La Clase `Service`.

CONCEPTOS.

Un `Service` es el componente de una aplicación que realiza operaciones de larga ejecución en segundo plano y no proporciona una interfaz de usuario. Otro componente de la aplicación puede iniciar un servicio y seguirá funcionando en segundo plano, incluso si el usuario cambia a otra aplicación. Además, un componente puede vincularse a un servicio para interactuar con él e incluso realizar la comunicación entre procesos (IPC). Por ejemplo, un servicio puede manejar las transacciones de red, reproducir música, ejecutar E/S de archivos, o interactuar con un proveedor de contenidos, todo ello en segundo plano.

Un servicio puede tomar dos formas:

- **Started**

Un servicio se inicia cuando un componente de la aplicación (por ejemplo, una actividad) lo inicia mediante una llamada a `startService`(). Una vez iniciado, el servicio se ejecuta en segundo plano de forma indefinida, incluso si el componente que lo inició se destruye.

- **Bound**

Un servicio se liga cuando un componente de la aplicación se enlaza a él invocando a `bindService`(). Un servicio ligado ofrece una interfaz cliente-servidor que permite que los componentes interactúen con el servicio, envíen solicitudes, obtengan resultados, e incluso hacerlo a través de procesos con la comunicación entre procesos (IPC). Un servicio ligado se ejecuta solamente mientras otro componente de la aplicación se encuentre vinculado al mismo. Múltiples componentes pueden unirse al servicio a la vez, pero cuando todos ellos se desligan, el servicio se destruye. Se puede declarar el servicio como privado, en el archivo de manifiesto, y bloquear el acceso desde otras aplicaciones.

Nota: Un servicio se ejecuta en el hilo principal del proceso que lo posee; el servicio no crea su propio hilo y no se ejecuta en un proceso separado (a menos que se especifique lo contrario).

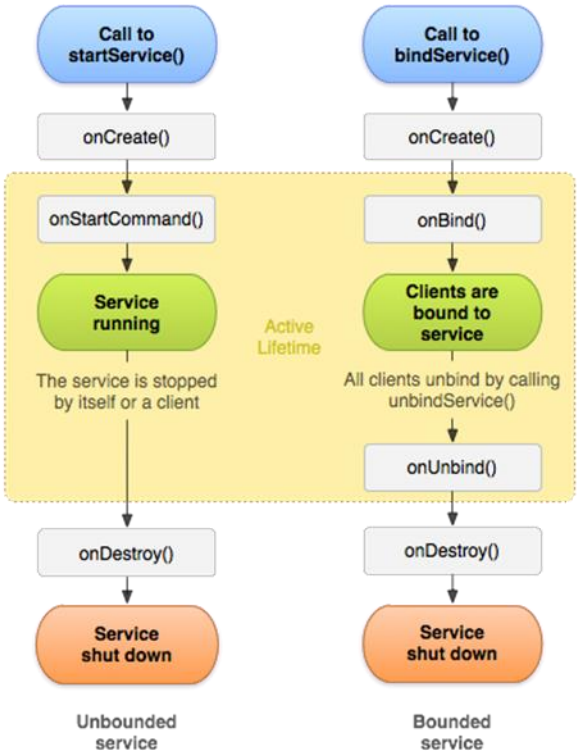

Figura 1. El ciclo de vida de `Service`. El diagrama izquierdo muestra el ciclo de vida cuando se crea el servicio con `startService`() y el diagrama derecho muestra el ciclo de vida cuando se crea el servicio con `bindService`().

Lo básico.

Qué utilizar ¿Un servicio o un hilo?

Un servicio es sólo un componente que se ejecuta en segundo plano, incluso cuando el usuario no está interactuando con la aplicación. Por lo tanto, se debe crear un servicio solamente si eso es lo que se necesita.

Si se necesita realizar un trabajo fuera del hilo principal, pero sólo mientras el usuario está interactuando con la aplicación, entonces se debería crear en su lugar un nuevo hilo y no un servicio. Por ejemplo, si se desea reproducir música, pero solamente mientras la actividad se está ejecutando, se puede crear un hilo con `onCreate` (),`onStart`() y `onStop`(). Considerar el uso de `AsyncTask` o `HandlerThread` , en lugar de la clase `Thread` tradicional.

Si se hace uso de un servicio, éste aún se ejecuta en el hilo principal de la aplicación de forma predeterminada, por lo que aun así se debe crear un nuevo hilo en el servicio si éste realiza operaciones intensivas o de bloqueo.

Para crear un servicio, se crea una subclase de `Service` (o una de sus subclases); además, sobrescribir algunos métodos importantes que manejen el ciclo de vida y sean el mecanismo para que los componentes se liguen al servicio. Los métodos más importantes son:

`onStartCommand`()	Se invoca cuando otro componente solicita que se inicie el servicio. El servicio se inicia, corre indefinidamente y se le debe detener cuando se termine, con `stopSelf`() o `stopService`().
`onBind`()	Se invoca si otro componente se liga con el servicio. Debe proveer una interface para que los clientes se comuniquen con el servicio, retornando un `IBinder`. El método `onBind`() siempre se debe implantar, pero si no se desea el ligado, entonces se debe regresar `null`.
`onCreate`()	Se invoca cuando se crea el servicio y ejecuta configuraciones iniciales (antes de invocar a `onStartCommand`() o `onBind`()). Si el servicio está en ejecución, no se invoca a este método.
`onDestroy`()	Se invoca cuando el servicio ya no se utiliza y se destruye. El servicio debe implantar este método para limpiar los recursos, como hilos, escuchas, receptores, y otros. Esta invocación es la última que el servicio recibe.

Declaración del `Service` en el AndroidManifes.xml.

Al igual que las actividades, los servicios se deben declarar en el archivo de manifiesto de la aplicación:

```
<manifest ... >
  :
  <application ... >
      <service android:name=".MiServicio" />
      :
  </application>
</manifest>
```

Siempre se debe utilizar un `Intent` explícito para iniciar o ligar un servicio y no declarar `intent-filters` en el servicio.

Creación de un Servicio Started (iniciado).

Un servicio se inicia cuando otro componente invoca a `startService`(), que resulta en una llamada al método `onStartCommand` () del servicio.

Para crear un servicio iniciado, se puede heredar de dos clases, `Service` o `IntentService`:

`Service` Es la clase base de todos los servicios. Es importante que se cree un nuevo hilo en el que se realice todo el trabajo del servicio, debido a que el servicio utiliza el hilo principal de la aplicación, de forma predeterminada, y podría retardar la ejecución de cualquier actividad de la aplicación.

`IntentService` Es una subclase de `Service`, que utiliza un hilo para manejar todas las peticiones de inicio, una a la vez. Es la mejor opción si no se requiere que el servicio maneje múltiples peticiones simultáneamente. Solamente se implanta `onHandleIntent`(), que recibe el intento por cada solicitud de inicio, por lo que sólo resta hacer el trabajo de segundo plano.

Heredando de la Clase `IntentService`.

La mayoría de los servicios iniciados no requieren manejar múltiples solicitudes simultáneamente y lo mejor es utilizar la clase `IntentService` para implantar el servicio. El `IntentService` hace lo siguiente:

- Crear un hilo predeterminado que ejecuta todas los intentos entregados a `onStartCommand`(), separado del hilo principal de la aplicación.
- Crear una cola de trabajo, que pasa un intento a la vez al `onHandleIntent`(), para que nunca se tenga que preocupar de múltiples hilos.
- Detiene el servicio después de que todas las solicitudes de inicio se han realizado, por lo que nunca tiene que llamar `stopSelf`().
- Proporciona la implantación predeterminada de `onBind` (), que devuelve un valor `null`.
- Proporciona la implantación predeterminada de `onStartCommand`() que envía el intento a la cola de trabajo y luego a la aplicación `onHandleIntent`().

Por ejemplo:

```
public class HelloIntentService extends IntentService {
  public HelloIntentService() {
      super("HelloIntentService");
  }
  @Override
  protected void onHandleIntent(Intent intent) {
      try {
          Thread.sleep(5000);
      } catch (InterruptedException e) {
          // Restore interrupt status.
          Thread.currentThread().interrupt();
      }
  }
}
```

Heredando de la Clase `Service`.

El uso de `IntentService` hizo muy sencilla la implantación del servicio iniciado. Si se requiere que el servicio realice multi-threading (en lugar de procesar las peticiones de inicio a través de una cola de trabajo), se puede heredar de la clase `Service` para manejar cada intento.

El código siguiente es una implantación de `Service` que hace exactamente lo mismo que el ejemplo anterior con `IntentService`. Es decir, por cada solicitud de inicio, se utiliza un hilo para realizar el trabajo y procesar una sola solicitud a la vez. Es evidente que es más trabajo, por ejemplo:

```
public class HelloService extends Service {
  private Looper mServiceLooper;
  private ServiceHandler mServiceHandler;
  private final class ServiceHandler extends Handler {
      public ServiceHandler(Looper looper) {
          super(looper);
      }
      @Override
      public void handleMessage(Message msg) {
          try {
              Thread.sleep(5000);
          } catch (InterruptedException e) {
              Thread.currentThread().interrupt();
          }
          stopSelf(msg.arg1);
      }
  }
  public void onCreate() {
```

```
    HandlerThread thread = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
  }
  public int onStartCommand(Intent intent, int flags, int startId) {
      Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
      Message msg = mServiceHandler.obtainMessage();
      msg.arg1 = startId;
      mServiceHandler.sendMessage(msg);
      return START_STICKY;
  }
  public IBinder onBind(Intent intent) {
      return null;
  }
  public void onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
  }
}
```

Creación de un Servicio Bound (Ligado).

Un servicio bound permite que los componentes de una aplicación se le unan llamando a `bindService` () con el fin de crear una conexión de larga duración (no permiten que los componentes lo inicien llamando a `startService()`).

Para crear un servicio bound, se implanta el método `onBind` () para devolver un `IBinder` que define la interfaz para la comunicación con el servicio. Otros componentes de la aplicación pueden llamar a `bindService` () para recuperar la interfaz y comenzar a llamar a métodos en el servicio. Además, lo primero que hay que hacer es definir la interfaz, entre el servicio y el cliente, y debe ser una implantación de `IBinder` y es lo que el servicio debe devolver de la llamada a `onBind`(). Una vez que el cliente recibe el `IBinder`, se empieza a interactuar con el servicio a través de esa interfaz.

Varios clientes se pueden enlazar con el servicio a la vez. Cuando un cliente interactúa con el servicio, llama a `unbindService`() para desenlazarse. Una vez que no hay clientes unidos al servicio, el sistema destruye el servicio.

DESARROLLO

EJEMPLO 1.

Paso 1. Crear un nuevo proyecto Servicios. En la actividad principal MainActivity.java, capturar el siguiente código:

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.*;
public class MainActivity extends Activity { // ServiceTimerActivity
    private TextView    jtv;
    private Button      jbn;
    @Override
    public void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        jtv = (TextView) findViewById(R.id.xtvT);
        jbn = (Button) findViewById(R.id.xbnI);
        jbn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                initCrono();
            }
```

```
        });
        Button stopButton = (Button) findViewById(R.id.xbnT);
        stopButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                stopCrono();
            }
        });
        MiCrono.setUpdateListener(this);
    }
    @Override
    protected void onDestroy() {
        stopCrono();
        super.onDestroy();
    }
    private void initCrono() {
        Intent in = new Intent(this, MiCrono.class);
        startService(in);
    }
    private void stopCrono() {
        Intent in = new Intent(this, MiCrono.class);
        stopService(in);
    }
    public void refreshCrono(double t) {
        jtv.setText(String.format("%.2f", t) + " segs");
    }
}
```

Paso 2. En el archivo activity_main.xml predeterminado, capturar el siguiente código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/xtvT"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:layout_marginBottom="20dp"
        android:layout_marginTop="20dp"
        android:background="#147"
        android:textColor="#fff"
        android:gravity="center" />
    <Button
        android:id="@+id/xbnI"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Iniciar" />
    <Button
        android:id="@+id/xbnT"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Terminar" />
</LinearLayout>
```

Paso 3. En la carpeta java/com.example.escom.servicios, crear el archivo Java MiCrono.java y capturar el siguiente código:

```
import java.util.*;
import android.app.Service;
import android.content.Intent;
import android.os.*;
public class MiCrono extends Service {
    private Timer t = new Timer();
    private static final long INTERVALO_ACTUALIZACION = 10; // En milisegundos
    public static MainActivity UPDATE_LISTENER;
    private double n=0;
    private Handler h;
    public static void setUpdateListener(MainActivity sta) {
        UPDATE_LISTENER = sta;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        iniciarCrono();
        h = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                UPDATE_LISTENER.refreshCrono(n);
            }
        };
    }
    @Override
    public void onDestroy() {
        pararCrono();
        super.onDestroy();
    }
    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }
    private void iniciarCrono() {
        t.scheduleAtFixedRate(new TimerTask() {
            public void run() {
                n += 0.01;
                h.sendEmptyMessage(0);
            }
        }, 0, INTERVALO_ACTUALIZACION);
    }
    private void pararCrono() {
        if (t != null)
            t.cancel();
    }
}
```

Nota: El `Handler` se declara como clase interna y evita que la clase exterior se destruya con el recolector de basura.
No hay problema si el `Handler` utiliza un looper o `MessageQueue` para un subproceso distinto del hilo principal.
Si el `Handler` utiliza el `Looper` o `MessageQueue` del hilo principal, se debe modificar la declaración del `Handler`:

- Declarar el `Handler` como una clase estática;
- En la clase externa, instanciar un `WeakReference` a la clase externa y pasar este objeto a su `Handler` cuando se instancia el `Handler`.
- Hacer las referencias a todos los miembros de la clase externa utilizando el objeto `WeakReference`.

Paso 4. En la carpeta app/manifests, abrir el archivo AndroidManifest.xml y modificarlo con la inserción de la etiqueta <service>, entre las etiquetas <application> y </application>, es decir:
<service android:name="MiCrono" />

Como se indica enseguida, con **letras negritas**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.escom.servicios">
    <application
            :
            : >
        <activity
            android:name=".MainActivity"
                  : >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MiCrono" />
    </application>
</manifest>
```

Paso 5. Por último, ejecutar la aplicación. Digitar el botón Iniciar. Enseguida se inicia el servicio con la cuenta del cronómetro. Digitar el botón Terminar, para detener el cronómetro.

EJERCICIO 1.
Realizar los siguientes cambios al Ejemplo 1 anterior.
a. Incluir un nuevo botón de pausa, en la plantilla principal, para detener momentáneamente el cronómetro.
b. Incluir un nuevo botón de continuar, en la plantilla principal, para continuar la cuenta del cronómetro después de la pausa.

La aplicación debe ser similar a la siguiente imagen:

Nota: Analizar la reutilización de los botones de inicio y/o término.