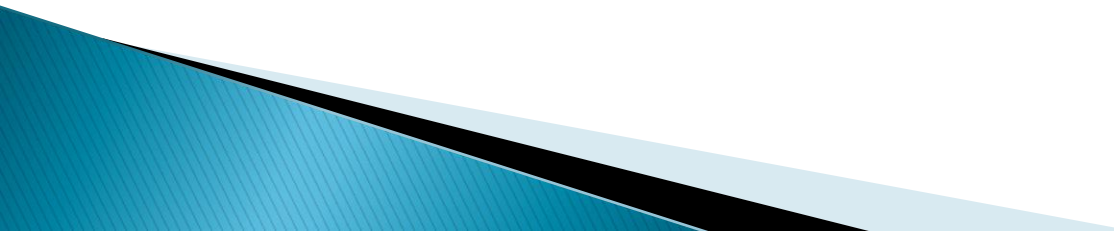


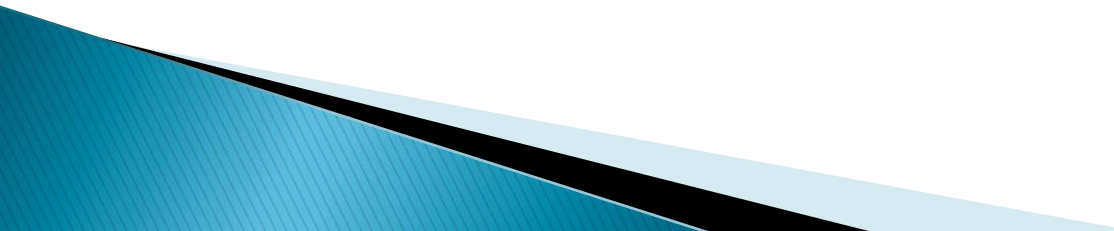
# Android: Activity

`android.app.Activity`

# Activity: concepto

- ▶ Una actividad es una componente de aplicación visual con la que el usuario puede interactuar.
  - ▶ Una aplicación suele consistir en una serie de actividades interrelacionadas entre ellas.
  - ▶ Cada vez que una actividad es ejecutada, la actividad actual es detenida y almacenada en la pila (*back stack*).
  - ▶ La pila de actividades se gestiona según el algoritmo LIFO.
- 

# Activity: máquina de estados

- ▶ Las actividades tienen un ciclo de vida en el que van pasando de un estado a otro.
  - ▶ Cada vez que se produce un cambio de estado en una actividad, se puede programar una respuesta a esa transición.
  - ▶ Por ejemplo, cuando una actividad pasa del estado activo al estado parado, es importante liberar los recursos pesados, como las conexiones de red o de bases de datos.
- 

# Activity: creación

- ▶ Se debe crear una clase que extienda la clase Activity.
- ▶ La clase debe sobrescribir el método onCreate(), en el que se debe llamar en primer lugar al propio método de la superclase (super.onCreate()). Lo habitual es que además se establezca el diseño de la actividad con el método setContentView().

# Activity: AndroidManifest.xml

- ▶ Todas las actividades de una aplicación se tienen que declarar en el manifiesto.
- ▶ La actividad principal o inicial de una aplicación debe declararse con los *intent filters* MAIN y LAUNCHER.

```
<activity
  android:name="com.izv.pasoapaso.Principal"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

# Activity: Intent I

- ▶ Lanzar una actividad

```
Intent i = new Intent(this,ActividadDestino.class);  
startActivity(i);
```

- ▶ Lanzar una actividad pasando datos

```
Intent i = new Intent(this,ActividadDestino.class);  
Bundle b = new Bundle();  
b.putString("id", valorString);  
b.putSerializable("objeto", objeto);  
i.putExtras(b);  
startActivity(i);
```

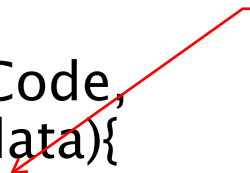
# Activity: Intent II

- ▶ Lanzar una actividad para obtener un resultado

```
private final int ACTIVIDAD=1;  
Intent i = new Intent(this, ActividadDestino.class);  
startActivityForResult(i, ACTIVIDAD);
```

```
@Override  
public void onActivityResult(int requestCode,  
                             int resultCode, Intent data){  
    if (resultCode == Activity.RESULT_OK &&  
        requestCode == ACTIVIDAD) {  
    }  
}
```

El boton return  
devuelve reultado  
negativo



# Activity: Bundle I

getParcelable en lugar de getSerializable. Se le pone a clase implements Parcelable. Necesita los metodos describeContents() y writeToParcel(Parcel p, int flags) en el que se usa writeStringArray que es un metodo al que se le pasa new String[] {valor, valor}

- Recoger datos obtenidos como resultado

```
int tiempo = data.getIntExtra("tiempo",0);  
String nom = data.getStringExtra("nombre");  
Clase o = (Clase)data.getSerializableExtra("objeto");
```

- Recoger datos obtenidos al ser llamado

```
Bundle b = getIntent().getExtras();  
Clase o = (Clase)b.getSerializable("objeto");  
String s = b.getString("id");
```



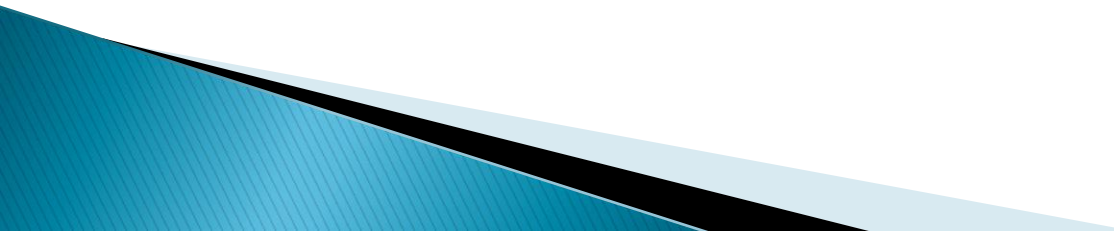
# Activity: Bundle II

- ▶ Devolver resultado

```
Intent i = new Intent();  
Bundle bundle = new Bundle();  
bundle.putLong("tiempo", valor);  
bundle.putString("nombre", valor2);  
bundle.putSerializable("objeto", valor3);  
i.putExtras(bundle);  
setResult(Activity.RESULT_OK, i);  
finish();
```

- ▶ No devolver resultado

```
Intent i = new Intent();  
setResult(Activity.RESULT_CANCELED, i);  
finish();
```

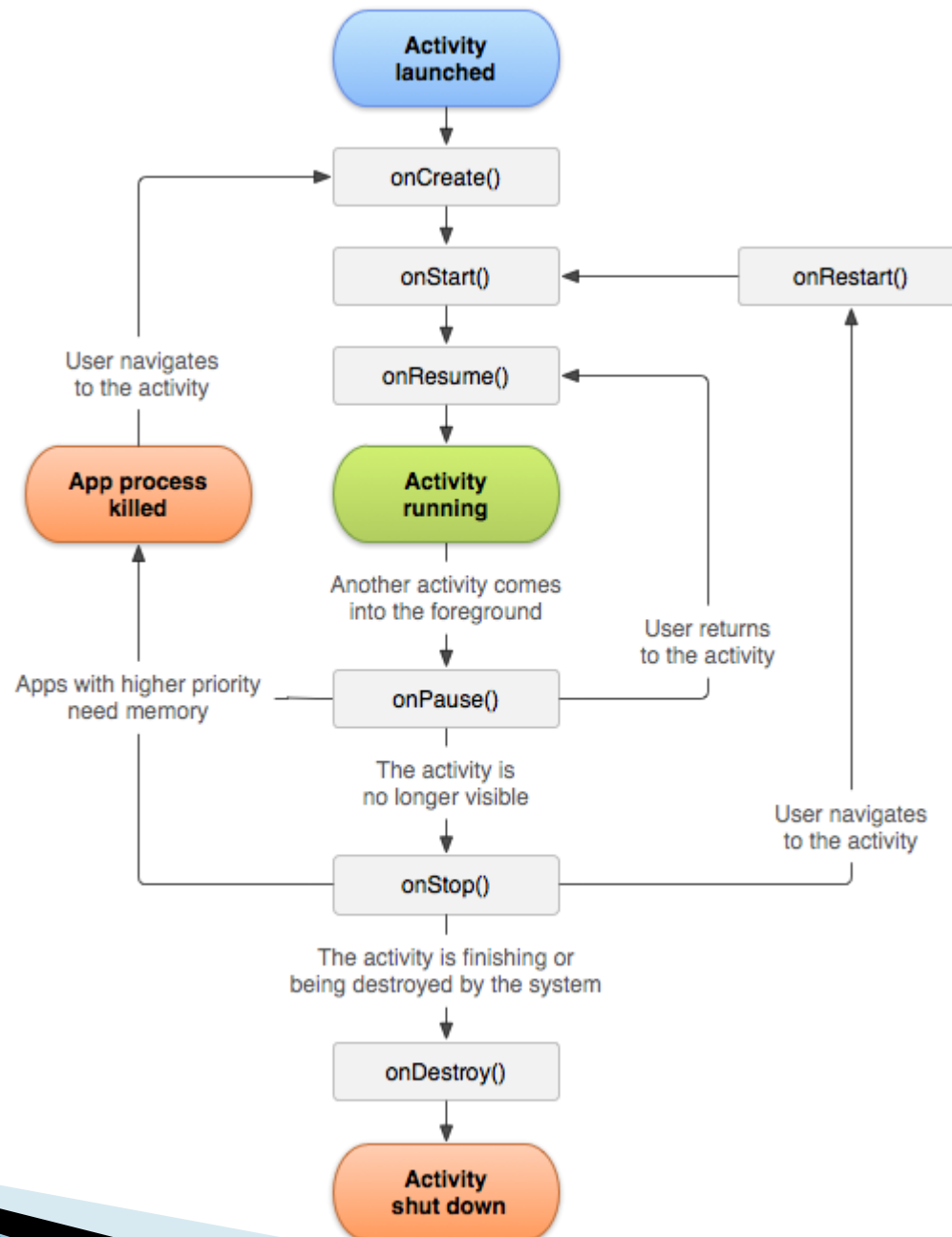


# Activity: lifecycle

- ▶ El método `finish()` finaliza la actividad.
- ▶ El método `finishActivity()` finaliza una actividad diferente.
- ▶ Una actividad se puede encontrar esencialmente en uno de los siguientes estados:
  - *Resumed*, actividad actual en ejecución.
  - *Paused*, actividad visible pero sin el foco.
  - *Stopped*, actividad no visible, es susceptible de ser eliminada por el sistema.

# Activity: lifecycle callbacks

```
public class Actividad extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);} //creando  
    @Override  
    protected void onStart() {  
        super.onStart();} //visibilizando  
    @Override  
    protected void onResume() {  
        super.onResume();} //visible  
    @Override  
    protected void onPause() {  
        super.onPause();} //visible, sin foco  
    @Override  
    protected void onStop() {  
        super.onStop();} //no visible  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();} //destruyendo  
}
```



# Activity: callbacks methods

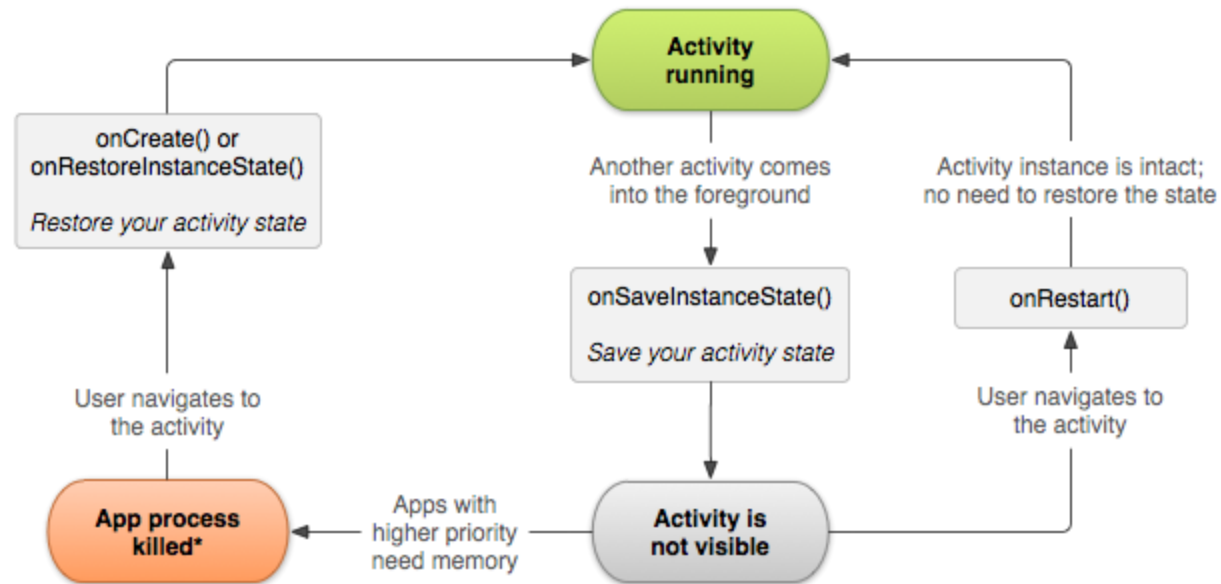
onCreate	Al crear la actividad por primera vez.
onRestart	Al haber detenido una actividad y volver a iniciarla.
onStart	Justo antes de mostrar la actividad.
onResume	Al mostrarse justo antes de obtener el foco.
onPause	Justo antes de ejecutar otra actividad. Debe guardar todos los datos pendientes y soltar los recursos que consuman CPU.
onStop	Al dejar de estar visible la actividad.
onDestroy	Al destruir la actividad. El método <code>isFinishing()</code> permite distinguir si se destruye por una llamada <code>finish()</code> o si es el sistema el que destruye la actividad. Debe destruir las hebras en segundo plano que pudiera haber activas.

Si el sistema destruye una actividad, no siempre se garantiza la ejecución de `onStop` y `onDestroy`.

# Activity: onSaveInstanceState

Antes de pasar la actividad al estado pausado el método `onSaveInstanceState()` puede ser sobrescrito para almacenar información que ayude a reconstruir la actividad.

El método más indicado para recuperar la información es `onRestoreInstanceState()`.



\*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

# Activity: onRestoreInstanceState

```
private String valor;
```

```
@Override
```

```
protected void onSaveInstanceState(Bundle savingInstanceState) {  
    super.onSaveInstanceState(savingInstanceState);  
    savingInstanceState.putString("cadena", valor);  
}
```

```
@Override
```

```
protected void onRestoreInstanceState(Bundle savedInstanceState) {  
    super.onRestoreInstanceState(savedInstanceState);  
    valor = savedInstanceState.getString("cadena");  
}
```

# Activity: onCreate

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.layout_principal);  
    if (savedInstanceState != null) {  
        valor = savedInstanceState.getString("cadena");  
    }  
}
```

@Override

```
protected void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState);  
    savedInstanceState.putString("cadena", valor);  
}
```

En lugar de sobrescribir el método `onRestoreInstanceState()` se puede usar el método `onCreate()` para recuperar los valores almacenados antes de haber pausado la actividad.



# Activity: retener un objeto

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    Clase dato = (Object) getLastNonConfigurationInstance();
    if (dato == null) {
        dato = new Clase();
    }
}

@Override
public Object onRetainNonConfigurationInstance() {
    return datoOriginal;
}
```

Esta forma de retener objetos está actualmente desaconsejada.

# Activity: orientación

- ▶ Cuando se produce un cambio de orientación del dispositivo, algunas partes de la actividad se tienen que reconstruir usando los métodos estudiados.
- ▶ En el manifiesto se puede indicar si una actividad admite o no cambios de orientación.

```
<activity  
    android:name=".NombreActividad"  
    android:screenOrientation="portrait"  
    android:label="@string/titulo_actividad" >
```