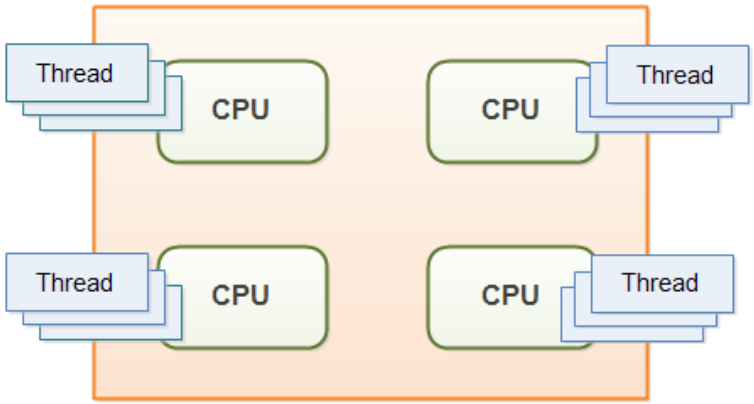
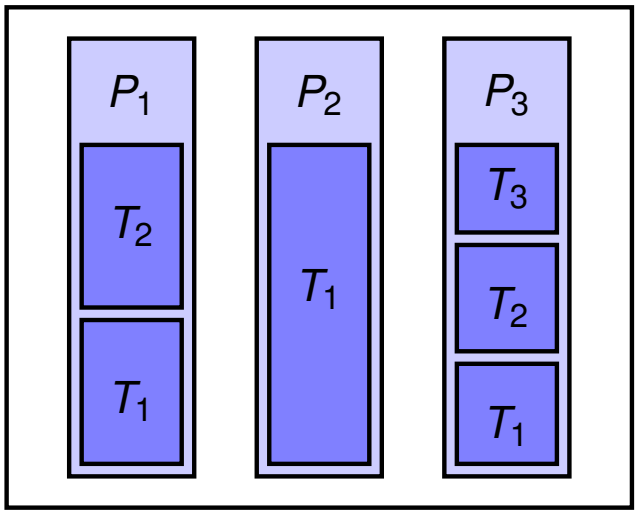
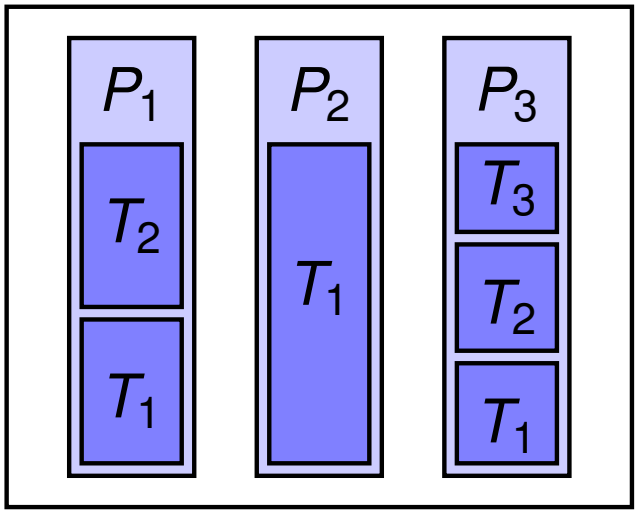
Effectuer des calculs en tâche de fond : AsyncTasks





Multithreading

- aujourd'hui les matériels sont très performants et sont souvent dotés de plusieurs coeurs permettant l'exécution de plusieurs Threads simultanément
- chaque Thread possède sa propre pile d'exécution (variables qui lui sont propres)
- partage des espaces de mémoire avec les autres Thread du même processus système



UI Thread

- les applications Android ont un Thread principal : le UI Thread
- Ce Thread a la charge de la gestion de l'interface utilisateur
  $\rightarrow$ interaction avec l'utilisateur
- problème : si une Activity ne réagit pas à une interaction avec l'IU (par exemple une touche pressée) dans un délai de 5 secondes (10s pour certains composants), l'Activity sera détruite par l'ActivityManager

  $\rightarrow$ ceci est nommé ANR (Application Not Responding).
- cette situation doit être évitée

  $\Rightarrow$ le UI Thread ne doit pas exécuter de longues opérations

UI Thread et threads en tâche de fond

- toute tâche intensive doit être exécutée dans un autre Thread en arrière plan (ou tâche de fond),
- types de tâches intensives : opérations de communication (réseau), traitement d'images, gestion de bases de données, etc.
- exécuter une tâche dans un autre Thread est simple :

  $\rightarrow$ new Thread(new Runnable() {...}).start(); (cf. cours Programmation répartie)

$\rightarrow$ *un exemple*

Comment exécuter une très longue opération?

Situation

- On considère une interface graphique composée d'un bouton et d'un TextView
- le bouton permet de lancer l'exécution d'une longue opération et, lorsque l'opération se termine, la réponse doit apparaître dans le TextView
- Nous nous intéressons à plusieurs solutions potentielles

Première (mauvaise) solution

- implémenter directement la méthode dans la méthode appelée lorsque le bouton est pressé (android:onClick dans le layout.xml)

```
public void doLongComputation(View v) {
    Random alea = new Random(System.currentTimeMillis());
    // we simulate the computation by this sleeping time
    try {
        Thread.sleep(10000);
    } catch (InterruptedException ie) {}
    // after the computation we get the solution
    tv.setText("generated number "+alea.nextInt(100));
}
```

- vous avez une erreur de type ANR (app not responding) ou
- durant l'exécution du votre méthode l'interface devient inopérante

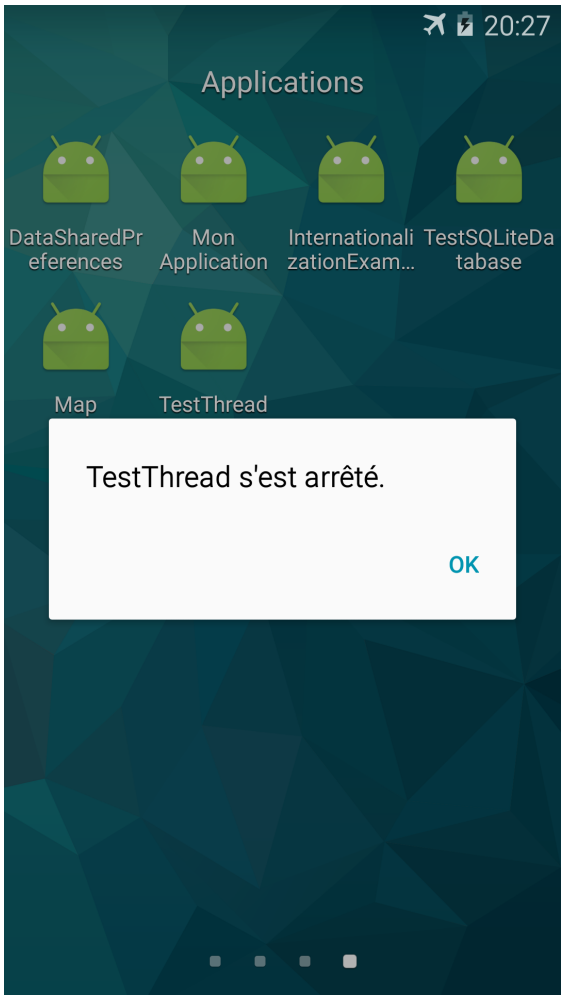
Seconde (mauvaise) solution

- effectuer l'opération dans un Thread dédié

```
public void doLongComputation(View v) {
    new Thread(new Runnable() {
        public void run() {
            Random alea = new Random(System.currentTimeMillis());
            // we simulate the computation by this sleeping time
            try { Thread.sleep(6000); } catch (InterruptedException ie) {}
            // after the computation we get the solution and display it on tv
            tv.setText("generated number "+alea.nextInt(100));
        }
    }).start();
}
```

Seconde (mauvaise) solution

```
FATAL EXCEPTION: Thread-1415
Process: mobapp.testthread, PID: 12298
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
```



- encore pire que précédemment
- pourquoi ?

Seconde (mauvaise) solution

```
FATAL EXCEPTION: Thread-1415
Process: mobapp.testthread, PID: 12298
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
```

- Violation de la règle :

  les Views ne peuvent être accédés QUE par le UI Thread

Une bonne solution **runOnUiThread()**

- Android met à notre disposition une *helper* méthode pour exécuter certaines instructions dans le contexte du Main UI Thread

  $\longrightarrow$ void runOnUiThread (Runnable)
- ce code est inséré dans un nouveau Thread

```
new Thread(new Runnable() {
        public void run() {
                long calcul
                runOnUiThread(new Runnable() {
                        public void run() {
                                changement de l'UI
                        }
                )};
        }
}).start()
```

RunOnUiThread

Première solution

```
public void doLongComputation(View v) {
    new Thread(new ComputingThread2()).start();
}

private class ComputingThread2 implements Runnable {
    @Override
    public void run() {
        Random alea = new Random(System.currentTimeMillis());
        try {
            Thread.sleep( millis: 2000);
        } catch (InterruptedException ie) { ie.printStackTrace(); }
        final int newInt = alea.nextInt( bound: 1000);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                tv.setText("the number is: "+newInt);
            }
        });
    }
}
```

RunOnUiThread

Seconde solution

```
public void doLongComputation(View v) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            Random alea = new Random(System.currentTimeMillis());
            try {
                Thread.sleep( millis: 2000);
            } catch (InterruptedException ie) { ie.printStackTrace(); }
            final int newInt = alea.nextInt( bound: 1000);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    tv.setText("number:: "+newInt);
                }
            });
        }
    }).start();
}
```

UI Thread et Thread en arrière plan

- problème principal : les interactions entre l'UI Thread et les autres Thread
- en effet vous devez être conscients que les views dans une application ne peuvent être accédées que par l'UI Thread
- lorsque le code est compliqué ou lorsque l'on souhaite avoir une idée de la progression de l'exécution, la solution précédente n'est pas forcément la mieux adaptée,
- heureusement, Android fournit d'autres solutions pour implémenter ces interactions, parmi lesquelles

  $\longrightarrow$ **AsyncTask**

AsyncTask

- les AsyncTask font partie des différentes solutions fournies par Android pour exécuter de longues opérations,
- les AsyncTask fournissent une manière simple et transparente d'exécuter des tâches en arrière plan
- considérons une application simple composée de deux boutons et de deux TextView
- chaque bouton correspond à l'exécution d'une méthode :
  - Button 1 $\rightarrow$ méthode 1 simule une opération rapide
  - Button 2 $\rightarrow$ méthode 2 simule une longue opération
- chaque méthode affiche le résultat dans les TextView (le nombre de clics)

AsyncTask

$\Longrightarrow$ étendre la classe générique AsyncTask

- qui a quatre méthodes principales

```
class AsyncTask<XXX,YYY,ZZZ> {
        void onPreExecute(...) {...}
        ZZZ doInBackground(XXX...) {...}
        void onProgressUpdate(YYY...) {...}
        void onPostExecute(ZZZ) {...}
}
```

seule la méthode doInBackground() est requise

AsyncTasks

- une façon de procéder est d'étendre la classe générique AsyncTask :

```
class AsyncTask<Params,Progress,Result>
{...}
```

- Params: type des données utilisées par le calcul en arrière plan
- Progress: optionnel, le type des données qui indiquent la progression du calcul
- Result: le type du résultat qui sera exploité dans l'UI Thread

AsyncTasks

```
public class myAT extends AsyncTask<Params,Progress,Result> {
        void onPreExecute() {...}
}
```

- contient des opérations qui doivent être exécutées avant le long calcul
- souvent utilisé pour l'initialisation de la longue opération,
- exécutée par l'UI Thread, donc cette méthode peut accéder aux Views

AsyncTasks

```
public class myAT extends AsyncTask<Params,Progress,Result> {
        void onPreExecute() {...}
        Result doInBackground(Params... p) {...}
}
```

- la partie du calcul qui nécessite du temps
- exécutée en arrière plan (dans un autre Thread)
- le resultat du calcul est retourné comme un Object du type Result (Integer, Double, String, Bitmap, etc.)

AsyncTasks

```
public class myAT extends AsyncTask<Params,Progress,Result> {
        void onPreExecute() {...}
        Result doInBackground(Params... p) {
                publishProgress(values);
        }
}
```

- au cours de l'exécution de la longue opération, cette méthode peut être invoquée à l'intérieur de la méthode `doInBackground()`
- les paramètres de cette méthode peut être un ensemble de variables qui indique l'état d'avancement du calcul
- un appel à cette méthode entraîne un appel à onProgressUpdate() (qui doit être implémentée par le développeur)

AsyncTasks

```
public class myAT extends AsyncTask<Params,Progress,Result> {
        void onPreExecute() {...}
        Result doInBackground(Params... p) {
                publishProgress(values);
        }
        void onProgressUpdate(Progress... vals) {...}
}
```

- exécuté dans l'UI Thread ⇒ cette méthode a donc accès aux Views

AsyncTasks

```
public class myAT extends AsyncTask<Params,Progress,Result> {
        void onPreExecute() {...}
        Result doInBackground(Params... p) {
                publishProgress(values);
        }
        void onProgressUpdate(Progress... vals) {...}
        void onPostExecute(Result result) {...}
}
```

- également exécutée par l'UI Thread $\Rightarrow$ a donc un accès aux Views
- l'appel à cette méthode correspond également à la fin de la méthode du calcul en arrière plan

Parameters

- au moment de la déclaration de la classe, le type des paramètres doit être renseigné :

  class Blabla extends AsyncTask<String,Double,Integer>
- le premier paramètre correspond au type de la donnée utilisée par le calcul dans la méthode doInBackGround() :

  protected Integer doInBackground(String... values)

  → le type du résultat est Integer (troisième paramètre de la déclaration)

  → les valeurs sont regroupées dans un tableau de String Array of String

AsyncTasks

Parameters

- au moment de la déclaration de la classe, le type des paramètres doit être renseigné :

  class Blabla extends AsyncTask<String,Double,Integer>

- le second paramètre est utilisé dans la méthode onProgressUpdate() pour informer l'utilisateur sur l'avancement du calcul :

  protected void onProgressUpdate(Double... val)

  $\rightarrow$ val est un Array de Double

AsyncTasks

Usage

- une manière simple de procéder consiste à créer une classe interne (MyAsyncTask) dans votre classe Activity et de l'instancier
- sur la nouvelle instance il faut ensuite invoquer la méthode MyAsyncTask.execute(Params)

Exemple

```
class AsyncTaskForMethodTwo extends AsyncTask<XXX,YYY,ZZZ> {

    protected void onPreExecute() { }

    @Override
    protected ZZZ doInBackground(XXX... xxx) {
        ZZZ result = null;
        YYY progress = null;
        publishProgress(progress);
        return result;
    }

    protected void onProgressUpdate(YYY... yyy) { }

    protected void onPostExecute(ZZZ res) { }

}
```

il faut ensuite répondre à quelques questions... lesquelles ?

Exemple

Questions

1. Quel doit être le type de Params, Progress et Result?
2. Quelles sont les opérations qui doivent être exécutées par onPreExecute()?
3. Quelles sont les opérations qui doivent être exécutées par doInBackground()?
4. Quel type de données doivent être transmises à `onProgressUpdate()` par l'intermédiaire de publishProgress()?
5. Quelles sont les opérations qui doivent être exécutées par onProgressUpdate()?
6. Quelles sont les opérations qui doivent être exécutées par onPostExecute()?

Exemple d'utilisation

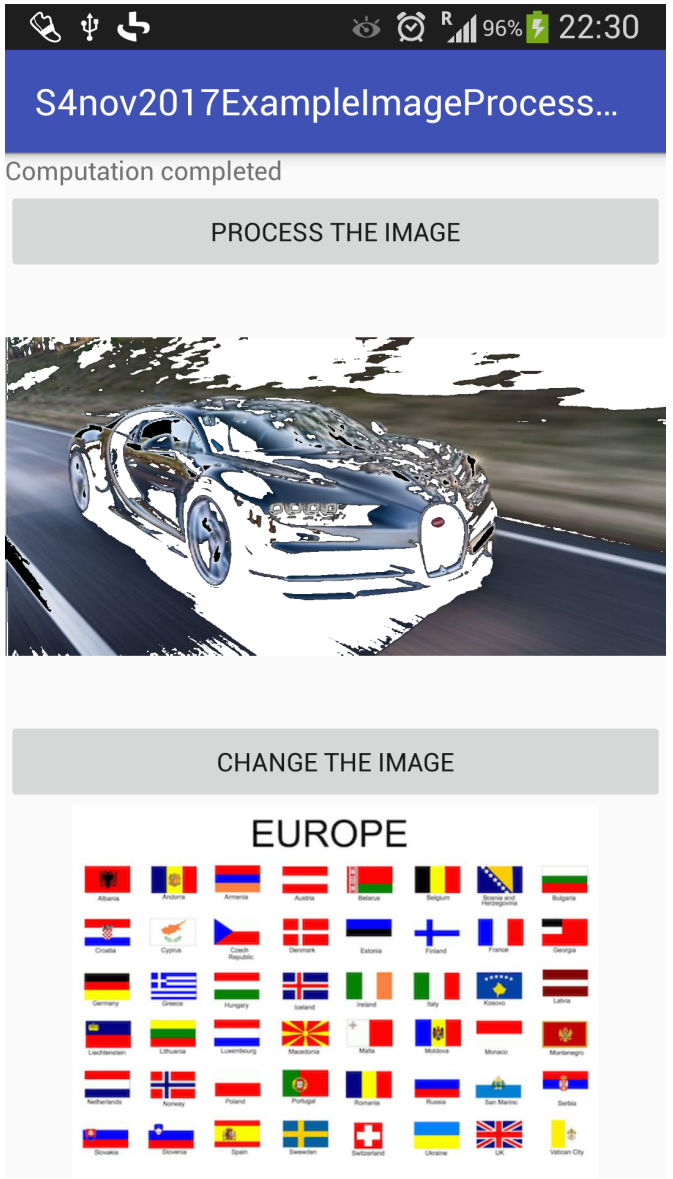
```
class AsyncTaskForLongOperation extends AsyncTask<Integer,Double,Integer> {

    protected void onPreExecute() {
        // I change the value of the TextView to "computing..."
        tvm2.setText("Computing...");
    }

    protected Integer doInBackground(Integer... values) {
        Integer val = values[0]*values[0];
        // simulates a long operation of several seconds
        try {
            for(int i=0;i<30;i++) {
                Thread.sleep(100);
                publishProgress(10*(double)i/3);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        publishProgress(100.0);
        return val;
    }

    protected void onProgressUpdate(Double... values) {
        int intval = values[0].intValue();
        tvm2.setText("task completed at " + intval + "%");
    }

    protected void onPostExecute(Integer vals) {
        tvm2.setText("Nb of clicks: " + nbClicksOnButton2 + " val=" + vals);
    }

}
```

Remarque : annulation/suspension d'une AsyncTasks

- la tâche peut être supprimée en effectuant un appel à la méthode cancel(boolean) sur l'instance de AsyncTask.
- dans une telle situation, la méthode onCancelled() est invoquée après la fin de l'exécution de la méthode doInBackground() à la place de la méthode onPostExecute(), ce qui veut dire qu'il n'y a pas de remontée de résultat.
- Pour accélérer le processus de terminaison il est possible au sein de l'AsyncTask de vérifier périodiquement la valeur retournée par un appel à isCancelled().

AsyncTask et traitement d'images

- utilisé des AsyncTasks pour le traitement d'images est un choix pertinent
- nous considérons une app avec 2 boutons et deux ImageView
- le premier bouton déclenche le traitement d'une image
- le second bouton permet de choisir une nouvelle image



Traitement d'images

Layout

```
<Button android:text="Process the image" android:onClick="processImage"
    android:layout_width="match_parent" android:layout_height="wrap_content" />

<ImageView android:id="@+id/imageTwo"
    android:layout_width="match_parent" android:layout_height="wrap_content" />

<Button android:text="Change the image" android:onClick="changeImage"
    android:layout_width="match_parent" android:layout_height="wrap_content" />

<ImageView android:id="@+id/imageOne"
    android:layout_width="wrap_content" android:layout_height="wrap_content" />
```

Traitement d'images

Ajouter une ressource à une ImageView

```
ImageView ivOne, ivTwo;
Random alea;
TextView tvProgress;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ivOne = (ImageView)findViewById(R.id.imageOne);
    ivTwo = (ImageView)findViewById(R.id.imageTwo);
    tvProgress = (TextView)findViewById(R.id.tvComputationProgress);
    alea = new Random();
}


public void changeImage(View v) {
    int id = 1+alea.nextInt( bound: 10);
    int imageID = getResources().getIdentifier( name: "image_"+id,
            defType: "drawable", getPackageName());
    ivOne.setImageResource(imageID);
}
```

Traitement d'images

Modifier une image

- la méthode `BitmapFactory.decodeResource()` produit une image Bitmap non modifiable
- pour permettre la modification d'une Bitmap elle doit être déclarée mutable

  $\Longrightarrow$ copie de la Bitmap dans une autre qui peut être modifiée

Traitement d'images

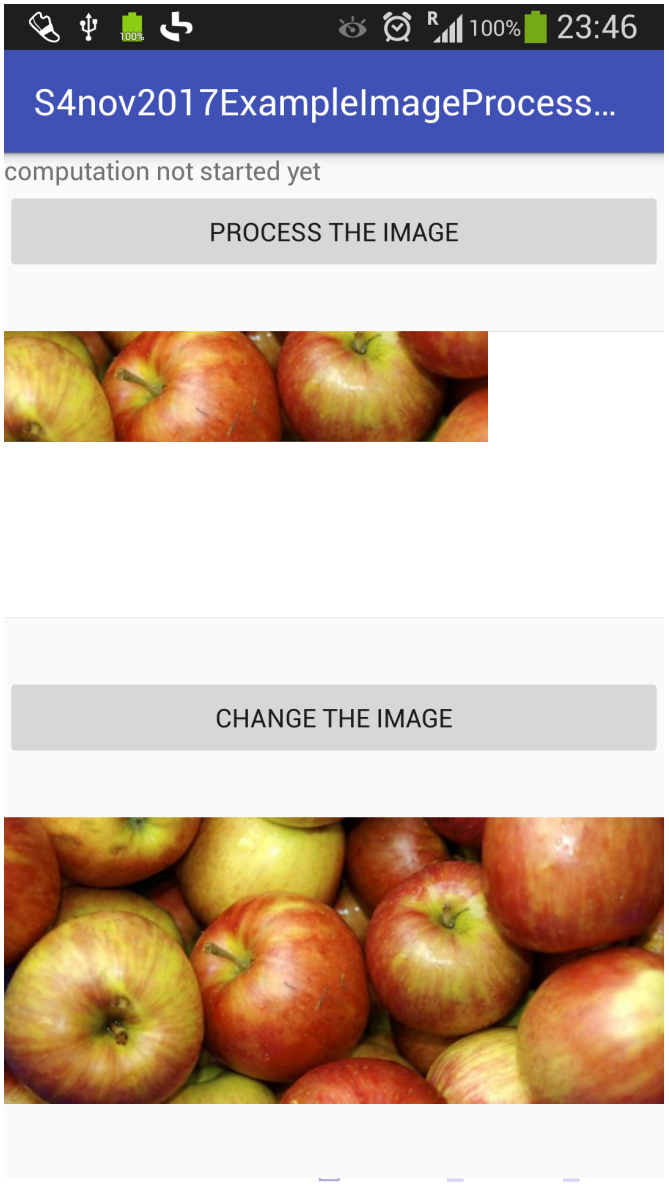
Modifier une image

Pixels

- obtenir les pixels à partir d'une bitmap : `Bitmap.getPixels(pixels[],int offset, int stride, int x, int y, int width, int height`
  - pixels est un array of int (les Colors des pixels) qui doit être de la taille bitmap.width*bitmap.height (la taille de l'image complète)
  - offset est le premier index pour écrire dans le tableau (par défaut à 0)
  - stride: nombre de pixels entre les lignes (classiquement la valeur de la largeur du bitmap)
  - x et y: position du premier pixel à considérer (souvent 0 et 0)
  - width et height: le nombre de pixels à lire pour chaque ligne et le nombre de lignes (souvent la largeur et la hauteur du bitmap)

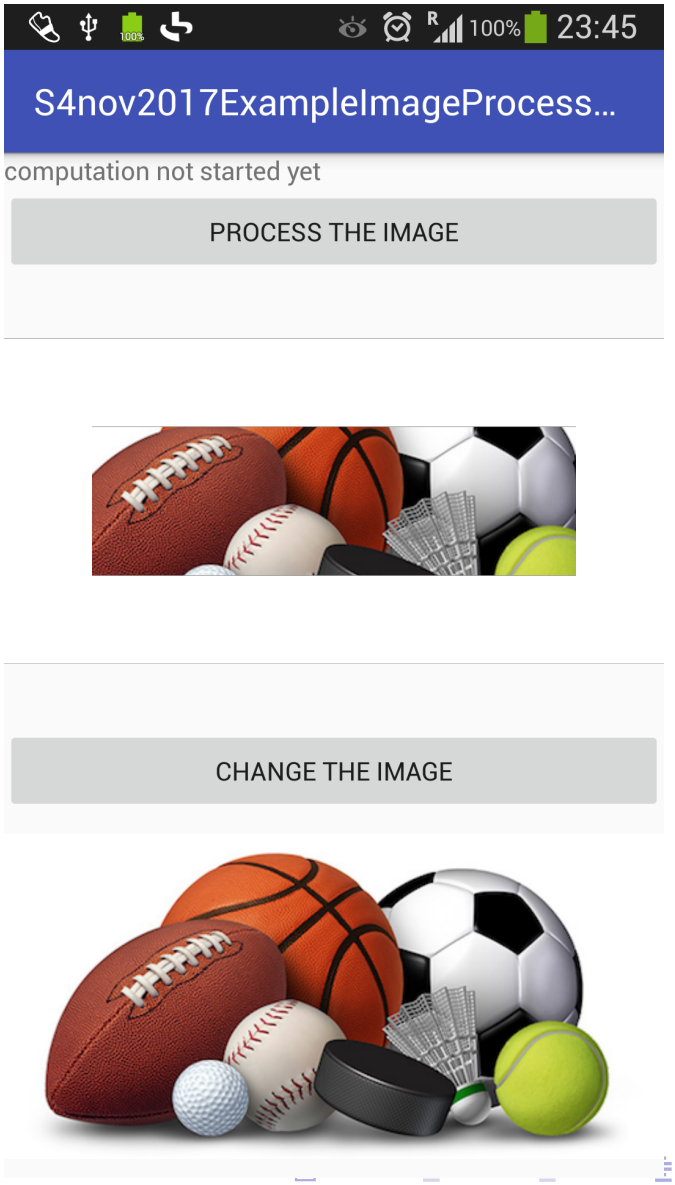
Traitement d'images : getPixels

```
getPixels(pixels,
          0,
          bitmap.width,
          200,
          200,
          bitmap.width-400,
          bitmap.height-400
```



Traitement d'images : getPixels

```
getPixels(pixels,
          200*bitmap.width+200,
          bitmap.width,
          200,
          200,
          bitmap.width-400,
          bitmap.height-400
```



Retour au traitement d'mages

du Bitmap au Pixels[] au Bitmap

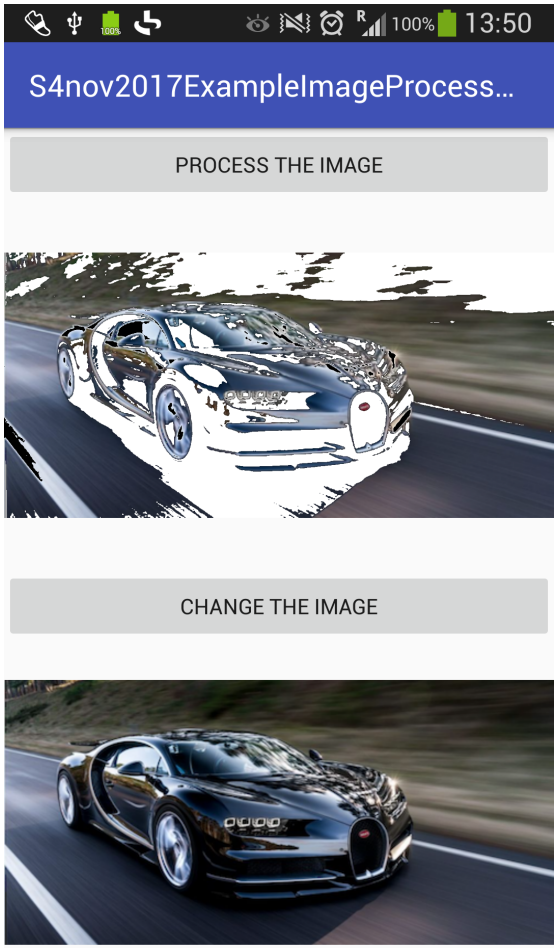
```
public void processImage(View v) {
    int id = 1+alea.nextInt( bound: 10);
    int imageID = getResources().getIdentifier( name: "image_"+id,
            defType: "drawable", getPackageName());
    Bitmap myImageOriginal = BitmapFactory.decodeResource(getResources(),imageID);
    Bitmap myImage = myImageOriginal.copy(myImageOriginal.getConfig(), isMutable: true);
    ivTwo.setImageBitmap(myImage);
    int imgWidth = myImage.getWidth();
    int imgHeight = myImage.getHeight();
    int pixels[] = new int[imgWidth*imgHeight];
    myImage.getPixels(pixels, offset: 0,imgWidth, x: 0, y: 0,imgWidth,imgHeight);
    modifyImage(pixels,imgWidth,imgHeight);
    myImage.setPixels(pixels, offset: 0,imgWidth, x: 0, y: 0,imgWidth, imgHeight);
    ivTwo.setImageBitmap(myImage);
}
```

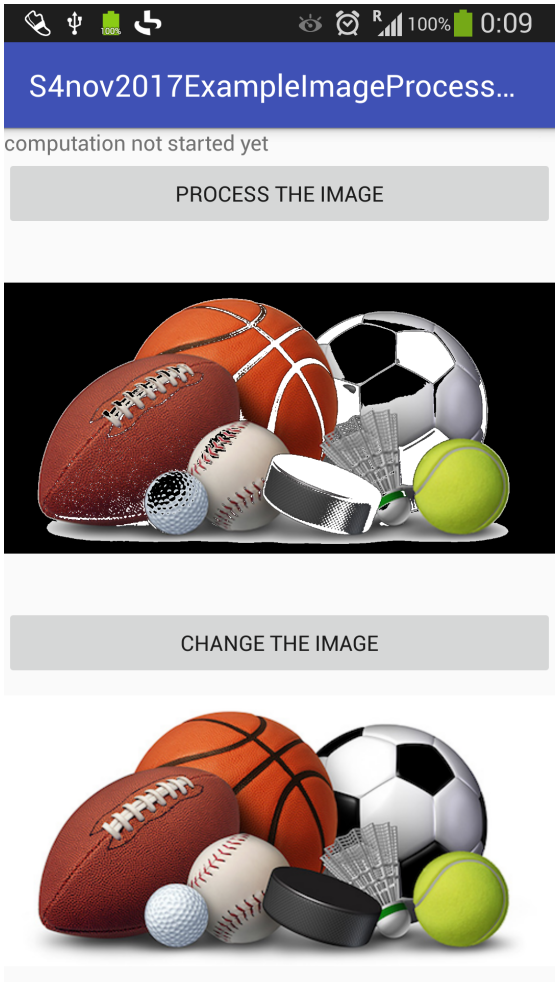
Retour au traitement d'mages

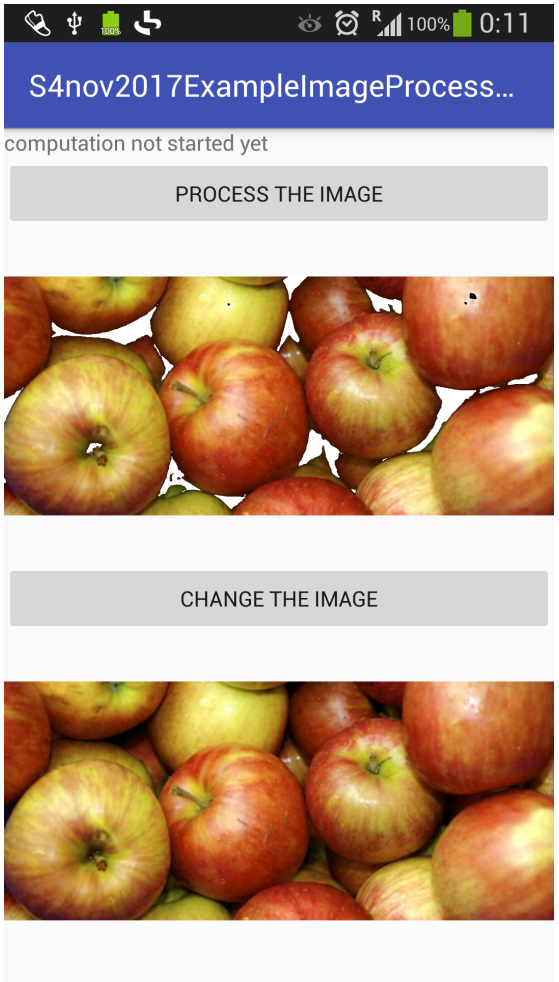
filtrage

```
public void modifyImage(int[] pixels, int imgWidth, int imgHeight) {
    int red = Color.red(pixels[0]);
    int green = Color.green(pixels[0]);
    int blue = Color.blue(pixels[0]);
    int whiteThreshold = 50;
    int blackThreshold = 220;
    for(int line=0;line<imgHeight;line++) {
        for(int col=0;col<imgWidth;col++) {
            int index = col+imgWidth*line;
            if((Color.red(pixels[index]) < whiteThreshold)
                    && ((Color.blue(pixels[index]) < whiteThreshold)
                    && (Color.green(pixels[index]) < whiteThreshold))) {
                pixels[index] = Color.WHITE;
            } else
                if((Color.red(pixels[index]) > blackThreshold)
                    && ((Color.blue(pixels[index]) > blackThreshold)
                    && (Color.green(pixels[index]) > blackThreshold))) {
                    pixels[index] = Color.BLACK;
                }
        }
    }
}
```

Résultats


13:50
S4nov2017ExampleImageProcess...
PROCESS THE IMAGE
CHANGE THE IMAGE


0:09
S4nov2017ExampleImageProcess...
computation not started yet
PROCESS THE IMAGE
CHANGE THE IMAGE


0:11
S4nov2017ExampleImageProcess...
computation not started yet
PROCESS THE IMAGE
CHANGE THE IMAGE

Références

- http://developer.android.com/training/articles/perf-anr.html
- http://developer.android.com/reference/android/os/AsyncTask.html
- http://developer.android.com/reference/java/lang/Runnable.html
- http://developer.android.com/training/multiple-threads/index.html