BURKINA FASO

***

UNITÉ-PROGRÈS-JUSTICE

MASTER2: RÉSEAUX INFORMATIQUES ET MULTIMÉDIA

PROGRAMMATION MOBILE

Evènements et intentions

COMPAORE MOCTAR

4 May 2022

PLAN

- ✓ Gestion des événements
- ✓ Débogage : LogCat
- ✓ Toasts
- ✓ Pile des activités
- ✓ Passage d’activités
- ✓ Transfert des données
- ✓ Détection et résolution des exceptions

GESTION DES ÉVÉNEMENTS

- ✓ Sous Android, il existe plusieurs façons d'interagir avec une interface graphique.
- ✓ Ces interactions s'appellent des évènements,
- ✓ par exemple :
  - ✓ cliquer sur un bouton,
  - ✓ entrer un texte,
  - ✓ sélectionner un texte, etc.
- ✓ Pour réagir au déclenchement d'un évènement, il faut utiliser un écouteur (Listener) qui va détecter et traiter l'évènement.

GESTION DES ÉVÉNEMENTS

- ✓ Un listener est une interface Java à implémenter en redéfinissant des méthodes callback qui sont appelées quand l'évènement associé est déclenché

- ✓ Pour intercepter l'évènement clic sur un Button, et ainsi déclencher un comportement, Android offre 2 méthodes différentes :

GESTION DES ÉVÉNEMENTS

- ✓ Méthode 1: En utilisant un listener

Cette méthode consiste à implémenter l'interface View.OnClickListener, qui contient la méthode callback onClick(View v) où v est la vue sur laquelle le clic a été effectué. La méthode onClick sera appelée à chaque clic sur le bouton.

Pour associer le listener du clic à un bouton, il faut utiliser une méthode du type setOnClickListener(OnClickListener l). Après avoir récupéré la référence du bouton en utilisant la méthode findById(…).

GESTION DES ÉVÉNEMENTS

- ✓ Méthode 1:En utilisant un listener

```java
public class ComplexActivity extends AppCompatActivityy {
        @Override
        public void onCreate(Bundle savedInstanceState) {
                ...
                Button b = (Button) findViewById(R.id.btn);
                        b.setOnClickListener(new View.OnClickListener() {
                         @Override
                         public void onClick(View v){
                                Log.i("tag", "bouton cliqué!!!");
                         }
                });
        }
}
```

GESTION DES ÉVÉNEMENTS

- ✓ Méthode 1: En utilisant un listener

Dans le cas où le comportement lors d'un clic est utilisé une seul fois, il est préférable de passer par une classe anonyme qui implémente le listener pour chaque évènement, et éviter ainsi d'augmenter le nombre de classes de l'application.

L'exemple précédent, une classe anonyme est créée pour implémenter l'interface View.OnClickListener.

GESTION DES ÉVÉNEMENTS

- ✓ Méthode 2 : au niveau du layout

À partir de la version 1.6, Android propose l'attribut XML android:onClick dans toutes les vues, afin d'accélérer le développement.

En effet, il suffit d'implémenter dans l'activité, la méthode à déclencher lors du clic, par exemple fonction(View v), puis la déclarer dans le layout associé. Cette méthode est conseillée lorsque le clic sur plusieurs boutons provoque le même comportement.

GESTION DES ÉVÉNEMENTS

- ✓ Méthode 2: au niveau du layout

```
public class ComplexActivity extends AppCompatActivity {
        @Override
        public void onCreate(Bundle savedInstanceState) {
       ...
        }
       public void fonction(View v) {
            Button b = (Button) v; // on force le view à devenir un bouton
            Log.i(''tag'', ''Bouton cliqué ''.concat(b.getText().toString()'');

       }
}
java
```

GESTION DES ÉVÉNEMENTS

- ✓ Méthode 2: au niveau du layout

```xml
...
<Button
        android:onClick="fonction"
        android:id="@+id/btn" />
...
```

xml

GESTION DES ÉVÉNEMENTS

- ✓ Autres Evènements 2:

Android permet de gérer d'autres évènements pour une vue donnée, en proposant d'autres listeners. La liste suivante énumère les principaux listeners disponibles dans View

GESTION DES ÉVÉNEMENTS

- ✓ Autres Evènements 2:

La liste suivante énumère les principaux listeners disponibles dans View:

```
OnClickListener                    // clic
OnLongClickListener                // clic long
OnDragListener                     // Glissement
OnTouchListener                    // Touché
OnHoverListener                    // Passage
OnKeyListener                      // Frappe de clavier
OnAttachStateChangeListener        // Changement de l'état d'attachement
OnLayoutChangeListener             // Changement du layout
OnCreateContextMenuListener        // Création du menu contextuel
OnFocusChangeListener              // Changement du focus
OnGenericMotionListener            // Un mouvement (mouse, finger, ...)
OnSystemUiVisibilityChangeListener // Changement de la visibilité de
                                      la barre d'état
```

GESTION DES ÉVÉNEMENTS

- ✓ Par exemple:

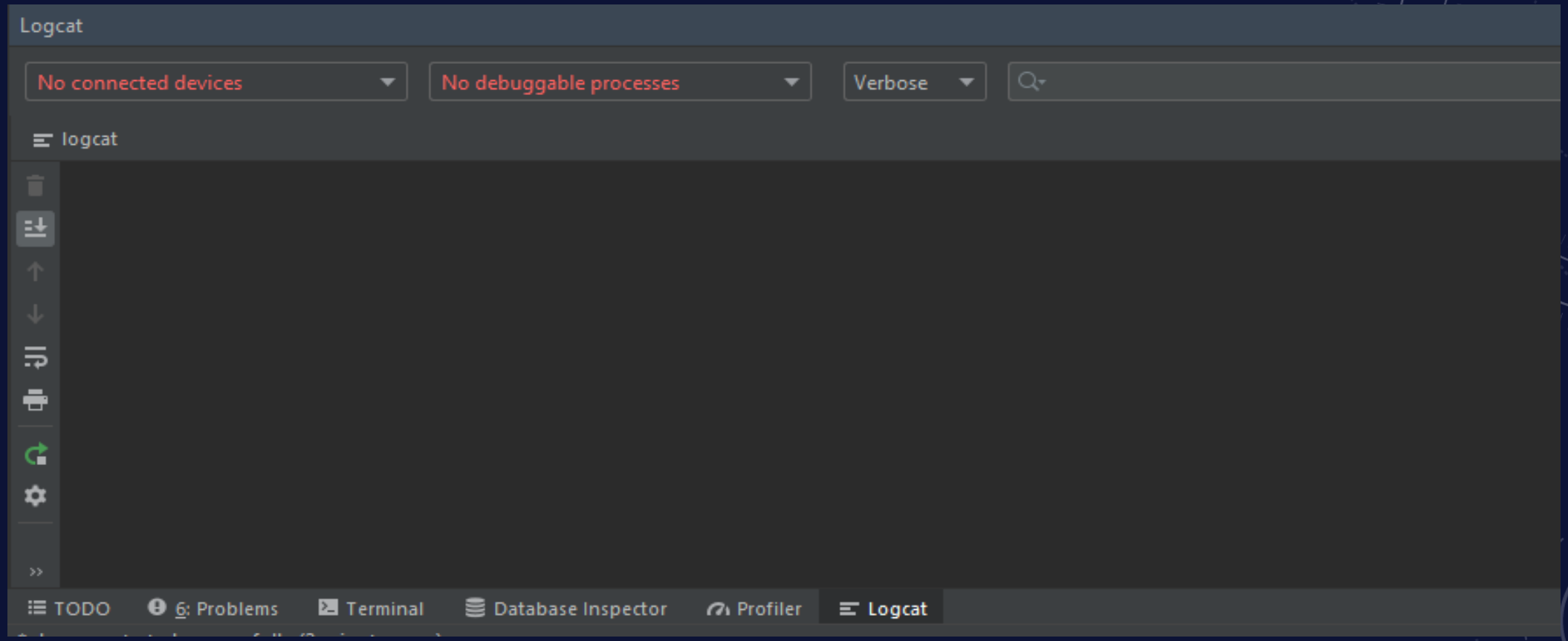
View.OnLongClickListener est utilisé pour les clics qui durent longtemps, avec l'implémentation de la méthode onLongClick(View v).

Par ailleurs, View.OnKeyListener sert à gérer l'appui sur une touche du clavier, en implémentant la méthode onKey(View v, int code, KeyEvent e).

DÉBOGAGE : LOGCAT

- ✓ La méthode de débogage probablement la plus triviale, est d'utiliser des primitives d'affichage sur la sortie standard, System.out.println(...).
- ✓ Les messages de journalisation Android s’appellent des logs, pour les visualiser il faut utiliser l’outil "LogCat".
- ✓ Cet outil est disponible dans la fenêtre "Android Monitor"

DÉBOGAGE : LOGCAT



- Si "Android Monitor" n’est pas visible dans Android Studio, il suffit de cliquer sur View → Tool Windows → Android Monitor.
- Il faut préciser l’appareil et l’application pour que les logs s’affichent

DÉBOGAGE : LOGCAT

Niveaux de verbosité

- Verbose, debug, info, erreur, avertissement

Méthodes statiques de la classe Log

- Log.v(…), Log.d(…), Log.i(…), Log.e(…), Log.w(…)

Paramètres : "tag", "message"

- Exemples

```
Log.d("MainActivity", "onCreate: Création de l’activité.");
Log.e("MainActivity", "onClick: Une erreur!!!")
```

MESSAGES D'INFORMATION (TOASTS)

- Comme tous système, Android permet d'afficher des messages d'alerte ou d'information à l'utilisateur.
- Un Toast est un message d'information, dit transitoire, car il ne nécessite aucune intervention de la part de l'utilisateur et il disparaît après quelques instants.
- Pour créer un Toast, il faut utiliser la méthode statique makeText(…):

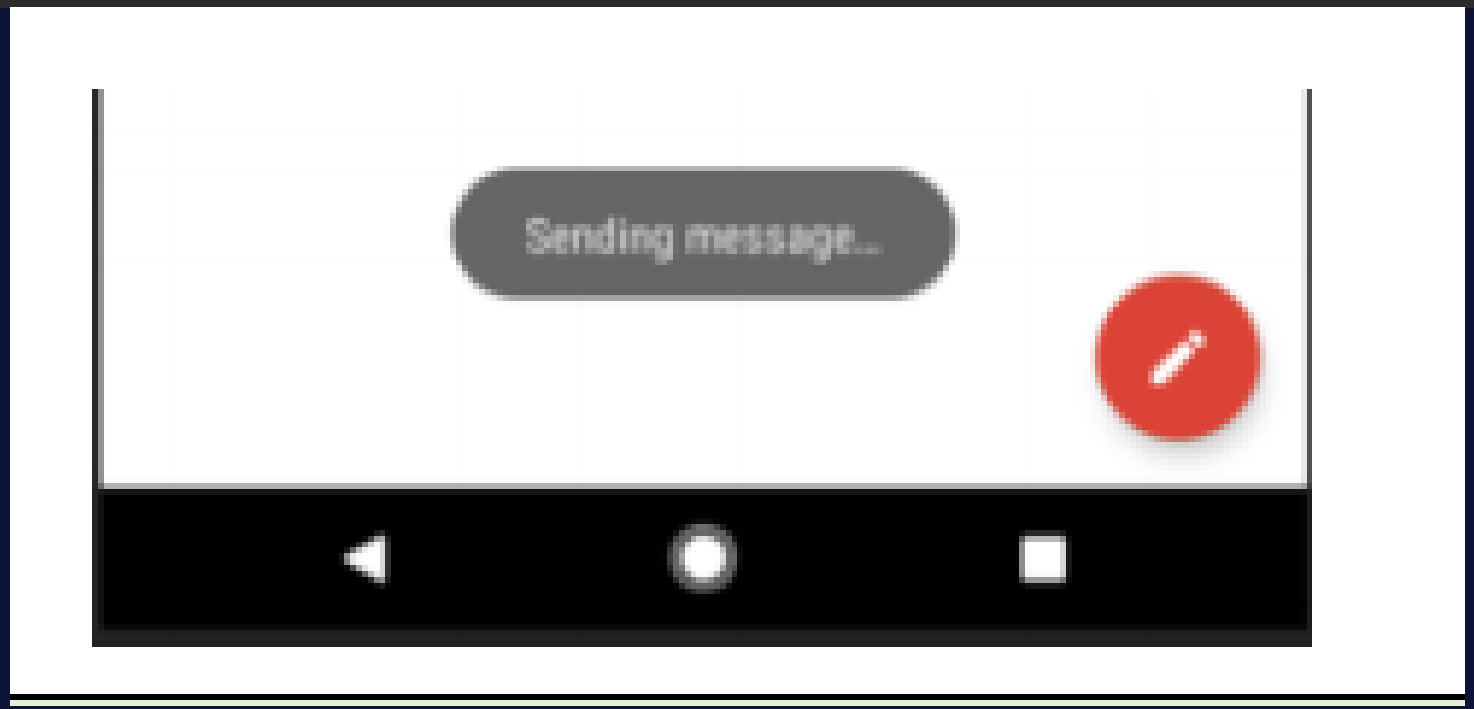
MESSAGES D'INFORMATION (TOASTS)

public static Toast makeText(Context context, CharSequence text, int duration)

- En effet, chaque Toast possède 3 paramètres : le contexte de l'application, le texte à afficher et la durée
- d'affichage, pouvant prendre 2 valeurs : Toast.LENGTH_SHORT ou Toast.LENGTH_LONG.
- Ensuite, pour afficher le toast, il faut utiliser la méthode show().

MESSAGES D'INFORMATION (TOASTS)

```
Toast t = Toast.makeText( this, "message...", Toast.LENGTH_SHORT)
;
t.show();
```



MESSAGES D'INFORMATION (TOASTS)

Pour récupérer le contexte d'une application Android, il existe 4 méthodes

- getApplicationContext():
- getContext() (méthode de la classe View)
- getBaseContext()
- this (disponible uniquement dans une activité)

TRAVAUX PRATIQUES

PRATIQUES

TRAVAUX PRATIQUES

Dans votre projet LoginActivity

Lors du clic sur le Button

1. Vérifier si : L'identifiant = "IST" et

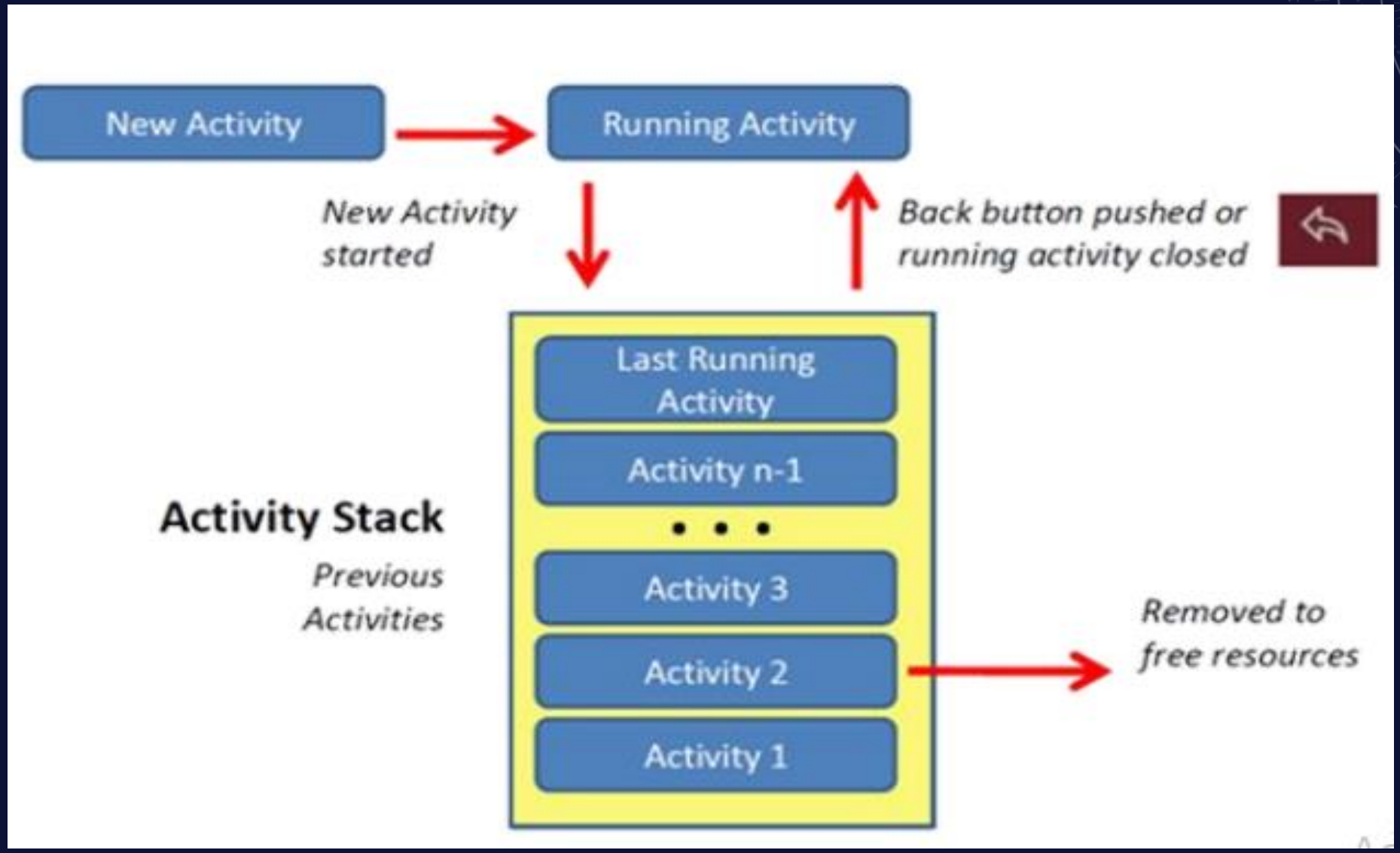
Le mot de passe = " Wayalghin "

2. Afficher un Log et un Toast en fonction du résultat

PILE DES ACTIVITÉS

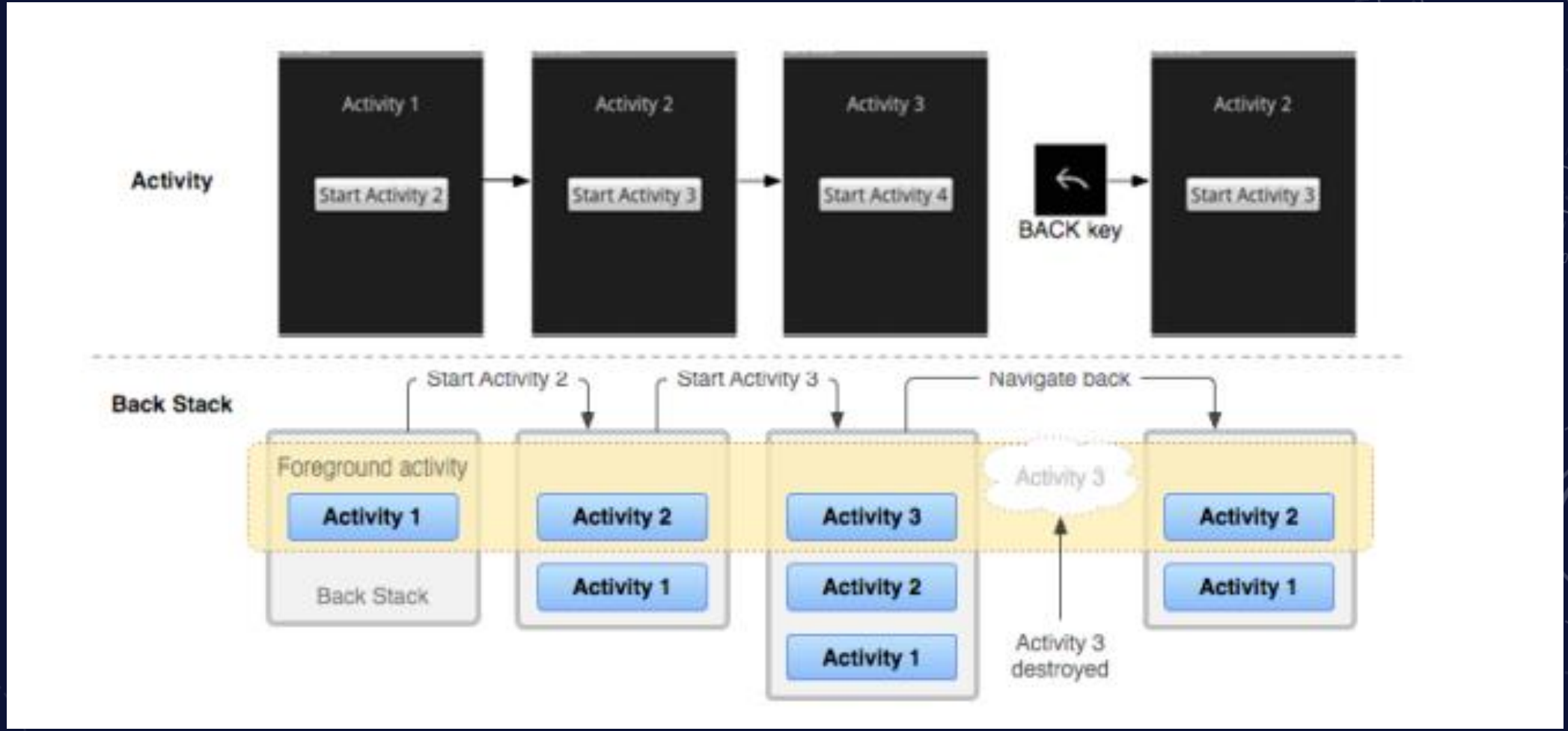
- ✓ Android gère les activités dans une pile. Chaque fois où une nouvelle activité est lancée, elle arrive à la première position de la pile et devient l'activité qui est actuellement exécutée.
- ✓ L'activité précédente reste dans la pile et ne pourra être relancée qu’une fois la première activité terminée, ou si l’utilisateur appuie sur le bouton (Back). Une pression sur le bouton (Home) ne dépile pas l'activité, mais la met en pause et réduit l’application en arrière plan.

PILE DES ACTIVITÉS



PILE DES ACTIVITÉS

Les activités sont empilées/dépilées



INTENTIONS

- Les applications Android doivent se décomposer en activités distinctes.
- les activités doivent s'enchaîner entre elles.
- Pour mettre en place cet enchainement, Android utilise des intentions, en permettant l'envoi et la réception des messages/données entre les activités afin de les faire coopérer

INTENTIONS

- Une intention (android.content.Intent) est considérée comme de la colle entre des activités permettant de les lier les unes aux autres.
- Elle permet de déléguer une action à un composant, une application ou une activité de l'application courante.

Une intention peut être utilisée pour :

INTENTIONS

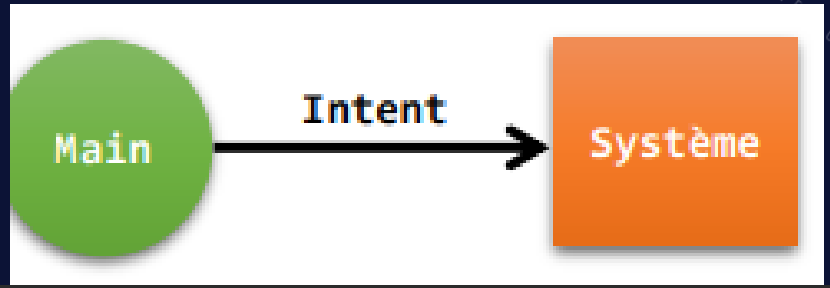
Une intention peut être utilisée pour :

- Démarrer une activité en utilisant startActivity(Intent),
- Communiquer avec un service en appelant startService(Intent),
- Envoyer des données à des récepteurs de diffusion avec sendBroadcast(Intent)

INTENTIONS

Types d'intentions

- Une Intent implicite précise l'action à déclencher avec les données nécessaires à cette action. Par exemple, déclencher un appel téléphonique

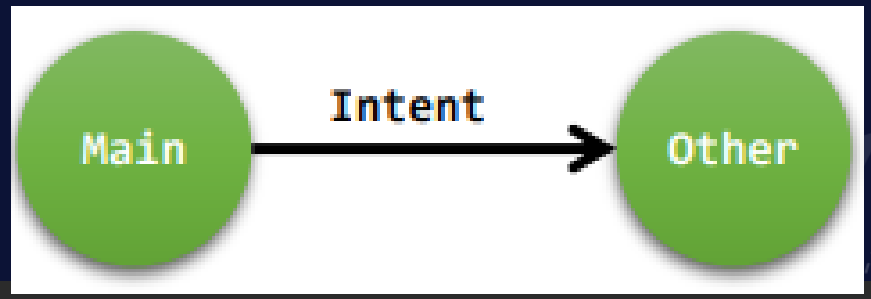



```java
...
Intent intent = new Intent( Intent.ACTION_CALL,
Uri.parse("tel:00226xxxxxx") );
startActivity(intent);
...
```

INTENTIONS

Types d'intentions

- Une Intention explicite définit explicitement l'activité qui doit être appelée en utilisant le nom de classe Java comme identifiant. Typiquement, les activités doivent être de la même application.
- Exemple

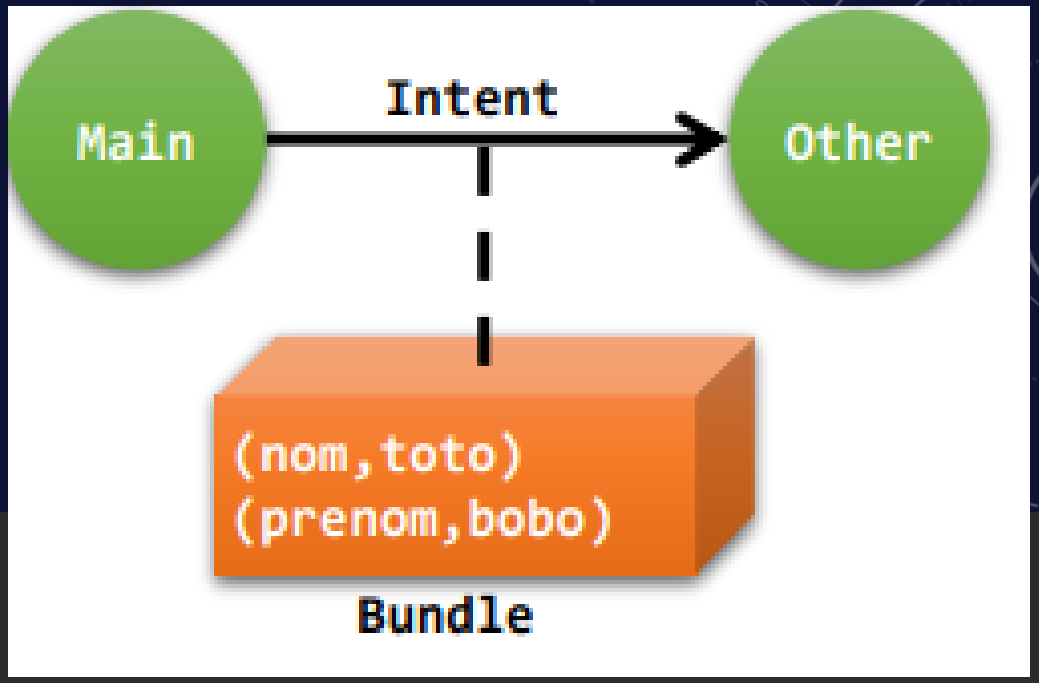



```java
...
Intent intent = new Intent(this, OtherActivity.class);
startActivity(intent);
...
```

TRANSFERT DES DONNÉES

Une intention peut contenir des données supplémentaires stockées dans un Bundle d'informations permettant de transférer ces données à l'activité de destination,

TRANSFERT DES DONNÉES



Transfert des données primitives

```java
...
..
Intent intent = new Intent(this, OtherActivity.class);
Bundle bundle = new Bundle() ; // on instancie le bundle
bundle.putString("nom", "toto") ; // on met les elements
bundle.putString("prenom", "bobo") ;
intent.putExtras(bundle); // on rattache le bundle à l'intent
startActivity(intent); // on lance la nouvelle activité
...
```

java

TRANSFERT DES DONNÉES

Transfert des données primitives

Dans l'activité de destination, l'objet intent peut-être récupéré par la méthode getIntent() les données peuvent aussi être récupérées par la méthode getExtrats(),

```java
...
Intent intent = getIntent();
Bundle bundle = intent.getExtras();
String nom = bundle.getString("nom");
String prenom = bundle.getString("prenom");
...
```

TRANSFERT DES DONNÉES

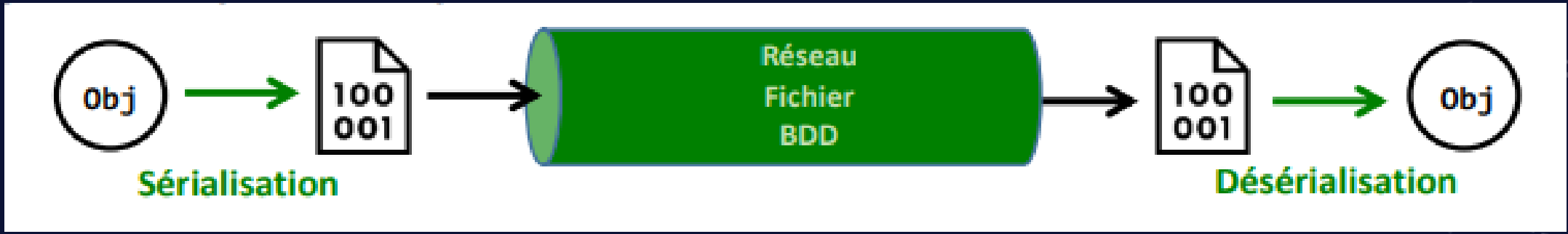
Transfert des données complexes

- Lorsque l'on développe sous Android, il est parfois nécessaire d’envoyer un objet complexe d'une activité à une autre.
- Ce processus qui consiste à convertir un objet en un tableau d'octets pour pouvoir être persisté (ou transféré) s'appelle la sérialisation.
- En effet, La sérialisation (Marshaling en anglais) permet de rendre un objet persistant pour un stockage ou un échange,

TRANSFERT DES DONNÉES

Transfert des données complexes

- À la réception, l'opération inverse qui consiste à reconstituer l’objet à l'identique (sa reconversion le tableau d’octets vers sa représentation initiale) s’appelle la désérialisation (Unmarshaling en anglais)



TRANSFERT DES DONNÉES

Transfert des données complexes

Pour qu'un objet Java soit sérialisable, il faut que sa classe implémente l'inteface java.io.Serialisable .

Ce qui donne un résultat semblable au code suivant :

```
...
public class Module implements Serializable {
                                ...
}
...
java
```

TRANSFERT DES DONNÉES

Transfert des données complexes

- Pour envoyer l'objet complexe au niveau d'une activité appelante, il suffit de le passer comme paramètre dans la méthode putSerialisable(…) d'un Bundle, avec la clé correspondante. Ensuite, il faut associer le bundle à l'intent de passage avec de la lancer.

```java
...
Module m = new Module(...);
Intent intent = new Intent(this, OtherActivity.class);
Bundle bundle = new Bundle() ;
bundle.putSerializable("obj", m);
intent.putExtras(bundle);
startActivity(intent);
...
```

TRANSFERT DES DONNÉES

Transfert des données complexes

- Dans l'activité appelée, on récupère l'objet grâce à la méthode getSerialisable(…), puis on le reconstruit avec un sub-casting

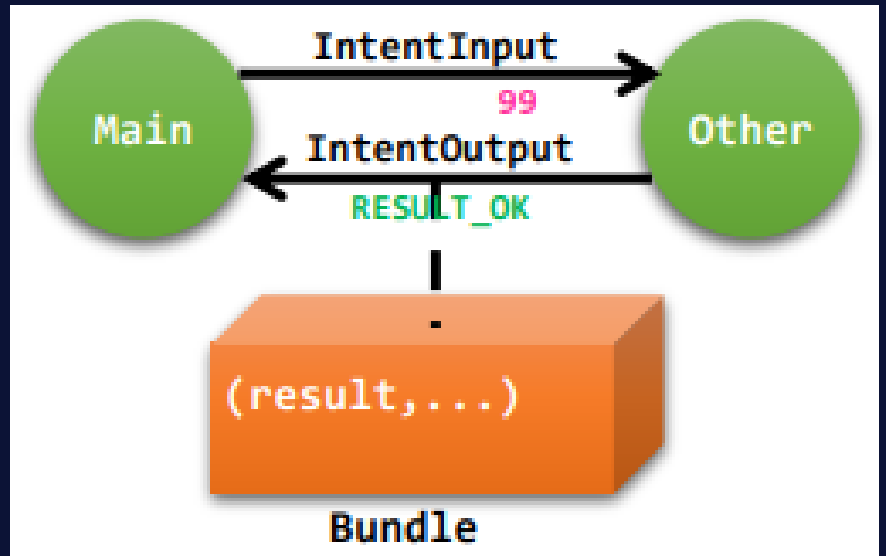
```java
...
Intent intent = getIntent();
Bundle bundle = intent.getExtras();
Module m = (Module) bundle.getSerializable("obj");
...
```

PRATIQUES

RETOUR D'UNE ACTIVITÉ

La méthode startActivity(…) permet seulement de lancer une activité en lui transférant des données.

Cependant, pour lancer une activité et récupérer un résultat en retour, il faut utiliser la méthode startActivityForResult(…), au lieu de startActivity(…).



RETOUR D'UNE ACTIVITÉ

L'activité appelante qui recevra le résultat, devra implémenter la méthode onActivityResult(…)

```java
...
Intent intentInput = new Intent(this, OtherActivity.class);
startActivityForResult(intentInput, 99) ;
...
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intentOutput){
        if(requestCode == 99 && resultCode == RESULT_OK)){
                intentOutput.getExtra("result");
                ...
        }
}
...
```

java

RETOUR D'UNE ACTIVITÉ

Deux principaux codes de retour de l'activité appelée sont :

- RESULT_OK : L'activité s'est bien déroulée,
- RESULT_CANCELED : L'activité est abandonnée.

```java
Intent intentInput = getIntent();
Bundle bundle = intentInput.getExtras();
...
finish();
...
@Override
public void finish(){
        Intent intentOutput = new Intent();
        intentOutput.putExtra("result", "...");
        setResult( RESULT_OK, intentOutput);
        super.finish();
}
...
```

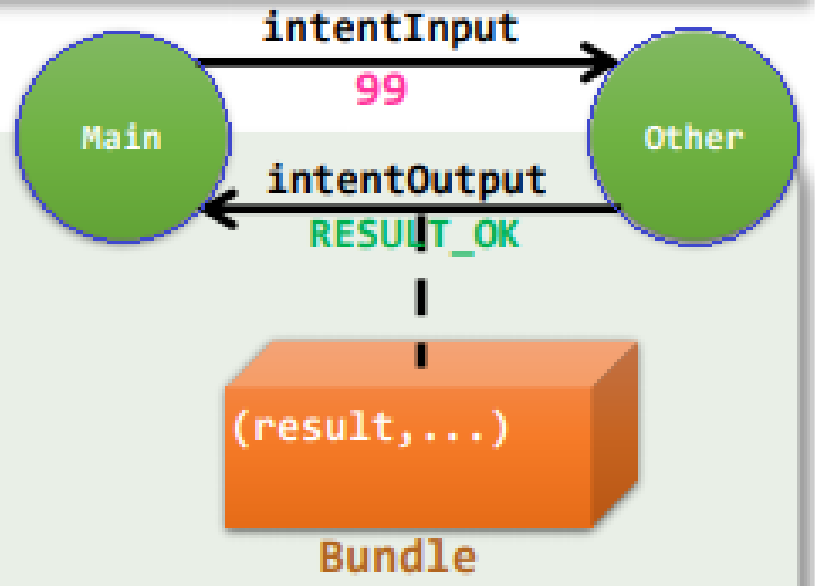
RETOUR D'UNE ACTIVITÉ

/java/MainActivity.java

```
1  Intent intentInput = new Intent(this, OtherActivity.class);
   startActivityForResult(intentInput, 99);
   ...
4  protected void onActivityResult(int requestCode, int resultCode, Intent intentOutput){
       if(requestCode == 99 && resultCode == RESULT_OK)){ ... }
   }
```

/java/OtherActivity.java

```
2  Intent intentInput = getIntent();
   Bundle bundle = intentInput.getExtras();
   ...  ...  ...  finish();
   @Override
   public void finish(){
       Intent intentOutput = new Intent();
3      intentOutput.putExtra("result", "...");
       setResult(RESULT_OK, intentOutput);
       super.finish();
   }
```



RETOUR D'UNE ACTIVITÉ

Il est possible de retourner d'autres données que l'activité appelante devra les récupérer.

Il faut donc utiliser dans l'activité appelée, la méthode setResult(…), qui est souvant implémentée dans la méthode finish().

Pour retourner à l'activité appelante, il suffit de terminer l'activité appelée avec finish().

C'est cette action qui permettra de déclencher la méthode onActivityResult(…) de l'activité appelante qui pourra récupérer alors les données retournées par l'activité appelée.

DÉTECTION ET RÉSOLUTION DES EXCEPTIONS

Le LogCat permet tracer l'exécution de l'application pas à pas. Donc, il est possible d'intercepter toute exception pouvant être déclenchée lors de l'éxecution.

Pour ce faire, il faut suivre les étapes suivantes :

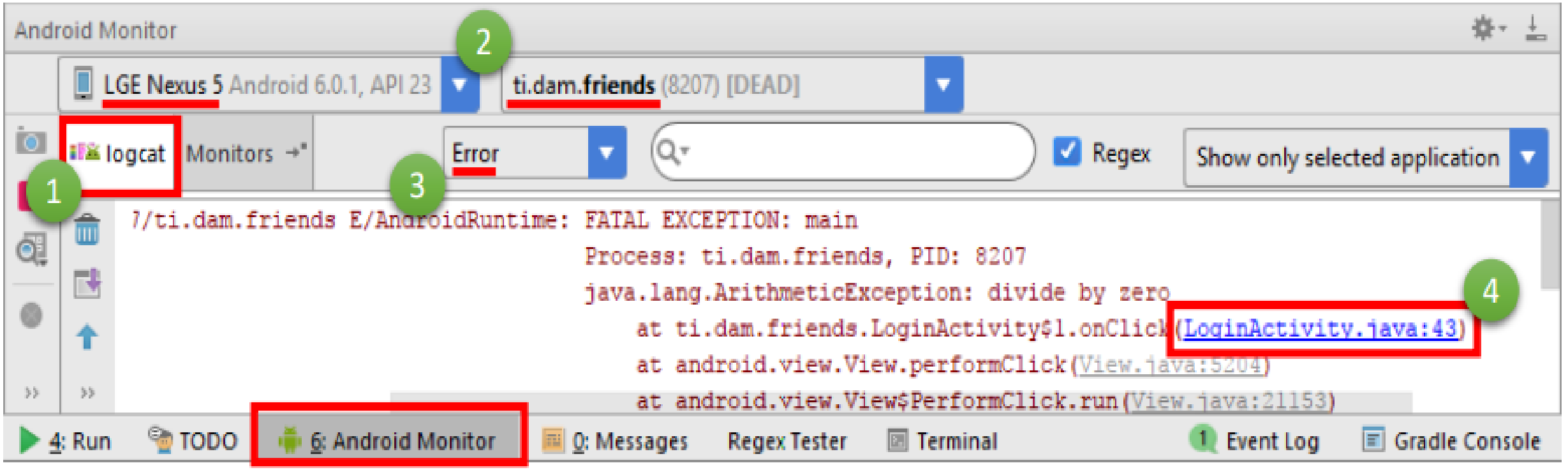
1) Aller à : Android monitor → LogCat,

2) S'assurer que l'appareil et l'application sont bien sélectionnés,

3) Filtrer les Logs de type "Error" en le sélectionnant,

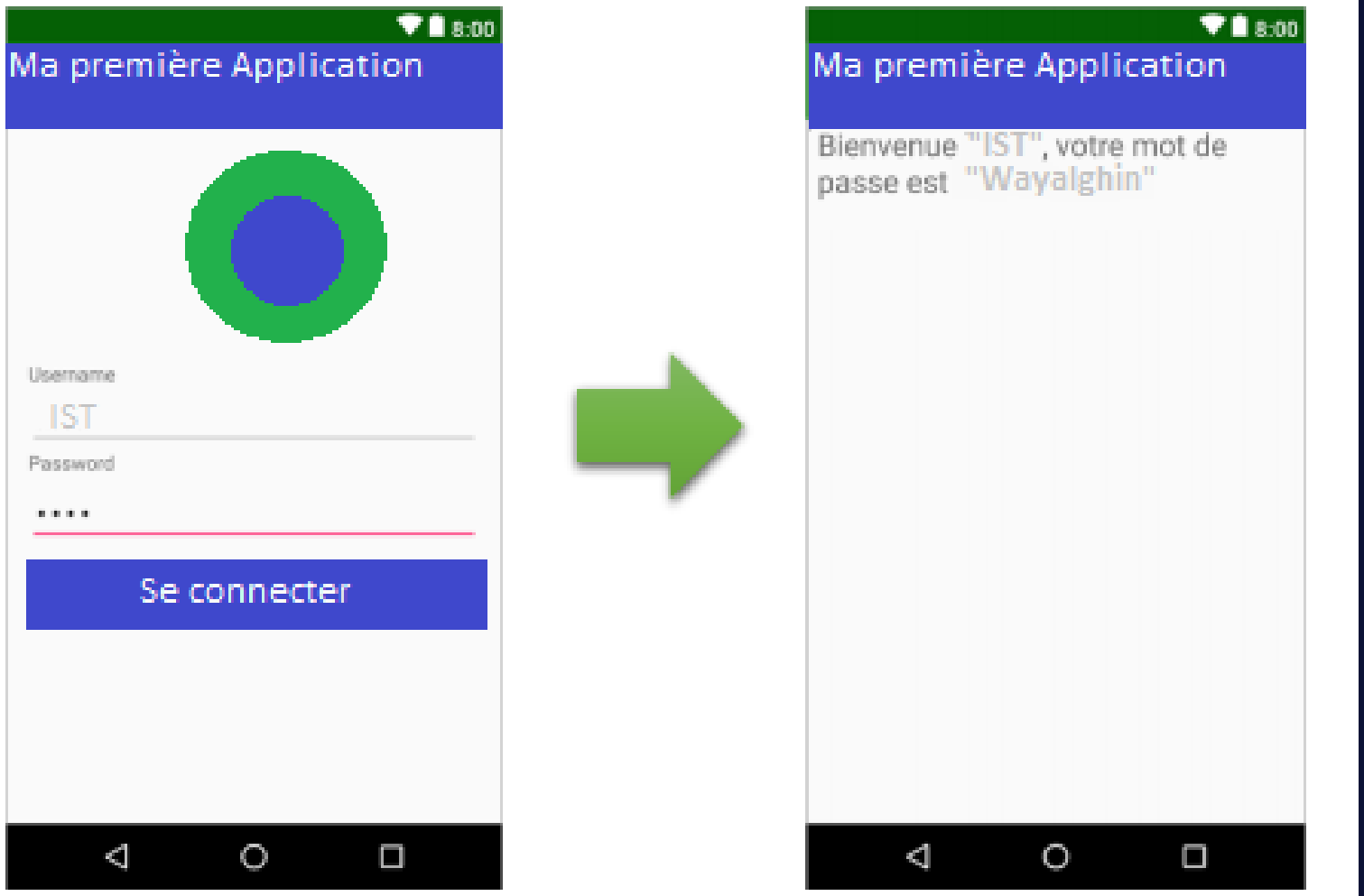
4) Localiser l'exception et corriger le code en fonction.

DÉTECTION ET RÉSOLUTION DES EXCEPTIONS



TRAVAUX PRATIQUES TP4

Dans le projet LogoActivity
Faire le passage vers ModulesActivity
Transférer l'identifiant et le mot de passe dans l'Intent
Afficher l'identifiant et le mot de passe dans un TextView



LIENS UTILES

Les étudiants peuvent consulter ces références pour approfondir leurs connaissances :

- ✓ Evènements et listeners graphiques :
  - ❖ http://developer.android.com/reference/android/view/View.html
- ✓ Messages de journalisation (Logs) :
  - ❖ https://cyrilmottier.com/2009/03/11/utilisation-des-logx/
- ✓ Toasts :
  - ❖ http://supertos.free.fr/supertos.php?page=1091
- ✓ Intentions :
  - ❖ http://vogella.developpez.com/tutoriels/android/utilisation-intents/
- ✓ Sérialisation sous Android :
  - ❖ http://www.supinfo.com/articles/single/1550-serialisation-objet-androi