7. Comunicaciones.

Contenido

1. Programación de comunicaciones

Cada vez es más habitual que las aplicaciones hagan uso de la comunicación a Internet, integrándose en las redes sociales o utilizando servicios web. En este apartado se verá cómo hacer uso de las comunicaciones del dispositivo para comunicarse con el exterior.

Dado que Android es un sistema diseñado para dispositivos móviles, la **conexión continua** es uno de sus aspectos fundamentales. Véase comunicación web y los servicios de mensajería instantánea y multimedia.

1.1. Comunicaciones en Android

El primer paso para trabajar con la API de comunicaciones en Android es pedir los permisos para su aplicación al documento de manifiesto. Esto lo puede hacer añadiendo las siguientes líneas en dicho documento:

```
1 <uses-permission android:name =
              "android.permission.INTERNET"/>
2 <uses-permission android:name=
        "android.permission.ACCESS_NETWORK_STATE "/>
```

Esto permite a su aplicación tener acceso a Internet (abrir *sockets* de red) e información del estado de la red (por ejemplo, para saber si el Wi-Fi está activado o desactivado).

Antes de hacer un intento de conexión a la red, se deberá verificar si el dispositivo tiene una conexión de red disponible y funcionando correctamente. Por ello se debe utilizar objetos de la clase ConnectivityManager y NetworkingInfo de la siguiente manera:

```
1 // Obtenemos un gestor de las conexiones de red
2 ConnectivityManager connMgr = (ConnectivityManager)
         getSystemService (Contexto.CONNECTIVITY_SERVICE);
3
4 // Obtenemos el estado de la red
5  NetworkInfo networkInfo = connMgr.getActiveNetworkInfo ();
6
7  // Si está conectado
8  if (networkInfo! = null && networkInfo.isConnected ()) {
9       // Red OK
10     Toast.makeText (this, "Red ok", Toast.LENGTH_LONG) .show ();
11 } else {
12      // Red no disponible
13     Toast.makeText (this, "Red no disponible",
                  Toast.LENGTH_LONG).show ();
14  }
```

Con getSystemService () se obtiene un objeto ConnectivityManager que sirve para gestionar la conexión de red. Con este objeto tiene una forma el objeto NetworkInfo, con getActiveNetworkInfo () que devuelve una instancia de NetworkInfo, que representa la primera interfaz de red que puede encontrar o un *null* si ninguna conexión está conectada.

NetworkInfo tiene el método isConnected (), que indica si existe conectados actividad en la red y si es posible establecer conexiones. En caso de que se desee obtener información individual de los diferentes tipos de redes, se puede hacer de forma individual con getNetworkingInfo (), con un argumento que es una constante que indica el tipo de red que desea comprobar, como se puede ver en el siguiente código:

```
1 // Obtenemos el estado de la red móvil
2 NetworkInfo networkInfo = connMgr.getNetworkInfo
         (ConnectivityManager.TYPE_MOBILE);
3 boolean connectat3G = networkInfo.isConnected();
4
5 // Obtenemos el estado de la red wifi
6 networkInfo = connMgr.getNetworkInfo (
          ConnectivityManager.TYPE_WIFI);
7 boolean connectatWifi = networkInfo.isConnected();
```

Es muy importante que siempre se compruebe la conexión a la red y no de por hecho que la red está disponible. Piense que las operaciones de red a un dispositivo móvil suelen ser móviles, 3G, 4G o Wi-Fi, y existen posibilidades reales de caída de la conexión. Por lo tanto, se deben hacer estas comprobaciones cuando se enciende la aplicación o cuando se vuelva después de haber salido. El lugar ideal para hacerlo sería el método onStart () de su aplicación.

Imagina que enciendes la aplicación y se hace la comprobación de conexión al comienzo, y encuentra que existe conexión. Si el usuario ejecuta la aplicación de configuración del dispositivo y activa el modo de avión, su aplicación seguirá funcionando correctamente, ya que cuando el usuario vuelva a su aplicación onStart() volverá a ser llamado y detectará que la red está desconectada.

1.2. BroadcastReceiver

Pero ¿qué pasa cuando la conexión a Internet cambia de estado mientras está dentro de su aplicación? Esto puede ocurrir porque el dispositivo pierde la cobertura o la conexión a la red. Como no se vuelve a llamar onStart (), no podría comprobar la conectividad. Por eso, lo que necesita hacer es registrar un B*roadcastReceiver* (receptor multidifusión) que escuchará los cambios en el estado de la conexión y comprobará cuál es el estado dela red.

En primer lugar, se creauna clase que herede de BroadcastReceiver e implemente el método abstracto onReceive ().

```
1 public class ReceptorRed extends BroadcastReceiver {2
3@Override
4 public void onReceive (Context arg0, Intent arg1) {
5        // Actualizar el estado de la red
6          ActualizaEstadoRed();
7  }
8}
```

Si esta clase se utilizará dentro de su actividad no debe crear la clase como un archivo .java independiente en su proyecto, puede ser definida dentro de la clase de su aplicación. Dentro del método onReceive () actualizar el estado de la red a su aplicación.

El siguiente paso es registrar el receptor de *broadcast* para que escuche los mensajes relacionados con los cambios de la conectividad de la red. Para ello, se crea una instancia de la clase receptora y se registra con un filtro de *intent* para escuchar únicamente los mensajes de *broadcast* para abrir la conectividad del dispositivo.

```
1 private ReceptorRed receptor;
2
3 @Override
4 public void onCreate (Bundle savedInstanceState) {
5     (...) // Resto de código onCreate ()
6
7 IntentFilter filter = new IntentFilter
      (ConnectivityManager.CONNECTIVITY_ACTION);
8
9 receptor = new ReceptorRed ();
10 this .registerReceiver (receptor, filter);
```

El método registerReceiver () registra un receptor de *broadcast* para ejecutarse el hilo principal de la aplicación. El receptor de *broadcast* escuchará cualquier *intent* de *broadcast* que coincida con el filtro definido en el hilo principal de la aplicación.

En este momento, los mensajes de *broadcast* que anuncien los cambios en la conectividad de la red serán recibidos por su receptor, el cual actualizará el estado de la red de la aplicación. Tener un receptor de *broadcast* que es llamado constante o innecesariamente puede derivar en un consumo excesivo de los recursos del sistema (entre otros, de la batería). Si declara el receptor de *broadcast* en el documento de *manifiesto, este* puede poner en marcha la aplicación automáticamente, aunque esta no esté en ejecución.

Por este motivo, lo más adecuado sería registrar el receptor de *broadcast* cuando se crea la aplicación onCreate () y darlo de baja al método onDestroy ().

```
1 public void onDestroy () {
2     super.onDestroy ();
3
4    // Damos de baja el receptor de broadcast cuando se destruye la
      // aplicación
5     if (receptor! = null) {
6          this.unregisterReceiver(receptor);
7     }
8}
```

1.3. Permisos a partir de la versión 6

A partir de Android 6 Marshmallow (API 23) los permisos deben estar en el manifest pero el usuario debe aprobar cada uno de los permisos explícitamente durante la ejecución.

Por ejemplo, si deseamos obtener el permiso de comunicación a través de Internet para cargar una imagen en segundo plano, habrá que incluir este código:

```
void permisosCargaImagen(){
     //Comprobamos que la versión sea mayor que Marshmallow y
       todavía no tenemos asignado el permiso de Internet
     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
          && getActivity().checkSelfPermission(Manifest.permission.INTERNET)
!= PackageManager.PERMISSION_GRANTED) {
        //Solicitamos el permiso, al finalizar llamará
           a la función onRequestPermissionsResult
        requestPermissions(new String[]{Manifest.permission.INTERNET},
             MY_PERMISSIONS_REQUEST_INTERNET);
    }
    else {
         //El permiso está concedido y procedemos a cargar la imagen
            en segundo plano
        CargaImagen cargaImagen = new CargaImagen();
        cargaImagen.execute("http://4.bp.blogspot.com/-
SiQNg1gmn5o/UyDYG1mMjjI/AAAAAAAAATs/wqi8OvtYvME/s1600/periodico.png");
    }
}
```

```
@Override
public void onRequestPermissionsResult(int requestCode,String permissions[],
int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_INTERNET: {
            // Si el permiso se ha concedido volvemos a llamar
                  a la función para cargar la imagen
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(getActivity(),
                        "Permiso concedido", Toast.LENGTH_LONG).show();
                permisosCargaImagen();
            } else {
                Toast.makeText(getActivity(),
                        "Permiso denegado",  Toast.LENGTH_LONG).show();
            }
            return;
        }
    }
}
```

1.4. Mostrar páginas web con un 'widget'

Posiblemente, el servicio de comunicación más utilizado hoy en día son las páginas web, por lo que comenzaremos mostrando cómo podemos visualizarlas dentro de las nuestras aplicaciones.

El procedimiento es muy sencillo porque hay un componente llamado **WebView** que ya hace casi todo el trabajo, sólo hay que decirle qué dirección queremos visitar.

Crea un nuevo proyecto Android. Como que su aplicación accederá, lógicamente, en Internet, se debe añadir el permiso correspondiente al fichero **AndroidManifest.xml** (a continuación del uses-sdk):

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

Guarda y ve al **layout.xml** para añadir el componente WebView, además de un EditText para introducir la dirección web y un botón para acceder:

```
1<?xml version="1.0" encoding="utf 8"?>
2<LinearLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
3 android:layout_width="fill_parent"
4 android:layout_height="fill_parent"
5 android:orientation="vertical">6
7       <LinearLayout
8       android:id="@+id/linearLayout1"
9        android:layout_width="match_parent"
10      android:layout_height="wrap_content"
11      android:orientation="horizontal">12
13      <EditText
14      android:id="@+id/editText1"
15      android:layout_width="264dp"
16      android:layout_height="wrap_content"
17      android:hint="Introduce la dirección web">18
19              <requestFocus/>
20      </EditText>
21
22      <Button
23      android:id="@+id/button1"
24      android:layout_width="wrap_content"
25      android:layout_height="wrap_content"
26      android:onClick="onClickIr"
27      android:text="Ir"/>
28
29      </LinearLayout>
30
31      <WebView
32      android:id="@+id/webView1"
33      android:layout_width="match_parent"
34       android:layout_height="match_parent"/>
35
36 </LinearLayout>
```

Solo hay que ir al código de la actividad para añadir la funcionalidad necesaria. Básicamente, se trata de escribir el método onClickIr() quese activará cuando pulsamos el botón, y que tiene que decir alWebView que abra una página:

```
1 public void onClickIr (View v) {
2   WebView webView = (WebView) findViewById(R.id.webView1);
3   EditText editText = (EditText) findViewById (R.id.editText1);
4
5   String direccion = editText.getText().ToString ();
6   webView.loadUrl(direccion);
7}
```

Con ello podemos ejecutar la aplicación, indicar una web e ir, tal como se puede ver en la imagen:

Observará que, si escribe una dirección sin el prefijo "http://", la aplicación da un error. Esto es porque realmente la dirección completa debe incluir este prefijo (que es el protocolo de aplicación). Por lo tanto, puede añadir el siguiente código para que lo haga, editando el método onClickIr():

```
1 String direccion = editText.getText().ToString ();2
3  if (!direccion.startsWith ("http://") &&
                !direccion.startsWith ("https://")) {
4                       direccion = "http://" + direccion;
5     editText.setText (direccion);6   }
7
8   webView.loadUrl(direccion);
```

Se puede comprobar haciendo doble clic en la página la vista se acerca (hace zoom) y le permite arrastrarla para irse desplazando por todos sus contenidos según necesite, así que puede pulsar enlaces y hacer cualquier otra actividad con las páginas que vaya abriendo.

2. Proveedores de contenidos

Los *content providers* (proveedores de contenidos) dan acceso a una serie estructurada de datos y sirven de interfaz de datos estándar para conectar datos de un proceso con el código que se está ejecutando con otro proceso. **Los proveedores de contenidos son la forma recomendada de compartir datos entre aplicaciones.** Son almacenes de datos que sirven para compartir datos entre aplicaciones. Se comportan de forma similar a una base de datos (puede hacer consultas, editar el contenido, añadir y borrar), pero, a diferencia de estas, utilizan diferentes formas para almacenar sus datos. Estas pueden estar en una base de datos, en ficheros o incluso en la red, pero para el proceso que las utiliza esto es transparente (y nos resulta indiferente).

Android utiliza una serie de proveedores de contenidos, estándares del sistema, que pueden utilizar el resto de aplicaciones, por ejemplo:

- *Browser: almacena datos como los marcadores del navegador, el historial* de navegación, etc.
- *CallLog: almacena datos como llamadas perdidas, detalle de las llamadas,* etc.
- *ContactsContract: almacena información de los contactos: nombre,* email, teléfono, fotos, etc.
- *MediaStore: almacena datos multimedia como imágenes, audio y vídeo.*
- *Settings: almacena datos de configuración y preferencias del dispositivo* como el *bluetooth, wifi, etc.*

Aparte de estos proveedores de contenidos, puede crear sus propios. Cuando quiera acceder a los datos de un proveedor de contenidos deberá servir un objeto **ContentResolver** en el contexto de su aplicación. Este objeto trabaja como cliente de un objeto proveedor **ContentProvider,** que trabaja como servidor. El proveedor recibe los datos de los clientes, realiza la tarea y devuelve los resultados. Se debe crear un ContentProvider si se desea compartir datos de su aplicación con otros, pero para utilizar datos de otras aplicaciones simplemente se necesita un ContentResolver.

2.1.1. Accediendo al proveedor de contenidos

El ContentProvider presenta datos a las aplicaciones, como una o más tablas similares a las que encontramos en las bases de datos relacionales. Cada fila representa un elemento del proveedor de contenidos, y cada columna un dato concreto del mismo elemento de la fila.

La aplicación accede a los datos del proveedor a través del objeto cliente ContentResolver. Este objeto contiene métodos que llaman otros métodos (por cierto, con el mismo nombre) en la clase ContentProvider. Por ejemplo, para obtener una lista de contactos puede llamar al método ContentResolver.query(), y este llamará al método query() de ContentProvider. El método query() tiene una serie de argumentos que sirven para definir la consulta que se quiere hacer al proveedor. Estos argumentos tienen un paralelismo con las opcionesde una consulta a una base de datos:

- Uri uri: URI que representa la tabla de donde se obtendrán los datos.
- String [] projection: lista de las columnas que se devolverán por cadafila de la tabla.
- String selection: filtro que indica qué filas devolver con el mismo formato que la cláusula WHERE de SQL (sin la palabra "WHERE"). Si se pasa un *null* devolverá todas las filas de la URI.
- String [] selectionArgs: en el argumento anterior *(selection),* en lugar incluir el valor de las columnas, directamente puede indicar "=?" en la selección. Los "?" Serán sustituidos por los valores del *array* selectionArgs en el orden en que aparecen en selection.
- String sortOrder: establece como ordenan las filas, con el mismo formato que la cláusula de SQL ORDER BY (excluyendo las palabras "ORDER BY "). Si se pasa *null* se utilizará el orden predeterminado. Se pueden consultar los argumentos y su equivalente en un SELECT de SQL en la tabla.

Equivalencia de argumentos entre query () y una consulta SELECT deSQL :

Argumento de query ()	**Equivalente en** SELECT **de SQL**
Uri uri	FROM tabla
String [] projection	columna, columna, columna String
selection	WHERE col = valor
String [] selectionArgs	No existe un equivalenteString
sortOrder	ORDER BY col, col, ...

El método devuelve un objeto de la clase Cursor, posicionado antes de la primera entrada o *null* en caso de encontrar algún problema. El URI especifica los datos en el proveedor. Los URI incluyen el nombre del proveedor (llamado *authority, autoridad) y un nombre que apunta a una tabla* (path). El ContentProvider utiliza el *path* del URI para escoger la mesa a la que accederá (tiene un *path* para cada tabla). Laestructura de un URI es ésta:

```
1 <prefijo>://<authority>/<path>/<id>
```

donde:

- El *prefijo* para los proveedores de contenidos siempre es: content://.
- *authority* especifica el nombre del proveedor de contenidos. Para los estándares del sistema, por ejemplo: media, call_log, browser. Para proveedores de terceros, mejor introducir un nombre de dominio completo como: com.prueba.aplicacion.
- El *path* especifica la tabla dentro del proveedor.
- El *id* es un identificador único para una fila concreta de la tabla especificada.

Algunos ejemplos de proveedores de contenidos del sistema.

String	***Descripción***
content://media/internal/images	*Lista de las imágenes en la memoria del dispositivo*
content://media/external /images	*Lista de las imágenes en la memoria externa del dispositivo (por ejemplo, en la tarjeta SD)*
content://call_log/calls	*Lista de llamadas hechas con el dispositivo*
content://browser/bookmarks	*Lista de los marcadores del navegador web*

Por otra parte, muchos proveedores de contenidos tienen una constante con el URI de acceso a ellos mismos. Por ejemplo, en lugar de crear vosotros el URI para acceder el diccionario de palabras del dispositivo puede utilizar lo que él mismo tiene definido: UserDictionary.Words.CONTENT_URI. O para acceder a la lista de contactos, ContactsContract.Contacts.CONTENT_URI.

Para obtener datos de un proveedor, su aplicación necesita de un **permiso de lectura** por parte del proveedor. No se puede pedir este permiso durante la ejecución, así que para obtenerlo será necesario que utilice el elemento <uses-permission> en el su archivode *manifiesto* especificando el permiso que desea obtener delproveedor.

Al quedar definido en el *manifiesto, cuando el usuario instala esta aplicación le está* dando los permisos implícitamente. Para saber exactamente de qué permisos consta el proveedor, así como su nombre, consulte la documentación del proveedor. Por ejemplo, para pedir que su aplicación tenga permisos de lectura sobre el proveedor de contactos del dispositivo debe indicarlo en el archivo de *manifiesto* de la siguiente manera, antes de la etiqueta
<application>:

```
1<uses-permission android:name="android.permission.READ_CONTACTS">
2</uses-permission>
```

El siguiente paso es realizar una consulta al proveedor para obtener sus datos. El siguiente código muestra cómo hacer una consulta para obtener todos los contactos de la lista de contactos:

```
1  // Las columnas que queremos obtener para cada elemento.
2   String [] projection = new String [] {
3         ContactsContract.Contacts._ID,
4         ContactsContract.Contacts.DISPLAY_NAME,
5         ContactsContract.Contacts.HAS_PHONE_NUMBER
6  };
7
8   // Condición: queremos obtener todas las filas (por eso es null).
9   String where = null;
10  String [] whereArgs = null;
11
12  // Orden: que estén ordenados de forma ascendente.
13  String sortOrder = ContactsContract.Contacts.DISPLAY_NAME +
          "COLLATE localized ASC ";
14
15   Cursor c = getContentResolver().Query
16    (
17        ContactsContract.Contacts.CONTENT_URI,
18          projection, // Columnas para obtener de cada fila
19          where,// Criterio de selección
20          whereArgs, // Criterio de selección
21          sortOrder // Orden
22   );
```

Este código obtendrá toda la lista de contactos y devolverá un cursor en el resultado, la variable c. Cuando haya obtenido el cursor, se debe mirar su valor para saber si ha habido un error, o si el cursor tiene datos o no. Lo puede comprobar del siguiente modo:

```
1 // Si ha habido un error
2  if (c == null) {
3      // Código para tratar el error, escribir logs, etc.
4
5 }
6  // Si el cursor está vacío, el proveedor no encontró resultados.
7  elseif (c.getCount () <1) {
8        // El cursor está vacío, el contento provider no tiene elementos.
9       Toast.makeText (this, "No hay datos",
                Toast.LENGTH_SHORT).show ();
10} else {
11      // Datos obtenidos
12      Toast.makeText (this, "OK", Toast.LENGTH_SHORT) .show ();
13}
```

En estos momentos tiene un cursor a los datos obtenidos del proveedor, así que

podría recorrer los datos con el cursor y trabajar. Por ejemplo, el siguiente código muestra cómo obtener el identificador y el nombre, y como saber si el contacto tiene número de teléfono:

```
1 // Mientras tenemos un nuevo elemento en el cursor
2 while (c.moveToNext ()) {
3      // Obtener ID
4      String contactId = c.getString (c.getColumnIndex
            (ContactsContract.Contacts._ID));
5
6      // Obtener nombre
7      String nommbreContacto = c.getString (c.getColumnIndex
             (ContactsContract.Contacts.DISPLAY_NAME));
8
9       // Saber si tiene teléfono
10      String hasPhone = c.getString(c.getColumnIndex
              (ContactsContract.Contacts.HAS_PHONE_NUMBER));
11
12       // Mostrar
13       Toast.makeText (this, "id:" + contactId + "\ n" + "Nombre:"
               + nombreContacto + "\ n Tiene teléfono: "
               + hasPhone, Toast.LENGTH_SHORT) .show ();
14}
15
16 c.close ();
```

Para obtener una columna a partir del cursor debe utilizar el método Cursor.getString(int ColumnIndex), al que se le pasa el índice de la columna que desea obtener. Para obtener el número de columna a partir del identificador del campo que desea, utilice Cursor.getColumnIndex(String columnName). Todo esto lo puede hacer a la vez:

```
1 String nombreContacto = c.getString(c.getColumnIndex
      (ContactsContract.Contacts.//DISPLAY_NAME//));
```

El código anterior le muestra por pantalla la información de todos los contactos del proveedor tal como están en el cursor. Pero ¿qué ocurre si desea filtrar esta información? Trabajando con SQL puede hacerlo con una sentencia *WHERE columna =* valor.

Esto lo puede definir el método query () con los argumentos where y whereArgs. En

realidad, podría obtener fácilmente la misma sentencia WHERE modificando valor del argumento where. Para obtener el cursor únicamente los contactos que tengan teléfono, por ejemplo, puede definir el argumento como:

```
1 String where = ContactsContract.Contacts.HAS_PHONE_NUMBER + "= 1";
```

O, para obtener únicamente los contactos con nombre Juan:

```
1 String where = ContactsContract.Contacts.DISPLAY_NAME + "= 'Juan";
```

Por cuestiones de seguridad y para evitar que el usuario pueda introducir código SQL malicioso, se puede utilizar el carácter "?" en la variable where dentro de la sentencia.

Los caracteres "?" Serán sustituidos con los valores del *array* de *strings* whereArgs.

Por ejemplo, la sentencia anterior sería equivalente a:

```
1 String where = ContactsContract.Contacts.DISPLAY_NAME + "=?";
2 String [] whereArgs = {"Juan"};
```

SQL Injection

*El objetivo de whereArgs y de la sustitución de los "?" es impedir la inclusión de código SQL malicioso dentro de la sentencia . Imagínese que se crea un código para acceder a todos los contactos que tengan un nombre igual a una variable que introduce el usuario ("WHERE name= valor_variable"). Este usuario, en lugar de introducir el nombre de una persona, introduce: "nothing; DROP TABLE * ". La sentencia SQL resultante, en seres ejecutada: "WHERE name = nothing. DROP TABLE * "lograría borrar todas las tablasde la base de datos (si tuviera permiso para hacerlo).*

Hasta ahora se ha mostrado el nombre, el identificador y una variable que indica si el contacto tiene o no teléfono. Pero obtener el **teléfono** y el **correo electrónico** es un poco más complicado. Dado que en la lista un contacto puede tener varios números de teléfono y varias direcciones de correo electrónico, éstas no están almacenadas como variables estáticas sino como un proveedor de contenidos dentro del proveedor de contenidos de los contactos. Por lo tanto, para obtenerlas se debe hacer otra consulta y obtener un cursor en la lista de teléfonos y correos electrónicos, respectivamente. El siguiente código recorre la lista de teléfonos y correos y les guarda en una variable de tipo String (se podrían mostrar o hacer cualquier otra cosa con ellos). Al final, la variable se queda con el último teléfono y correo que será el que se muestre del contacto (aunque se podría mostrar el primero, o todos).

```
String telefono = null;
String email = null;

if (hasPhone.compareTo("1") == 0) {
      // Obtenemos los teléfonos
      Cursor telefonos = getContentResolver().query (
         ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
         null,
         ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "="
         + contactId,
         null,
         null);

       // Recorremos los teléfonos
     while (telefonos.moveToNext ()) {
          telefono = phones.getString(phones.getColumnIndex
           (ContactsContract.CommonDataKinds.Phone.NUMBER));
     }

     // Cerramos el cursor
     telefonos.close ();
}

// Obtener cursor correos
Cursor emails = getContentResolver().query
          (ContactsContract.CommonDataKinds. Email.CONTENT_URI,
          null, ContactsContract.CommonDataKinds.Email.CONTACT_ID+
          "=" + ContactId, null, null);

// Recorremos los correos
while (emails.moveToNext ()) {
     email = emails.getString emails.getColumnIndex
              (ContactsContract.CommonDataKinds.Email.DATA));
}
// Cerramos el cursor
emails.close ();

// Mostrar
Toast.makeText (this, "id:" + contactId + "\ n" + "Nombre:" +
        nomContacto + "\ n
        Teléfono: "+ telefono +"\n email: "+ email,
        Toast.LENGTH_SHORT).show ();
```

2.1.2. Insertando datos

La forma de insertar datos a un proveedor de contenidos es similar a la consulta. Para hacerlo, existe el método ContentResolver.insert(). Este método inserta una nueva fila al proveedor de contenidos y devuelve la URI de la fila creada. Los valores de la fila quedan definidos con un objeto de la clase ContentValues. Por ejemplo, el siguiente código sirve para insertar una nueva palabra al diccionario personal del usuario del dispositivo.

Para que funcione, se deben dar permisos de escritura el diccionario de datos del dispositivoinsertando la siguiente sentencia en el documento de manifiesto:

```
1 <uses-permission android:name =
              "android.permission.WRITE_USER_DICTIONARY">
2</uses-permission>
```

El siguiente fragmento de código muestra cómo se inserta un nuevovalor al proveedor de contenidos:

```
1 Uri UriNueva;
2
3 ContentValues Valores = new ContentValues();
4
5 // Creamos el valor de la nueva entrada
6 Valors.put (UserDictionary.Words.APP_ID, "com.android.ioc");
7 Valors.put (UserDictionary.Words.LOCALE, "es_ES ");
8 Valors.put (UserDictionary.Words.WORD, "Hospitalet");
9 Valors.put (UserDictionary.Words.FREQUENCY, "100");
10
11 // Insertamos
12 UriNueva = getContentResolver (). Insert (
13          UserDictionary.Words.CONTENT_URI,
14          Valores
15);
```

3. Modelo de los hilos

Cuando se pone en marcha una aplicación, el sistema Android pone enmarcha un nuevo proceso para la aplicación con un único hilo de ejecución.

Hilos de ejecución

Un hilo de ejecución es la unidad de procesamiento más pequeña que se puede programar en un sistema operativo. Pueden existir múltiples hilos dentro de un proceso y compartir recursos (como memoria, código y contexto), lo que no sucede entre diferentes procesos.

Cuando se utiliza un widget como WebView no es necesario un AsyncTask porque el WebView ya genera un hilo propio internamiento.

Por defecto, todos los componentes de la aplicación se ejecutan en este hilo de ejecución. Pero en algunos casos puede ser conveniente ejecutar diferentes tareas de la aplicación en hilos de ejecución independientes.

Cuando una aplicación accede a Internet a otros recursos externos, se produce un retraso de duración variable pero que puede llegar a ser de muchos segundos, o incluso de algunos minutos, desde que la aplicación pide los datos hasta que éstas llegan.

Si la aplicación funciona con un único hilo de ejecución, éste se puede quedar parado en la llamada a un método (por ejemplo, alguna que descargue datos de un servidor externo, el cual puede tardar en enviar la información). En este caso, la ejecución de la aplicación se quedará **bloqueada** hasta que este método devuelva los datos y así se continúeejecutando el resto del código.

3.1. AsyncTask

El mecanismo que permite que las aplicaciones puedan hacer dos o más operaciones al mismo tiempo son los hilos *(threads,* en inglés). Crear y sincronizar dos hilos es una tarea relativamente compleja, y por este motivo la librería de Android nos facilita la clase AsyncTask, que se hace cargo de la mayor parte del trabajo por nosotros.

Lo que hace el **AsyncTask** es crear otro hilo que puede utilizar para cualquier operación que prevéis que pueda tener una duración notable y que, por tanto, pueda bloquear la interfaz de usuario. Concretamente, cualquier operación de acceso a Internet debería hacerse en un hilo separado obligatoriamente, y la manera más sencilla es usar una AsyncTask.

Una tarea asíncrona se define como una tarea que se ejecuta en el hilo de *background* (hilo BG, por *background, es decir, que se ejecuta por detrás) y los resultados de la* que se muestran en el hilo de la interfaz de usuario (lo llamaremos hilo UI, de *User* Interface).

Para crear una AsyncTask debe crear una nueva clase que derive de AsyncTask, e implementar los métodos descritos a continuación. Lo más importante es saber que algunos métodos se ejecutan al hilo UI y otros al hilo que hace la operación en el background.

Esto es fundamental porque sólo se pueden modificar las vistas de la aplicación (escribir valores, mostrar mensajes, hacer avanzar barras de progreso o mostrar otro tipo de contenidos) desde el hilo UI.

Un AsyncTask se define utilizando tres tipos genéricos:

- **Params:** el tipo de los parámetros enviados a la tarea durantela ejecución.
- **Progress:** el tipo de unidades de progreso publicadas durante lacomputación en el background. Esto sirve por si desea mostrar un progreso de la tarea realizada mientras ésta se ejecuta en el background. Por ejemplo, podría animar una barra para mostrar el progreso de la carga de un fichero.
- **Result:** el tipo del resultado de la computación enla background.

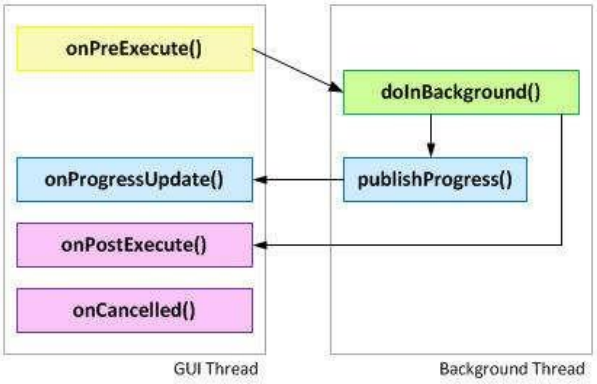
No siempre se utilizan todos los tipos en una tarea asíncrona. Para marcar un tipo que no se utilizará, simplemente utilice el tipo void. Porejemplo, la siguiente clase asíncrona define que tendrá un argumento de tipo String y devolverá un Bitmap. No utilizará ningún tipo de progreso de la tarea:

```
1 private class CargaImagen extends AsyncTask
            <String, Void, Bitmap> {
2        ...
3 }
```

Cuando se ejecuta una tarea de forma asíncrona, ésta realiza cuatro pasos:

- **onPreExecute ()** (UI): ejecutado al UI inmediatamente despuésde que la tarea es ejecutada. Permite inicializar los elementos de la interfaz que sean necesarios para la tarea, tales como crear una barra de progreso.
- **doInBackground (Params ...)** (BG): es llamada después de que onPreExecute () acaba de ejecutarse. Pone en marcha la operación que desea ejecutar en otro hilo en el background. Este método recibe los parámetros necesarios para realizar la operación asíncrona. El resultado de la ejecución se debe devolver y se pasará de vuelta en el último paso. Dentro de doInBackground () se puede llamar publishProgress (Progress...) por publicar una o más unidades de resultado. Estos valores se publican en el hilo UI, a onProgressUpdate (Progres ...).

- **onProgressUpdate (Progress ...)** (UI): es llamado en el hilo UI después de haber llamado publishProgress (Progress ...). Le permite actualizar algún elemento de la interfaz para mostrar el progreso de la operación mientras la operación en el *background* todavía está en ejecución, tales como actualizar una animación o hacer avanzar una barra de progreso.
- **onPostExecute (Result)** (UI): este método es llamado cuando la ejecución en el *background* finaliza. Recibe el resultado de la operación que ha ejecutado el AsyncTask y como se ejecuta en el hilo UI permite visualizar los resultados obtenidos (como una imagen, una página web o una lista de tweets).



Para utilizar AsyncTask necesita crear su clase, que herede de AsyncTask e implemente el método de *callback* doInBackground (), que se ejecuta en un hilo en el background. Para actualizar su interfaz de usuario (o UI, *user interface) debe implementar el método onPostExecute () que incorpora los* resultados de haber ejecutado doInBackground () y se ejecuta en el hilo UI, y por tanto puede actualizar el estado de su interfaz sin problemas. Cuando tenga la clase creada, para lanzar el hilo de la AsyncTask debe elaborar un nuevo objeto de la subclase que ha creado y llamar al método execute (), pasándole la información necesaria para que pueda realizar la operación.

Por ejemplo, la siguiente clase descarga una imagen de la red, mediante el método propio Bitmap DescargaImagenDeLaRed (String url), y usa el Bitmap para mostrarlo en un widget:

```
private class CargaImagen extends AsyncTask<String, Void, Bitmap> {
    @Override
    protected Bitmap doInBackground(String... param) {
        Bitmap bitmap = null;
        try {
            // Descargamos la imagen desde una URL
            InputStream input = new java.net.URL(param[0]).openStream();
            // Decodificamos el  Bitmap
            bitmap = BitmapFactory.decodeStream(input);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return bitmap;
    }

    protected void onPostExecute(Bitmap result) {
        // Recibimos el resultado de doInBackground (el Bitmap)
        // y lo usamos para cargar la imagen a la UI
        ImageView iv = (ImageView) findViewById(R.id.miImagen);
        iv.setImageBitmap(result);
    }
}
```

3.2. Thread

Podemos utilizar los hilos como en Java, lo que debemos tener en cuenta que desde los hilos que creemos no podemos acceder a la interfaz de usuario, para solucionar esto, Android proporciona varias alternativas, entre ellas la utilización del método post() para actuar sobre cada control de la interfaz, o la llamada al método runOnUiThread() para “enviar” operaciones al hilo principal desde el hilo secundario:

```
new Thread(new Runnable() {
    public void run() {
        pbarProgreso.post(new Runnable() {
        public void run() {
            pbarProgreso.setProgress(0);
        }
    });

    for(int i=1; i<=10; i++) {
        tareaLarga();
        pbarProgreso.post(new Runnable() {
            public void run() {
                pbarProgreso.incrementProgressBy(10);
                }
            });
    }

    runOnUiThread(new Runnable() {
        public void run() {
            Toast.makeText(MainHilos.this, "Tarea finalizada!",
                Toast.LENGTH_SHORT).show();
        }
    });
    }
}).start();
```

3.3. Handler

Un Handle permite enviar y procesar mensajes y objetos ejecutables asociados con el MessageQueue de un hilo. Cada instancia del Handlerestá asociada con un único hilo y la cola de mensajes de ese hilo. Cuando crea un nuevo Handler, está vinculado a la fila / cola de mensajes del subproceso que lo está creando. A partir de ese momento, entregará los mensajes y los ejecutables a esa cola de mensajes y los ejecutará a medida que salgan de la cola de mensajes.

Hay dos usos principales para un controlador: programar mensajes y runnables para que se ejecuten como un punto en el futuro; y para poner en cola una acción que se realizará en un hilo diferente al suyo.Por ejemplo, para ejecutar un código:

```
Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            public void run() {
                //TODO
            }
        }, milisegundos);
```

4. Conexiones HTTP

El protocolo HTTP *(HyperText Transfer* Protocol, protocolo de transferencia de hipertext) es un protocolo estándar de Internet que sirve para la transferencia de hipertexto. Es el protocolo base que está fundamentada la WWW *(World Wide* Web, 'red de extensión mundial'). Utilizando HTTP a su aplicación se podrán realizar diversas tareas, como descargar información de texto y binaria (como páginas web o imágenes, respectivamente).

Un *hipertexto* es un texto mostrado en un ordenador u otro dispositivo electrónico con enlaces *(hyperlinks)* a otros textos, que son de acceso inmediato por el lector a través de un clic del ratón o de la pantalla táctil. Aparte de texto, puede contener tablas, imágenes y otros formatos de representación de contenidos multimedia.

También se puede utilizar el protocolo HTTPS *(HTTP Secure, HTTP seguro) para* comunicaciones seguras a través de una red de ordenadores. HTTPS proporciona autentificación del sitio web y encriptación bidireccional en las comunicaciones entre el cliente y el servidor.

Android incluye dos clientes HTTP que puede utilizar a su aplicación: HttpURLConnection y el cliente de Apache HttpClient. Ambos soportan HTTPS, flujos de subida y descarga y IPv6.

Una conexión **HttpURLConnection** permite enviar y recibir datos por la web. Los datos pueden ser de cualquier tipo y tamaño, incluso puede enviar y recibir datos de las que no conoce el tamaño previamente.

Para usar esta clase debe seguir los siguientes pasos:

- Obtenga un objeto nuevo HttpURLConnection llamando el método URL.openConnection() y forzando su resultado en HttpURLConnection.
- Prepare la petición. La principal propiedad de la petición es su URI. Las cabeceras de la petición pueden tener otros metadatos como credenciales, tipo de contenido, cookies de sesión, etc.
- Si va a subir datos a un servidor, la instancia debe estar configurada con setDoOutput(true). Debe transmitir datos escribiendo al flujo de datos devuelto por getOutputStream().
- Lea la respuesta. Las cabeceras de la respuesta generalmente incluyen metadatos como el tipo de datos de la respuesta y su tamaño, la fecha de modificación o cookies de sesión, entre otros. El cuerpo de la respuesta se puede leer del flujo devuelto por getInputStream (). Si la respuesta no tiene cuerpo *(Body* eninglés), el método devuelve un flujo vacío.
- Desconecte. Cuando haya leído la respuesta, debe cerrar el HttpURLConnection con disconnect ().

Por ejemplo, con el siguiente código estaríais leyendo el contenido de la web del diario el pais:

```
1 URL url = new URL ("http://www.elpais.es/");
2 HttpURLConnection urlConnection=
          (HttpURLConnection) url.openConnection ();
3  try {
4         InputStream in = new buffered InputStream
               (urlConnection.getInputStream ());
5         trabajarConElFlujoDeDatos (in);
6  finally {
7          urlConnection.disconnect ();
8 }
9}
```