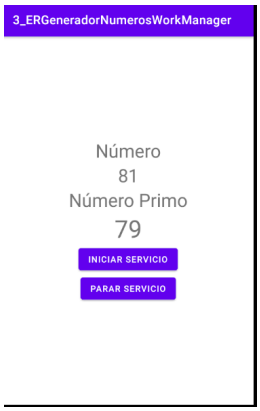
# Bloque 10 . Ejercicio Resuelto Numeros Primos

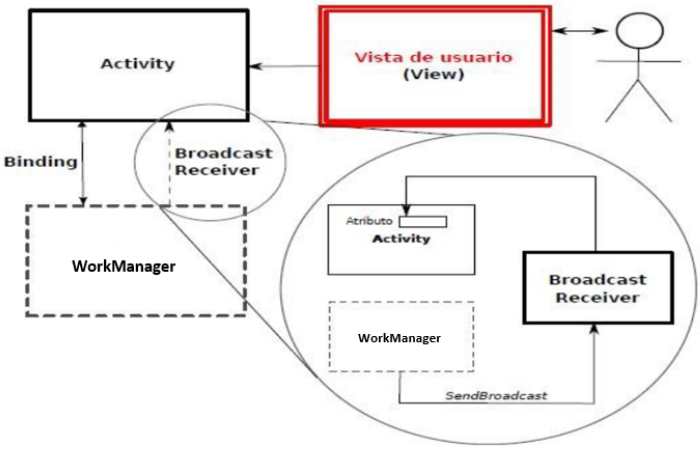
Descargar este ejercicio

Vamos a realizar una aplicación que recibe números a través de un trabajo, creado con `WorkManager`, y actualiza la interfaz gráfica, concretamente se visualizará el último número generado y el último número primo detectado por la pantalla del dispositivo. La aplicación tendrá dos botones para iniciar y detener el servicio.




Los componentes de nuestra aplicación serán los siguientes:

- Activity: con una vista asociada que permita al usuario parar y reiniciar el servicio y mostrar los datos correspondientes.
- WorkManager: un trabajo que genere en background los números independientemente de la activity.
- BroadcastReceiver: un receptor de mensajes de forma que cada vez que se recibe un número, lo procesa (para saber si es primo) y actualiza la vista de la activity. El broadcast receiver puede ser definido junto a la activity o en una clase a parte.




Empecemos definiendo el código de nuestro trabajo que llamaremos **GeneradorNumeros.kt**. Hay que tener en cuenta que cuando se implementa un `WorkManager` como una clase derivada de `Worker` es necesario definir el método `doWork()`, es aquí donde se implementan las instrucciones que permiten proporcionar el servicio a través de un hilo diferente.

El código de nuestro servicio:

```
class GeneradorNumeros(var context: Context, workerParams: WorkerParameters) :
    Worker(context, workerParams) {
    override fun doWork(): Result {
        val datos=inputData
        var contador= datos.getInt("INICIO",0)
        while (!isStopped) {
            var i = Intent(ACCION)
            i.putExtra("VALORCONTADOR", contador)
            applicationContext.sendBroadcast(i)
            contador++
            Thread.sleep(1000)
        }
        return Result.success()
    }
    companion object
    {
        const val ACCION = "com.ejemplos.b10.a3_ergeneradornumerosworkmanager.recibirNume
    }
}
```

📌 **Línea 2** heredamos de la clase Worker que nos obligará a implementar el método necesario. En el método doWork() se codifica la funcionalidad del trabajo. **Línea 4 - 5** aquí recuperamos un Data para extraer la información mandada al crear el trabajo, en este caso el valor de inicio del contador. Recordad que este lo único que hace es generar números (bucle con contador), es en el broadcast donde se realizarán las comprobaciones de si es o no primo y se comunicará a la activity principal. Observar que la constante ACCION del intent **i**, **Línea 7**

tiene como valor una cadena única que identificará el trabajo en el `BroadcastReceiver`, usamos una variable estática para hacerlo **Línea 15**. Mandamos el Broadcast pasando como parámetros el valor del contador **Línea 9**. Para cada iteración se mandará el broadcast, por lo que irán llegando al receiver los números.

Veamos ahora la implementación de la **MainActivity.kt**:

```
class MainActivity : AppCompatActivity() {
    lateinit var mreceiver: MyReceiver
    lateinit var t1: TextView
    lateinit var t2: TextView
    lateinit var idWork:UUID
    var iniciado=false
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        t1 = findViewById(R.id.textView1)
        t2 = findViewById<TextView>(R.id.textView2)
        mreceiver = MyReceiver(t1, t2)
        val b1 = findViewById<Button>(R.id.button1)
        val b2 = findViewById<Button>(R.id.button2)
        b1.setOnClickListener { iniciarServicio() }
        b2.setOnClickListener { pararServicio() }
    }
}
```

Por ahora nada que comentar, solamente creamos el broadcast receiver y asociamos elementos a las view del layout. Implementamos también la funcionalidad de los botones, llamando a dos métodos que implementaremos a continuación.Nos crearemos también una propiedad para guardar el id del trabajo y poder acceder para su cancelación.

El botón de inicio de servicio tiene el siguiente código:

```
    fun iniciarServicio() {
2           if(!iniciado)
            {
                iniciado=true
5               val filter = IntentFilter(GeneradorNumeros.ACCION)
                this.registerReceiver(mreceiver, filter)
                val datos=Data.Builder()
8               datos.putInt("INICIO",t1.text.toString().toInt())
9               val workManager = WorkManager.getInstance(MainActivity@ this)
10              val work = OneTimeWorkRequest.Builder(GeneradorNumeros::class.java)
11                  .setInputData(datos.build())
                    .build()
                idWork=work.id
14              workManager.enqueue(work)
15
16          }
            else  Toast.makeText(this, "El trabajo ya se ha lanzado", Toast.LENGTH_LONG)
            .show()
        }
```

📌**Línea 2** se comprueba si se había iniciado con anterioridad el trabajo, para no lanzar otro si antes no ha sido parado. **Línea 5** se crea el `IntentFilter` para registrar el receiver con una cadena única que permita al BroadcastReceiver reconocer de donde viene la información, registramos el receiver añadiendo el filter creado. **Línea 9-10** se crea un `Data.Build` para pasar la información necesaria al `Work` , en este caso un entero para que el contador conserve el número por el que estaba funcionando, aunque se reinicie el trabajo. `Línea 11-14` se crea un trabaja unico, se añaden los datos anteriores y se construye. `Línea 15` se guarda el id generado por este trabajo para poder cancelarlo posteriormente. `Línea 16` se encola el trabajo para que comience su funcionamiento cuando el sistema pueda hacerlo.

El botón de parada de servicio tiene el siguiente código:

```
    fun pararServicio() {
2               if (iniciado) {
                        iniciado=false
4                       WorkManager.getInstance(MainActivity@ this)
                        .cancelWorkById(idWork)
6                       unregisterReceiver(mreceiver)
                        Toast.makeText(this, "Trabajo detenido", Toast.LENGTH_LONG)
                        .show()
                  } else Toast.makeText(this, "El Trabajo ya estaba detenido",
                                        Toast.LENGTH_LONG).show()
        }
```

📌**Línea 2**, comprobamos que el servicio todavía no está parado. Localizamos el servicio y lo cancelamos **Línea 4** y anulamos el registro del `BroadcastService()` **Línea 6**.

Pasemos ahora a definir el código del receptor **MyReceiver**, lo haremos como clase interna a MainActivity.

Cada vez que el trabajo genera un nuevo número, envía un mensaje de broadcast a través de un `IntentFilter`. El receptor se registró para atender mensajes a través de un IntentFilter donde se especifica la acción que va a atender, es por esto que cada vez que reciba un mensaje el receptor MyReceiver disparará el método `onReceive()`. Se obtendrán los datos extras del intent y se actualizarán los dos TextView. Uno de ellos se actualiza siempre y otro solamente si el contador que se ha pasado en un número primo.
Recordad que en el código de la activity principal se pasan al constructor del receptor dos parámetros, que son las referencias de las vistas que tenemos que actualizar.

```
class MyReceiver(private val t1: TextView, private val t2: TextView) : BroadcastReceiver() {
        fun esPrimo(n: Int): Boolean {
            var i: Int
            i = 2
            while (i < n) {
                if (n % i == 0) return false
                i++
            }
            return true
        }
        override fun onReceive(context: Context?, intent: Intent) {
            val s = intent.extras!!["VALORCONTADOR"].toString()
            t1.text = s
            if (esPrimo(s.toInt())) t2.text = s
        }
        init {
            t1.text = "0"
            t2.text = "0"
        }
    }
```