Florida UNIVERSITÀRIA

Multitasca en Android

Programació de Serveis i Processos

2n. DAM

Multitasca en Android

- Què és?

Ens permet realitzar varias tasques de manera simultània.

http://developer.android.com/guide/components/processes-and-threads.html

Quan engeguem una app, aquesta s'executa en un únic fil, anomenat "fil principal" (en anglés, main thread) o també anomenat "fil de la Interfície Gràfica" (UI thread).

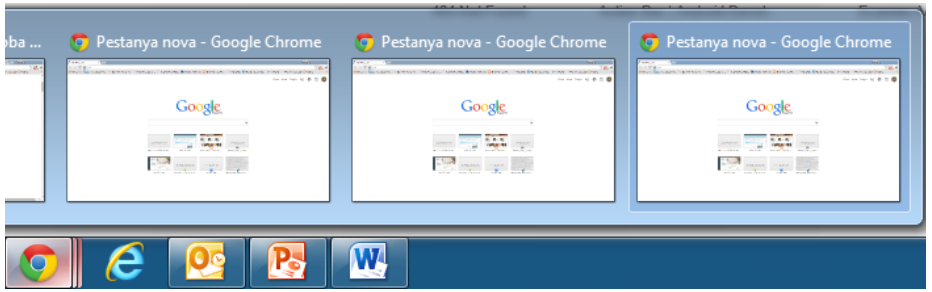
Exemple: Com preferirieu que funcionara una app que ha de carregar 100 imatges des d'internet?

1. Descarrega les 100 imatges mostrant a l'usuari una pantalla d'espera i quan les té totes les mostra
2. Conforme van descarregant-se les imatges, les va mostrant.

En Android, el S.O penalitza les activitats que tarden molt en respondre (més de 5 segons), mostrant el missatge ANR

Multitasca en Android

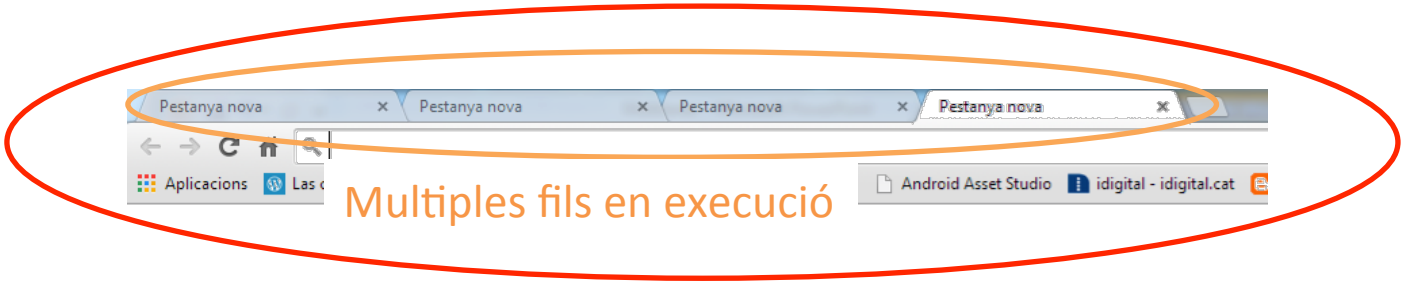
Un **procés** i un **fil** és el mateix?



Un **procés** és un programa en execució.

Podem tindre varios processos executant el mateix programa alhora.

Un **fil** és una tasca que executa algún procés. Un mateix procés pot llençar varies tasques.



Quan una app utilitza fils, aprofitem millor els nuclis dels processadors actuals (2, 4 i més nuclis), ja que cada fil es pot executar en un nucli diferent.

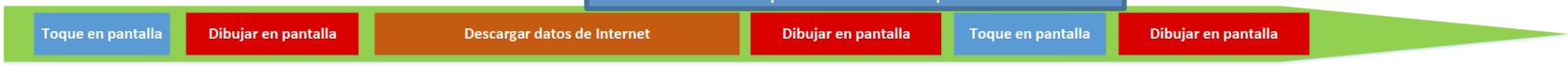
Florida UNIVERSITÀRIA

Multitasca en Android

- Treballar amb un únic fil.

El que fa l'aplicació i aprecia l'usuari

El que fa l'UI thread al llarg del temps

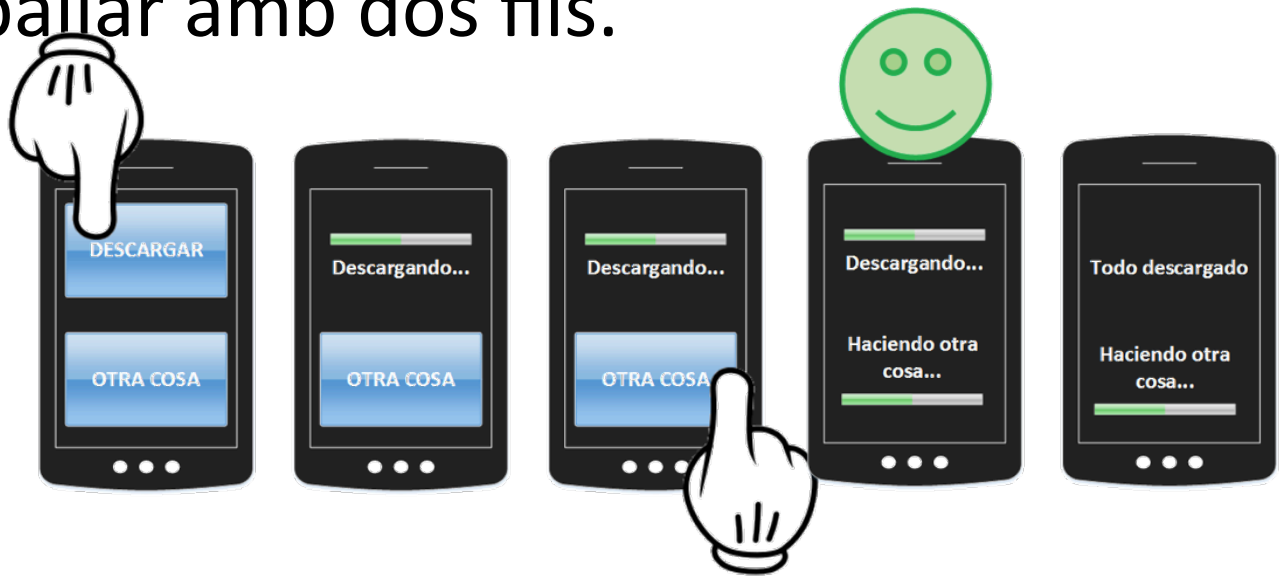


Florida UNIVERSITÀRIA

Multitasca en Android

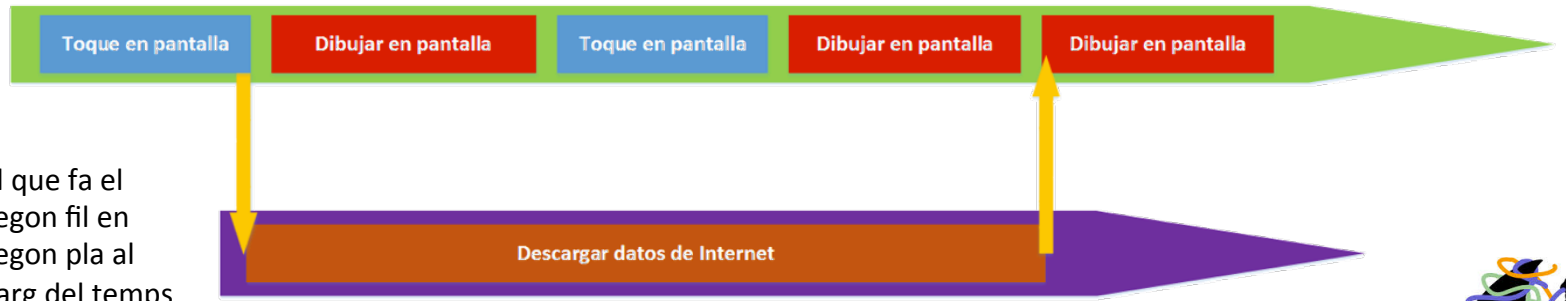
- Treballar amb dos fils.

El que fa l'aplicació i aprecia l'usuari



El que fa l'UI thread al llarg del temps

El que fa el segon fil en segon pla al llarg del temps



Florida UNIVERSITÀRIA

Multitasca en Android

- Conclusions

El **fil principal** s'hauria d'encarregar únicament de:

- **Escoltar events de la interfície d'usuari**. És a dir, fer allò que calga en el moment que ocòrrega un event.
- **Dibuixar en pantalla els view o widget**. És a dir, mostrar tots els components en pantalla a l'usuari. Todo allò que pertany a l'"Android UI toolkit".

Normes per a treballar amb el **fil principal**:

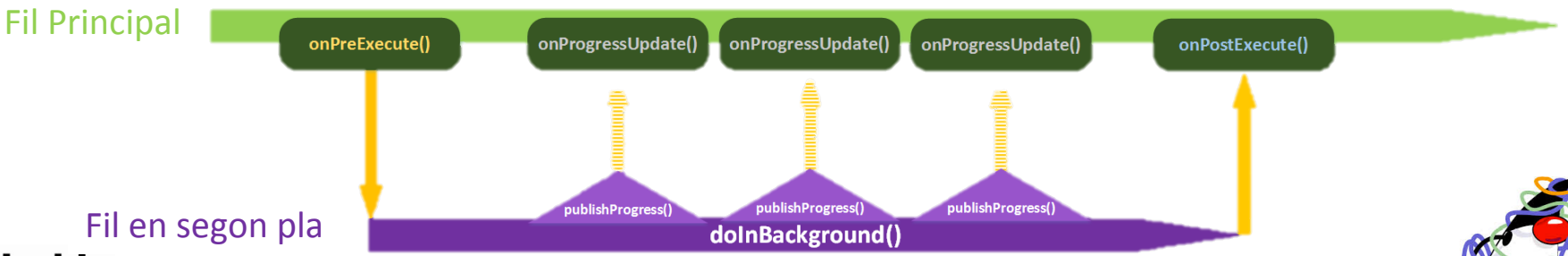
- **No bloquejar** el fil principal en cap circunstància (més de 5 segons bloquejat = ANR -Aplicació No Respòn-
- **No accedir** a res que estiga dibuixat en pantalla **desde fora del fil** principal (per exemple, cambiar el color d'un text des d'un fil en segon plà provocarà un error)

Els fils en segon plà s'haurien d'encarregar de totes les demés tasques.

Nota: Els fils poden crear altres fils.

AsyncTask en Android

- Podem crear classes que hereten d'AsynTask, per tal de facilitar-nos la feina.
- Aquestes classes:
  1. Crearan un entorn d'execució adient,
  2. Realitzaran una tasca de llarga durada
  3. Determinaran la manera en que el UI thread i la tasca es comuniquen.



AsyncTask en Android

- Quan creem una classe que hereta d'AsyncTask…
  - Només és obligatori implementar el mètode

    protected Object **doInBackground**(Object... objects){}
  - La resta de mètodes són opcionals
  - El paràmetre *Object...objects* és la manera que té Java de passar un nombre indefinit de paràmetres. S'anomena varargs.

    http://docs.oracle.com/javase/1.5.0/docs/guide/language/varargs.html

    https://geekytheory.com/varargs-o-argumentos-variables-en-java/
  - El tipus de retorn de *doInBackground()* serà el paràmetre d'entrada que rebrà el mètode

protected void **onPostExecute**(Object obj){}

AsyncTask en Android

- Quan creem un objecte d'una classe que hereta d'AsyncTask…

  - Executarem la tasca en segon pla cridant al mètode *execute()*, (tal i com ho fèiem amb els Threads en Java).

new ElMeuAsyncTask().***execute***(Object... objects);

  - Una vegada un objecte AsyncTask ha finalitzat la seua execució, **no pot tornar-se a executar!!, de la mateixa manera que ocorre amb els Threads.**
  - Les dades passades al mètode **execute()** des del fil principal, són enviades al mètode **doInBackground()** del fil secundari.
    - Per tant, internament s'accedirà de manera sincronitzada a eixes dades (amb Exclusió mútua)

Florida UNIVERSITÀRIA

AsyncTask en Android

- Quan creem una classe que hereta d'AsyncTask…
  - Només és obligatori implementar el mètode

    protected Object **doInBackground**(Object... objects){}
  - La resta de mètodes són opcionals
  - El paràmetre *Object...objects* és la manera que té Java de passar un nombre indefinit de paràmetres. S'anomena varargs.

    http://docs.oracle.com/javase/1.5.0/docs/guide/language/varargs.html

    https://geekytheory.com/varargs-o-argumentos-variables-en-java/
  - El tipus de retorn de *doInBackground()* serà el paràmetre d'entrada que rebrà el mètode

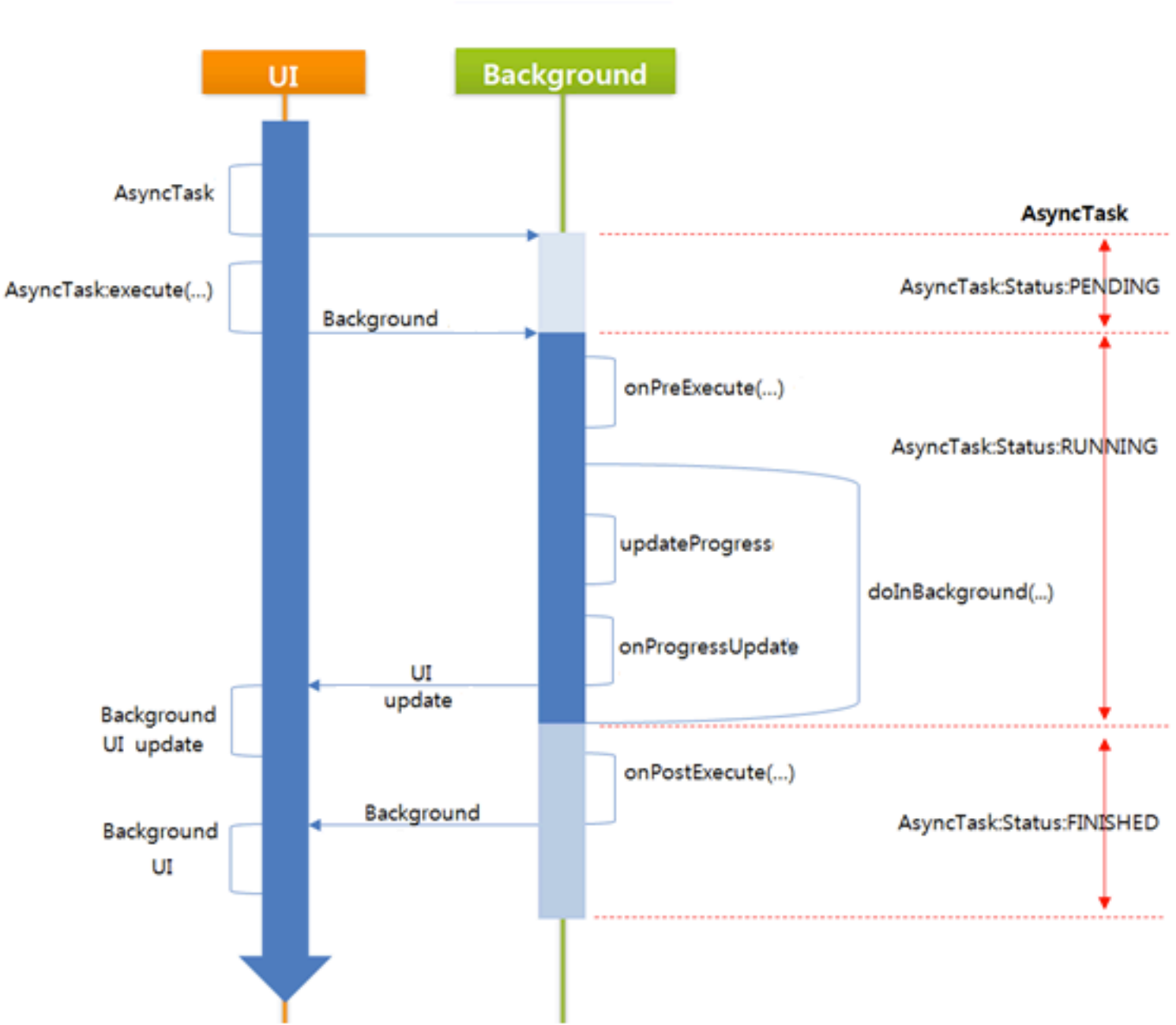
protected void **onPostExecute**(Object obj){}

AsyncTask en Android

Esquelet d'una classe AsyncTask:

```
public class LaMeuaTasca extends AsyncTask < Params, Progress, Result> {
  @Override
  protected void onPreExecute() { … }
  @Override
  protected Result doInBackground(Params… params) { … }
  @Override
  protected void onProgressUpdate(Progress… progress) { … }
  @Override
  protected void onPostExecute(Result … result) { … }
  @Override
  protected void onCancelled(Result … result) { … }
}
```

Params: Dades d'entrada per a la tasca que s'executarà en segon pla.
Progress: Dades de l'evolució de l'execució de la tasca, per a que s'actualitze la IU
Result: El resultat produït per la tasca en segón pla, i que s'enviarà a la IU.



Florida
UNIVERSITÀRIA

AsyncTask en Android

1. Creem una Instància de l'AsyncTask
2. Iniciem l'execució de la tasca.
3. Des del UI Thread, es crida al mètode onPreExecute(). Prepara la IU per a l'execució d'una tasca de llarga durada. (exemple: Creem un progressDialog).
4. Després del onPreExecute, s'executa el mètode doInBackground(), que és qui realitza la tasca en segon pla.
5. Des del doInBackground(), podem cridar al mètode publishProgress(Progress), que s'encarregarà per indicar com avança el progres de la tasca. Aquest cridarà al mètode onProgressUpdate(Progress) que és capaç d'actualitzar la IU.
6. Quan acaba la tasca, s'executa el mètode onPostExecute(Result). Aquest mètode, també pot modificar la IU (generalment tancar/destruir el ProgressDialog).
7. Però podria passar que l'usuari cancel·lara la tasca. Aleshores es cridaria al mètode onCancelled(Result). Només un dels dos mètodes descrits en aquest punt es podrà executar.

AsyncTask en Android

Podem passar paràmetres a l'AsyncTask mitjançant el mètode execute al doInBackground().

Exemple: .execute(Object… objectes);

objectes és una llista de paràmetres de qualsevol tipus.

CANCEL·LACIÓ

Quan tenim un objecte AsyncTask, podem cancel·lar-lo cridant al mètode cancel i passant-li el paràmetre true:

AsyncTasc laMeuaTasca = new LaMeuaTasca().execute(/ el que siga */);*
*laMeuaTasca.cancel(**true**);*

D'aquesta manera, iniciem una interrupció, que devem comprovar dins el mètode doInBackgrount(), mitjançant les cridades a **isCancelled();**
Tal i com feiem amb els Threads amb el mètode isInterrupted();

Que cancel·lem un AsyncTask, no implica que s'interrumpisca immediatament!!!
L'execució de l'AsyncTask no executarà el mètode onPostExecute, sino que del doInBackgrount() passarà al onCancelled().

No omplim el doInBackground de comprovacions **isCancelled()**, fem-ho al principi d'un bucle, o al principi d'una tasca llarga.

Florida UNIVERSITÀRIA

AsyncTask en Android

- Cal tindre en compte que:
  - Quan creem una classe interna AsyncTask, i aquesta està executant-se. Si es produeix la rotació de l'activity que l'executa, és responsabilitat del programador **cancel·lar** l'execució de la tasca, i actualitzar la IU.
  - Quan cancel·lem un AsyncTask, aquest no s'interromp immediatament, sinó que cal afegir punts de comprovació al mètode doInBackground(). Si no ho fem, l'AsyncTask **continuarà la seua execució encara que l'activity ja no existisca**!!!

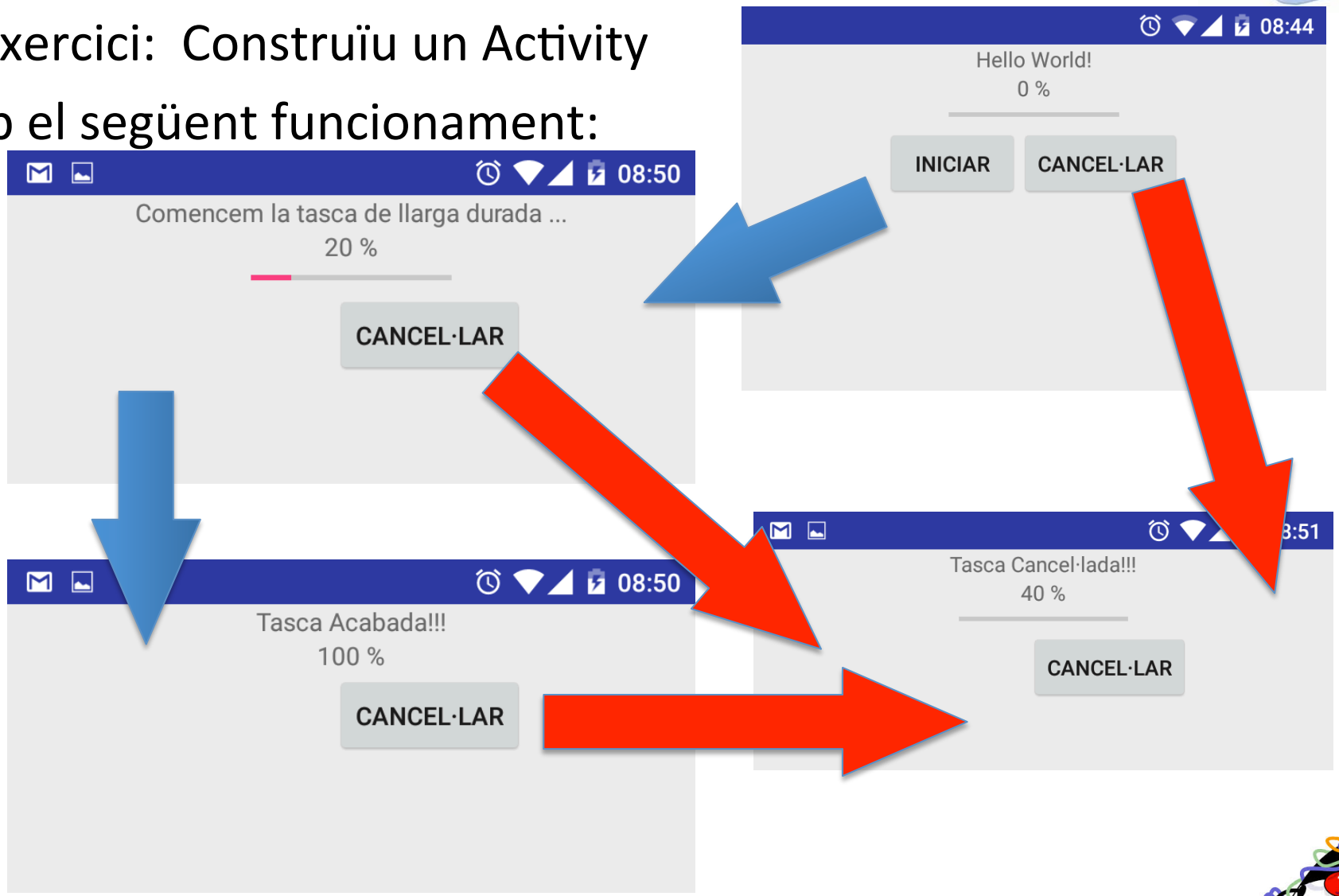
AsyncTask en Android

Cal tindre en compte que:

- Al crear un objecte d'una classe que hereta d'AsynTask, automàticament comença a executar-se el mètode **onPreExecute()**, que s'executa sobre el Fil Principal.
- Només acaba d'executar-se, es crea el fil en segon pla, i aquest realitza la seua tasca que esta implementada en el mètode **doInBackground()**
- Com que des d'un fil secundàri no podem modificar la interfície gràfica, si volguerem actualitzar algun objecte gràfic (una barra de progrés), podem fer-ho cridant des del fil secundàri al mètode **publishProgress()** i sobreescrivint el mètode **onProgressUpdate()**
- Podem cancel·lar la tasca en segon pla executant el mètode **cancel()**. Aleshores ja no s'executarà el mètode onPostExecute() sino que ho farà el **onCancelled()**.

AsyncTask en Android

- Exercici: Construïu un Activity amb el següent funcionament:



AsyncTask en Android

- Exercici (cont):

Al polsar sobre el botó **Inici**, començarà a executar-se una tasca en segon pla.

Aquesta tasta l'única cosa que farà serà, incrementar la barra de progrés durant cinc segons. I acabarà.

Si en qualsevol moment l'usuari polsa el botó **Cancel·lar**, acabarem l'AsyncTask.

En qualsevol moment l'usuari estarà informat del que està ocorreguent. Per això, utilitzarem dos TextView.

- El primer mostrarà els següents missatges:

  · Hello World!! -> Quan es carrega l'activity, i encara no s'ha polsat res.

  · Comencem la tasca de llarga durada -> Mentre s'esta executant l'AsyncTask.

  · Tasca Acabada -> Quan l'AsyncTask acaba de manera natural.

  · Tasca Cancel·lada -> Quan l'usuari cancel·la l'AsyncTask.

- El segon TextView, mostrarà el percentatge de les accions realitzades.
- La barra de progrés anirà incrementant-se a mesura que avança l'execució.

AsyncTask en Android

- Podeu descarregar-vos els layouts necessaris als següents enllaços:
  - content_inici.xml: https://drive.google.com/open?id=0B5RUHd05GgKbTmc4V2JjNEdIOVk
  - activity_inici.xml: https://drive.google.com/open?id=0B5RUHd05GgKbN3ByOGNHV1JNcWs

Problema

- **Què ocòrre quan rotem el terminal?**
- **L'AsyncTask continua executant-se en la RAM del terminal**.
- L'activity que l'havia llençat s'ha destruït i s'ha tornat a crear (en altra orientació).
- Al declarar l'asyncTask com una inner class, aquesta guarda una referència de la classe que el conté (que s'ha perdut, aldestroir-se l'activity).
- Per tant, el progressbar, i els textView que utilitzavem per a mostrar el progrés, s'han destruït i se n'han creat altres nous.
- Per eixes raons no s'actualitzen.

Sol·lució

- Si la tasca la generem amb una inner class, assegurent-no que aquesta és **static**, per a evitar tindre "filtrats de memòria"
- Vegeu: https://developer.android.com/topic/performance/threads.html#implicit
- Per tant, els components de l'activity que actualitzen des de dins l'asyncTasc, i els mètodes de l'activity que utilitzem dins l'assynkTask, també seràn **static**.

Sol·lució (cont)

- Utilitzarem els mètodes:
  - public Object onRetainCustomNonConfigurationInstance()
  - getLastCustomNonConfigurationInstance()
- El primer per a evitar perdre la referència de la tasca que esta desenvolupant-se.
- El segon per a recuperar la referència de la tasca.
- Podeu descarregar l'exemple:

https://github.com/mvielcor/BasicAsyncTask

```
public class Inici extends AppCompatActivity {

   LaMeuaTasca lmt;
//aquest mètode s'executa abans de destruir l'Activity que s'ha rotat
   public Object onRetainCustomNonConfigurationInstance(){
    //lmt és la referència a la tasca
      if(lmt!=null){
        lmt.activityAQuiPertany=null; //trenquem la ref. a l'activity
                        //que es va a destruïr. Com és un WeakReference,
                        //el garbage Collector la podrà eliminar
        return lmt;
      }else{
         return super.onRetainCustomNonConfigurationInstance();
      }

.....
}
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
//Comprovem si ja s'ha llençat prèviament la tasca

lmt = (LaMeuaTasca)getLastCustomNonConfigurationInstance();
if (lmt==null){  //No hi ha cap tasca llençada anteriorment
   lmt = new LaMeuaTasca(this); // Per tant, en crearé una nova,passant-li la ref.de
                                // l'Activity actual

 }else{
    // La tasca ja s'ha llençat previament
   lmt.activityAQuiPertany=new WeakReference(this); //Li passe una referència a
                                                    //l'actual Activity
   botoIniciar.setVisibility(View.INVISIBLE);
}


.....
}
```


- I la nostra tasca quedaría així:

```
private static class LaMeuaTasca extends AsyncTask<Void, Object, Void> {
    WeakReference activityAQuiPertany;   // Guardem una referència
de l'activity que llença la tasca

    public LaMeuaTasca(Activity a){

        super();
        activityAQuiPertany=new WeakReference(a);
    }
....
//onPreExecute(), doInBackground(), ...
```