Juan Pablo García Plaza Pérez (jgarciapft@alumnos.unex.es)

ASEE | **HOJA DE ACTIVIDADES 02**

ÍNDICE DE CONTENIDOS

PREGUNTA 1

FUENTES: Tema 4, https://developer.android.com/guide/topics/ui/layout/linear

Para las dimensiones de los widgets existen 2 atributos comunes a todas las vistas: *android:width* y *android:height*, y son comunes los valores *match_parent* y *wrap_content*, que sirve, respectivamente, para heredar la dimensión del padre y para ajustar la dimensión al contenido que encierra el widget. Pero para la posición la respuesta varía según el tipo de Layout raíz sobre el que esté construido la interfaz. Vamos a suponer que se trata de un *ConstraintLayout,* un tipo de Layout especialmente diseñado para ser manipulado desde Android Studio. En este Layout los elementos se posicionan según una restricción vertical y otra horizontal, por lo menos. Por ejemplo, podemos indicar estas restricciones como el margen hasta un borde. De esta manera, podemos desplazar a la izquierda y abajo un elemento en el Layout indicando un margen derecho como restricción horizontal y un margen superior como restricción vertical.

En Android una interfaz de usuario se especifica como un recurso *layout*, que es un documento XML conforme al esquema definido en http://schemas.android.com/apk/res/android. Por lo tanto, se almacena en la carpeta *layout* dentro de la carpeta de recursos *res*.

Una posible implementación en XML para la interfaz que se muestra:

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:paddingLeft="24dp"
android:paddingRight="24dp">

<EditText
android:id="@+id/editTextTextPersonName"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_weight="0"
android:ems="10"
android:hint="To"
android:inputType="textPersonName"/>

<EditText
android:id="@+id/editTextTextPersonName2"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_weight="0"
android:ems="10"
android:hint="Subject"
android:inputType="textPersonName"/>

<EditText
android:id="@+id/editTextTextMultiLine"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_weight="1"
android:ems="10"
android:gravity="start|top"
android:hint="Message"
android:inputType="textMultiLine"/>

<Button
android:id="@+id/button"
```

```
android:layout_width="100dp"
android:layout_height="wrap_content"
android:layout_gravity="right"
android:layout_weight="0"
android:text="Send"/>
</LinearLayout>
```

PREGUNTA 2

FUENTES: Tema 4, https://www.geeksforgeeks.org/handling-click-events-button-android/

Existen 2 formas de definir el comportamiento para un evento *onClick* para cualquier vista de Android en general:

- **Desde el recurso XML** que define el layout de la interfaz en la que se encuentra el botón. Dentro del botón, en el elemento XML *android:onClick* podemos escribir el nombre del método que será invocado cuando se produzca el evento. Sin embargo, este método tiene la limitación de no funcionar en fragments, pues el método manejador solo puede estar en una clase *Activity*.

- **Desde código Java** registrando un listener *View.OnClickListener* en el objeto *Button* correspondiente al botón, el cual se puede recuperar, por ejemplo, con *findViewById*. El registro debe ocurrir antes de que el botón pueda escuchar eventos onClick, por lo tanto, un buen lugar para registrar el listener es en el callback *onCreate* del ciclo de vida de la actividad dependiente de la interfaz en la que se encuentra el botón. El código definido en el método *onClick* de *View.OnClickListener* será invocado cada vez que el botón reciba el evento *onClick*.

A continuación, se muestra el ejemplo de un botón que provoca que la aplicación lea los contactos almacenados en el dispositivo:

```
m_b_readContacts=(Button)findViewById(R.id.bReadContacts);

m_b_readContacts.setOnClickListener(newView.OnClickListener(){
      @Override
            publicvoidonClick(Viewv){
                  readContacts();
            }
      });
```

PREGUNTA 3

FUENTES: https://developer.android.com/guide/topics/ui/notifiers/notifications

1. **Icono pequeño**. Es obligatorio y se establece con *setSmallIcon()*. Sirve para que el usuario pueda asociar rápidamente la notificación con el evento del que proviene para poder interpretar el contenido en un contexto, por ejemplo, un recordatorio, una alarma o un mensaje, como es el caso del ejemplo.
2. **Nombre de la app**. Es proporcionado automáticamente por Android. Sirve para que el usuario pueda asociar directamente una notificación con la aplicación que la ha generado.
3. **Marca de tiempo**. También es proporcionada por Android automáticamente, aunque es posible sobreescribirla con el método *setWhen()* y esconderla con *setShowWhen(false)*. Indica al usuario la antigüedad de la notificación.
4. **Icono grande**. Es opcional y no debe utilizarse para establecer el icono de la aplicación. En su lugar debe utilizarse para añadir información de contexto para el usuario como, por ejemplo, la foto de perfil de un contacto en el caso de que la notificación fuese un mensaje de texto. Se establece con *setLargeIcon()*.
5. **Título**. Es opcional. Se establece con *setContentTitle()*.
6. **Texto**. Es opcional. Se establece con *setContentText()*.

Adicionalmente, desde la API de Android 24 en adelante las notificaciones pueden contener acciones en su forma expandida. Un caso de uso sería, por ejemplo, responder directamente a un mensaje desde la notificación introduciendo la respuesta por teclado en un campo de texto.

PREGUNTA 4

FUENTES: Temas 4 y 5, https://developer.android.com/reference/android/content/SharedPreferences

En el paradigma de programación tradicional de Java los datos generados por el código se almacenan en memoria mientras el programa esté en ejecución, una vez termina, con él mueren sus datos. En plataformas móviles ocurre que los usuarios están constantemente mirando y guardando su móvil, por lo que las aplicaciones se cierran y abren continuamente. Y estos no quieren perder los datos generados cada vez que apagan el móvil. Por lo tanto, es necesario implementar una capa de persistencia de datos para poder sincronizar las ejecuciones de la aplicación con los datos almacenados. Android implementa la persistencia de datos a través de 4 mecanismos:

- **Almacenamiento específico de la app**. El sistema operativo Android está basado en Linux. Cada aplicación recibe su propio directorio en la que puede escribir y leer ficheros de datos arbitrarios.
- **Almacenamiento compartido**. Ficheros disponibles para compartir con otras aplicaciones.
- **Preferencias**. Un mecanismo de almacenamiento de datos primitivos en un almacén de pares de clave - valor gobernado por la API *SharedPreferences*.
- **Bases de datos**. Android ofrece almacenamiento en bases de datos relacionales ligeras en formato SQLite, privadas para cada aplicación.

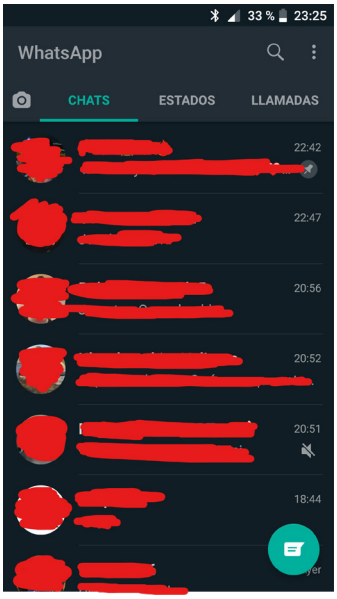
En el caso de una app de recetas, suponemos que el dato principal que almacena son recetas culinarias. Los usuarios quieren que sus recetas persistan cada vez que vuelven a ejecutar la aplicación. Suponemos que una receta es un tipo complejo que requiere almacenar varios campos como el título, el tiempo de preparación, lista de ingredientes, lista de pasos, etc… Por lo que no hay una forma sencilla de representar una receta como un par de clave-valor, las preferencias quedan descartadas. Como no sabemos tampoco si esta aplicación necesita compartir sus recetas el almacenamiento compartido también queda descartado. Por otra parte, podríamos diseñar un formato de texto para cada receta e implementar un proceso de serialización y de-serialización a archivos en formato receta que guardásemos en el almacenamiento específico de la app. Y otra posibilidad es crear un esquema relacional e implementar las recetas como una base de datos SQLite.

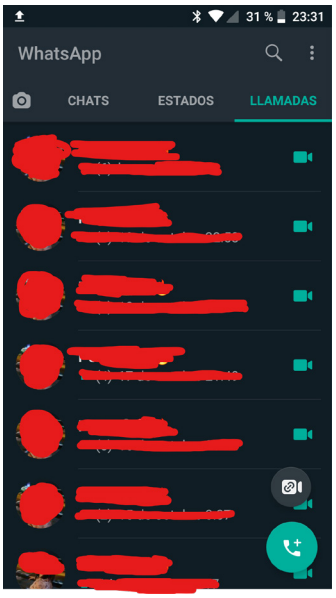
PREGUNTA 5

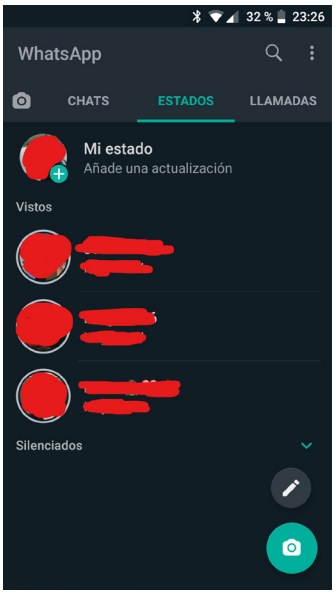
FUENTES: Tema 4

Las pantallas hermanas son vistas de información complementaria u otros ítems de la misma colección que la vista actual, lo que conforma dos tipos distintos de pantallas hermanas: relacionadas por sección y relacionadas por sección, respectivamente.

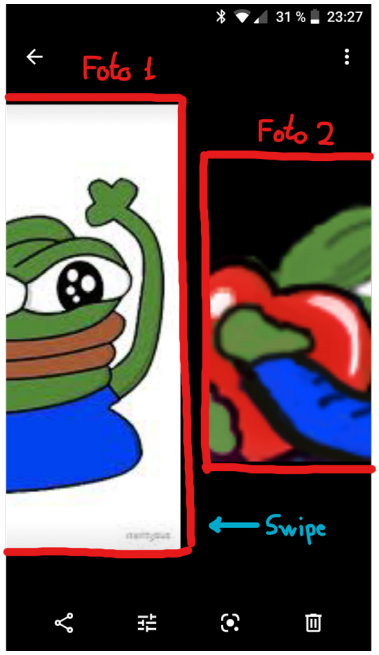
Un ejemplo real de **pantallas hermanas relacionadas por sección** son las pantallas de chats abiertos, estados y llamadas en Whatsapp. La pantalla de chats muestra una lista con tus conversaciones abiertas, la pantalla de estados muestra de forma complementaria los estados de tus contactos y la de llamadas tus llamadas recientes. La navegación está implementada con el patrón de **Paginación Horizontal** o *Swipe Views* porque es un número reducido de secciones, ningún elemento responde a gestos horizontales que puedan disminuir la usabilidad y todas las pantallas deben mostrar listas de contenido variable, que puede ser cuantioso, por lo que disponer de mucho espacio en pantalla mejora su usabilidad. Para navegar entre secciones se puede realizar un gesto de desplazamiento horizontal







Un ejemplo real de **pantallas hermanas relacionadas por colección** es la colección de fotos de la galería de Android cuando la vista actual es el detalle de una foto. Podemos ver el resto de las fotos de la colección haciendo un gesto lateral con un dedo a izquierda o derecha, no hace falta volver a la lista de miniaturas para consultar el resto de las fotos. Este patrón se conoce como **Carrusel.**



PREGUNTA 6

FUENTES: Tema 4

a. Una **colección de fotos**. El patrón de **Carrusel** porque podemos ver el resto de las fotos de la colección haciendo un gesto lateral con un dedo a izquierda o derecha, no hace falta volver a la lista de miniaturas para consultar el resto de las fotos.

b. Una **colección de contactos**. El patrón de **Paginación Horizontal** (también conocido como *Swipe Views*) principalmente porque nos permite desplazarnos por la información de cada contacto sin necesidad de volver a la lista de contactos. Para navegar por los distintos contactos almacenados utilizamos gestos de desplazamiento horizontales hacia la izquierda para ver el siguiente contacto, y hacia la derecha para regresar al contacto previo.

c. **Pantallas con información complementaria sobre un sitio turístico**. El patrón **Tabs** porque relaciona en el elemento padre (el sitio turístico) la navegación a todas las secciones a la vez que informa al usuario qué puede esperar encontrar en ellas con el título de la pestaña. Las pestañas estarán agrupadas en la parte superior de la vista. Una alternativa podría ser la Paginación Horizontal, pero yo la he descartado porque una de las secciones es un mapa, que puede ser susceptible de implementar su propio comportamiento para los gestos de desplazamiento horizontal. Esta situación puede confundir al usuario y disminuir la usabilidad de la pantalla.

d. **Días del calendario**. El patrón de **Paginación Horizontal** por motivos similares al de la colección de contactos, no es necesario volver a la vista de días en el mes para navegar a un día concreto. Además, puede resultar especialmente intuitivo para el usuario desplazarse por los días con gestos de desplazamiento horizontal porque los asocia con la noción del paso del tiempo.

PREGUNTA 7

FUENTES: Tema 4

La navegación *drill-down* consiste en la disposición jerárquica que resulta de agrupar los datos como una lista que a su vez contiene datos agrupados como otras listas. Es un problema que puede derivar del uso de patrones de navegación basados en listas porque resulta frustrante para el usuario encontrar el dato que busca si tiene que estar retrocediendo y avanzado por la jerarquía continuamente. Para evitar este fenómeno es recomendable que la profundidad máxima sea de 2 niveles.

PREGUNTA 8

FUENTES: Tema 4

La **navegación temporal** en Android es el paradigma de esperar que el sistema te lleve a la pantalla que has visitado inmediatamente anterior y que converse el estado en el que la abandonaste cuando pulsas en el botón *Back*, siendo el lanzador de Android la última pantalla a la que es posible retroceder. Es tradicional en Android y está asociada a la *Pila Back*, una estructura que opera como una pila y que almacena la secuencia de actividades por las que el usuario ha navegado mientras usa el dispositivo, no está limitado al ámbito de la ejecución de una aplicación. En ocasiones este paradigma puede ser frustrante para el usuario si desea navegar a la pantalla principal de la aplicación en ejecución, pero, como resultado de presionar varias veces el botón *Back*, es devuelto al lanzador de Android.

Por otra parte, la **navegación ancestral** es un concepto más nuevo que busca resolver el problema de la navegación temporal expuesto anteriormente. Asocia la metáfora *Arriba* a volver a la vista del padre del elemento de la jerarquía de vistas en la que se encuentra el usuario actualmente, por lo que nunca podrá devolverle al lanzador de Android. Cada pantalla tiene un solo ancestro, por tanto, su navegación ancestral conducirá siempre a una única pantalla dentro de la misma aplicación. Por ejemplo, si el usuario está viendo los detalles de una película en una aplicación de cartelera de cine, la navegación ancestral le debería llevar a la lista de películas de la cartelera.

Aplicando los conceptos de navegación temporal y ancestral al esquema de pantallas de la pregunta, tenemos que:

1. El usuario abre la aplicación Play Store desde el lanzador de Android. Este es un ejemplo de navegación entre dos aplicaciones distintas y si el usuario

pulsase el botón *Back* la navegación temporal le lleva de vuelta a la primera aplicación, al lanzado de Android. El usuario se encuentra actualmente en la pantalla principal de la Play Store, por lo que la navegación ancestral le llevaría de nuevo a esta misma pantalla.

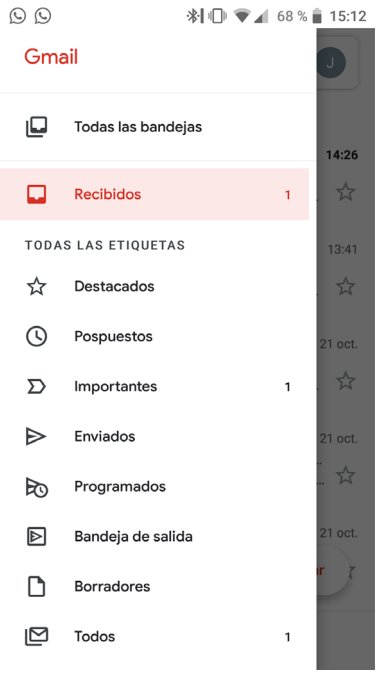
2. El usuario decide navegar hasta el detalle de un libro. La navegación temporal y la ancestral coinciden porque la pantalla anterior es la pantalla del padre de esta nueva vista. Cabe destacar que, aunque la navegación temporal le permitiese volver al lanzador de Android, con la navegación ancestral nunca podría salir de la aplicación. En la vista actual al usuario se le presenta la opción de "compartir" este libro.
3. El usuario decide compartir este libro por correo electrónico con Gmail, por lo que se abre la aplicación de Gmail directamente en la pantalla para redactar un correo. El usuario ha navegado a una aplicación distinta, en consecuencia, la navegación ancestral ya no le conducirá a la pantalla principal de la Play Store, sino a la bandeja de entrada de Gmail. Sin embargo, el usuario puede navegar de vuelta a la Play Store a través de la navegación temporal. Si presiona el botón *Back* será llevado de vuelta a la pantalla de detalles del libro que está compartiendo.

PREGUNTA 9

FUENTE: Tema 4

Un panel de navegación lateral es un elemento visual oculto por defecto que agrupa varias de las secciones de la aplicación a las que el usuario necesita navegar. El patrón **Navigation Drawer** lo implementa como una lista vertical que se despliega y oculta al realizar gestos de desplazamiento horizontal a izquierda y a derecha, respectivamente. Es especialmente útil cuando no hay sitio suficiente en la interfaz para implementar pestañas o el número de secciones es grande.

Un ejemplo del mundo real que implementa este patrón es la aplicación de Gmail



Se utiliza para ofrecer diferentes puntos de vista de alto nivel de la bandeja de entrada filtrada por las etiquetas y para ofrecer accesos directos a funciones menos frecuentes: los ajustes y la ayuda, así como a aplicaciones relacionadas: contactos y calendario.

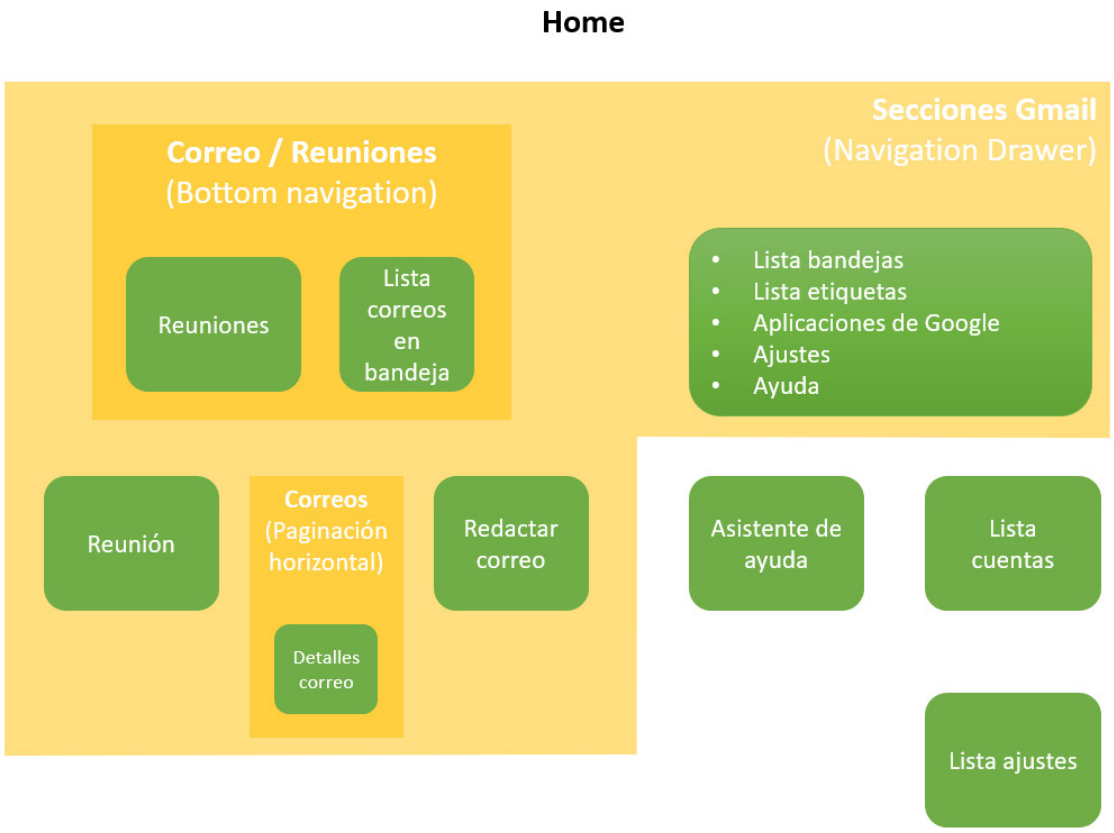
PREGUNTA 10

FUENTES: Tema 4

1. Identificar las pantallas y sus relaciones

- **Lista de correos en la bandeja seleccionada**. Muestra la lista de correos en la bandeja seleccionada. Incluye un botón para redactar un correo.
- **Detalles correo**. La vista del contenido de un correo. Con accesos directos para archivar, etiquetas, borrar y responder al correo.
- **Redactar correo**. Para redactar un nuevo correo o una respuesta a un correo.
- **Reunión**. Vista de una reunión en curso
- **Reuniones**. Opciones de inicio / hospedado de una reunión
- **Secciones Gmail**. Secciones principales de la aplicación: lista de bandejas, lista de etiquetas, otras aplicaciones de Google relacionadas (Calendar y Contactos), ajustes y ayuda. Disponible desde casi toda la aplicación.
- **Lista de cuentas**. Selector de cuentas de correo configuradas en el dispositivo para modificar sus preferencias
- **Lista de ajustes**. Lista con todos los ajustes disponibles para una cuenta de correo.
- **Asistente de ayudas**. Un solucionador de problemas de Gmail al estilo de una FAQ

2. Proporcionar navegación Descendente y Lateral.



3. Proporcionar navegación ancestral y temporal. Juntar todo