



R4.Real.11 – Développement pour applications mobiles

- TP 2 -

Gestion des évènements et passage d'activités

Pr Chaouche A.-C.

ac.chaouche@gmail.com

Résumé

Prérequis

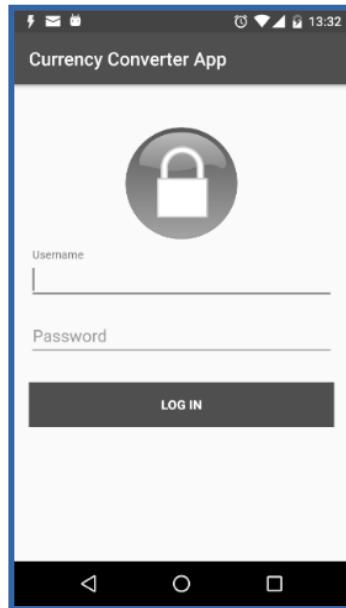
- Programmation Java
- Cycle de développement d'Android



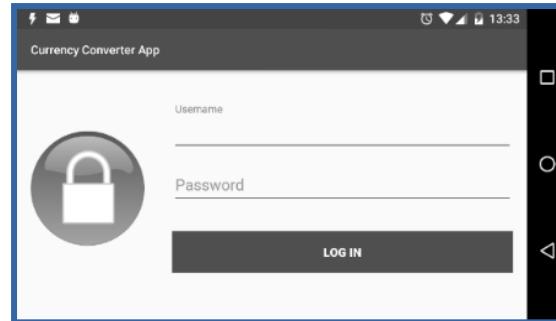
Objectifs

- Adapter l'activité à l'orientation paysage
- Internationaliser l'activité
- Gérer les évènements graphiques des vues
- Déboguer l'application avec des messages de journalisation

Vue portrait vs. vue paysage



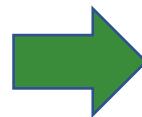
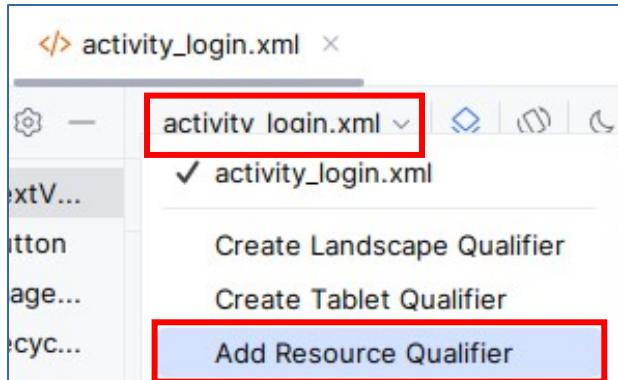
Portrait



Paysage

Prise en charge des différentes configurations

Il est possible de personnaliser les ressources, en fonction de :



- Country Code
- Network Code
- Locale
- Layout Direction
- Smallest Screen Width
- Screen Width
- Screen Height
- Size
- Ratio
- Roundness
- Orientation
- UI Mode
- Night Mode
- Density
- Touch Screen
- Keyboard
- Text Input
- Navigation State
- Navigation Method
- Dimension
- Version

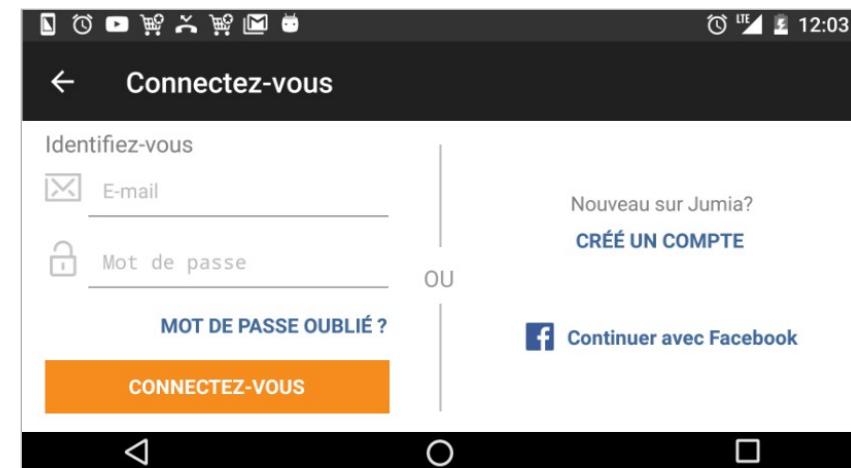
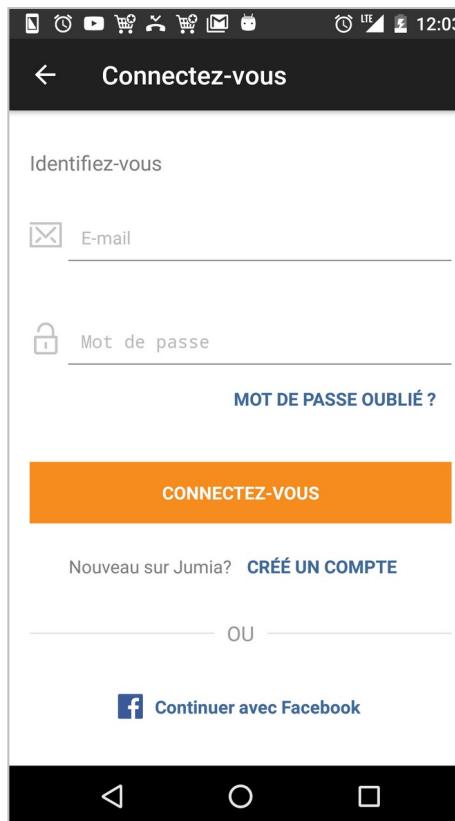
En utilisant des qualificateurs : par exemple :

- Par défaut : **layout** : layouts par défaut
- Orientation : **layout-land** : versions **paysage** des layouts
- Densité : **mipmap-xxhdpi** : icônes pour des écrans de densité **xxhdpi**
- Taille : **layout-large** : layouts pour des écrans de taille **large**
- Langue : **values-fr/strings.xml** : chaînes de caractères en **français**
- ...

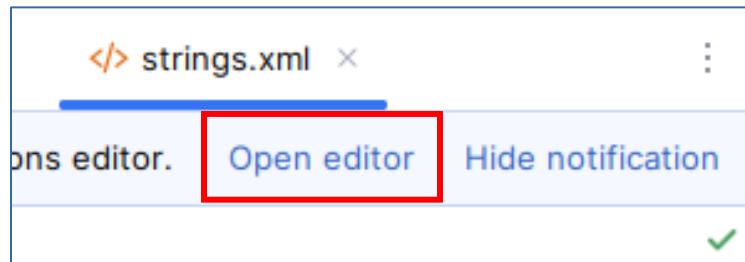
TP2a : Vue paysage de l'authentification

Adapter la vue de l'activité `LoginActivity`

- Proposer 2 vues (portrait et paysage)



Internationalisation (1/2)



/res/values/strings.xml

```
<resources>
    <string name="hello"> Hello </string>
    ...
</resources>
```

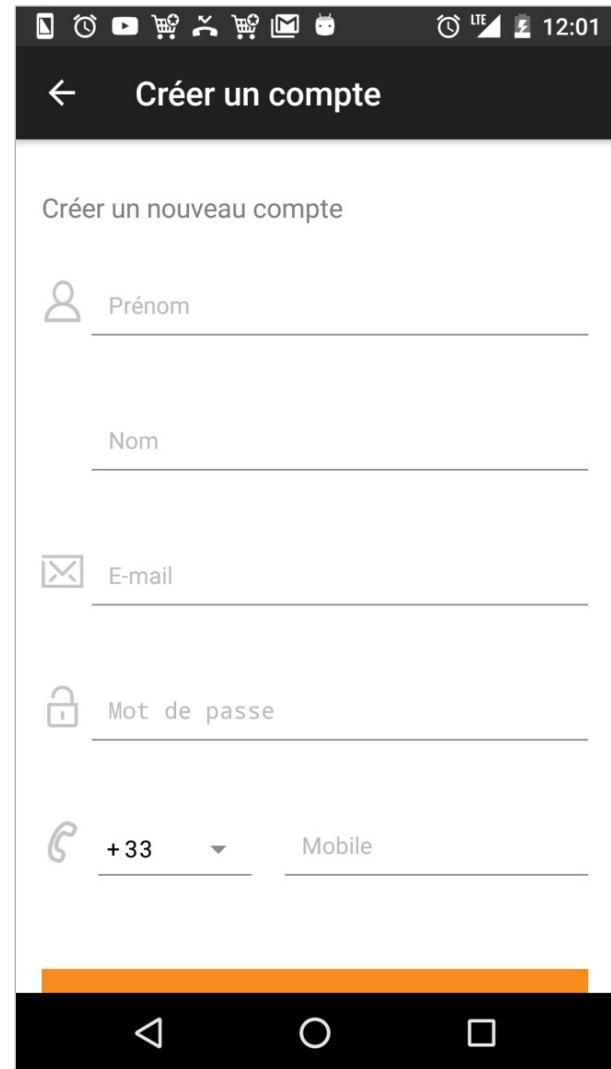
/res/values-fr/strings.xml

```
<resources>
    <string name="hello"> Salut </string>
    ...
</resources>
```

TP2b : Vue d'inscription

Créer une activité **RegisterActivity**

- Des **EditText** pour introduire les informations de l'utilisateur
- Un **Button** pour valider
- Internationalisation des champs
 - EN (par défaut)
 - FR
 - ES
 - ...



Identification d'une vue (1/2) – Layout

/res/layout/activity_login.xml (Text)

```
<?xml version="1.0" encoding="utf8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        android:id="@+id/tv_msg"/>
</LinearLayout>
```

- **@+id** permet d'ajouter un nouveau nom de ressource à la classe R

Identification d'une vue (2/2) – Activity

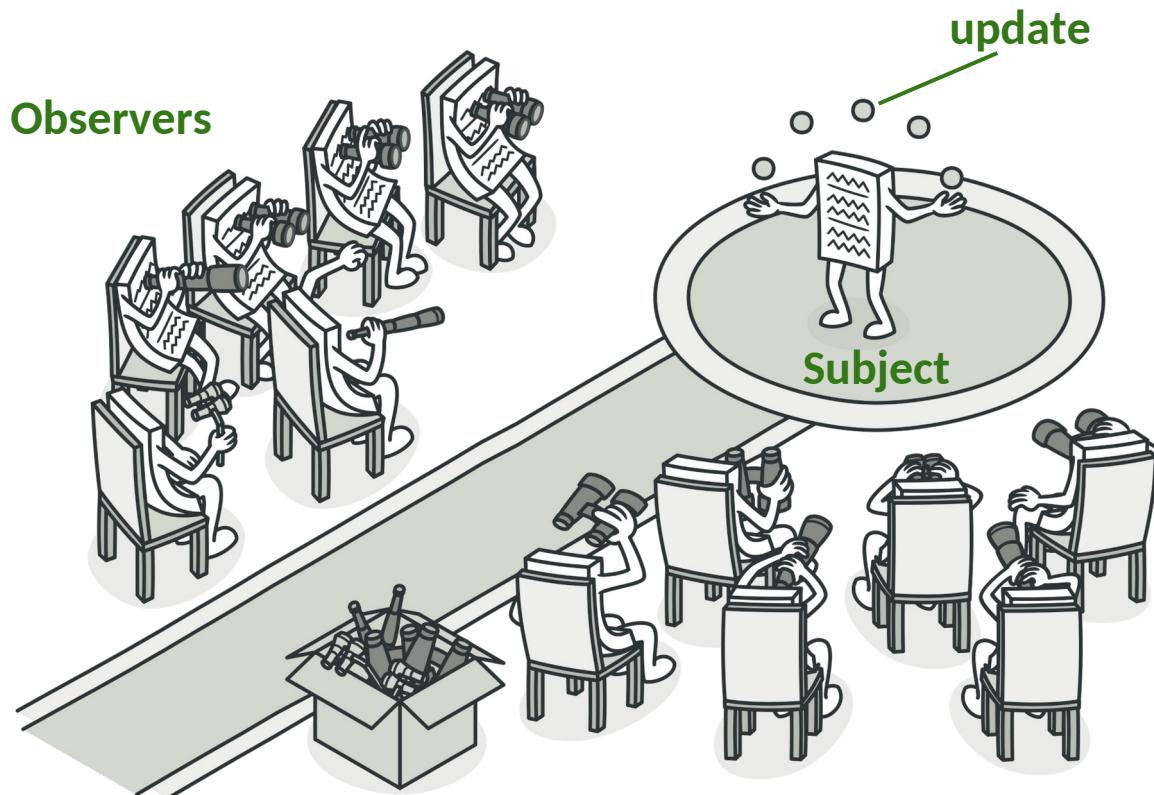
/java/LoginActivity.java

```
public class LoginActivity extends AppCompatActivity {  
    TextView msgTV;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_login);  
        ...  
        msgTV= findViewById(R.id.tv_msg);  
        msgTV.setText("Bienvenue Chaouche");  
    }  
}
```

- **findViewById(int id)** renvoie un objet **View** référencé par **id**

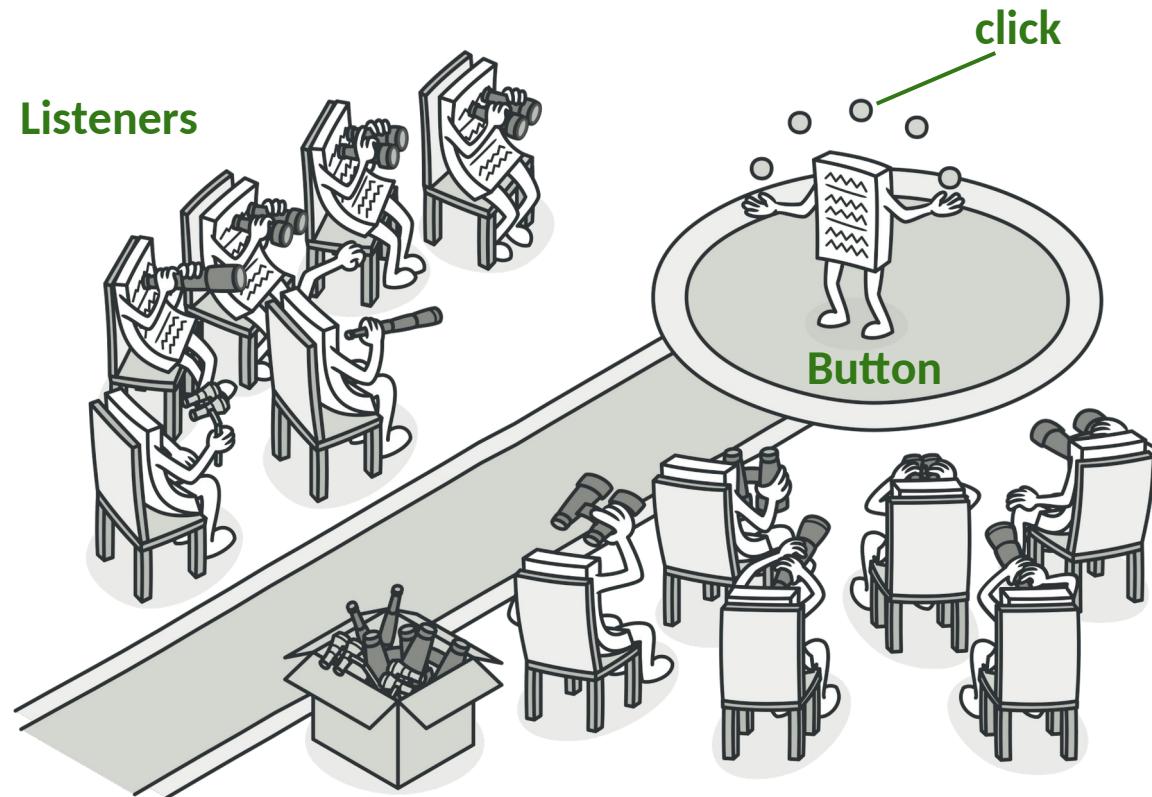
Gestion des événements (1/3) – Clic

Design pattern : Observer



Gestion des événements (2/3) – Clic

Design pattern : Observer



Gestion des événements (2/3) – Clic

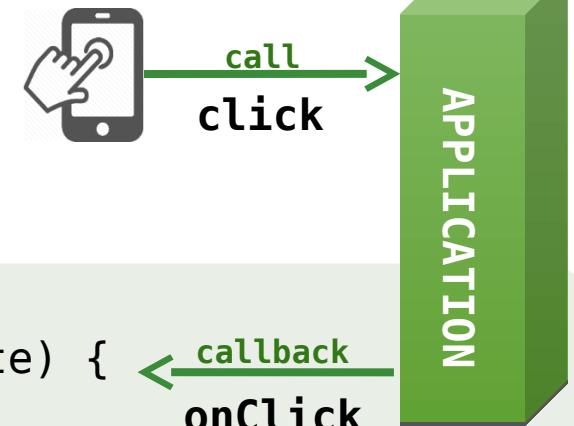
Méthode 1 : En utilisant un listener

Abonner l'activité à des évènements spécifiques

- Listener = Observer design pattern

/java/MainActivity.java

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
    Button b = findViewById(R.id.btn);  
    b.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v){  
            ... // bouton cliqué!!!  
        }  
    });  
}
```



Gestion des événements (3/3) – Clic

Méthode 2 : au niveau du layout

/res/layout/activity_main.xml

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

/java/MainActivity.java

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
}  
public void func(View v) {  
    ... // bouton cliqué!!!  
}
```

Listeners graphiques

android.View

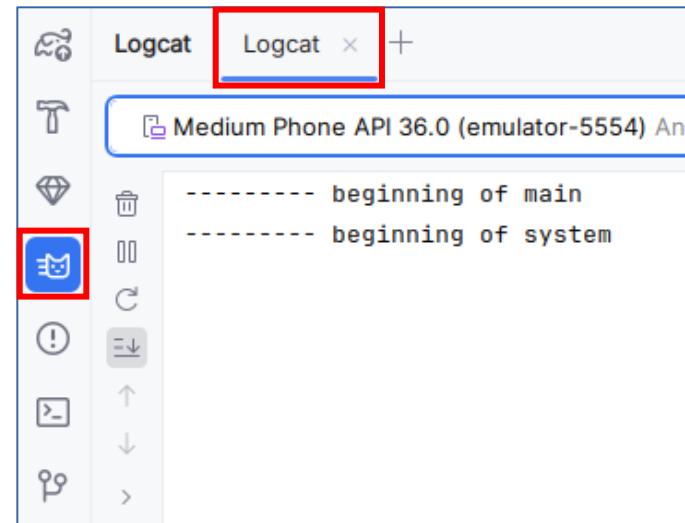
OnClickListener	// clic
OnLongClickListener	// clic long
OnDragListener	// glissement
OnTouchListener	// touché
OnHoverListener	// survol
OnKeyListener	// frappe de clavier
OnAttachStateChangeListener	// changement de l'état d'attachement
OnLayoutChangeListener	// changement du layout
OnCreateContextMenuListener	// création du menu contextuel
OnFocusChangeListener	// changement du focus
OnGenericMotionListener	// un mouvement (mouse, pen, finger, ...)
OnSystemUiVisibilityChangeListener	// changement de la visibilité de // la barre d'état

Débogage : LogCat

Affiche des messages dans le **Logcat** de façon structurée

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

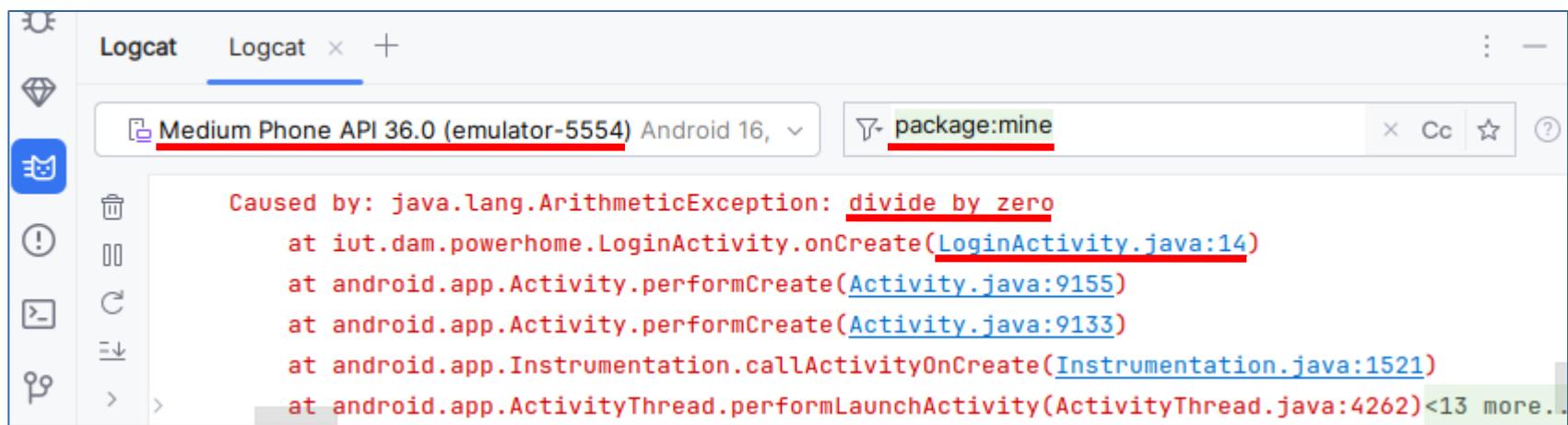
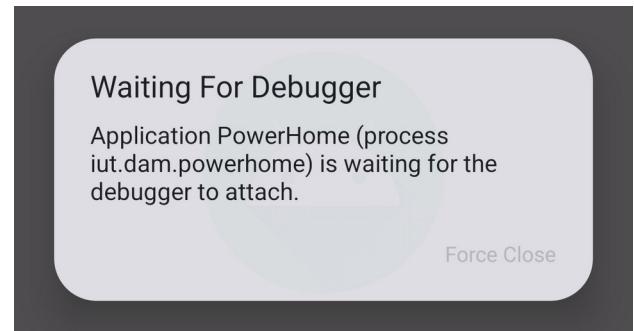
```
Log.d("MainActivity:onCreate", "Démarrage de l'activité.");
```

```
Log.e("MainActivity:onClick", "Une erreur!!!");
```

Détection et résolution des exceptions

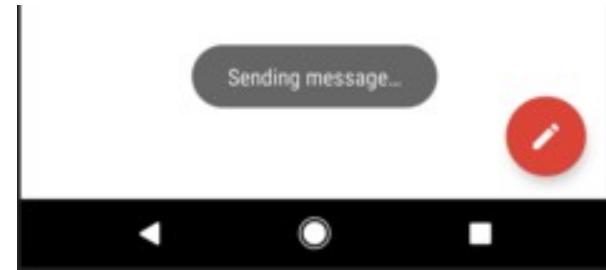
Etapes :

1. Aller à **LogCat**
2. S'assurer que l'**appareil** est bien sélectionné
3. Filtrer les Logs de type "**Error**"
4. Identifier l'**exception** et corriger le **code** en fonction



Toasts

affiche des messages d'information à l'utilisateur



Méthodes statiques de la classe Toast

- `makeText(Context context, CharSequence text, int duration)`

Exemple

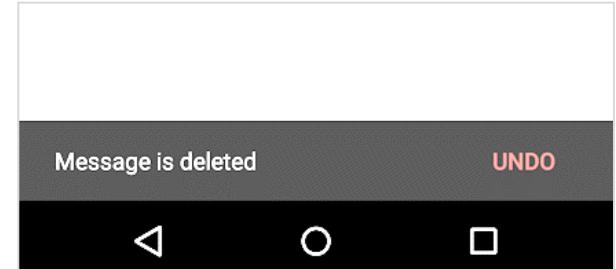
```
Toast t = Toast.makeText(v.getContext(), "msg", Toast.LENGTH_SHORT);
t.show();
```

Il est possible de personnaliser un Toast



Snackbars

affiche des messages et interagit avec l'utilisateur



Méthodes statiques de la classe Snackbar

- make(View view, CharSequence text, int duration)

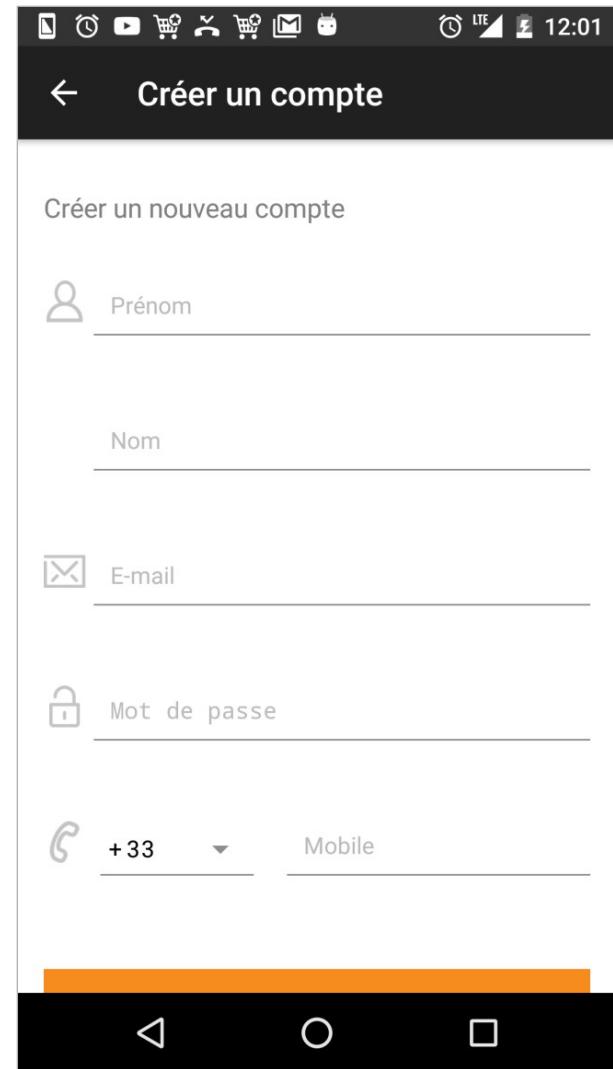
Exemple

```
Snackbar s = Snackbar.make(view, "Message...", Snackbar.LENGTH_LONG);
s.setAction("UNDO", new View.OnClickListener() {
    @Override
    public void onClick(View v) { ... }
});
s.show();
```

TP2c : Validation des champs

Valider les données des **EditText** de l'activité **RegisterActivity**

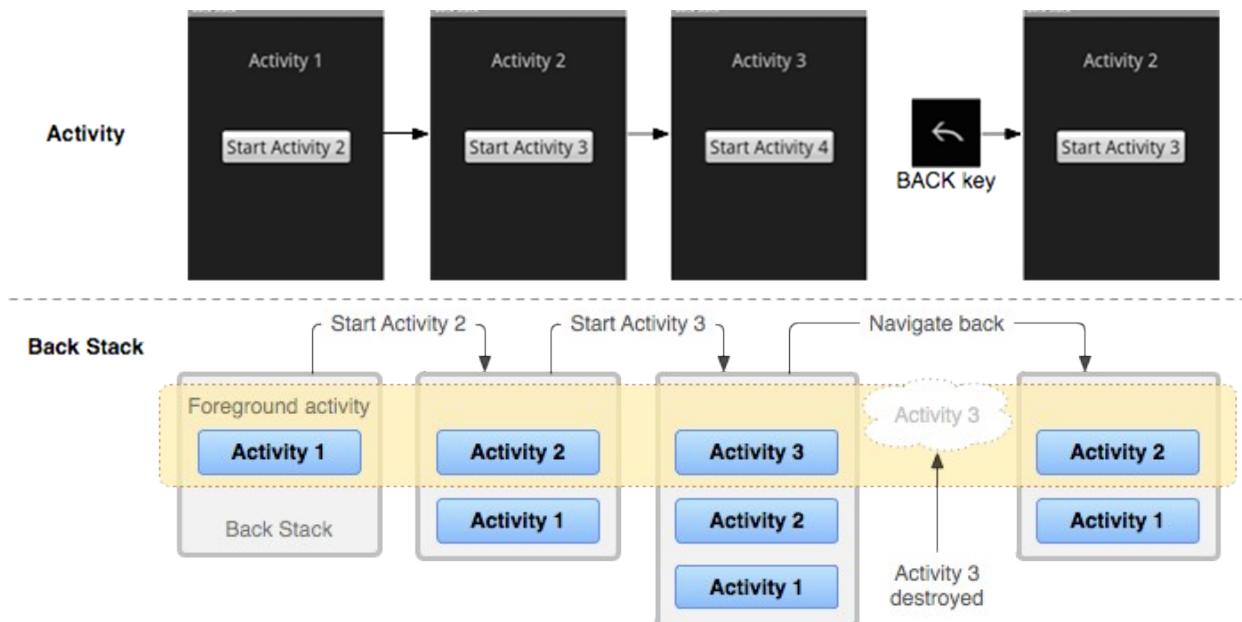
- Lors du clic sur le **Button**
- Utilisation des expressions régulières (RegEx)
- Règles spécifiques :
 - **Nom et Prénom** : alphabétique, [2 à 25 lettres]
 - **Mot de passe** : alphanumérique + symboles, min. 8 caractères



Pile des activités

Les activités sont empilées/dépilées

- Empilée quand une activité démarre
- Dépilée (détruite) quand on presse le bouton "BACK"
- Une pression sur le bouton "HOME" ne dépile pas l'activité (passe simplement en arrière plan)



Intentions

- Gérer l'envoi et la réception de messages afin de faire coopérer les activités (ou même les applications)
- Déléguer une action à un **composant**, une **application** ou une **activité de l'application courante**.

3 cas d'usage principaux des intents

Pour démarrer une activité :

- en utilisant **startActivity(Intent)**, l'intent décrit l'activité et ses paramètres

Pour démarrer un service :

- en appellant **startService(Intent)**, démarrer un service (application sans IHM)

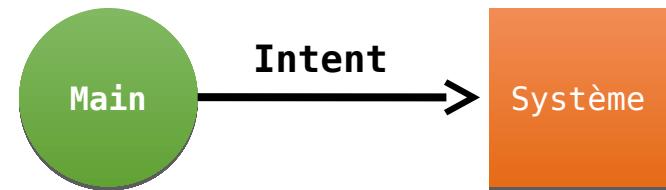
Pour envoyer un broadcast :

- en utilisant **sendBroadcast(Intent)**, un broadcast est un message que toute application peut recevoir

Types d'intentions (1/2)

Intent implicite

- Donner le nom d'une action générale
- Un composant d'une autre application peut traiter l'action
- Le système trouve la bonne application en utilisant les **intent-filters** déclarés dans le **manifest**



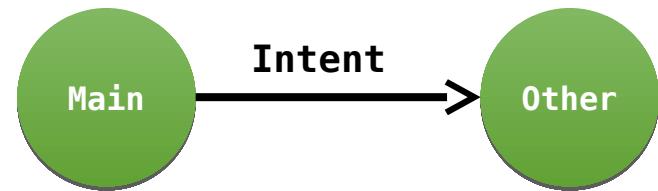
/java/MainActivity.java

```
...
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:06xxx"));
startActivity(intent);
```

Types d'intentions (2/2)

Intent explicite

- Fournir le nom de la classe de l'activité à démarrer
- Les activités doivent être de la même application



/java/MainActivity.java

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

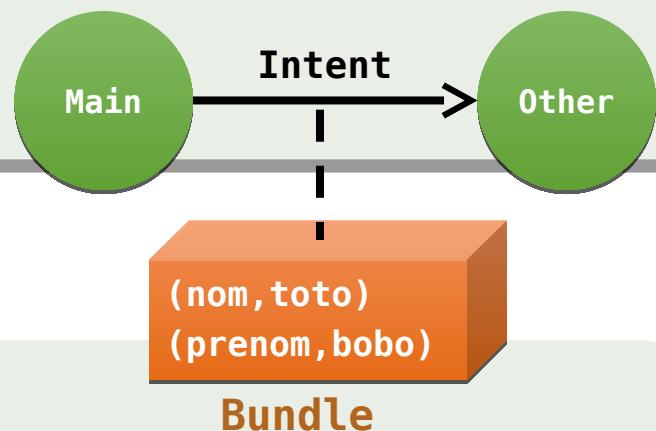
Transfert de données

/java/MainActivity.java

```
Intent intent = new Intent(this, OtherActivity.class);
Bundle bundle = new Bundle();
bundle.putString("nom", "toto");
bundle.putString("prenom", "bobo");
intent.putExtras(bundle);
startActivity(intent);
```

/java/OtherActivity.java

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



- Les types complexes (c-à-d les objets) doivent implémenter l'interface **Parcelable**, ou **Serializable**

Transfert d'objets complexes (1/2)

- Passage d'un type complexe (objet) est réalisé à travers la sérialisation
- La sérialisation (Marshaling)** permet de rendre un objet persistant pour un stockage ou un échange
- Sérialiser un objet consiste à le convertir en un tableau d'octets, qui pourra être reconstitué à l'identique à la réception
- Sa reconversion vers sa représentation initiale (dans la mémoire) est appelée **désérialisation (unmarshaling)**
- La classe de l'objet à sérialiser doit implémenter l'interface **Serializable**



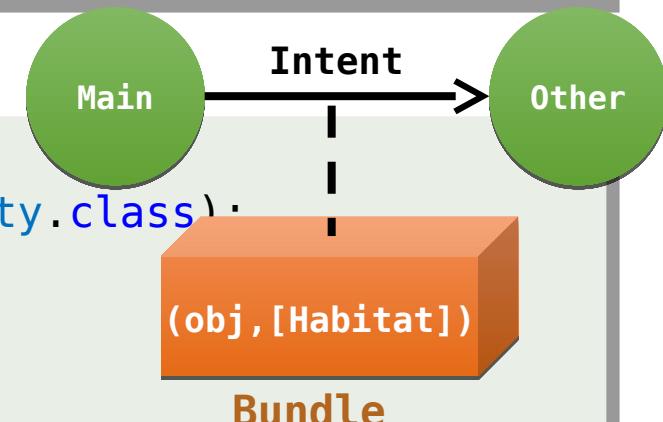
Transfert d'objets complexes (2/2)

/java/Habitat.java

```
public class Habitat implements Serializable {  
    ...  
}
```

/java/MainActivity.java

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



/java/OtherActivity.java

```
Intent intent = getIntent();  
Bundle bundle = intent.getExtras();  
Habitat h = (Habitat) bundle.getSerializable("obj");
```

TP2d : Passage vers l'activité principale

Faire le passage vers **MainActivity**

- Si les identifiants de connexion sont correctes (u: **abcd** & p: **EFGH**)
- Transférer l'identifiant et le mot de passe dans l'**Intent**
- Afficher l'identifiant et le mot de passe dans un **TextView**

