# DESARROLLO DE SERVICIOS TELEMÁTICOS

## Tema 4

3er. curso

Graduado en Informática

Mención de Tecnologías de la Información

Autora: Lidia Fuentes

# Computación distribuida en pequeños dispositivos

- Lenguajes y plataformas de programación
- Desarrollo de servicios web en dispositivos móviles
- Programación en redes de corto alcance

# Contenido

- Introducción al Desarrollo de Servicios en Dispositivos Móviles
- Desarrollo de Servicios sobre Java ME
- Desarrollo de Servicios sobre Android
- Programación en redes de medio y corto alcance

# 4.1. Introducción al desarrollo de servicios en dispositivos móviles

- Objetivos de aprendizaje
  - Conocer las diferentes plataformas de programación para aplicaciones móviles
  - Conocer la arquitectura de algunas plataformas de programación
  - Desarrollar aplicaciones para móviles capaces de conectarse vía sockets
  - Desarrollar aplicaciones para móviles capaces de conectarse vía conexiones HTTP
  - Uso de redes de medio y corto alcance en servicios para móviles

# Plataformas de desarrollo de servicios en móviles

- Dispositivos móviles
  - Recursos limitados
    - Capacidad de procesamiento, memoria, etc.
  - Conectividad
    - Telefonía
      - GSM (operador de telefonía)
    - Datos
      - Red WiFi (permanente o intermitente)
      - Interfaces de corto alcance (ej: Bluetooth, NFDirect, WiFiDirect, etc.)
  - Localización
    - Localización geográfica (ej: GPS)
  - Sensores
    - Sensores de orientación
    - acelerómetros
  - Cámara integrada



# Limitaciones de los dispositivos móviles

- Procesadores con capacidad de cómputo limitada
- La memoria es muy limitada comparado con un ordenador convencional
- El almacenamiento de datos es limitado (poca memoria interna, tarjetas SD o similar)
- Batería

# Programación para Móviles

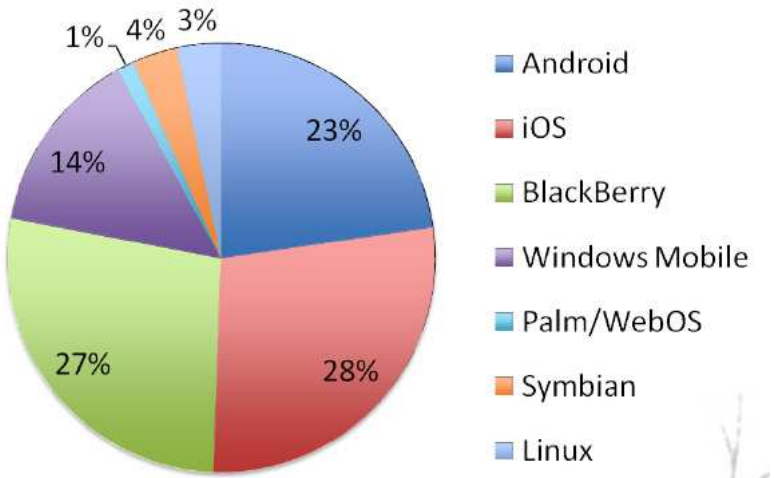
- Java Mobile Edition (Java ME)
  - Java, usado en múltiples plataformas (ej: Symbian/Nokia)
- iOS/Apple
  - Lenguaje Objective-C, IDE Xcode
- BlackBerry OS/BlackBerry
  - Java ME, BlackBerry Java Development Environment
- Windows Phone 8/Windows
  - C#, VisualBasic sobre .NET
- Android/Google
  - Java for Android

**¿Qué plataforma de programación es mejor aprender?**

Depende de la evolución del mercado de telefonía móvil

# Elección de plataforma

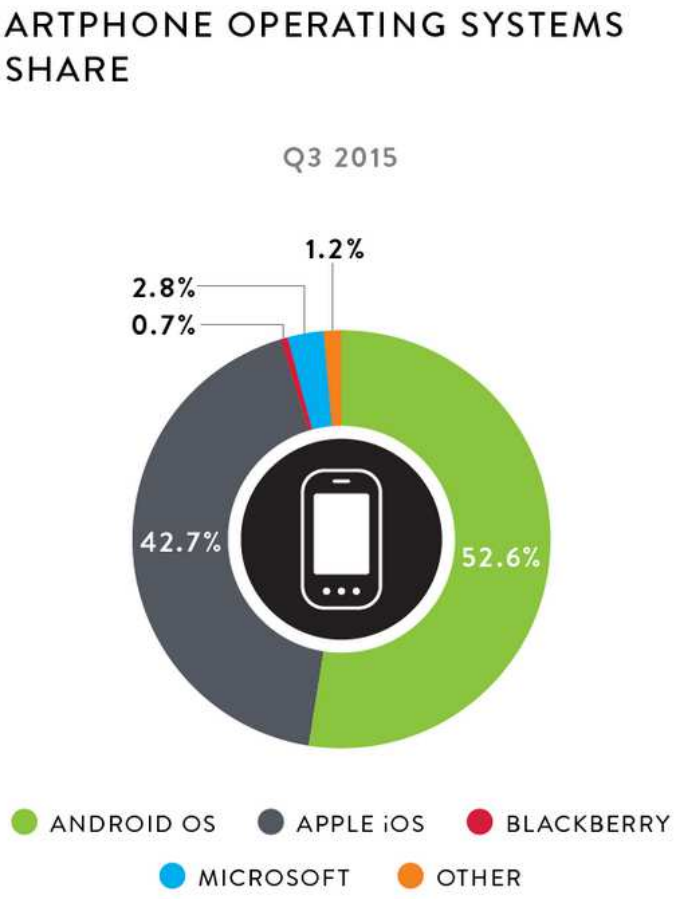
- Java ME (2006)
  - Fue la más usada hasta 2011
  - Multiplataforma (móviles Nokia y BlackBerry fundamentalmente y sensores)
  - Otros dispositivos de la IoT
- Android (2008)
  - Crece su demanda
  - BlackBerry y otros fabricantes de móviles lo están incorporando
  - Mucha documentación
- Ambas son de código libre




Según consultora Nielsen

# Elección de plataforma

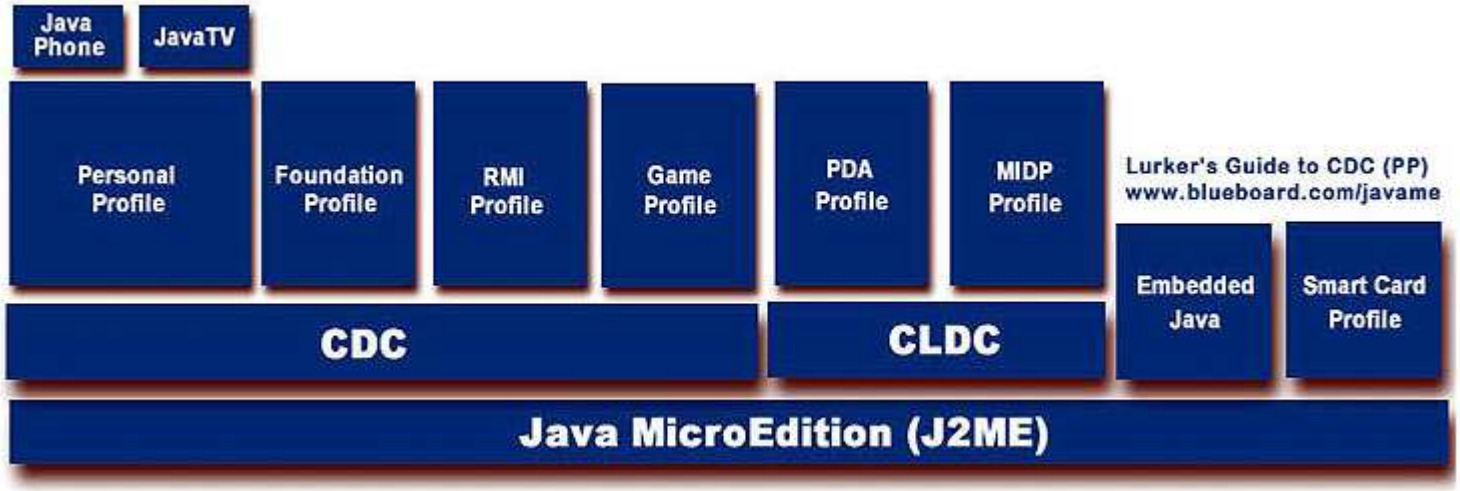
- Java ME
  - Casi no se usa en telefonía móvil
  - Se usa en dispositivos de la IoT/IoE
    - Qualcomm (ej: sensores, ordenador de a bordo, AAL, etc.)
    - Raspberry Pi (placa base para aprendizaje de Tec. Informática)
- Android (2008)
  - Se ha consolidado en el mercado de los móviles
  - Intentan consolidar su uso en otros dispositivos
    - TV, relojes, etc.
- No sólo los smartphones marcan la tendencia sino la IoT




Según consultora Nielsen

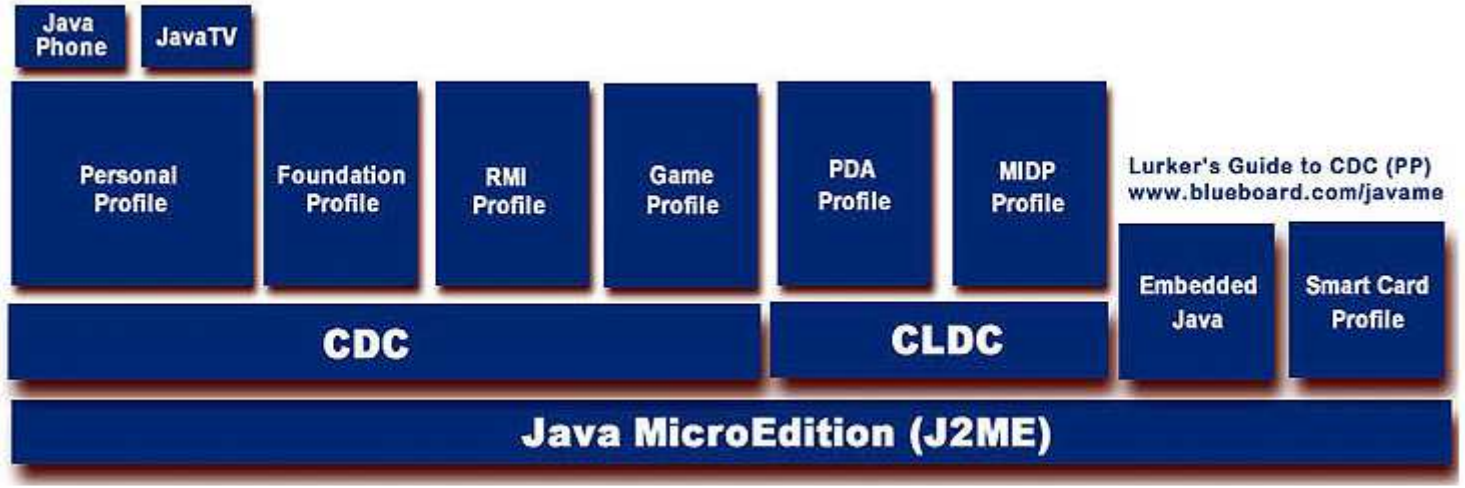
# 4.2. Desarrollo de Servicios sobre Java ME

- Java ME
  - Tiene una estructura modular para adaptarse a los diferentes dispositivos (ej: Móviles de diferentes fabricantes, sensores SunSPOT, etc.)



# Arquitectura Java ME

- Configuraciones CDC (*Connected Device Configuration*) y CLDC (*Connected Limited Device Configuration*)
  - Definen el entorno mínimo necesario para la ejecución de aplicaciones java en un grupo amplio de dispositivos móviles (JVM/KVM+paquetes básicos)
- Perfiles (profiles)
  - Definen paquetes adicionales para soportar funcionalidades básicas imprescindibles de cada familia de dispositivos
- Paquetes opcionales
  - Los paquetes opcionales sirven para cubrir funcionalidades específicas: Bluetooth, soporte 3D, etc.



# Configuración CDC

- CDC (*Connected Device Configuration*) se usa en smartphones avanzados o tabletas
  - JVM y 13 paquetes de Java SE
- Admite los perfiles:
  - Foundation Profile
    - proporciona soporte básico de red y E/S
  - Personal Basis Profile
    - contiene todas las clases del Foundation Profile más soporte limitado de Javabeans y AWT
  - Personal Profile
    - incluye los dos perfiles anteriores más soporte completo de AWT y Javabeans

# Configuración CLDC

- CLDC (*Connected Limited Device Configuration*) para dispositivos con recursos limitados (smartphones normales)
  - Máquina virtual reducida KVM y 5 paquetes Java (versiones reducidas)
  - MIDP era el perfil más común de CLDC
    - Interfaz de usuario sencilla, adaptada a móviles
    - API especializado para juegos
    - Persistencia básica
    - Conexión por red
    - Sonido
  - MEEP (ME Embedded Profile)
    - Específico para dispositivos de la IoE (Internet of Everything)

# Paquetes opcionales

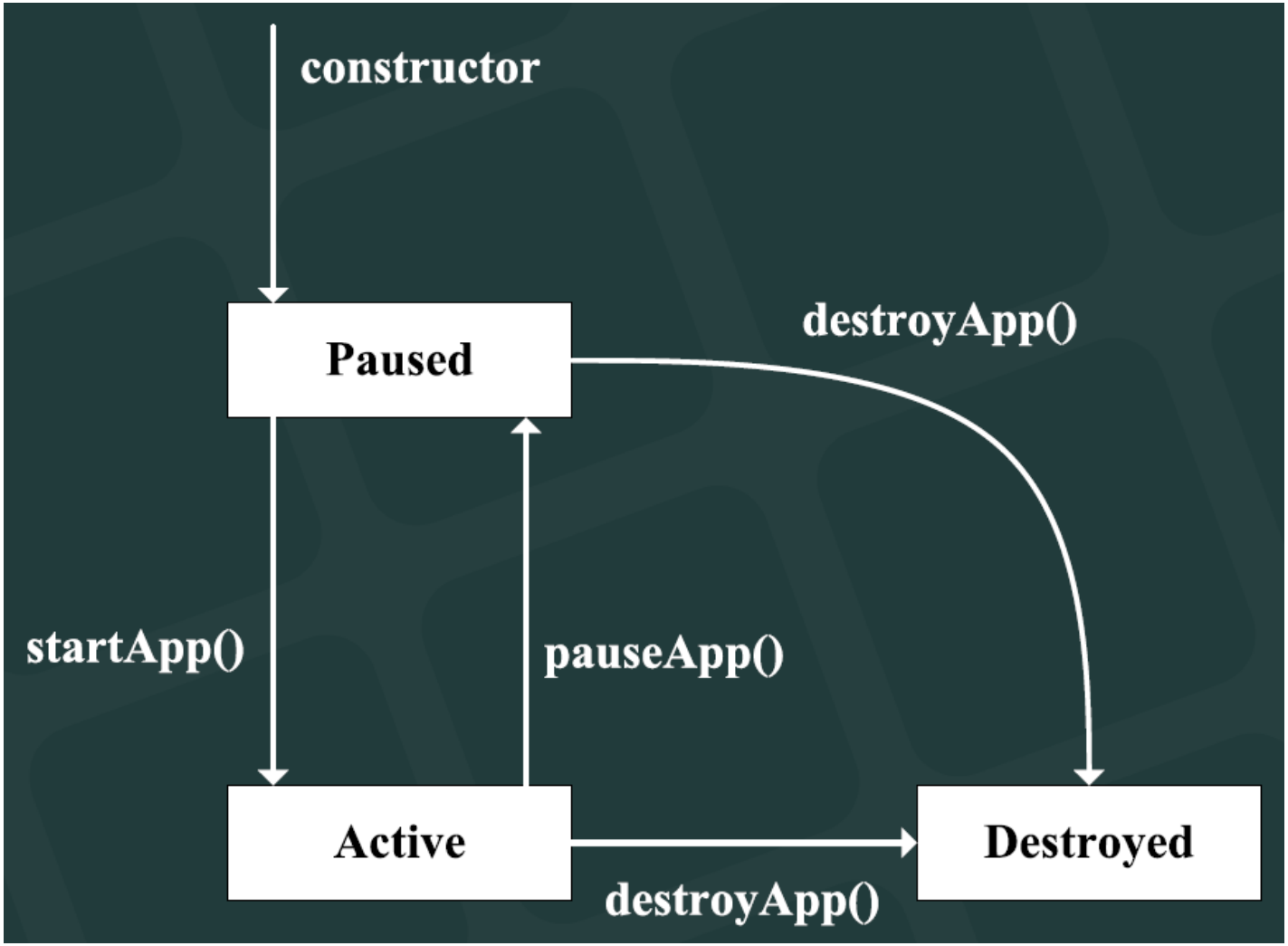
- Permiten extender la funcionalidad básica de los perfiles para aprovechar todas las posibilidades del dispositivo
- Mobile Media API (MMAPI) JSR135.
  - Soporte de audio y video, tanto reproducción como captura
- Location API JSR179.
  - Localización geográfica del dispositivo, mediante GPS u otros mecanismos
- Java ME Web Services API (WSA) JSR172.
  - Soporte de servicios web en dispositivos móviles
- Bluetooth API JSR 82
- Java ME RMI JSR 66.
  - Llamada a métodos de objetos remotos

# Programación MIDP

- Una aplicación MIDP requiere la implementación de un MIDlet, cuya estructura recuerda los Applets y Servlets de Java SE

```
import javax.microedition.midlet.MIDlet;

public class EjemploMidlet extends MIDlet {
        public void startApp() {
        // Arrancar servicio
        }
        public void pauseApp() {
        // Parar servicio
        }
        public void destroyApp(boolean unconditional) {
        // Eliminar recursos
        }
}
```

# Ciclo de vida de un MIDlet



# MIDP: sockets TCP y HTTP

- MIDP es especialmente potente en lo que se refiere a la conexión por red mediante sockets, http y otros protocolos
- La clase *Connection* representa una conexión genérica y es extendida a tres conexiones que admiten E/S mediante streams:
  - InputConnection,
  - OutputConnection
  - StreamConnection
    - CommConnection, HttpConnection, HttpsConnection, SocketConnection, etc.

# MIDP: sockets TCP y HTTP

- La clase *Connector* es una factoría que a partir de una url devuelve la clase que modela la conexión correspondiente:
  - `Connection Connector.open(String url)`

```
HttpConnection con1;
con1 = (HttpConnection) Connector.open("http://www.google.es/");
SocketConnection con2;
con2 = (SocketConnection) Connector.open("socket://servidorTCP:6667");
```

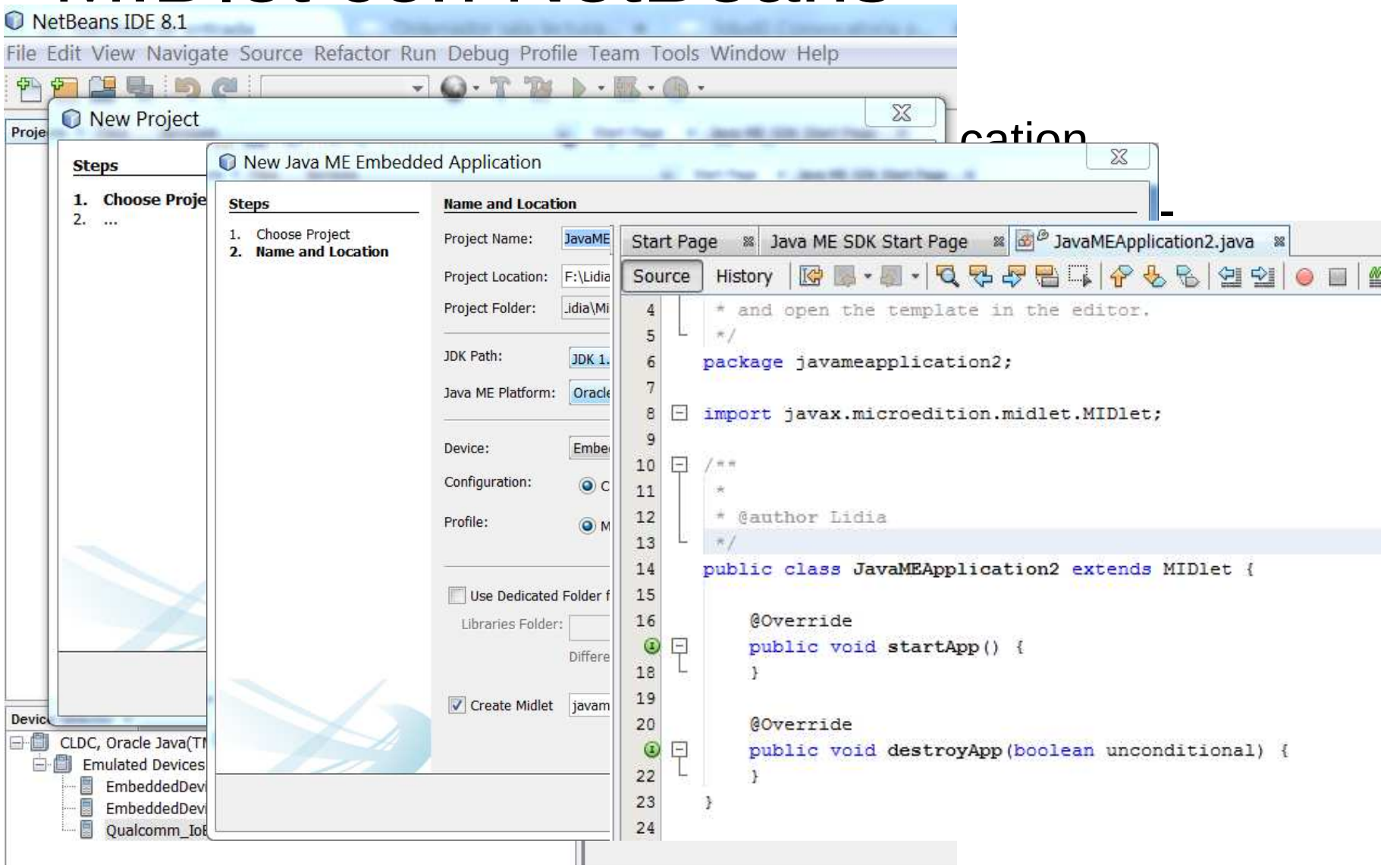
- Se obtiene streams de lectura/escritura
- La conexión se cierra con close()

```
// idem para SocketConnection
InputStream in = con1.openInputStream();
PrintStream out = new PrintStream(con1.openOutputStream());
// leer y escribir de ambos streams
con1.close();
con2.close();
```

# MIDlet con NetBeans

- Instalación del Java ME SDK
- Instalación del plugin Java ME SDK for NetBeans
- Crear la clase principal del MIDlet
- Modificar la clase Midlet creada por defecto

# MIDlet con NetBeans



# 4.3. Desarrollo de Servicios sobre Android

- Sistema operativo y entorno de desarrollo integrado
- Versiones


# Android SDK

- Componentes Android (sdk Android)
  - Framework de aplicaciones
    - API basada en Java SE (no todas las clases Java SE se pueden usar en un programa Android)
  - Entorno de desarrollo completo
    - Emulador de dispositivo
    - Plug-in para el IDE Eclipse o el IDE Android Studio
    - Herramientas para depurar, gestión de memoria y perfiles de rendimiento
  - Máquina virtual Dalvik (JVM optimizada para dispositivos móviles)
  - Navegador Web integrado
  - Gráficos optimizados
  - SQLite para almacenamiento de datos
  - Soporte multimedia
  - GPS, Telefonía GSM, Bluetooth, WiFi, …

# Instalación Android SDK

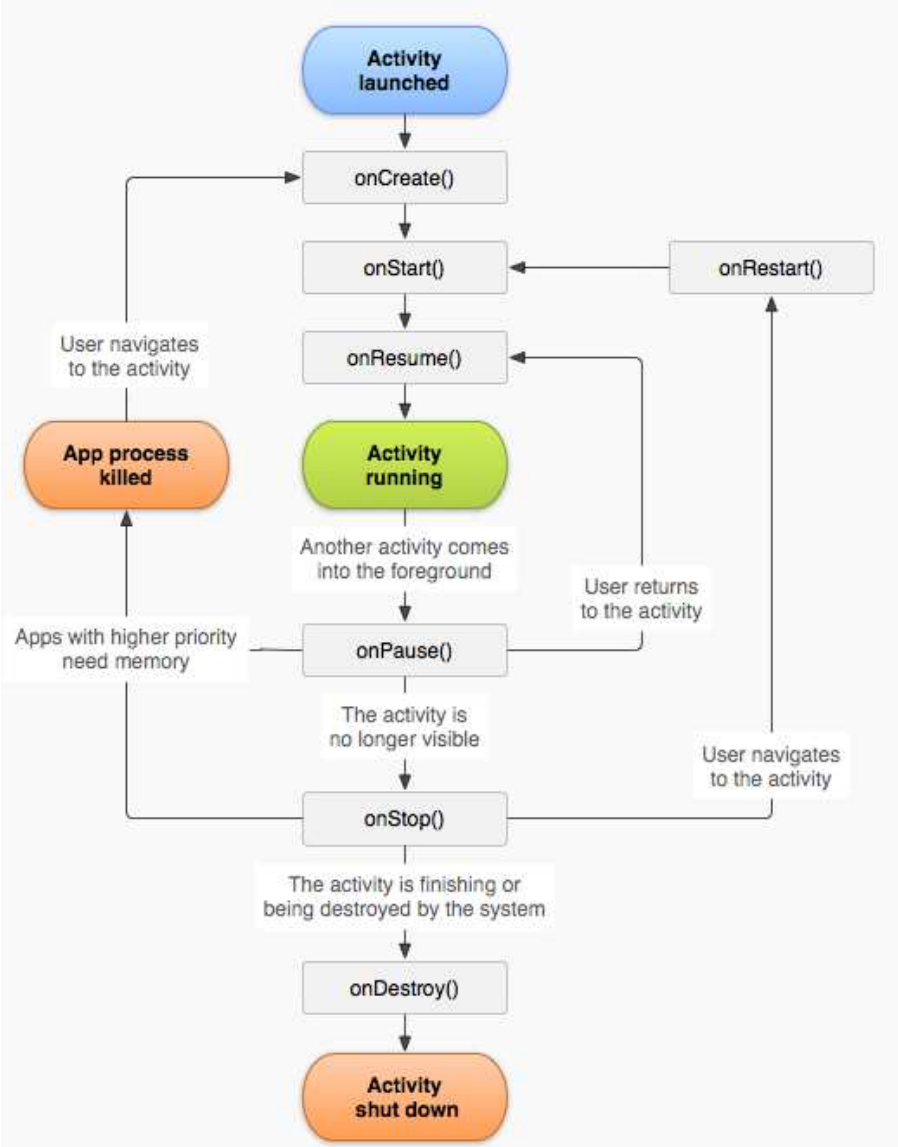
- Instalación del SDK de Android desde Eclipse y el emulador de dispositivos
  - Java JDK
  - Eclipse IDE
  - Android SDK
  - ADT Plug-in para Eclipse
  - Emulador Android (AVD, Android Virtual Device)
    - Permite ejecutar y verificar las aplicaciones Android sin utilizar un dispositivo físico
    - Dispone de herramientas de depuración (ej: Log, consola de mensajes)
    - Es posible simular la interrupción de la aplicación con llamadas, SMSs, …
- Android Studio
- Web para el desarrollo de aplicaciones Android en:

  http:// developer.android.com

# Modelo de desarrollo Android

- Componentes de una aplicación Android:
  - **Activity**: Es el elemento principal de la interfaz de usuario. Análogo al elemento window o dialog en una aplicación de escritorio. Debería crearse al menos una actividad en una aplicación Android
  - **Content Providers**: Permiten almacenar datos en el dispositivo y hacerlos accesibles para otras aplicaciones.
  - **Services**: Se ejecutan permanentemente, en segundo plano. Útil para operaciones con gran carga de computación o acciones que no bloqueen la interacción con el usuario
  - **Broadcast Receiver**: Reciben y reaccionan ante la difusión de eventos generales del sistema.
- Una aplicación Android está formada por uno o más componentes de cada uno de los cuatro tipos de componentes
- Cada componente es un punto distinto de entrada mediante el cual el sistema puede invocar la aplicación
  - Cualquier aplicación puede comenzar un componente de otra aplicación

# Ciclo de vida de una actividad




Ver http:// developer.android.com

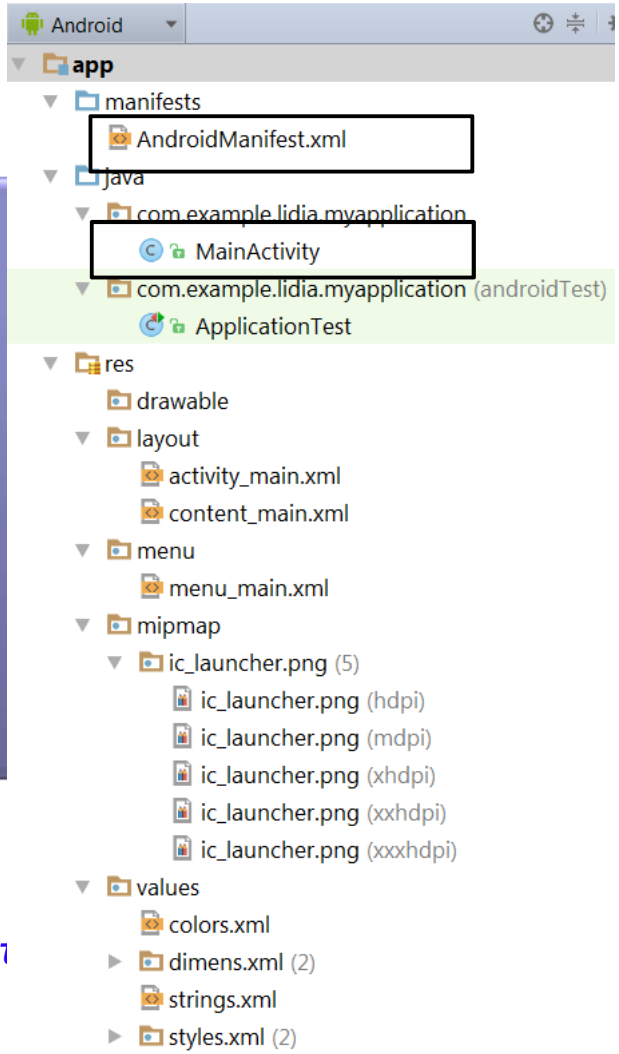
```
package com.example.clientehttp;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
  // Inicializar el menu
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }
…
}
```

# Activación de componentes

- Cada aplicación no tiene un único punto de inicio de ejecución, sino que hay que indicarle al sistema qué componente debe activar
  - El sistema ejecuta cada aplicación en un proceso separado con restricciones de permisos
- Para realizar la activación es necesario especificar el archivo `AndroidManifest.xml`
  - Se indica el paquete Java de la aplicación (es su identificador)
  - Describe los componentes de la aplicación y sus condiciones para ser ejecutados (ej: qué actividad es la principal )
  - Declara los permisos para acceder a ciertas APIs (ej: acceso a la red, a bluetooth, etc.)
  - Se indica la versión mínima de la API de Android requerida
  - Lista las bibliotecas de clases adicionales requeridas

# Proyecto Android Studio

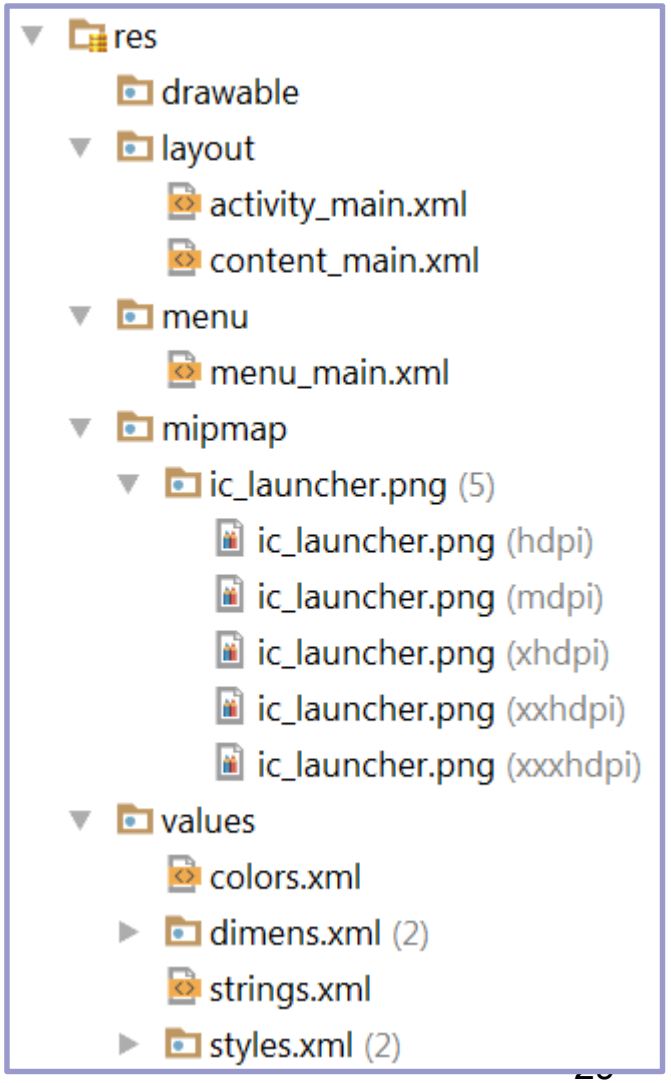
- Abrir nuevo proyecto
- Crear la actividad principal
  - Hay varias plantillas de actividad
    (ej: *BlankActivity*, *LoginActivity*, etc.)
  Ej: *MainActivity.java*
- La actividad principal se llama desde el fichero *AndroidManifest.xml*




```
…
<activity
    android:name="com.example.primerejemplo.MainActivi
    android:label="@string/app_name" >
   <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
…
```

# Proyecto Android Studio

- Componentes del proyecto creado
  - **java** Código fuente de la aplicación organizado en paquetes
  - **manifests** contiene el fichero **AndroidManifest.xml** con la descripción de la aplicación y valores de configuración
  - **res** Directorio que almacena los recursos utilizados por la interfaz gráfica
    - **layout** Interfaz de usuario
    - **menu** Menús de la aplicación
    - **values** Definición de estilos, cadenas de texto, etc.
    - **drawable** Imágenes
    - **mipmap** Iconos



# Estructura Fichero **AndroidManifest.xml**

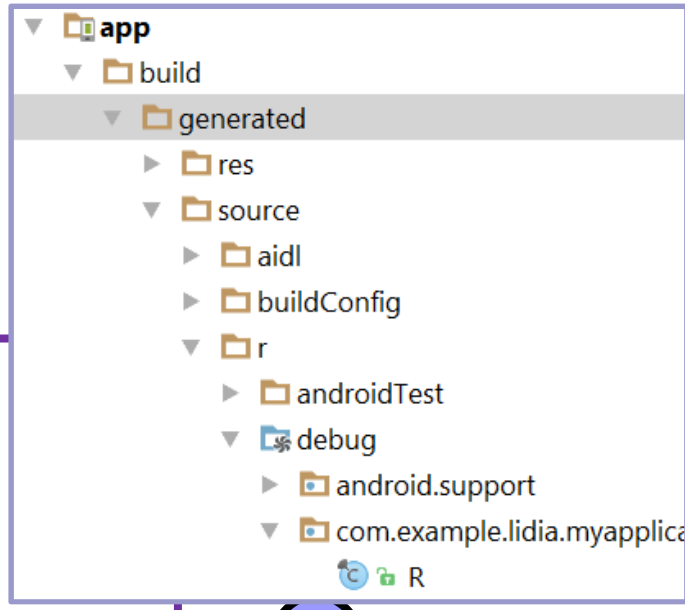
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.primerejemplo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.primerejemplo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```
<!- res.values.strings.xml →
<resources>
    <string name="app_name">PrimerEjemplo</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
</resources>
```

```
<!- res.values.styles.xml →
<resources>
  <style name="AppTheme" parent="AppBaseTheme">
  </style>
</resources>
```

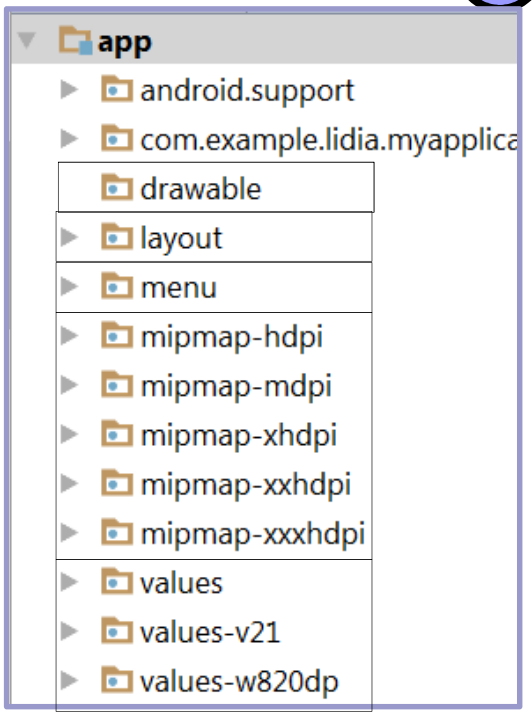
Condiciones para mostrar una actividad

```java
// PROYECTO PrimerEjemplo
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main,
                    menu);
    return true;
  }
}
```




1

2




```
public final class R {
    public static final class mipmap {
        public static final int ic_launcher=0x7f020;…
    }
    public static final class layout {
        public static final int activity_main=0x7f030;…
    }
    public static final class menu {
        public static final int activity_main=0x7f060;…
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello_world=0x7f04001;
        public static final int menu_settings=0x7f042;…
    }
…
```

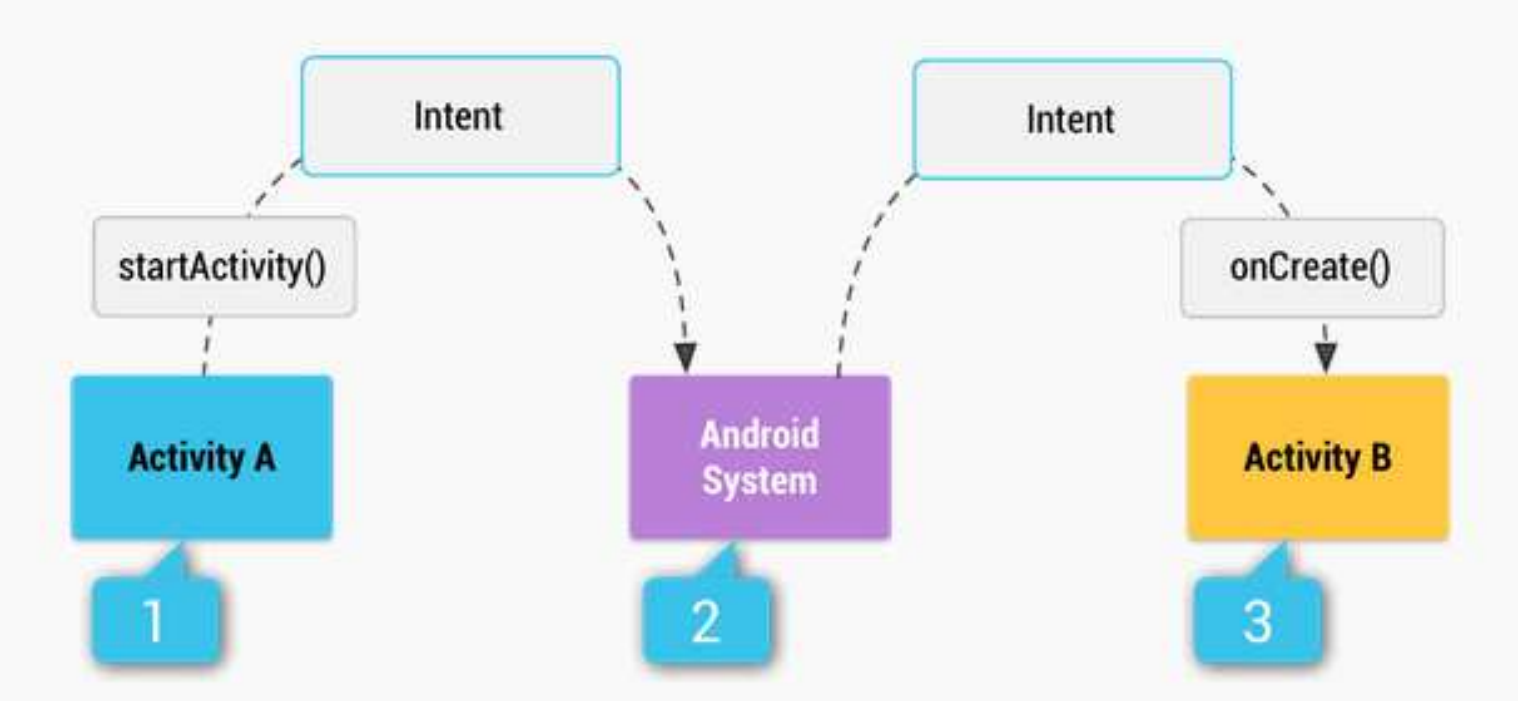
# Tipos de identificadores a recursos

- Al generar la clase R.java se transforma el identificador de cada recurso a un valor único. De esta forma se pueden referenciar recursos externos (ej: layout, menú, iconos, etc.) desde el código Java
- La siguiente tabla muestra la correspondencia entre identificadores de recursos

| Recurso | Referencia Java | Referencia XML |
|---|---|---|
| `res/layout/activity_main.xml` | `R.layout.activity_main` | `@layout/activity_main` |
| `res/drawable-mdpi/ic_launcher.png` | `R.drawable.ic_launcher` | `@drawable/icon` |
| `< String hello = “Hello” >` | `R.string.hello` | `@string/hello` |

# Relación Activities e Intents

- Una actividad puede invocar a otra actividad mediante **Intents**
- **Intents**: Descripción de una acción de una actividad para que pueda ejecutarse desde otra actividad (ej: visualizar una página Web)






Ver http:// developer.android.com

# Relación Activities e Intents

- Una actividad puede invocar a otra actividad mediante **Intents**
- **Intents**: Descripción de una acción de una actividad para que pueda ejecutarse desde otra actividad (ej: visualizar una página Web)
- Elementos de un **Intent**
  - **Action** (acción Android o de usuario)
  - **Category** (indica dónde se mostrará tu actividad, ej: en el LAUNCHER menú, etc.)
  - **Data** (datos para realizar la acción en formato URI)
  - **Type** (tipo MIME indicando el tipo de recurso en el que la actividad va a operar)
  - **Component** (la clase de actividad que debe recibir el *intent*, es opcional y alternativo al resto de campos del *intent*)
  - **Extras** (información adicional)

# Ejecutar actividades

- Inicio de actividad

```
startActivity(Intent)
```

- Inicio de actividad con obtención de resultado

```
// Parametro adicional: id llamada
startActivityForResult(Intent, int)
```

- Obtener el resultado de la ejecución de una actividad

```
// Codigo de resultado, id llamada
onActivityResult(int, int, Intent)
```

- La actividad llamada pasará el resultado con:

```
setResult(int)
```

# Acciones de comunicaciones

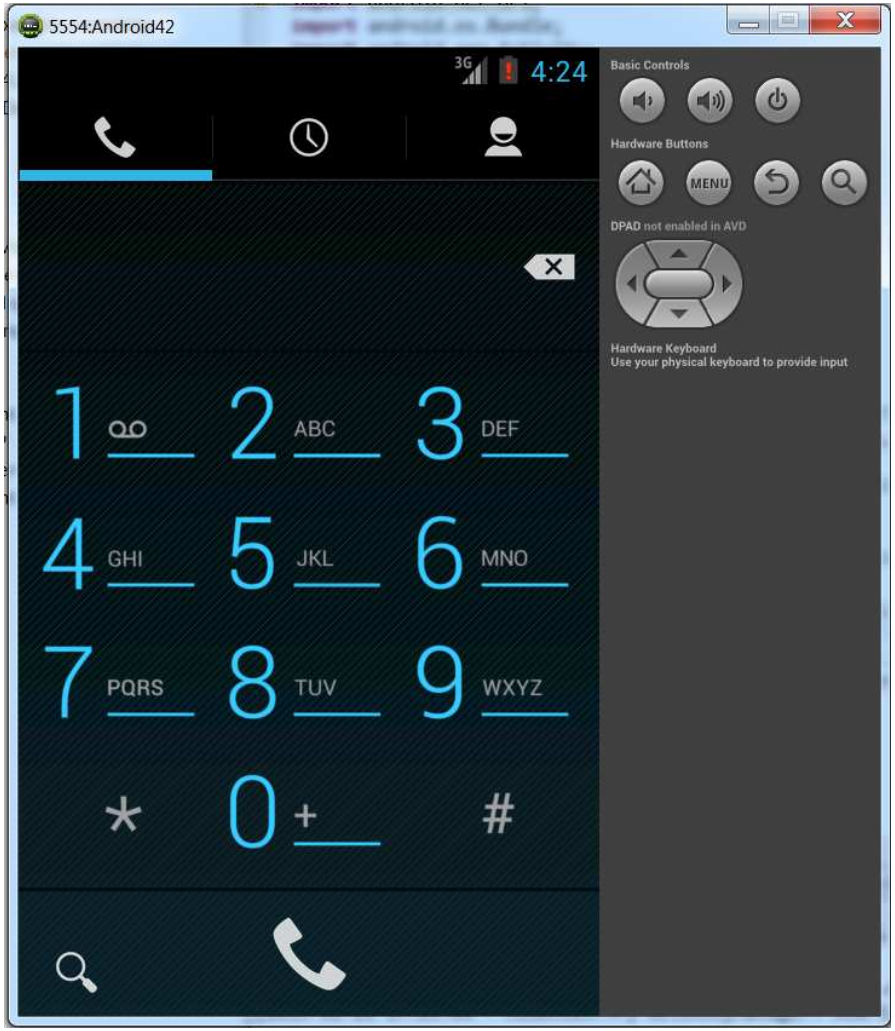
- Cargar una página web (ACTION_VIEW)
- Hacer una llamada telefónica (ACTION_DIAL, ACTION_CALL)
- Hacer una búsqueda en la Web (Google por defecto) ACTION_WEB_SEARCH
- Enviar un SMS (ACTION_SENDTO)
- Enviar correo electrónico (ACTION_SEND)

# Action ACTION_VIEW

```
String link="http://www.lcc.uma.es";
Intent intent = new Intent(Intent.ACTION_VIEW,Uri.parse(link));
startActivity(intent);
```

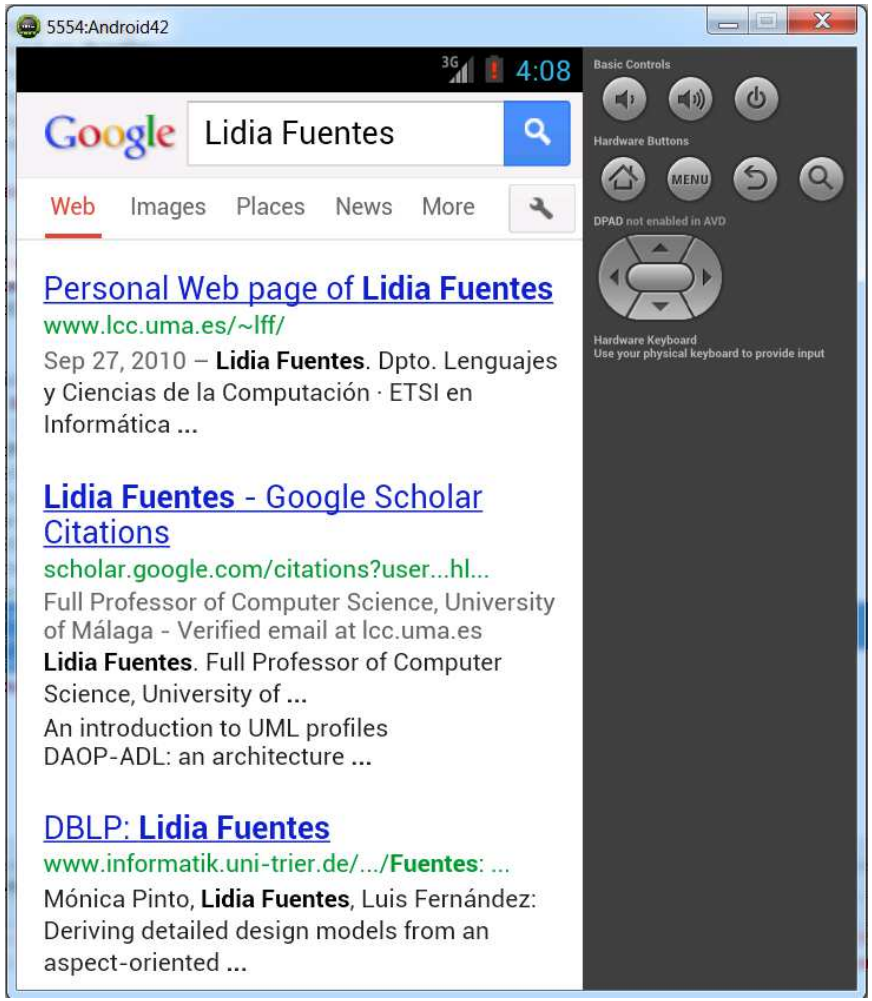
# Action ACTION_DIAL

```
Intent intent = new Intent(Intent.ACTION_DIAL);
startActivity(intent);
```



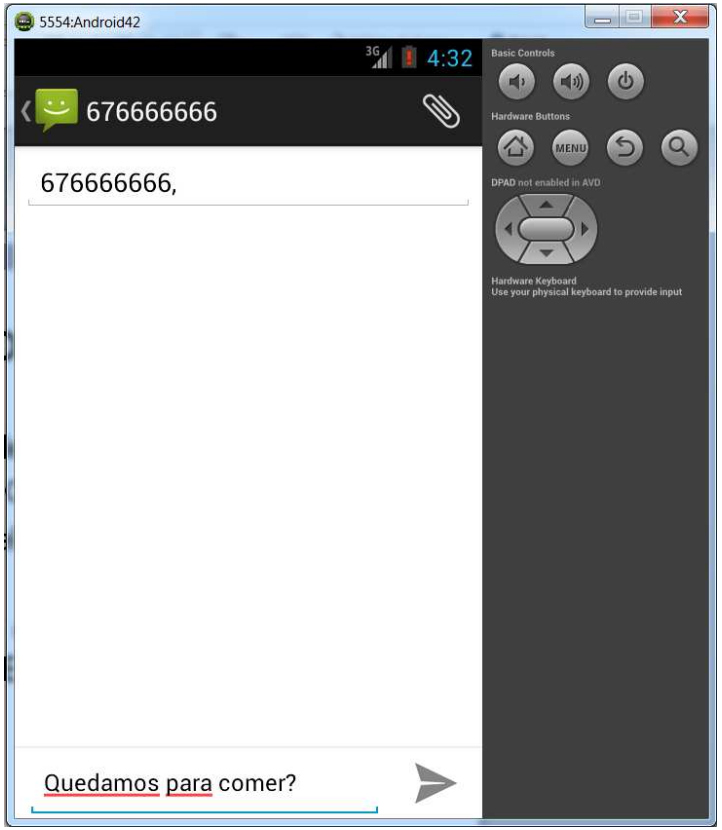
# Action ACTION_WEB_SEARCH

```
Intent search = new Intent(Intent.ACTION_WEB_SEARCH);
search.putExtra(SearchManager.QUERY, "Lidia Fuentes");
startActivity(search);
```



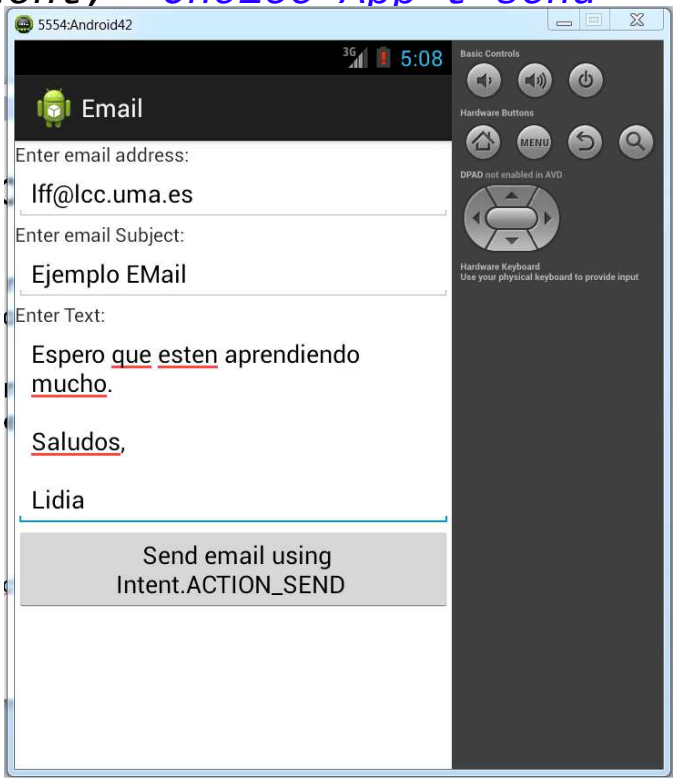
# Action ACTION_SENDTO

```
String smsNumber="676666666";
Uri uri = Uri.parse("smsto:" + smsNumber);
Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
String smsText="Quedamos para comer?";
intent.putExtra("sms_body", smsText);
startActivity(intent);
```



# Action ACTION_SEND

```
Intent intent = new Intent(Intent.ACTION_SEND);
   intent.setType("plain/text");
   intent.putExtra(Intent.EXTRA_EMAIL, “lff@lcc.uma.es”);
   intent.putExtra(Intent.EXTRA_SUBJECT, “Ejemplo EMail”);
   intent.putExtra(Intent.EXTRA_TEXT,
        “Espero que esten aprendiento mucho. Saludos, Lidia”);
   startActivity(Intent.createChooser(intent, "Choice App t send
email:"));


// Si no tenemos el Gmail u otro cliente
// de correo electronico, no funcionara
// este ejemplo
```



# Comunicaciones

- Acceso al paquete Java estándar java.net
  - Socket
  - Clases Java para conexiones HTTP (ej: HttpURLConnection, URL, etc.)
- Acceso a la librería Apache para conexiones http (httpClient)
- Permisos de comunicaciones (fichero manifiesto)
  - Agregar Uses Permission (elegir el requerido por la apps)
    - Permisos para aplicaciones con sockets y conexiones HTTP
      `android.permission.INTERNET`
    - Permisos para verificar la conexión de nuestro sistema
      `android.permission.ACCESS_NETWORK_STATE`

# Verificar conexión

```
// Verificar que tenemos conexión a Internet
 private boolean isNetworkAvailable() {
   ConnectivityManager cm = (ConnectivityManager)
              getSystemService(Context.CONNECTIVITY_SERVICE);
   NetworkInfo networkInfo = cm.getActiveNetworkInfo();

   if (networkInfo != null && networkInfo.isConnected()) {
      return true;
   }
   else {
      return false;
   }
 }
```

# Ejemplo Sockets

```
public boolean Connect() {
  try {
    // Creo socket al estilo java.net
    miCliente = new Socket(“127.0.0.1”, 6543);
    // si conseguimos conectarnos
    if (miCliente.isConnected() == true) {
      return true;
    }
    else {
      return false;
    }
  } catch (Exception e) {
    Log.e("Error connect()", "" + e);
    return false;
  }
}
// DESCARGAR LA APLICACIÓN Y EJECUTARLA DEL CAMPUS VIRTUAL
```

# Apache HTTP

- Creación de un cliente HTTP

```
HttpClient client = new DefaultHttpClient();
```

- Creación de una petición Get

```
HttpGet request = new HttpGet("http://localhost:8084");
```

- Ejecutamos el cliente HTTP

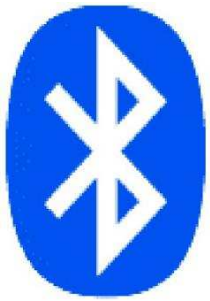
```
HttpResponse response = client.execute(request);
```

- Leemos la página Web

```
BufferedReader rd = new BufferedReader(
  new InputStreamReader(response.getEntity().getContent()));
String line = "";
while ((line = rd.readLine()) != null) {
   paginaWeb.append(line);
}
```

# 4.4. Programación en redes de medio y corto alcance

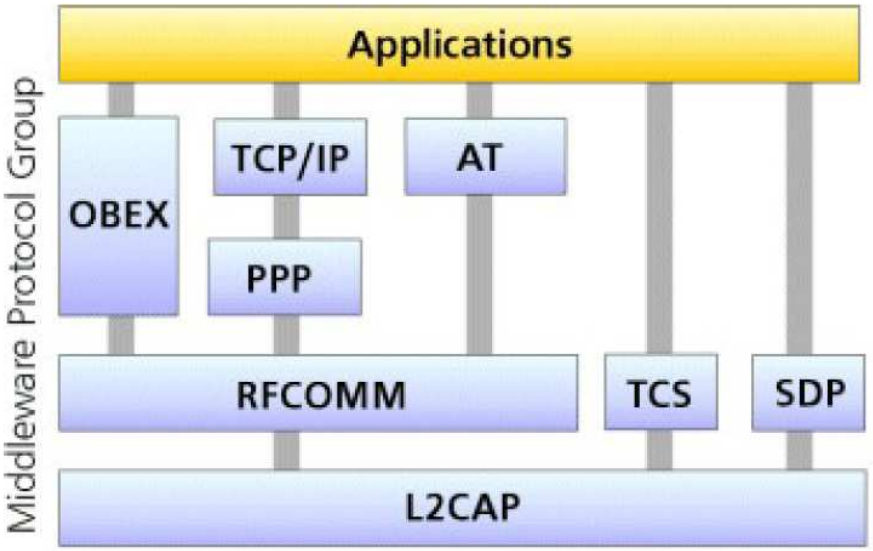
- Protocolo Bluetooth



- Redes WiFi

# Programación en redes de corto alcance

- Protocolos de transporte de Bluetooth
  - RFCOMM
    - Radio Frequency Communications
    - Protocolo orientado a flujo (similar a TCP)
  - L2CAP
    - Logical Link Control and Adaptation Protocol
    - Protocolo orientado a paquete que puede configurase con varios niveles de fiabilidad
    - La configuración será compartida por TODAS las conexiones L2CAP a un mismo dispositivo
    - Sirve para encapsular conexiones RFCOMM



# Programación en redes de corto alcance

- Programación de servicios con Bluetooth
- API Android
  - Búsqueda de otros dispositivos Bluetooth
  - Establecimiento de canales RFCOMM
  - Conexión con otros dispositivos a través del servicio de “descubrimiento de servicios”
  - Transferencia de datos (entrada/salida) Bluetooth
  - Gestión de múltiples conexiones

# Bluetooth Android API

## Paquete android.bluetoooth

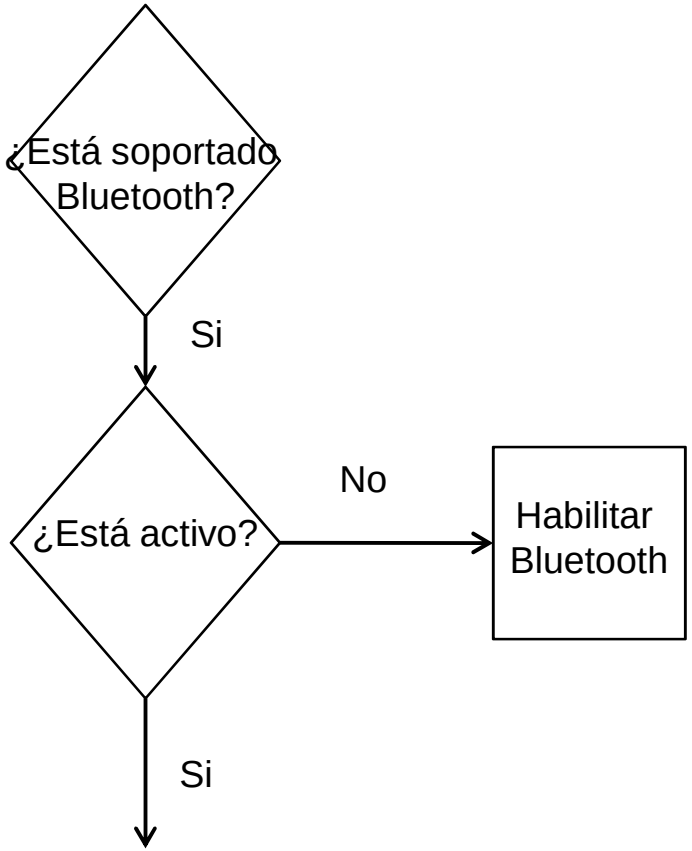
- *BluetoothAdapter*
  - Representa el adaptador bluetooth local, es decir, el dispositivo físico. Es el punto de entrada para interactuar con la interfaz Bluetooth de Android.
- *BluetoothDevice*
  - Modela un dispositivo Bluetooth remoto. Se utiliza para solicitar una conexión con dicho dispositivo
- *BluetoothSocket*
  - Representa una conexión con otro dispositivo. Similar a un socket conectado TCP
- *BluetoothServerSocket*
  - Representa la escucha de conexiones Bluetooth. Similar a un socket pasivo TCP
- *BluetoothClass*
  - Ésta clase contiene un conjunto de propiedades (de solo lectura) que definen las características del dispositivo bluetooth al que representan.
- *BluetoothHeadSet*
  - Proporciona soporte para el manos libres del móvil a través de Bluetooth

# Permisos Bluetooth

- Los permisos relacionados con Bluetooth son **BLUETOOTH** y **BLUETOOTH_ADMIN**
- Para realizar cualquier comunicación Bluetooth (ej: petición de conexión, aceptar una conexión o transferencia de datos) es necesario el permiso BLUETOOTH
- Necesario permiso de administrador (BLUETOOTH_ADMIN) para iniciar el servicio de descubrimiento de servicios o manipular la configuración Bluetooth

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

# Activación Bluetooth




```
private static final int REQUEST_ENABLE_BT = 1;
BluetoothAdapter mBluetoothAdapter =
        BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // El dispositivo no soporta Bluetooth
}
// El dispositivo si soporta Bluetooth
else {
  // Comprueba si Bluetooth esta activo
  if (!mBluetoothAdapter.isEnabled()) {
    // Habilita Bluetooth
    Intent enableBtIntent = new Intent
       (BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent,
                           REQUEST_ENABLE_BT);
  }
}
```

# Buscando de Dispositivos

- Búsqueda de dispositivos conocidos
  - Dispositivos para los que ya existe una conexión RFCOMM establecida

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// Si hay algun dispositivo conocido
if (pairedDevices.size() > 0) {
    // Itero en la lista de dipositivos conocidos encontrados
    for (BluetoothDevice device : pairedDevices) {
        // Agrega a un array los dispositivos por nombre y direccion MAC
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
```

# Buscando Dispositivos

- Descubrimiento de nuevos dispositivos

`mBluetoothAdapter.startDiscovery()`

  - Es asíncrono y la búsqueda dura unos 12 seg., además de solicitar el nombre de cada uno de los dispositivos encontrados

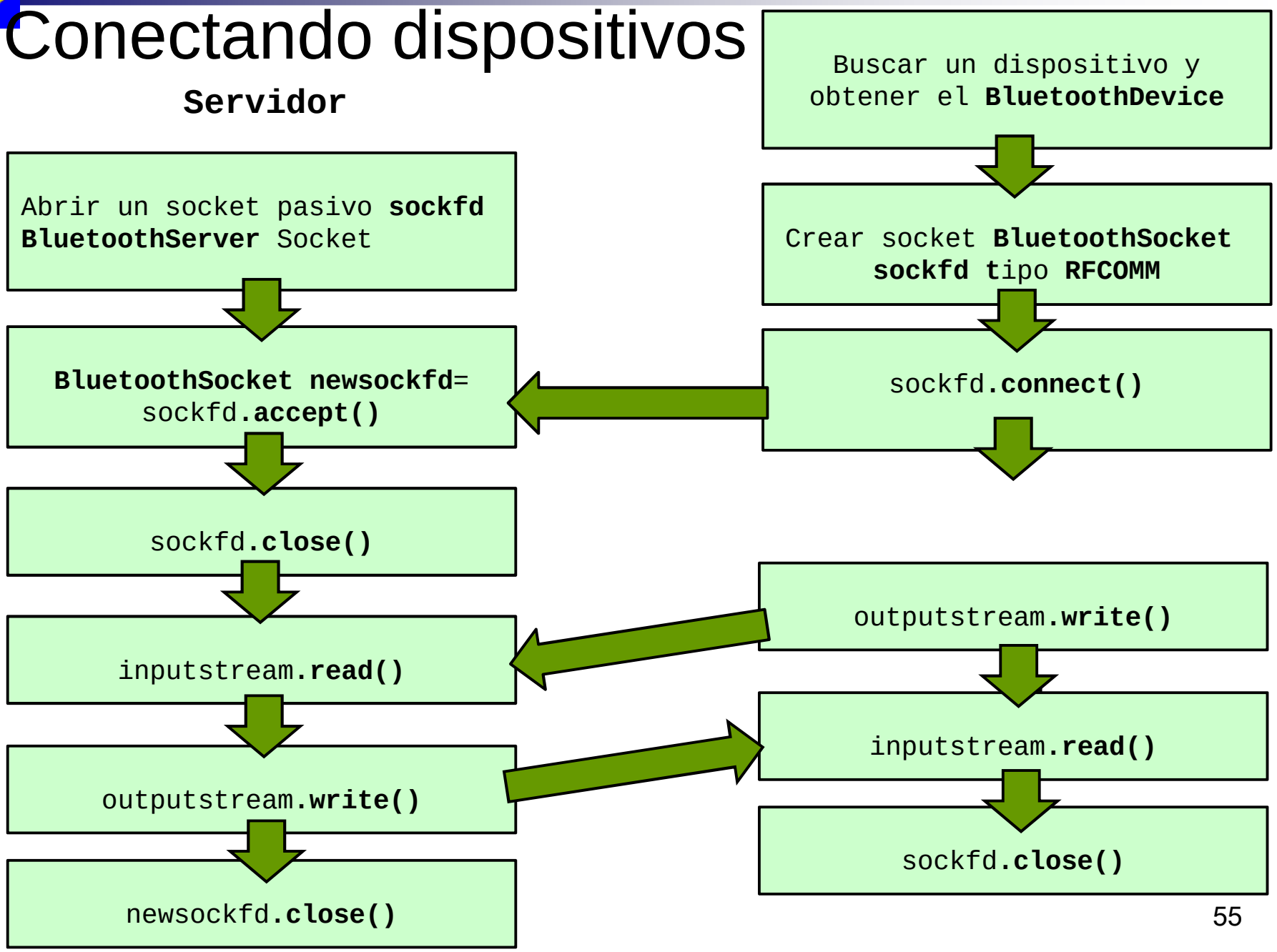
```
// Crea un BroadcastReceiver para la accion ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    // Se ha encontrado un dispositivo
    if (BluetoothDevice.ACTION_FOUND.equals(action)) {
      // Obtengo el objeto BluetoothDevice del Intent
      BluetoothDevice device = intent.
                              getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
      // Agrego nombre y direccion MAC a un array para mostrar la lista
      mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
  }};
// Registrar el BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
```

# Modo descubrimiento

- Para poner a tu dispositivo en modo descubrimiento (para que otros dispositivos lo encuentren)
  - Es posible establecer un intervalo de tiempo
    - 0: tiempo ilimitado
    - máximo 3600 segundos

```
Intent discoverableIntent = new Intent
                            (BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

# Conectando dispositivos



# Android: WiFi

- El paquete `android.net.wifi` provee los mecanismos por los cuales una aplicación Android puede acceder a la pila Wifi del sistema
- Las clases contenidas en android.net.wifi informan desde los puntos de acceso detectados hasta el estado de la propia conexión
- El paquete también proporciona los métodos necesarios para escanear el entorno, iniciar y detener conexiones, configurar nuevas conexiones, etc
- Permisos
  - ACCESS_WIFI_STATE
  - CHANGE_WIFI_STATE

# WiFi Android API

- *WifiManager*
  - Es el punto de entrada a los servicios WIFI del sistema.
- *WifiManager.MulticastLock*
  - Permite a una aplicación recibir paquetes Multicast a través de la WIFI.
- *WifiManager.WifiLock*
  - Permite a una aplicación mantener la radio WIFI activa aún cuando no se esté utilizando y hasta que se elimine el bloqueo.
- *ScanResult*
  - Contiene información relativa a los dispositivos descubiertos durante un escaneo del entorno
- *WifiConfiguration*
  - Clase encargada de representar una configuración WIFI válida

# Ejemplo: Obtener la lista de redes WiFi

```
// Clase WiFiDemo
WifiManager wifi;

// Gestionar la WiFi
wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);

// Obtener el estado de la red WiFi local
WifiInfo info = wifi.getConnectionInfo();
textStatus.append("\n\nWiFi Status: " + info.toString());

// Obtener la lista de redes WiFi registradas
List<WifiConfiguration> configs = wifi.getConfiguredNetworks();
for (WifiConfiguration config : configs) {
        textStatus.append("\n\n" + config.toString());
}

// Registrar esta clase como Broadcast Receiver
BroadcastReceiver receiver = new WiFiScanReceiver(this);

registerReceiver(receiver, new IntentFilter(WifiManager.
                                    SCAN_RESULTS_AVAILABLE_ACTION));
```

# Clase BroadcastReceiver

```
public class WiFiScanReceiver extends BroadcastReceiver {
  WiFiDemo wifiDemo;

  public WiFiScanReceiver(WiFiDemo wifiDemo) {…}

  @Override
  public void onReceive(Context c, Intent intent) {
    List<ScanResult> results = wifiDemo.wifi.getScanResults();
    ScanResult bestSignal = null;
    for (ScanResult result : results) {
      if (bestSignal == null
          || WifiManager.compareSignalLevel(bestSignal.level, result.level) < 0)
        bestSignal = result;
    }

    String message = String.format("%s networks found. %s is the strongest.",
        results.size(), bestSignal.SSID);
    Toast.makeText(wifiDemo, message, Toast.LENGTH_LONG).show();
}

}
```