RcycleView

Bağımlılıkları Dahil Etme

```
dependencies {
    implementation 'com.android.support:recyclerview-v7:28.0.0'
}
```

XML Dosyasında Tanımlama

```
<?xml version="1.0" encoding="utf-8"?>
<!-- A RecyclerView with some commonly used attributes -->
<android.support.v7.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Activity Üzerinde Tanımlama

```
public class MyActivity extends Activity {
    private RecyclerView recyclerView;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager layoutManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_activity);
        recyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);

        // use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        recyclerView.setHasFixedSize(true);

        // use a linear layout manager
        layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);

        // specify an adapter (see also next example)
        mAdapter = new MyAdapter(myDataset);
        recyclerView.setAdapter(mAdapter);
    }
```

```
    // ...
}
```

🗁 Adapter Sınıfı

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.MyViewHolder> {
    private String[] mDataset;

    // Provide a reference to the views for each data item
    // Complex data items may need more than one view per item, and
    // you provide access to all the views for a data item in a view holder
    public static class MyViewHolder extends RecyclerView.ViewHolder {
        // each data item is just a string in this case
        public TextView textView;
        public MyViewHolder(TextView v) {
            super(v);
            textView = v;
        }
    }

    // Provide a suitable constructor (depends on the kind of dataset)
    public MyAdapter(String[] myDataset) {
        mDataset = myDataset;
    }

    // Create new views (invoked by the layout manager)
    @Override
    public MyAdapter.MyViewHolder onCreateViewHolder(ViewGroup parent,
                                                     int viewType) {
        // create a new view
        TextView v = (TextView) LayoutInflater.from(parent.getContext())
                .inflate(R.layout.my_text_view, parent, false);
        ...
        MyViewHolder vh = new MyViewHolder(v);
        return vh;
    }

    // Replace the contents of a view (invoked by the layout manager)
    @Override
    public void onBindViewHolder(MyViewHolder holder, int position) {
        // - get element from your dataset at this position
        // - replace the contents of the view with that element
        holder.textView.setText(mDataset[position]);

    }

    // Return the size of your dataset (invoked by the layout manager)
    @Override
    public int getItemCount() {
        return mDataset.length;
```

```
    }
}
```

Faydalı Bağlantılar

{% embed url="https://developer.android.com/guide/topics/ui/layout/recyclerview" %}

AlertDialog

Zaman Seçme

```
public static class TimePickerFragment extends DialogFragment
implements TimePickerDialog.OnTimeSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);
        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
            DateFormat.is24HourFormat(getActivity()));
    }
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />
```

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

Tarih Seçme

```
public static class DatePickerFragment extends DialogFragment
implements DatePickerDialog.OnDateSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);
        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }
    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user
    }
}
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_date"
    android:onClick="showDatePickerDialog" />
```

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

---

description: Android üzerinde verileri kaydetme

Veri Saklama Yöntemleri

Shared Preference vs Saved Instance State

Shared Preference	Saved Instance State
Uygulama hatta cihaz kapatılsa dahi veriler korunur	Sadece Activity üzerinde verileri saklar
Temel veriler için kullanılır	Anlık veriler için tercih edilir
Az sayıda anahtar / değer çifti ile saklanır	Az sayıda anahtar / değer çifti ile saklanır
Veriler uygulamaya özgüdür (gizli)	Veriler uygulamaya özgüdür (gizli)

Shared Preference	Saved Instance State
Genellikle kullanıcı tercihleri için kullanılır	Cihazın yapılandırma ayarlarında veri kaybını engellemek için kullanılır

{% hint style="info" %} Detaylar için Shared preferences vs. saved instance state alanına bakabilirsin. {% endhint %}

Database vs Shared Preference

Database	Shared Preference
Yüksek miktarda veriler	Kullanıcı yapılandırmaları
Veriler içerisinde kompleks aramalar (SQL)	Anahtar / Değer çiftleri
Daha yavaş okuma hızı	Daha hızlı okuma hızı

{% hint style="info" %} Detaylı bilgi için Pros and Cons of SQL and Shared Preferences alanına bakabilirsin. {% endhint %}

Faydalı Bağlantılar

{% embed url="https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-4-saving-user-data/lesson-9-preferences-and-settings/9-0-c-data-storage/9-0-c-data-storage.html" %}

---

description: JSON verilerini ayrıştırma ve parsing işlemi

JSON Yönetimi

Nedir?

- Web sayfalarında saklanan veri formatıdır
- JavaScript Object Notation

Örnek JSON Verisi

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}
```

Ayrıştırma İşlemi

```
JSONObject data = new JSONObject(responseString);
JSONArray menuItemArray = data.getJSONArray("menuitem");
JSONObject thirdItem = menuItemArray.getJSONObject(2);
String onClick = thirdItem.getString("onclick");
```

Faydalı Bağlantılar

{% embed url="https://stackoverflow.com/a/9606629/9770490" %}

---

description: Android üzerinde dahili veya harici dosya sistemine erişme ve yönetme

Dosya İşlemleri

Internal Storage (Dahili)

- Kalıcı depolama dizini `getFilesDir()`
- Geçici depolama dizini `getCacheDir()`
  - 1 MB ve daha hafif dosyalar için önerilir
  - Hafıza yetmemeye başladığında burası silinebilir

```
File file = new File(context.getFilesDir(), filename);
```

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

```
public File getTempFile(Context context, String url) {
    File file;
    try {
        String fileName = Uri.parse(url).getLastPathSegment();
        file = File.createTempFile(fileName, null, context.getCacheDir());
    } catch (IOException e) {
```

```
            // Error while creating file
        }
        return file;
    }
```

External Storage (Harici)

- Öncelikle okuma izinleri alınır
- Harici depolama sisteme bağlı mı kontrol edilir
- Erişim hakkımızın olup olmadığı kontrol edilir
- Ardından dosyalara erişim yapılır

Okuma İzinlerini Alma

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```

Erişimi ve Bağlantıyı Kontrol Etme

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

Dosyaların Yönetimi

```
// Harici dosyalara erişebilmek için (public external storage)
File path = Environment.getExternalStoragePublicDirectory(
```

```
            Environment.DIRECTORY_PICTURES);
File file = new File(path, "DemoPicture.jpg");
```

```
// Gizli harici dosyalara erişmek için (private external storage)
File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
```

Dosyaları Temizleme

- Kullanılmayan her dosya temizlenmelidir

```
myFile.delete();
myContext.deleteFile(fileName);
```

Faydalı Kaynaklar

{% embed url="https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-4-saving-user-data/lesson-9-preferences-and-settings/9-0-c-data-storage/9-0-c-data-storage.html#files" %}

---

description: Telefonda Shared Preference yapısı ile veri saklama

Shared Preferences

Dosyaları Oluşturma

```
private String sharedPrefFile = "com.example.android.hellosharedprefs";
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

Dosyaları Kaydetme

```
@Override
protected void onPause() {
    super.onPause();
    SharedPreferences.Editor preferencesEditor = mPreferences.edit();
    preferencesEditor.putInt("count", mCount);
    preferencesEditor.putInt("color", mCurrentColor);
    preferencesEditor.apply();
}
```

Dosyaları Geri Alma

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
if (savedInstanceState != null) {
    mCount = mPreferences.getInt("count", 1);
    mShowCount.setText(String.format("%s", mCount));

    mCurrentColor = mPreferences.getInt("color", mCurrentColor);
    mShowCount.setBackgroundColor(mCurrentColor);
} else { ... }
```

🧹 Dosyaları Temizleme

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
preferencesEditor.putInt("number", 42);
preferencesEditor.clear();
preferencesEditor.apply();
```

👀 Değişikleri Takip Etme

```
public class SettingsActivity extends PreferenceActivity
                              implements OnSharedPreferenceChangeListener {
    public static final String KEY_PREF_SYNC_CONN =
       "pref_syncConnectionType";

    // ...

    public void onSharedPreferenceChanged(
                              SharedPreferences sharedPreferences,
                              String key) {
        if (key.equals(KEY_PREF_SYNC_CONN)) {
            Preference connectionPref = findPreference(key);
            // Set summary to be the user-description for
            // the selected value
            connectionPref.setSummary(
               sharedPreferences.getString(key, ""));
        }
    }
}
```

```
@Override
protected void onResume() {
    super.onResume();
    getPreferenceScreen().getSharedPreferences()
            .registerOnSharedPreferenceChangeListener(this);
}
```

```
@Override
protected void onPause() {
    super.onPause();
    getPreferenceScreen().getSharedPreferences()
            .unregisterOnSharedPreferenceChangeListener(this);
}
```

```
SharedPreferences.OnSharedPreferenceChangeListener listener =
    new SharedPreferences.OnSharedPreferenceChangeListener() {
       public void onSharedPreferenceChanged(
                           SharedPreferences prefs, String key) {
          // listener implementation
       }
};

prefs.registerOnSharedPreferenceChangeListener(listener);
```

🗑️ Eski Notlar

🏗️ Veri Oluşturma ve Alma

- `val veri= this.getSharedPreferences(this.packageName, android.content.Context.MODE_PRIVATE) // Veri kaydını değişkene atama`
  - this.packageName : paket ismi (com.... en üst satırdaki)
  - MODE_PRIVATE : sadece benim uygulamamdan erişilebilirlik
- `var age1 = 30`
- `veri.edit().putInt("userAge", age1).apply() // Veriyi kaydetme`
  - userAge : anahtar
  - age1 : değer / değişken
- `val age2= veri.getInt("userAge", 0) // Kayıtlı veriyi alma`
  - userAge : anahtar (put'takini almak için aynı olmalı)
  - 0 : Eğer anahtar yoksa, varsayılan değer ataması
- `println("stored age : $storedAge") // veriyi gösterme`

✨ Veri Güncelleme

```
age = 31
veri.edit().putInt("userAge", age).apply() // Daha önceden olan bir anahtarın üstüne kaydedilirse güncelleme olur.
```

🩹 Veri Silme

- `veri.edit().remove("userAge").apply() // Veri silindi`
  - userAge : silinecek anahtar

- `val age3 = veri.getInt("userAge", 0) // Veri olmadığı için age3 = 0 olacak.`
  - userAge : anahtar
  - 0 : varsayılan değer

Faydalı Bağlantılar

- 🕮 Concepts 9.1: Shared preferences
- Code 9.1: Shared preferences.

---

description: Android üzerinde SQLite ile veri tabanı oluşturma

SQLite

SQLite Giriş Temelleri

İlk olarak try - catch yapısı kurulur ve olası sorunda programın kapanması engellenir.

```
try {
    ...
}
catch (e : Exception){
    e.printStackTrace()
}
```

> Bütün kodları `...` olan yere yazacağız. Artık başlayabiliriz.

SQLite ile Basit DB Oluşturma

```
database = openOrCreateDatabase("Datas", Context.MODE_PRIVATE, null)
```

- "Datas" : Oluştumak istediğimiz database'in adı ("Veriler", "Hey", "hop" vb.)
  - Yazım kuralları gereği database adı büyük harfle başlamalı
- Context.MODE_PRİVATE : Database'i private (özel) sadece bizim erişebileceğimiz halde kurmak.
  - (Context.MODE yazıp ALT+ SPACE yaparsanız detaylar çıkacaktır karşınıza)
- null : CursorFactory

SQLite DB Oluşturma Kodları

```
try {
    val database = openOrCreateDatabase("Datas", Context.MODE_PRIVATE, null)

    database.execSQL("CREATE TABLE IF NOT EXIST datas (name VARCHAR, age INT(2)")
// INT(2) ile 2 rakam olacağını belli ediyoruz
} catch (e : Exception){
    e.printStackTrace()
}
```

- `CREATE TABLE IF NOT EXITS` table oluşturma
- `datas` table ismi
- `VARCHAR` char
- `INT` Int

SQLite DB İşlemleri Değiştirme

Temel yapısı `database.execSQL("...")` şeklindedir.

```
database.execSQL("INSERT INTO datas (name, age) VALUES ('Yunus' , 21)") // Veri
Ekleme
database.execSQL("INSEER INTO datas (name, age) VALUES ('Emre', 15") // Veri Ekleme
database.execSQL("UPDATE datas SET age = 21 WHERE name = 'Yunus") // Veri güncelleme
database.execSQL("DELETE FROM datas WHERE age = 15") // Veri silme
database.execSQL("SELECT FROM datas WHERE name = 'Yunus") // Yunus isimli olan
dataları alır
database.execSQL("SELECT FROM datas WHERE name LIKE '%s'") // sonunda 's' harfi
olanları alır
database.execSQL("SELECT FROM datas WHERE name LIKE 'y%") // başında 'y' harfi
olanları alır
database.execSQL("SELECT FROM datas WHERE name LIKE '%u%") // içinde 'u' harfi
olanları alır
```

- `INSERT INTO` Veri ekleme için SQL kodu
- `UPDATE` Veri güncelleme
- `SELECT` Veri seçme
- `datas` table ismi
- `name` değişken ismi
- `age` değişken ismi
- `VALUES` değerleri atamak için SQL kodu
- `'Yunus'` VARCHAR (string) tipindeki veri
- `21` INT(2) (Int) tipindeki veri

SQLite DB Okuma

```
if (database != null) {
    val cursor = database!!.rawQuery("SELECT * FROM datas", null)

    val nameIndex = cursor.getColumnIndex("name")
    val ageIndex = cursor.getColumnIndex("age")

    cursor.moveToFirst()

    while (cursor != null){
        println("İsim : ${cursor.getString(nameIndex)}" )
        println("Yaş : ${cursor.getString(ageIndex)}")
```

```
            cursor.moveToNext()
        }
    }
```

- `rawQuery(...)` SQL kodu ile veri alma
- `SELECT * FROM` Bütün verileri almak için SQL kodu
- `nameIndex` name sütünundaki verilerin indexi
- `ageIndex` age sütunundaki verilerin indexi
- `Cursor.moveToFirst()` Cursoru ilk elemana atıyoruz
- `Cursor.getString()` İstenen indexteki string olarak döndürür.
- `Cursor.moveToNext()` cursoru bir sütün aşağı indirme

---

description: Android üzerinde SQLite yerine üretilmiş yeni db formatı

Room Database

Room Database Nedir

- 🤓 SQL komutları ile uğraşmadan direkt android kodları ile çalışmamızı sağlar
- 🌟 Optimize edilmiş bir veri tabanı sunar (LiveData)

{% hint style="warning" %} 📣 Sayfanın en altındaki linklerden resmi bağlantılara erişebilirsin. {% endhint %}

Projeye Dahil Etme

```
dependencies {
  def room_version = "2.2.3"

  implementation "androidx.room:room-runtime:$room_version"
  annotationProcessor "androidx.room:room-compiler:$room_version" // For Kotlin use
kapt instead of annotationProcessor

  // optional - Kotlin Extensions and Coroutines support for Room
  implementation "androidx.room:room-ktx:$room_version"

  // optional - RxJava support for Room
  implementation "androidx.room:room-rxjava2:$room_version"

  // optional - Guava support for Room, including Optional and ListenableFuture
  implementation "androidx.room:room-guava:$room_version"

  // Test helpers
  testImplementation "androidx.room:room-testing:$room_version"
}
```

{% hint style="info" %} 🤓 Detaylar için Declaring dependencies alanına bakabilirsin. {% endhint %}

📦 Temel Yapı

☆ Entity Yapısı

- 📦 DB'ye aktarılacak sütun isimlerini temsil ederler
- 🏷️ Annotation yapısı ile özellikleri belirlenir
- 🔶 Tablodaki sütün isimleri entity üzerindeki değişkenlerle temsil edilir
- 👮 **Primary key** ve **Entity** etiketini eklemek zorunludur

```
@Entity(tableName = "word_table")
public class Word {
    @PrimaryKey (autoGenerate=true)
    private int wid;

    @ColumnInfo(name = "first_word")
    private String firstWord;

    @ColumnInfo(name = "last_word")
    private String lastWord;

    // Getters and setters are not shown for brevity,
    // but they're required for Room to work if variables are private.
}
```

{% hint style="info" %} 👀 Daha fazlası için Entity ve Defining data using Room entities dokümanlarına bakabilirsin. {% endhint %}

🏭 DAO Yapısı

- 👊 Tablolara erişmek için kullanılan yapıdır
- 📦 Abstract veya Interface olmak zorundadır
- 🏷️ SQL query metinleri metotlara Annotation yapısı ile tanımlanır
- ✨ LiveData yapısı ile güncel verileri döndürür

```
@Dao
public interface WordDao {

   // The conflict strategy defines what happens,
   // if there is an existing entry.
   // The default action is ABORT.
   @Insert(onConflict = OnConflictStrategy.REPLACE)
```

```
    void insert(Word word);

    // Update multiple entries with one call.
    @Update
    public void updateWords(Word... words);

    // Simple query that does not take parameters and returns nothing.
    @Query("DELETE FROM word_table")
    void deleteAll();

    // Simple query without parameters that returns values.
    @Query("SELECT * from word_table ORDER BY word ASC")
    List<Word> getAllWords();

    // Query with parameter that returns a specific word or words.
    @Query("SELECT * FROM word_table WHERE word LIKE :word ")
    public List<Word> findWord(String word);
}
```

{% hint style="info" %} 👀 Daha fazlası için The DAO (data access object) dokümanına bakabilirsin. {% endhint %}

🗃 Room Database

- 📦 Abstract olmak zorundadır
- 🏗 `Room.databaseBuilder(...)` yapısı ile db tanımlanır
- 🏷 Database etiketi içerisinde
  - `entities`alanında tablo verilerini temsil eden Entity Class'ınızın objesi verilir
  - `version` alanında db'nin en son sürümünü belirtin
  - 🔩 Versiyon geçişleri arasındaki sorunları engellemek için `fallbackToDestructiveMigration()` özelliği eklenir

```
@Database(entities = {Word.class}, version = 1)
public abstract class WordRoomDatabase extends RoomDatabase {

    public abstract WordDao wordDao();

    private static WordRoomDatabase INSTANCE;

    static WordRoomDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (WordRoomDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            WordRoomDatabase.class, "word_database")
                              // Wipes and rebuilds instead of migrating
                              // if no Migration object.
                            .fallbackToDestructiveMigration()
                            .build();
```

```
                }
            }
        }
        return INSTANCE;
    }
}
```

{% hint style="info" %} 👀 Daha fazlası için Room database dokümanına bakabilirsin. {% endhint %}

👮 DB'yi Koruma

- 🚫 Veri tabanına birden çok istek gelmesini engeller
- 🐞 Birden çok isteğin eş zamanlı yapılmaya çalışması **conflict** oluşturacaktır
- 💔 Conflict yapısı veri tabanındaki verilerin uyuşmazlığını belirtir
- Birden fazla Thread gelmesi durumunda engellemek için **synchronized** anahtar kelimesi kullanılır
- ✨ Gereksiz Thread engelinden sakınmak için, synchronized yapısı içerisinde tekrardan **if kontrolü** yapılmalıdır

{% hint style="info" %} 👀 Detaylar için Multi-threading alanına bakabilirsin. {% endhint %}

🏗 Repository Yapısı

- 🏢 Alt katmanda olan tüm sınıfları tek bir sınıfmış gibi gösterir
  - 😉 Bu sayede **ViewModel** üzerinden birden fazla sınıfla uğraşmak zorunda kalmayız
  - 🚧 DB üzerinde yapılacak olan tüm işlemlerinde burada metot olarak tanımlanması lazımdır
- ✨ **LiveData** yapısı sayesinde verileri otomatik günceller
  - 🦄 Verilerin aktarımı bir defaya mahsus **Constructor** üzerinde yapılır
- 🩹 Verilerin aktarılması **asenkron** olması gerektiğinden AsyncTask yapısı kullanılır

```
public class WordRepository {

    private WordDao mWordDao;
    private LiveData<List<Word>> mAllWords;

    WordRepository(Application application) {
        WordRoomDatabase db = WordRoomDatabase.getDatabase(application);
        mWordDao = db.wordDao();
        mAllWords = mWordDao.getAllWords();
    }

    LiveData<List<Word>> getAllWords() {
        return mAllWords;
    }

    public void insert (Word word) {
        new insertAsyncTask(mWordDao).execute(word);
    }
```

```
    private static class insertAsyncTask extends AsyncTask<Word, Void, Void> {

        private WordDao mAsyncTaskDao;

        insertAsyncTask(WordDao dao) {
            mAsyncTaskDao = dao;
        }

        @Override
        protected Void doInBackground(final Word... words) {
            for (Word word : words) {
                mAsyncTaskDao.insert(word);
            }

            return null;
        }
    }
}
```

ViewHolder

- 📦 Yapılandırma değişikliklerine karşı dayanıklıdır
- 🐣 Repository ile DB'ye erişir
- 🐝 Activity context objesi gönderilmez, çok maliyetlidir
- 🥚 Context verisi miras alınmalıdır
- 📝 UI ile alakalı bilgilerin kaydı ile uğraşır

```
public class WordViewModel extends AndroidViewModel {

    private WordRepository mRepository;

    private LiveData<List<Word>> mAllWords;

    public WordViewModel (Application application) {
        super(application);
        mRepository = new WordRepository(application);
        mAllWords = mRepository.getAllWords();
    }

    LiveData<List<Word>> getAllWords() { return mAllWords; }

    public void insert(Word word) { mRepository.insert(word); }
}
```

LiveData

- 🔄 Verileri güncel tutmak için kullanılır
- 📈 Performansı artırır
- 📦 Yapılandırma değişikliklerine karşı dayanıklıdır
  - 📝 Telefonu çevirme vs.
- 🖥️ Tüm katmanlardaki metotlar kapsüllenmelidir
  - Repository
  - DAO
  - ViewHolder

```
wordsViewModel.getAllNews().observe(
    this,
    words -> fillView(new ArrayList<>(news))
);

private void fillView(ArrayList<Words> words) {
    // XML layoutu üzerinden tanımlanması lazımdır
    RecyclerView recyclerView = findViewById(R.id.rv_words);

    // Class olarak tanımlanması lazımdır
    WordsAdapter wordsAdapter = new WordsAdapter(this, words);

    recyclerView.setAdapter(wordsAdapter);
    recyclerView.setLayoutManager(new LinearLayoutManager(this));
}
```

{% hint style="info" %} 🤓 Detaylar için RecycleView alanına bakabilirsiniz. {% endhint %}

🔗 Faydalı Bağlantılar

- 📃 Room, LiveData and ViewModel
- 🤖 Android Room with a View - Java
- 👁️ Android, RecycleView

---

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

🚀 Asenkron İşlemler

🔰 Asenkron İşlemleri Tanıyalım

- 🚀 Ayrı bir Thread üzerinden gerçekleşen bu işlemleri sistemin ilerlemesi engellemez
- 🤹 İşleri tamamlandığı zaman UI Thread'e dahil olurlar
- ⭐ **AsyncTask** veya **AsycnTaskLoader** yapıları kullanılır

VS 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) diyaloğunu 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 arttırır
- 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

Metot	Açıklama
`doInBackground(Params...)`	`onPreExecute()` metodu bittiği an çalışır, arkaplan işlemlerini yapan kısımdır. `publishProgress()` metodu ile değişikleri UI Thread'e aktarır. Bittiğinde `onPostExecure()` metoduna sonucu aktarır.
`onProgressUpdate(Progress...)`	`publishProgress()` metodundan sonra çalışır, genellikle raporlama veye 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 `isCancalled()` 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-asynctask-and-asynctaskloader/7-1-c-asynctask-and-asynctaskloader.html" %}

---

description: Android üzerinde arkaplanda çalışan arayüzü olmayan Activity'ler

Servisler

Yazılı Notlarım

Servislere Genel Bakış

- 🤔 Arkaplanda çalışan arayüzü olmayan Activity'ler olarak adlandırılabilir
- 🚧 Android'de arkaplanda çalışmak için Background Tasks dokümanına bakılmalıdır
- ⏳ Uzun süreli işlemler için hızı ve verimliliği artırma adına multi-threading yapısı önerilir

Android Pil Koruması Önlemleri

- 👮 Android arka plan işlemlerini bataryayı korumak adına kısıtlar.
- 🚧 Kullanıcıya arayüz (UI) sağlamayan her arkaplan işlemi (Background Service) kısıtlı sürede çalışır
- ⚙️ Kısıtlanmayı engellemek adına Foreground Service yapısı kullanılmalıdır
    - 🔔 Kullanıcıya kaldırılamayan bir bildirim gösterilir
    - 👁 Kullanıcı arkaplan işlemlerinden haberdar olur

- 🌛 Cihaz uyku moduna girdiğinde arka plan işlemleri aksamaya başlar.
  - 🧸 WakeLock özelliğinin aktif olması gerekir

{% page-ref page="foreground-service.md" %}

{% hint style="info" %} 🕵 Detaylı bilgi için Challenges in background processing alanına bakabilirsin. {% endhint %}

☑ İstek Türüne Göre Servis Seçimi

- 🏇 Servis tek bir işle baş edecek ise IntentService (eski adı JobIntentService) yapısı kullanılabilir
  - 👮 Çok fazla kısıtlamaya tabi tutulan
  - ❌ İşi bittiğinde kapanan bir sistemdir
- 👯 Servisin birden fazla istekle baş etmesi gerekirse IntentService yerine Service kullanılır

📣 Servis Tanımlama

```
<manifest ... >
  ...
  <application ... >
      <service android:name="ExampleService"
               android:exported="false" />
      ...
  </application>
</manifest>
```

⏰ Alarm

👀 Alarm Yapısına Bakış

- ⏰ Belirli sürelerde tetiklenen `Intent` işlemleridir
- 🧸 Uygulama kapalı olsa hatta telefon uykuda olsa bile çalışır
- 📈 Arka plan işlemlerinin tekrarlanma sıklığını azalttığından verimliliği artırır
- 🤵 `AlarmManager` üzerinden, `Intent-Filter` yapısı gibi yönetilir

> 🙄 Telefondaki alarmdan bahsetmiyorum.

{% hint style="info" %} 🕵 Detaylı bilgi için Introduction ~ 8.2 Alarm alanına bakabilirsin. {% endhint %}

👮 Kullanmaman Gereken Durumlar

- 🚩 Uygulaman üzerinde çalışacak olan eylemlerde kullanılmaz
- 🌟 `Handler` , `Timer` veya`Thread` yapısı tercih edilmelidir
- 🔄 Sunucu ile güncelleme işlemlerini bu yapı ile yapmayın
  - 🔥 Firebase Cloud Messaging üzerindeki `SyncAdapter` yapısı ile yapılır
- 🕛 Beklemeli işlemler için `JobScheduler` yapısını tercih edin

- 📶 Wi-Fi bağlandığında haberleri veya hava durumunu güncelleme gibi

🔷 Alarm Türleri

📦 Temel İşlemler

🏗️ Alarm Kurma

- ✨ Dakikada çok fazla tekrarlanacak işlemler için `Handler` yapısını tercih edin
- 🐣 `getSystemService(ALARM_SERVICE)` metodu ile `AlarmManager` sınıfı alınır
- ✔️ Alarm türünü belirleyin ve `set...(<tip>, <süre>, <PendingIntent>)` metodu ile tanımlayın

```
alarmMgr.set(AlarmManager.ELAPSED_REALTIME,
             SystemClock.elapsedRealtime() + 1000*300,
             alarmIntent);
```

{% hint style="info" %} 🤓 Detaylı bilgi için Scheduling an alarm alanına veya GitHub Koduna bakabilirsin. {% endhint %}

🕰️ Tekrarlı Alarmlar

```
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP,
               calendar.getTimeInMillis(),
               AlarmManager.INTERVAL_FIFTEEN_MINUTES,
               alarmIntent);
```

🤷 Olan Alarmı Kontrol Etme

```
boolean alarmExists =
 (PendingIntent.getBroadcast(this, 0,
       alarmIntent,
       PendingIntent.FLAG_NO_CREATE) != null);
```

☠️ Alarmı Öldürme

```
alarmManager.cancel(alarmIntent);
```

🛌 WakeUp (Uyandırma)

Görülebilir Alarmlar

- Kullanıcıya gösterilen alarm türleridir
- `AlarmClock` yapısı olarak ele alınır
- Genellikle uyandırma çağrıları için kullanılır

Faydalı Bağlantılar

{% embed url="https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-3-working-in-the-background/lesson-8-alarms-and-schedulers/8-2-c-alarms/8-2-c-alarms.html" %}

{% embed url="https://codelabs.developers.google.com/codelabs/android-training-alarm-manager/#0" %}

İnternete Bağlanma

El Notlarım

Gerekli İzinlerin Alınması

- `AndroidManifest.xml` dosyası üzerinden internet izni alınmalıdır
- `<uses-permission android:name="android.permission.INTERNET" />` ile internet erişimi
- `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />` ile internet bağlantısı durumu izni alınır

{% hint style="info" %} Detaylı bilgi için Including permissions in the manifest dokümanına bakabilirsin {% endhint %}

Bağlantı Durumunu Yönetme

- Tüm sistem servislerine `getSystemService` metodu ile erişilir
- `ConnectivityManager` ve `NerworkInfo` sınıflarından bağlantı bilgileri yönetilir

```
private static final String DEBUG_TAG = "NetworkStatusExample";

ConnectivityManager connMgr =
          (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();

networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();

Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

{% hint style="info" %} Detaylı bilgi için Managing the network state dokümanına bakabilirsin. {% endhint %}

Dikkat Etmen Gerekenler

- Bağlantı işlemleri uzun sürebilir
- UI Thread üzerinden yapılmamalıdır, aksi halde uygulamayı engelleyebilir
- Bağlantı işlemleri Asenkron İşlemler yazısına göre yapılmalıdır

Güvenlik Notları

- Veri tabanına direkt erişim **olmamalı**, API üzerinden erişim olmalı
- Reverse Engineering ile bağlantıda kullandığın bilgileri elde edebilirler

Faydalı Kaynaklar

{% embed url="https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-3-working-in-the-background/lesson-7-background-tasks/7-2-c-internet-connection/7-2-c-internet-connection.html" %}

---

description: 'Android üzerinden HTTP istekleri, request ve response yapısı'

HTTP İstekleri

HTTP İsteği Nedir?

- Web sitelerinden içerik istemek için kullanılırlar
- API işlemlerinde sıklıkla kullanılırlar
- GET, SET; POST vb. istek tipleri vardır

Volley ile Retrofit Farkı

Volley	Retrofit
Önbellek mekanizması	—
Bağlantıyı tekrardan gönderme	—
Gömülü olarak resim yüklenme desteği	—
—	JSON verilerini otomatik ayrıştırma

Volley Örneği

```
/**
 * https://developer.android.com/training/volley/simple.html
 *
 * @return
```

```
 */
static void requestNewsData(Context context, ResponseListener responseListener) {

    RequestQueue queue = Volley.newRequestQueue(context);
    StringRequest stringRequest = new StringRequest(Request.Method.GET, MAIN_URL,
(response) -> {
        // https://stackoverflow.com/a/9606629/9770490
        try {
            // new
JSONObject(response).getJSONArray("articles").get(1).getString("source")
            JSONObject responseObject = new JSONObject(response);

            String status = responseObject.getString("status");
            if (status.equals("ok")) {
                JSONArray articles = new
JSONObject(response).getJSONArray("articles");

                ArrayList<News> newsDataList = new ArrayList<>();
                for (int i = 0; i < articles.length(); i++) {
                    JSONObject article = articles.getJSONObject(i);
                    News news =  new News();

                    news.setTitle(article.getString("title"));
                    news.setDescription(article.getString("description"));
                    news.setUrlToImage(article.getString("urlToImage"));
                    news.setUrl(article.getString("url"));
                    news.setContent(article.getString("content"));

news.setSource(article.getJSONObject("source").getString("name"));
                    news.setPublishedAt(article.getString("publishedAt"));

                    newsDataList.add(news);

                    Log.i(TAG, "getNewsData: " + newsDataList.get(i));
                }

                responseListener.onResponse(newsDataList);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }, Throwable::printStackTrace);

    queue.add(stringRequest);
}

interface ResponseListener {
    void onResponse(ArrayList<News> newsDataList);
}
```

description: 'Android üzerinde internet üzerinden dosya indirme, ayrıştırma ve kullanma'

Dosya İndirme

URI Oluşturma

```
// Base URL for the Books API.
final String BOOK_BASE_URL =
   "https://www.googleapis.com/books/v1/volumes?";

// Parameter for the search string
final String QUERY_PARAM = "q";
// Parameter to limit search results.
final String MAX_RESULTS = "maxResults";
// Parameter to filter by print type
final String PRINT_TYPE = "printType";

// Build up the query URI, limiting results to 5 printed books.
Uri builtURI = Uri.parse(BOOK_BASE_URL).buildUpon()
       .appendQueryParameter(QUERY_PARAM, "pride+prejudice")
       .appendQueryParameter(MAX_RESULTS, "5")
       .appendQueryParameter(PRINT_TYPE, "books")
       .build();
```

İndirme İsteğinde Bulunma

```
private String downloadUrl(String myurl) throws IOException {
    InputStream inputStream = null;
    // Only display the first 500 characters of the retrieved
    // web page content.
    int len = 500;

    try {
        URL url = new URL(myurl);
        HttpURLConnection conn =
          (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000 /* milliseconds */);
        conn.setConnectTimeout(15000 /* milliseconds */);
        // Start the query
        conn.connect();
        int response = conn.getResponseCode();
        Log.d(DEBUG_TAG, "The response is: " + response);
        inputStream = conn.getInputStream();

        // Convert the InputStream into a string
        String contentAsString =
           convertInputToString(inputStream, len);
```

```
        return contentAsString;

    // Close the InputStream and connection
    } finally {
          conn.disconnect();
        if (inputStream != null) {
            inputStream.close();
        }
    }🕵 Detaylı bilgi için TEMP alanına bakabilirsin
```

{% hint style="warning" %} 📣 İstekte bulunmadan önce 🤱 Bağlantı Durumunu Yönetme alanından bağlantını kontrol etmelisin {% endhint %}

🔄 Veri Akışını Objeye Çevirme

```
// Reads an InputStream and converts it to a String.
public String convertInputToString(InputStream stream, int len)
        throws IOException, UnsupportedEncodingException {
    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

😜 Sonucu Ayrıştırma

{% page-ref page="../veriler/json-yoenetimi.md" %}

🧬 Faydalı Bağlantılar

- 📖 Concepts - 7.2: Internet connection
- 🎎 Code 7.2: AsyncTask and AsyncTaskLoader.

🦶 Giriş | Broadcast

👀 Broadcast Hızlı Bakış

- 📣 Haber salma olayı olarak ele alınabilir
- 🐟 İşletim sistemindeki her uygulamaya bildirilir
- ☆ 📶 `Wi-Fi'ya bağladım`, 🎧 `Cihazı başlattım` gibi örneklendirilebilir

✍ Yazılı Notlarım

description: Android üzerinde haber (Broadcast) oluşturma

🏗 Oluşturma | Broadcast

👀 Metotlara Göz Atma

🎙 Normal Broadcast

- 🏙 Sırasız olarak tüm uygulamalara duyurulan haber yapısıdır

```
public void sendBroadcast() {
   Intent intent = new Intent();
   intent.setAction("com.example.myproject.ACTION_SHOW_TOAST");
   // Set the optional additional information in extra field.
   intent.putExtra("data","This is a normal broadcast");
   sendBroadcast(intent);
}
```

🚄 Ordered Broadcast (Sıralı)

- 🚩 XML üzerinde belirlenen`android:priority` sırasına göre uygulamalara haber verir
- 🎲 Birden fazla aynı `android:priority` değerine sahip uygulamalara için seçim rastgele olur
- 👩‍🔧 Her duyuru alan uygulama, `intent` verisini değiştirme hatta silme hakkına sahiptir

```
public void sendOrderedBroadcast() {
   Intent intent = new Intent();

   // Set a unique action string prefixed by your app package name.
   intent.setAction("com.example.myproject.ACTION_NOTIFY");
   // Deliver the Intent.
   sendOrderedBroadcast(intent);
}
```

🏘 Local Broadcast (Yerel)

- 🏠 Sadece uygulama içerisinde haber salınır
- 👮 Daha güvenlidir, çünkü diğer uygulamalar erişemez
- 📈 Daha verimlidir, tüm sisteme haber salmakla uğraşılmaz

```
LocalBroadcastManager.getInstance(this).sendBroadcast(customBroadcastIntent);
```

{% hint style="info" %} 🕵 Detaylı bilgi için Broadcasts alanına bakabilirsin. {% endhint %}

👮 İzin Gerektirenler

- 📝 Manifest dosyası üzerinde `uses-permission` ile izin alınması gerekir
- 🚫 İzni olmayanlar uygulamaların erişmesi engellenir

```
sendBroadcast(new Intent("com.example.NOTIFY"),Manifest.permission.SEND_SMS);
```

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

---

description: Android üzerinde haber (broadcast) alma veya alıcılarının kullanımı

📡 Receiver | Broadcast

🎗 Broadcast Receiver Hakkında

- 🚫 UI thread üzerinden gerçekleştiğinden uzun işlemler yapılmamalı
- ⛔ `onReceive()` metodu içerisinde asenkron işlemler yapmayın
  - 👶 Yapsanız bile `return`metodundan sonra broadcast işlemleri sonlandırılır
  - ☠ Haliyle işlem asenkron olsa bile broadcast yapısına bağlı olduğundan ölecektir
- 🗨 `AlertDialog` gibi işlemler yerine `Notification` yapısı tercih edilmelidir

```
//Subclass of the BroadcastReceiver class.
private class myReceiver extends BroadcastReceiver {
   // Override the onReceive method to receive the broadcasts
   @Override
   public void onReceive(Context context, Intent intent) {
      //Check the Intent action and perform the required operation
        if (intent.getAction().equals(ACTION_SHOW_TOAST)) {
            CharSequence text = "Broadcast Received!";
            int duration = Toast.LENGTH_SHORT;

            Toast toast = Toast.makeText(context, text, duration);
            toast.show();
        }
   }
}
```

{% hint style="info" %} 🕵 Detaylı bilgi için Broadcast receivers alanına bakabilirsin. {% endhint %}

Receiver Türleri

Static Receiver

- Manifest üzerinden kayıt edilmeleri gerekir
- Uygulamamızı hedef almayan yayınlarını Android 8.0'dan itibaren alamaz
- implicit broadcast exceptions yayınlarını hala alabilmektedir

```
<receiver
 android:name=".AlarmReceiver"
 android:exported="false">
 <intent-filter>
     <action android:name=
          "com.example.myproject.intent.action.ACTION_SHOW_TOAST"/>
 </intent-filter>
</receiver>
```

Dynamic Receiver

- Uygulama üzerinden ilgilendiğimiz broadcast'e erişmek için `IntentFilter` kullanırız
- Genel kullanımı `onCreate` üzerinde yapılmaktadır (?)

```
 IntentFilter intentFilter = new IntentFilter();
 filter.addAction(Intent.ACTION_POWER_CONNECTED);
 filter.addAction(Intent.ACTION_POWER_DISCONNECTED);
```

Broadcast Kayıtları

- İlk olarak `receiver` yapısını uygulamamıza `registerReceiver` ile kaydederiz
- Genelde `onResume` içerisinde `registerReceiver` işlemi yapılır
- `onPause` içerisinde `unregisterReceiver` metodu ile kaldırırız

```
mReceiver = new AlarmReceiver();
this.registerReceiver(mReceiver, intentFilter);
```

```
 unregisterReceiver(mReceiver);
```

{% hint style="info" %} Detaylı bilgi için Broadcast receivers alanına bakabilirsin. {% endhint %}

Local Broadcast Alımı

- Local Broadcast, Dynamic Receiver ile alınmak zorundadır

```
LocalBroadcastManager.getInstance(this)
    .registerReceiver(mReceiver,
           new IntentFilter(CustomReceiver.ACTION_CUSTOM_BROADCAST));
```

```
 LocalBroadcastManager.getInstance(this)
    .unregisterReceiver(mReceiver);
```

İzin Gerektirenlerin Alımı

```
<receiver android:name=".MyBroadcastReceiver"
    android:permission="android.permission.SEND_SMS">
    <intent-filter>
        <action android:name="android.intent.action.AIRPLANE_MODE"/>
    </intent-filter>
</receiver>
```

```
IntentFilter filter = new IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
registerReceiver(receiver, filter, Manifest.permission.SEND_SMS, null );
```

Broadcast Kısıtlamaları

{% hint style="info" %} Detaylı bilgi için Restricting broadcasts alanına bakabilirsin. {% endhint %}

Broadcast Tavsiyeleri

Faydalı 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/7-3-c-broadcasts/7-3-c-broadcasts.html#broadcast_receivers" %}