ANDROID

Laurent Provot <laurent.provot@uca.fr>

Septembre 2017

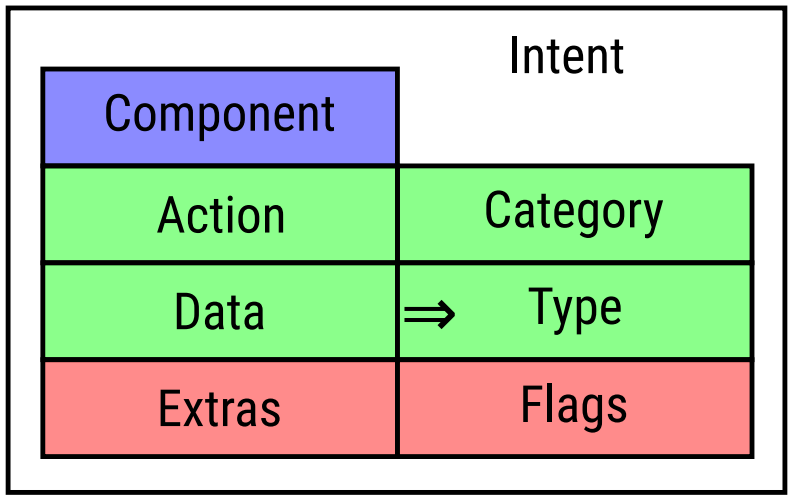
Dynamique au sein d'une application Android

Démarrage d'une activité

- On veut lancer une activité `B` depuis une activité `A`
  - Pas de `new B()` directement dans `A`
  - On doit demander au système de le faire (permissions, cycle de vie, ...)
- `void startActivity (Intent intent)`
- `Intent` : décrit quoi lancer et comment

Les intentions

- Composant de communication au sein du système
- Agrégat d'informations diverses utiles au système pour faire des choix



- 2 types : explicite et implicite, suivant les infos renseignées
- En commun :
  - Extras : `Bundle` de données supplémentaires
  - Flags : comment l'activité est lancée

Les intentions

1. Intentions explicites (*Excplicit Intent*)

- On décrit explicitement quel composant démarrer
  - `Intent(Context packageContext, Class<?> cls)`
    - `packageContext` : contexte du pkg applicatif de la classe
    - `cls` : méta-classe représentant l'activité à lancer
  - Setters (`setComponent(…)` , `setClass(…)` , `setClasseName(…)`)
- L'activité à lancer doit être déclarée dans le Manifest sinon `ActivityNotFoundException`

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" …>
    <application>
        …
        <activity android:name=".CheatActivity" />
    </application>
</manifest>
```

Intention explicite : exemple

```
public class A extends Activity {
    protected void onCreate(Bundle state){
        super.onCreate(state);
        setContentView(R.layout.activity_main);

        Button launchButton = (Button) findViewById(R.id.button_launch);
        launchButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent launchIntent = new Intent(A.this, B.class);
                startActivity(intent);
            }
        });
    }
}
```

Les intentions

2 Intentions implicites (*Implicit Intent*)

- On décrit l'action à faire et on laisse l'OS choisir les candidats
- Création d'un `Intent` «vide» puis customisation avec setters
- Infos portées par l'intention :
  - **Action** :
    le «truc» générique à faire `ACTION_VIEW`, `ACTION_SEND`, …
    (cf. classe `Intent` et méthode `setAction(…)`)
  - **Data & Type** :
    URI vers la donnée et/ou type MIME (cf. `setData(…)`, `setType(…)`, `setDataAndType(…)`)
  - **Category** :
    info additionnelle sur l'action (`CATEGORY_BROWSABLE`, `CATEGORY_HOME`, …)
- `resolveActivity(…)` sur l'intent pour être sûr qu'il existe une activité qui l'accepte

Intention implicite : exemple

```
public class A extends Activity {
    …
    public void composeEmail(String[] addresses, String suject) {
        Intent mailIntent = new Intent(Intent.ACTION_SENDTO);
        mailIntent.setData(Uri.parse("mailto:");
        mailIntent.putExtra(Intent.EXTRA_EMAIL, addresses);
        mailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
        if (mailIntent.resolveActivity(getPackageManager()) != null) {
            startActivity(mailIntent);
        }
    }
    …
}
```

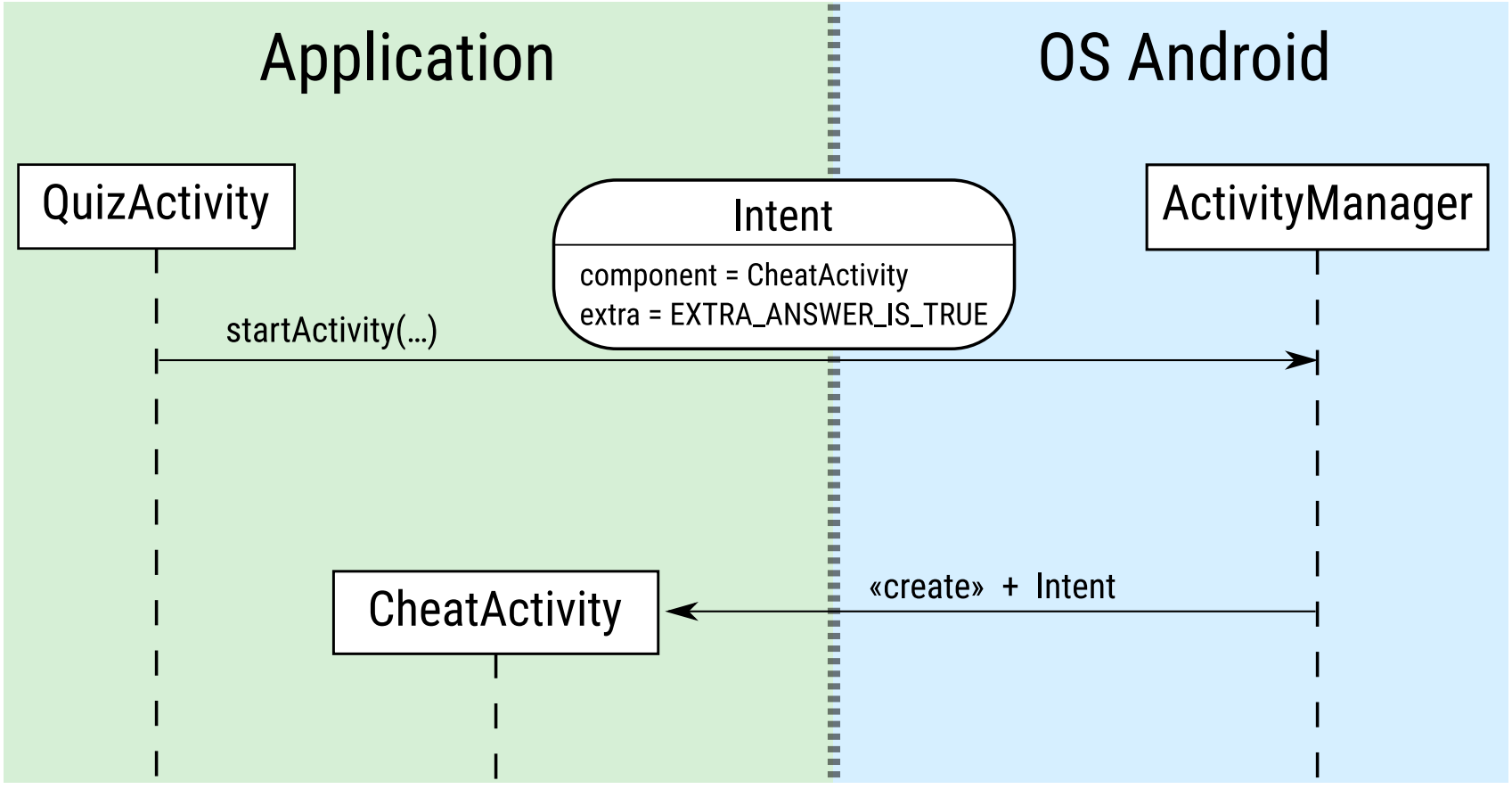
Passage d'arguments

- L'activité `B` à lancer a besoin de données en entrée
- Utilisation du bundle `extras` de l'intent
  (`putExtra(String key, XXX value)`
  ou `getExtras()` + setters)
- Bundle : table associative, clé → valeur
  - L'activité qui lance `B` doit connaître les clés ?
  - Découplage grâce à une *factory method*

```
public class B extends Activity {
    private static final String EXTRA_NAME_KEY = "org.pkg.name_key";

    public static Intent newIntent(Context context, XXX x) {
        Intent i = new Intent(context, B.class);
        i.putExtra(EXTRA_NAME_KEY, x);
        return i;
    }
    …
}
```

Passage d'arguments : récapitulatif (Quiz)



Démarrage d'une activité

- Quid du cycle de vie des 2 activités `A` et `B` ?
- Il faut conserver une bonne expérience utilisateur
- Tant que `B` n'est pas prête à être interactive `A` reste visible

1. `A.onPause()`
2. `B.onCreate(…)` → `B.onStart()` → `B.onResume()`
3. `A.onStop()`

- Attention à ne pas mettre des opérations coûteuses dans ces méthodes

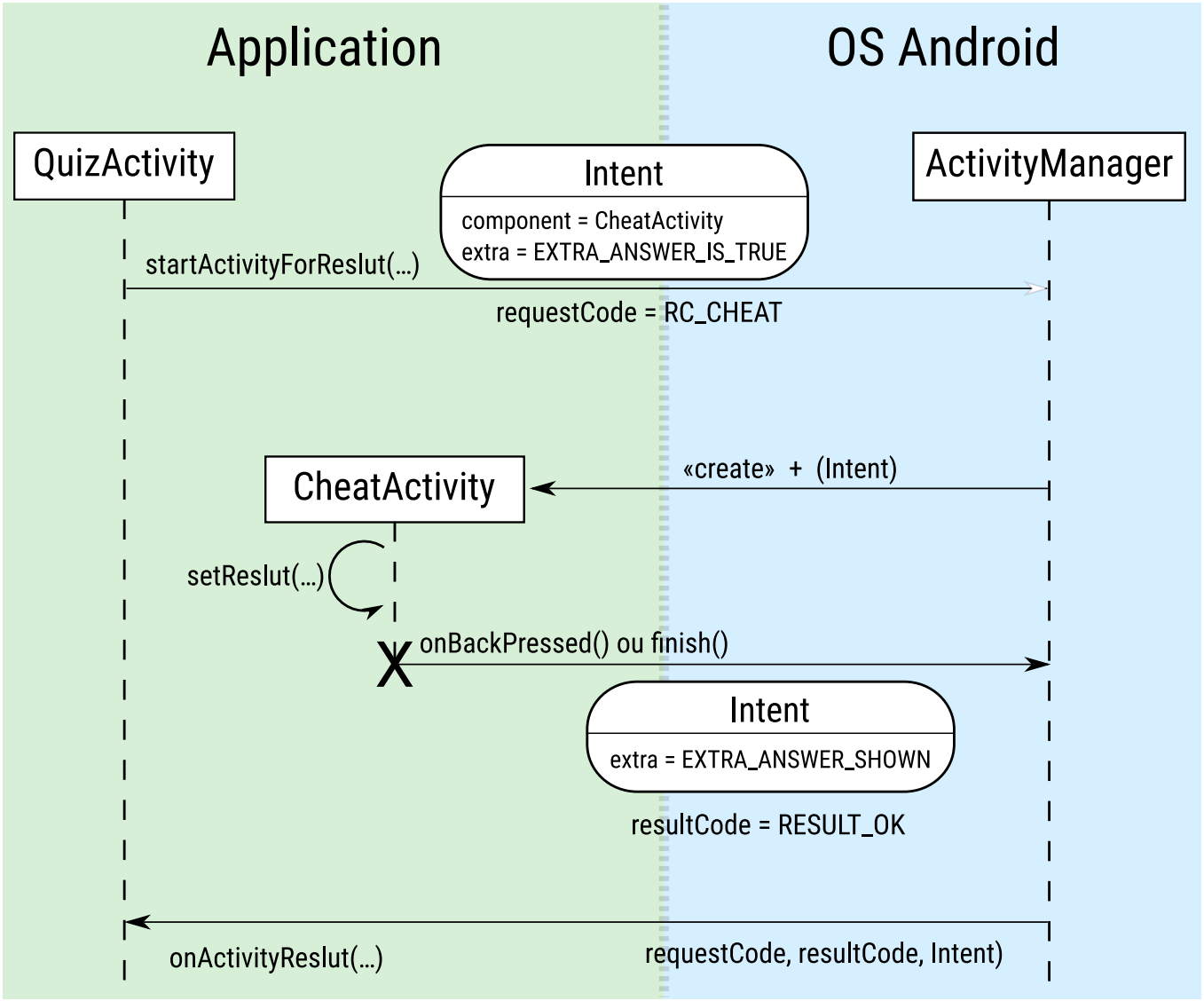
Données en retour d'une activité

- `B` n'a pas forcément connaissance de l'activité `A` qui l'a lancée
- Le même mécanisme de passage d'arguments ne peut pas être employé
- `A` devra préciser qu'elle lance `B` en espérant un résultat
- `void startActivityForResult ( Intent intent, int requestCode)`
  - `intent` : décrit quoi lancer
  - `requestCode` : associe un code à la demande

Données en retour d'une activité

- `B` spécifiera les informations à retourner à travers une intent
- `void setResult (int resultCode, Intent data)`
  - `resultCode` : entier décrivant l'issue de l'action
    `RESULT_OK`, `RESULT_CANCELED`, `RESULT_FIRST_USER`
  - `data` : intent pour passer des données en retour
- `A` doit redéfinir `onActivityResult(…)` pour traiter le retour de `B`
- `protected void onActivityResult (`
  `int reqCode, int resultCode, Intent data)`
  - `reqCode` : le code associé au lancement de `B`
  - `resultCode` : le code de retour de `B`
  - `data` : les données retournées par `B`

Données en retour : récapitulatif (Quiz)



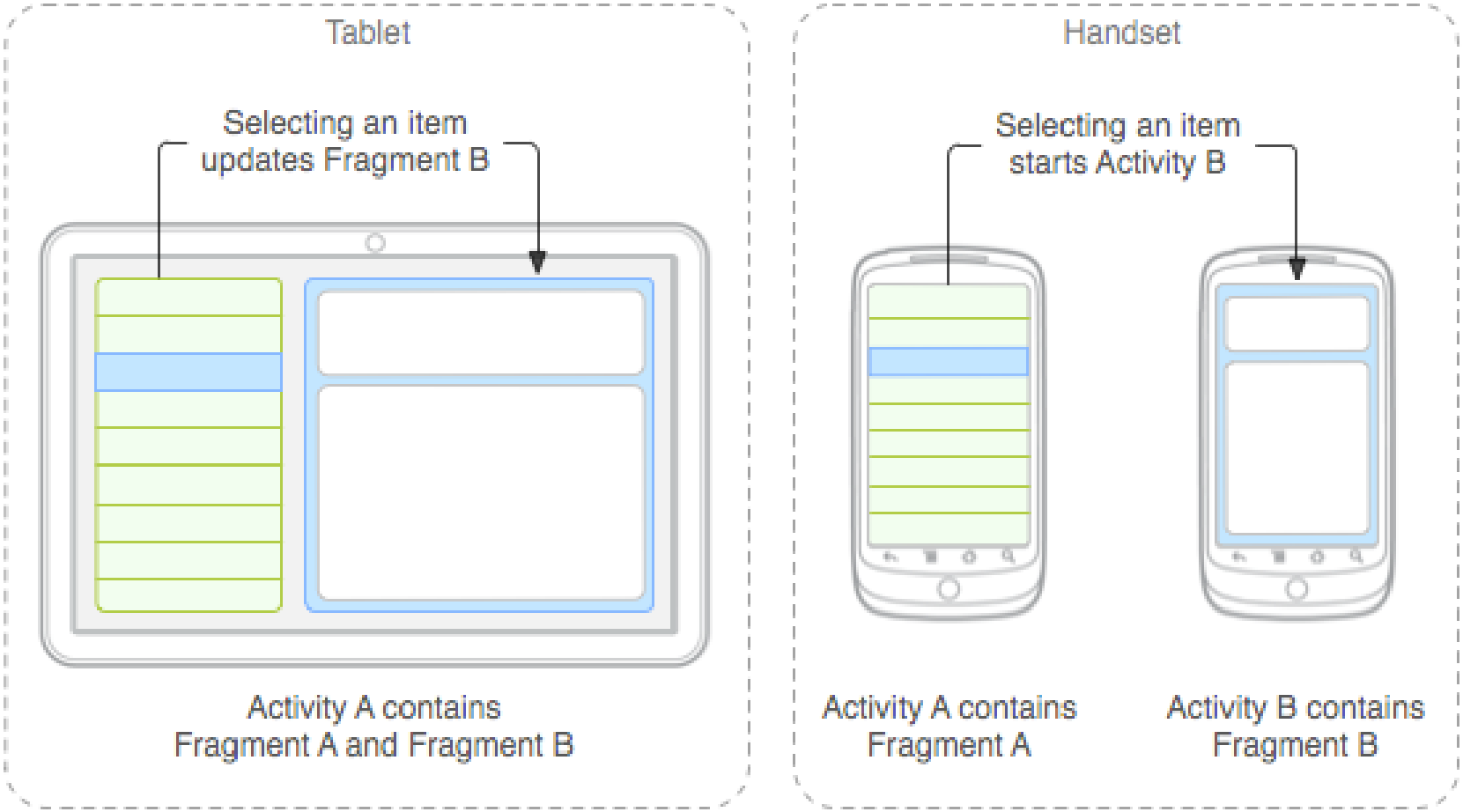
Données en retour d'une activité

- `onActivityResult(…)` est appelée entre `onStart()` et `onResume()`
- `setResult(…)` doit être appelé avant un appel à `finish()` sinon `resultCode = RESULT_CANCELED` et `data = null`
- Attention `onBackPressed()` appelle par défaut `finish()`
- Du coup `setResult(…)` dans `onPause()` : c'est trop tard !

Master-Detail et Fragments

Master Detail et Fragments

- Préserver l'ergonomie pour différentes tailles d'écran



Fragment

- But : séparer les composants d'une activité (complexe) en sous-composants autonomes
- Apparu avec HoneyComb (API v11)
- Un fragment = (en quelque sorte) une sous-activité
- Un fragment doit être lié avec une activité pour être utile
- Un fragment ne doit pas être dépendant de l'activité qui le contient (découplage)
- Un fragment possède son propre cycle de vie

Fragments

- 2 possibilités pour utiliser les fragments

1. Utiliser les fragments natifs :
   - Dispo de base dans la classe `Activity` si $>$ API v11
   - Profite des dernières avancées du framework
   - Pas de support pour les appareils pré Andoid HoneyComb
2. Utiliser la *Support Library* :
   - Dériver de `FragmentActivity` (ou `AppCompatActivity`)
   - Utiliser certaines méthodes « alternatives » `getSupportFragmentManager()` , `getSupportActionBar()`
   - Backport pour les anciennes versions d'Android (jusqu'à v4)
   - Mises à jour régulières
   - En prime : `Loader` , `Toolbar` , Material Design, ...

- Conseil : utiliser la *Support Library*

Cycle de vie Fragment vs Activity

Cf. Cycle de vie complet

Ajout d'un fragment statique

- Implémentation rapide
- Design figé, on ne peut pas changer de fragment

1. Au niveau du fragment : création de la vue dans le `onCreateView(…)`

```
package fr.iut.MyFragment;
public class MyFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_layout,
                                     container, false);
        // configuration de la vue (findViewById, …)
        return view;
    }
}
```

Ajout d'un fragment statique

2. Au niveau de l'activité : ajout d'un élément `fragment` dans le layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" … >
    <fragment
        android:id="@+id/id_fragment"
        android:name="fr.iut.MyFragment" />
</LinearLayout>
```

3. Il y a juste à lier l'activité à son layout

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    MyFragment fragment = (MyFragment) findViewById(R.id.id_fragment);
}
```

Ajout d'un fragment dynamiquement

- On utilise le `FragmentManager`
- Il opère des transactions pour ajouter/remplacer/supprimer les fragments
- Il permet de gérer la backstack des fragments (gestion non automatique)

1. Au niveau de `MyFragment` rien ne change, la vue est toujours créée dans le `onCreateView(…)`
2. Au niveau de l'activité : le layout contient un « placeholder » pour le fragment (généralement un `FrameLayout`)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" … >
    <FrameLayout
        android:id="@+id/id_fragment" />
</LinearLayout>
```

Ajout d'un fragment dynamiquement

3. Il faut lier l'activité à son layout et charger le fragment manuellement

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    FragmentManager fragmentManager = getFragmentManager();
    if (fragmentManager.findFragmentById(R.id.id_fragment) == null) {
        fragmentManager.beginTransaction()
                       .add(R.id.id_fragment, new MyFragment())
                       .commit();
    }
}
```

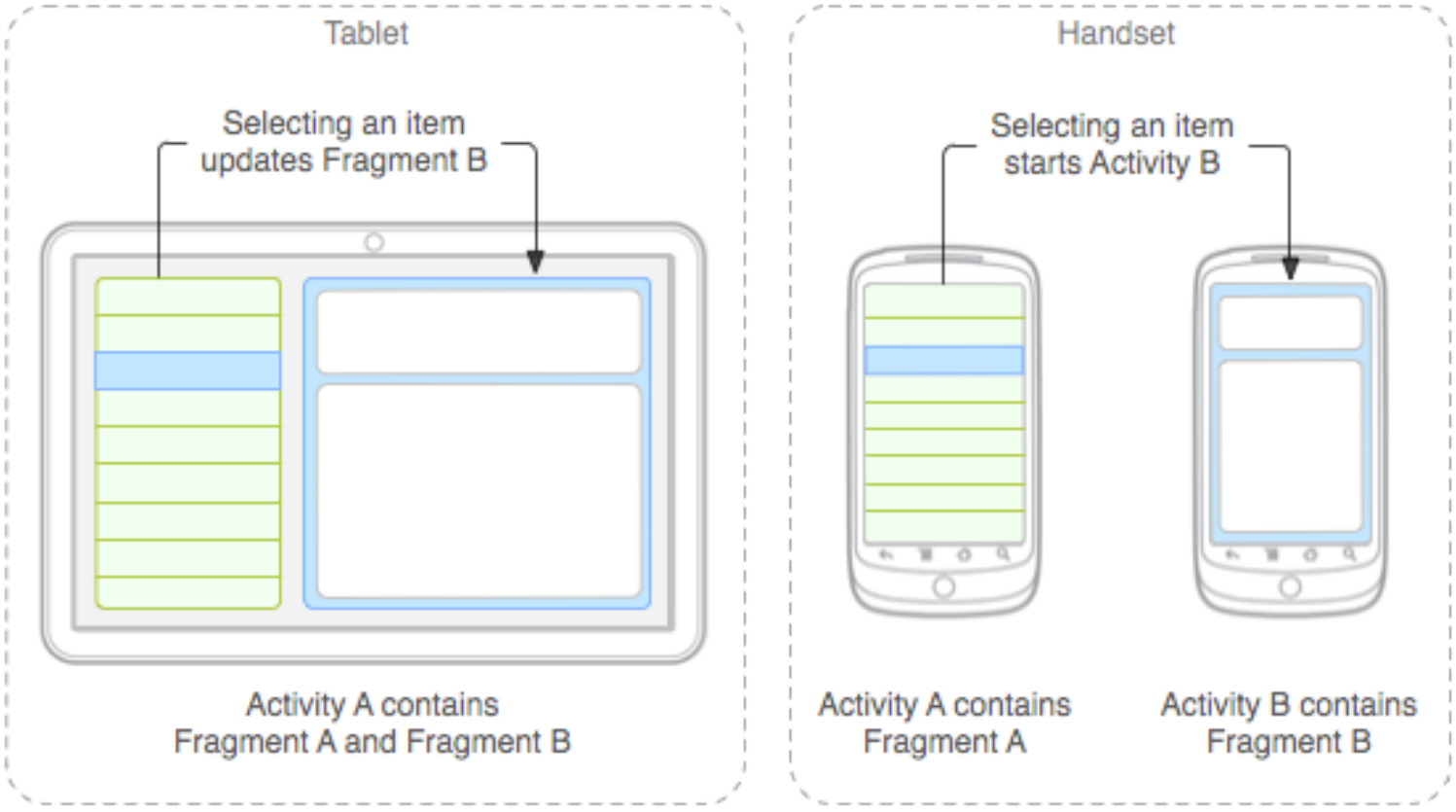
Backstack des fragments

- L'`ActivityManager` gère la backstack des activités automatiquement
- Pour les fragments il faut dire explicitement au `FragmentManager` de le faire

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
// gestion de la transaction
ft.addToBackStack("Tag transaction");
ft.commit();
```

```
@Override
public void onBackPressed() {
    FragmentManager fm = getFragmentManager();
    if (fm.getBackStackEntryCount() > 0) {
        fm.popBackStack();
    } else {
        super.onBackPressed();
    }
}
```

Retour au Master Detail



- Gérer la vue de type liste
- Gérer les activités / fragments suivant le type d'affichage

RecyclerView

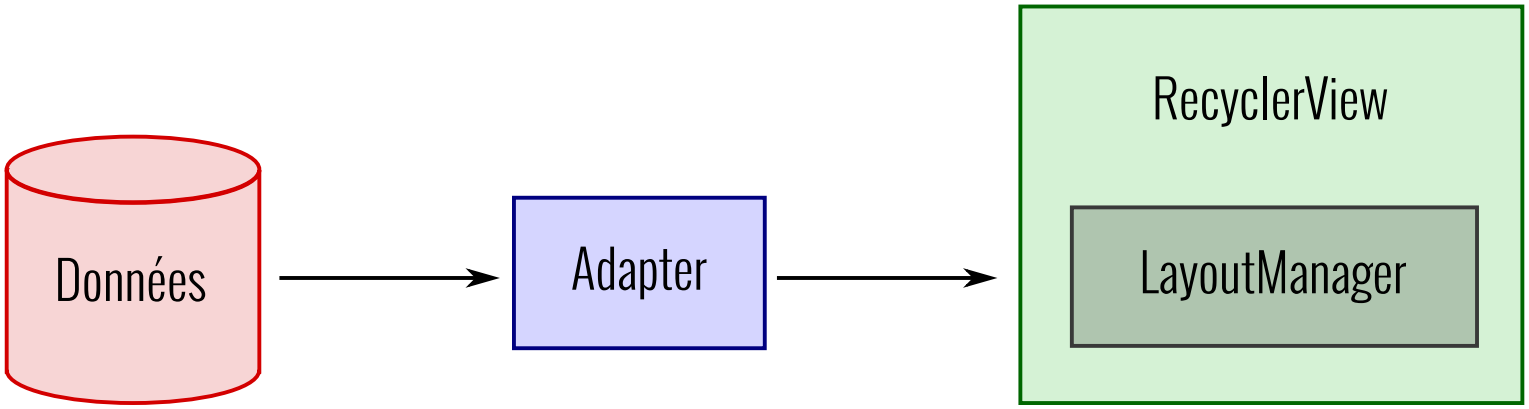
Une liste écolo

- On veut afficher des données (un grand nombre)
- Il faut définir « comment » les afficher
- Il faut en afficher seulement un sous-ensemble restreint à la fois
- Il faut conserver une certaine efficacité
- Il faut rester évolutif

- C'est le rôle de la `RecyclerView` (*Support Library*)

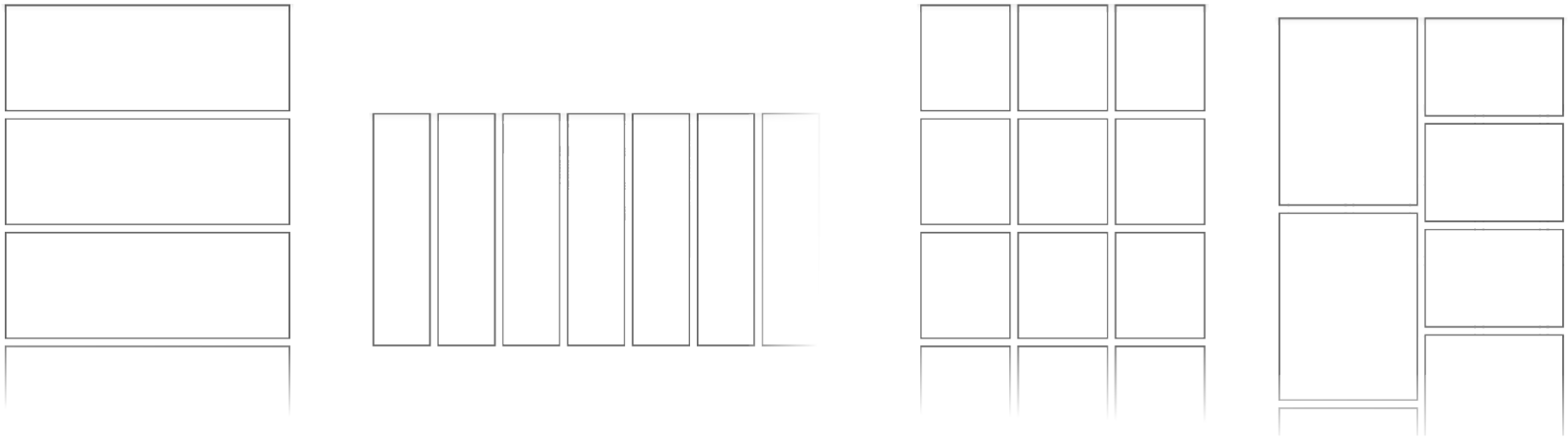
Lien avec le modèle

- Articulation de la `RecyclerView`



- La `RecyclerView` décide combien d'éléments afficher
- Le `LayoutManager` gère l'agencement des vues des éléments
- L'`Adapter` permet d'obtenir des vues en fonction des données

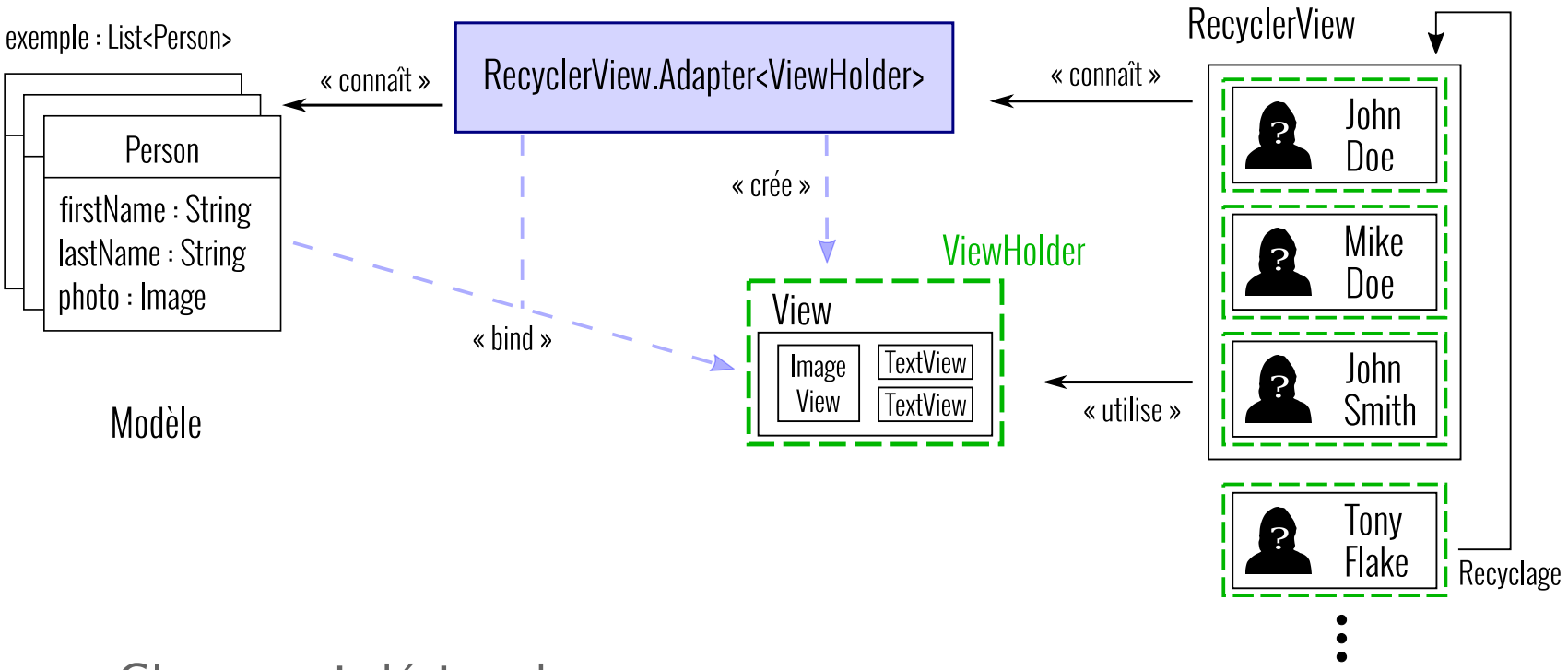
L'agencement des vues



- Classes dérivées de `RecyclerView.LayoutManager`
- De base : `LinearLayoutManager`, `GridLayoutManager`, `StaggeredGridLayoutManager`

```
mRecyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
// optimisation si la taille de la RecyclerView ne change pas
mRecyclerView.setHasFixedSize(true);
```

Adaptation des données



- Classe qui dérive de

  `RecyclerVew.Adapter<RecyclerView.ViewHolder>`

  - Connaître la donnée à afficher
  - Pouvoir créer les vues
  - Être efficace avec beaucoup d'éléments

Méthodes à redéfinir

- `public int getItemCount()`
  permet à la `RecyclerView` de savoir combien d'éléments elle va devoir gérer
- `public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)`
  permet de créer un `ViewHolder` qui gère une vue d'un élément
- `public void onBindViewHolder(ViewHolder holder, int position)`
  permet de lier la donnée métier à sa vue; leur relation est leur position (index)

- À quoi sert le `ViewHolder` ? Efficacité (car ré-instanciation de vue et `findViewById(…)` coûteux)

Lien entre le Master et le Detail

1. Les activités agencent les fragments
2. Les fragments sont indépendants de l'activité

- Comment l'information est-elle passée entre un fragment et une activité ?
- Clic sur un élément de liste : lancement d'activité ou remplacement de fragment ?

Passage d'arguments

- Le fragment est lié à une activité : `getActivity()`
- On pourrait utiliser `getActivity().getIntent()` ?
- Pour un vrai découplage les fragments possèdent leur propres arguments
- Bundle géré avec les accesseurs `(get|set)Arguments(…)`
- Même principe que pour le lancement d'activité → *factory method* pour l'instanciation du fragment :

```
public static MyFragment newInstance(XXX param) {
    Bundle args = new Bundle();
    args.putXXX(param);
    MyFragment fragment = new MyFragment();
    fragment.setArguments(args);
    return fragment;
}
```

Communication entre le fragment et son activité

- On veut conserver une indépendance du fragment
- Comme le fragment ne connait pas son contexte, il notifie juste des interactions en son sein
- Principe des *listeners* en Java
- On définit une interface pour les échanges
- L'activité implémente cette interface

```
public class MyFragment extends Fragment {
    public interface OnSomethingListener {
        public void onSomething(XXX param);
    }
    …
}

public class MyActivity implements MyFragment.OnSomethingListener {
    @Override public void onSomething(XXX param) { … }
}
```