

# Fragmentos

---

1TEL05 - Servicios y Aplicaciones para IoT

# Fragmentos

Google lo introdujo en Android 3.0 (Honeycomb - API 11), como una forma de modularizar las interfaces de usuario de las aplicaciones y hacerlas más flexibles, especialmente para dispositivos con diferentes tamaños de pantalla y orientaciones.

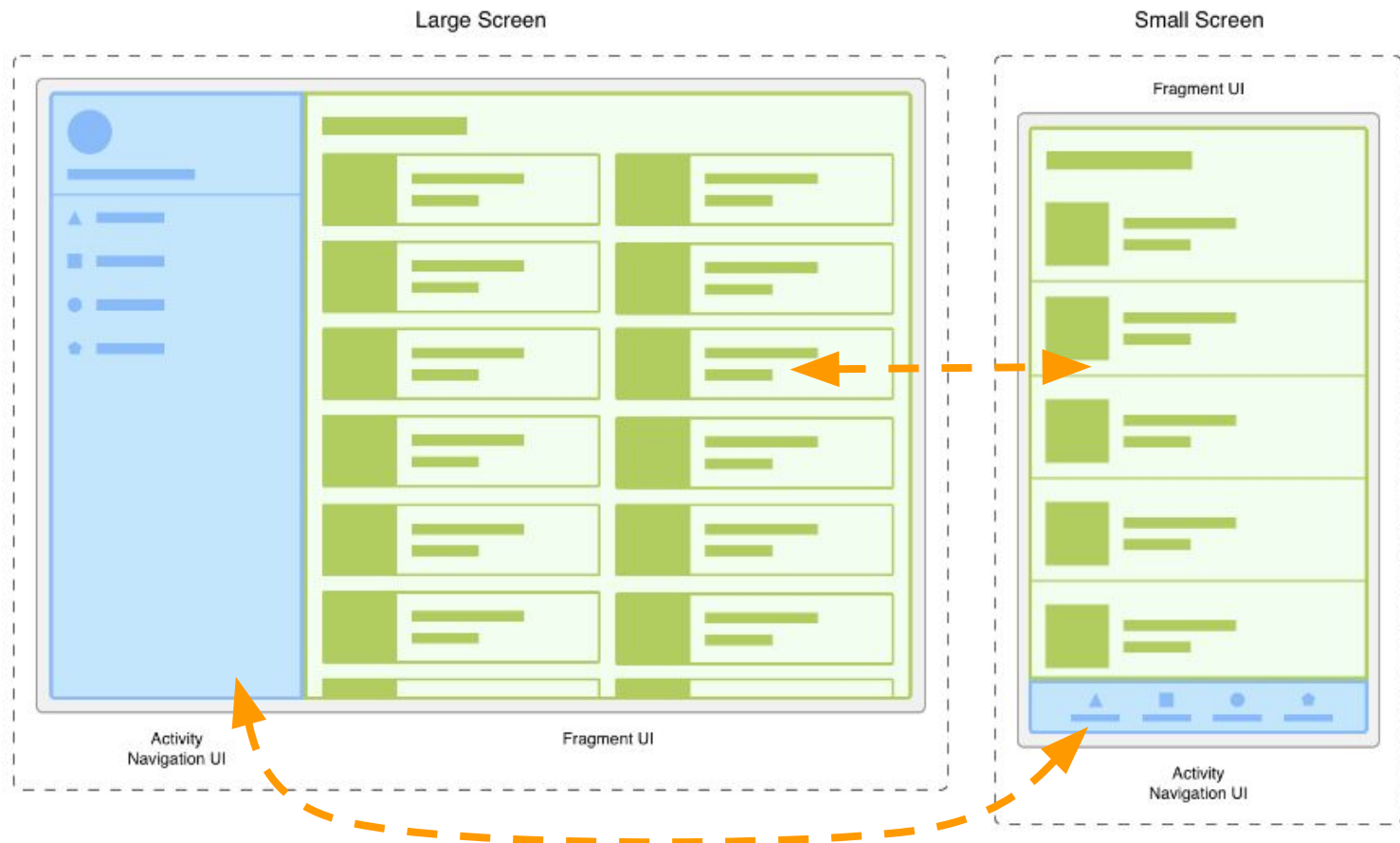
- Mayor flexibilidad en el diseño de la interfaz de usuario, especialmente para diferentes tamaños de pantalla y orientaciones.
- Reutilización de código para mayor eficiencia.
- Gestión simplificada del ciclo de vida de la interfaz de usuario.
- Mejor soporte para tablets y dispositivos de pantalla grande.

# ¿Qué es un fragmento?

- Es un componente autocontenido que tiene su propia interfaz de usuario (UI) y su propio ciclo de vida, el cual puede ser reutilizado en diferentes parte de una aplicación.
- Un fragmento recibe sus propios eventos, como presión, clicks, entre otros.
- Un fragmento siempre está vinculado a la actividad que lo crea.
- Así mismo, puede estar siempre presente en la actividad ocupando un espacio o puede ser agregado y borrado dinámicamente de la actividad.

# Modularidad

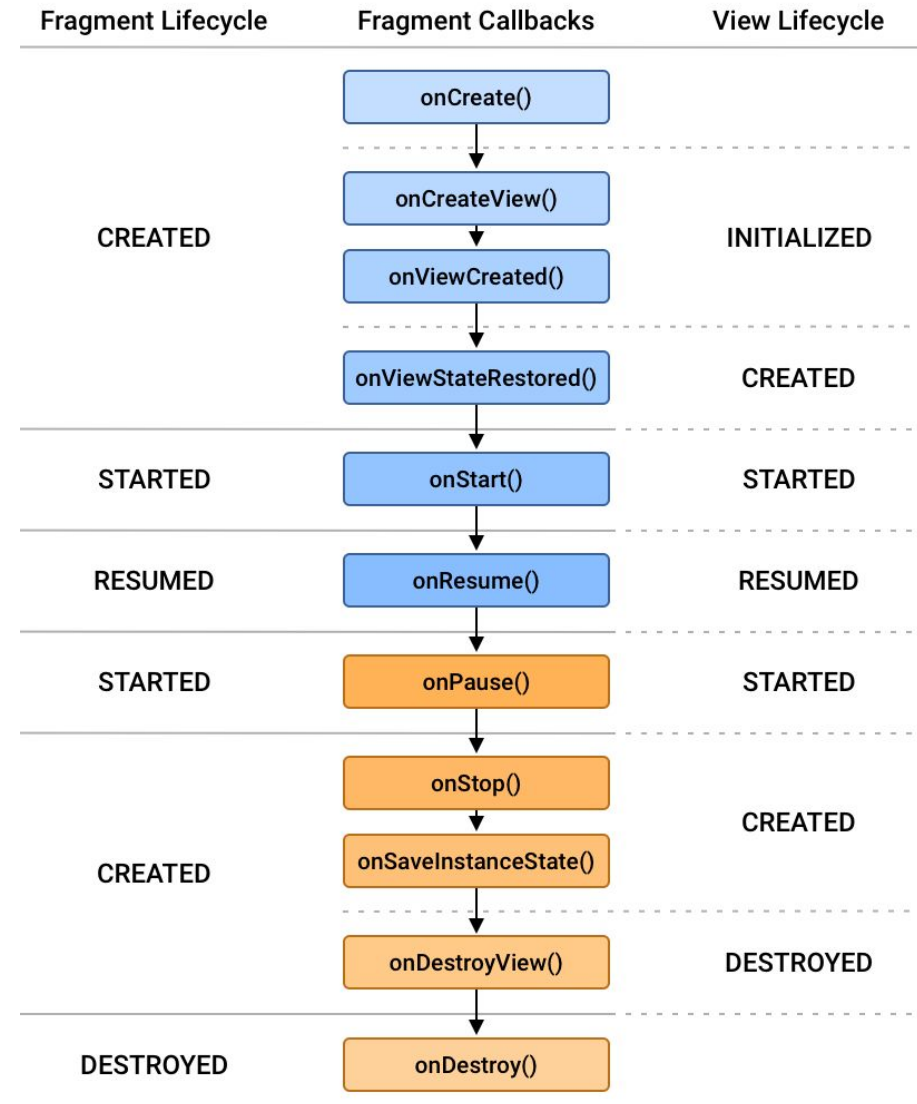
Los fragmentos permiten dividir la pantalla en pedazos autónomos entre sí pero gestionados por una misma actividad.



# Ciclo de vida de un fragmento

Al igual que una actividad, los fragmentos tienen su propio ciclo de vida, desde que son creados hasta su destrucción.

<https://developer.android.com/guide/fragments/lifecycle?hl=es-419>



# Crear Fragmentos

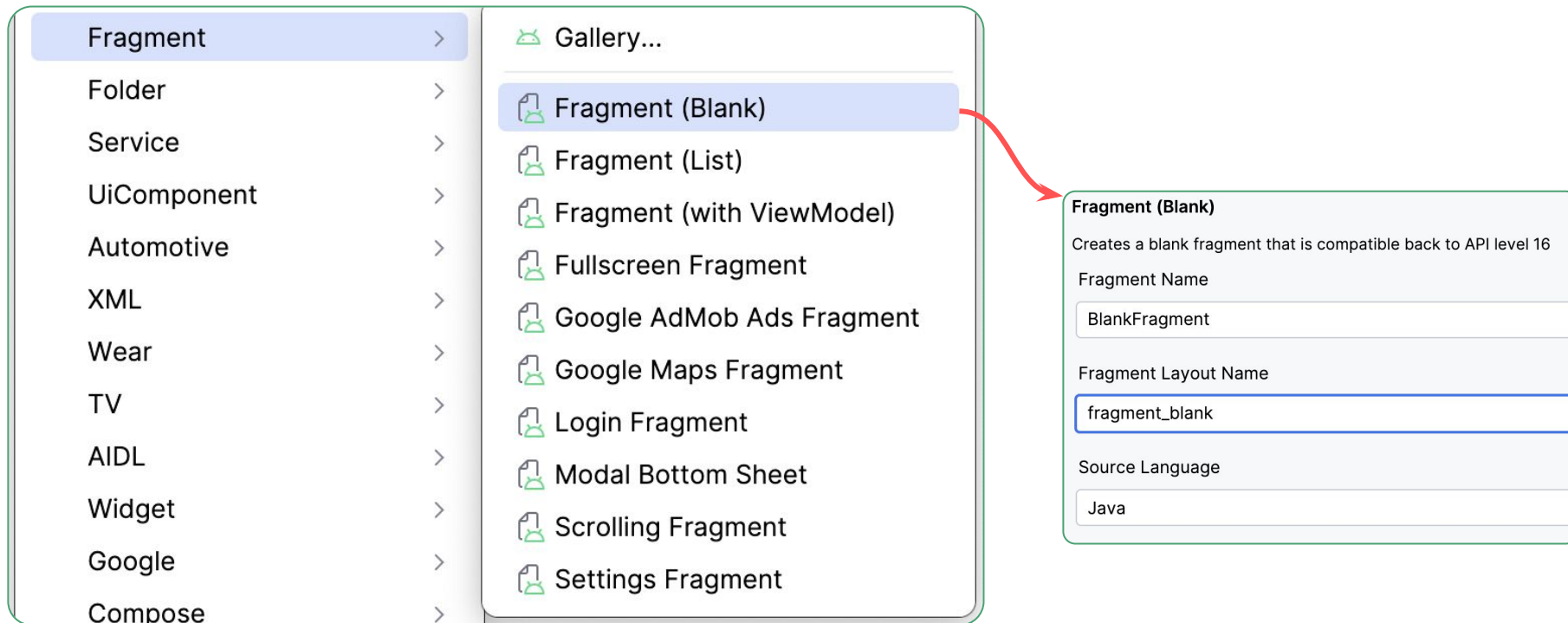
---

# Pasos para crear un fragmento

- Crear una subclase de Fragment
- Crear un layout para el fragmento
- Agregar el fragmento a la Actividad

# Creando un Fragmento - con el IDE

File → New → Fragment → Fragment (Blank)





# Estructura básica del fragmento - con IDE

Al crear un fragmento con el IDE le crea mucho “**Boilerplate code**” el cual no es necesario en este punto.

```
public class BlankFragment extends Fragment {

    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    2 usages
    private static final String ARG_PARAM1 = "param1";
    2 usages
    private static final String ARG_PARAM2 = "param2";

    // TODO: Rename and change types of parameters
    1 usage
    private String mParam1;
    1 usage
    private String mParam2;

    public BlankFragment() {
        // Required empty public constructor
    }
```

# Estructura básica del fragmento - con IDE

Un fragmento necesita al menos:

- El método **onCreateView()**.

Puede usar **ViewBinding**:

```
public class BlankFragment extends Fragment {  
    2 usages  
    FragmentBlankBinding binding;  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
  
        binding = FragmentBlankBinding.inflate(inflater, container, attachToParent: false);  
  
        return binding.getRoot();  
    }  
}
```

# Agregando el fragmento a la actividad

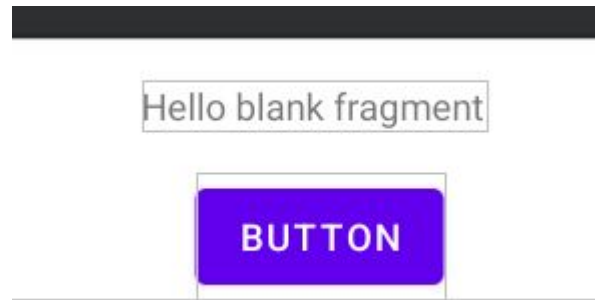
Se puede realizar de dos formas:

- **Estáticamente**, como parte del layout de la Actividad
- **Por código**, agregándole o quitándole en tiempo de ejecución.

# Agregar fragmentos estáticamente

En el Layout del fragmento → ***fragment\_blank***:

Se configurará un estilo visual al fragmento, en este ejemplo, un Textview y un botón.

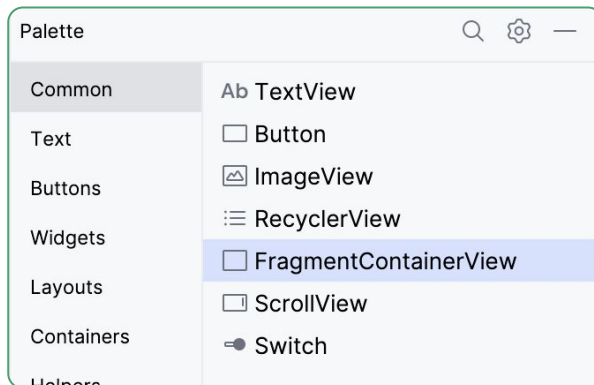


# En el Activity

En el layout del Activity, usando el IDE puede colocar el utilizando

**FragmentManager**. Debe indicarle:

- **android:name** → Nombre de la clase del fragmento
- **tools:layout** → Nombre del layout del fragmento



```
<androidx.fragment.app.FragmentManager
    android:id="@+id/FragmentManager"
    android:name="com.example.clase61.BlankFragment"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    tools:layout="@layout/fragment_blank" />
```

Pruebe su aplicación

# Agregar fragmentos mediante código

Para agregar, borrar o reemplazar fragmentos, se utiliza la clase **FragmentManager**, mediante la clase **FragmentManager**.

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

    if (savedInstanceState == null) {
        getSupportFragmentManager().beginTransaction()
            .setReorderingAllowed(true)
            .add(R.id.fragmentContainerView, BlankFragment.class, null)
            .commit();
    }
}
```

Parámetros de  
entrada al fragmento

Debe validar **savedInstanceState == null** para evitar agregar dos veces un fragmento, pues savedInstanceState restaura automáticamente el fragmento.

# Envío de parámetros básicos: Activity → Fragment

Para enviar parámetros al momento de crear el fragmento, utilice **Bundle**.

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

    if (savedInstanceState == null) {
        Bundle bundle = new Bundle();
        bundle.putInt("edad", 23);

        getSupportFragmentManager().beginTransaction()
            .setReorderingAllowed(true)
            .add(R.id.fragmentContainerView, BlankFragment.class, bundle)
            .commit();
    }
}
```

Parámetros de  
entrada al fragmento

# Recibir parámetros en el Fragment

```
@Override
public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    binding = FragmentBlankBinding.inflate(inflater, container, false);

    int edad = requireArguments().getInt("edad");
    Log.d("msg-test", "edad recibida: " + edad);

    return binding.getRoot();
}
```



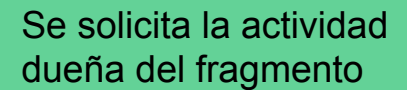
# Envío de parámetros avanzados: Activity → Fragment

Para enviar parámetros **cuyo tamaño sea considerable**, es mejor utilizar live data.

```
public class ExampleViewModel extends ViewModel {  
  
    private MutableLiveData<ArrayList<Persona>> listaPersonas = new MutableLiveData<>();  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        binding = ActivityMainBinding.inflate(getLayoutInflater());  
        setContentView(binding.getRoot());  
  
        if (savedInstanceState == null) {  
            ExampleViewModel exampleViewModel = new ViewModelProvider(this).get(ExampleViewModel.class);  
  
            ArrayList<Persona> listaPersonas = new ArrayList<>();  
            listaPersonas.add(new Persona("Juan", "Perez", "123"));  
            listaPersonas.add(new Persona("Carlos", "Sánchez", "234"));  
            listaPersonas.add(new Persona("Cesar", "Martinez", "345"));  
            exampleViewModel.getListPersonas().setValue(listaPersonas);  
  
            getSupportFragmentManager().beginTransaction()  
                .setReorderingAllowed(true)  
                .add(R.id.fragmentContainerView, BlankFragment.class, null)  
                .commit();  
        }  
    }  
}
```

# Recibir parámetros en el Fragment

Se solicita la actividad dueña del fragmento




```
@Override
public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    binding = FragmentBlankBinding.inflate(inflater, container, false);

    ExampleViewModel exampleViewModel = new ViewModelProvider(requireActivity())
                                                .get(ExampleViewModel.class);

    exampleViewModel.getListaPersonas().observe(getViewLifecycleOwner(), personas -> {
        for(Persona p: personas){
            Log.d("msg-test", "Name: " + p.getNombre());
        }
    });
}
```

# Agregar el Fragmento al backStack

Si desea agregar el fragmento agregado dinámicamente al back stack, es decir, para que el usuario al presionar la tecla back, deshaga el agregar el fragmento, debe incluir **addToBackStack(*null*)**, antes de hacer commit.



```
getSupportFragmentManager().beginTransaction()  
    .setReorderingAllowed(true)  
    .addToBackStack(null)  
    .add(R.id.fragmentContainerView, BlankFragment.class, null)  
    .commit();
```

# Borrar un fragmento dinámicamente

Puede borrar un fragmento dinámicamente, para esto debe obtenerlo desde el `FragmentManager` (buscándolo por el ID de su contenedor) y luego borrarlo.

```
public void borrarFragmento() {  
    Fragment fragmentById = getSupportFragmentManager()  
                                .findFragmentById(R.id.fragmentContainerView);  
  
    if (fragmentById != null) {  
        getSupportFragmentManager().beginTransaction()  
            .remove(fragmentById)  
            .commit();  
  
        getSupportFragmentManager().popBackStack();  
    }  
}
```

Solo se necesita si agregó el  
fragmento al backstack

# Extra

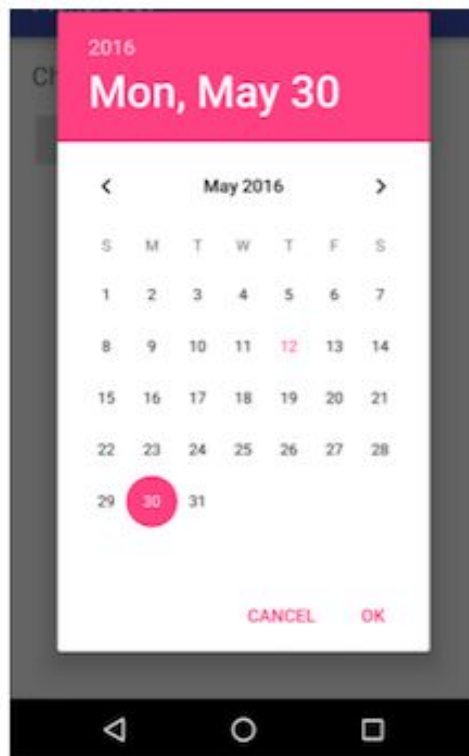
Date and time pickers



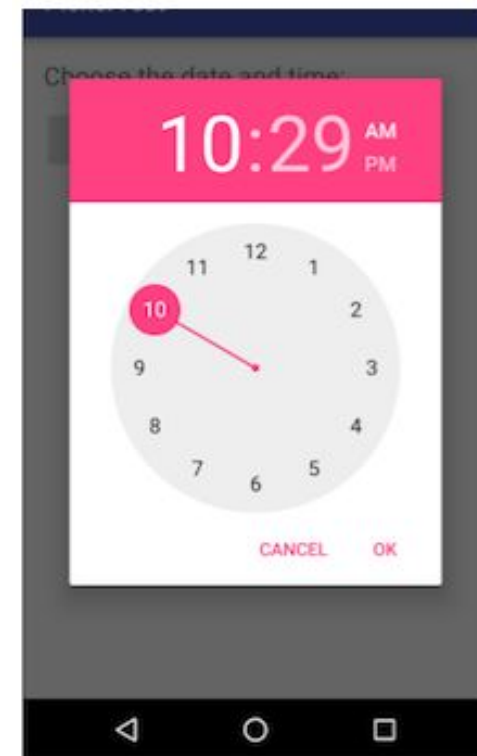
# Date and time pickers

Android brinda unos “**pickers**” para poder seleccionar la fecha y hora, esto mediante fragmentos pre-definidos.

Fecha



Hora



# Clase DialogFragment

Se crea una clase tradicional **DatePickerFragment** esta hereda de **DialogFragment**, la cual tiene la funcionalidad para mostrar el fragmento y a su vez debe implementar la interfaz **DatePickerDialog.OnDateSetListener**, la cual le indica que usará el DatePicker (para las **fechas**).

Si quisiera para las **horas**, debe implementar el **TimePickerDialog.OnTimeSetListener**.

```
public class DatePickerFragment extends DialogFragment
    implements DatePickerDialog.OnDateSetListener {

    @Override
    public void onDateSet(DatePicker datePicker, int year, int month, int day)
    {

    }

}
```

# Gestionar respuesta

En el fragmento, el método **onDateSet()** se lanza cuando el usuario selecciona una fecha.

Para enviarle información del fragmento a la actividad, usar `getActivity()`, el cual le devuelve la actividad que creó al fragmento y un método creado en la actividad.

PD: también puede usar Live Data para mandar la respuesta.

```
//En el fragmento
@Override
public void onDateSet(DatePicker datePicker, int year, int month, int day) {
    MainActivity mainActivity = (MainActivity) getActivity();
    mainActivity.respuestaDateDialog(year, month, day);
}
```

```
//En el activity
public void respuestaDateDialog(int year, int month, int day ){
    Log.d("msg-test", "year selected: " + year);
    Log.d("msg-test", "month selected: " + month);
    Log.d("msg-test", "day selected: " + day);
}
```



# Configuración de la hora y fecha actual

Sobreescriba el método **onCreateDialog** del fragmento, el cual le permitirá mostrar el picker como un Dialog, justamente, la funcionalidad del fragmento. Aquí debe enviar la fecha actual. Puede usar la librería Calendar.

```
@NonNull
@Override
public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
    Calendar c = Calendar.getInstance();

    return new DatePickerDialog(getActivity(), this,
        c.get(Calendar.YEAR),
        c.get(Calendar.MONTH),
        c.get(Calendar.DAY_OF_MONTH));
}
```

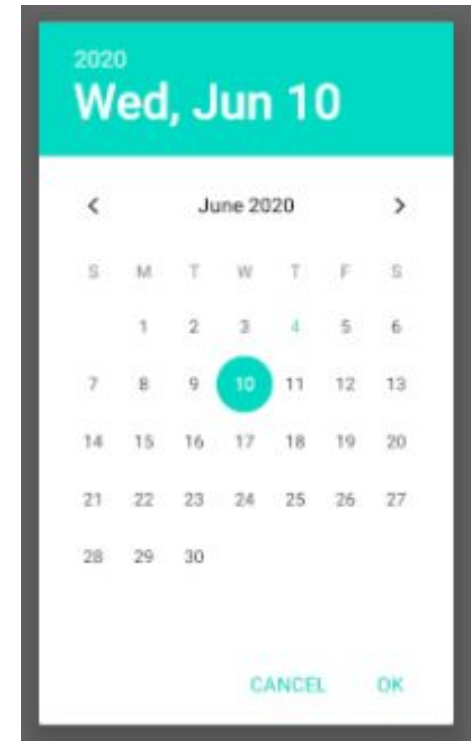
# Mostrando el picker

En el activity abra el fragmento.

```
public void mostrarDateDialog() {  
    DatePickerFragment datePickerFragment = new DatePickerFragment ();  
    datePickerFragment.show(getSupportFragmentManager() , "datepicker");  
}
```

# Probando

MOSTRAR FECHA



```
D year selected: 2023
D month selected: 3
D day selected: 24
```

# ¿Preguntas?

---

Muchas gracias

---