



description: "Android üzerinde farklı thread üzerinde çalışma (\U0001F6A7 yapım aşamasında)"

Asenkron İşlemler

Asenkron İşlemleri Tanıyalım

-  Aynı bir Thread üzerinden gerçekleşen bu işlemleri sistemin ilerlemesi engellemez
-  İşleri tamamlandığı zaman UI Thread'e dahil olurlar
- ☆ **AsyncTask** veya **AsyncTaskLoader** yapıları kullanılır

İkisi Arasındaki Temel Farklar

Her ikisi de sistemi bloklamadan çalışan bir yapıya sahiptir

AsyncTask	AsyncTaskLoader
Direkt olan çalışır	Dolaylı olarak çalışır
Yapılandırma ayarları değiştiğinde iptal olur ve yeniden başlatılır	Yapılandırma ayarlarından etkilenmez
Geri dönüş vermeyecek işlemlerde kullanılır	Geri dönüşümlü işlemlerde kullanılır
Kısa ve iptal edilebilir işlemlerde tercih edilir	Uzun ve iptal edilemeyecek işlemlerde tercih edilir

Telefonu döndürme gibi işlemler yapılandırma ayarlarını değiştirir.

{% hint style="success" %} Genel olarak **AsyncTaskLoader** en sık kullanılan yapıdır. {% endhint %}

UI Thread

Android'teki tüm görüntü işlemlerinin yapıldı alandır.

- UI Thread engellenmemeli
- UI Thread sadece görsel işlemler için kullanılmalıdır
- Tüm işlemler 16ms'den kısa bir sürede tamamlanmalıdır



{% hint style="danger" %} Yaklaşık olarak 5s'den uzun süren işlemler "application not responding" (ANR) diyalogunu oluşturur ve kullanıcı bunu görmesi durumunda uygulamayı kapatıp, siler 😞 {% endhint %}

AsyncTask

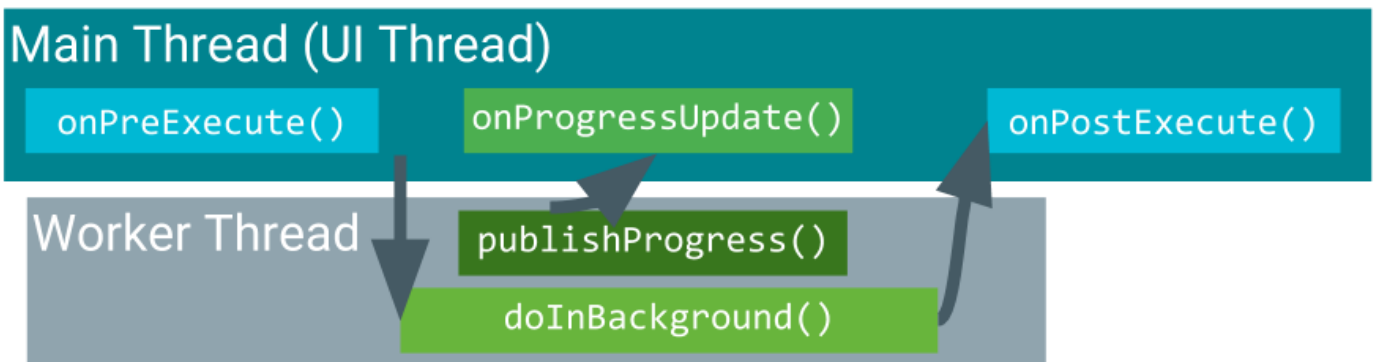
Verilen işlemi arkaplanda, sistemi bloklamadan tamamlar.

- Yapılandırma ayarlarından etkilenir, işlem yok edilip yeniden başlatılır
 - Telefonu döndürme vs gibi işlemler yapılandırma ayarlarını değiştirir
 - Aynı işlemin çokça yapılması RAM tüketimini artırır
- Uygulama kapatıldığında cancel() metodu çalıştırılmadığı sürece çalışmaya devam eder

{% hint style="warning" %} Önemli ve kritik işlemler için **AsyncTaskLoader** tercih edilir {% endhint %}



{% tabs %} {% tab title="🔗 Kullanım" %}



Metot

Açıklama

`onPreExecute()`

İşlem tamamlanmadan önce ara ara çağrılan metottur, genellikle % dolum bilgisi vermek için kullanılır

`doInBackground(Params...)`

`onPreExecute()` metodu bittiği an çalışır, arkaplan işlemlerini yapan kısımdır. `publishProgress()` metodu ile değişiklikleri UI Thread'e aktarır. Bittiğinde `onPostExecute()` metoduna sonucu aktarır.

`onProgressUpdate(Progress...)`

`publishProgress()` metodundan sonra çalışır, genellikle raporlama veya ilerleme adımlarını kullanıcıya göstermek için kullanılır

`onPostExecute(Result)`

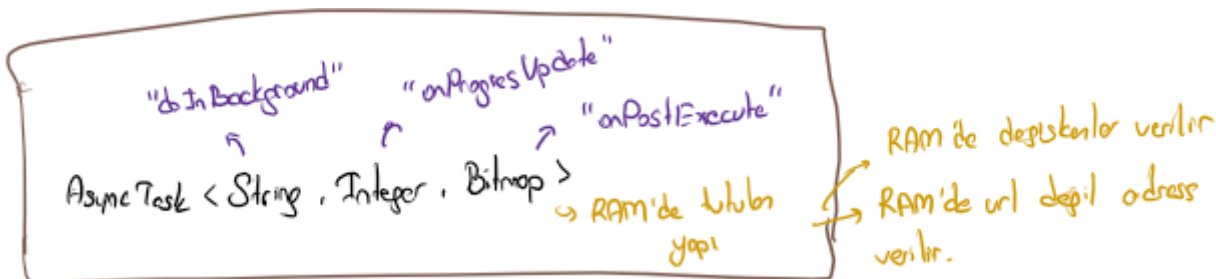
Arkaplan işlemi tamamlandığında sonuç buraya aktarılır, UI Thread bu metot üzerinden sonucu kullanır.

{% hint style="warning" %} `onProgressUpdate` metodunda tüm adımları ele alırsanız, asenkron çalışma yapısı bozulur ve senkronize olarak çalışır {% endhint %} {% endtab %}

{% tab title="📁 Prototip" %}

```
public class MyAsyncTask extends AsyncTask <String, Void, Bitmap>{}
```

- `String` değişkeni, `doInBackground` metoduna aktarılacak verilerdir
- `Void` yapısı, `publishProgress` ve `onProgressUpdate` metotlarının kullanılmayacağını belirtir
- `Bitmap` tipi de, `onPostExecute` ile aktarılan işlem sonucunun tipini belirtir



{% hint style="warning" %} Son iki parametre (`Void` ve `Bitmap`) dışarıdan verilmez, sınıf içi parametrelerdir {% endhint %} {% endtab %}

{% tab title="✖ İşlemi İptal Etme" %} İşlemi istediğin zaman `cancel()` metodu ile iptal edebilirsin

- `cancel()` metodu işlem tamamlanmışsa `False` döndürür
- Biten işlemi iptal edemezsin 😞
- İşlemin iptal edilme durumunu `doInBackground` metodunda `isCancelled()` metodu kontrol etmemiz gerekmektedir
- İşlem iptal edildiğin `doInBackground` metodundan sonra `onPostExecute` yerine `onCancelled(Object)` metodu döndürülür

{% hint style="info" %} Varsayılan olarak `onCancelled(Object)` metodu `onCancelled()` metodunu çağırır, sonuç görmezden gelinir. {% endhint %} {% endtab %}

{% tab title="📁 Kod" %}

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        // 0. eleman en son adımı belirtir. (FIFO)
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}

// UI Thread'te kullanımı
new DownloadFilesTask().execute(url1, url2, url3);
```

{% endtab %} {% endtabs %}

Harici Bağlantılar

{% embed url="https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-3-working-in-the-background/lesson-7-background-tasks/7-1-c-async-task-and-async-task-loader/7-1-c-async-task-and-async-task-loader.html" %}