوایم ازش استفاده کنیم چک میکنم که ایا null هست یا نه و حالا بعضی مواقع هم پیش میاد که نیاز باشه قبل از این که یک متغیر رو چک کنیم که null هست یا، لازم باشه متغیر های قبل از اونو هم چک کنیم و اون موقع است که کلی if تو در تو داریم.

کاتلین میاد یک اپراتوری معرفی میکنه به شکل "؟" که خودش میاد میگه "اگر" مقدار این متغیر برابر null نبود عمل رو انجام بده و اگر برابر null بود از روش عبور کن. مثلا عبارت بالامون رو تنها کافیه اینجوری بنویسیم:

fun main**(**args: Array<String>**)** {  
// val message :String = null

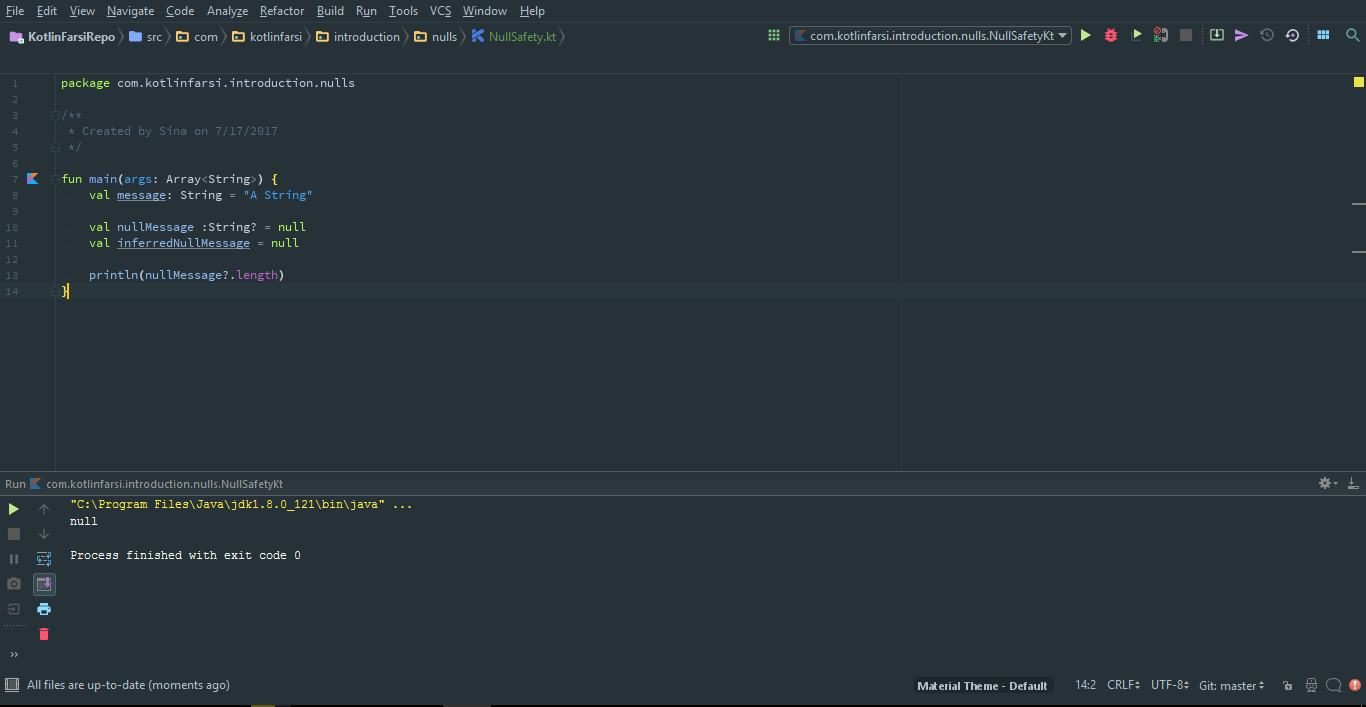
val nullMessage :String? = null

// val inferredNullMessage = null

// println**(**message.length**)** println**(**nullMessage?.length**)**}

و ارورمون هم اینجا از بین میره حتی!

البته اگه کد بالا رو اجرا کنیم جواب زیر رو خواهیم داشت



چیزی که اینجا میبینین اینه که null چاپ شد، خب پس چرا null چاپ شد؟

در واقع println یک null رو میگیره و تبدیلش میکنه به یک رشته با محتوای null و این خوبه، چراکه حداقل null رو چاپ میکنه!

البته یک روش خطرناک دیگه هم هست برای اینکه بدون "؟" بخوایم یک متغییر null رو چاپ کنیم یا به مانند این با یک متغیر null عملی رو انجام بدیم. واونم اینه که از عملگر "!!" استفاده کنیم، یعنی

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

val nullMessage = null

println**(**nullMessage!!.length**)**}

ولی وقتی این رو اجرا کنیم متوجه میشیم که برنامه به ارور میخوره !! نه ارور قبل کامپایل نه! بلکه ارور در زمان اجرا! همون ارور ها که باعث ضرر ملیون دلاری میشه!

خب چرا ؟ مگه ما نگفتیم کاتلینن NullSafe هستش ؟ جواب اینه که اینجا ایرادی به کاتلین و کامپایلرش نیست، بلکه ایراد به ما و توسعه دهنده هاست، درواقع ما با اضافه کردن اوپراتور "!!" به کامپایلر میگیم " من میدونم دارم چیکار میکنم، بذار کار خودم رو انجام بدم" و کامپایلر هم میذاره روند کامپایل و اجرای برنامه پیش بره ولی اگر اینجا به مشکل خورد مشکل خودتونه. البته سوالی هم هست که چرا ما بخوایم همچین کاری کنیم؟ درواقع زمان هایی هست مخصوصا زمانی که بحث ارتباط با جاوا پیش بیاد و این منطقی میشه که صریحا به کامپایلر بگیم نیازی به نگران شدنت نداریم، من میدونم دارم چیکار میکنم.

خلاصه بخش 7:

1. کاتلین به صورت پیشفرض یک زبان Null Safe هستش
2. شما میتونین متغیر هایی داشته باشین که صریحا بیان کنن که میتونن null باشن، ما این کار رو با ? انجام میدیم
3. عملگر ؟ به ما این قدرت رو میده که بتونیم به متغیر ها زمانی که null نیستند دسترسی داشته باشیم
4. عملگر !! این اجازه رو میده که بدون این که چک کنیم این متغیر null هست یا نه به اون متغیر دسترسی داشته باشیم