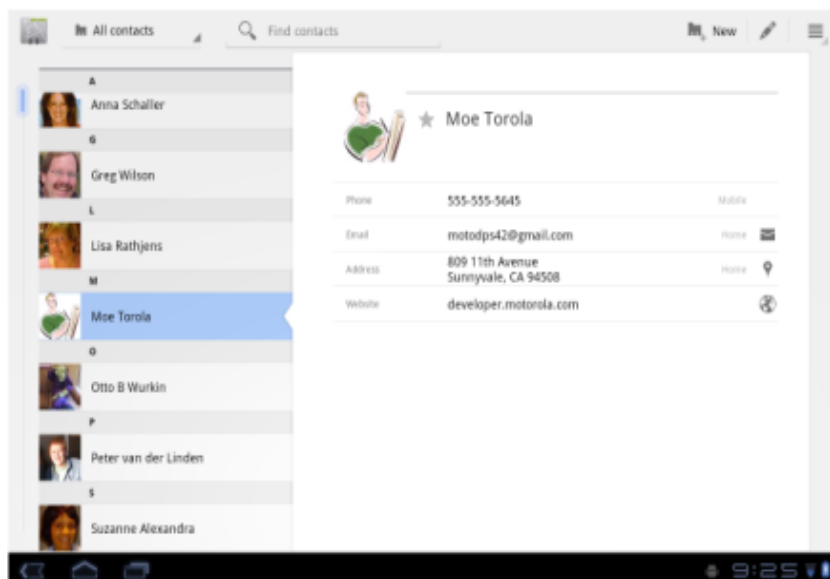




# Fragmentos





## Contenidos

Necesidad de fragmentos .....	3
Definición .....	3
Clase Fragment.....	4
Librerías de compatibilidad.....	4
Añadir fragmento a la actividad .....	6
Práctica con fragmentos estáticos (P_40_Fragmentos_01).....	6
Práctica con fragmentos dinámicos (P_41_Fragmentos_02) .....	10
Manejando eventos con respuesta en el propio fragmento (P_42_Fragmentos_03) .....	12
Manejando eventos con respuesta en otro fragmento (Fragmentos_04) .....	13
Número de fragmentos dependiendo de la orientación (P_44_Fragmentos_05).....	15



## Necesidad de fragmentos

La necesidad de usar fragmentos nace con la versión 3.0 (API 11) de Android debido a los múltiples tamaños de pantalla que estaban apareciendo en el mercado y a la capacidad de orientación de la interfaz (*Landscape* y *Portrait*). Estas características necesitaban dotar a las aplicaciones Android de la capacidad para adaptarse y responder a la interfaz de usuario sin importar el dispositivo.

Hoy en día, su uso es más necesario incluso: Imaginemos que queremos desarrollar una aplicación para diversos dispositivos Android existentes (las pulgadas y formas son orientativas):

- Reloj (1 a 2 pulgadas, cuadrado)
- Smartphone o dispositivo GPS (3 a 5 pulgadas, rectangular alargado)
- Phablet (5 a 7 pulgadas, rectangular alargado)
- Tablet (7 a 12 pulgadas, proporción aurea)
- Pantalla de un ordenador (14 a 30 pulgadas, rectangular alargado o proporción aurea)
- Televisión (30 a 80 pulgadas, proporción aurea)
- Proyector (80 a 300 pulgadas, proporción aurea)

Como buenos desarrolladores, sabemos que debemos conseguir siempre el mejor aprovechamiento del espacio de cada pantalla (pensando en tamaño, forma u orientación) y que no podemos olvidarnos de la comodidad del usuario.

Podríamos haberlo hecho implementado diferentes actividades con diferentes layouts para cada configuración de pantalla, pero esto nos habría obligado a duplicar gran parte del código en cada actividad.

Los Fragmentos nos proporcionan la modularidad y reusabilidad que necesitamos ya



que nos permiten segmentar la aplicación de tal modo que podemos crear aplicaciones únicas que se comporten de manera diferente en función del tamaño o la forma del dispositivo sobre el que se ejecuten o de la orientación del mismo., reusando código y

ahorrando tiempo de diseño a la hora de desarrollar una aplicación.

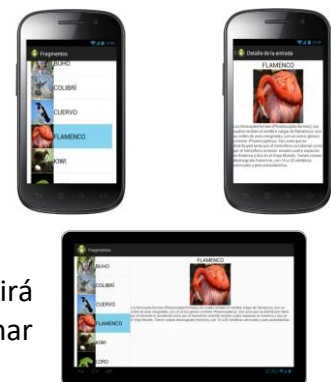
Así podremos dar una solución visual para teléfonos, otra para tabletas, otra para relojes, etc.

## Definición

Un *fragment* podría definirse como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de la actividad de forma independiente al resto de elementos de la actividad.

Por lo tanto tendremos la posibilidad de combinar múltiples fragmentos en una sola actividad o incluso reutilizar fragmentos en otras actividades.

Cada fragmento tendrá su propio diseño, ciclo de vida, recibirá sus propios eventos de entrada y se podrá agregar o eliminar mientras la actividad de acogida esté en marcha.





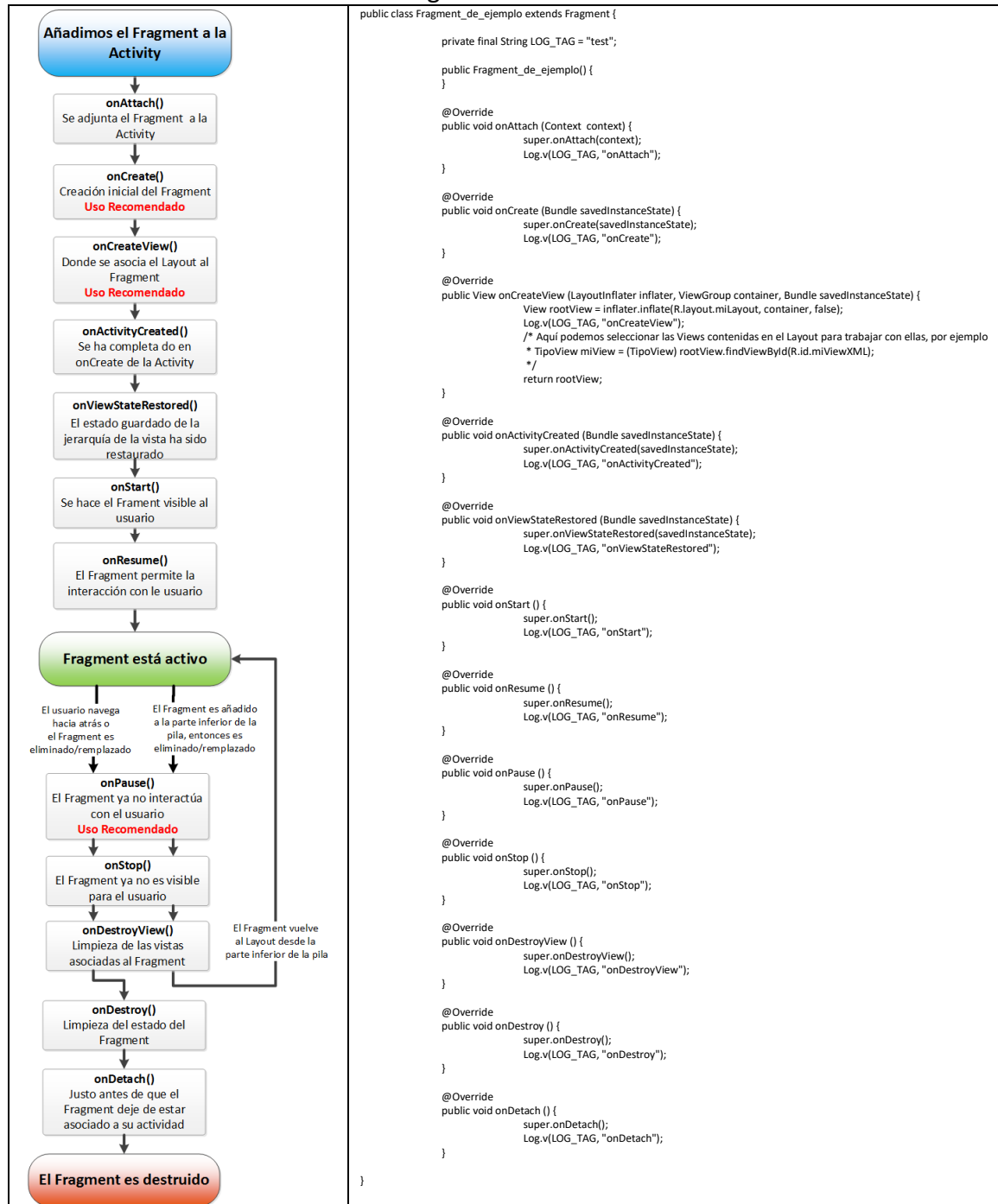


## Ciclo de vida

Se puede decir que un fragmento es una sección de una actividad, que tiene su propio ciclo de vida, recibe sus propios eventos de entrada y que se puede agregar o quitar mientras que la actividad está en marcha (y que además puede reutilizarse en diferentes actividades).

Un Fragment siempre ha de estar en una Activity, no puede existir independiente, siempre debe estar integrado en el ciclo de vida de una actividad y, por tanto, está directamente afectado por dicho ciclo (cuando la actividad se destruye, sus Fragments también serán destruidos).

Los métodos del ciclo de vida del fragmento son:





Observaciones:

- Si el fragmento es una subclase de ListFragment, la implementación predeterminada de dicha subclase devuelve ya el ListView asociado por layout por lo que no es necesario implementar el código del método onCreateView().
- La API 23 ha cambiado el parámetro del método inAttach para que pase a ser de tipo Context, en muchos textos te encontrarás el viejo de tipo Activity y AS lo marcará como obsoleto
- En cualquiera de los métodos tendremos acceso a la actividad contenedora de un fragmento mediante la llamada al método **getActivity()**.

## Añadir fragmento a la actividad

Puesto que los fragmentos no son componentes de una aplicación, éstos no deben ser declarados en el Manifest.

Entonces, cómo añadimos un fragmento a una actividad?

Hay dos opciones:

1. De manera **estática**: Declarando el fragmento dentro del layout de la actividad. El fragmento no podrá ser eliminado o sustituido por nada o tendremos errores.
2. De manera **dinámica**: Agregando en tiempo de ejecución (mediante programación en el código de la clase de la actividad) el fragmento a una vista de tipo ViewGroup (se recomienda el uso de FrameLayout) existente en el layout de la actividad. El fragmento sí que se podrá eliminar o sustituir por otro fragmento u otro contenido.

## Práctica con fragmentos estáticos (P\_40\_Fragmentos\_01)

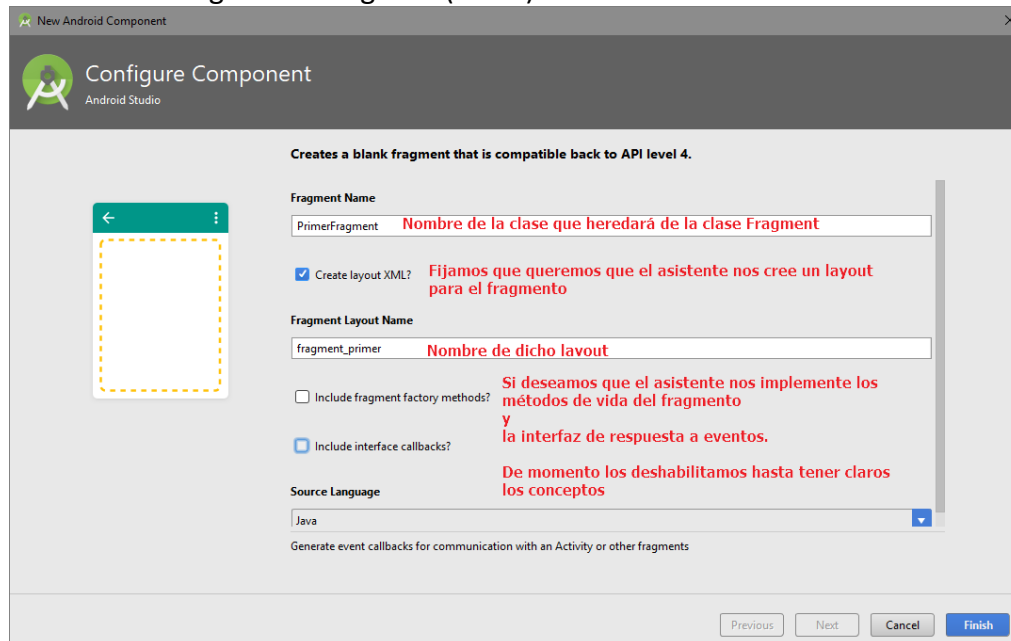
Deseamos una aplicación "muy básica" que usando fragmentos estáticos consiga que para orientación vertical cuente con una pantalla con un TextView con el típico "Hola" y para orientación horizontal con 1/3 de la pantalla con un TextView con la fecha del día y el resto con una imagen.

Crea el proyecto Fragmentos\_1 desde la plantilla Basic Activity, **con la opción "Use a fragment" desactivada**. Aunque AS nos proporciona asistente para la creación de fragmentos desde el inicio de creación del proyecto, te aconsejo que hasta que no tengas claros los conceptos no lo utilices.



### Fijando la configuración en vertical:

Para crear el primer fragmento (que contiene un TextView) sí que podemos usar el asistente: New->Fragment->Fragment(Blank)



### Abre los ficheros que se han creado para estudiar su contenido:

- Clase: PrimerFragmento.java

```
public class PrimerFragment extends Fragment {  
  
    public PrimerFragment() {  
        // Required empty public constructor  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.fragment_primer, container, false);  
    }  
}
```

El único método que nos ha proporcionado el asistente es el de crear la vista del fragmento:

Al sobrescribir el método onCreateView() observa que de serie da la posibilidad de utilizar un LayoutInflater, un ViewGroup y un Bundle:

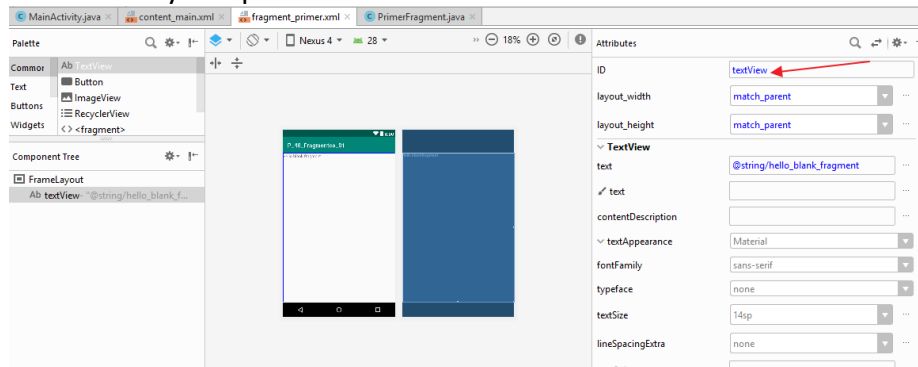
- El LayoutInflater normalmente se utiliza para inflar el layout del fragmento.
- El ViewGroup será la vista padre donde se insertará el layout del fragmento.
- Y el Bundle puede utilizarse para recuperar datos de una instancia anterior del fragmento.

El método inflate() toma tres parámetros:

- En el primero se indica el id del layout del fragmento.
- En el segundo se fija la vista que contiene al fragmento.
- Y por último un booleano que sirve para indicar si el inflado del layout debe ser insertado en el ViewGroup (En este caso es false porque directamente se está insertando el layout en el ViewGroup).

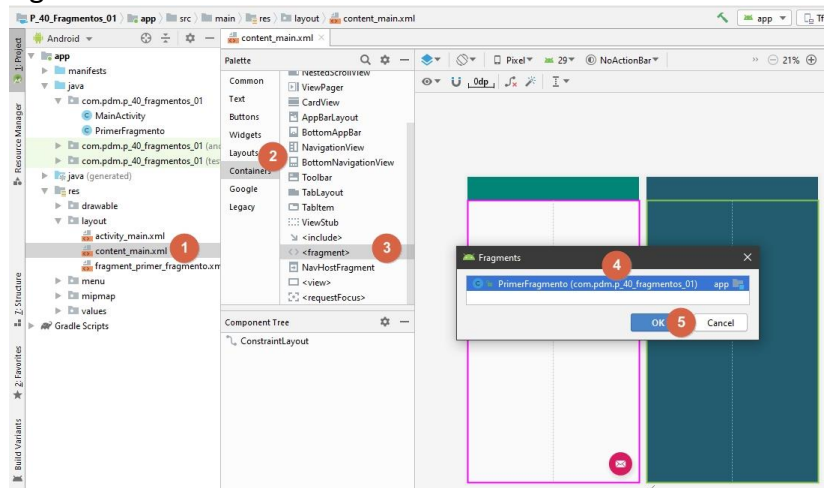


- Layout del fragmento: fragment\_primer.xml  
Un `FrameLayout` que contiene un `TextView`. Identifícalo como `textView`:

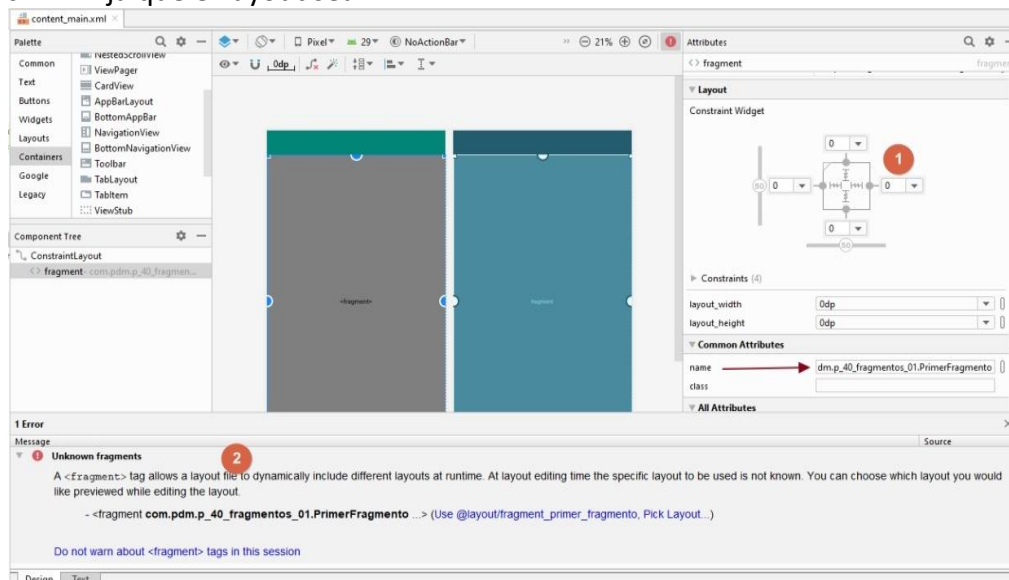


Fijando la interfaz gráfica en vertical:

Modifica el fichero `content_main.xml`, eliminando el `TextView` y declarando de manera estática el fragmento:



Fija los atributos ancho y alto para que sean los de su vista contenedora y desde "QuickFix" fija que el layout sea :



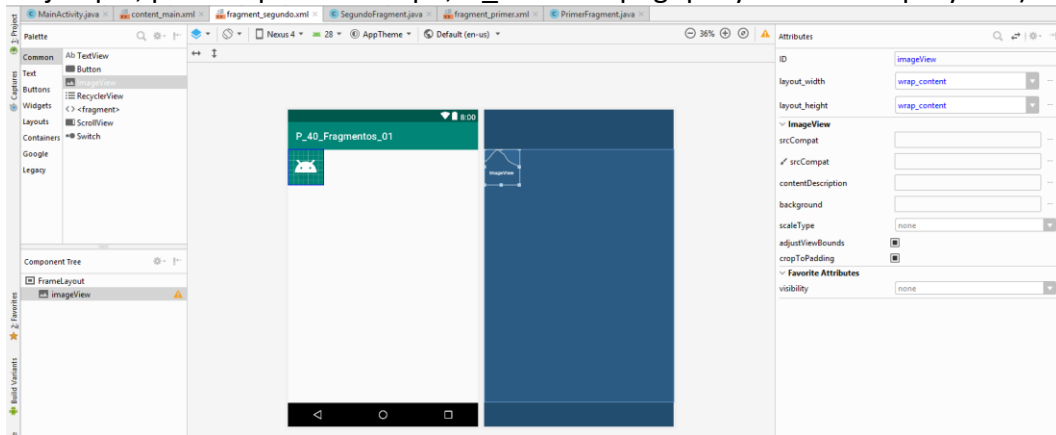
Observa que se añadido un atributo **name** con la forma `paquete.nombre_clase`:  
`android:name="com.pdm.ejercicios.p_40_fragmentos_01.PrimerFragmento"`





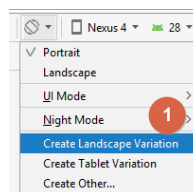
### Preparando la interfaz gráfica en horizontal:

Repite los pasos para crear un segundo fragmento (el que contiene la imagen) llamado SegundoFragment. El layout debe contener un ImageView con la imagen que prefieras (por ejemplo, para no perder tiempo, ic\_launcher.png que ya está en el proyecto).

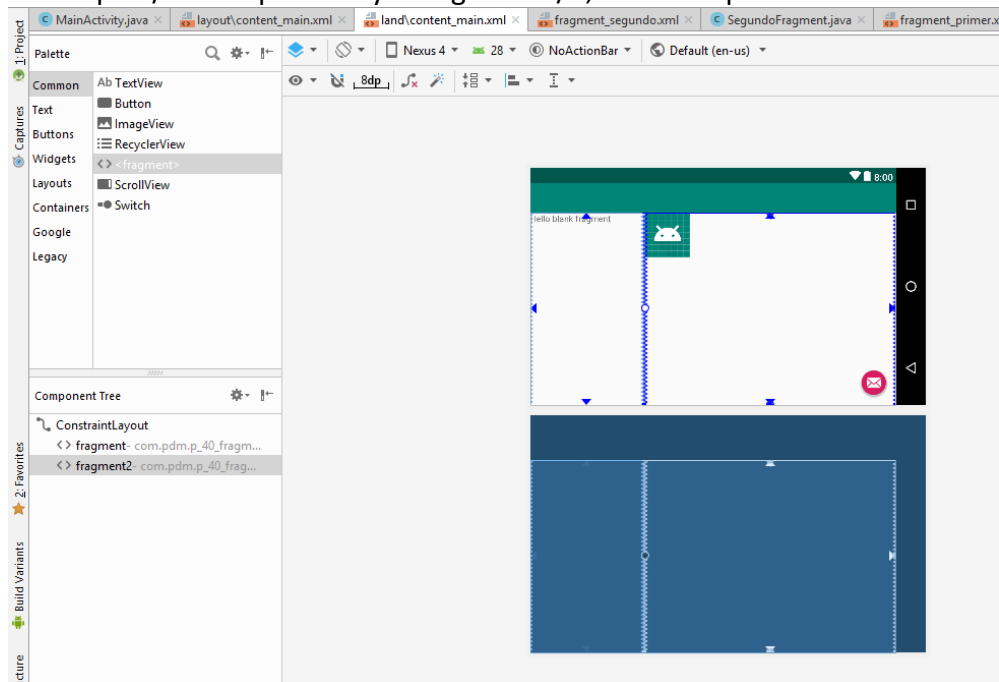


### Fijando la configuración en horizontal:

Creamos la variación del layout content\_main para orientación horizontal (recuerda que la manera más rápida es:



Modificamos la variación para añadir el segundo fragmento: Como queremos que el primero ocupe 1/3 de la pantalla y el segundo 2/3, el diseño puede ser:



(para conseguir que se "vean" los fragmentos haz caso a las sugerencias de la herramienta Lint)



Si ejecutamos la aplicación, los fragmentos funcionan pero el TextView siempre es el mismo. La clase MainActivity debe fijar la fecha si hay dos fragmentos. Cómo saber si hay dos fragmentos o solo uno? En el método onCreate() de la actividad:

- Inicializamos un booleano a falso:

```
boolean dosPaneles=false;
```

- Buscamos si existe la vista del segundo fragmento para cambiar el valor del booleano:

```
if (findViewById(R.id.fragment2)!=null)  
    dosPaneles=true;
```

- Si hay dos fragmentos, cambiamos el contenido del TextView para que sea la fecha actual:

```
if (dosPaneles){  
    TextView textView= findViewById(R.id.textView);  
    long ahora = System.currentTimeMillis();  
    Date fecha = new Date(ahora);  
    DateFormat df = DateFormat.getDateInstance();  
    String salida = df.format(fecha);  
    textView.setText(salida);  
}
```

Ejecuta la aplicación y comprueba que ya sabes trabajar con fragmentos estáticos.

A partir de ahora, si al escoger Basic Activity seleccionas "Use a fragment", el asistente de AS crea el layout y la clase del fragmento estático que nosotros hemos llamado fragment\_primer.xml y PrimerFragment.java y que el asistente llama fragment\_main.xml y MainActivityFragment.java.

## Práctica con fragmentos dinámicos (P\_41\_Fragmentos\_02)

Deseamos una aplicación "muy básica" que usando fragmentos dinámicos consiga que para orientación vertical cuente con una pantalla con el típico "Hola mundo" y para orientación horizontal con una pantalla con una imagen.

Creamos el proyecto P\_41\_Fragmentos\_02 y los dos fragmentos (llamados VerticalFragment y HorizontalFragment) de manera similar a la práctica anterior, además modificamos content\_main eliminando el TextView y añadiendo un FrameLayout con id (por ejemplo, miLayout).



### Trabajar con fragmentos dinámicos en la clase Activity:

Un concepto relevante es que existe una interfaz administradora de fragmentos llamada [FragmentManager](#).

También es importante entender que un fragmento puede ser añadido, reemplazado y eliminado en tiempo de ejecución. A esas tres operaciones se les llama Transacciones y en Android una transacción es representada con la clase [FragmentTransaction](#).

En el método onCreate() de la actividad debemos añadir el fragmento del TextView o el de la imagen.

Los pasos a efectuar son:

- Obtener la orientación del dispositivo: Usando el valor del campo *orientation* de la clase [Configuration](#) (guarda información de la configuración del dispositivo) a la que se accede con el método getConfiguration() de la clase Resources:

```
int orientacion=getResources().getConfiguration().orientation;
```

- Dependiendo de la orientación, crear el fragmento adecuado y añadirlo:

```
Fragment fragmentInsertado;  
if (orientacion== Configuration.ORIENTATION_PORTRAIT) {  
    fragmentInsertado = new VerticalFragment();  
}  
else{  
    fragmentInsertado = new HorizontalFragment();  
}
```

Observa que las clases de los fragmentos creados por el asistente utilizan la librería de soporte de compatibilidad, por tanto, usala también en MainActivity

- Usando la librería de compatibilidad androidx.fragment.app.FragmentManager, obtener la instancia del administrador de fragmentos que nos servirá para manejar los fragmentos

```
FragmentManager fragmentManager = getSupportFragmentManager();
```


- Crear una nueva transacción usando el método beginTransaction() del administrador de fragmentos:

```
FragmentTransaction fragmentTransaction=fragmentManager.beginTransaction();
```

- Añadir el fragmento

```
fragmentTransaction.add(R.id.miLayout,fragmentInsertado);
```

El método add() recibe dos parámetros. El primero es el identificador del contenedor donde insertaremos el fragmento (en nuestro caso es el FrameLayout de la actividad) y el segundo es el fragmento a insertar. Algunos de los métodos para transacciones de fragmentos son:

- **add()**: para agregar un fragmento
- **remove()**: para suprimir un fragmento
- **replace()**: para reemplazar un fragmento
- **addToBackStack()**: permite que el usuario vuelva al estado anterior del fragmento, cuando pulsa .

- Por último ordenamos que los cambios surtan efecto mediante el método commit() de la clase FragmentTransaction:

```
fragmentTransaction.commit();
```

- Todas las líneas de código anteriores obviamente podrían reemplazarse por:

```
getSupportFragmentManager().beginTransaction().add(R.id.miLayout,fragmentInsertado).commit();
```

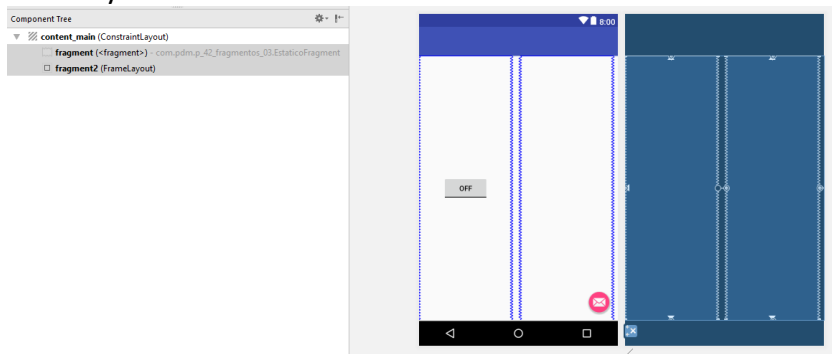
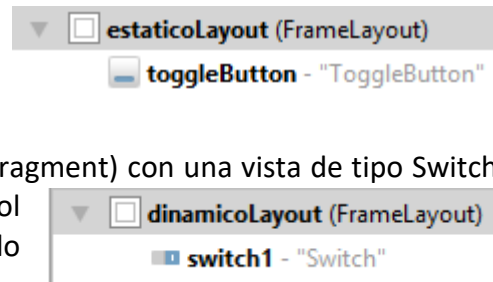
Ejecuta para comprobar el funcionamiento. Si giras el dispositivo observarás que los fragmentos se van "amontonando" uno encima de otro. Para solucionarlo cambia el método **add** por **replace**.



## Manejando eventos con respuesta en el propio fragmento (P\_42\_Fragmentos\_03)

Crea un proyecto llamado P\_42\_Fragmentos\_03 que tiene:

- un fragmento estático (llamado por ejemplo EstaticoFragment) con una vista de tipo ToggleButton dentro (así de paso conocemos este control que usaremos para cambiar el color de fondo del fragmento)
- un fragmento dinámico (llamado DinamicoFragment) con una vista de tipo Switch dentro (así de paso conocemos este control que usaremos para cambiar el color de fondo del fragmento).
- Modificamos convenientemente content\_main para que cada fragmento ocupe la mitad de su contenedor (**hay que identificar los componentes que se necesitarán en el código**). Recuerda que el estático es un fragmento y el dinámico un framelayout



- Creamos la transacción en la clase MainActivity para el fragmento dinámico (si ejecutas comprobarás que funciona pero que los controles no sirven para nada!):

```
Fragment fragment=new DinamicoFragment();  
getSupportFragmentManager().beginTransaction().add(R.id.fragment2,fragment).commit();
```



En la clase EstaticoFragment hay que instanciar el layout para poder cambiar su color, instanciar el ToogleButton y asignarle listener, pero no podemos utilizar directamente el conocido método findViewById(R.id.pepito) ya que el único que puede referenciar a todos los Views de un fragmento es el View padre del fragmento. Dicho view es obtenido cuando se infla el layout.

Para tener una referencia a dicho layout simplemente crearemos una instancia de tipo View y asignaremos el resultado al inflar el código XML. Luego invocaremos el método findViewById() desde la nueva instancia.

Importantísimo: no olvides cambiar el return por defecto para que devuelva esa instancia de la vista o no funcionará:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_estatico, container, false);
    final ToggleButton toggleButton = view.findViewById(R.id.toggleButton);
    final FrameLayout frameLayout = view.findViewById(R.id.estaticoLayout);
    toggleButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (toggleButton.isChecked())
                frameLayout.setBackgroundColor(ContextCompat.getColor(getActivity(), R.color.colorPrimary));
            else
                frameLayout.setBackgroundColor(ContextCompat.getColor(getActivity(), R.color.colorAccent));
        }
    });
    return view;
}
```

Los cambios en el SwitchFragment son similares salvo que el listener asignado es setOnCheckedChangeListener. Ejecuta la aplicación para comprobar que sabes manejar eventos dentro del propio fragmento. Qué ocurre al girar el dispositivo? ¿Por qué?

## Manejando eventos con respuesta en otro fragmento (Fragmentos\_04)

Cómo conseguir que el evento se produzca en un fragmento y la acción se realice en otro? Probemos.

Un fragmento no tiene por qué conocer la existencia de ningún otro, es más, deberían diseñarse de tal forma que fueran lo más independientes posible, de forma que puedan reutilizarse en distintas situaciones sin problemas de dependencias con otros elementos de la interfaz.

Dado que los fragmentos no se relacionan entre sí directamente, lo deben hacer a través de interfaces (será la actividad la que implemente los métodos de respuesta a los eventos del fragmento); necesitamos hacer uso de callbacks (repasa Java de 1º!).

Un callback nos permite implementar una interfaz con nuestra actividad y obtener una instancia de esa actividad implementada en nuestro fragmento y que sea este objeto quien llame al método que pertenece a la actividad.

Haz una copia Fragmentos\_03 llamada Fragmentos\_04, abre desde AS y refactoriza para borrar cualquier referencia Fragmentos\_03.



En la clase EstaticoFragment creamos un evento personalizado de la siguiente forma:

- declaramos una variable de tipo OnBotonTocadoListener, éste es nuestro "escuchador" del evento para la actividad:

```
OnBotonTocadoListener miListener;
```

- declaramos la interface (con un único método que se encargará de enviar el evento y los datos del fragmento a la actividad):

```
public interface OnBotonTocadoListener {  
    void onBotonTocado(int color);  
}
```

- En el método onCreateView creamos una instancia al listener que implementaremos en la clase MainActivity:

```
try {  
    miListener = (OnBotonTocadoListener) getActivity();  
} catch (ClassCastException e) {  
    throw new ClassCastException(getActivity().toString() + " falta implementar listener Toogle");  
}
```

(Por cierto, la gestión de la excepción con try/catch es para asegurarnos que la Activity implementa el escuchador)

- y cambiamos el método onClick y le aplicamos el método de la interface indicando como argumento los datos que se enviarán

```
@Override  
public void onClick(View v) {  
    int colorNuevo;  
    if (toggleButton.isChecked())  
        colorNuevo = R.color.colorPrimary;  
    else  
        colorNuevo = R.color.colorAccent;  
    miListener.onBotonTocado(colorNuevo);  
}
```

En la clase MainActivity debemos implementar la interfaces y gestionarlas:

```
public class MainActivity extends AppCompatActivity implements EstaticoFragment.OnBotonTocadoListener,  
DinamicoFragment.OnSwitchTocadoListener {  
    ...  
    @Override  
    public void onSwitchTocado(boolean chequeado) {  
        FrameLayout frameLayout = findViewById(R.id.fragment);  
        if (chequeado)  
            frameLayout.setBackgroundColor(ContextCompat.getColor(getApplicationContext(), R.color.colorPrimary));  
        else  
            frameLayout.setBackgroundColor(ContextCompat.getColor(getApplicationContext(), R.color.colorAccent));  
    }  
    @Override  
    public void onBotonTocado(int color) {  
        FrameLayout frameLayout = findViewById(R.id.fragment2);  
        frameLayout.setBackgroundColor(ContextCompat.getColor(getApplicationContext(), color));  
    }  
}
```

Observa que DinamicoFragment también debe fijar una interface de nombre onSwitchTocadoListener que funciona de manera semejante (en vez de enviar el entero que representa al color se envía el booleano que representa si está chequeado o no). Tienes que implementar los cambios (no hay recorta y pega!)

Ejecuta la aplicación y comprueba que funciona.

Gira el dispositivo y hemos vuelto al principio! Necesitamos mejoras. En otro ejercicio las implementaremos.



## Número de fragmentos dependiendo de la orientación (P\_44\_Fragmentos\_05)

Nuestro proyecto Fragmentos\_5 es para una aplicación que dependiendo de la provincia escogida (Zaragoza, Huesca o Teruel) mostrará una foto y una descripción de la ciudad. El número de actividades dependerá de la orientación (en vertical 2 y en horizontal 1).

Creamos el proyecto (partiendo de la plantilla Basic Activity), le añadimos dos fragmentos llamados SelectorFragment (con layout fragment\_selector.xml) y DetalleFragment (con layout fragment\_detalle.xml) y usando el asistente creamos la segunda actividad llamada DetalleActivity (con layouts activity\_detalle.main y content\_detalle.xml, si hemos partido de la plantilla Basic Activity).

Modificamos los layouts proporcionados por el asistente para que sean los siguientes:  
Layout's de los fragmentos:

fragment_selector.xml	fragment_detalle.xml

Layout's de las actividades:

content_detalle.xml	content_main.xml	land/content_main.xml
		<p>Observa que hemos fijado el id del FrameLayout con el mismo nombre que en content_detalle</p>

Mejoras con respecto a la práctica anterior:

- Solo se han identificado los elementos que deben ser referenciados en el código de las clases.
- El id del contenedor del fragmento dinámico (FragmentDetalle) es **content\_detalle** tanto en land/content\_main.xml como en content\_detalle.xml para facilitarnos la optimización de código java.



### Cambios en la clase SelectorFragment:

El cambio más importante es el cambio del momento en que se hace la instanciación del listener. En ejemplos anteriores lo hacíamos en el método onCreateView, pero tal y como aconseja Android Developers debería hacerse en el momento en el que el fragmento se adhiere a la actividad, es decir en el método onAttach.

Otro cambio (menor) es la implementación del método onDetach() que se llama cuando el fragmento deja de estar asociado a la actividad, lo empleamos para vaciar el listener. Entendido todo lo anterior, el código de nuestra clase será:

```
public class SelectorFragment extends Fragment {

    public SelectorFragment() {
        // Required empty public constructor
    }

    public interface OnCambiosListener{
        void onCambios(int seleccionado);
    }

    OnCambiosListener mListener;

    @Override
    public void onAttach (Context context) {
        super.onAttach(context);
        try {
            mListener = (OnCambiosListener) getActivity();
        } catch (ClassCastException e) {
            throw new ClassCastException(getActivity().toString() + " falta implementar listener");
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view= inflater.inflate(R.layout.fragment_selector, container, false);
        RadioGroup radiogroup= view.findViewById(R.id.horizontal);
        radiogroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId) {
                mListener.onCambios(checkedId);
            }
        });
        return view;
    }
}

@Override
public void onDetach () {
    super.onDetach();
    mListener=null;
}
}
```





### Cambios en clase MainActivity:

El más importante es que al seleccionar una provincia:

- si tiene un solo fragmento tiene que enviar el dato a la segunda actividad
- si tiene dos tiene que cargar el segundo fragmento con los datos de dicha provincia

Por tanto, en ambos casos creamos un bundle que recoja ese dato ya sea:

- para añadirlo al intent implícito que llama a DetalleActivity:  
intent.putExtras(bundle);
- para enviárselo a la clase DetalleFragmento:  
detalleFragment.setArguments (bundle);

```
public class MainActivity extends AppCompatActivity implements
SelectorFragment.OnCambiosListener{

    private boolean dosFragmentos;
    private static final String ID_CLAVE="chequeado";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        if (findViewById(R.id.content_detalle)!=null)
            dosFragmentos=true;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ...
    }

    @Override
    public void onCambios(int seleccionado) {
        Bundle bundle=new Bundle();
        bundle.putInt(ID_CLAVE, seleccionado);
        if (dosFragmentos){
            DetalleFragment detalleFragment=new DetalleFragment();
            detalleFragment.setArguments(bundle);
            getSupportFragmentManager().beginTransaction().replace(R.id.content_detalle,detalleFragment).co
mmmit();
        }
        else{
            Intent intent=new Intent(this,DetalleActivity.class);
            intent.putExtras(bundle);
            startActivity(intent);
        }
    }
}
```



### Cambios en la clase DetalleFragment

El fragmento debe recoger los datos enviados en un bundle, el método más indicado es onCreate():

```
public class DetalleFragment extends Fragment {

    private static final String ID_CLAVE = "chequeado";
    private int seleccionado;
    private Context contexto;

    public DetalleFragment() {
        // Required empty public constructor
    }

    @Override
    public void onAttach (Context context) {
        super.onAttach(context);
        contexto=context;
    }

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        Bundle recibidos=getArguments();
        seleccionado=recibidos.getInt(ID_CLAVE);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view= inflater.inflate(R.layout.fragment_detalle, container, false);
        String texto=null;
        Drawable imagen=null;
        switch (seleccionado){
            case R.id.radioButton:
                texto=getString(R.string.zaragoza);
                imagen= ContextCompat.getDrawable(contexto,R.drawable.zaragoza);
                break;
            case R.id.radioButton2:
                texto=getString(R.string.huesca);
                imagen= ContextCompat.getDrawable(contexto,R.drawable.huesca);
                break;
            case R.id.radioButton3:
                texto=getString(R.string.teruel);
                imagen= ContextCompat.getDrawable(contexto,R.drawable.teruel);
                break;
        }
        TextView textView= view.findViewById(R.id.textView);
        textView.setText(texto);
        ImageView imageView= view.findViewById(R.id.imageView);
        imageView.setImageDrawable(imagen);
        return view;
    }
}
```



### Cambios en la clase DetalleActivity:

Las consideraciones a tener en cuenta son:

- La actividad solo debe existir si el dispositivo está en vertical, un cambio de su orientación debe hacer que se finalice y se vuelva a MainActivity con dos fragmentos.
- El título de la actividad queremos "customizarlo" y que no sea DetalleActivity sino el de la provincia a visualizar.
- Hay que hacer uso de la clase detalleFragmento.

```
public class DetalleActivity extends AppCompatActivity {

    private static final String ID_CLAVE = "chequeado";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE) {
            finish();
            return;
        }
        setContentView(R.layout.activity_detalle);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });
        Bundle bundle = getIntent().getExtras();
        int datoRecibido = bundle.getInt(ID_CLAVE);
        String texto = null;
        switch (datoRecibido) {
            case R.id.radioButton:
                texto = getString(R.string.zaragoza);
                break;
            case R.id.radioButton2:
                texto = getString(R.string.huesca);
                break;
            case R.id.radioButton3:
                texto = getString(R.string.teruel);
                break;
        }
        if (getSupportActionBar() != null) {
            getSupportActionBar().setDisplayHomeAsUpEnabled(true);
            getSupportActionBar().setTitle(texto);
        }
        DetalleFragment detalleFragment = new DetalleFragment();
        detalleFragment.setArguments(bundle);
        getSupportFragmentManager().beginTransaction().replace(R.id.content_detalle,
detalleFragment).commit();
    }
}
```

Ejecuta la aplicación



### Mejoras en DetalleActivity:

El botón Up finaliza la actividad y vuelve a MainActivity pero se pierde el "checked" en la selección de la provincia de la que habíamos partido (es decir, la actividad MainActivity se crea de nuevo).

La solución es fijar que en DetalleActivity el comportamiento del botón Up sea el de finalizar la actividad en curso:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == android.R.id.home) {
        finish();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Otra posible solución (en este caso) sería que el comportamiento fuese el mismo que el de pulsar el botón "Atrás", es decir, en vez de usar el método finish() usar el método onBackPressed().

### Otras posibles mejoras:

- Si la imagen y el texto del fragmento\_detalle se "apelotonan" puedes cambiar su tipo a RelativeLayout/ConstraintLayout.
- Si los textos contenidos en TextView no se ven por falta de espacio, una posible solución es meterlos dentro de ScrollView.
- Para aprender el concepto de fragmentos estamos trabajando con layouts horizontal y vertical. Recuerda que deberíamos pensar también en layout small, large, x-large, etc y sus posibles orientaciones. Con las imágenes tampoco estamos teniendo mucho cuidado, habría que pensar en las imágenes para distintas densidades (drawable-mdpi, drawable-hdpi, etc.)