Una aplicación Android consiste en una serie de componentes poco acoplados y perfectamente estructurados, los cuales son unidos por el archivo de manifiesto AndroidManifest.xml. Estos componentes son:

- **Actividades**: Una actividad es una pantalla de una aplicación, es decir la interfaz gráfica (UI – User Interface) con la que interactúa el usuario. Una aplicación está compuesta de múltiples actividades. Las Actividades utilizan Fragmentos (Fragments) y Vistas (Views) para la disposición (layout) de los elementos y así mostrarlos al usuario.

- **Servicios**: Los Servicios son procesos que trabajan en segundo plano (Background). Los servicios funcionan sin interfaz gráfica (UI), y suelen actualizar datos de los recursos, generar notificaciones, entre otro tipo de eventos.

- **Proveedores de Contenido** (Content Providers): Los proveedores de contenido exponen y almacenan información a todas las aplicaciones Android. Esta es la manera en que Android comparte información entre aplicaciones. Algunos ejemplos de proveedores de contenido son: Audio, vídeo, imágenes, información de contactos, etc.

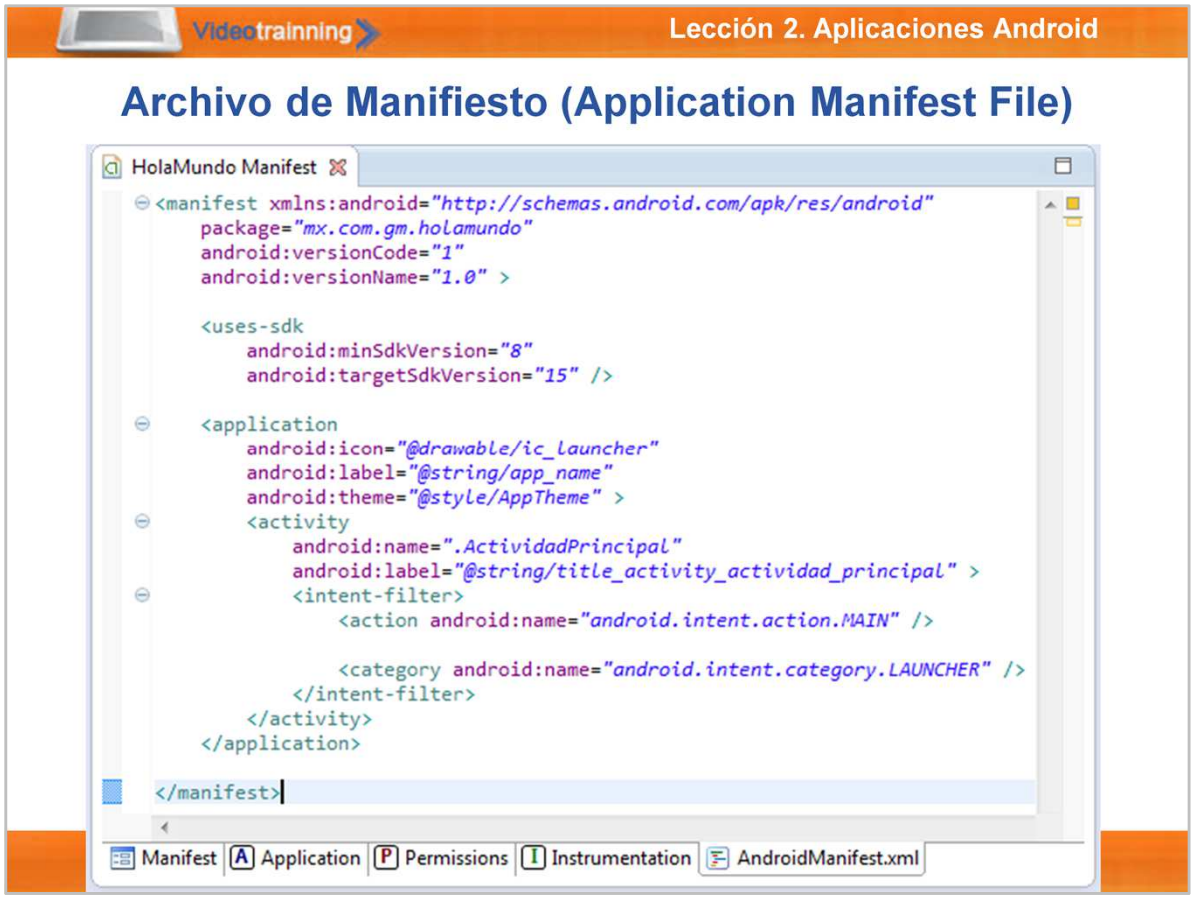
- **Intenciones** (Intents): Los intents es la forma en que Android envía mensajes a otras aplicaciones. Nuestras aplicaciones pueden indicar al sistema Android que queremos iniciar otro componente (otra actividad por ejemplo) de la misma aplicación u otra distinta, para ello utilizaremos un Intent.

- **Receptores de Mensajes** (Broadcast Receivers): Puede recibir los mensajes del sistema y las solicitudes (intents). Se pueden utilizar para responder a cambios, por ejemplo una aplicación se puede iniciar ante un modificaciones en el sistema.

- **Widgets**: Son componentes visuales que se agregan típicamente al Home Screen del teléfono.

- **Notificaciones**: Las notificaciones permiten avisar a los usuarios algún evento de nuestra aplicación sin interrumpirlos de su actividad actual, esto sin importar si nuestra aplicación está visible o no.

Una de las características más importantes de Android, es que cualquier aplicación puede iniciar otro componente de cualquier otra aplicación, por ejemplo, si queremos capturar una fotografía, podemos utilizar otra aplicación que ya tenga esta funcionalidad y aprovecharla. Así únicamente nos enfocaremos en desarrollar los requerimientos base de nuestra aplicación.



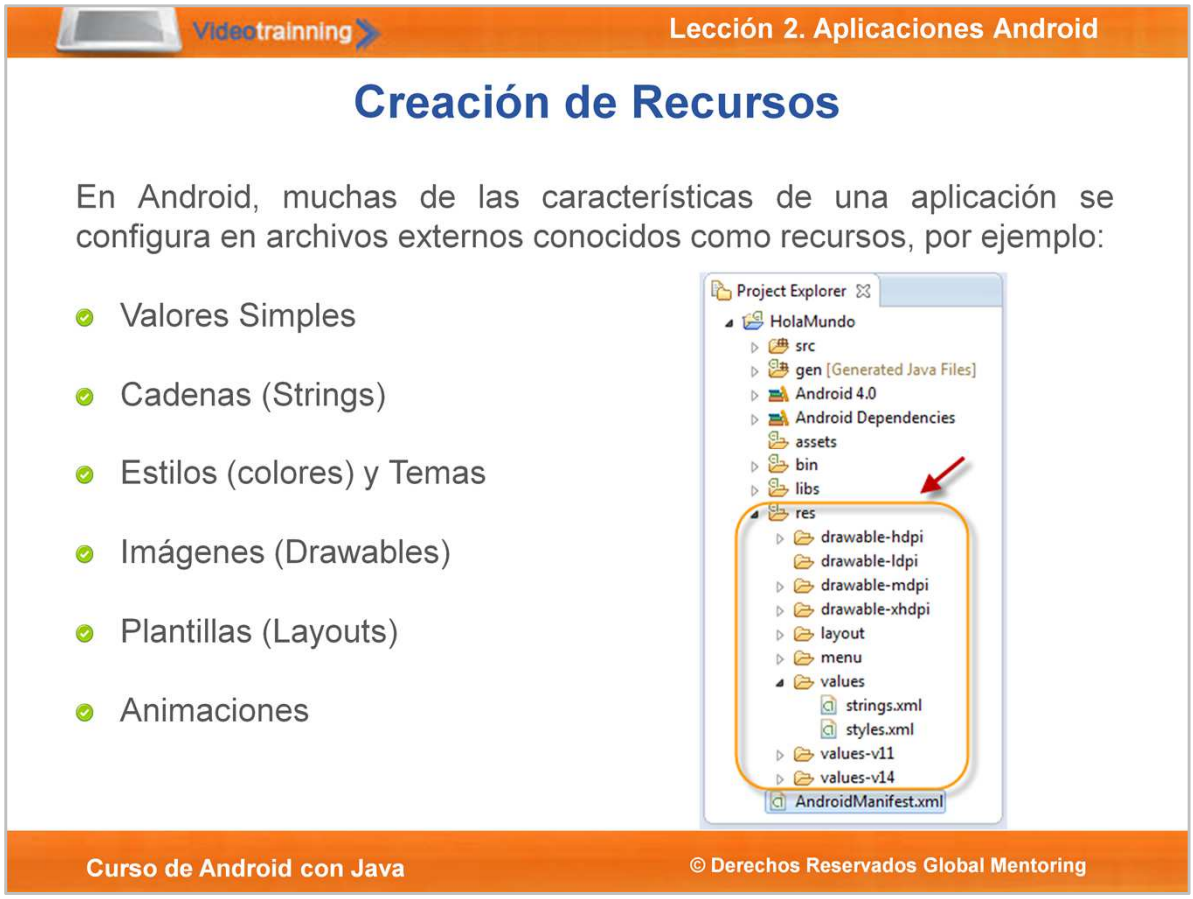
El archivo de manifiesto de Android permite especificar muchas de las características que incluirá nuestra aplicación:

- Cada proyecto Android contiene un archivo llamado AndroidManifest.xml
- Este archivo xml permite definir la estructura, metadatos, componentes, permisos, etc, de la aplicación Android.
- Permite Declarar Actividades, Servicios, Intents, Content Providers y demás elementos a utilizar en nuestra aplicación.
- Podemos utilizar el hardware si solicitamos el permiso respectivo. El hardware que podemos utilizar puede ser Audio, Bluetooth, Cámara, GPS, Micrófono, Sensores, USB, Wi-Fi, entre otros.
- Permite especificar de metadatos, tales como la versión de la aplicación, el ícono a utilizar, tema a aplicar, entre otros metadatos.

En la figura podemos ver un ejemplo de un archivo AndroidManifest.xml de nuestra aplicación HolaMundo. Podemos observar la versión de nuestra aplicación, el sdk mínimo que soportará nuestra aplicación, y dentro de la etiqueta <application/> observamos metadatos de la aplicación tales como el ícono a utilizar, el nombre de la aplicación, etc. Así mismo observamos una actividad, la cual al definirla dentro del intent-filter será la actividad principal de nuestra aplicación.

Eclipse nos facilita la edición del archivo de manifiesto al agregar un editor gráfico que nos facilita configurar cada una de las posibilidades de este archivo.

A lo largo del curso iremos conociendo y aplicando varios de los aspectos que podemos configurar en nuestro archivo manifiesto AndroidManifest.xml



Es una buena práctica de programación mantener los recursos que no son propiamente nuestro código, definidos en archivos externos. Estos recursos pueden ser imágenes (Drawables), cadenas (String), colores, temas, plantillas (layouts), animaciones, menús, entre algunos elementos más.

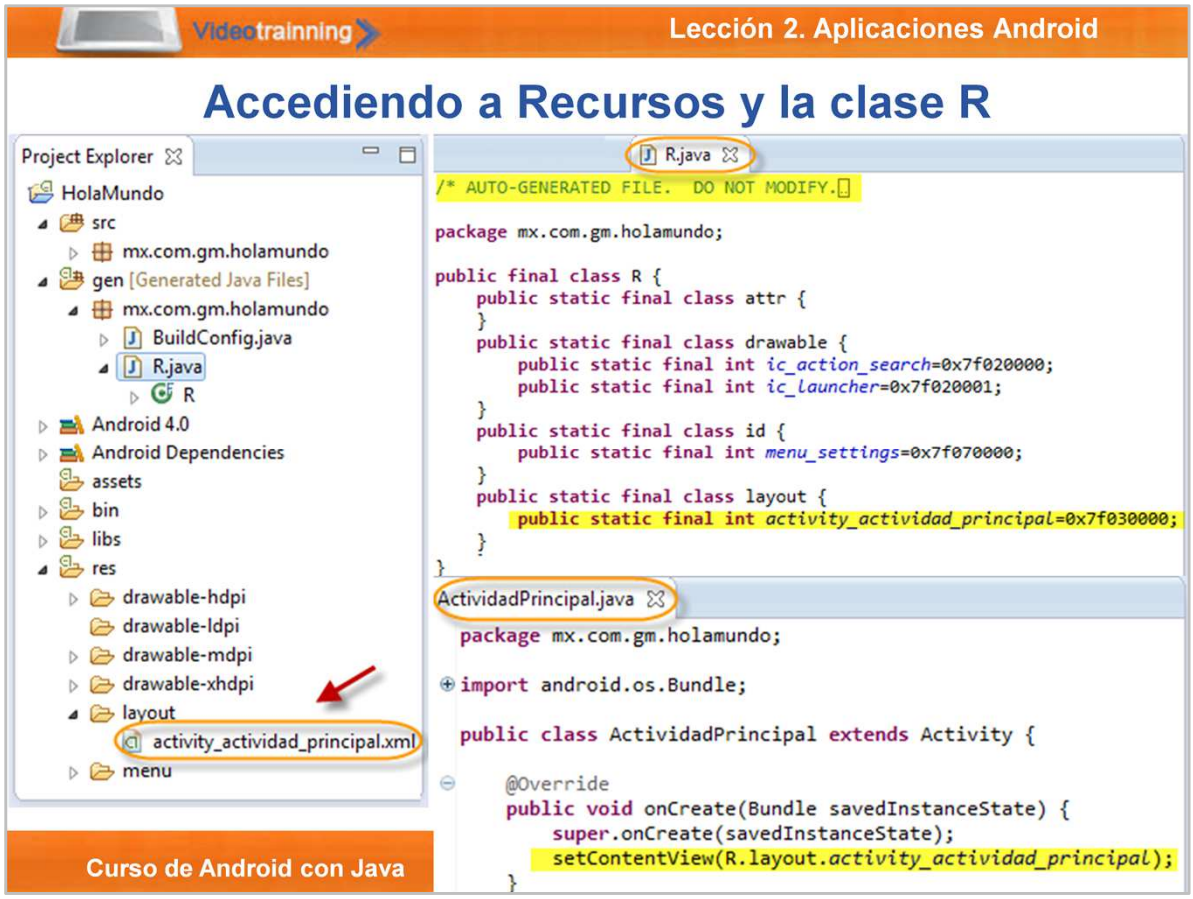
Al externalizar estos recursos, es más simple gestionar estos recursos. Además, es posible definir recursos alternos dependiendo de varias configuraciones específicas, tales como tipo de dispositivo, tamaño de pantalla y resolución, pantalla en horizontal o vertical, manejo de distintos idiomas, entre otras variaciones.

Cuando la aplicación se inicia, Android selecciona de manera automático el recurso más adecuado, sin necesidad de escribir una sola línea de código para ello.

Como podemos observar en la figura, los recursos se encuentran en la carpeta de ***res.*** Esta carpeta contiene a su vez subfolders, los cuales contiene recursos tales como imágnes (drawable), y esta carpeta a su vez contiene variantes según el tipo de dispositivo que estemos utilizando. drawable-ldpi significa es para dispositivos con baja densidad, mdpi para mediana densidad, hdpi para alta densidad y xhdpi para extra alta densidad. Este concepto lo estudiaremos más adelante.

También tenemos carpetas como layout para almacenar las plantillas de nuestras actividades, menus, la carpeta de values para agregar las cadenas que utilizaremos en nuestra aplicación. También dependiendo de la versión del dispositivo (level API) podemos especificar características en particular para los estilos (styles.xml).

Para más información de tipos de recursos consultar:
http://developer.android.com/guide/topics/resources/available-resources.html



Además de los recursos que nosotros podemos definir, Android define varios recursos del sistema que podemos utilizar en nuestras aplicaciones.

Android permite acceder a los recursos definidos desde las clases Java. Para ello crea la clase R. Esta es una clase estática generada de manera automática al momento de compilar nuestro proyecto y su contenido está basado en los recursos externos definidos en nuestro proyecto.

La clase R contiene subclases estáticas para cada uno de los tipos de recursos que hemos definido. Por ejemplo, si hemos agregado imágenes o cadenas, tenemos acceso en automático a las subclases R.drawable y R.string respectivamente.

Si utilizamos el plug-in ADT de Eclipse, la clase R se crea y actualiza de manera automática cada que agregamos o modificamos nuestros recursos. Cada subclases de la clase R, expone las variables con el nombre de cada identificador, ej. R.string.nombre_aplicacion o R.drawable.icon. El valor de estas variables es un integer que representa la ubicación del recurso, no es el valor en sí. Esto lo podemos observar en el código de la clase R. Por lo mencionado, no debemos modificar manualmente esta clase, ya que perderemos cualquier cambio debido a que se actualiza en automático por le IDE.

Por ejemplo, en nuestro código Java podemos especificar cual es la plantilla (layout) que queremos utilizar.

```
setContentView(R.layout.activity_actividad_principal);
```

Lo anterior, establece que utilizaremos el layout activity_actividad_principal, el cual se ubica en los recursos de nuestro proyecto. Este es sólo un ejemplo de cómo podremos utilizar los recursos definidos en nuestro proyecto y accederlo desde nuestro código Java. A lo largo del curso pondremos en práctica varios de los tipos de recursos según avancemos en los temas.

Según hemos comentado, una actividad representa una pantalla de nuestra aplicación, con la cual interactúa el usuario. Cada actividad puede iniciar otra actividad con el fin de realizar diferentes acciones.

Cada vez que se comienza una nueva actividad, la actividad anterior se detiene, pero el sistema mantiene la actividad en una pila. Cuando se inicia una nueva actividad, se inserta en la parte superior de la pila y se muestra al usuario. La pila sigue el mecanismo de colas llamado LIFO (Last-in-fist-out) "el último en entrar, es el primero en salir", por lo que, si el usuario ha terminado con la actividad actual y presiona el botón Back, se extrae de la pila, se destruye dicha actividad y se reanuda la actividad anterior.

Cuando una actividad se detiene debido a que una nueva actividad se inicia, este cambio es notificado a la actividad a través de los métodos callback de ciclo de vida de la actividad. Hay varios métodos callback que una actividad puede recibir debido a un cambio en su estado, ya sea que el sistema Android la ha creado, detenido, reanudado o destruido. Cada uno de estos métodos nos permite realizar una tarea en específico, según sea necesario, para cada uno de los estados del ciclo de vida, los cuales son:

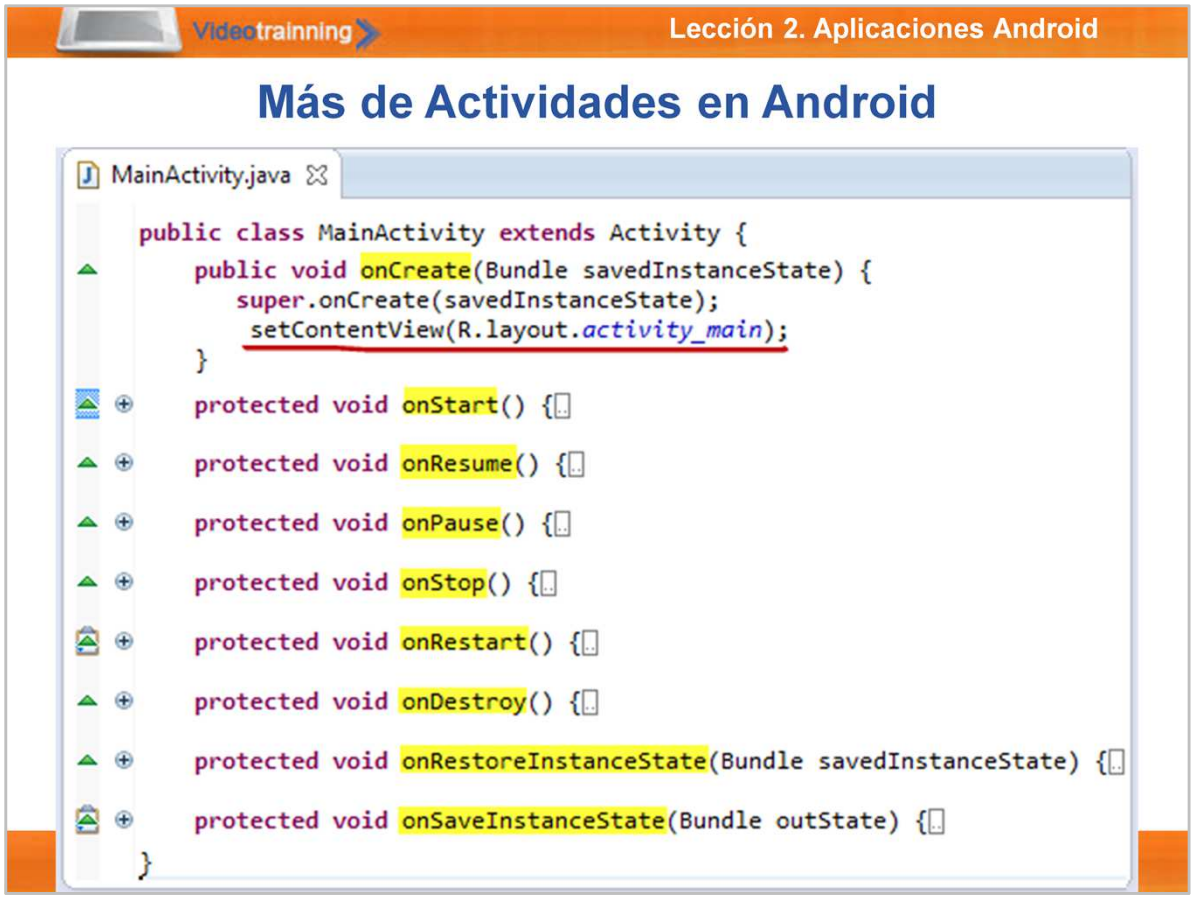
- **onCreate**: Este método se utiliza para iniciar nuestra actividad, por ejemplo, desplegar (inflate) nuestra vista (layout).
- **onStart**: Es el método que se ejecuta al iniciar una actividad. Se puede utilizar para iniciar recursos, tales como conexiones de red o base de datos.
- **onResume**(): Este médodo se utiliza para reanudad cualquier actividad que hubiese estado pendiente. Se llama después del método onStart.
- **onPause**: Se manda a llamar cuando la actividad no está visible.
- **onStop**: Este método se puede utilizar para detener cualquier recurso, como puede ser una animación, hilos, listener de sensores, gps, etc.
- **onRestart**(): Este método se ejecuta al regresar a la actividad una vez que estuvo en pausa, y es visible nuevamente.
- **onDestroy**: Se utiliza para limpiar y/o desconectar cualquier recurso , tales como conexiones de red o base de datos.

Por ejemplo, si se detiene la actividad libera todos los objetos grandes, por ejemplo conexiones de red o base de datos. Cuando la actividad se reanuda, podemos recuperar nuevamente recursos necesarios y reanudar las acciones que fueron interrumpidos. Estas transiciones de estado son parte del ciclo de vida de la actividad.

Un ejemplo de las llamadas del ciclo de vida sobre una actividad en específico es el siguiente:

- Se inicia la actividad: onCreate(), onStart() y onResume()
- Se llama otra aplicación o actividad: onPause() y onStop, además se liberan objetos
- Presionamos el botón Back: onRestart(), onStart() y onResume()
- Cambiamos a modo landscape: onPause(), onStop, onDestroy(), onCreate(), onStart() y onResume()
- Nos salimos de la aplicación con botón Back: onPause(), onStop y onDestroy()

En la figura podemos observar el ciclo de vida y los métodos callbacks asociados a una actividad. En el siguiente ejercicios crearemos el proyecto CicloVidadActividad para entender a más detalle cómo funcionan estos métodos y en qué momento son ejecutados.



La clase Activity representa una pantalla vacía, la cual encapsula el manejo de la ventana a utilizar.

Podemos observar en la figura, el método onCreate(), el cual dentro de su código contiene: `super.onCreate(savedInstanceState)`. Este método recibe un objeto de tipo Bundle, el cual se utiliza para restaurar el estado de la vista, si es que hubiera algo previo que recuperar de nuestra vista. Por ejemplo cuando se cambia la vista de orientación de vertical (portrait) a horizontal (landscape), la actividad se destruye y se vuelve a construir, sin embargo con ayuda del objeto Bundle, es posible recuperar el estado de la vista con cualquier información previa.

La siguiente línea de código:
`setContentView(R.layout.activity_actividad_principal)` establece la vista (layout) a mostrar en la actividad (pantalla) creada. La vista se puede crear ya sea con el archivo xml en la carpeta de layout, o con código Java, agregando elemento a elemento en la actividad. Sin embargo, por simplicidad se recomienda utilizar el archivo xml para la creación de layout, debido a las ayudas gráficas que tenemos, y así se simplifica el mantenimiento de las vistas.

Por otro lado podemos observar los métodos del ciclo de vida de una actividad, y estos nos pueden servir para realizar ciertas tareas al momento de ejecutarse la llamada al método respectivo (callback methods).

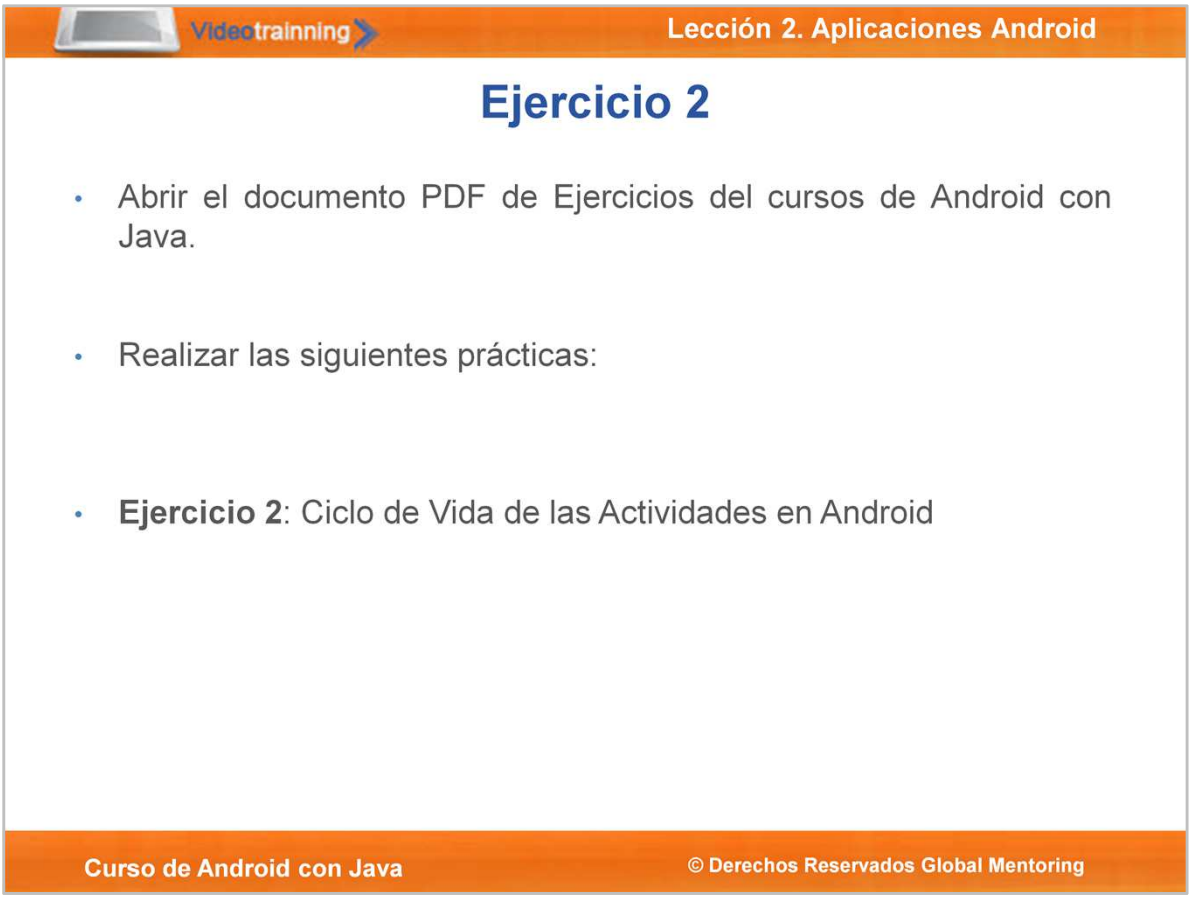
Para utilizar una Actividad en Android, es necesario agregarla al archivo de manifiesto. Además, si queremos que esta actividad sea la pantalla principal, debemos agregar el elemento `<intent-filter/>`, el cual indica al sistema Android que al iniciar nuestra aplicación, esta será la actividad que deberá ejecutarse.

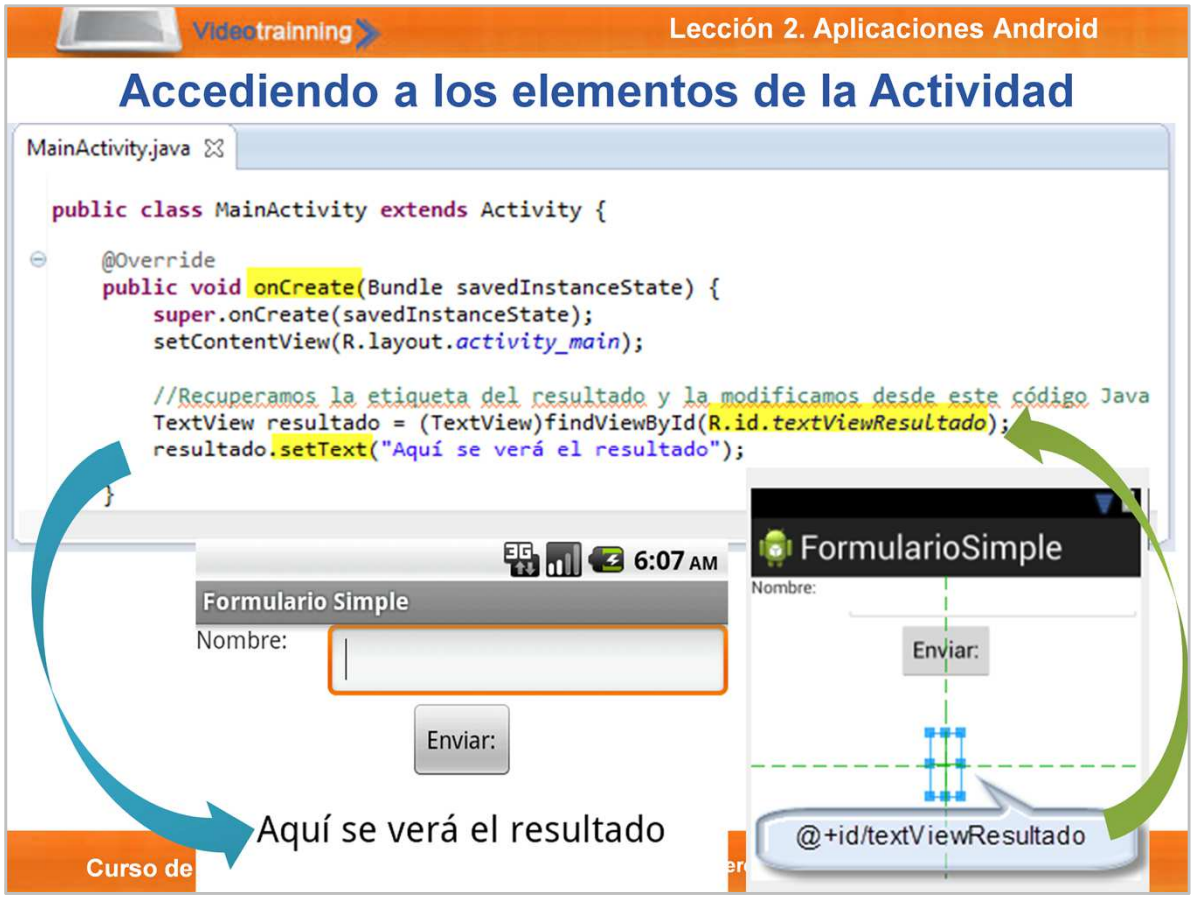
```
<activity
    android:name=".ActividadPrincipal">

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

En el nombre de la actividad es necesario especificar el nombre completo de la clase de la Actividad, incluyendo el nombre del paquete donde está contenido. Sin embargo, si al inicio del archivo de manifiesto especificamos el paquete donde se encuentras las actividades, es posible únicamente agregar un punto y el nombre de la actividad, y Android complementará el nombre del paquete a la clase de manera automática. Por ejemplo:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mx.com.gm.ciclovidaactividad >
```

El manejo a detalle de ***Vistas*** e ***Intents*** lo estudiaremos en lecciones posteriores.





Similar al manejo de recursos, para acceder a los elementos de nuestra interfaz gráfica (layout), utilizaremos la clase R.

Cada uno de los elementos en nuestro layout debe tener un nombre, por ejemplo, la etiqueta (label) donde se muestra el resultado tiene como nombre: textViewResultado. Sin embargo, para que sea accesible desde la clase R, es necesario indicar a qué subclase se agregará, y además si es un elemento nuevo o no.

Para ello se utiliza la sintaxis `@+id` , con la cual indicamos que este elemento se debe agregar (símbolo +) a la clase R en la subclase de id. Esto se agrega de manera automática, por lo que únicamente es necesario guardar los cambios y habremos agregado esta nueva variable a la clase R.

Una vez que hemos configurado nuestro elemento en la interfaz gráfica, podemos acceder a este elemento desde el código Java, con la sintaxis:

R.id.textViewResultado, con lo que indicando que estamos buscando en la subclase de id el componente textViewResultado.

Para recuperar este elemento desde el código Java, utilizaremos el método findViewById(), el cual se encuentra definido en la clase Activity. Recordemos que nuestra clase MainActivity extiende de la clase Activity.

El método findViewById() regresa un tipo Object, por ello es necesario hacer un cast al tipo definido en el layout. En este caso el tipo es TextView, por ello agregamos la conversión antes de asignar esta variable.

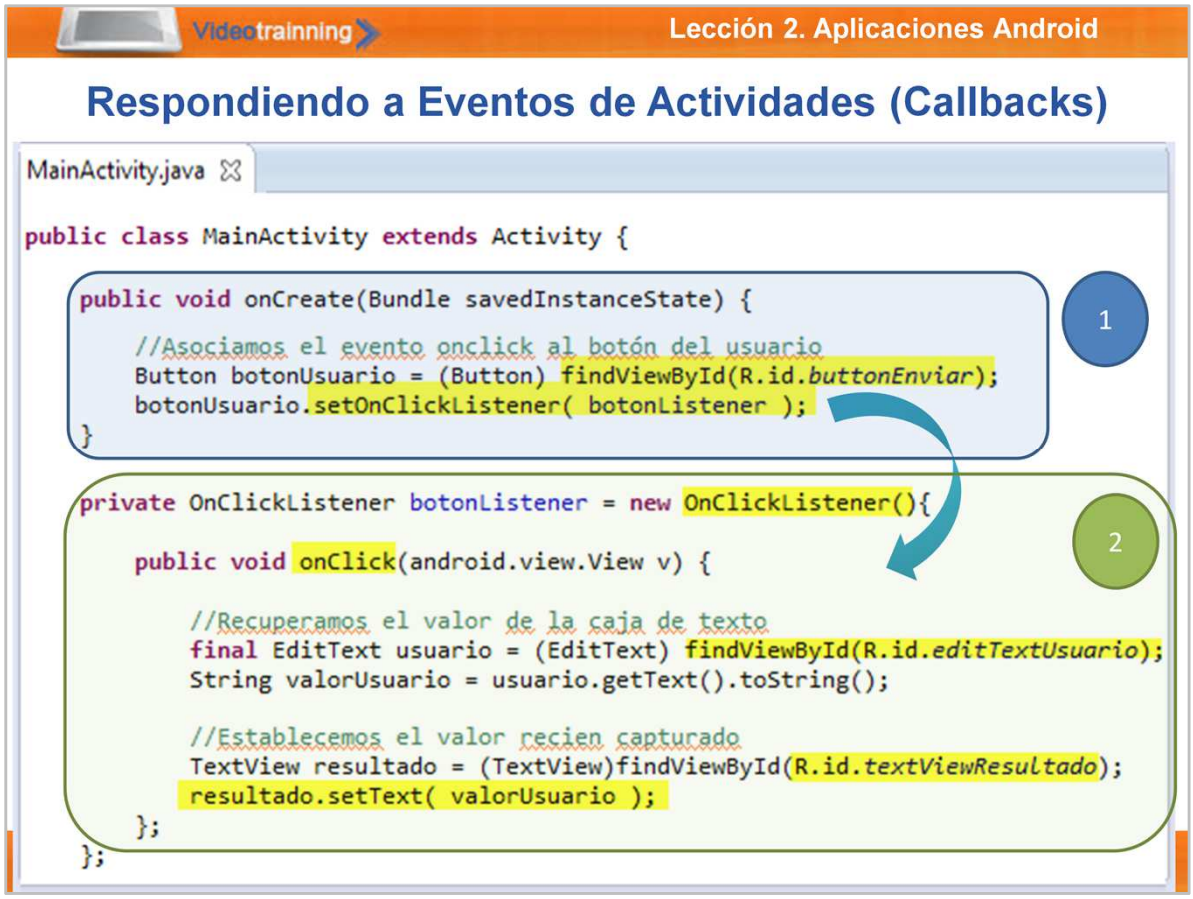
```
TextView resultado = (TextView)findViewById(R.id.textViewResultado);
```

Una vez que tenemos la referencia del textView, podemos utilizar el método setText() para modificar lo que se muestra en este componente.

```
resultado.setText("Aquí se verá el resultado");
```

La primera vez que mostramos este formulario, estamos accediendo a este componente que no tenía ningún texto relacionado, y lo modificamos para que se visualice: ***Aquí se verá el resultado***

Este proceso es el básico para recuperar la referencia de cualquier elemento de nuestra interfaz gráfica, y modificar sus valores. En la siguiente lámina veremos como procesar eventos asociados a un botón.



Android permite responder a eventos ejecutados desde la interfaz gráfica. Por ejemplo al presionar un botón, podemos responder a este evento.

La forma en que Android responde a estos eventos se conoce como métodos callback. Estos métodos se mandan a llamar una vez que se produce el evento, por ejemplo, el evento onClick. Para poder responder a estos eventos, debemos asociar un método listener, los cuales son métodos que están "escuchando" en todo momento si se ha ejecutado el evento asociado.

En la primera parte del código observamos que recuperamos la referencia a un botón, y posteriormente asociamos a través del método setOnClickListener() una clase que implemente la interfaz OnClickListener. Existen varias formas de resolver este requerimiento. Es posible definir una clase anónima y esta clase debe implementar el método onClick(). También es posible asociar la implementación de esta interfaz a una variable, de tal manera que podamos reutilizar este código en caso que sea necesario.

```
Button botonUsuario = (Button) findViewById(R.id.buttonEnviar);
botonUsuario.setOnClickListener( botonListener );
```

En la segunda parte del código, hemos definido la implementación de la interfaz OnClickListener como una implementación anónima. Esta clase contiene la implementación del método onClick(), el cual nos permite procesar dicho evento asociado al botón del layout.

```
private OnClickListener botonListener = new OnClickListener(){

  public void onClick(android.view.View v) {…};
};
```

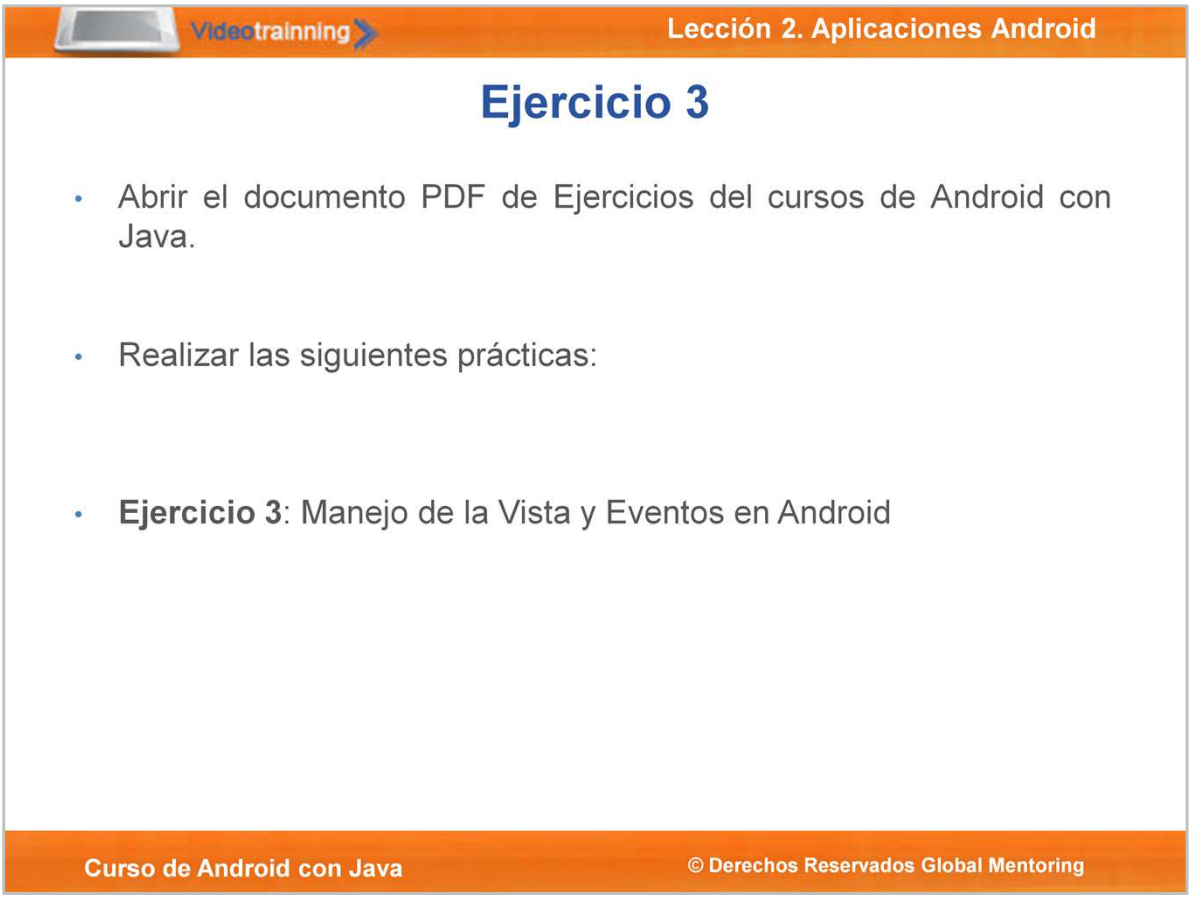
Como vemos en la figura, este código procesa el valor de la caja de texto llamado editTextUsuario. Sin embargo, para poder recuperar la cadena asociada a esta caja de texto, es necesario utilizar la siguiente sintaxis: cajaTexto.getText().toString(). Esto es necesario ya que el método getText() no regresa la cadena, sino un CharSequece. Así que para recuperar la cadena es necesario llamar al método toString(). Esta variable debe ser final, ya que es una condición que se debe cumplir en una clase anónima, según la especificación de Java.

```
final EditText usuario = (EditText) findViewById(R.id.editTextUsuario);
String valorUsuario = usuario.getText().toString();
```

Una vez que ya hemos procesado el evento y recuperado la cadena que el usuario capturó, es posible mostrar el resultado en el objeto textViewResultado utilizando el método setText().

```
TextView resultado = (TextView)findViewById(R.id.textViewResultado);
resultado.setText( valorUsuario );
```

Este es el proceso general para procesar eventos desde la interfce UI y el código Java asociado. Para más información del manejo de eventos en Android pueden consultar: http://developer.android.com/guide/topics/ui/ui-events.html



En Global Mentoring promovemos la *Pasión por la Tecnología Java*.

Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados.

Además agregamos nuevos cursos para que continúes con tu preparación como consultor Java de manera profesional.

A continuación te presentamos nuestro listado de cursos en constante crecimiento:

- Fundamentos de Java
- Programación con Java
- Java con JDBC
- HTML, CSS y JavaScript
- Servlets y JSP's
- Struts Framework
- Hibernate Framework
- Spring Framework
- JavaServer Faces
- Java EE (EJB, JPA y Web Services)
- JBoss Administration
- Android con Java

Datos de Contacto:

Sitio Web: www.globalmentoring.com.mx

Email: informes@globalmentoring.com.mx

Ayuda en Vivo: www.globalmentoring.com.mx/chat.html