

Fragments y Navigation

Projecte Integrat de Software (PIS)

Universitat de Barcelona

victor.campello@ub.edu

carlos.martinisla@ub.edu

4 de Maig de 2022

1 Fragments y Navegación

- Fragments
- Fragments y Activities
- Fragments y navegación
- Interfaces

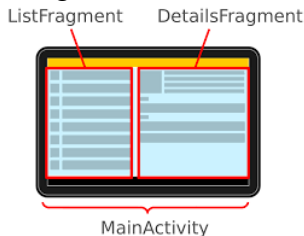
2 Diagrama de clases / Modelo de dominio

- Diagrama de clases
- Diagrama de clases específico para android
- Diagrama de clases genérico
- Modelo de dominio global específico para Android

Fragments

Un **Fragment** representa una parte de la interfaz de usuario. Puedes combinar varios fragmentos en una sola actividad y **volver a usar un fragmento en diferentes actividades**.

Un fragmento siempre debe estar alojado en una actividad y el ciclo de vida del fragmento se ve afectado directamente por el ciclo de vida de la actividad anfitriona. Es posible referenciar y acceder su parent activity así como comunicarse entre fragments mediante **Interfaces**



Ciclo de vida de un fragment

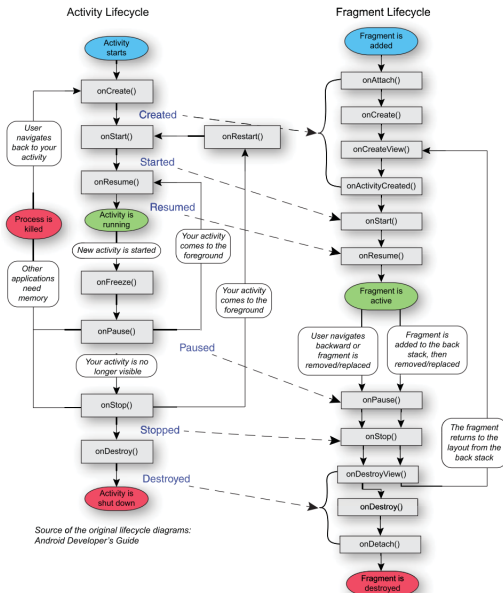


Figure 20.1 Activity and fragment lifecycles

Fragments y Activities

- A diferencia de una Activity, el método principal de su ciclo de vida es **OnCreateView**.
- Al pausar una Activity se pausarán todos los Fragments definidos dentro de esta.
- Si se destruye una Activity, todos los Fragments que contenga serán también destruidos.
- Si la Activity se encuentra en ejecución, es posible manipular de manera independiente cada Fragment, inclusive destruirlo, con el objeto **FragmentManager**.
- Se puede usar el método **getActivity()** para recuperar la parent Activity que contiene un fragment.

Fragments y Navegación

Este es el aspecto de un fragment vacío con su método **OnCreateView()**

```
public class DashboardFragment extends Fragment {  
    public View onCreateView(@NonNull LayoutInflater inflater,  
                            ViewGroup container, Bundle savedInstanceState) {  
  
        View root = inflater.inflate(R.layout.fragment_dashboard, container, attachToRoot: false);  
        return root;  
    }  
}
```

Fragments y navegación

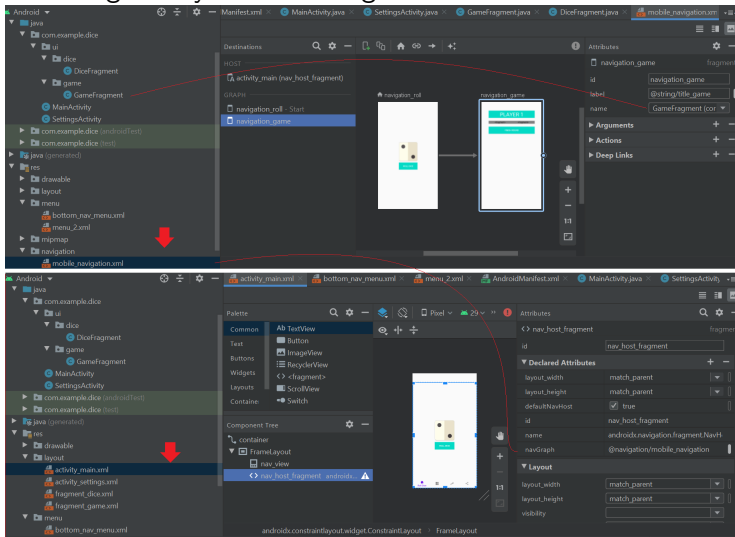
Como se observa al crear por ejemplo **Bottom Navigation Activity**, se añaden una serie de fragments en la carpeta **java/com.example.appname/UI** con sus correspondientes layouts en la carpeta **res/layout**. A su vez, se generó una layout de **navigation** en **res/navigation**

El componente **Navigation**, en este caso el **BottomNavigationView**, consta de tres partes clave que se describen a continuación:

- **Gráfico de navegación:** Es un recurso XML que contiene toda la información relacionada con la navegación en una ubicación centralizada.
- **NavHost:** Es un contenedor vacío que muestra los destinos de tu gráfico de navegación, en este caso Fragments.
- **NavController:** Es un objeto que administra la navegación de la app dentro de un NavHost. Se puede usar su método **navigate** para navegar a cualquier elemento del navHost.

Fragments y navegación

Gráfico de navegación y NavHostFragment



Fragments y navegación

Acceso al objeto NavController desde la MainActivity.

```
// Passing each menu ID as a set of Ids because each
// menu should be considered as top level destinations.
BottomNavigationView navView = findViewById(R.id.nav_view);
AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(
    R.id.navigation_roll, R.id.navigation_game)
    .build();

NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment);
NavigationUI.setupActionBarWithNavController( activity: this, navController, appBarConfiguration);
NavigationUI.setupWithNavController(navView, navController);
```

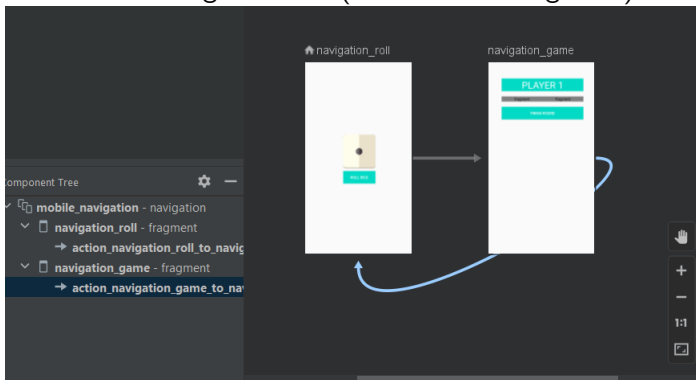
En la aplicación presentada, se han combinado activities con fragments en el mismo BottomNavigationView.

- **Setings y Share:** Usan OnClick para lanzar intents a sus respectivas actividades/ acciones
- **Roll y Game:** Los botones del bottom-nav-menu.xml comparten ID con los diferentes elementos mobile-navigation.xml. De esta forma, la navegación es implementada de forma automática.

Pero también se pueden crear navegaciones más complejas usando **Actions y NavController**

Fragments y Navigation

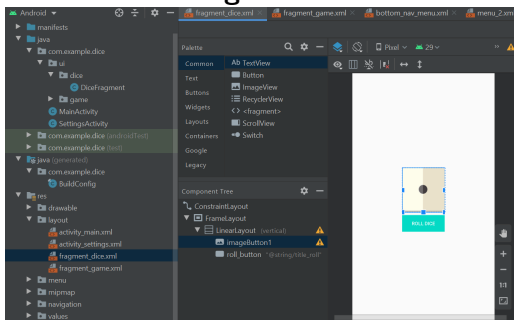
Una **Action** se define en el gráfico de navegación y se ejecuta mediante el método **navigate** de **NavController**. Dicho método también puede navegar usando las IDs de los fragments definidas en `mobile-navigation.xml` (Gráfico de Navegación).



Para acceder a los items de un menú no podemos usar un `onClickListener` convencional sino que tendremos que setear un listener del tipo **`OnNavigationItemSelectedListener()`**, para posteriormente usar el método **`navigate`** de **`NavController`** según convenga.

Ejercicio de Fragments: Dice

Integraremos en nuestra aplicación un Fragment simple que contenga un dado. Para ello modificamos la layout de nuestro primer fragment, al que hemos renombrado como **DiceFragment**.



Situad una `ImageView` o `ImageButton` y un botón adicional de "roll", como se muestra en pantalla.

Ejercicio de Fragments: Dice

En nuestro código java **DiceFragment** añadimos los siguientes atributos y el `onClick`Listener del botón "roll".

```
public class DiceFragment extends Fragment {

    private ImageButton dice;           // ImageButton from fragment_dice.xml
    private Drawable[] dice_drawable_resources; // array of the .png drawable images
    private Handler handler = new Handler(); // methods to create a simple animation
    private Timer t = new Timer();        //
    public int currentValue = 2;          //current value of the dice
    public Button rollBtn;                //button to roll the dice

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        View root = inflater.inflate(R.layout.fragment_dice, container, attachToRoot: false);
        initDice(root);

        rollBtn = root.findViewById(R.id.roll_button); //get the roll button
        rollBtn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Log.w("tag: DiceFragment", "msg: rollDice");
                //roll the dice
                rollDice();
            }
        });
        return root;
    }
}
```

Ejercicio de Fragments: Dice

El método roll dice genera una ráfaga de valores aleatorios ejecutados con delay, actualiza para generar un efecto animado.

```
private void initDice(View root){
    this.dice = root.findViewById(R.id.imageButton1);
    dice_drawable_resources = new Drawable[6];
    dice_drawable_resources[0] = getResources().getDrawable(R.drawable.d_1);
    dice_drawable_resources[1] = getResources().getDrawable(R.drawable.d_2);
    dice_drawable_resources[2] = getResources().getDrawable(R.drawable.d_3);
    dice_drawable_resources[3] = getResources().getDrawable(R.drawable.d_4);
    dice_drawable_resources[4] = getResources().getDrawable(R.drawable.d_5);
    dice_drawable_resources[5] = getResources().getDrawable(R.drawable.d_6);
}

private void rollDice(){ //roll the dice with a decreasing rolling time
    for (int step=200; step <3200; step *=1.2) {
        t.schedule(new TimerTask() {
            public void run() {
                handler.post(new Runnable() {
                    public void run() {
                        Random r = new Random();
                        int i = r.nextInt( bound: 6);
                        dice.setImageDrawable(dice_drawable_resources[i]);
                        currentValue = i +1;
                    }
                });
            }
        }, step);
    }
}
```

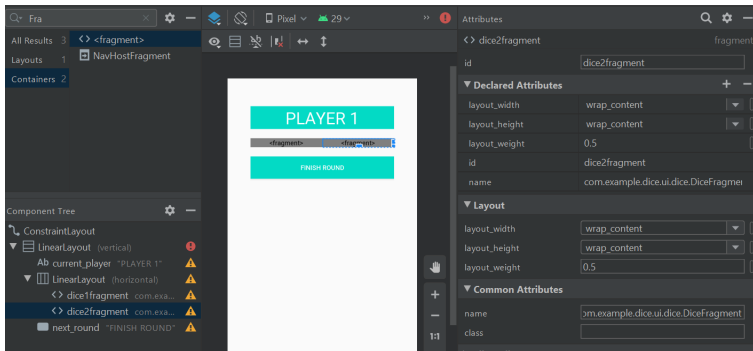
Ejercicio de Fragments: Dice

Comprueba el funcionamiento de tu dado ejecutando el emulador y presionando el botón de "roll".

Una vez generado nuestro fragment **DiceFragment**, vamos a desarrollar el segundo fragment, **GameFragment**, un sencillo juego de dos dados. Su layout anidará dos **DiceFragment**

- Incluiremos dos containers de tipo Fragment e indicaremos que sean DiceFragments.
- Vamos a añadir un textView que mostrará el nombre de los jugadores introducidos en **SettingsActivity** y guardados en **MainActivity** a través de un **Intent Explícito**.
- Además, añadiremos un botón para pasar de ronda.

Ejercicio de Fragments: Game



Ejecuta el emulador y comprueba que puedes navegar con el **Bottom Navigation Menu** a **GameFragment**

Ejercicio 1

Añade el código necesario en la clase **GameFragment** para cambiar el nombre del jugador de su correspondiente TextView cada vez que se pulse el botón de "Next Round". Recuerda que puedes usar el método **getActivity()** para acceder a la activity Parent de un **Fragment**.

Ejercicio de Fragments: Acceso y anidado de Fragments

Puedes acceder a los fragment Children con ids **dice1fragment** y **dice2fragment** con el método getChildFragmentManager(). También podrías añadir un tercer DiceFragment realizando una FragmentTransaction si fuera necesario.

```
private DiceFragment dice1,dice2; //Dice Fragments to recover dice values generated by
                                   //DiceFragment.rollDice() and stored in DiceFragment.currentValue
public View onCreateView(@NonNull LayoutInflater inflater,
                        ViewGroup container, Bundle savedInstanceState) {

    root = inflater.inflate(R.layout.fragment_game, container, attachToRoot: false);
    dice1 = (DiceFragment) getChildFragmentManager().findFragmentById(R.id.dice1fragment);
    dice2 = (DiceFragment) getChildFragmentManager().findFragmentById(R.id.dice2fragment);

    FragmentManager fragmentManager = getParentFragmentManager();
    FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
    fragmentTransaction.add(R.id.multiple_fragments_layout, new DiceFragment());
    fragmentTransaction.commit();
}
```

Añade el código necesario en la clase **GameFragment** para obtener los valores de los dados cuando se pulse el botón de nextRound y mostrarlos en un mensaje **Toast**.

```
Toast toast = Toast.makeText(getActivity(), hint, Toast.LENGTH_LONG);  
toast.setGravity(Gravity.CENTER, xOffset: 0, yOffset: 0);  
toast.show();
```

Ejercicio de Fragments: Interfaces

Para comunicar Eventos entre Fragments o Activities, debemos hacer uso de una **interfaz**. La interfaz de un Fragment será usada para transmitir Eventos a los Fragments externos o activities que implementen los métodos definidos en la dicha interfaz.

Ejercicio de Fragments: Interfaces

Vamos a crear una interfaz en DiceFragment que permitirá a GameManager detectar el evento OnClick del "roll button".

```
rollBtn.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        Log.w( tag: "DiceFragment", msg: "rollDice");
        //roll the dice
        rollDice();
        //call interface method implemented on GameFragment to detect clicks on child DiceFragments
        if (mListener != null) {
            mListener.onFragmentClick(v);
        }
    }
});
return root;
}

private FragmentClickListener mListener;


// This is the interface.
public interface FragmentClickListener {
    void onFragmentClick(View v);
}

// This method can take any class that implements FragmentClickListener
public void registerListener(FragmentClickListener mListener) {
    this.mListener = mListener;
}
```


Ejercicio de Fragments: Interfaces

Implementamos el método de la interfaz creada en GameFragment y registramos los listeners.

```
public class GameFragment extends Fragment implements DiceFragment.FragmentClickListener {  
  
    @Override  
    public void onFragmentClick(View v) {  
        Log.w( tag: "GameFragment", msg: "DiceFragment Clicked");  
        v.setVisibility(View.INVISIBLE);  
    }  
}
```



```
public View onCreateView(@NonNull LayoutInflater inflater,  
    ViewGroup container, Bundle savedInstanceState) {  
  
    root = inflater.inflate(R.layout.fragment_game, container, attachToRoot: false);  
  
    dice1 = (DiceFragment) getChildFragmentManager().findFragmentById(R.id.dice1fragment);  
    dice2 = (DiceFragment) getChildFragmentManager().findFragmentById(R.id.dice2fragment);  
  
    dice1.registerListener( mListener: this);  
    dice2.registerListener( mListener: this);  
}
```



En concreto, haremos que un botón de DiceFargemnt no sea visible una vez pulsado, para evitar tiradas adicionales antes de iniciar un nuevo turno.

Ejercicio 3

Vuelve a hacer visibles los botones de los dados 1 y 2 cuando se pulse el botón de "next round".

Diagrama de clases y modelo de dominio

- En un diagrama de clases completo está la relación entre la interfaz y la implementación, por lo tanto relacionaremos nuestras clases entre ellas y además, con sus activities/fragments a través de sus viewModels.
- No obstante, podemos subdividir el proceso en **diagrama de clase de dominio** y **diagrama de clase de diseño**. Esto es particularmente aconsejable para favorecer la interpretación de dichos modelos.

Diagrama de clases específico para android

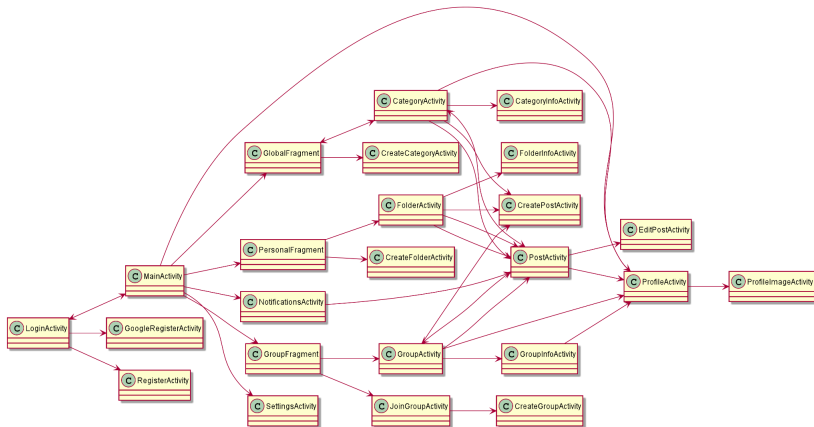
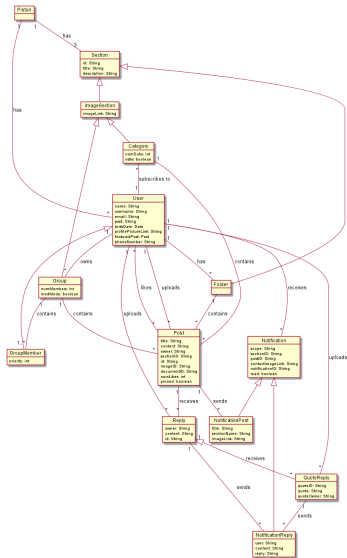
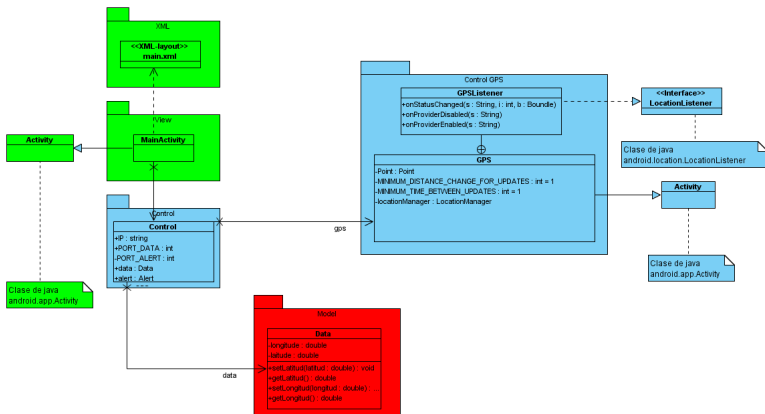


Diagrama de clases genérico



Modelo de dominio global específico para Android



End