Kotlin'in null-safety özelliği, diğer birçok programlama dilinden farklıdır çünkü dilin tasarımında null referansların neden olduğu yaygın hataları önlemek ve daha güvenli bir kodlama deneyimi sağlamak hedeflenmiştir. Bu özellik, Kotlin'i daha güvenli ve hataya dayanıklı hale getirirken, aynı zamanda geliştiricilerin daha az hata yapmalarına ve daha az hata ayıklamalarına olanak tanır. İşte Kotlin'in null-safety özelliğini diğer dillerden farklı kılan bazı ana nedenler:

1. Nullable ve Non-nullable tipler: Kotlin, tüm değişkenleri varsayılan olarak nullable olmayan (**non-nullable**) olarak tanımlar. Bu, bir değişkenin null değer almasını önlemeye yardımcı olur ve NullPointerException hatalarının önüne geçer. Eğer bir değişkene null değer atanmasına izin vermek isteniyorsa, o değişkenin tipini nullable hale getirebilirsiniz.
2. Null kontrol operatörleri: Kotlin, **?.** ve **!!.** gibi null kontrol operatörlerini sunar. **?.** operatörü, null olmayan bir nesne üzerinde bir işlem yaparken null referans hatalarını önlemek için kullanılır. **!!.** operatörü ise geliştiricinin bilerek bir null referansı kabul ettiği durumlarda kullanılır. Bu operatör, null referanslarından dolayı potansiyel hata olasılığını belirtir.
3. Safe Cast: Kotlin, güvenli bir dönüşüm yapmak için **as?** operatörünü sağlar. Bu operatör, bir nesnenin belirli bir tipe dönüştürülmesini dener ve başarılı olursa sonucu döndürür; başarısız olursa null döner. Bu sayede, hatalı dönüşümler nedeniyle ortaya çıkabilecek hataları önlemek mümkün olur.
4. Platform tipi: Kotlin, Java ile uyumlu olmak için bazı durumlarda null olmayan bir değeri bile platform türü olarak kabul eder. Bu, Java kütüphaneleriyle uyum sağlamak için kullanılırken, bu türlerin null kontrolü konusunda dikkatli olunmasını gerektirir.
5. Neredeyse tüm Kotlin standart kütüphanesi, nullable değerlerle etkili bir şekilde çalışmayı destekleyecek şekilde tasarlanmıştır. Bu, geliştiricilere nullable değerleri ele almak için daha rahat bir ortam sağlar.

Bu özellikler, Kotlin'i diğer dillere kıyasla daha güvenli ve daha az hata eğilimli bir dil yapar. Geliştiricilere, null referanslarından kaynaklanan hataları önlemek ve daha güvenli kodlar yazmak için daha iyi araçlar ve yöntemler sunar.

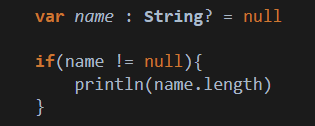
Kotlin'de bir değişken tanımladığımızda bu değişkene bir değer vermek yani onu **initialize**etmek zorundayız, eğer bir değer vermek istemiyorsak;

Bu resim için metin sağlanmadı

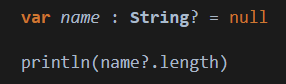
Şeklinde bir tanımlama yaparak, **?** sayesinde değişkenimizin başlangıçtaki değerini null verebilir ve null değer alabilirsin demiş oluyoruz. Aslında biz bunu yaparak Kotlin'e değeri olmayan değişkene erişimi engelle demiş oluyoruz. Bu işlemin sonucunda da hata almaktan kurtulmuş ve eğer değişkenimizin bir değeri yok ise bize null değerini ver demiş oluyoruz.

**Safe call operatörü (?.) :**

Bu operatörün yaptığı işlemi if kontrolleri ile de yapabiliriz.



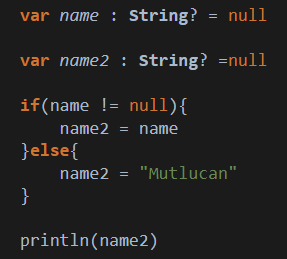
Tahmin edeceğiniz üzer kodu çalıştırdığımızda herhangi bir çıktı almayacağız fakat hata da almayacağız. Evet bu yöntem kullanılabilir fakat bunu tek satırda yapmak varken neden birden fazla satırda yapalım. Tek satırda yapmak için ise safe call operatörü olan **?.** kullanıyoruz



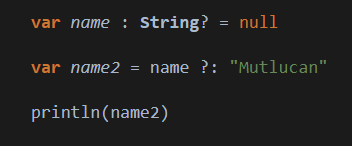
Bu işlemin sonucunda konsolda null değerini göreceğiz yani burada **?.** operatörü ile değişkenimizin değeri null değilse name değişkeninin karakter uzunluğunu ver eğer null ise null değerini dönder demiş oluyoruz

**Elvis operatörü (?:) :**

Elvis operatörü ile atanmak istenen değer null ise default olarak başka bir değer atanmasını sağlayabiliyoruz. Bir üst konuda safe call operetörünün yaptığı işlemi if kontrolleri ile gerçekleştirmiştik tahmin edeceğiniz üzere if kontrolleri ile elvis operatörünün yaptığı işlemi de gerçekleştirebiliriz fakat gene aynı yere geliyoruz bunu bir satırda yapmak varken neden birden fazla satırda yapalım



if kontrolleri



Elvis operatörü

Görmüş olduğunuz gibi iki kod parçası da aynı işlemi yapıyor fakat if kontrolleri ile altı satırda yapılacak olan işlemi elvis operatörü ile tek satırda indirgemiş oluyoruz bu da bizim hızlanmamızı ve kod okunurluğunu arttırmamızı sağlıyor.