Window Insets, OnApplyWindowInsetsListener, WindowInsetsCompat, TypeMasks

Window Insets Nedir

Android window insets, bir ekranın kenarları ile içeriği arasındaki boşluğu belirtmek için kullanılan bir kavramdır. Bu, özellikle ekran düzenini ve bileşenlerin konumunu hassas bir şekilde kontrol etmek istediğinizde önemlidir. Uygulamanızın farklı ekran boyutlarına ve cihazlara uyumlu olmasını sağlamak için window insets'i dikkate alarak kullanıcı arayüzünüzü tasarlamanız önemlidir.

"setOnApplyWindowInsetsListener" yöntemi, Android uygulama geliştirme sürecinde "edge-to-edge" tasarımın yaygınlaşmasıyla birlikte daha sık kullanılmaya başlamıştır. Bu yöntem, uygulama içeriğinin ekranın kenarlarına yayılmasını ve sistem çubuklarının arkasında görünmesini sağlar.

Modern Android cihazlarının çoğu çerçevesiz veya neredeyse çerçevesiz tasarıma sahip olduğu için, içeriğin ekranın kenarlarına kadar genişlemesi önemlidir. "Edge-to-edge" tasarım, kullanıcı deneyimini geliştirir ve kullanıcıların daha etkileyici bir deneyim yaşamasını sağlar.

Geliştiriciler, "setOnApplyWindowInsetsListener" gibi yöntemleri kullanarak uygulama penceresine gönderilen pencere içerik bölgelerinin boyutunu ve konumunu belirleyebilirler.

!! İlk kod örneği deprecated fonksiyonlar içermektedir. Farklı kullanımları göstermek amacıyla kod örneği bulunmaktadır. İlk kod örneğinden sonra yeni kullanımlar gösterilmektedir.

ViewCompat.setOnApplyWindowInsetsListener

→ **`ViewCompat.setOnApplyWindowInsetsListener`** fonksiyonunu kullanarak, sistem UI öğeleriyle etkileşime geçebilir ve içeriğinizi bu öğelere uygun şekilde ayarlayabilirsiniz. Bu fonksiyon arka planda istenilen View veya ViewGroup'a **`OnApplyWindowInsetsListener`** eklemektedir. Bu dinleyici, pencere içine yerleştirilen öğelerin inset (kenar boşlukları) bilgilerini almak ve bunları görünüm üzerinde işlemek için kullanılır.

Bu yöntem, özellikle Android'in daha yeni sürümlerinde (Android 5.0 ve sonrası) sistem UI öğelerinin değişkenliğinin olduğu durumlarda kullanışlıdır.

Öncelikle fonksiyonun kullanımı için parametre ve return bilgilerini tablodan inceleyelim:

Parameters	
`v`	`View` : Insetleri uygulayan view, bu değer `null` olamaz.
`insets`	`WindowInsetsCompat` : Uygulanacak insetler, bu değer `null` olamaz.
Returns	
`WindowInsetsCompat`	Tüketilen insetlerin çıkarılmış hali döndürülür, bu değer `null` olamaz.

Aşağıdaki örnekte bu fonksiyon kullanılarak statusBar ve NavigationBar uzunlukları bulunmuş ve kullanılacak textView'e bu uzunluklar baz alınarak padding değerleri verilmiştir. Son olarak kullanılan insetler tüketime tabi tutulmuştur.

```
ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
    val statusBarHeight = insets.systemWindowInsetTop
    val navigationBarHeight = insets.systemWindowInsetBottom

    // Örneğin, bir TextView'in kenar boşluklarını ayarla
    textView.setPadding(
        textView.paddingLeft,
        textView.paddingTop + statusBarHeight,
        textView.paddingRight,
        textView.paddingBottom + navigationBarHeight
    )

    // insets tüketilir
    insets.consumeSystemWindowInsets()
}
```

Fonksiyon Adımları:

1. **Listener Ekleme**: **`setOnApplyWindowInsetsListener`**, belirtilen **`View`**'e bir listener ekler. Bu dinleyici, **`View`**'e yerleştirilen içeriğin pencere kenarlarına olan etkilerini kontrol etmek için kullanılır.
2. **Inset Bilgilerini Alma**: Dinleyici, **`onApplyWindowInsets`** yöntemi ile sistem tarafından sağlanan **`WindowInsets`** nesnesini alır. Bu nesne, sistem pencere kenar boşlukları (inset) bilgilerini içerir. Örneğin, **`insets.systemWindowInsetTop`**, **`insets.systemWindowInsetBottom`** gibi değerlerle kenar boşluklarını alabilirsiniz.
3. **Görünümü Ayarlama**: Dinleyici, bu inset bilgilerini kullanarak **`View`**'in görünümünü ayarlar. Örneğin, bir **`TextView`**'in kenar boşluklarını alarak metni bu boşluklara göre konumlandırabilirsiniz.
4. **İnsetlerin Tüketilmesi**: İşlemler tamamlandıktan sonra, **`insets.consumeSystemWindowInsets()`** yöntemi ile bu inset bilgilerini tüketmek önemlidir. Bu, sistem UI'nın daha sonra bu insetleri kullanmasını engeller ve çakışmaları önler.

Ayrıca bu fonskiyonun view'a **`OnApplyWindowInsetsListener`** eklediğinden bahsetmiştik. Arka planda çalışan bu listener olduğundan dolayı Fragment veya Activity'e bu listeneri implement ederekte bir kullanım sağlayabiliriz. İmplementasyon işleminden sonra fonksiyon override edilebilir hale gelecektir. Yukarıda ki kullanımdan herhangi bir farkı yoktur, listener kısmı yazılmak yerine **`**this**`** keywordu verilerek class içerisindeki fonksiyona yönlendirilir. Aşağıda kullanımı gösterilmiştir.

```
class MainActivity : AppCompatActivity(), OnApplyWindowInsetsListener {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
this)

    }

    override fun onApplyWindowInsets(v: View, insets: WindowInsetsCompat):
WindowInsetsCompat {

        val systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars())
        v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
```

```
        return  WindowInsetsCompat.CONSUMED

    }
}
```

Insets

Yukarıda gördüğünüz iki örnekte insetlere farklı şekilde erişim sağlanmıştır. İlk örnekte görülen **`insets.systemWindowInsetTop`** kullanımı artık deprecate olmuştur ve kullanımı önerilmemektedir. Fakat yazımızda böyle bir kullanım da olduğunu gösterebilmek için örnek içerisinde kullandık. Güncel kullanım ikinci örnekte olduğu gibi **`insets.getInsets(WindowInsetsCompat.Type.{type})`** şeklinde olmalıdır. Ayrıca bu iki örneğin consume şekilleri de birbirinden farklıdır, **`insets.consumeSystemWindowInsets()`** kullanımı deprecate olmuştur. Yeni kullanım **`WindowInsetsCompat.CONSUMED`** şeklinde olmalıdır. Bir WindowInsets örneğinin farklı türlerini ayrı ayrı tüketme işlemi pratik değildir ikinci kullanımla birlikte tüm türler tüketime tabi tutulabilir.

`WindowInsetsCompat.Type`, Android'in **`WindowInsets`** sınıfındaki farklı inset türlerini temsil eden bir enum'dur. Bu enum, cihazın ekranındaki sistem UI öğelerinden (status bar, navigation bar, system gestures) kaynaklanan boşlukları tanımlamak için kullanılır. **`WindowInsetsCompat`** kütüphanesinde kullanılan bu tipler, Android'in farklı sürümlerindeki sistem UI öğeleri için tutarlı bir şekilde çalışmayı sağlar.

`WindowInsetsCompat.Type` Türleri:

- **`systemBars()`**
    - Bu, sistem çubuklarının (status bar, caption bar ve navigation bar) insetlerini temsil eder.
- **`captionBar()`**
    - Metin açıklamaları veya bildirimler gibi üstte görüntülenen metin bilgileri için sistem UI boşluğunu temsil eder.
- **`ime()`**
    - Klavyenin açıldığı durumda, klavyenin kapladığı alanı temsil eder.

- **`mandatorySystemGestures()`**
    - Bu inset türü, zorunlu (mandatory) sistem hareketlerinden kaynaklanan insetleri temsil eder.
    - Zorunlu sistem hareketleri, Android'in belirli sürümlerinden itibaren cihazlarda kullanılabilir olmuş olan ve cihazın ekran alanını genişletmek veya uygulamalar arası geçişleri kolaylaştırmak için kullanılan sistem hareketleridir.
- **`systemGestures()`**
    - Bu inset türü ise, sistem hareketlerinden zorunlu olmayan (örneğin, sistem navigasyon çubuğu için swipe alanı) kaynaklanan insetleri temsil eder.
- **`tappableElement()`**
    - Tıklanabilir öğelerin (örneğin, dokunmatik öğeler) sistem UI öğeleriyle olan etkileşimine izin veren insetleri temsil eder.
    - Örneğin, cihazın altındaki sistem navigasyon çubuğu alanı ile ekranın altındaki tıklanabilir bir düğme arasında bir boşluk varsa, bu boşluğu belirtmek için kullanılabilir.
- **`displayCutout()`**
    - Ekranın kenarındaki çentiği (notch) temsil eden insetleri belirtir, özellikle notch olan cihazlarda, bu insetlerle çentiğin ekran içeriğine olan etkisi belirtilebilir.
- **`statusBars()`**
    - Ekranın üst kenarındaki status bar (durum çubuğu) insetlerini temsil eder.
- **`navigationBars()`**
    - Ekranın alt kısmındaki sistem navigasyon çubuklarının (navigation bar) insetlerini temsil eder.

Insetler için **`getInsets`** ’den farklı olarak kullanılabilecek **`isVisible`** fonksiyonu da vardır. Bu fonksiyon inset’in visibility özelliğini döndürmektedir. Aşağıdaki örnek kullanımda klavyenin görünür olup olmadığı kontrol edilmiştir.

```
ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
  val imeVisible = insets.isVisible(WindowInsetsCompat.Type.ime())
  WindowInsetsCompat.CONSUMED
}
```

- **Selin Ayten Cengiz**