CAI-IP03 SMB116

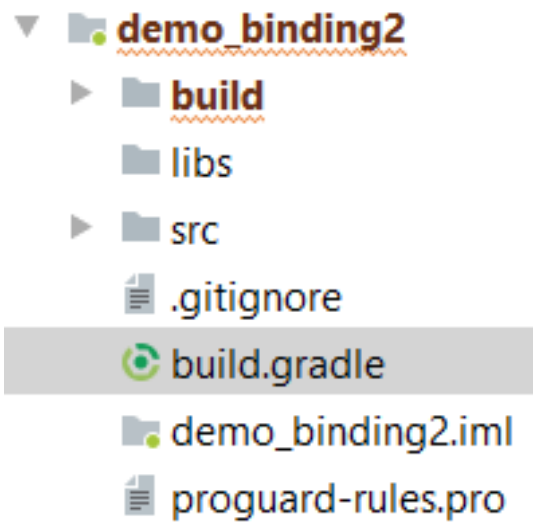
Le **Binding**

Binding ou DataBinding

- Le **Data Binding** est le processus qui permet d'établir un lien entre l'**UI** et la **logique métier implantée dans un contrôleur**.
- La représentation des données est liée aux données → Le changement des données est reflété automatiquement dans l'UI.
- Le lien entre les événements déclenchés par des interactions utilisateurs et leur traitement peut aussi être réalisé en utilisant le **Binding**.
- Sous Android la solution de Binding est réalisée en utilisant Graddle pour générer les classes nécessaires

Implémentation (1) – Configuration du Build

- Activation du Data Binding dans le cycle de Build.
- Dans le **build.gradle** un élément **dataBinding** avec la propriété **enabled** à la valeur **true**




```
android {
    ....
    dataBinding {
        enabled = true
    }
}
```

Implémentation (2)

- Les **données à lier** sont de simples classes.

- Les **propriétés utilisables dans le Binding** sont **exposées** par les **Accesseurs** et les **Modifieurs**

```
public class Personne {

    protected String nom;
    protected String prenom;

    public Personne() {
    }

    public Personne(String nom, String prenom) {
        this.nom = nom;
        this.prenom = prenom;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}
```

Implémentation (3) – Le Layout

- Les données à lier sont **déclarées** en début du Layout dans la balise **Data**

```
<data>
    <variable name="personne" type="edu.cai.demo_binding2.bindable.Personne" />
</data>
```

- Les **données sont liées** dans le **Layout** en tant que valeur des propriétés des composants déclarés.

- Les valeurs liées sont définies par des expressions préfixées par @ et écrites entre {}

```
android:text="@{personne.nom}"
```

Implémentation (4) - Le Layout

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

     <data>
          <variable name="personne" type="edu.cai.demo_binding2.bindable.Personne" />
      </data>

    <LinearLayout
        . . .
        <TextView
            android:id="@+id/txtNom"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@{personne.nom}"
            android:textSize="30sp"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/txtPrenom"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@{personne.prenom}"
            android:textSize="30sp"
            android:textStyle="bold" />

    </LinearLayout>
</layout>
```

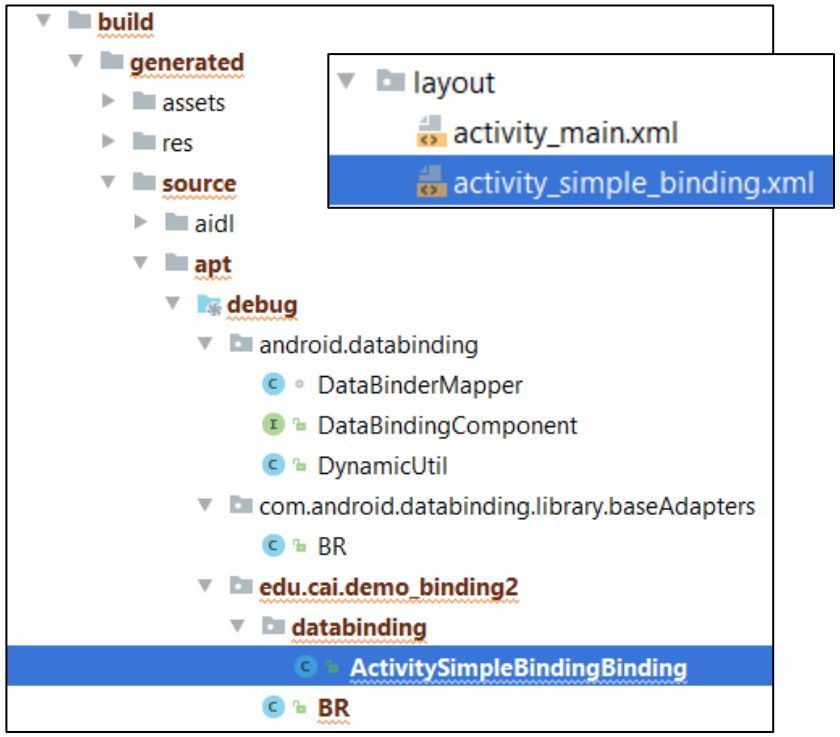
Implémentation (5) - Binding de listener

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable name="listener" type="edu.cai.demo_binding2.MonListener" />
        <variable name="personne" type="edu.cai.demo_binding2.bindable.Personne" />
    </data>

    <LinearLayout
        . . .
        <TextView
            android:id="@+id/txtNom"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@{personne.nom}"
            android:textSize="30sp"
            android:textStyle="bold"
            <!-- Méthode par référence (possibilité d'utiliser Lambda) -->
            android:onClick="@{listener::clickSurTexte}"
            />

            . . .
    </LinearLayout>
</layout>
```

```java
/**
 * Classe Listener
 */
public class MonListener {
    public void clickSurTexte(View view) {
        Toast.makeText(view.getContext(),
                text: "clickSurTexte",
                Toast.LENGTH_SHORT).show();
    }
}
```

Implémentation (6) - Le lien avec l'Activity

- Le Build avec **Graddle** génère une classe dont le nom est celui du Layout de l'activité suffixée par **« Binding »** :




```
public class ActivitySimpleBindingBinding extends android.databinding.ViewDataBinding  {

    . . .

    public void setListener(@Nullable edu.cai.demo_binding2.MonListener Listener) {
        this.mListener = Listener;
        synchronized(this) {
            mDirtyFlags |= 0x1L;
        }
        notifyPropertyChanged(BR.listener);
        super.requestRebind();
    }
    @Nullable
    public edu.cai.demo_binding2.MonListener getListener() {
        return mListener;
    }
    public void setPersonne(@Nullable edu.cai.demo_binding2.bindable.Personne Personne) {
        this.mPersonne = Personne;
        synchronized(this) {
            mDirtyFlags |= 0x2L;
        }
        notifyPropertyChanged(BR.personne);
        super.requestRebind();
    }
    @Nullable
    public edu.cai.demo_binding2.bindable.Personne getPersonne() {
        return mPersonne;
    }
. . .
```

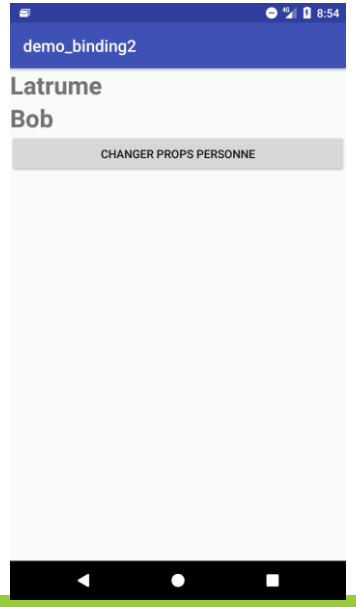
Implémentation (7) - Le Binding dans l'Activity

- La classe générée par **Graddle** pendant le Build est utilisée dans l'Activity :

```
public class SimpleBindingActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_binding);

        Personne personne = new Personne("Latrume", "Bob");
        // Obtention d'une instance de la classe génére pour le Binding
        ActivitySimpleBindingBinding binding = DataBindingUtil.setContentView(this, R.layout.activity_simple_binding);
        // Alimentation des données à lier (celle définies dans Data du Layout)
        binding.setPersonne(personne);
        binding.setListener(new MonListener());
    }

}
```

Démo Simple Binding

- **Bilan :**
  - Binding fonctionnel: **données -> UI**
  - Sur changement dans UI la données est elle mise à jour ? **NON**
  - Sur changement des données : l'UI est elle mise à jour automatiquement ? **NON**
  - Pour **mettre à jour l'UI** il faut **réaffecter la donnée à la classe Binding**




```
public void onBtnChgNomPersonne(View view) {
    personne.setNom(personne.getNom() + (new Date().getTime() % 30) );
    personne.setPrenom(personne.getNom() + (new Date().getTime() % 98)
);
    binding.setPersonne(personne);
}
```

Astuces diverses

- L'attribut **« class »** de la balise **data** permet de spécifier le nom de la classe de Binding à générer.

```
<data class="ClasseDeBindingGeneree">
    <variable name="listener" type="edu.cai.demo_binding2.MonListener" />
    <variable name="personne" type="edu.cai.demo_binding2.bindable.Personne" />
</data>
```

- Possibilité d'intégrer des **fonctions et** des **opérateurs dans** les **expressions**:

  android:text="@{String.valueOf(index + 1)} " **(Fonction)**

  android:visibility="@{personne.nom.length() < 5 ? View.GONE : View.VISIBLE} " **(Ternaire)**

  android:transitionName='@{"image_" + id} ' **(Concaténation)**

Astuces diverses

- Importation de types pour pouvoir les utiliser

```
<data class="ClasseDeBindingGeneree">
    <import type="java.util.List"
    <import type="edu.cai.demo_binding2.bindable.Personne"
    <variable name="listener" type="edu.cai.demo_binding2.MonListener" />
    <variable name="personne" type="Personne" />
    <variable name="listePersonnes" type="List&lt;Personne&gt;" />
</data>
```

- Possibilité si nécessaire de créer des **Setter** spécifiques :

```
<ImageView app:imageUrl="@{venue.imageUrl}" app:error="@{@drawable/venueError}" />
```

```
@BindingAdapter({"imageUrl", "error"})
public static void loadImage(ImageView view, String url, Drawable error) {
  Picasso.get().load(url).error(error).into(view);
}
```

Astuces diverses

- Création de **Converter** spécifiques (utilisé lorsque la valeur affectée n'est pas du type attendu)

```
@BindingConversion
public static ColorDrawable convertColorToDrawable(int color) {
   return new ColorDrawable(color);
}
```

- Utilisation des **Collections** :

```
<data>
    <import type="android.util.SparseArray"/>
    <import type="java.util.Map"/>
    <import type="java.util.List"/>
    <variable name="list" type="List&lt;String&gt;"/>
    <variable name="sparse" type="SparseArray&lt;String&gt;"/>
    <variable name="map" type="Map&lt;String, String&gt;"/>
    <variable name="index" type="int"/>
    <variable name="key" type="String"/>
</data>
```

```
android:text="@{list[index]}"
…
android:text="@{sparse[index]}"
…
android:text="@{map[key]}"
```

Binding Bidirectionnel - Donnée

- Données à lier sont des observables (Design Pattern Observateur)
- Possibilité de notifier les observateurs sur changement de valeur

```
public class PersonneObservable extends BaseObservable {

    protected String nom;
    protected String prenom;

    public PersonneObservable() {
    }

    public PersonneObservable(String nom, String prenom) {
        this.nom = nom;
        this.prenom = prenom;
    }

    @Bindable
    public String getNom() {
        return nom;
    }
```

```
    @Bindable
    public void setNom(String nom) {
        this.nom = nom;
        notifyPropertyChanged(BR.nom);
    }

    @Bindable
    public String getPrenom() {
        return prenom;
    }

    @Bindable
    public void setPrenom(String prenom) {
        this.prenom = prenom;
        notifyPropertyChanged(BR.prenom);
    }
}
```

Binding Bidirectionnel - Layout

- **Layout identique à unidirectionnel** mis à part: **@=** en préfixe des expressions de binding
- **@=** est une directive pour que la **classe de Binding générée** utilise le Modifieur (Setter) pour affecter la **nouvelle valeur lorsqu'elle change**.

```
<layout . . .>

    <data class="ClasseDeBindingBidirectionelGeneree">
        <variable name="personne" type="edu.cai.demo_binding2.bindable.PersonneObservable" />
    </data>
    . . .
   <EditText
     android:id="@+id/txtNom"
     android:text="@={personne.nom}"
     android:inputType="textPersonName" />

    <EditText
      android:id="@+id/txtPrenom"
      android:text="@={personne.prenom}"
      android:inputType="textPersonName" />

    <TextView
      android:id="@+id/lblNomPrenom"
      android:text="@{personne.prenom + ' ' + personne.nom}" />
</layout>
```

Binding Bidirectionnel – L'activity

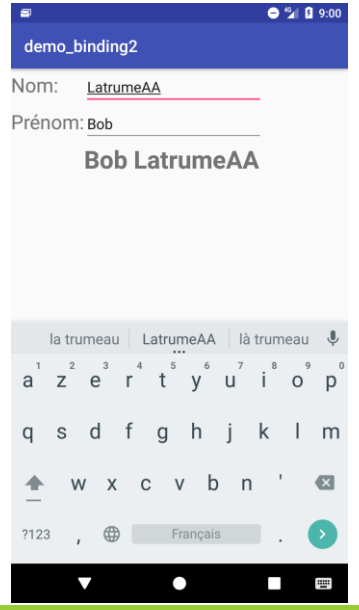
- Obtention instance de la classe de Binding générée
- Affectation à la classe de Binding d'une instance de Personne Observable
- Lorsque la valeur d'un EditText lié change, la classe de Binding appelle le Setter associé
- Dans le code du setter associé nous avons ajouté un instruction pour notifier les observateurs:

```
notifyPropertyChanged(BR.nom);
```

```
public class BidirectionalBindingActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_bidirectional_binding);

        PersonneObservable personne = new PersonneObservable("Latrume", "Bob");

        ClasseDeBindingBidirectionelGeneree binding = DataBindingUtil.setContentView(this, R.layout.activity_bidirectional_binding);
        binding.setPersonne(personne);

    }
}
```

Démo Binding Bidirectionnel

- **Bilan :**
  - Binding bidirectionnel fonctionnel: **données <---> UI**
  - Sur changement dans UI la données est elle mise à jour ? **OUI** (Appel du Setter)
  - Sur changement des données : l'UI est elle mise à jour automatiquement ? **OUI Presque**
  - Le changement est notifié par l'Observable. L' observateurs se se charge d'actualiser l'UI




```
@Bindable
public void setPrenom(String prenom) {
    this.prenom = prenom;
    notifyPropertyChanged(BR.prenom);
}
```