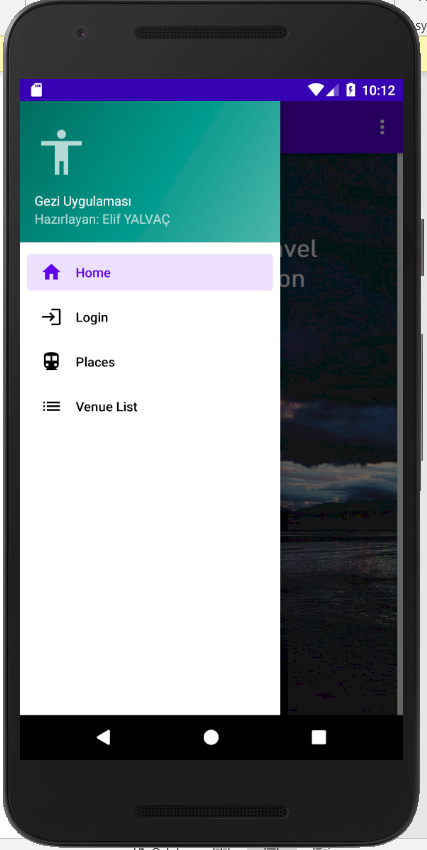
**KULLANIM KILAVUZU**

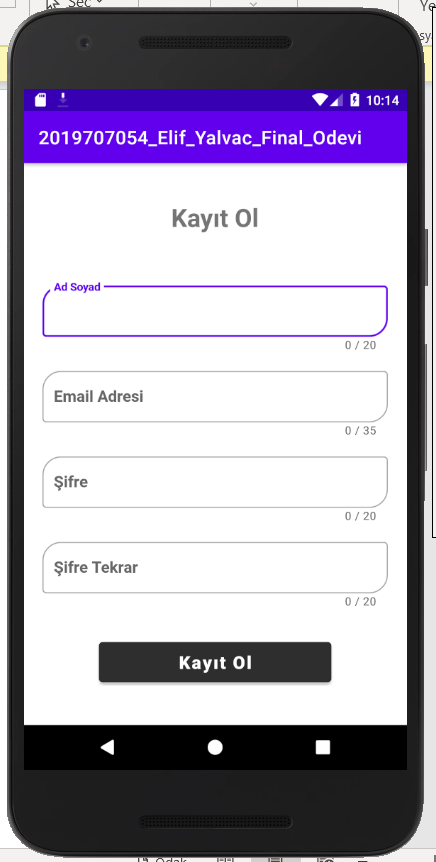
Uyulamayı çalıştırdığımızda karşımza ilk gelen ekran.

Sol tarafta Home, Login, Places ve Venue List isimlerinde dört adet menü var. Bu menüleri inceliycek olursak;

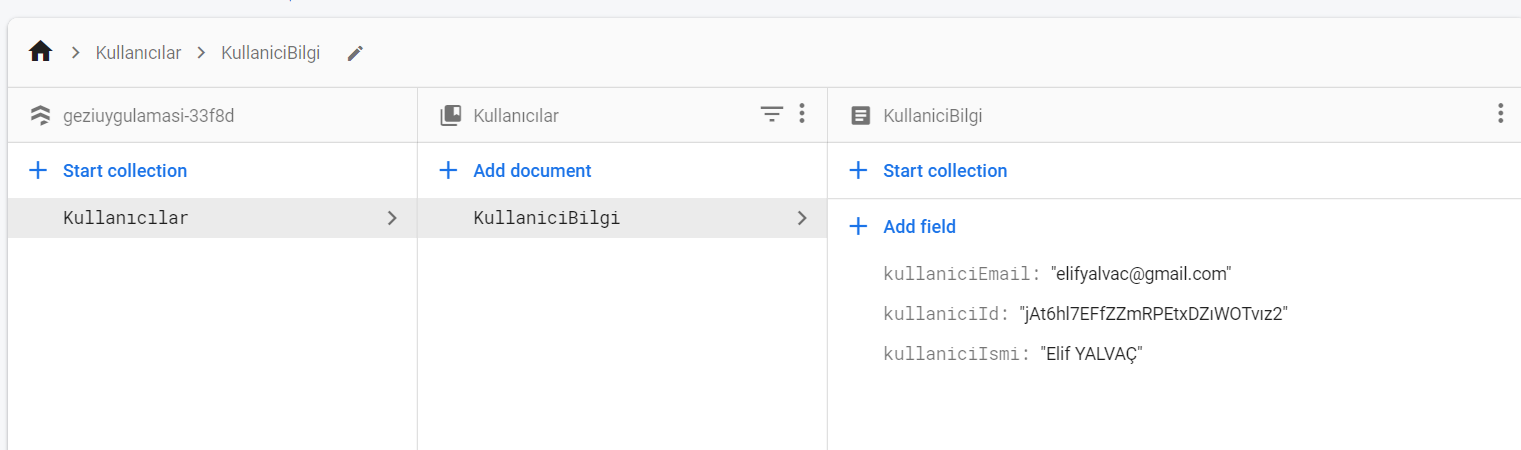
Home: İlk açılış ekranına yönlendiriyor. (İlk resim açılış sayfası)

Login: Kullanıcı girişi sayfasına yönlendiriyor.

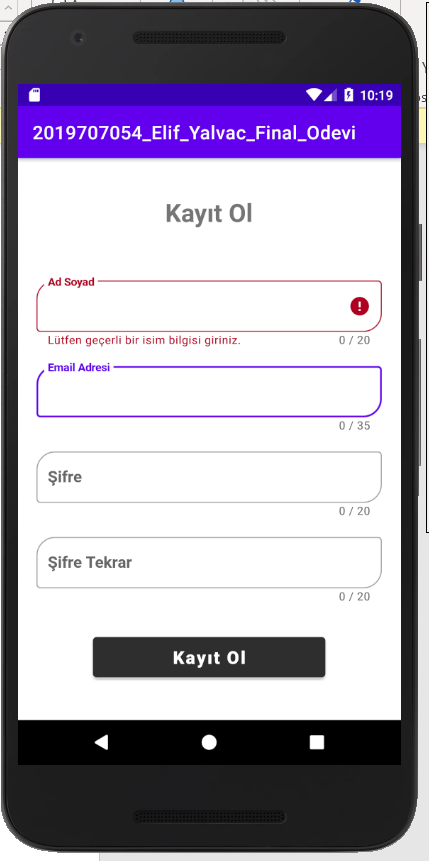
 

Eğer kayıtlı değilsek aşağıdaki Kayıt Ol butonuna basılınca da Kayıt olmak için ayrı bir form sayfasına atıyor bizi. Bilgileri yazıp kayıt ol tuşuna bastığımız zamana eposta, şifre gibi bilgiler firebase üzerinde kaydediliyor.



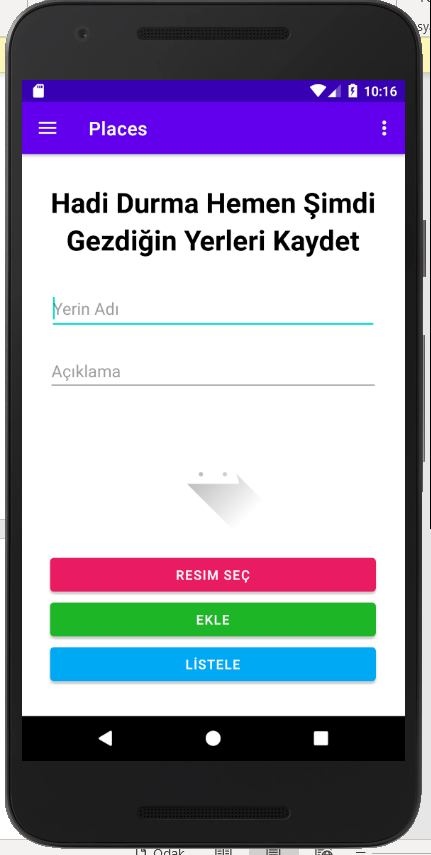


Ayrıca bu kısımda boş geçilme ya da yanlış veri girmeye karşın hata mesajları verilmekte.



Bu kısma da firebase kullandım, giriş yap kaydol kısımlarında. Bundan sonraki yerlerde de sqlite kullanarak resim yazı gibi verilerin eklemesini yaptım.

Places: Gezdiğimiz yerleri database kısmına eklememizi sağlayan form kısmına yönlendiriyor.



Venue List: Eklenen verilen yani gidilen yerlerin listelendiği sayfaya gitmektedir. (Bir önceki resimde de listele diye bir buton var bu buton aracılığı ile de bu sayfaya Venu List sayfasına gidilmektedir.)



**Kayit Ol kısmında kullandığım bazı önemli kodlar:**

public void btnKayitOl(View v) {  
 txtAdSoyad = editAdSoyad.getText().toString();  
 txtEmail = editEmail.getText().toString();  
 txtSifre = editSifre.getText().toString();  
 txtSifreTekrar = editSifreTekrar.getText().toString();  
  
 if (!TextUtils.*isEmpty*(txtAdSoyad)) {  
 if (!TextUtils.*isEmpty*(txtEmail)) {  
 if (!TextUtils.*isEmpty*(txtSifre)) {  
 if (!TextUtils.*isEmpty*(txtSifreTekrar)) {  
 if (txtSifre.equals(txtSifreTekrar)) {  
 mProgress = new ProgressDialog(this);  
 mProgress.setTitle("Kayıt Olunuyor...");  
 mProgress.show();  
  
 mAut.createUserWithEmailAndPassword(txtEmail, txtSifre)  
 .addOnCompleteListener(KayitOlActivity.this, new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful()) {  
 mUser = mAut.getCurrentUser();  
 if (mUser != null) {  
 mKullanici = new Kullanici(txtAdSoyad, txtEmail, mUser.getUid());  
 mFirestore.collection("Kullanıcılar").document(mUser.getUid())  
 .set(mKullanici)  
 .addOnCompleteListener(KayitOlActivity.this, new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 if (task.isSuccessful()) {  
 progressAyar();  
 Toast.*makeText*(KayitOlActivity.this, "Kaydınız alındı.", Toast.*LENGTH\_SHORT*).show();  
 finish();  
 startActivity(new Intent(KayitOlActivity.this,MainActivity.class).setFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP*));  
 } else {  
 progressAyar();  
 Toast.*makeText*(KayitOlActivity.this, task.getException().getMessage(), Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 }  
 } else {  
 progressAyar();  
 Toast.*makeText*(KayitOlActivity.this, task.getException().getMessage(), Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 } else  
 Toast.*makeText*(this, "Şifreler uyuşmuyor.", Toast.*LENGTH\_SHORT*).show();  
 } else  
 inputSifreTekrar.setError("Lütfen şifre bilgisini tekrar giriniz.");  
 } else  
 inputSifre.setError("Lütfen geçerli bir şifre belirleyiniz.");  
 } else  
 inputEmail.setError("Lütfen geçerli bir email adresi giriniz.");  
 } else  
 inputAdSoyad.setError("Lütfen geçerli bir isim bilgisi giriniz.");  
}

**Giriş Yap kısmında kullandığım bazı önemli kodlar:**

private void btnGirisYap (View v) {  
 txtEmail = editEmail.getText().toString();  
 txtSifre = editSifre.getText().toString();  
  
 if(!TextUtils.*isEmpty*(txtEmail)) {  
 if(!TextUtils.*isEmpty*(txtSifre)) {  
 mAuth.signInWithEmailAndPassword(txtEmail,txtSifre)  
 .addOnCompleteListener(GirisYapActivity.this, new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if(task.isSuccessful()) {  
 Toast.*makeText*(GirisYapActivity.this, "Başarıyla giriş yaptınız.", Toast.*LENGTH\_SHORT*).show();  
 finish();  
 startActivity(new Intent(GirisYapActivity.this,MainActivity.class).setFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP*));  
 }  
 else  
 Toast.*makeText*(GirisYapActivity.this, task.getException().getMessage(), Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 } else  
 inputSifre.setError("Lütfen geçerli bir şifre giriniz.");  
 } else  
 inputSifre.setError("Lütfen geçerli bir email adresi giriniz.");  
}

**Gidilen yerleri database ekleme kısmında kullandığım bazı sqlite kodlar:**

public class SQLiteHelper extends SQLiteOpenHelper {  
 public SQLiteHelper(@Nullable Context context, @Nullable String name, @Nullable SQLiteDatabase.CursorFactory factory, int version) {  
 super(context, name, factory, version);  
 }  
  
 public void queryData(String sql) {  
 SQLiteDatabase database = getWritableDatabase();  
 database.execSQL(sql);  
 }  
  
 public void insertData(String name, String aciklama, byte[] image) {  
 SQLiteDatabase database = getWritableDatabase();  
 String sql = "INSERT INTO Places VALUES (NULL, ?, ?, ?)";  
  
 SQLiteStatement statement = database.compileStatement(sql);  
 statement.clearBindings();  
  
 statement.bindString(1, name);  
 statement.bindString(2, aciklama);  
 statement.bindBlob(3, image);  
  
 statement.executeInsert();  
 }  
  
 public Cursor getData(String sql) {  
 SQLiteDatabase database = getReadableDatabase();  
 return database.rawQuery(sql, null);  
 }

**Eklenen verileri listeletmek için kullandığım kodların bir kısmı (PlaceListAdapter kısmı):**

package com.example.a2019707054\_elif\_yalvac\_final\_odevi;  
  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.BaseAdapter;  
import android.widget.ImageView;  
import android.widget.TextView;  
  
import java.util.ArrayList;  
  
public class PlacesListAdapter extends BaseAdapter {  
  
 private Context context;  
 private int layout;  
 private ArrayList<Places> placesList;  
  
 public PlacesListAdapter(Context context, int layout, ArrayList<Places> placesList) {  
 this.context = context;  
 this.layout = layout;  
 this.placesList = placesList;  
 }

@Override  
 public int getCount() {  
 return placesList.size();  
 }  
  
 @Override  
 public Object getItem(int position) {  
 return placesList.get(position);  
 }

@Override  
 public long getItemId(int position) {  
 return position;  
 }  
  
 private class ViewHolder {  
 ImageView imageView;  
 TextView textName, textAciklama;  
 }

@Override  
 public View getView(int position, View view, ViewGroup viewGroup) {  
 View row = view;  
 ViewHolder holder = new ViewHolder();  
  
 if(row == null) {  
 LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.*LAYOUT\_INFLATER\_SERVICE*);  
 row = inflater.inflate(layout, null);  
  
 holder.textName = (TextView) row.findViewById(R.id.*editName*);  
 holder.textAciklama = (TextView) row.findViewById(R.id.*editAciklama*);  
 holder.imageView = (ImageView) row.findViewById(R.id.*imagePlace*);  
 row.setTag(holder);  
 }  
 else {  
 holder = (ViewHolder) row.getTag();  
 }  
  
 Places places = placesList.get(position);  
  
 holder.textName.setText(places.getName());  
 holder.textAciklama.setText(places.getAciklama());  
  
 byte[] placesImage = places.getImage();  
 Bitmap bitmap = BitmapFactory.*decodeByteArray*(placesImage, 0, placesImage.length);  
 holder.imageView.setImageBitmap(bitmap);  
  
 return row;  
 }  
}

**Asynctask Sınıfı İle Arka Plan İşlemleri**

AsyncTask adında bir sınıf sunmaktadır. Bu sınıf içindeki metodlar yardımıyla arka planda farklı bir akış (Thread) üzerinde gerçekleştirilen işlemin kolayca ön yüzü etkileyen ana akışı (Main Thread) entegre edilmesini sağlar.

NOT: Arka planda Java Thread sınıfıyla çalıştırılan iş parçacıkları ön yüz elemanlarına müdahale edemez. Bunun için Handler adında bir sınıf kullanılmalıdır.

Aşağıda örnek bir AsyncTask sınıfı görüntülenmektedir;

public class BackgroundTask extends AsyncTask<Void, Void, Void> {

      @Override

      protected void onPreExecute() {

         super.onPreExecute();

      }

      @Override

      protected Void doInBackground(Void... params) {

         return null;

      }

      @Override

      protected void onPostExecute(Void result) {

         super.onPostExecute(result);

      }

      @Override

      protected void onProgressUpdate(Void... values) {

         super.onProgressUpdate(values);

      }

      @Override

      protected void onCancelled(Void result) {

         super.onCancelled(result);

      }

   }

AsyncTask sınıfı abstract bir sınıftır ve kullanılması için başka bir sınıf üzerinden **extend** edilmesi gerekir. AsyncTask sınıflarında **doInBackground** metodu bulunması zorunludur ve arka planda gerçekleştirilecek bütün işlemler bu metod içerisinde yazılır. Diğer metodları da açıklarsak;

* **onPreExecute:**Arka plan işlemi başlamadan önce ön yüzde değiştirilmesi istenen değişkenlerin (ProgressBar gibi animasyonlar) ve AsyncTask içinde gerekli değişkenlerin değer ataması yapılır.
* **doInBackground:**Arka planda yapılması istenen işlem burada gerçekleşir. Bu metod içinde yapılan işlemler ön yüzde kullanıcının uygulamayı kullanmasını kesinlikle etkilemez. Eğer buradaki işlemler sonucunda ana akışa bir değişken gönderilmesi gerekiyorsa **return** metodu ile bu değişken **onPostExecute** metoduna paslanabilir.
* **onPostExecute:** doInBackground metodu tamamlandıktan sonra işlemlerin sonucu bu metoda **result** değişkeni ile gönderilir. Buradaki işlemler ana akışı etkiler ve herhangi bir hataya sebep olmaz. Arka plandaki işlemden gelen bir veri ön yüzde gösterilmek isteniyorsa bu metod içinde gösterim işlemi yapılabilir.
* **onProgressUpdate:**Eğer doInBackground metodu içerisinde yaptığınız işlemin ilerleme durumunu (örneğin dosya indirme yüzdesi) kullanıcıya bildirmek istiyorsanız bu metodu kullanabilirsiniz. doInBackground metodu içerisinde publishProgress metodunu kullanarak ilerleme durumunu onProgressUpdate metoduna iletip önyüz içerisinde buna göre bir animasyon yapabilirsiniz.
* **onCancelled:**Eğer herhangi bir sebepten dolayı AsyncTask iptal edilirse bu metod uyarılır. Burada kullanıdığınız kaynakları temizleyebilirsiniz.

Şimdi basit bir uygulama ile AsyncTask kullanarak nasıl bir işlem yapacağımızı ve işlem sürerken kullanıcıya işlemin durumunu nasıl bildireceğimizi gösterelim. Öncelikle sadece tek bir düğmeden oluşan bir layout dosyası hazırlayalım ve düğmeye aşağıdaki aksiyonu verelim;

public class MainActivity extends Activity {

   private ProgressDialog progressDialog;

   @Override

   protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.activity\_main);

      ((Button) findViewById(R.id.startAsync)).setOnClickListener(new OnClickListener() {

         @Override

         public void onClick(View v) {

            new BackgroundTask().execute((Void) null);

         }

      });

   }

}

Yukarıdaki kodumuzda kullanıcı düğmeye bastığında **BackgroundTask** adındaki AsyncTask sınıfı **execute** metodu kullanılarak başlatılacaktır. Eğer AsyncTask’a bir değişken göndermek istiyorsak bunu **execute** metodu içerisinde verebiliriz. Bu durumda değişkenler direkt **doInBackground** metoduna gönderilecektir. **progressDialog** değişkeni ise kullanıcıyı arka plan işleminin durumunu bildirmek için kullanılır.

NOT: AsyncTask’ın oluşturulma yapısına dikkat ederseniz AsyncTask<Tip1,Tip2,Tip3> şeklinde bir yapı görürsünüz. Burada Tip1 doInBackground metoduna verilecek parametrelerin tipini ya da sınıfını belirler (örnekte Void). Tip2 doInBackground metodunun işleyişi sırasında onProgressUpdate metoduna paslanacak değişkenin tipini ya da sınıfını belirtir. Tip3 ise onPostExecute metoduna verilen değişkendir ve aynı zamanda doInBackground metodunun **return** tipidir. (http://developer.android.com/reference/android/os/AsyncTask.html)

public class BackgroundTask extends AsyncTask<Void, Integer, Void> {

      @Override

      protected void onPreExecute() {

         super.onPreExecute();

         progressDialog = new ProgressDialog(MainActivity.this);

         progressDialog.setMax(100);

         progressDialog.setProgress(0);

         progressDialog.setProgressStyle(ProgressDialog.STYLE\_HORIZONTAL);

         progressDialog.show();

      }

      @Override

      protected Void doInBackground(Void... params) {

         for (int i = 0; i < 101; i = i + 10) {

            try {

               publishProgress(i);

               Thread.sleep(1000);

            }

            catch (InterruptedException e) {

            }

         }

         return null;

      }

      @Override

      protected void onPostExecute(Void result) {

         super.onPostExecute(result);

         progressDialog.dismiss();

      }

      @Override

      protected void onProgressUpdate(Integer... values) {

         super.onProgressUpdate(values);

         Integer currentProgress = values[0];

         progressDialog.setProgress(currentProgress);

      }

      @Override

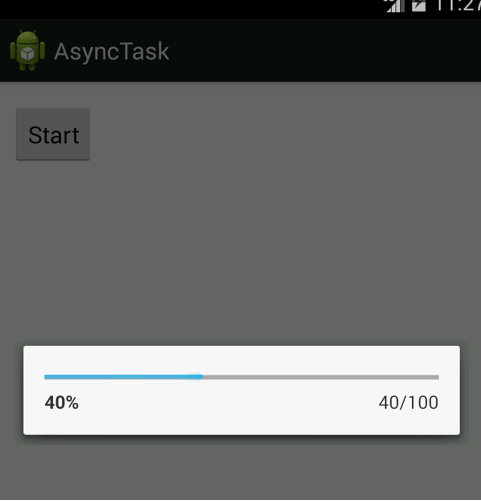
      protected void onCancelled(Void result) {

         super.onCancelled(result);

         progressDialog.dismiss(); } }

Buradaki metodların yaptıklarını sırasıyla incelersek;

* **onPreExecute:**Bu metod içerisinde işlem başlatılmadan önceki tanımlamalar yapılmıştır. ProgressDialog değişkenine ait tanımları bu metod içerisinde gerçekleştiririz. **show** metodu ile ekrana durum çubuğunu getiriyoruz.
* **doInBackground:**Burada yaklaşık 10 saniye sürecek bir işlem gerçekleştiriyoruz. Bu işlem sırasında her 1 saniyede publishProgress metodunu kullanarak ilerlemeyi onProgressUpdate metoduna gönderiyoruz.
* **onPostExecute:** İşlem tamamlandıktan sonra ekrandaki durum çubuğunu **dismiss** metoduyla yok ediyoruz.
* **onProgressUpdate:**Durumu çubuğunun ilerlemesi doInBackground metodundan publishProgress ile gönderiliyor. Burada da gelen rakamsal değerle durum çubuğunu güncelliyoruz.
* **onCancelled:**Eğer işlem herhangi bir sebepten dolayı iptal olursa durum çubuğunu **dismiss** metoduyla yok ediyoruz.



*NOT: AsyncTask’lar içerisinde çalıştıkları Activity herhangi bir sebepten dolayı ölürse yaptıkları işlemler tamamlanmayabilir. Bu yüzden Android SDK içerisinde kısa süreli işlemler ya da Activity yaşadığı sürece gerçekleşmesi beklenen işlemler için kullanılmaları tavsiye edilir. Çok uzun süreli arka plan işlemleri (örneğin veritabanı güncelleme, dosya indirme) için****Service****sınıflarının kullanılması tavsiye edilir.*

# **Broadcast Receivers (Yayın Alıcılar) Nedir?**

# Android sistem tarafından, sistem genelinde yapılan duyurulara cevap veren bir bileşendir. Birçok yayın, sistem tarafından yapılır. Bu yayınlara örnek olarak, resim çekilmesi, batarya gücünün az olması gibi durumlarda sistem genelinde bir yayın yapılır. Bu bileşenlerin arayüzü yoktur. Bir yayın olayı meydana geldiğinde kullanıcı bildirimler vasıtasıyla uyarılır. Bir yayın alıcı oluşturmak için BroadcastReceiver sınıfı kullanılır.

### BroadcastReceiver Sınıfı

sendBroadcast() tarafından gönderilen intent nesnesini alan bir sınıftır. Bir yayın alıcı oluşturmak için bu sınıf kullanılır. Sistem tarafından gönderilen yayınlar, bu sınıf ile gelen onReceive() metodu ile alınır. Oluşturulan sınıf kayıt edildikten sonra, sistem onReceive() metodunu başlatır.

### BroadcastReceiver Oluşturmak

BroadcastReceiver oluşturmak için öncelikle bir sınıf oluşturmalıyız. Oluşturulan bu sınıfın bir yayın alıcısı olması için BroadcastReceiver sınıfından extend edilmesi gerekiyor. Daha sonra onReceive() metodunun eklenmesi ile yayın alıcımız meydana gelir.

public class FirstReceiver extends BroadcastReceiver{

@Override

public void onReceive(Context context, Intent intent) {

// Code

}

}

### Broadcast Receiver (Yayın Alıcının) Kayıt Edilmesi

Uygulamaların sistem tarafından verilen yayınları alabilmesi için oluşturulan broadcast receivers kayıt edilmesi gerekmektedir. Eğer bu işlem yapılmazsa sistemin verdiği yayınlar uygulama tarafından alınamazlar. Oluşturulan bir broadcast receiver kayıt etmek için iki yol bulunur.

* AndroidManifest.xml dosyasına kayıt etmek.
* registerReceiver() metodu ile dinamik olarak kayıt edilmesi.

#### **AndroidManifest.xml Dosyasına Kayıt Etmek**

BroadcastReceiver sınıfı ile oluşturduğumuz yayın alıcısını, maifest dosyasına kayıt etmek için <receiver> etiketini kullanırız.

<application>

<receiver android:name="FirstReceiver">

<intent-filter>

<action android:name="com.codejuks.CUSTOM\_INTENT"/>

</intent-filter>

</receiver>

</application>

Bu şekilde bir broadcast receiver kayıt etmiş oluruz.

#### **registerReceiver() ile Kayıt Edilmesi**

Broadcast Receiver dinamik olarak kayıt etmek için registerReceiver() metodunu kullanabiliriz. Broadcast receiver dinamik olarak kayıt ettiğimizde uygulamamız daha güvenli olur.

FirstReceiver mReceiver = new FirstReceiver();

registerReceiver(mReceiver,new IntentFilter("com.codejuks.CUSTOM\_INTENT"));

Bu metodun iki adet parametresi bulunur.

**receiver**, kayıt edilecek yayın alıcı bu parametre ile belirlenir.

**filter**, alınacak intent yayınları bu parametre ile belirlenir. Buraya verilen “com.codejuks.CUSTOM\_INTENT” değerini kendimiz belirledik. Bu eylem tipine sahip olan yayınlar alınabilir.

### Receiver Lifecycle (Alıcı Yaşam Döngüsü)

Bir yayın alıcısının yaşam döngüsünde sadece onReceive(Context,Intent) metodu bulunmaktadır. BroadcastReceiver bir intent yayını aldığında bu metot çağrılır. Bu metot daima aynı sürecin main thread içerisinde çağırılır. Bundan dolayı uzun bir zanaba yayılan işlemleri bu metot içinde yapmamalıyız. Burada 10 sn’lik bir zaman aşımı vardır. Bu zaman aşımından sonra broadcast receiver işlev yapamayacak hale gelir.

## INTENT NEDİR?

**Intent**, uygulama bileşenleri arasında veri alışverişini ve bilgi aktarımını sağlayan bir haberleşme **nesnesidir**. Bu nesne Android uygulamalarının vazgeçilmez bir bileşenidir. Bir **Activity**‘ den başka bir **Activity**‘ e geçişi sağlayan veya **bilgi aktarımını** yapan bu**Intent nesnesidir.** Nesne olmasının sebebi **Intent**sınıfından üretilerek kullanılmasından dolayıdır. Şimdi **intent**nesnesini kullanarak neler yapabiliyoruz, buna bakalım.

## Activity Bileşenini Başlatmak

Önceki yazımızdan hatırlayacak olursak **Activity**, uygulamada **tek bir ekranı** temsil ediyordu. Bir ekran oluşturmak için **iki bileşene ihtiyacımız** vardır. Biri **.xml**uzantılı dosyamız diğer de . **java**uzantılı bir sınıftır. Bu ikisi bir araya gelince kullanıcı için yapmış olduğumuz o şaheser arayüz tasarlamış oluyoruz.

**Activity** dediğimiz aslında bir bileşendir fakat aynı zamanda da bir sınıftır. O yüzden yazdığımız java sınıfını **Activity** sınıfından **extend** etmemiz gerekir. Java dosyasında **xml** ile iletişimi sağlamak için **setContentView()** metodunu kullanılırız.

.setContentView(R.layout.activity\_main);

**activity\_main**, **.xml uzantılı** dosyamızın adıdır.

Oluşturulan bir Activity sınıfını başlatmak için bir intent nesnesi oluşturulur. ve bu intent nesnesini **startActivity()** metodu ile başlatırız. Bu metod ile başlatılan Activity bileşeninden herhangi bir bilgi, veri alınmaz. Sadece bizi diğer Activity ekranına götürür.

Intent nedir?

Intent intent = new Intent(getApplicationContext(), Main2Activity.class);

startActivity(intent);

İlk satırda Intent sınıfından**intent**nesnesini oluşturduk. İlk parametremiz yani **getApplicationContext()** yazan kısıma; hangi **Activity**içerisinde olduğumuzu belirtiyoruz. Oraya **this** anahtar kelimesini de yazabilirdik. Yani bulunduğumuz sınıfı temsil ettiğini programa söylüyoruz. İkinci parametrede ise yani**Main2Activity.class** yazan kısım **hangi Activity’e,**hangi sınıfa gitmek istediğimizi belirtiyoruz.

İkinci satırda oluşturduğumuz intent nesnesini **startActivity(intent)** diyerek başlatmış olduk.

Eğer başlatılan Activity bileşeni bitirildikten sonra bir veri almak istersek **startActivity()** metodu yerine bu sefer **startActivityForResult()** metodunu yazarız. Yani başlatılan Activity’den bir sonuç, veri beklediğimizi belirtiyoruz. Bu metot ile alınan veri **onActivityResult()** metoduna iletilir.

## Bir Service(Servis) Başlatmak

Servisler, kullanıcıdan bağımsız olarak arka planda uzun süren işlemlerimizi yapmamızı sağlayan ve arayüzü olmayan uygulama bileşenleridir. Buna en iyi örnek müzik uygulamaları verilebilir. Eğer müzik uygulamaları **service** bileşeni olarak tasarlanmasaydı o zaman biz müzik ekranından çıktığımız gibi müzik kesilecekti. Bu yüzden de telefonda hem müzik dinleyip hem Instagram’da, Twitter’da dolaşamayacak hatta mesajlaşma bile yapamayacaktır.

Hazırlanan bir servis bileşenini başlatmak için **startService()** metodu kullanılır. Bu metod ile **intent** başlatılır ve servis başlatılır**.** **Activity** ile aynı, sadece **Activity** yerine **Service** diyoruz. Servis **client – server arayüzü** ile tasarlanmış ise bu durumda **bindService()** metodu kullanılır.

## Broadcast Almak

Android işletim sistemi tarafından yayınlanan mesajlardır. Örneğin telefonumuz açıldığında veya bataryamız azalmaya başladığında bazı yayınlar yapılır. Herhangi bir uygulama, sisteminin yayınladığı bu yayınları alıp kullanabilir.

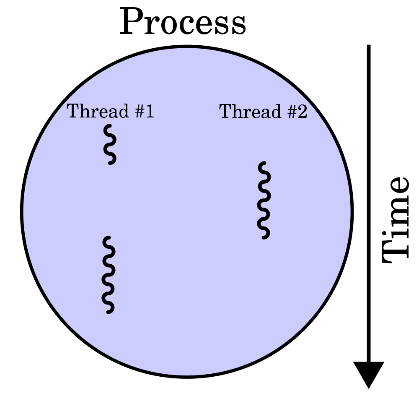
**Intent** nesnesi,**sendBroadcast() , sendOrderedBroadcast()** veya **sendStickBroadcast()**metodları ile broadcast yayınını alabilir. Örneğin telefon açıldığında başlayan bir uygulamamız olsun. Bunun için önce yayını alıp ondan sonra uygulamayı başlatması gerekecektir. İşte Intent nesnesi önce bu yayını alır ardından uygulamayı başlatır.

Fark ettiyseniz biz önceki makalemizde 4 tane uygulama bileşeninden bahsetmiştik. Bunlar; Activity, Service, Broadcast Receivers ve Content Providers bileşenleriydi. Bu makale de ise sadece 3 tane bileşenden bahsettik. Çünkü; Content Providers, Intent nesnesi ile değil, ContentResolver() ile başlatılır. O yüzden bu bileşeni burada anlatmadık.

Bu makalemizi daha fazla uzatmamak için burada bırakıyorum. Bir sonraki makalemizde Intent türlerine bakacağız. İyi günler dilerim.

#### **Threadler**

Birden fazla işlemin tek bir program akışı içinde gerçekleştirilmesi Java'da Thread sınıfları sayesinde gerçekleştirilmektedir. Şu ana kadar yaptığımız bütün örnekler ana program akışını takip ederek gerçekleştirilen işlemleri içermekteydi. Thread (iş parçacığı) kullanımı, birden fazla işlemin tek bir akışı paylaşarak neredeyse eşzamanlı bir şekilde gerçekleşmesini sağlar.



Thread kullanımına en iyi örnek oyun uygulamalarıdır. Kelime oyunu uygulamasında kullanıcının ana ekranda oyunu oynadığını düşünelim. Biz ise tam oyunun en heyecanlı yerinde rasgele bir reklam gösterip kullanıcının bütün dikkatini reklama yöneltmek isteyelim :). Bu amaçla bizim reklamı uzaktaki bir sunucudan çekip ana ekrana basan bir kod parçası yazmamız gerekmektedir. Uzaktan dosya yükleme uzun bir işlem olduğu için  işlem sırasında oyunu bloke etmememiz gerekir. Bu yüzden Thread mantığını kullanan bir kod yazarak ana akışta hem oyunu hem de yükleme işlemini ilerleterek iki işlemin de mevcut kaynakları paylaşarak kullanmasını sağlayabiliriz. Bu sayede hem kullanıcı oyununu kesintisiz oynamaya devam eder hem de reklam dosyaları uygulamaya yüklenmiş olur.

Java'da Thread kullanan program örneği ise aşağıda verilmiştir;

public class MyThread implements Runnable {

private int end;

private String name;

public MyThread(String name, int end) {

this.end = end;

this.name = name;

}

@Override

public void run() {

for (int i = 0; i < end; i++) {

System.out.println(name + " : " + i);

}

}

}

Thread içerisinde gerçekleştirilecek işlemler Öncelikle **Runnable** Interface'ten üretilmiş herhangi bir sınıfta tanımlanmalıdır. Yukarıda yer alan örnek kodda MyThread adında tanımlanmış bir iş parçacığı bulunmaktadır. Runnable içerisindeki **run** metodu içinde ise yapılması gereken işlemler belirtilir. Örnek kod yapıcı (constructor) metot içinde verilen bir int değeri kadar sayma işlemi yapacak ve bunu konsola Thread'e verilen isimle birlikte yazacaktır.

public class ThreadLesson {

public static void main(String[] args) {

Thread thread1 = new Thread(new MyThread("thread1", 6));

Thread thread2 = new Thread(new MyThread("thread2", 5), "thread2");

thread1.start();

thread2.start();

}

}

MyThread adlı iş parçacığını çalıştırmak için **Thread** adlı sınıftan faydalanırız. Yukarıdaki örnekte farklı Thread tanımlamaları ve bunların kullanımları görülmektedir. Thread'ler tanımlanırken yapıcı içerisinde Thread'e ait bir isim de verilebilir. Yukarıdaki kodun çıktısı aşağıdaki gibidir;

thread1 : 0  
thread2 : 0  
thread2 : 1  
thread2 : 2  
thread1 : 1  
thread2 : 3  
thread1 : 2  
thread2 : 4  
thread1 : 3  
thread1 : 4  
thread1 : 5