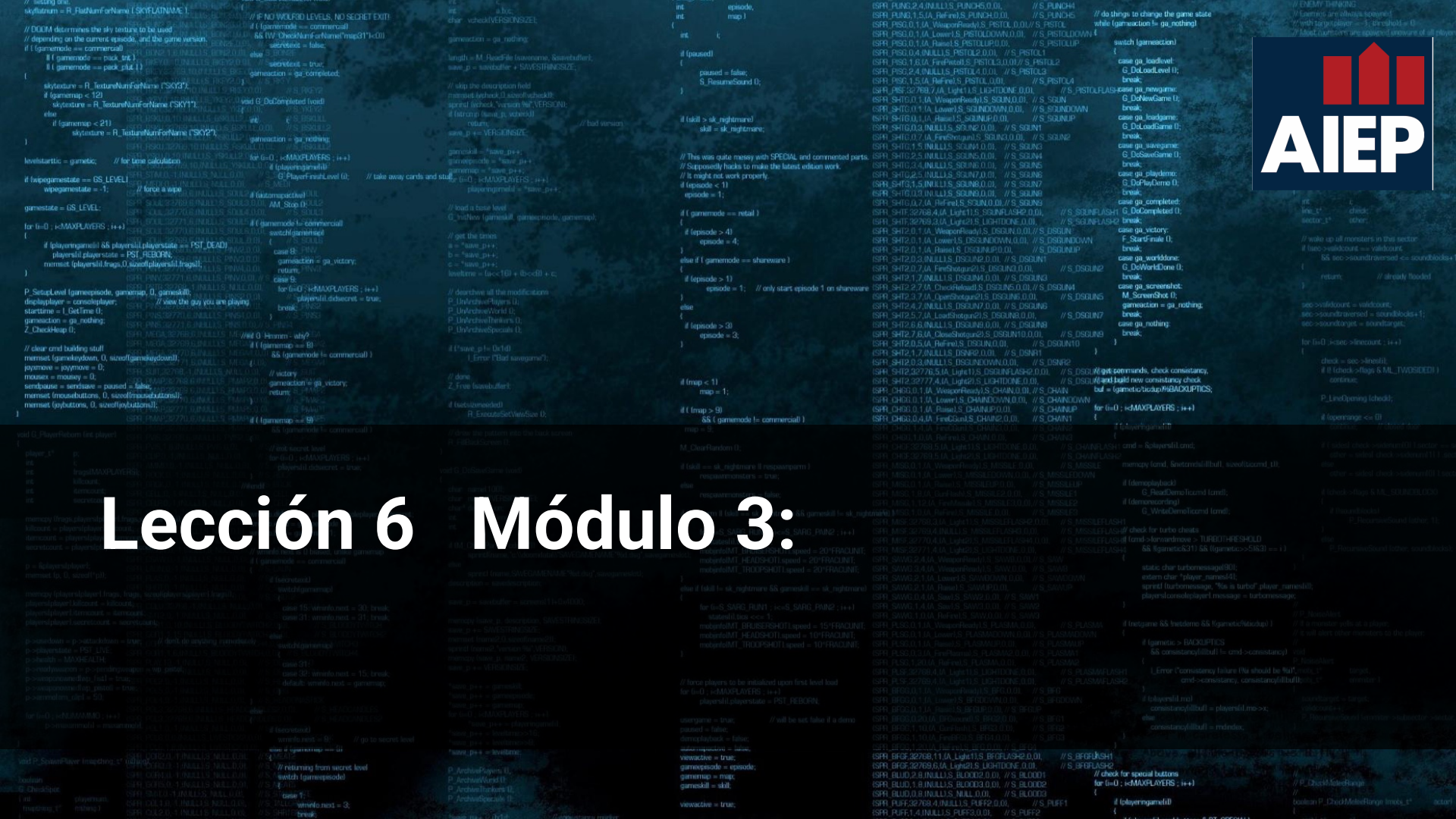


Lección 6 Módulo 3:

Manejo de eventos

<title> Objetivos </title>

- Eventos de entrada
- Event Handlers (manejadores de eventos)
- Event Listeners (escuchadores de eventos)
- Formas de agregar un nuevo listener
- Ejercicio

Eventos de entrada

Objetivos

- Definir un concepto común de callback
- Interacción con el usuario
- Manejadores de eventos como
  - onKeyDown()
  - onKeyUp()
  - onTouchEvent()

Callback

Callback (o *call-after function*) es un código que se entrega como parámetro a otra función para que sea ejecutado en un momento dado, en forma síncrona o asíncrona.

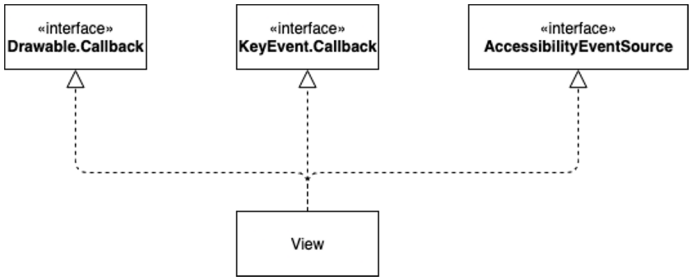
La implementación depende del lenguaje de programación usado, por ejemplo:

- Métodos
- Lambdas
- Bloques
- Punteros a función.

Eventos de entrada

- Un evento de entrada es un **evento de interacción entre usuario y app**
- Cada objeto View maneja los eventos con los que el usuario interactúa
- La clase View tiene varios callbacks que sirven para los eventos de UI
- El framework de Android utiliza los callbacks cuando ocurre la acción respectiva
- Hay más de una forma de interceptar los eventos desde una interacción de usuario

Clase View



[Drawable.Callback]

[KeyEvent]

[AccessibilityEventSource]

Category	Methods	Description
Creation	Constructors	There is a form of the constructor that are called when the view is created from code and a form that is called when the view is inflated from a layout file. The second form should parse and apply any attributes defined in the layout file.
	`onFinishInflate()`	Called after a view and all of its children has been inflated from XML.
Layout	`onMeasure(int, int)`	Called to determine the size requirements for this view and all of its children.
	`onLayout(boolean, int, int, int, int)`	Called when this view should assign a size and position to all of its children.
	`onSizeChanged(int, int, int, int)`	Called when the size of this view has changed.
Drawing	`onDraw(android.graphics.Canvas)`	Called when the view should render its content.
Event processing	`onKeyDown(int, android.view.KeyEvent)`	Called when a new hardware key event occurs.
	`onKeyUp(int, android.view.KeyEvent)`	Called when a hardware key up event occurs.
	`onTrackballEvent(android.view.MotionEvent)`	Called when a trackball motion event occurs.
	`onTouchEvent(android.view.MotionEvent)`	Called when a touch screen motion event occurs.
Focus	`onFocusChanged(boolean, int, android.graphics.Rect)`	Called when the view gains or loses focus.
	`onWindowFocusChanged(boolean)`	Called when the window containing the view gains or loses focus.
Attaching	`onAttachedToWindow()`	Called when the view is attached to a window.
	`onDetachedFromWindow()`	Called when the view is detached from its window.
	`onWindowVisibilityChanged(int)`	Called when the visibility of the window containing the view has changed.

Vista personalizada

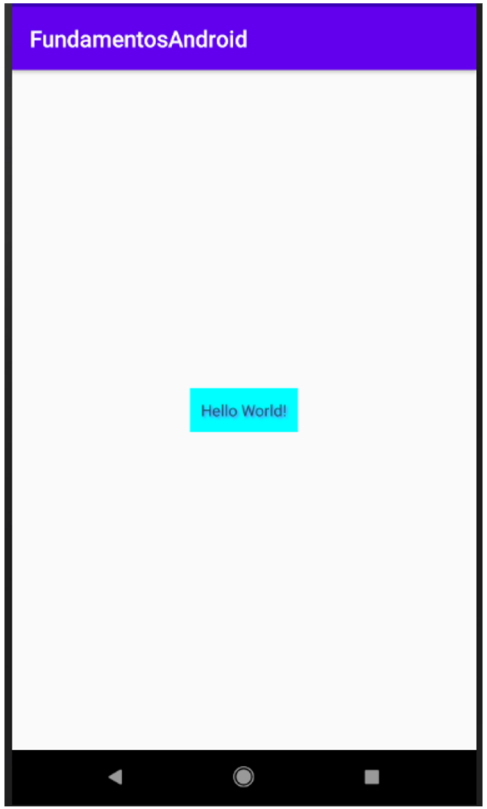
- Hereda de TextView
- Sobreescribir el método onTouchEvent
- Modifica el aspecto del texto

```
public class MyCustomView extends TextView {

    public MyCustomView(Context context, @Nullable AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        Timber.d("onTouchEvent " + event.toString());
        decorateText();
        return super.onTouchEvent(event);
    }

    private void decorateText() {
        if (!getText().toString().isEmpty()) {
            // set the characteristics and the color of the shadow
            setShadowLayer(6, 4, 4, Color.rgb(250, 00, 250));
            setBackgroundColor(Color.CYAN);
        } else {
            setBackgroundColor(Color.RED);
        }
    }
}
```

developer.android.com/create-custom-view

Ejercicio

- Agregar la clase MyCustomView que herede de la clase View
- Agregar un layout con MyCustomView
- Asignar padding a la vista
- Que el tamaño de la vista se ajuste al contenido



Event handlers

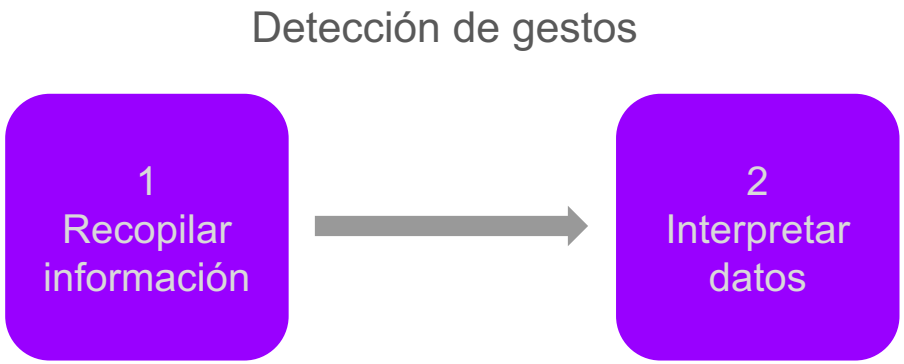
Event Handler

Al construir un componente personalizado existen varios métodos utilizados (event handlers) como *callback* para manejar eventos.

Event Handler	Descripción
`onKeyDown(int, KeyEvent)`	Llamado cuando se presiona una tecla
`onKeyUp(int, KeyEvent)`	Llamado cuando se libera un tecla
`onTrackballEvent(MotionEvent)`	Cuando ocurre un movimiento de trackball
`onTouchEvent(MotionEvent)`	Cuando el usuario toca la pantalla
`onFocusChanged(boolean, int, Rect)`	Cuando la vista gana o pierde el foco

Detección de gestos comunes

El usuario coloca uno o más dedos en la pantalla táctil y la app interpreta ese patrón de toques como un gesto específico



developer.android.com

Gesto táctil personalizado

- Extraer la acción realizada por el usuario (datos sin procesar)
- Procesar el evento para determinar si se produjo un gesto
- Para gestos comunes como presionar 2 veces o mantener presionado, existe GestureDetector


```
public class MyCustomView extends TextView {

  public MyCustomView(Context context, @Nullable AttributeSet attrs) {
     super(context, attrs);
  }

  public boolean onTouchEvent(MotionEvent event) {
   int action = event.getAction();

   switch (action) {
      case (MotionEvent.ACTION_DOWN):
         Timber.d("Action was DOWN");
         return true;
      case (MotionEvent.ACTION_MOVE):
         Timber.d("Action was MOVE");
         return true;
      case (MotionEvent.ACTION_UP):
         Timber.d("Action was UP");
         return true;
      case (MotionEvent.ACTION_CANCEL):
         Timber.d("Action was CANCEL");
         return true;
      case (MotionEvent.ACTION_OUTSIDE):
         Timber.d("Movement occurred outside bounds " +
               "of current screen element");
         return true;
      default:
         return super.onTouchEvent(event);
   }
}
```

Detector de gestos

GestureDetector permite detectar gestos comunes como:

- onDown()
- onLongPress()
- onFling()

Se puede utilizar GestureDetector junto con onTouchEvent()

Ejemplo

GestureDetector expone algunas interfaces (como OnDoubleTapListener) para eventos específicos y OnGestureListener que notifica cuando ocurre un gesto genérico.

developer.android.com/detector

```
public class MainActivity extends AppCompatActivity implements
        GestureDetector.OnGestureListener,
        GestureDetector.OnDoubleTapListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Instanciar gesture detector con el contexto y
        // la implementación de OnGestureListener
        mDetector = new GestureDetectorCompat(this, this);
        // Asignar el gesture detector como listener de double tap
        mDetector.setOnDoubleTapListener(this);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event){
        if (this.mDetector.onTouchEvent(event)) {
            return true;
        }
        return super.onTouchEvent(event);
    }

    @Override
    public boolean onSingleTapConfirmed(MotionEvent e) {
        Timber.d("onSingleTapConfirmed " + e.toString());
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent e) {
        Timber.d("onDoubleTap " + e.toString());
        return true;
    }
}
```

Event listeners

Event Listeners

Un receptor de eventos (*event listener*) es una interfaz en la clase View que contiene un único callback.

Los métodos serán llamados por el framework de Android cuando la View que tiene registrado el listener sea gatillado por la interacción del usuario.

Interfaz	Event listener
View.OnClickListener	onClick(View v)
View.OnLongClickListener	onLongClick(View v)
View.OnKeyListener	onKey(View v, int keyCode, KeyEvent event)
View.OnTouchListener	onTouch(View v, MotionEvent event)
View.OnFocusChangeListener	onFocusChange(View v, boolean hasFocus)

Alternativas para implementar event listeners

A) Implementar la interfaz

- Es la Activity quien implementa el listener

- Para utilizarlo, se debe usar el parámetro **View v** para conocer la vista que fue *clickeada*

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

  private ActivityMainBinding binding;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    binding.textViewMain.setOnClickListener(this);
    binding.buttonMain.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    Timber.d("onClick() called with: v = " + v.getId());
  }
}
```

B) Implementación anónima

- Se define una variable que implementa el listener sin crear una clase

- Para utilizarlo, se debe usar el parámetro **View v** para conocer la vista que fue *clickeada*

```
public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    private View.OnClickListener onClickListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Timber.d("onClick() called with: v = " + v.getId());
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        binding.textViewMain.setOnClickListener(onClickListener);
        binding.buttonMain.setOnClickListener(onClickListener);
    }
}
```

C) Implementación anónima

Cada vista tiene asociado su propio *listener*

```
public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        binding.textViewMain.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              Timber.d("onClick() called with: v = " + v.getId());
            }
        );

        binding.buttonMain.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
              Timber.d("onClick() called with: v = " + v.getId());
            }
        );
    }
}
```

D) Lambda

- Cada vista tiene asociado su propio listener

- Se implementa usando predicado

```
public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        binding.textViewMain.setOnClickListener(v -> Timber.d("onClick()"));
    }
}
```

Resumen de alternativas para implementar event listeners

A. Implementación de la interfaz por parte de la actividad

B. Implementación anónima como variable de instancia

C. Implementación anónima en la vista

D. Uso de funciones lambda con sintaxis reducida

Lambda expressions

Las *lambda expressions* son una de las funcionalidades más importantes incorporadas en Java 8

Se debe configurar el proyecto como compatible con Java 8 para usar las funciones del lenguaje

```
android {
 ...
 defaultConfig {
   ...
 }
 compileOptions {
   sourceCompatibility JavaVersion.VERSION_1_8
   targetCompatibility JavaVersion.VERSION_1_8
 }
}
```

Retrolambda entrega compatibilidad de lambda expression para proyectos con versiones anteriores

Lambda expressions: Predicado

```
( argument ) -> { expression body }
```

Predicado: Argumento

(argument) -> { expression body }

argument (argumento) puede ser uno o más para parámetros

```
() -> Timber.d("Lambda sin parámetros")
```

```
event -> Timber.d("Lambda un parámetro");
```

```
(param1, param2) -> Timber.d("Lambda con 2 params: " + param1 + " - " + param2)
```

Predicado: Cuerpo

(argument) -> { expression body }

expression body es el cuerpo de la función y puede tener o no valor de retorno

```
(parameter) -> {
   Timber.d("event");
}
```

```
parameter -> {
   Timber.d("onLongClick event");
   return true;
}
```

Ejemplo usando lambda

View.OnLongClickListener retorna un boolean indicando indicando si el callback consumió el evento

View.OnClickListener no tiene valor de retorno asociado

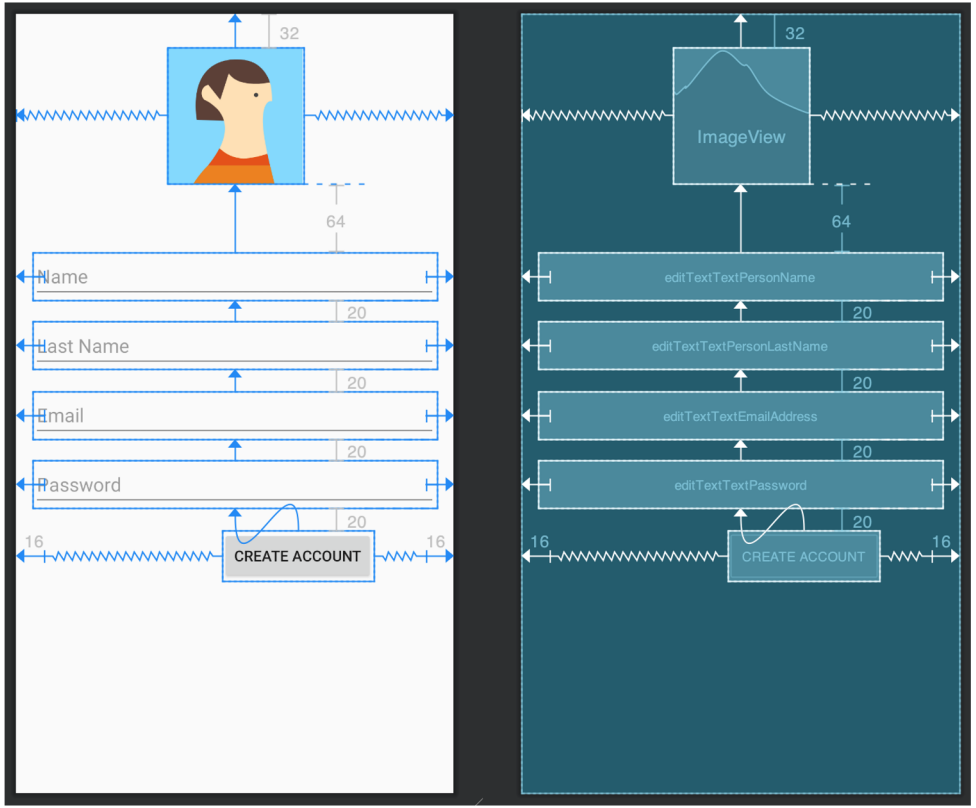
```
public class MainActivity extends AppCompatActivity {

  private ActivityMainBinding binding;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    binding.textViewMain.setOnLongClickListener( v -> {
      Timber.d("onLongClick event");
      return true;
    });

     binding.textViewMain.setOnClickListener(v -> Timber.d("onClick event"));
  }
}
```

Ejercicio

El botón “CREATE ACCOUNT” despliega un mensaje al usuario indicando si los datos son correctos

Acciones a realizar:

- Implementar el listener del botón
- Obtener y validar los datos
- Implementar el despliegue de mensaje con los datos ingresados



Buen día!

Foto de David Besh en Pexels