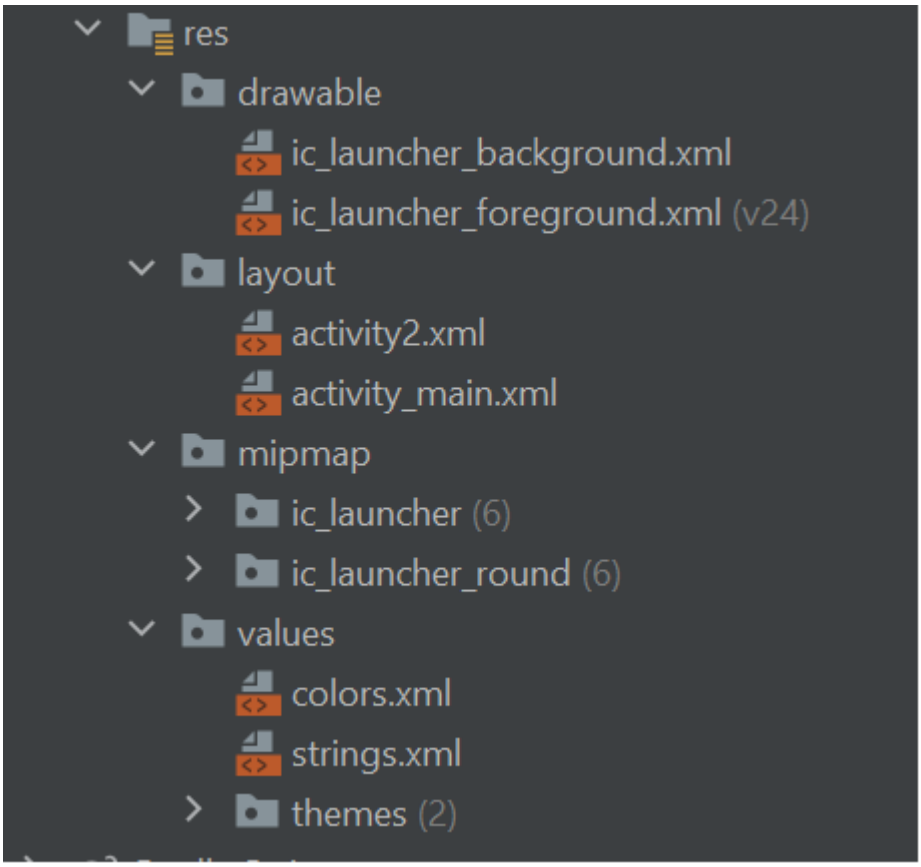
Tema 2

Recursos Android

Subdirectorios



- **drawable**: Recursos gráficos que vamos a utilizar como .png, .jpg, .gif...
- **mipmap**: Unicamente guardaremos iconos de la aplicación en las diferentes densidades de pantalla.
- **layout**: Archivos XML que contienen definiciones de la interfaz de usuario(vistas).
- **menu**:Archivos XML que establecen las características para los menús usados en la interfaz.
- **values**: Archivos XML que contienen datos simples como enteros, strings, booleanos, colores.

Carpetas Mipmap y Drawable

Densidades:

- **Mediun Dots per Inch(mdpi):** 160pulgadas
- **High Dots per Inch(hdpi):** 240pulgadas

- **Extra high dots per inch(xhdpi):** 340pulgadas
- **Extra Extra high dots per inch(xxhdpi):** 480pulgadas

Carpeta values

Se crean por defecto 3 carpetas colors, strings, theme.

Strings: almacena todas las cadenas que se muestran

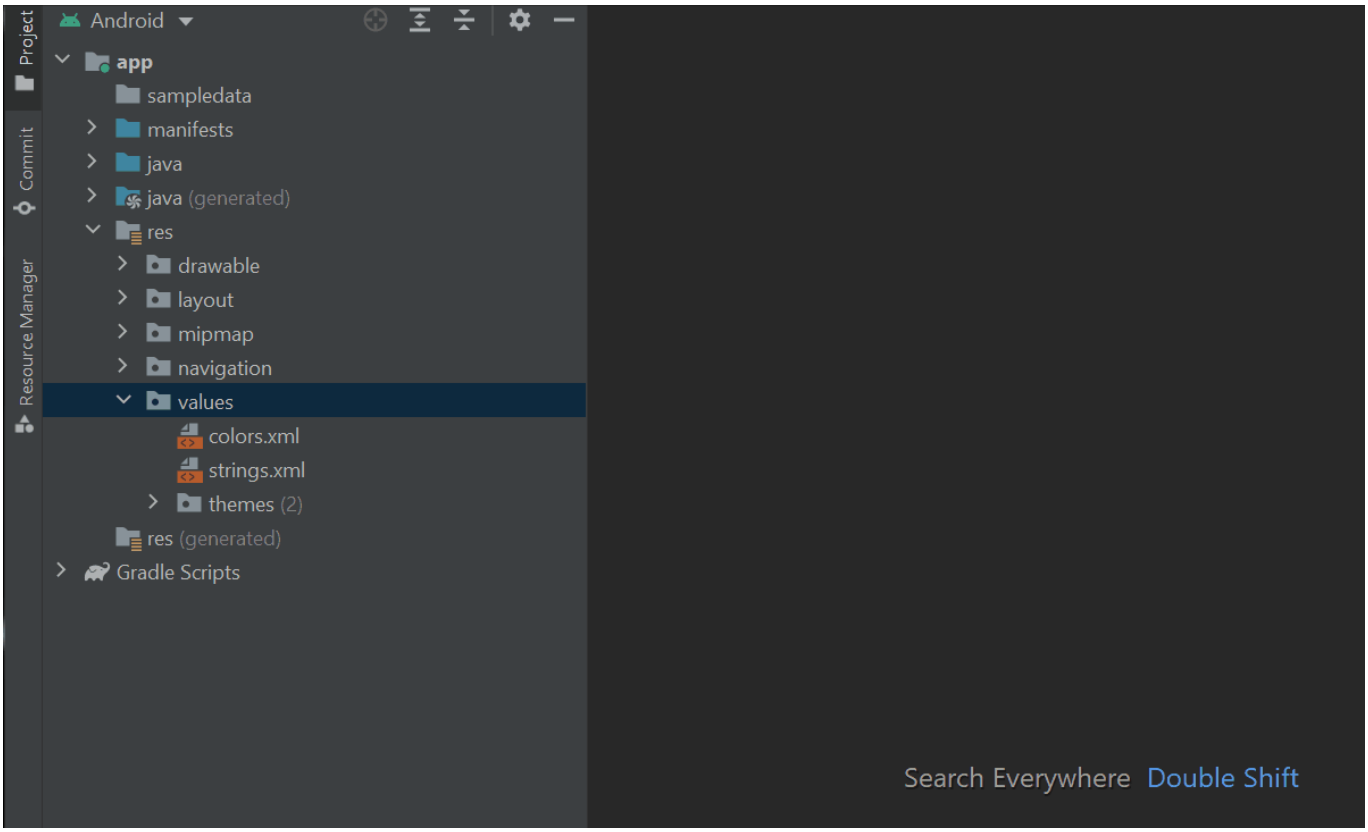
```xml
<resources>
    <string name="app_name">ejemplo1</string>
</resources>
```

Color: a cada estiqueta <color> se le asigna un nombre y una referencia #hexadecimal que es la que representa el color

strings.xml colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Al crear un recurso se le pueden dar propiedades adicionales como por ejemplo la nacionalidad y lengua en el caso de un strings



Acceder a los Recursos en Android

Desde Kotlin:

Cada identificador se ubica en una clase llamada R a través de una constante entera **R.tipoRecurso.nombreRcurso**.

```
package com.ejemplos.myapplication

import ...

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        R.drawable.
    }
}
```

```
ic_launcher_background  Int
ic_launcher_foreground  Int
```

Desde XML:

Para ello usamos la sintaxis @[Paquete:]tipoRecurso/nombreRecurso

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"
```

Android Manifest

Manifest: Es el archivo más esencial de nuestra aplicación que contiene nodos descriptivos sobre las características de la App como la versión de SDK,permisos necesarios, servicios, activitys.

Etiqueta aplicacion:

- allowBackup(true/false): Indica si la aplicacion sera persistente al cerrar nuestro AVD.
- icon: Indica donde esta ubicado el icono de la aplicación.
- label: Nombre de la aplicación que vera el usuario en su telefono
- theme: Apunta al archivo styles.xml donde se define el estilo visual de la aplicación
- supportsRtl: Declara si la aplicacion esta dispuesta a admitir diseños de derecha a izquierda

Etiqueta activity:

Representa actividades de la aplicaón

- label: Texto que se mostrara en la cabecera de la actividad
- android:name: Especifica el nombre de la actividad
- android:screenOrientation: Define la orientación de la aplicación vertical o apaisada
- android:configChanges: se definen los diferentes eventos que manejar con el fin de evitar que se destruya la actividad (al cambiar de orientacion o cuando aparece un teclado) se separan por | ejemplo keyboard|keyboardHydden|orientation

Con la etiqueta <intent-filter> y su elemento <action> estamos indicando que esta activity va a ser un punto de entrada para nuestra aplicación. Sólo puede haber una activity que reaccione a este intent. Con elemento le decimos a Android que queremos que esta activity sea añadida al lanzador de la aplicación.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Etiqueta uses-permissions:

Android restringe el uso de los recursos del sistema: tarjeta SD, WIFI, HW de audio, etc. Mediante este Tag especificamos los permisos que va a necesitar nuestra aplicación.

- android.permisssion.RECORD_AUDIO: Acceso al hw de audio y grabación.

- android.permisssion.INTERNET: Permiso a todas las API's de red.
- android.permisssion.WRITE_EXTERNAL_STORAGE:Almacenamiento externo.
- android.permisssion.WAKE_LOCK: nos permite antibloqueo

Permisos en momento de ejecución:

Permisos en el momento de ejecución

1. Declara los permisos en el manifest
2. Esperar a que el usuario invoque la accion que requiere
    1. Comprueba que los permisos todavia no han sido otorgados
    2. Muestra justificación (Dialogo) por la que se requieren los permisos, en caso que se vea necesario.**Linea 6 y 10**
    3. Solicita los permisos. **Linea 14**
3. En caso afirmativo accede al recurso, si no es asi se le proporciona un mensaje explicando que no puede acceder

Tema2 pagina 13

```
    val RESPUESTA_PERMISOS = 111

    @RequiresApi(Build.VERSION_CODES.Q)
    fun solicitarPermisos() {
        if (checkSelfPermission(READ_EXTERNAL_STORAGE) ==
PackageManager.PERMISSION_DENIED
            || checkSelfPermission(READ_CONTACTS) ==
PackageManager.PERMISSION_DENIED
        ) {
            if (shouldShowRequestPermissionRationale(READ_EXTERNAL_STORAGE)) {
//Si se decide explicar los motivos de los permisos
// con algo similar a un dialogo
            }
            if (shouldShowRequestPermissionRationale(READ_CONTACTS)) {
//Si se decide explicar los motivos de los permisos
// con algo similar a un dialogo
            }
//Con requestPermissions se pide al usuario que permita los permisos,
//en este caso dos
            requestPermissions(
                arrayOf(READ_EXTERNAL_STORAGE, READ_CONTACTS),
                RESPUESTA_PERMISOS
            )
        }
    }
```