Temari

1) Introducció
2) Framework Android
3) Projectes Android i Android SDK
4) Activity
5) Fragments, Views i ListViews
6) Intents
7) Layouts i Custom Views
8) Resources i Themes
9) Dialogs, Menus i WebView
10) Persistència de dades
11) Tasques en Background i internet
12) SQLite i content providers
13) Notificacions

pue IMPULSANDO EL CONOCIMIENTO TIC CUALIFICADO

11 – Tasques en Background

- Threads al sistema android
- Threads i handlers
- AsyncTask
- Services

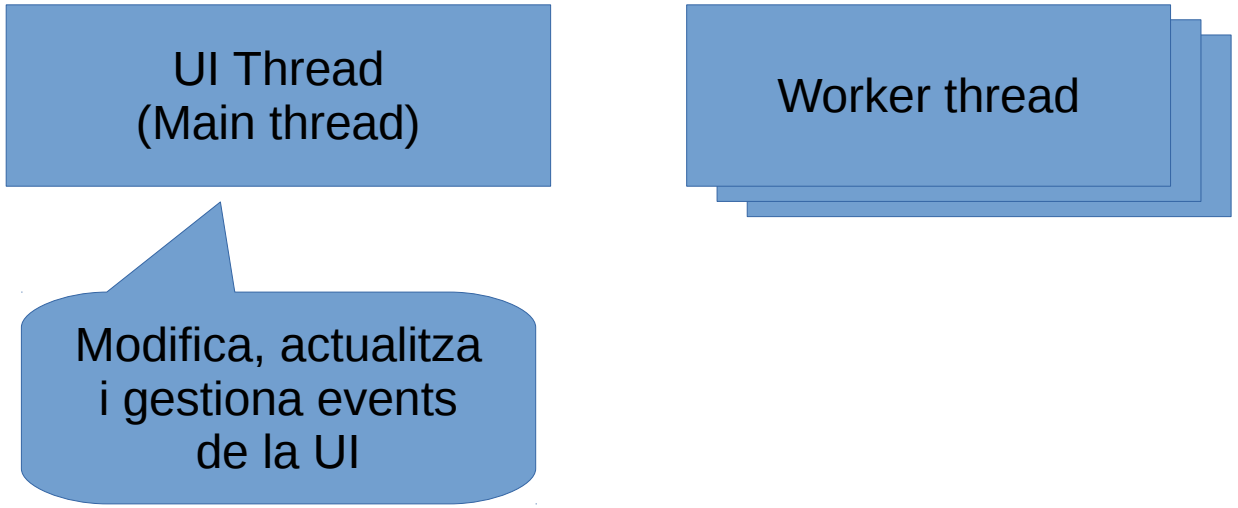
Threads al sistema android

És una característica de Java

Fils d'execució que executen codi en **paral·lel**

No hi ha garantia sobre quines operacions s'executen abans en dos threads separats

A Android diferenciem entre **Main/UI thread** i **worker threads**



Threads al sistema android

UI thread

- No es pot bloquejar → Si es bloqueja: ANR (Application Not Responding)
- Es l'unic que pot modificar la UI → Si es modifica des d'un altre thread: exception

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```


Threads i Handlers

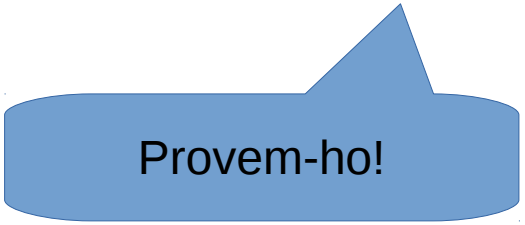
Crear un nou Thread:

Cal implimentar la interficie Runnable → te un sol metode **run()**

I passar-li a un nou objecte **Thread** al constructor

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Per aturar un thread: **Thread.sleep(ms)**



Threads i Handlers

Per modificar la UI des d'un altre thread directament

Varies opcions:

- **Activity.runOnUiThread(Runnable)**
- **View.post(Runnable)**
- **View.postDelayed(Runnable)**

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
                    loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```


Threads i Handlers

Una altra opció: **Handlers**

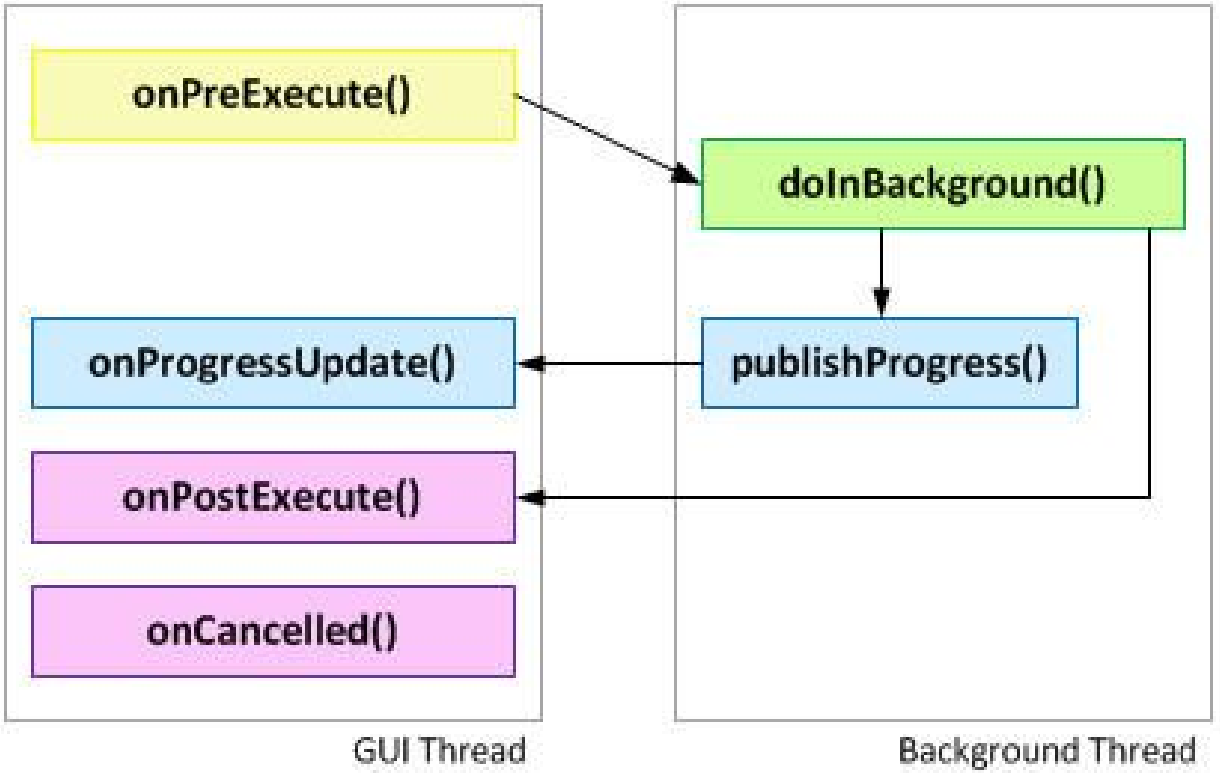
- Cal extendre la classe **Handler** → implementar el metode **handleMessage()**
- Instanciar el handler des del **UI thread** o passant-li **Looper.getMainLooper()**
- Permet executar **Runnables** amb
  - post(Runnable)
  - postAtTime(Runnable, long)
  - postDelayed(Runnable, long)
- Permeten enviar **Message** amb un Bundle, que faran que s'executi el **handleMessage al UI Thread**

Threads i Handlers

```
// Defines a Handler object that's attached to the UI thread
mHandler = new Handler(Looper.getMainLooper()) {
    /*
     * handleMessage() defines the operations to perform when
     * the Handler receives a new Message to process.
     */
    @Override
    public void handleMessage(Message inputMessage) {
        ...
    }
...
}
```


AsyncTask

Per la majoria de casos senzills, Android ofereix una classe que ho gestiona



11 – Tasques en background

AsyncTask

Cal extendre la classe, i especificar els tipus parametritzats:

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
   }
```

Provem-ho!

11 – Tasques en background

Services

Component per a fer tasques en background **amb un cicle de vida independent del component que l'ha iniciat**

→ S'executa, per defecte en el **Main thread** → cal crear un nou thread si la tasca no és immediata

→ Pot funcionar de dues maneres **no exclusives entre si**:

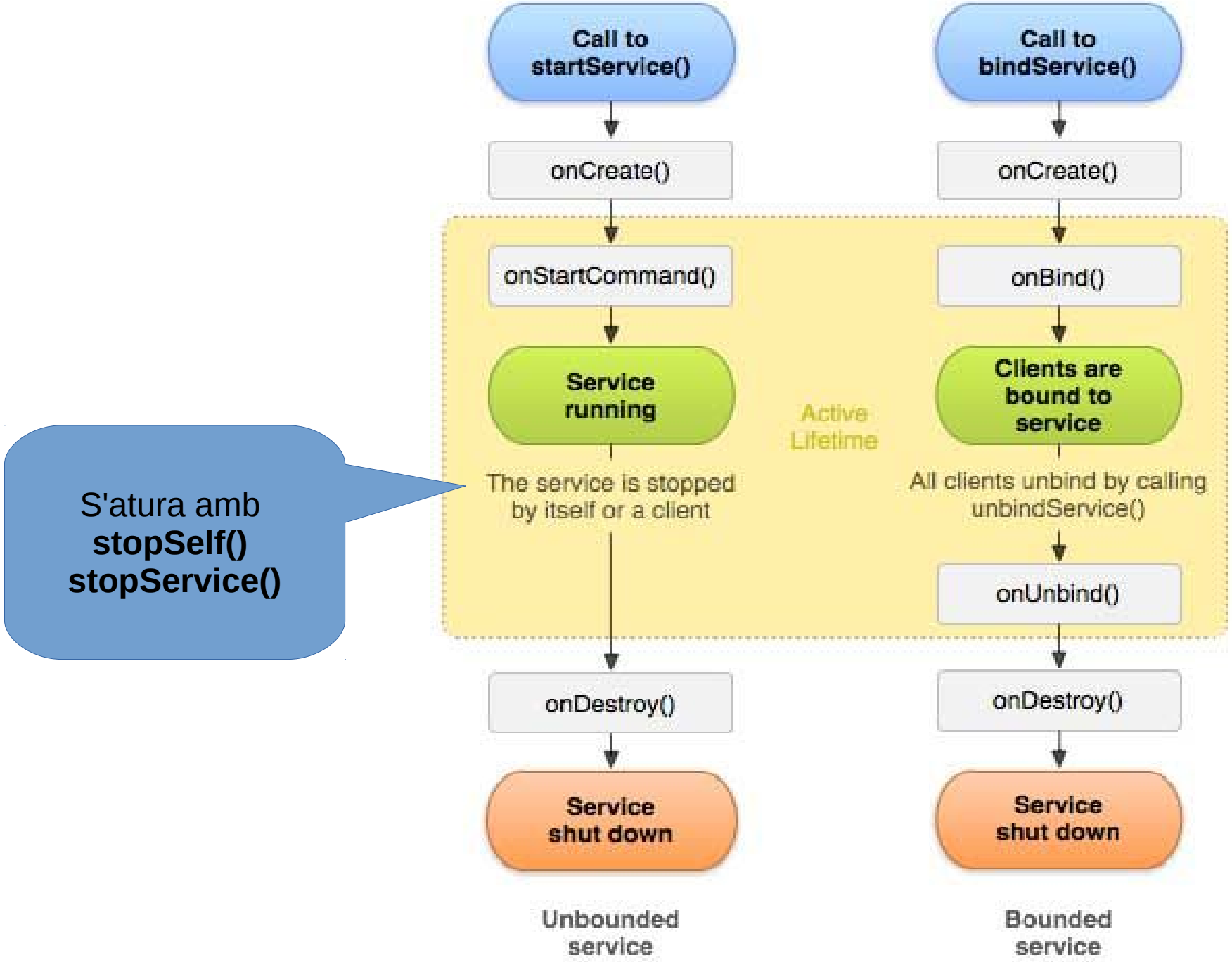
- **Started:** s'inicia per fer una tasca, i s'acaba en acabar-la. Implementar el **onStartCommand()**

- **Bound**: els altres components s'hi connecten, amb una arquitectura client-servidor. Esta actiu mentre hi hagi algun component connectat. Implementar el **onBind()**

Services

Cal declarar-lo al manifest:

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

11 – Tasques en background

Services

Iniciar un servei es fa amb un intent i **startService()** → per seguretat, sempre iniciar-lo amb un intent explicit

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Aturar un servei es fa amb:

- **stopSelf()** o **stopSelf(int id)** des del propi servei, si s'ha acabat la tasca a fer (el id per si hi ha peticions concurrents, només aturar-lo amb la ultima).
- **stopService(Intent intent)** des d'un altre component.

Services: IntentService

Subclasse de Service amb una funcionalitat habitual.
Implementa una **cua**, i va responent als Intents **d'un en un** en un **worker thread**

Si encaixa aquest comportament, només cal **extendre'l** i implementar **onHandleIntent(Intent i)**

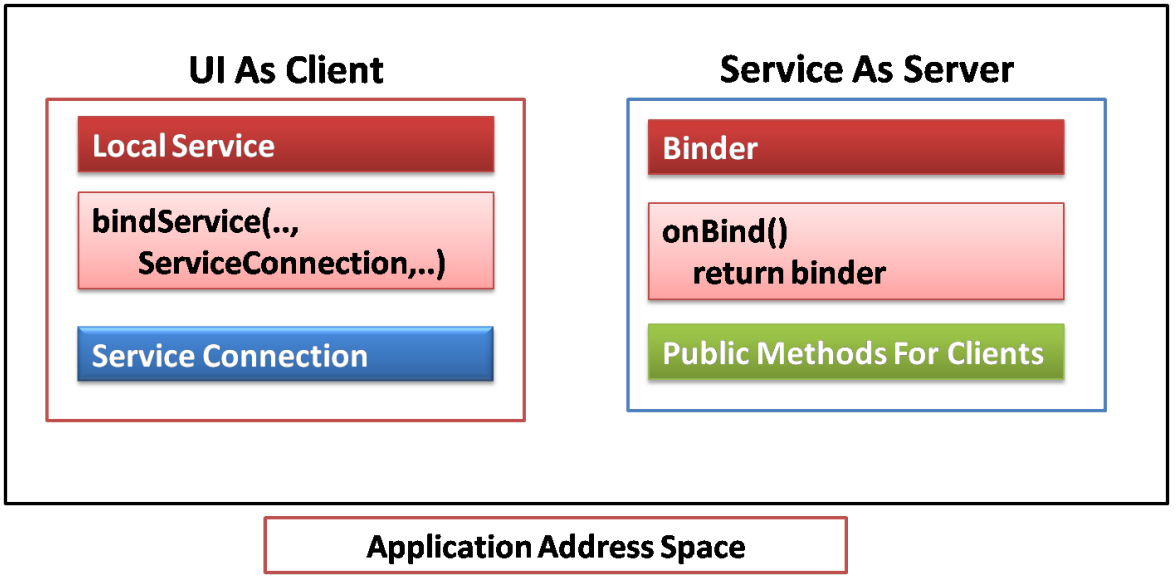
```
@Override
protected void onHandleIntent(Intent intent) {
    // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        // Restore interrupt status.
        Thread.currentThread().interrupt();
    }
}
```


Services: Bound Service

L'arquitectura **client-servidor**

Cal:

- implementar la interficie **IBInder**, que permet executar metodes al client
- El metode **onBind()** al service, que retorna el IBinder



Permet la comunicació entre processos diferents:
-Messages
-AIDL