Geolocalización

Crearemos la **App_03_03**
En esta actividad desarrollaremos una aplicación que muestre la última localización GPS registrada. Posteriormente actualizaremos las coordenadas automáticamente a medida que van cambiando.

PASOS PREVIOS

Usaremos la API de servicios de localización de Google. Añadiremos la última versión al fichero build.gradle:

```
implementation 'com.google.android.gms:play-services-location:21.0.1'
```

Como nuestra aplicación va a usar servicios de localización añadiremos los permisos en el archivo de manifiesto.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

ACCESS_FINE_LOCATION: para una precisión alta (habitualmente cuando se usa GPS)
ACCESS_COARSE_LOCATION: para una precisión baja (habitualmente cuando se usan antenas de telefonía o WiFi).

DISEÑO

En el activity_main.xml crearemos un FrameLayout que contendrá algún fragment y un menú inferior:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <FrameLayout
      android:id="@+id/layoutFragmentHolder"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      app:layout_constraintBottom_toTopOf="@+id/viewBottomNavigation"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.5"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
```

```
    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/viewBottomNavigation"
        android:layout_width="match_parent"
        android:layout_height="75dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:menu="@menu/menu_abajo"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Crearemos un fichero de configuración del menú llamado **menu_abajo.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/localizacion"
        android:icon="@android:drawable/ic_menu_mylocation"
        android:title="@string/opcion_localizacion" />
    <item
        android:id="@+id/mapa"
        android:icon="@android:drawable/ic_dialog_map"
        android:title="@string/opcion_mapa" />
</menu>
```

Crea los strings de los títulos de los menús en el fichero **strings.xml**

Establece **viewBinding** en build.gradle

En nuestra aplicación tendremos 2 fragments: **FragmentLocalizacion** y **FragmentMapa**.
Créalos utilizando el asistente y deja únicamente le método onCreateView en cada uno de ellos.

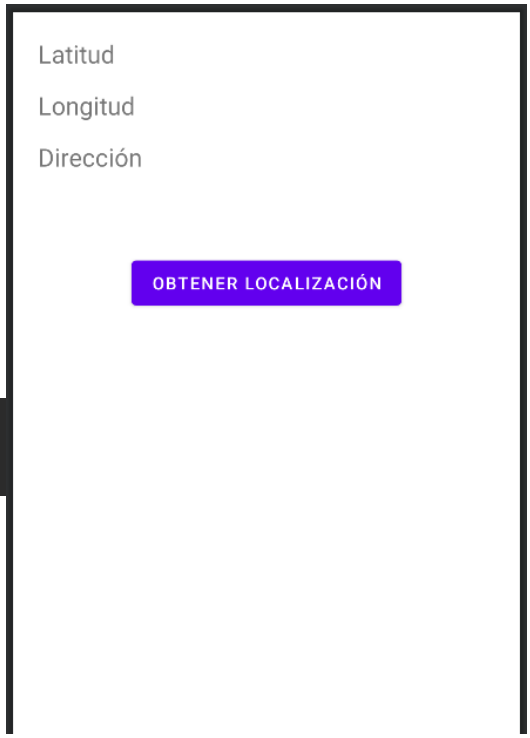
En la clase **MainActivity** asociaremos el fragment en función de la opción de menú seleccionada. El código podría ser el siguiente:

```
class MainActivity : AppCompatActivity()
{
    private lateinit var binding: ActivityMainBinding

    private val fragmentLocalizacion = FragmentLocalizacion()
    private val fragmentMapa = FragmentMapa()


    override fun onCreate(savedInstanceState: Bundle?)
```

```
    {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setFragment(fragmentLocalizacion)
        opcionesMenu()
    }


    private fun setFragment(fragmentToChange: Fragment) {
        supportFragmentManager.beginTransaction()
            .replace(R.id.layoutFragmentHolder, fragmentToChange)
            .commit()
    }

    private fun opcionesMenu() {
        binding.viewBottomNavigation.setOnItemSelectedListener {
            when (it.itemId) {
                R.id.localizacion -> setFragment(fragmentLocalizacion)
                R.id.mapa -> setFragment(fragmentMapa)
            }
            true
        }
    }
}
```

FragmentLocalizacion

Tendremos 3 TextView en el que se mostrará la latitud, longitud y la dirección respectivamente.
Además, habrá un botón OBTENER LOCALIZACIÓN



En el fichero strings.xml añade las 3 cadenas de los TextView:

```
<string name="latitud">Latitud: %1$f</string>
<string name="longitud">Longitud: %1$f</string>
<string name="direccion">Dirección: %1$s</string>
```

IMPLEMENTACIÓN

Crea el binding para poder hacer uso de los componentes de la interfaz de usuario:

```
private lateinit var binding: FragmentLocalizacionBinding
```

Lo primero que haremos será comprobar los permisos.
Se necesita que el usuario haya aceptado los permisos de localización que hemos declarado en el archivo de manifiesto para poder continuar.

Esto es así desde la API 23 de Android para los llamados permisos "riesgosos". Si ejecutáramos la aplicación en un dispositivo de API Android inferior a la 23 no habría ningún problema.
Para una API mayor o igual a la 23, se requiere codificar que se le pida al usuario conceder el permiso y se controle si lo ha dado o no.

Llamaremos a este método desde onViewCreated cuando se pulse el botón:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
   binding.botonObtenerLocalizacion.setOnClickListener {
      comprobarPermisos()
   }
}
```

Por tanto, vamos a crear un método privado llamado **comprobarPermisos()**. En él crearemos un array con los 2 permisos anteriores y llamaremos a un método que compruebe si estos permisos ya están concedidos (puede que el usuario ya haya aceptado para siempre los permisos de esta aplicación). En caso que no estén concedidos se solicitarán al usuario.
Si ya estuviesen concedidos geolocalizaremos.

```
private fun comprobarPermisos()
{
   val permisos = arrayOf(Manifest.permission.ACCESS_COARSE_LOCATION,
Manifest.permission.ACCESS_FINE_LOCATION)
   if (!tienePermisos(permisos))
      solicitarPermisos(permisos)
   else
      geolocalizar()
}
```

Ahora tenemos que hacer el método tienePermisos que compruebe si los permisos pasados están concedidos:

```
private fun tienePermisos(permisos: Array<String>): Boolean {
   return permisos.all {
      return ContextCompat.checkSelfPermission(
        requireContext(),
        it
```

```
        ) == PackageManager.PERMISSION_GRANTED
    }
}
```

En caso que no haya algún permiso concedido se vuelven a pedir y si aceptamos todos geolocalizaremos:

```
private val peticionPermisos =
    registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions()) { permissions ->
        val aceptados = permissions.entries.all {
            Toast.makeText(context, "${it.key} = ${it.value}", Toast.LENGTH_SHORT).show()
            it.value
        }
        if(aceptados)
            geolocalizar()
    }

private fun solicitarPermisos(permisos: Array<String>)
{
    peticionPermisos.launch(permisos)
}
```

Una vez solucionado el tema de los permisos tendremos que hacer el método que nos geolocalice:

Para ello declaramos como atributo de la clase un objeto **FusedLocationProviderClient**.

```
private lateinit var fusedLocationClient: FusedLocationProviderClient
```

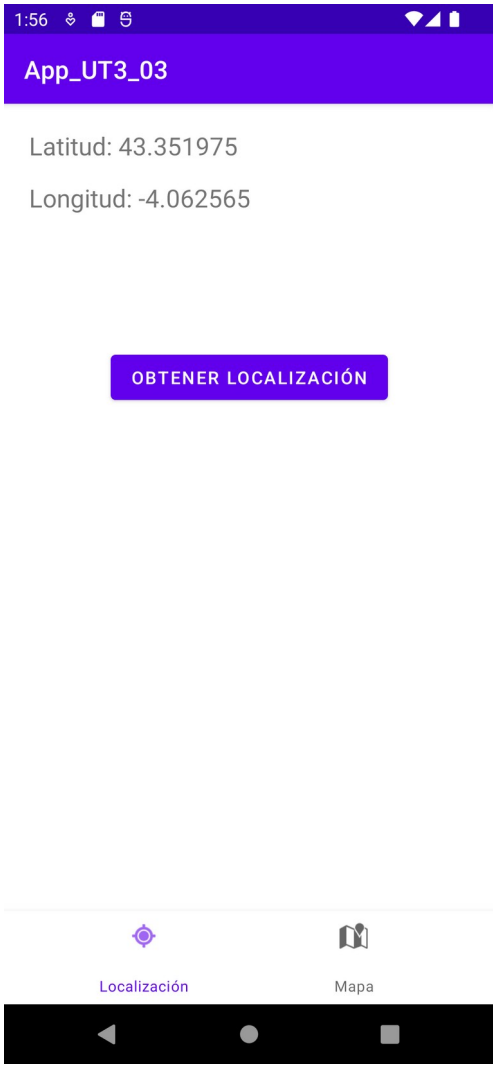
Luego crearemos un método que podemos llamar geolocalizar:

```
@SuppressLint("MissingPermission")
// Hasta aquí sabemos que los permisos ya están concedidos, por eso añado la anotación
private fun geolocalizar()
{
    fusedLocationClient = LocationServices.getFusedLocationProviderClient(requireContext())

    fusedLocationClient.lastLocation.addOnSuccessListener {
        if (it != null)
            imprimirUbicacion(it)
    }
}
```

Lo que hará el método es instanciar un objeto **FusedLocationProviderClient**.
Por ahora, queremos que inicie una búsqueda de la última localización reconocida (no un proceso de actualización automática).
Llamando a la propiedad **lastLocation** buscará la última ubicación conocida.
Este proceso puede tardar un tiempo, así que se llama a un **listener** que escucha cuando termina el proceso.
Si la ubicación es distinta de nulo se imprime. Simplemente mostraremos la latitud y longitud:

```
private fun imprimirUbicacion(ubicacion: Location)
{
   binding.textViewLatitud.text = getString(R.string.latitud, ubicacion.latitude)
   binding.textViewLongitud.text = getString(R.string.longitud, ubicacion.longitude)
}
```

Prueba la aplicación. Debería mostrar algo similar a lo siguiente:



Buscar dirección

Para buscar la dirección utilizaremos la clase **Geocoder**. Nos proporcionará resultados a partir de una latitud y longitud.

```
private fun buscarDireccion(ubicacion: Location): String
{
   val geocoder = Geocoder(context, Locale.getDefault())
```

```
    val direcciones = geocoder.getFromLocation(
        ubicacion.latitude, ubicacion.longitude, 1
    )

    if (direcciones.isNotEmpty())
    {
        val direccion = direcciones.first()
        val textoDireccion =
            (0..direccion.maxAddressLineIndex)
                .joinToString("\n") { i -> direccion.getAddressLine(i)}
        return textoDireccion
    }
    return ""
}
```

Por último, estableceremos el texto del TextView de la dirección con el resultado de llamar a método anterior.

Iniciar actualizaciones de localización constantes

Ahora, vamos a hacer que al pulsar el botón no se obtenga una localización estática sino que se inicie un proceso que actualice automáticamente los datos de localización del dispositivo y se muestren.

Para ello declaramos en la clase un objeto de la clase **LocationCallback** que recibirá actualizaciones de la ubicación:

```
private lateinit var locationCallback: LocationCallback
```

Creamos el método **geolocalizacionConstante** que realizará peticiones cada 10 segundos (aunque podríamos modificar ese valor). Para cada localización recibida se imprime la ubicación:

```
@SuppressLint("MissingPermission")
private fun geolocalizacionConstante()
{
    fusedLocationClient = LocationServices.getFusedLocationProviderClient(requireContext())

    val locationRequest = LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY,
        10000).build()

    locationCallback = object : LocationCallback()
```

```
    {
        override fun onLocationResult(locationResult: LocationResult)
        {
            for (location in locationResult.locations)
                imprimirUbicacion(location)
        }
    }
    fusedLocationClient.requestLocationUpdates(
        locationRequest,
        locationCallback, Looper.getMainLooper()
    )
}
```

Sólo nos faltaría cambiar la llamada a geolocalizacionConstante en vez del método geolocalizar desde el método **comprobarPermisos**.

Podríamos crear un Toast en el método imprimirUbicacion para comprobar cuándo se realiza la actualización de la ubicación:

```
private fun imprimirUbicacion(ubicacion: Location)
{
    Toast.makeText(context, "${ubicacion.latitude}, ${ubicacion.longitude}",
Toast.LENGTH_SHORT).show()
    binding.textViewLatitud.text = getString(R.string.latitud, ubicacion.latitude)
    binding.textViewLongitud.text = getString(R.string.longitud, ubicacion.longitude)
    binding.textViewDireccion.text = buscarDireccion(ubicacion)
}
```