Programmation avec android Cours 3

Hadjila Fethallah

Maître de Conférences au Département d'Informatique

F_hadjila@mail.univ-tlemcen.dz

Multi threading

- tâche
  - Ensemble d'activités qui appartiennent à la meme application/processus ou non
  - Secnario d'execution géré par l'application "activity manager"
- processus
  - le lancement d'un composant d'une application cree un nouveau processus (si cette app n' a pas deja de composant(s) en cours d'execution)
  - Par defaut une app= un seul processus avec un seul thread (UI thread ou Main Thread ou looper thread)
  - Le developpeur peut assoicier des processus separés à des composants de la meme app (android:process)
  - La verification de l'appartenance des composants par rapport aux processus est assurée par : (int pid = android.os.Process.myPid();)
- Thread (d'un processus)
  - Flux d'instructions qui s'execute en parallèle avec d'autres flux et qui partage avec eux les variables statiques et le tas (heap) mais qui a sa propre pile et son propre compteur ordinal.

Threads

- Ui thread vs background thread
  - l'Ui-thread reagit aux evenements de l'utilisateur (interaction avec les widgets)
  - contient les methodes du cycle de vie des activités
  - Si un widget ne reagit pas au bout de 5 sec, alors android affiche un Boite de Dialogue proposant l'arret de cette app
  - Il est toujours conseillé de ne pas bloquer le UI thread avec une longue tâche qui s'éxecutant dans le même thread (ex dans onCreate)
- Exemple de tâches consommatrices de temps
  - : téléchargement, requête BDD, chargement d'images,….
- "UI toolkit " n'est *pas* " thread-safe ".
- Obtention de l'ID d'un thread est faite par:
  - android.os.Process.myTid(); ou Thread.currentThread().getId();

Priorité des processus

- 1.Processus en avant plan (activité en interaction utilisateur, service attaché à cette activité, **BroadCastReceiver** exécutant **onReceive()**)
- 2. Processus visible: il n'interagit pas avec l'utilisateur mais peut influer sur ce que l'on voit à l'écran (activité ayant affiché une boite de dialogue (**onPause()** a été appelée), service lié à ces activités "visibles").
- 3. Processus de service
- 4. Processus tâche de fond (activité non visible (**onStop()** a été appelée))
- 5. Processus vide (ne comporte plus de composants actifs, gardé pour des raisons de *cache*)

Durée de vie d’un thread

- Si la tâche concurrente (thread) est lancée:
  - par l'activité principale: sa vie durera le temps de l'activité
  - par un service: la tâche survivra à l'activité principale

Classes /fonctions impliquées

- Thread (implementant l'interface Runnable)
  - New thread (new runnable() {…})
  - Start()
  - Sleep(…)
  - Notify()
- Runnable
  - Run ()
- Activity.runOnUiThread(new runnable() {…}))
- View.post(new runnable() {…})
- AsychTask
  - onPreExecute() UI thread
  - Results doInBackground(param):BG thread
  - Publishprogress(progress…) B.G thread → onProgressUpdate(progress…) UI thread
  - onPostExecute(result) UI thread

Classes impliquées dans les threads

- **Handler**
  - **sendMessage(msg)**
  - **sendMessageAtFrontOfQueue(msg)**
  - **sendMessageAtTime(msg)**
  - **sendMessageDelayed(msg)**
  - **handleMessage(Message msg)**
  - **Post(new runnable() {….})**
  - **PostAtTime(runnable r, long timeMillis)**
  - **PostDelayed(runnable r, long timeMillis)**
  - **ObtainMessage() // plusieurs signatures**

1ere approche du multithreading

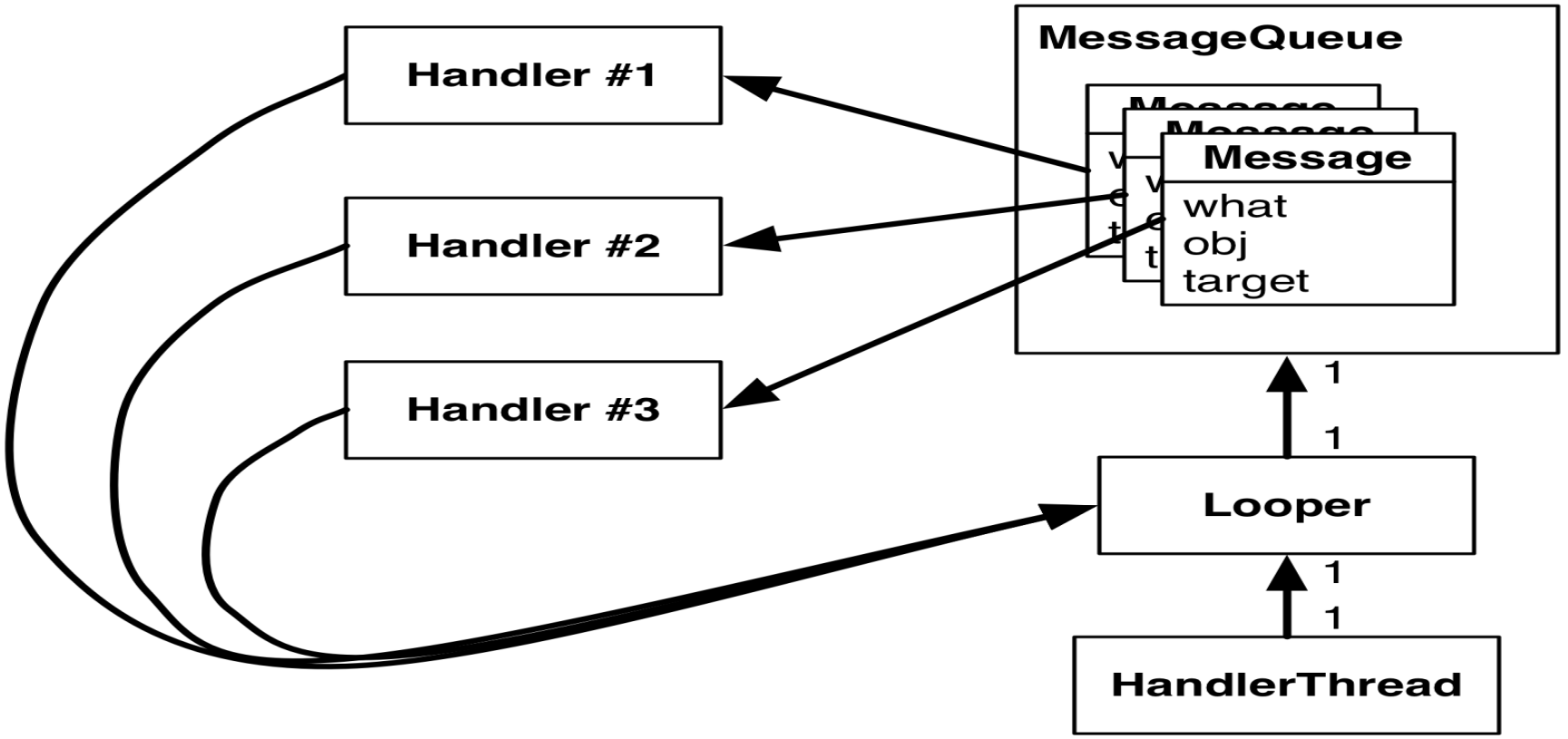
```
public void onClick(View v) {
Thread th = new Thread(new Runnable() {
public void run() {
…….
runOnUiThread(new Runnable() { public void run()
{ImageView image = (ImageView)
findViewById(R.id.imageView1);
image.setImageResource(R.drawable.ic_action_call);}
 ……..
});  th.start();  }
}}
```

2eme approche du multithreading

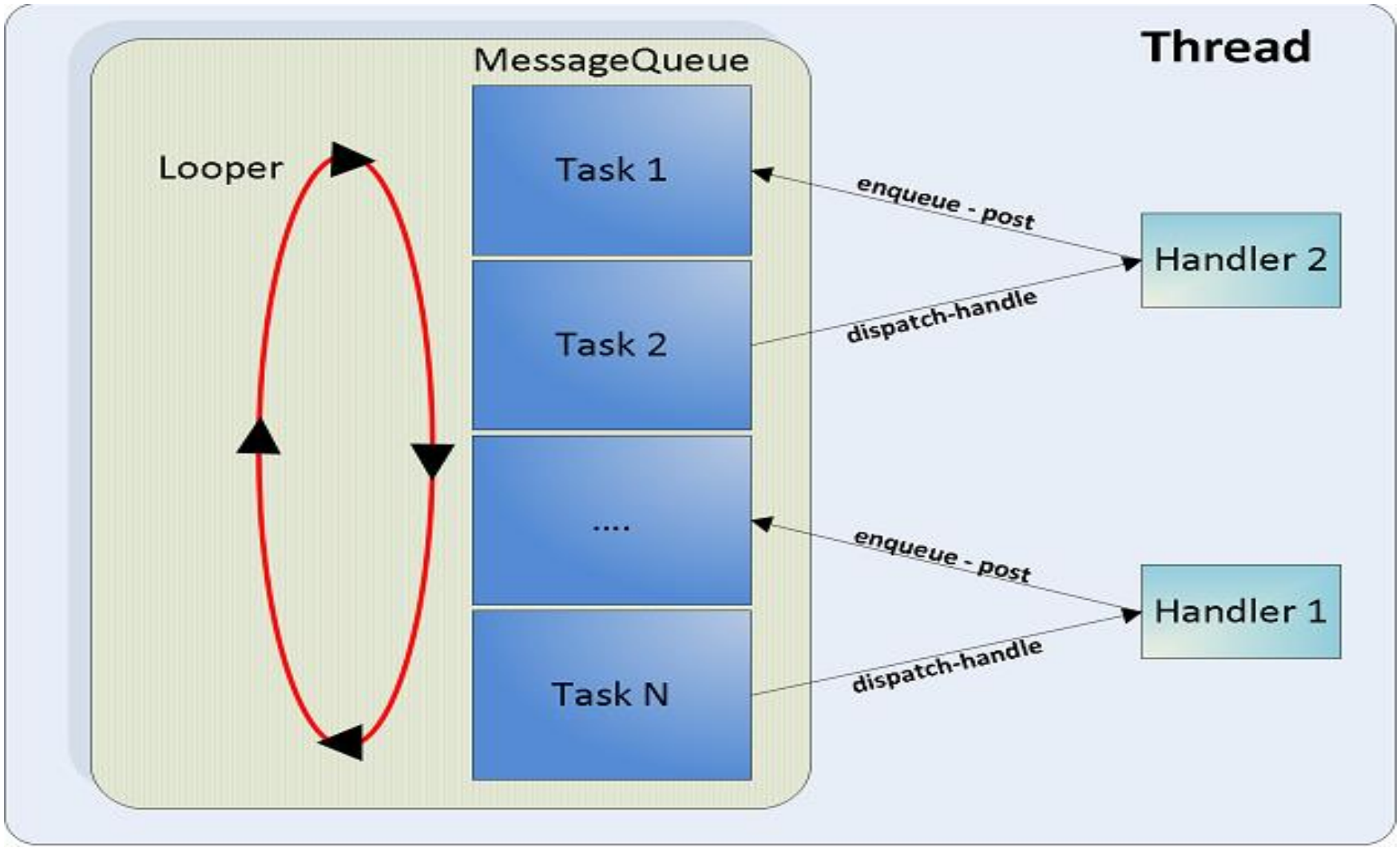
- la classe Handler permet la communication (ou délégation) des messages/taches à exécuter(runnable) entre les threads (pas forcement le UI thread et BG thread)
- un handler est associé à un seul thread
- Runnable: c'est une interface contenant le code à exécuter par le thread traiteur(le thread émetteur connait les étapes de traitement)
- Message :classe contenant plusieurs attributs: code-message, donnée de type object, 02 arguments entiers (le thread émetteur ne connait pas l'implementation de l'operation faite par le thread traiteur)

2eme approche du multi-threading

- Chaque thread contient un file d'attente « message queue » contenant des messages ou des runnables et un objet looper qui permet de gérer cette file



2eme approche du multi-threading



Tâches du looper

- Le looper route les elements de la file d'attente aux classes concernées
- Le traitement se fait selon la politique fifo
- Dans le cas d'un message il invoque la fonction du handlemessage() du handler
- Dans le cas d'un runnable il invoque simplement la fonction run() de cette classe

Exemple détaillé

- Voir le TP4

3eme approche du multi-threading

- AsynTask attend trois paramètres :
- Le type de l'information qui est nécessaire au traitement (ex:**URL,String** ,…)
- Le type de l'information qui est passé à sa tâche pour indiquer sa progression (**ex:Integer,Void,**…)
- Le type de l'information passé au code lorsque la tâche est finie (ex: **Long,Integer,…**).

exemple

public class MyAsyncTask extends AsyncTask<String, Void, Integer> {…. }

3eme approche du multi-threading

…….

protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main); MyAsyncTask myTask = new MyAsyncTask(this);

String param= ″ ` ;

myTask.execute(Param); }

AsynchTask vs Handler

- L'utilisation de AsynTask est plus facile que celle des Handlers
- AsynchTask est utile pour faire une communication entre un thread worker et un UI thread
- Avec AsynchTask plusieurs tâches s'executent dans le même thread sauf si on utilise l' option ThtreadPoolExecutor
- Avec AsynchTask on doit respecter le workflow par défaut (cad la tâche de fond est executée une seule fois et les points de communication avec le UI thread sont limités)
- Avec Handler, on peut assurer la communication entre 02 threads workers ou un thread worker et un autre Main.
- Les scenarios de communication avec les threads workers sont plus flexibles (échanges illimités)→utile pour les tâches répétitives

FIN du Cours3