Manejo de eventos

El manejo de eventos es básicamente escribir la lógica del programa en respuesta a una acción del usuario. Esta lógica del programa se escribe generalmente como parte de los métodos de Java.
Hay dos formas de manejar eventos: puedes hacerlo de manera declarativa o programática. En este documento se exploran ambas formas.

Descripción general del manejo de eventos

El proceso de manejo de eventos declarativamente se puede desglosar de la siguiente manera.

1. Define el objeto de vista (por ejemplo, una vista de Botón). Esto se puede hacer en modo de diseño o texto.

```
<Button
  android:id="@+id/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginStart="27dp"
  android:layout_marginTop="141dp"
  android:onClick="sayHello"
  android:tag="mybutton"
  android:text="Button"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent"
  tools:layout_editor_absoluteX="27dp"
  tools:layout_editor_absoluteY="141dp"/>
```

2. Elige a qué evento deseas que responda el programa: por ejemplo, un evento de clic (algunos objetos de vista pueden responder al rango de eventos: haz clic con el botón largo, desliza el dedo, y así sucesivamente).
3. Asocia el evento con un método de Java: por ejemplo, agrega el atributo onClick en el elemento XML de vista de botón.
   android: onClick = "sayHello"
   Alternativamente, puedes asociar un objeto de vista a un método en el inspector de atributos.
4. Implementar el método de Java en el archivo de programa principal (MainActivity.java). El nombre del método debe ser el mismo que el definido en el inspector de atributos.

```
void sayHello(View v) {
   System.out.println("Hello");
}
```

Para manejar eventos mediante programación, los pasos se pueden desglosar de la siguiente manera.

1. Define el objeto de vista (por ejemplo, una vista de Botón).
2. Dentro del programa principal, declara una variable para contener un objeto de vista de Botón
   Button objButton;
3. Obtén una referencia programática al objeto de vista de botón definido en el archivo de diseño.
   objButton = (Button) findViewById (R.layout.button)
4. Decide a qué evento deseas responder y especifica el objeto detector para él (por ejemplo, haz clic en)
   objButton.setOnClickListener (new View.OnClickListener () {});
5. Anula el método onClick: ahora puedes implementar la lógica del programa aquí. Haz lo que quieras hacer en respuesta a la acción del usuario.
   void onClick (View v) {
   // haz algo aquí
   }

Veremos ambas técnicas un poco más cerca en las siguientes secciones.

Gestión de eventos declarativa

1. Abre el proyecto Hello (el de la primera práctica!) de las últimas notas, si aún no está abierto. Abre el archivo de diseño (activity_main) desde la ventana de la herramienta del proyecto y visualízalo en modo de diseño.
2. Selecciona la vista del Botón.
3. Mientras se selecciona el Botón, ve al inspector de Atributos y encuentra el atributo onClick. Escribe el texto sayHello como se muestra en la figura 8.1.

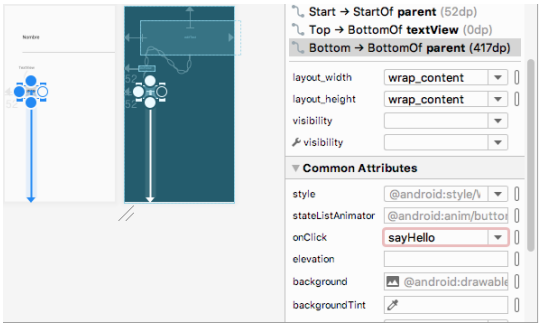


Figura 8.1: Atributo onClick.

```
<Button
  android:id="@+id/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginStart="27dp"
  android:layout_marginTop="141dp"
  android:onClick="sayHello" ❶
  android:text="Button"
  app:layout_constraintStart_toStartOf="parent"
  app:layout_constraintTop_toTopOf="parent"
  tools:layout_editor_absoluteX="27dp"
  tools:layout_editor_absoluteY="141dp"/>
```

Listado 8.1: app/src/main/res/layout/activity_main.xml.

(1) Esto fue agregado automáticamente por AS4 cuando escribiste sayHello en el atributo onClick. La actualización funciona en ambos sentidos: podrías haber editado este XML directamente y la actualización se habría reflejado en el inspector de atributos. Cuando ejecutas esta aplicación, el tiempo de ejecución de Android buscará el método sayHello en MainActivity.java; por el momento, aún no lo tenemos, pero implementaremos ese método dentro de un tiempo.

Completar la entrada del atributo onClick significa que deseas que ocurra algo cuando se hace clic en la vista de Botón; la entrada de atributo se convierte en el nombre de un método que buscará el tiempo de ejecución de Android cuando ocurra un clic en el botón.

Para implementar el método sayHello, vamos a abrir MainActivity.java en el editor principal y escribir nuestro método.

```
void sayHello(View v) {
   System.out.println("Hello");
}
```

Listado 8.2: Método sayHello.

Sugerencia. Mientras escribes el código, es posible que algunas palabras clave no se reconozcan. Probablemente sea porque aún no has importado las clases adecuadas. AS4 tiene una función de corrección rápida (Alt + Enter u Option + Intro).

Si pasas el mouse sobre la palabra clave no reconocida, AS4 importará automáticamente las clases necesarias para ti.

El método es simple, pero hay un par de cosas que deben señalarse (Tabla 8.1).

Item	Artefacto de código	Comentario
Return type	void	No es necesario que le devolvamos nada a la persona que llama, por lo que se declarará nulo.
Method name	sayHello()	Esto debe ser el mismo que está escrito en el atributo onClick del botón.
Arguments or parameters	View object	Cada controlador de eventos necesita aceptar un argumento de objeto de Vista. Este argumento es completado por el tiempo de ejecución de Android. Si necesitas saber en qué objeto de Vista se hizo clic, puedes usar este parámetro.

Tabla 8.1: Partes del método sayHello.

Ejecuta la aplicación en un AVD para que podamos ver cómo se comporta y podamos probar cómo funciona nuestro controlador de eventos.

Una vez que el AVD esté funcionando, abrimos la ventana de la herramienta Logcat para que podamos ver un volcado de la consola de todos los eventos en el emulador.

Para iniciar la ventana de la herramienta Logcat, haz clic en el iniciador "Logcat"; está ubicado en algún lugar en la parte inferior izquierda de la ventana de la aplicación AS4, como se muestra en la figura 8.2.

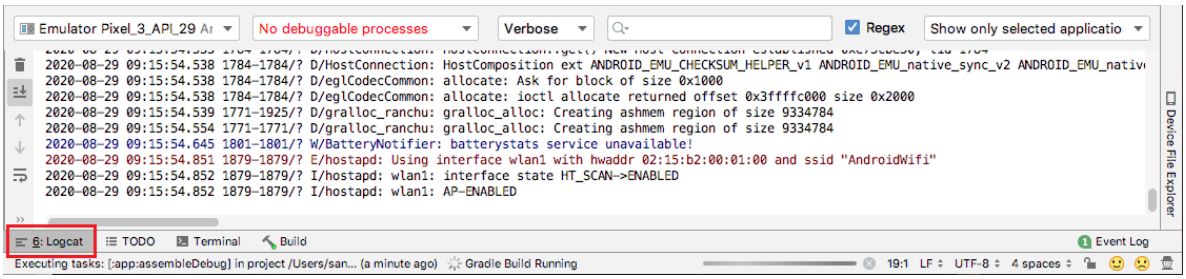


Figura 8.2: Ventana Logcat.

Ve al emulador y haz clic en el botón en nuestra aplicación. Mientras lo haces, trata de ver la ventana de Logcat (figura 8.3).

La salida de System.out no estará en la actividad; en su lugar, se redirige a la ventana Logcat de la consola.

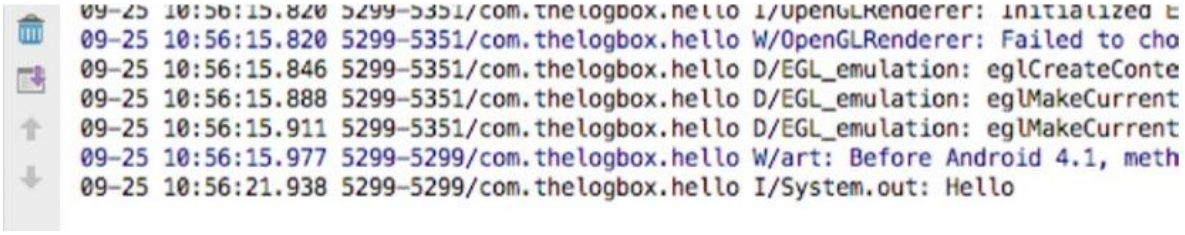


Figura 8.3: System.out en Logcat.

Manejo programático de eventos

Podría ser mejor crear un nuevo proyecto para este ejercicio, para que puedas mantener el último proyecto como referencia. Crea un nuevo proyecto con la información en la Tabla 8.2.

Application name	**Hello2**
Company domain	Usa su sitio web o inventa algo; recuerda que esto está en la notación DNS inversa.
Project location	Esto generalmente es mejor dejarlo solo. Usa el valor predeterminado, pero asegúrate de tomar nota de esta ubicación en caso de que surja la necesidad de acceder a ella. Ignora el soporte de C ++ y Kotlin.
Form factor	Solo teléfono y tableta.
Mínimum SDK	API 26 Oreo.
Type of activity	Vacío.
Activity name	Si dejas el valor predeterminado solo, será MainActivity, lo cual está bien.
Layout name	Si dejas el valor predeterminado solo, este será activity_main, que está bien.

Tabla 8.2: Información del proyecto para Hello2.

Lo mismo que para el proyecto Hello de la última sección, coloca los objetos editText, textView y Button view en el archivo de diseño y simplemente usa las “inferir restricciones” para poner algo de apariencia de diseño en tu proyecto.

Tu archivo de diseño debe parecerse al listado 8.3.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="372dp"
        android:layout_height="129dp"
        android:layout_marginStart="62dp"
        android:layout_marginTop="81dp"
        android:layout_marginEnd="62dp"
        android:layout_marginBottom="68dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="Nombre"
        app:layout_constraintBottom_toTopOf="@+id/textView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="52dp"
        android:layout_marginBottom="83dp"
        android:text="TextView"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="160dp"
        android:layout_marginBottom="417dp"
        android:onClick="sayHello"
        android:tag="mybutton"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 8.3: app/src/main/res/layout/activity_main.xml

Cuando manejes eventos programáticamente, trabajarás casi exclusivamente en el archivo principal del programa. Abre MainActivity.java si aún no está abierto, de modo que podamos agregar algunos códigos de manejo de eventos.

```
public class MainActivity extends AppCompatActivity {

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

    // Write your event handling codes below this line
  }
}
```

Listado 8.4: Archivo de programa principal.

Añadiremos todo nuestro código de manejo de eventos justo después del método setContentView; necesitamos completar esta llamada del método para que todos los objetos en el archivo de diseño finalicen el proceso de inflación.

El proceso de inflación crea todos los objetos View y ViewGroup como objetos Java que luego podemos referenciar programáticamente en el programa principal. El ejercicio completo se puede resumir de la siguiente manera.

1. Obtener una referencia programática a la vista de Botón.
2. Establecer un objeto de Escucha (Listener) para esta.
3. Anular el método abstracto definido en el objeto listener y proporcionar lógica de programa en respuesta a una acción del usuario

Obtener una referencia programática a la vista del botón se puede hacer con la siguiente declaración.

```
Button objButton = (Button) findViewById(R.id.button);
```

Puedes observar que a medida que escribes la declaración anterior, AS4 intenta inferir y ofrecer algunas opciones de autocompletado. El autocompletado puede ahorrarte tiempo en la programación, y te da cierta confianza de que estás en el camino correcto; si las cosas no aparecen en el autocompletado, generalmente significa que estás haciendo algo mal porque AS4 no lo reconoce.

En la figura 8.4, cuando se escribía la palabra “Botón”, había un par de entradas en las solicitudes de autocompletado. Cada vez que aparece este mensaje, puedes usar las teclas de flecha para elegir cualquiera de las entradas. Si presionas la tecla Intro, cualquiera que sea la entrada resaltada en la ventana emergente de autocompletado se pondrá en lugar de la posición actual del cursor.

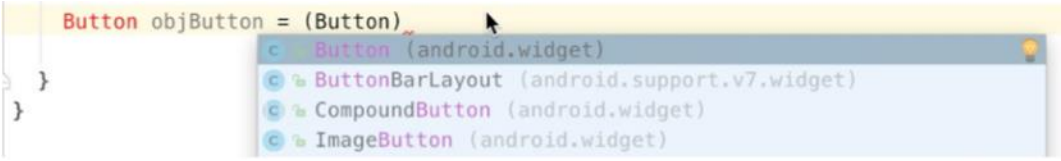


Figura 8.4: Autocompletado.

Otra cosa que puedes haber notado es que todas las instancias de la palabra Botón están en rojo; si colocas el mouse sobre “Botón”, AS4 te informará a través del cuadro de diálogo de herramientas que “no puede resolver el botón del símbolo” (consulta la figura 8.5). Este error está apareciendo y AS4 nos recuerda algo al respecto porque necesitamos importar la definición de Button en el archivo fuente actual, y aún no lo hemos hecho.

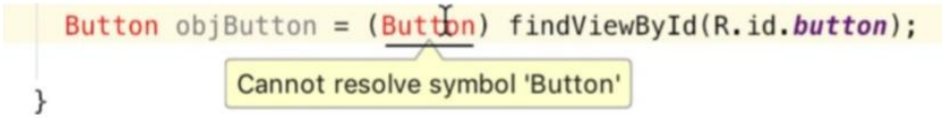


Figura 8.5: Advertencias y errores de AS3.

Para resolver el error, debemos importar android.view.Button en el programa principal escribiendo la instrucción de importación en cualquier lugar del archivo fuente antes de la declaración de clase para MainActivity. Ve el siguiente código de ejemplo.

```
import android.view.Button;
public class MainActivity extends AppCompatActivity {
}
```

Alternativamente, puedes pasar el mouse sobre “Botón” como se muestra en la figura 8.5, y luego realizar una solución rápida escribiendo ⌥ (Opción) + ↵ (Intro) si estás en macOS. Si estás en Windows o Linux, la clave de solución rápida es Alt + ↵ (Enter).

Las soluciones rápidas resuelven una variedad de problemas de codificación; las declaraciones de importación faltantes es uno de ellos. Después de la solución rápida, la declaración de importación para la vista del botón aparecerá en la parte superior del archivo fuente, junto con las otras declaraciones de importación.

Lo siguiente que necesitamos es seleccionar un oyente (listener) para la vista de Botón, crear un objeto oyente para él y, finalmente, anular el método abstracto necesario definido por el objeto oyente. Puedes usar la función de autocompletado mientras escribes esta construcción.

Ten en cuenta que a medida que escribes los códigos (figura 8.6), que crearán el objeto detector, hay varias coincidencias posibles para él. Lo que queremos en este caso es la segunda entrada desde la parte superior, la que tiene los puntos suspensivos encerrados en un par de llaves. Esto es en realidad un fragmento de código. Si elegimos esto, lo que obtendremos es el código en el listado 8.5.

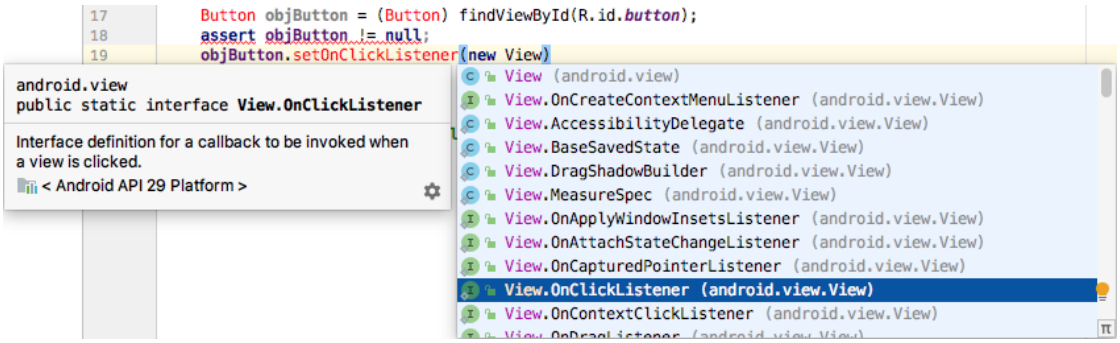


Figura 8.6: Autocompletado para el objeto oyente.

```
objButton.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {

  }
});
```

Listado 8.5: Fragmento de código para View.OnClickListener.

Después de esto, todo lo que tenemos que hacer es escribir nuestra lógica de programa dentro del cuerpo del método onClick().

El listado 8.6 es el listado completo del código para el programa principal.

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

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

    Button objButton = (Button) findViewById(R.id.button); ❶
    assert objButton != null; ❷
    objButton.setOnClickListener(new View.OnClickListener() { ❸
      @Override ❹
      public void onClick(View view) { ❺
        System.out.println("Hello World");
      }
    });
  }
}
```

Listado 8.6: MainActivity.java.

❶ Obtiene una referencia programática de nuestro objeto Button. findViewById localizará el objeto de vista de botón exacto para nosotros. R.id.button es la llave para encontrar este objeto. Cuando el tiempo de ejecución de Android infla el archivo de diseño, genera una clase llamada R.java; todas las representaciones de objetos Java de todos los objetos de vista definidos en el archivo de diseño se definirán allí.
Estamos utilizando la R.class (Recursos) para ubicar nuestro botón mediante programación.

❷ Esto es solo una programación defensiva; estamos intentando asegurarnos de que el método findViewById devuelva un objeto real y que no sea nulo. Si alguna vez has encontrado un NullPointerException en Java anteriormente, eso puede suceder aquí. Si

findViewById no devuelve un objeto, tu programa se bloqueará durante el tiempo de ejecución.

❸ Queremos responder a una acción de clic; es por eso que el método que elegimos es setOnClickListener. Este método requiere una instancia de un objeto detector como argumento. Implementamos el objeto detector en línea y como una clase anónima; consulta la sección “El lenguaje Java” en las clases anónimas que más adelante se detallarán.

❹ @Override es una anotación; le dice al compilador que tenemos la intención de anular un método en la superclase AppCompatActivity y que no estamos definiendo un nuevo método onCreate en MainActivity. La anotación simplemente aclara nuestra intención para el compilador.

❺ onClick es un método abstracto definido por la interfaz View.OnClickListener. Debe ser anulado para nuestra implementación

Ejecuta el programa en el emulador y abre la ventana de la herramienta Logcat para que puedas ver los volcados de la consola al hacer clic en la vista del botón.

Sugerencia: Cuando hayas iniciado el emulador y le hayas implementado la aplicación, puedes usar el botón “Aplicar cambios” para actualizar rápidamente el emulador sin crear una nueva APK y empujarla al emulador (figura 8.7). Esto permite un flujo de trabajo más rápido cuando realizan pequeños cambios de código o diseño.

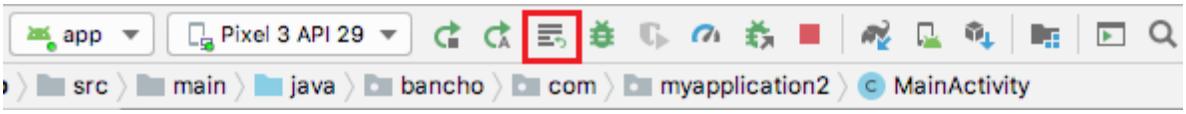


Figura 8.7: Aplicar el botón de cambios.

Trabajar con texto y botones

Entre los elementos de la interfaz de usuario en Android, los elementos de texto y botón son probablemente los más comunes.

En esta sección, profundizaremos en una pequeña aplicación de muestra que nos dará la oportunidad de trabajar con estos dos elementos. Los detalles del proyecto se muestran en la Tabla 8.3.

La figura 8.8 muestra la interfaz de usuario para el proyecto.

Application name	**NumberGuess**
Project location	Deja el valor predeterminado.
Form factor	Solo teléfono y tableta.
Mínimum SDK	API 26 Oreo.
Type of activity	Vacío.
Activity name	MainActivity (predeterminado).
Layout name	activity_main (predeterminado).

Tabla 8.3: Información del proyecto.

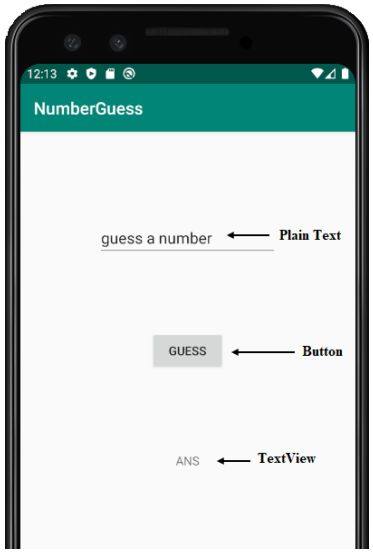


Figura 8.8: Ver elementos para NumberGuess.

Hubo algunos ajustes cosméticos y estéticos en los elementos para que parezcan un poco más agradables a los ojos.

```
<EditText
  android:id="@+id/editText"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginBottom="26dp"
  android:ems="10"
  android:gravity="center_vertical|center_horizontal" ❶
  android:hint="guess a number" ❷
  android:inputType="number" ❸
  app:layout_constraintBottom_toTopOf="@+id/button"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="parent"
  tools:layout_editor_absoluteX="84dp"
  tools:layout_editor_absoluteY="106dp"/>
```

❶ Esta entrada alinea el texto dentro de la vista de texto plano tanto vertical como horizontalmente.

❷ El atributo de sugerencia hace que el texto “adivinar un número” aparezca en gris; es una buena técnica para usar en lugar de una etiqueta real, como una vista de texto.

❸ Esto restringe la entrada a los números. Si no agregas este atributo, el usuario podrá escribir cualquier carácter alfanumérico en el campo; entonces, es posible que necesites manejar la validación de la entrada usando algunas técnicas de expresión regulares. Este enfoque es mucho más fácil.

Nota: Todos los ajustes anteriores también se pueden hacer en modo de diseño. Puedes establecer estos valores en el inspector de atributos.

Todavía estamos usando un diseño de restricción, por lo que debes tener cuidado de poner restricciones en cada uno de los elementos de la interfaz de usuario. Al igual que en los proyectos anteriores, esto se puede gestionar sin mucha dificultad colocando todos los elementos de la interfaz de usuario a mano, colocándolos en la ubicación aproximada donde deseas que aparezcan, y luego usando las herramientas en el inspector de restricciones.

Usa la herramienta “paquete”" para distribuirlos horizontalmente, y luego usa las “"limitaciones de inferir”. Eso debería encargarse del diseño. El flujo básico de esta aplicación es el siguiente:

1. Cuando se inicia la aplicación, se generará un número aleatorio de 100 a 150.
2. El usuario adivinará qué número se generó ingresando ese número en el campo de texto.
3. Si el usuario elige un número que es más alto (en valor) que el número aleatorio, mostraremos “Adivinar más abajo” usando la vista de texto estática.
4. Si el usuario elige un número más bajo que el número aleatorio, mostraremos “Adivinar más arriba” usando la vista de texto estática.
5. Si el usuario adivinó el número correctamente, mostraremos el número aleatorio y una nota de felicitación en el campo de texto.

Puedes observar que el enfoque de manejo de eventos para este proyecto no utiliza una clase anónima o interna; no es que haya algo de malo en esos enfoques, pero hacer de MainActivity el objeto oyente proporciona cierta comodidad para esta situación. La lógica principal de obtener la entrada del usuario y compararla con el número aleatorio generado residirá en el método onClick.

Si este método estaba dentro de una clase anónima o interna, habría requerido que las variables que contenían EditText y TextView se declararan finales. Esa es solo una de las reglas de Java sobre las clases internas; está bien hacer referencia a cualquier variable en su clase externa, siempre que sea final. Y eso habría hecho que el código sea un poco más complicado que cómo está estructurado, como se muestra en el listado 8.7.

Las siguientes secciones muestran la lista de códigos de onCreate, onClick e initNumberToGuess, comenzando con el listado 8.8.

```
public class MainActivity extends AppCompatActivity
  implements View.OnClickListener {

  int numberToGuess = 0;
  EditText e;
  TextView t;

  @Override
  protected void onCreate(Bundle savedInstanceState) { ... }

  @Override
  public void onClick(View view) { ... }

  int initNumberToGuess() { ... }

}
```

Listado 8.7. Código de manejo de eventos de MainActivity (métodos plegados).

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

  numberToGuess = initNumberToGuess();          ❶

  e = (EditText) findViewById(R.id.editText);   ❷
  t = (TextView) findViewById(R.id.textView);   ❸

  Button b = (Button) findViewById(R.id.button);
  b.setOnClickListener(this);

}
```

Listado 8.8: onCreate.

❶ El número ToGuess se inicializa durante onCreate pero se declaró como una variable miembro y no como una variable local de onCreate. Necesitamos hacer referencia a esta variable desde el método onClick; esa es la razón por la que se declaró como una variable miembro.

❷ La variable e también se inicializa aquí, pero también se declara como una variable miembro; al igual que numberToGuess, necesitamos hacer referencia a esta variable desde el método onClick.

❸ El mismo caso que en la variable t; también necesitamos hacer referencia a esto desde el método onClick

```
@Override
public void onClick(View view) {
  int number = Integer.parseInt(e.getText().toString());      ❶
  if (number == numberToGuess) {
    t.setText(number + " is the right number");
  }
  else if (number < numberToGuess) {
    t.setText("Guess higher");
  }
  else if (number > numberToGuess) {
    t.setText("Guess lower");
  }
  Log.i("Ted", numberToGuess + "");
}
```

Listado 8.9: onClick.

❶ El método getText de EditText devuelve un tipo de objeto editable; es casi como texto, pero es mutable, a diferencia de una cadena. El Integer.parseInt, sin embargo, espera un parámetro String; es por eso que necesitábamos convertir el valor de retorno de getText usando el método toString.

```
int initNumberToGuess() {
  Random r = new Random();                    ❶
  numberToGuess = r.nextInt(100) + 50;        ❷
  Log.i("Ted", numberToGuess + "");
  return numberToGuess;
}
```

Listado 8.10: initNumberToGuess.

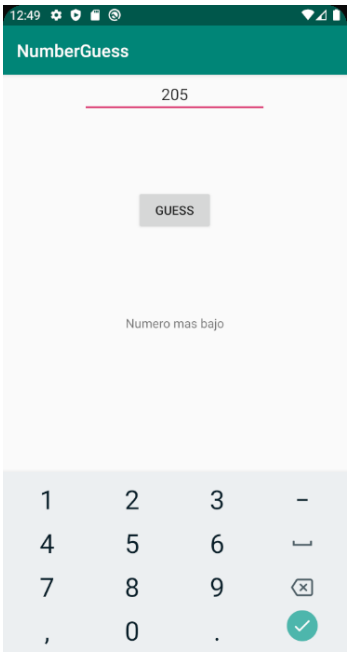
❶ La clase Random es de java.util. Asegúrate de importar este paquete. Alternativamente, cuando se ponga rojo en el editor principal, desplaza el mouse sobre él y usa la solución rápida (Alt + Intro para Windows y Linux | opt + enter para macOS).

❷ Esto establece el rango del número aleatorio de 100 a 150.

El listado 8.11 muestra el código completo para MainActivity, para su referencia.

```java
package bancho.com.numberguess;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import java.util.Random;

public class MainActivity extends AppCompatActivity
    implements View.OnClickListener {

        int numberToGuess = 0;
        EditText e;
        TextView t;

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

        numberToGuess = initNumberToGuess();

        e = (EditText) findViewById(R.id.editText);
        t = (TextView) findViewById(R.id.textView);

        Button b = (Button) findViewById(R.id.button);
        b.setOnClickListener(this);
    }

        @Override
        public void onClick (View view){
            int number = Integer.parseInt(e.getText().toString());
            if (number == numberToGuess){
               t.setText(number + "es el numero correcto");
            }
            else if (number < numberToGuess){
               t.setText("Numero mas alto");
            }
            else if (number > numberToGuess){
               t.setText("Numero mas bajo");
            }
            Log.i("Bancho", numberToGuess + "");
    }

    int initNumberToGuess(){
       Random r = new Random();
       numberToGuess = r.nextInt(100) + 50;
       Log.i("Bancho", numberToGuess + "");
       return numberToGuess;
    }

}
```

Listado 8.11: La MainActivity completa.



Más código de manejo de eventos

Usar clases anónimas para códigos de manejo de eventos debería resolver una amplia gama de desafíos de programación para ti, pero las clases anónimas no son la única forma de manejar eventos.

En esta sección, echaremos un vistazo a otras dos maneras de hacerlo. Para este ejercicio, crearemos un nuevo proyecto. Usa la información en la Tabla 8.4 para crear el proyecto.

Application name	**EventHandling1**
Company domain	Deja el valor predeterminado.
Project location	Deja el valor predeterminado.
Form factor	Solo teléfono y tableta.
Mínimum SDK	API 23 Marshmallow.
Type of activity	Vacío.
Activity name	MainActivity
Layout name	activity_main

Tabla 8.4: Información de proyecto.

Agrega tres vistas de botones al diseño. En este ejemplo, están alineados verticalmente entre sí y están obligados a permanecer en el centro de la pantalla. La forma más fácil de lograr este diseño es posicionar los botones a mano, aproximándote a la ubicación que deseas que sean en tiempo de ejecución.

A continuación, selecciónalas haciendo clic y arrastrando las vistas de los tres botones, y luego usa las herramientas en el inspector de restricciones para ajustar la alineación y las restricciones.

Puedes usar el botón “Paquete” (como se muestra en la figura 8.9) para distribuir las vistas verticalmente. Después de eso, usa las restricciones “Inferir” para alinear automáticamente las vistas entre sí y con el contenedor.

La Figura 8.10 muestra cómo se vería el diseño.

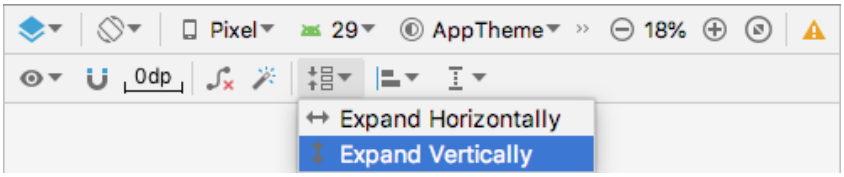


Figura 8.9: Empaquetar verticalmente.

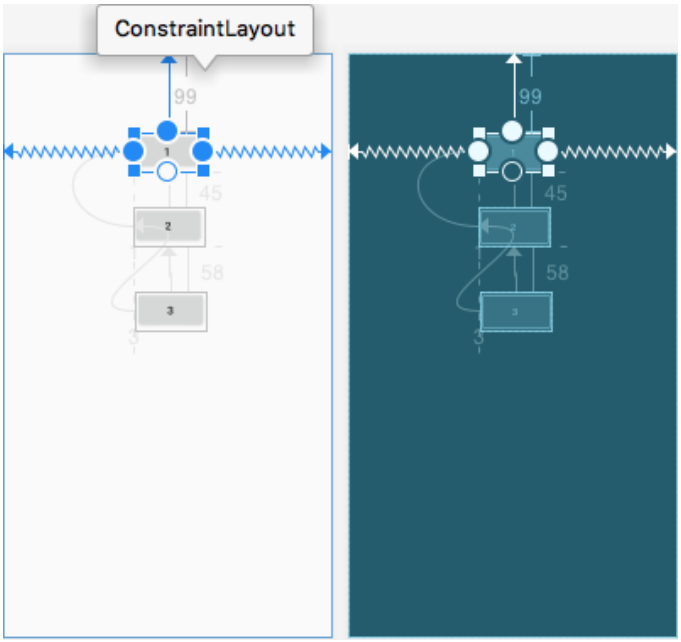


Figura 8.10: activity_main en la vista de diseño para EventHandling1.

Los ID de las vistas de los botones se han cambiado a button1, button2 y button3 en consecuencia, por lo que podemos consultarlos más adelante en el código con bastante facilidad.

Puedes usar otros nombres, por supuesto; los identificadores de vista son simplemente identificadores que tú, el programador, decidirá en última instancia. Pero para este proyecto, se nombran así.

El listado 8.6 muestra el código XML para nuestro diseño.

Uso de una clase interna como oyente

Podemos definir otra clase que está anidada dentro de MainActivity que puede servir como nuestro objeto oyente (listener) (Listado 8.13).

Java permite que las clases se aniden, por lo que vamos a aprovechar esto; hay algunas reglas que debemos observar al definir una clase interna, y las discutiremos cuando sean necesarias.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="48dp"
        android:text="1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:text="2"
        app:layout_constraintStart_toStartOf="@+id/button"
        app:layout_constraintTop_toBottomOf="@+id/button" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="12dp"
        android:text="3"
        app:layout_constraintStart_toStartOf="@+id/button2"
        app:layout_constraintTop_toBottomOf="@+id/button2" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Listado 8.12: activity_main.xml en Text View.

```
package bancho.com.eventhandling1;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import util.Log;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {...} 1

    private class ButtonHandler implements View.OnClickListener{ 2

    }
}
```

Listado 8.13: Una clase interna dentro de MainActivity.

❶ Esta línea muestra el método onCreate en un modo plegado. Las capacidades de plegado de código de AS3 son bastante útiles cuando trabajas con muchos códigos; borra el editor principal y te ayuda a enfocarte.

❷ Nuestra clase interna tal como se encuentra dentro de MainActivity. La definimos como privada porque no necesita ser visible desde fuera de MainActivity. Al igual que nuestra clase anónima en las secciones anteriores, esto también necesita implementar la interfaz View.OnClickListener porque la usaremos como el oyente para los clics de botones

Nota: Puedes ver algunas advertencias y errores en AS3 al agregar el código de manejo de eventos. Es muy probable que falten declaraciones de importación; simplemente coloca el mouse sobre las líneas onduladas y use la solución rápida (Option + Intro para macOS | Alt + Intro para Windows y Linux). Debes importar todos los paquetes necesarios.

```
private class ButtonHandler implements View.OnClickListener{

    @Override
    public void onClick(View view){ 1
        switch (view.getId()){ 2
            case R.id.button: 3
                show("Boton uno"); 4
                break;
            case R.id.button2:
                show("Boton dos");
                break;
            case R.id.button3:
                show("Boton tres");
                break;
            default:
                show("This ");
        }
    }
}
```

Listado 8.14. Implementación ButtonHandler.

❶ Estamos anulando el método onClick de View.OnClickListener; cuando se hace clic en cualquiera de los tres botones, se llama a este método, al igual que en nuestro código de clase anónima de las secciones anteriores. El tiempo de ejecución completará el parámetro Ver con la referencia del objeto del botón real que se hizo clic. Eso es lo que vamos a usar para identificar en qué botón se hizo clic exactamente usando la clase Toast de Android.

❷ El método getId del objeto View devuelve un valor entero que corresponde al ID del botón como se define en la clase R. Recuerda que los archivos de diseño se inflan durante el tiempo de ejecución para producir los objetos reales de Java que corresponden al elemento Ver descrito en el diseño. El tiempo de ejecución genera la clase R., que podemos utilizar para referirnos a los objetos definidos en el archivo de diseño de forma programática.

❸ Estamos simplemente comprobando si el valor que obtuvimos de view.getId es cualquiera de nuestros botones; en esta línea, estamos comprobando si es R.id.button1.

❹ Si se hizo clic en el botón 1, llamamos a un método llamado show() y le pasamos una cadena. Aún no hemos definido el método de show.

```
void show(String message) {
    Toast.makeText( context: this, message, Toast.LENGTH_LONG).show(); ❶
    Log.i(getClass().getName(), message); ❷
}
```

Listado 8.15: Método show().

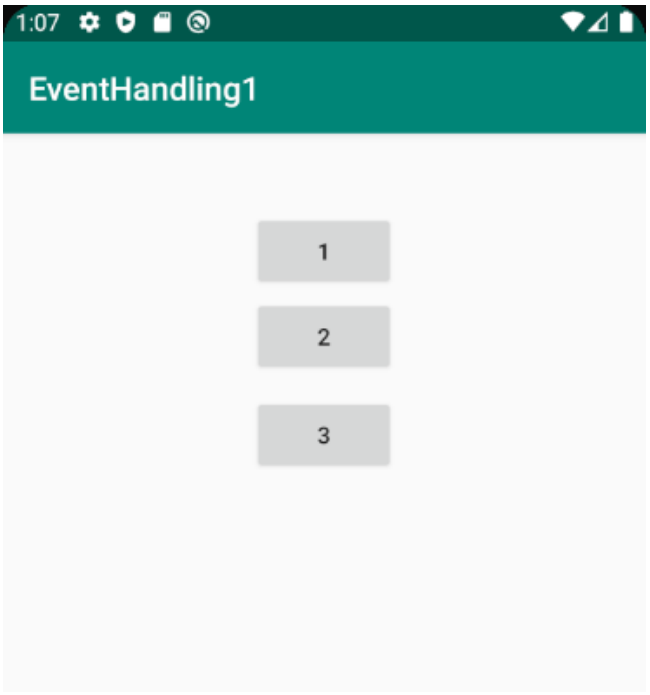
❶ Estamos mostrando un mensaje de Toast. Toast proporciona un pequeño comentario en forma de una pequeña ventana emergente. Aparece como una superposición en la actividad actual, y la apariencia es solo por una cierta duración, por lo que no oculta la actividad actual. Es una forma discreta de mostrar mensajes de estado.

❷ La clase Log nos permite crear entradas de registro de manera muy similar a System.out.println, pero es más apropiado usar la clase Log para generar mensajes de diagnóstico y de depuración. Puedes ver las entradas de registro creadas por la clase Log en la ventana de la herramienta Logcat.

El método show (Listado 8.15) es miembro de MainActivity. ButtonHandler tiene acceso a métodos (o variables) que están definidos en su clase adjunta. Podríamos haber definido este método dentro de la clase ButtonHandler, y eso también habría estado bien.

Figura 8.11: Mensaje de Toast.

El código completo para MainActivity se encuentra en el listado 8.16.



Uso de MainActivity como Listener

Otra forma de manejar eventos para los tres botones sería usar la clase MainActivity como el objeto detector. No queremos cambiar el archivo principal del programa tal como está ahora; de esa manera, podemos referirnos a esto más tarde. Podemos definir otra clase que servirá como nuestro archivo de programa principal; de esa manera, puedes estar al lado del programa principal original dentro del mismo proyecto. Desde la barra de menú principal, haz clic en Archivo ➤ Nueva ➤ clase Java y llénala como se muestra en la figura 8.12.

Alternativamente, también puedes hacer clic con el botón derecho en la carpeta que contiene MainActivity y usar el menú contextual para agregar una clase.

```java
package bancho.com.eventhandling1;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

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

        ButtonHandler bh = new ButtonHandler();
        findViewById(R.id.button).setOnClickListener(bh);
        findViewById(R.id.button2).setOnClickListener(bh);
        findViewById(R.id.button3).setOnClickListener(bh);
    }

    private class ButtonHandler implements View.OnClickListener{

        @Override
        public void onClick(View view){
            switch (view.getId()){
                case R.id.button:
                    show("Boton uno");
                    break;
                case R.id.button2:
                    show("Boton dos");
                    break;
                case R.id.button3:
                    show("Boton tres");
                    break;
                default:
                    show("Esto no deberia pasar");
            }
        }
    }

    void show(String message) {
        Toast.makeText(this, message, Toast.LENGTH_LONG).show();
        Log.i(getClass().getName(), message);
    }
}
```

Listado 8.16: Código completo para MainActivity.

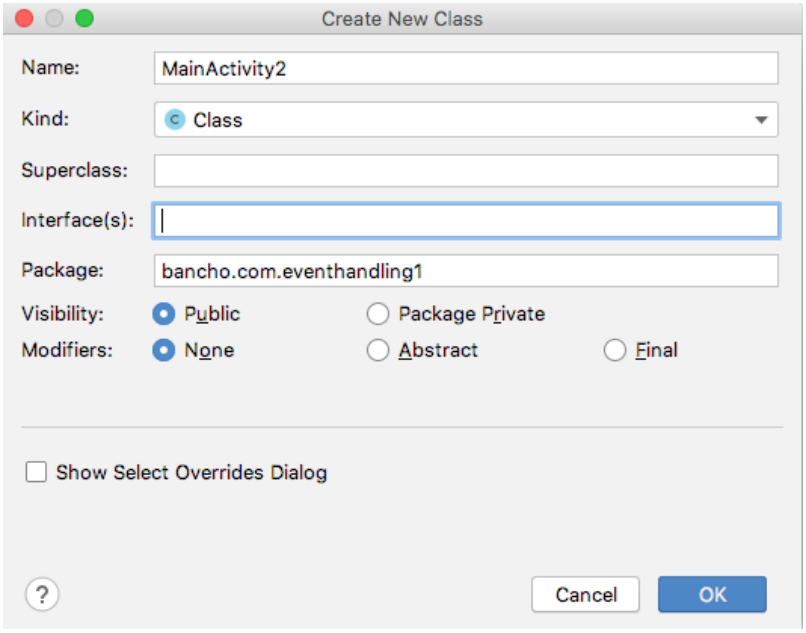


Figura 8.12: Crear una nueva clase de diálogo.

Nombraremos la nueva clase MainActivity2. Extiende la misma superclase que el programa principal original. Deja el campo “interfaces” en blanco y deja la entrada predeterminada en el “Paquete”. La visibilidad pública y ningún modificador deberían estar bien para nuestra configuración.

Es posible que veas algunas advertencias en la ventana del editor del archivo de programa recién creado; simplemente nos están advirtiendo que la nueva clase no está registrada en el archivo AndroidManifest (figura 8.13).

```
package bancho.com.eventhandling1;

public class MainActivity2 {
}
```

Figura 8.13: MainActivity2.

Deja esto por ahora; lo arreglaremos más tarde. Por el momento, nuestro nuevo archivo de programa no tiene todo lo que necesita para ser una clase de actividad adecuada, no anula el método onCreate y no tiene ningún archivo de diseño asociado. Arreglemos eso suministrando el código que falta.

```java
package bancho.com.eventhandling1;

import android.os.Bundle;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity2 extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Listado 8.17: Archivo de programa MainActivity2.

Simplemente podrías copiar el código en el Listado 8.17 en tu propio proyecto, o podrías probar algunas herramientas más de AS3. Mientras editas MainActivity2, coloca el cursor en algún lugar dentro del bloque de clase principal y presiona Ctrl + O (el mismo atajo de teclado para Windows, Linux y macOS).

Esto abrirá una ventana de diálogo donde puedes elegir anular el método de una clase. Es sensible al contexto: sabe que estás editando una clase AppCompatActivity, por lo que muestra solo los métodos de esa clase (consulta la figura 8.14). Alternativamente, puedes acceder a la ventana de diálogo de anulación desde la barra de menú principal Código ➤ Anular métodos.

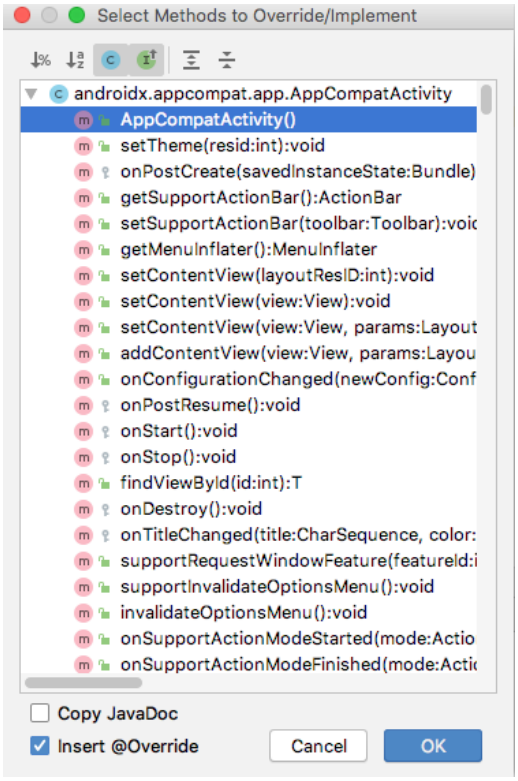


Figura 8.14: Anular ventana de diálogo.

Usar la ventana de diálogo es (a veces) mucho más fácil que escribir directamente el código. Si no estás familiarizado con la sintaxis del método y todos sus parámetros y anotaciones correspondientes, la ventana de diálogo los proporcionará encantados.

El listado 8.18 muestra el MainActivity2 (plegado) en su totalidad.

```
public class MainActivity2 extends AppCompatActivity                    ❶
    implements View.OnClickListener{

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState){...}    ❷

    @Override
    public void onClick(View view) {...}                                 ❸

    void show(String message) {...}                                      ❹
}
```

Listado 8.18: MainActivity2.

❶ Necesitas agregar la directiva implements en la clase. Lo que esto significa es que la clase MainActivity2 se comportará como si fuera un objeto OnClickListener. Eso es lo que significa implementar cualquier interfaz.
Básicamente, estamos de acuerdo con un determinado contrato de objeto que independientemente de los comportamientos que muestre la interfaz, nos comportaremos de la misma manera.

❷ El método onCreate contiene el mismo código que el del programa principal original (MainActivity.java); por supuesto, todavía deberíamos agregar la declaración setContentView y los registros de vista; pronto lo haremos.
Puedes observar el decorador @Nullable en el parámetro Bundle; esto simplemente significa que el objeto Bundle, en caso de que sea nulo, no es un gran problema y que se puede ignorar sin riesgo.

❸ El método onClick anulado de la interfaz OnClickListener ahora se implementa como un método miembro de MainActivity2.

❹ Esta es la misma implementación del método show como hemos visto en MainActivity original.

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

    Button b1 = (Button) findViewById(R.id.button);
    Button b2 = (Button) findViewById(R.id.button2);
    Button b3 = (Button) findViewById(R.id.button3);

    ButtonListener blistener = new ButtonListener();
    b1.setOnClickListener(blistener);
    b2.setOnClickListener(blistener);
    b3.setOnClickListener(blistener);
}
```

Listado 8.19: Ver registros.

El método onCreate en el ejemplo del código anterior no es muy diferente de nuestros códigos de manejo de eventos; lo único diferente es que estamos usando el mismo objeto para manejar los eventos para los tres botones.

Ninguno de los botones tiene su propio oyente dedicado, como fue el caso con el uso de clases anónimas. En este enfoque, la lógica del programa se enruta dentro del objeto detector (ButtonHandler).

Para probar nuestro código, debemos hacer un pequeño cambio en el archivo AndroidManifest. Por el momento, la clase de actividad declarada en el manifiesto es MainActivity, nuestro archivo de programa principal original.

Cuando el tiempo de ejecución de Android inicia una aplicación, echa un vistazo a la declaración de actividad en el manifiesto y ejecuta ese programa. Necesitamos cambiar esa entrada para que el tiempo de ejecución de Android inicie MainActivity2 en lugar de MainActivity.

Abre el archivo AndroidManifest.xml desde la ventana de la herramienta del proyecto. Debería estar en la aplicación ➤ manifiesto ➤ AndroidManifest.xml.

```java
package bancho.com.eventhandling1;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity2 extends AppCompatActivity
    implements View.OnClickListener{

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

    @Override
    public void onClick(View view) {
        switch (view.getId()){
            case R.id.button:
                show("Boton uno");
                break;
            case R.id.button2:
                show("Boton dos");
                break;
            case R.id.button3:
                show("Boton tres");
                break;
            default:
                show("Esto no deberia pasar");
        }
    }

    void show(String message) {
        Toast.makeText(this, message, Toast.LENGTH_LONG).show();
        Log.i(getClass().getName(), message);
    }
}
```

Listado 8.20: Código completo para MainActivity2.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="bancho.com.eventhandling1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="EventHandling1"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity2"> ❶
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Listado 8.21: Entrada de actividad en el archivo del manifiesto.

❶ Esta entrada es lo que le dice al tiempo de ejecución de Android qué archivo java es el programa principal o el archivo de inicio de la aplicación. Cambia el valor del elemento de actividad a ".MainActivity2", como se muestra en el Listado 8.13.

Ahora puedes ejecutarlo en el emulador