Les Application mobiles

Silem Abdelheq

1.0

Mars 2023

Table des matières

Objectifs 3

I - Chapitre 4 : Interfaces graphiques et widgets 4

1. Objectives 4

2. Introduction aux interfaces utilisateurs 5

2.1. View et Widgets 5

2.2. La hiérarchie de la classe View 5

2.3. La hiérarchie des Views des interfaces Android 6

3. Les Layouts 7

3.1. Linear Layout 7

3.2. RelativeLayout 9

3.3. Constraint Layout 11

4. Les Fragments 12

4.1. Cycle de vie d'un fragment 12

4.2. Comment utiliser le fragment 13

5. Les Widgets 15

5.1. Widgets de base 16

5.2. Widgets avancée 18

5.3. Les gestionnaires d'événements 20

Objectifs

la finalité de cette matière est d'apporter à l'étudiant des connaissances en matière de développement d'application et système informatique dans des environnements mobiles. Avec l'arrivée des smartphones les applications mobiles sont omniprésents que l'on soit client (BtoC), fournisseur (BtoB) ou collaborateur (BtoE). Le but de ce cours est aussi d'apprendre la programmation sous Android, sa plate-forme de développement et les spécificités du développement embarqué sur smartphone[

I Chapitre 4 : Interfaces graphiques et widgets

Les interfaces graphiques jouent un rôle crucial dans le développement des applications Android en offrant une expérience utilisateur conviviale et intuitive. Les utilisateurs peuvent interagir avec l'application via des boutons, des champs de texte, des listes déroulantes, des cases à cocher et d'autres widgets.

Les vues et les widgets sont essentiels pour créer des interfaces graphiques pour les applications Android. Bien que les termes "vues" et "widgets" soient parfois utilisés de manière interchangeable, les vues sont généralement considérées comme les composants fondamentaux de l'interface utilisateur. Les widgets, quant à eux, sont des éléments d'interface utilisateur principalement visuels et sont placés sur l'écran de l'application.

Ce chapitre explore les concepts fondamentaux des interfaces et des widgets dans le développement Android, y compris la hiérarchie de la classe View et la hiérarchie des vues dans les interfaces des applications Android. Il couvre les différents layouts et la manière de les optimiser en termes de performances, ainsi que les fragments et leurs méthodes de cycle de vie. Le chapitre explore également les différents types de widgets disponibles, des éléments d'interface utilisateur de base tels que les boutons et les TextViews aux widgets plus avancés tels que les RecyclerViews et les ListViews. Des conseils et des bonnes pratiques pour optimiser les performances et améliorer l'expérience de l'utilisateur sont fournis tout au long du chapitre.

1. Objectives

Fondamental

À la fin de ce chapitre, l'étudiant sera en mesure de :

- Comprendre les interfaces graphiques et les widgets et leur importance dans le développement d'une application Android.
- Utiliser les layouts pour organiser les différents éléments de l'interface graphique de l'application.
- Maîtriser l'utilisation des widgets de base tels que TextView, EditText, Button, etc.
- Créer des menus et des barres d'outils pour permettre à l'utilisateur d'interagir avec l'application.
- Implémenter des événements d'interaction utilisateur avec les widgets.
- Personnaliser l'apparence de l'application en utilisant des styles et des thèmes.
- Comprendre le concept de fragments et leur rôle dans la conception des interfaces graphiques, ainsi que comment les utiliser pour améliorer l'expérience utilisateur.

2. Introduction aux interfaces utilisateurs

Dans le cadre du développement d'Android, les interfaces utilisateurs sont essentielles pour offrir une excellente expérience aux utilisateurs de l'application. La conception de l'interface utilisateur doit être intuitive, visuellement attrayante et facile à utiliser. Android fournit un ensemble riche d'éléments d'interface utilisateur appelés vues et widgets qui permettent aux développeurs de créer des interfaces utilisateur attrayantes.

2.1. View et Widgets

Dans Android, un View est un élément fondamental d'une interface utilisateur, représentant une zone rectangulaire sur l'écran et responsable du dessin et de la gestion des événements dans cette zone. Les View peuvent être simples, comme une TextView qui affiche du texte, ou plus complexes, comme une vue personnalisée qui combine plusieurs éléments différents. Les vues peuvent également être regroupées dans une hiérarchie de vues, ce qui permet d'obtenir des layouts d'interface utilisateur plus complexes.

Les widgets, quant à eux, sont un type spécifique de View qui fournit une interaction spécifique ou un élément d'affichage dans l'interface utilisateur, tel qu'un bouton ou un champ de saisie de texte. Ces éléments sont souvent interactifs et permettent à l'utilisateur de fournir des informations ou de lancer une action dans l'application. Les widgets peuvent être ajoutés aux layouts et personnalisés avec divers attributs et propriétés pour répondre aux besoins de l'application.

Par essence, les View sont les éléments de base d'une interface utilisateur, tandis que les widgets sont des éléments d'interface utilisateur spécifiques et préconstruits, conçus pour un objectif ou une fonction spécifique.

Attention

Il est important de faire la distinction entre ces deux types de widgets lorsque l'on parle de développement Android, afin d'éviter toute confusion.

Le terme "widget", dans le contexte du développement Android, fait spécifiquement référence aux éléments d'interface utilisateur qui peuvent être ajoutés à l'interface utilisateur d'une application, tels que les boutons, les champs de texte et les images. Ces widgets sont souvent utilisés pour permettre à l'utilisateur d'interagir avec l'application ou pour afficher des informations à l'utilisateur.

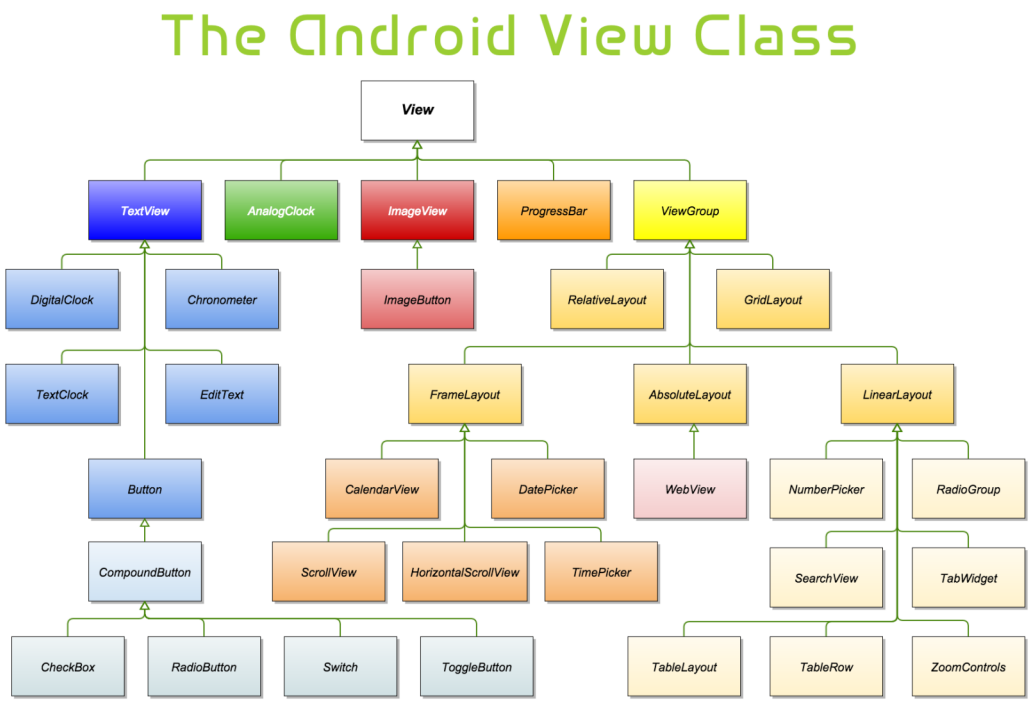
D'autre part, le terme "widget" peut également faire référence, de manière plus générale, à de petites applications qui offrent des fonctionnalités spécifiques sur l'écran d'accueil ou l'écran de verrouillage de l'utilisateur, telles que : widgets météo, widgets horloge, widgets lecteur de musique. Ces types de widgets sont différents des widgets UI utilisés dans le développement Android, car ils ne font généralement pas partie de l'interface d'une application.

2.2. La hiérarchie de la classe View

Dans Android, le "View" est un élément fondamental de l'interface utilisateur, et la classe "View" est la classe de base pour tous les composants de l'interface utilisateur. Les vues sont organisées dans une hiérarchie, dont la racine est la vue principale de l'application. L'ensemble de la hiérarchie des vues est géré par le cadre Android et est responsable du layout, du rendu et de la gestion des événements utilisateur.

La classe View est une classe abstraite qui hérite directement de la classe Java Object, et ses sous-classes sont utilisées pour créer divers éléments d'interface utilisateur, tels que des boutons, des champs de texte, des images, etc. La hiérarchie des classes de vues est très étendue, chaque sous-classe ajoutant une fonctionnalité unique à la classe de base. Par exemple, la sous-classe TextView permet d'afficher du texte et la sous-classe ImageView permet d'afficher des images.

La figure suivante illustre la hiérarchie des classes de "View" sous android



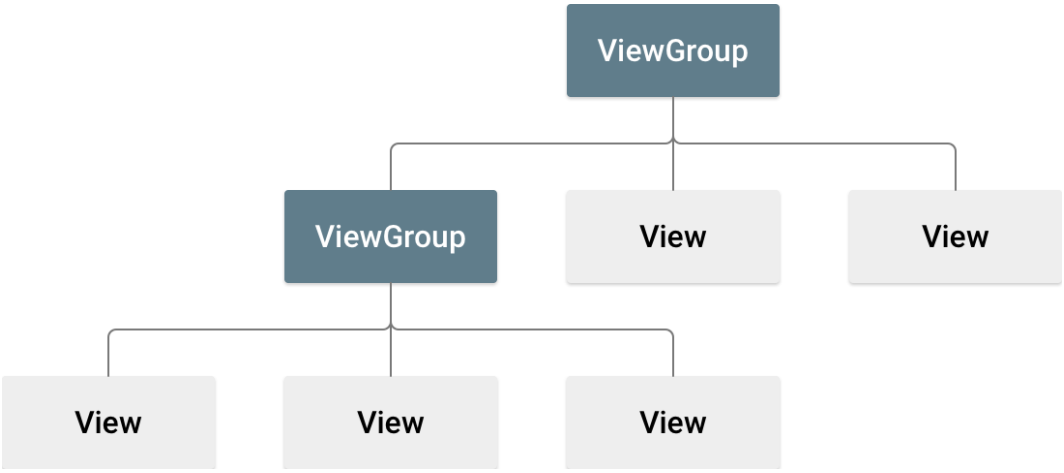
2.3. La hiérarchie des Views des interfaces Android

La hiérarchie des View dans une interface d'application Android représente la structure de toutes les View qui sont actuellement visibles sur l'écran d'une application. Il s'agit essentiellement d'une structure arborescente où chaque nœud représente une vue ou un groupe de vues, et chaque nœud enfant représente une vue ou un groupe de vues imbriqué.

La racine de la hiérarchie des View est toujours un groupe de View, qui peut contenir une ou plusieurs View ou groupes de View enfants. Les types de groupes de vues les plus couramment utilisés dans les layouts Android sont LinearLayout, RelativeLayout et ConstraintLayout.

Chaque View de la hiérarchie se voit attribuer un identifiant unique, qui est utilisé pour identifier la vue et manipuler ses propriétés de manière programmatique. Les vues peuvent également se voir attribuer une balise, qui est un objet facultatif pouvant être utilisé pour stocker des données supplémentaires sur la vue.

La hiérarchie des View peut être programmée ou créée à l'aide de fichiers de layout XML. Dans les deux cas, la hiérarchie est générée au moment de l'exécution, et le résultat est utilisé pour rendre l'interface utilisateur à l'écran.



Attention

Il est important de noter que la hiérarchie des View peut avoir un impact significatif sur les performances et la réactivité d'une application. Une hiérarchie des View profonde ou complexe peut entraîner des temps de layout et de rendu plus lents, ce qui peut donner à l'application une impression de lenteur ou de manque de réactivité de la part de l'utilisateur. C'est pourquoi il est généralement recommandé de conserver une hiérarchie des View aussi superficielle et simple que possible.

3. Les Layouts

Dans Android, un layout est un conteneur qui contient des éléments d'interface utilisateur tels que des widgets, des views et des layouts. Les layouts définissent la structure et l'apparence de l'interface utilisateur d'une application Android. Elles permettent aux développeurs de spécifier la manière dont les éléments de l'interface utilisateur sont positionnés, dimensionnés et alignés dans un espace donné.

Il existe plusieurs types de layout dans Android, chacun ayant ses propres caractéristiques et avantages. Il s'agit notamment de LinearLayout, RelativeLayout, FrameLayout, ConstraintLayout, etc. Chaque type de layout possède son propre ensemble de règles pour le positionnement et le dimensionnement des éléments de l'interface utilisateur, et le choix du bon layout pour une situation donnée peut avoir un impact significatif sur les performances et l'expérience utilisateur d'une application.

Outre ces types de layout de base, il existe également des layouts plus spécialisés, tels que TableLayout et GridLayout, qui sont utiles pour organiser les données dans un format tabulaire.

3.1. Linear Layout

LinearLayout est un type de layout dans Android qui permet aux développeurs de disposer les vues enfant verticalement ou horizontalement, en fonction de leur orientation. C'est l'un des layouts les plus couramment utilisés dans les applications Android, car il est simple et facile à utiliser. L'orientation du layout est définie à l'aide de l'attribut android:orientation dans XML ou de la méthode setOrientation() dans le code java.

La classe LinearLayout extends la classe ViewGroup, et peut donc contenir n'importe quelle vue en tant que vue enfant. Les vues enfant sont positionnées l'une après l'autre dans l'ordre dans lequel elles sont ajoutées au layout.

L'orientation par défaut est verticale, ce qui signifie que les vues enfant sont disposées de haut en bas. Si l'orientation est horizontale, les vues enfant sont disposées de gauche à droite.

L'un des avantages de l'utilisation de LinearLayout est qu'il est très facile de créer des layouts réactifs qui peuvent s'adapter à différentes tailles et résolutions d'écran. Par exemple, si l'orientation du layout est définie sur vertical, la hauteur de chaque view enfant peut être définie sur wrap_content, ce qui signifie que la view sera dimensionnée en fonction de son contenu. Cela permet à la layout de s'adapter automatiquement aux différentes tailles et résolutions d'écran.

Un autre avantage de l'utilisation de LinearLayout est qu'il peut être combiné avec d'autres layouts pour créer des layouts plus complexes. Par exemple, un LinearLayout peut être imbriqué dans un autre LinearLayout pour créer un layout bidimensionnel. Cela peut s'avérer utile pour créer des grilles de vues ou pour positionner des vues de manière plus complexe.

Voici un exemple de LinearLayout vertical en XML :

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click me!"/>

</LinearLayout>

```

Et voici un exemple de LinearLayout horizontal :

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"/>

</LinearLayout>

```

et enfin, vous pouvez créer un LinearLayout de manière programmatique dans le code Java comme suit :

```
// Create a new LinearLayout
LinearLayout linearLayout = new LinearLayout(context);

// Set the orientation of the LinearLayout
linearLayout.setOrientation(LinearLayout.VERTICAL);

// Create a new TextView
TextView textView = new TextView(context);
textView.setText("Hello, World!");

// Add the TextView to the LinearLayout
linearLayout.addView(textView);

// Create a new Button
Button button = new Button(context);
button.setText("Click me!");

// Add the Button to the LinearLayout
linearLayout.addView(button);

// Set the LinearLayout as the content view of the Activity
setContentView(linearLayout);

```

✎ *Remarque*

Il convient de noter que LinearLayout ne prend pas en charge l'utilisation de contraintes pour positionner les vues les unes par rapport aux autres. Pour cela, vous devrez utiliser un autre layout, tel que RelativeLayout ou ConstraintLayout. Cependant, LinearLayout reste un layout très utile pour de nombreux scénarios et est souvent utilisé en combinaison avec d'autres layouts pour créer des layouts plus complexes.

3.2. RelativeLayout

RelativeLayout est un autre type de layout dans Android qui permet aux développeurs de disposer les composants de l'interface utilisateur les uns par rapport aux autres. Contrairement à LinearLayout, RelativeLayout n'exige pas que les vues enfant soient disposées de manière linéaire. Au lieu de cela, chaque vue enfant est positionnée par rapport aux autres en fonction de sa relation avec la vue parentale ou avec d'autres vues enfant. RelativeLayout est donc un layout flexible et puissant qui permet de créer des interfaces utilisateur complexes.

Pour utiliser RelativeLayout, les développeurs peuvent définir la position de chaque vue enfant par rapport à la vue parent ou aux autres vues enfant à l'aide d'attributs tels que android:layout_above, android:layout_below, android:layout_toLeftOf et android:layout_toRightOf. Ces attributs permettent aux développeurs de spécifier la relation entre les View en termes de position relative, par exemple au-dessus, au-dessous, à gauche ou à droite d'une autre View.

L'un des avantages de RelativeLayout est qu'il permet de réduire le nombre de vues imbriquées nécessaires pour obtenir le layout souhaité. Avec LinearLayout, des vues imbriquées sont souvent nécessaires pour obtenir des layouts plus complexes. Cependant, avec RelativeLayout, les développeurs peuvent positionner les vues les unes par rapport aux autres sans avoir à recourir à des vues imbriquées.

Voici un exemple de RelativeLayout simple qui positionne deux TextViews l'un par rapport à l'autre :

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/textview1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello" />

    <TextView
        android:id="@+id/textview2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="World"
        android:layout_toRightOf="@id/textview1" />

</RelativeLayout>

```

le même exemple en code java

```
RelativeLayout relativeLayout = new RelativeLayout(context);
RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutParams(
        RelativeLayout.LayoutParams.MATCH_PARENT, RelativeLayout.LayoutParams.WRAP_CONTENT);
relativeLayout.setLayoutParams(layoutParams);

TextView textView1 = new TextView(context);
textView1.setId(View.generateViewId());
textView1.setText("Hello");

RelativeLayout.LayoutParams textView1Params = new RelativeLayout.LayoutParams(
        RelativeLayout.LayoutParams.WRAP_CONTENT, RelativeLayout.LayoutParams.WRAP_CONTENT);
textView1.setLayoutParams(textView1Params);

TextView textView2 = new TextView(context);
textView2.setId(View.generateViewId());
textView2.setText("World");

RelativeLayout.LayoutParams textView2Params = new RelativeLayout.LayoutParams(
        RelativeLayout.LayoutParams.WRAP_CONTENT, RelativeLayout.LayoutParams.WRAP_CONTENT);
textView2Params.addRule(RelativeLayout.RIGHT_OF, textView1.getId());
textView2.setLayoutParams(textView2Params);

relativeLayout.addView(textView1);
relativeLayout.addView(textView2);

// Add relativeLayout to a parent view
parentView.addView(relativeLayout);

```

✎ *Remarque*

Bien que RelativeLayout soit un outil puissant pour créer des layouts complexes, il nécessite une planification minutieuse et une prise en compte de la position de chaque view et de sa relation avec les autres views. S'il n'est pas utilisé correctement, RelativeLayout peut donner lieu à un layout encombré et difficile à maintenir.

Attention

lorsque vous utilisez le Relative Layout dans Android, sachez qu'il nécessite plus de codage manuel que les autres layouts. En effet, le Relative Layout positionne les éléments les uns par rapport aux autres, ce qui nécessite souvent de spécifier la position de chaque élément par rapport à ses voisins. En outre, l'utilisation de RelativeLayout nécessite souvent l'utilisation d'attributs XML et des ajustements manuels, plutôt que la simple opération de glisser-déposer fournie par d'autres types de layout comme ConstraintLayout. Bien que cela puisse sembler plus fastidieux, cela permet un contrôle plus précis du layout et du positionnement des vues, ce qui peut être particulièrement utile dans les conceptions d'interface utilisateur complexes.

3.3. Constraint Layout

ConstraintLayout est conçu pour s'adapter aux différentes tailles et orientations des appareils. C'est un gestionnaire de layout flexible qui permet aux développeurs de créer des interfaces utilisateur réactives capables de s'adapter à différentes tailles d'écran et à différents rapports d'aspect.

L'une des principales caractéristiques de ConstraintLayout est sa capacité à créer des conceptions réactives à l'aide de contraintes. Les contraintes définissent la position et la taille des vues par rapport à d'autres vues ou à la layout parent. Cela permet aux développeurs de créer des layouts qui s'adaptent à différentes tailles et orientations d'écran en ajustant les contraintes.

Par exemple, vous pouvez créer un layout avec un TextView centré horizontalement et verticalement sur l'écran, et un ImageView sous le TextView qui remplit le reste de l'écran. Pour ce faire, vous pouvez définir des contraintes entre les vues et le layout parent. Lorsque la taille ou l'orientation de l'écran change, le layout ajuste les contraintes afin de conserver la même position relative des vues.

Outre les contraintes, ConstraintLayout prend également en charge les lignes directrices, les chaînes et les barrières. Les lignes directrices sont des lignes invisibles qui permettent d'aligner les vues dans un layout. Les chaînes permettent de regrouper des vues et d'appliquer des contraintes communes au groupe. Les barrières sont utilisées pour créer une frontière virtuelle entre les vues.

Voici le code xml de l'exemple précédent :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3         xmlns:android="http://schemas.android.com/apk/res/android"
4         xmlns:app="http://schemas.android.com/apk/res-auto"
5         android:layout_width="match_parent"
6         android:layout_height="match_parent">
7
8     <TextView
9         android:id="@+id/textview1"
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:text="Hello World!"
13        app:layout_constraintBottom_toBottomOf="parent"
14        app:layout_constraintEnd_toEndOf="parent"
15        app:layout_constraintStart_toStartOf="parent"
16        app:layout_constraintTop_toTopOf="parent" />
17
18    <ImageView
19            android:id="@+id/imageview"
20            android:layout_width="0dp"
```

```
            android:layout_height="0dp"
            android:src="@drawable/image"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/textview1" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

4. Les Fragments

Les fragments sont un élément fondamental de l'interface utilisateur d'une application Android. Ils représentent une partie modulaire d'une interface utilisateur et peuvent être combinés entre eux pour créer un layout flexible et dynamique. Les fragments ont été introduits dans Android 3.0 (niveau API 11) pour relever le défi de la création d'interfaces utilisateur pouvant s'adapter à différentes tailles et résolutions d'écran. En divisant l'interface utilisateur en plus petits morceaux, les développeurs peuvent créer des layouts plus flexibles et réutilisables qui peuvent être facilement adaptés à différents appareils et facteurs de forme.

Par essence, un fragment est un composant d'interface utilisateur autonome doté de son propre cycle de vie, qui peut être ajouté ou supprimé du layout d'une activité au moment de l'exécution. Chaque fragment peut avoir son propre layout, son propre comportement et son propre ensemble de rappels, et peut communiquer avec d'autres fragments ou activités via une activité partagée ou via l'utilisation d'interfaces. Avec les fragments, les développeurs ont plus de contrôle sur le layout et le comportement de l'interface utilisateur de leur application, ce qui facilite la création d'interfaces utilisateur dynamiques et réactives.

Les fragments dans Android offrent plusieurs avantages, notamment

- Réutilisation : Les fragments peuvent être réutilisés dans plusieurs activités, ce qui permet un développement modulaire et efficace.
- Compatibilité : Les fragments sont compatibles avec un large éventail d'appareils et de tailles d'écran, ce qui facilite le développement d'applications fonctionnant sur plusieurs appareils.
- Code simplifié : Les fragments peuvent simplifier le code en permettant aux développeurs d'encapsuler le code d'éléments ou de fonctions spécifiques de l'interface utilisateur en un seul endroit, ce qui facilite la gestion et la maintenance.
- Amélioration des performances : L'utilisation de fragments peut améliorer les performances de votre application en réduisant la quantité de mémoire et de ressources nécessaires à l'affichage d'éléments d'interface utilisateur complexes.

Dans l'ensemble, les fragments constituent un moyen puissant et flexible de développer des applications dans Android, et leur utilisation peut conduire à un code plus efficace, modulaire et évolutif.

4.1. Cycle de vie d'un fragment

Les fragments dans Android ont un cycle de vie qui définit les différents états et événements par lesquels ils peuvent passer depuis leur création jusqu'à leur destruction. Il est important de comprendre le cycle de vie des fragments pour gérer le comportement de l'interface utilisateur et des données de votre application.

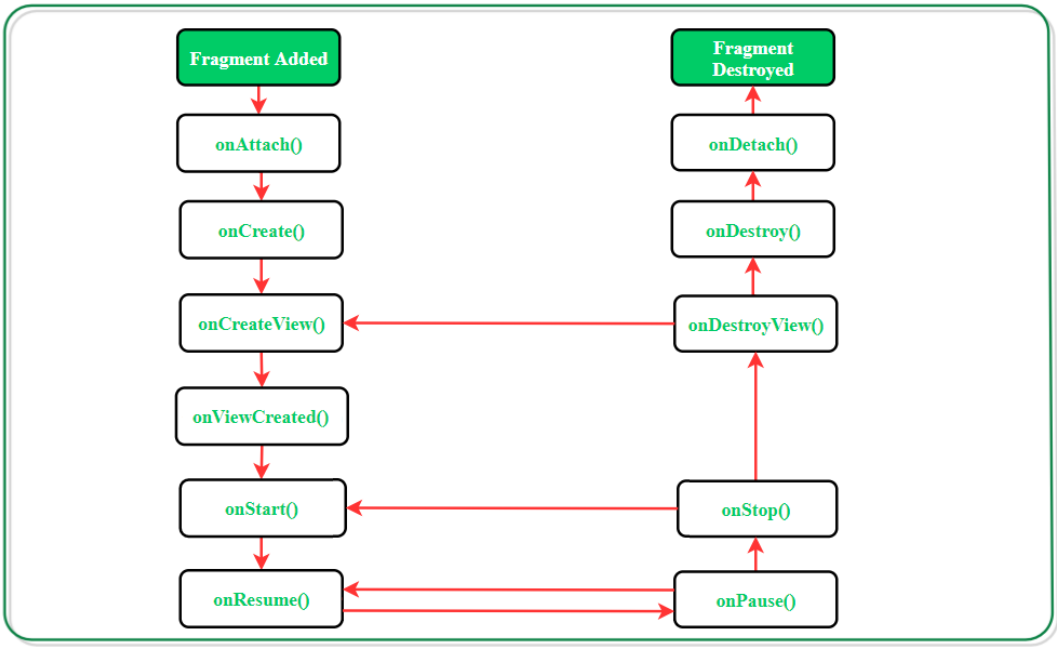
Le cycle de vie des fragments comprend les états suivants :

- Actif : Le fragment est visible et interactif.
- En pause : Le fragment est toujours visible mais n'est pas interactif. Cet état se produit lorsqu'une autre activité ou un autre fragment se trouve au-dessus du fragment actuel.
- Stoppé : Le fragment n'est plus visible. Cet état se produit lorsque l'activité d'hébergement est arrêtée.
- Détruit : Le fragment n'est plus en mémoire. Cet état survient lorsque l'activité d'hébergement est détruite.

Il existe plusieurs méthodes correspondant à chacun de ces états du cycle de vie. Voici quelques-unes des méthodes les plus importantes :

1. onAttach() : Appelée lorsque le fragment est associé à son activité d'hébergement.
2. onCreate() : Appelée lorsque le fragment est créé.
3. onCreateView() : Appelée lors de la création de l'interface utilisateur du fragment.
4. onViewCreated() : Appelée après le retour de onCreateView() et le gonflement de l'interface utilisateur du fragment.
5. onActivityCreated() : Appelé lorsque la méthode onCreate() de l'activité d'hébergement est terminée.
6. onStart() : Appelé lorsque le fragment est visible par l'utilisateur.
7. onResume() : Appelé lorsque le fragment est visible et interactif.
8. onPause() : Appelé lorsque le fragment est encore visible mais non interactif.
9. onStop() : Appelé lorsque le fragment n'est plus visible.
10. onDestroyView() : Appelé lorsque la vue du fragment est en cours de destruction.
11. onDestroy() : Appelé lors de la destruction du fragment.
12. onDetach() : Appelée lorsque le fragment n'est plus associé à son activité d'hébergement.

En implémentant ces méthodes dans votre fragment, vous pouvez gérer le comportement du fragment tout au long de son cycle de vie et vous assurer que l'interface utilisateur et les données de votre application sont correctement gérées.



4.2. Comment utiliser le fragment

Afin de créer un nouveau fragment dans votre projet Android, vous devez disposer de trois éléments: la classe Java du fragment, qui représente le contrôleur du fragment et implémente la méthode onCreateView() pour gonfler la mise

en page du fragment, le fichier de mise en page XML pour le fragment et l'activité à laquelle vous ajouterez le fragment. La classe Java du fragment étend la classe Fragment, tandis que le fichier de mise en page XML définit la disposition des vues dans le fragment. Ensuite, vous devez ajouter le fragment à l'activité en utilisant un FragmentManager pour commencer une FragmentTransaction et en appelant la méthode add() pour ajouter le fragment à un conteneur de mise en page (FrameLayout) dans l'activité.

Pour créer un nouveau fragment dans votre application Android, vous devez suivre les étapes suivantes :

1. Dans Android Studio, ouvrez le projet où vous souhaitez ajouter le nouveau fragment.
2. Dans le volet Project, cliquez avec le bouton droit de la souris sur le dossier où vous souhaitez ajouter le nouveau fragment.
3. Dans le menu contextuel, sélectionnez New > Fragment > Fragment (Blank).
4. Dans la boîte de dialogue qui apparaît, saisissez un nom pour votre nouveau fragment, ainsi que les autres options souhaitées (par exemple, l'emplacement où vous souhaitez créer le fragment).
5. Cliquez sur Finish pour créer le nouveau fragment et son layout associé.

Cela créera automatiquement une nouvelle classe de fragment, ainsi qu'un fichier de layout pour le fragment, et les ajoutera tous les deux à votre projet. Vous pouvez ensuite personnaliser le layout et le code de votre fragment pour répondre à vos besoins spécifiques.

Après la création d'un fragment dans Android Studio, voici les étapes pour l'ajouter à une activité :

- Dans l'éditeur de code d'Android Studio, ouvrez la classe de l'activité dans laquelle vous souhaitez ajouter le fragment.
- Repérez l'emplacement dans le layout de l'activité où vous souhaitez ajouter le fragment. Ajoutez un conteneur de type FrameLayout qui servira de réceptacle pour votre fragment.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

   <!-- Autres vues de l'activité -->

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/autre_vue"
        android:layout_alignParentBottom="true" />

</RelativeLayout>

```

- Dans la méthode onCreate() de l'activité, instanciez le fragment que vous avez créé et ajoutez-le au conteneur que vous venez de créer à l'étape précédente. Vous pouvez le faire en utilisant un FragmentManager et un FragmentTransaction.

Voici un exemple de code pour ajouter le fragment :

```
// Instantiate the fragment
MyFragment fragment = new MyFragment();

// Start the fragment transaction
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
```

```
6
7 // Add the fragment to the container
8 transaction.add(R.id.container, fragment);
9
10 // Commit the transaction
11 transaction.commit();
```

- Compilez et exécutez votre application pour voir votre fragment s'afficher dans l'activité.

Le code présenté précédemment n'est qu'un exemple de base, et il existe de nombreuses autres fonctionnalités et options disponibles pour travailler avec des fragments dans Android.

Voici quelques options et fonctionnalités supplémentaires liées aux fragments dans Android :

- Transactions de fragments : Outre l'ajout, la suppression et le remplacement de fragments, vous pouvez également effectuer d'autres opérations sur les fragments à l'aide des transactions de fragments. Ces opérations comprennent l'attachement et le détachement de fragments et la définition d'animations personnalisées pour les transitions de fragments.
- Arguments de fragment : Les arguments de fragment vous permettent de transmettre des données entre les fragments et entre un fragment et son activité parente. Vous pouvez transmettre des arguments en tant qu'objet Bundle à l'aide de la méthode setArguments() et les récupérer dans le fragment à l'aide de la méthode getArguments().
- Communication entre fragments : Les fragments peuvent communiquer avec leur activité parente et avec d'autres fragments à l'aide d'interfaces. En définissant une interface dans un fragment et en la mettant en œuvre dans l'activité ou dans un autre fragment, vous pouvez envoyer et recevoir des données et des événements entre les fragments.
- Layouts des fragments : Les fragments peuvent avoir leurs propres layouts, qui peuvent être créés à l'aide de XML ou par programme. Vous pouvez utiliser un fichier de layout pour définir les éléments de l'interface utilisateur et leurs propriétés dans le fragment, et déployer le layout dans la méthode onCreateView().
- Restauration de l'état du fragment : Lorsqu'un fragment est détruit et recréé à la suite d'un changement de configuration, tel qu'une rotation de l'écran, vous pouvez enregistrer et restaurer son état à l'aide des méthodes onSaveInstanceState() et onViewStateRestored(). Cela vous permet de préserver les données du fragment et l'état de l'interface utilisateur lors des changements de configuration.
- Transactions de fragments avec la pile arrière : Vous pouvez ajouter des fragments à la pile arrière et naviguer entre eux à l'aide du bouton "back". Cela permet d'offrir aux utilisateurs une expérience de navigation plus intuitive.
- Personnalisation des fragments : Les fragments peuvent être personnalisés avec différents styles et thèmes pour correspondre à l'aspect général de l'application. Vous pouvez également personnaliser le comportement des fragments en étendant la classe de base Fragment et en ajoutant des fonctionnalités personnalisées.

5. Les Widgets

Dans le développement d'applications Android, les widgets sont des composants de l'interface utilisateur graphique (GUI) qui sont utilisés pour afficher des données et interagir avec elles. Les widgets se présentent sous de nombreuses formes et peuvent aller du plus simple au plus complexe en fonction de leur fonctionnalité et de leur complexité.

5.1. Widgets de base

Les widgets de base sont les composants d'interface utilisateur les plus courants dans les applications Android. Ils comprennent des vues telles que TextView, EditText, Button, ImageView, ProgressBar, CheckBox et RadioButton, qui permettent aux utilisateurs d'interagir avec l'application et d'afficher du contenu.

TextView

Le widget TextView est utilisé pour afficher du texte à l'écran. Il peut être personnalisé avec des propriétés telles que la taille de police, la couleur de texte et l'alignement. Voici un exemple de TextView avec du texte en gras et centré:

```
<TextView
    android:id="@+id/textview_hello"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bonjour le monde!"
    android:textStyle="bold"
    android:textAlignment="center"/>

```

EditText

Le widget EditText permet à l'utilisateur d'entrer du texte. Il peut être utilisé pour saisir du texte ou des nombres dans l'application. Voici un exemple de EditText avec un texte d'aide et une limite de longueur de 10 caractères:

```
<EditText
    android:id="@+id/edittext_input"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Entrez un texte ici"
    android:maxLength="10"/>

```

Button

Le widget Button est utilisé pour déclencher des actions dans l'application lorsqu'il est cliqué. Il peut être personnalisé avec des propriétés telles que le texte, la couleur de fond et la taille. Voici un exemple de Button avec du texte en français:

```
<Button
    android:id="@+id/button_submit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Envoyer"/>

```

ImageView

Le widget ImageView est utilisé pour afficher des images dans l'application. Il peut être personnalisé avec des propriétés telles que la source de l'image, la taille et l'échelle. Voici un exemple de ImageView avec une image d'un chat

```
<ImageView
    android:id="@+id/imageview_cat"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/cat"/>

```

ProgressBar

Le widget ProgressBar est utilisé pour indiquer l'état de progression d'une tâche en cours. Il peut être personnalisé avec des propriétés telles que la couleur de la barre de progression et le style. Voici un exemple de ProgressBar avec un style en cercle

```
<ProgressBar
    android:id="@+id/progressbar_loading"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:indeterminate="true"
    style="?android:attr/progressBarStyleLarge"/>

```

CheckBox

Le widget CheckBox est utilisé pour permettre à l'utilisateur de sélectionner une ou plusieurs options à la fois. Il peut être personnalisé avec des propriétés telles que le texte d'option et l'état sélectionné. Voici un exemple de CheckBox avec trois options

```
<CheckBox
    android:id="@+id/checkbox_options"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Options"
    android:checked="true"/>

<CheckBox
    android:id="@+id/checkbox_option1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 1"/>

<CheckBox
    android:id="@+id/checkbox_option2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 2"/>

```

RadioButton

Le widget RadioButton est utilisé pour permettre à l'utilisateur de sélectionner une option à la fois parmi un ensemble d'options. Il peut être personnalisé avec des propriétés telles que le texte d'option et l'état sélectionné. Voici un exemple de RadioButton avec trois options

```
<RadioGroup
    android:id="@+id/radiogroup_options"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <RadioButton
        android:id="@+id/radiobutton_option1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Option 1"/>

    <RadioButton
        android:id="@+id/radiobutton_option2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Option 2"/>

    <RadioButton
        android:id="@+id/radiobutton_option3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Option 3"/>
</RadioGroup>

```

5.2. Widgets avancée

Les widgets avancés d'Android offrent des fonctionnalités plus avancées et spécialisées pour créer des interfaces utilisateur plus complexes et interactives. Ces widgets sont plus flexibles et personnalisables que les widgets de base, ce qui permet aux développeurs de concevoir des applications plus sophistiquées et esthétique

RecyclerView

Le widget RecyclerView est utilisé pour afficher une liste d'éléments en mode scrollable, avec une grande flexibilité pour personnaliser l'affichage des éléments individuels. Il nécessite l'utilisation d'un adaptateur pour fournir les données et définir la vue de chaque élément. Voici un exemple de RecyclerView avec une liste de noms

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerview_names"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>

// Définir l'adaptateur dans le code Java ou Kotlin
recyclerView.adapter = NamesAdapter(namesList)

```

DatePicker

Le widget DatePicker est utilisé pour permettre à l'utilisateur de sélectionner une date à partir d'un calendrier. Il peut être personnalisé avec des propriétés telles que la date initiale et les limites de date sélectionnables. Voici un exemple de DatePicker avec une date initiale

```
<DatePicker
    android:id="@+id/datepicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false"
    android:spinnersShown="true"
```

```
    android:minDate="2022-01-01"
    android:maxDate="2023-12-31"
    android:layout_margin="16dp"/>

```

WebView

Le widget WebView est utilisé pour afficher du contenu Web dans une vue intégrée dans l'application. Il permet d'afficher des pages Web complètes, des vidéos, des images et des scripts en utilisant les mêmes fonctionnalités qu'un navigateur Web standard. Voici un exemple de WebView qui affiche une page Web

```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>

// Charger la page Web dans le code Java ou Kotlin
webView.loadUrl("https://www.example.com")

```

SeekBar

Le widget SeekBar est utilisé pour permettre à l'utilisateur de sélectionner une valeur numérique dans une plage donnée en faisant glisser un curseur. Il peut être personnalisé avec des propriétés telles que la plage de valeurs et la valeur initiale. Voici un exemple de SeekBar avec une plage de valeurs de 0 à 100

```
<SeekBar
    android:id="@+id/seekbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="50"/>

```

TimePicker

Le widget TimePicker est utilisé pour permettre à l'utilisateur de sélectionner une heure à partir d'une horloge numérique. Il permet de personnaliser des propriétés telles que l'heure initiale, le format de l'heure et la plage de valeurs pour l'utilisateur. Voici un exemple de TimePicker avec une heure initiale de 12:00 et un format d'heure en 24 heures

```
<TimePicker
    android:id="@+id/timepicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:timePickerMode="spinner"
    android:format24Hour="true"
    android:hour="12"
    android:minute="0"/>

```

RatingBar

Le widget RatingBar est utilisé pour permettre à l'utilisateur de sélectionner une note à partir d'une barre d'évaluation. Il peut être personnalisé avec des propriétés telles que le nombre d'étoiles, la note initiale et la plage de valeurs. Voici un exemple de RatingBar avec 5 étoiles et une note initiale de 3

```
<RatingBar
    android:id="@+id/ratingbar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:rating="3"
    android:stepSize="1.0"/>

```

5.3. Les gestionnaires d'événements

En Android ou dans tout autre système, les applications ou programmes ont besoin d'un moyen d'interagir avec l'utilisateur, c'est là que les gestionnaires d'événements entrent en jeu. Les gestionnaires d'événements permettent au système d'écouter les actions de l'utilisateur et de répondre en conséquence, ce qui permet à l'utilisateur d'interagir avec l'application ou le programme de manière significative.

En Android spécifiquement, les gestionnaires d'événements sont couramment utilisés avec des widgets tels que les boutons, les vues de texte et les barres de progression pour fournir des fonctionnalités telles que la réponse aux clics, aux appuis longs, aux changements d'état et autres interactions de l'utilisateur. En utilisant efficacement les gestionnaires d'événements, les développeurs peuvent créer des interfaces utilisateur engageantes et réactives qui offrent une expérience utilisateur positive.

Dans Android, les gestionnaires d'événements sont mis en œuvre comme des méthodes de call-back, ce qui signifie qu'ils sont appelés par le système Android en réponse à un événement spécifique. Pour utiliser un gestionnaire d'événements dans votre application Android, vous devez définir une méthode qui correspond à la signature du gestionnaire d'événements que vous souhaitez utiliser, puis enregistrer la méthode avec la source d'événements appropriée.

Par exemple, pour gérer un événement de clic sur un bouton, vous pouvez définir une méthode avec la signature suivante

```
public void onClick(View view) {
    // Handle button click event here
}

```

Vous pouvez ensuite enregistrer cette méthode en tant que gestionnaire d'événements pour un bouton en appelant la méthode setOnClickListener() sur le bouton et en transmettant la référence de la méthode

```
Button button = findViewById(R.id.my_button);
button.setOnClickListener(this::onClick);

```

Voici quelques exemples des gestionnaires d'événements les plus utilisés dans Android pour les widgets

- ***OnClickListener*** : utilisé pour gérer l'événement de clic d'une View (bouton, ImageView, etc.)

```
// OnClick listener for a button
```

```
Button myButton = findViewById(R.id.my_button);
myButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Handle button click event
    }
});

```

- *OnLongClickListener* : utilisé pour gérer l'événement de clic long d'une View

```
// OnLongClick listener for a button
Button myButton = findViewById(R.id.my_button);
myButton.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        // Handle long click event
        return true; // return true to consume the event
    }
});

```

- *TextChangedListener*: utilisé pour surveiller les changements de texte dans un EditText

```
// TextWatcher for an EditText
EditText myEditText = findViewById(R.id.my_edit_text);
myEditText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        // Called before text changed
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        // Called during text changed
    }

    @Override
    public void afterTextChanged(Editable s) {
        // Called after text changed
    }
});

```

- *OnItemSelectedListener* : utilisé pour gérer les événements de sélection d'un Spinner

```
// OnItemSelected listener for a Spinner
Spinner mySpinner = findViewById(R.id.my_spinner);
mySpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        // Handle item selection event
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        // Handle no selection event
    }
});

```

- ***OnCheckedChangeListener*** : est une méthode qui permet de définir un récepteur pour les changements d'état d'une case à cocher ou d'un bouton radio dans une View.

```
checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        // Code to handle checkbox state change
    }
});

```

Le tableau suivant contient une liste des gestionnaires d'événements courants utilisés dans le développement d'Android, ainsi que leur description et des exemples d'utilisation.

Gestionnaire d'événements	Description	Usage
onTouch	Gère les événements de toucher pour une vue	view.setOnTouchListener(new View.OnTouchListener() {...});
onKey	Gère les événements de clavier pour une vue	view.setOnKeyListener(new View.OnKeyListener() {...});
onScrollChanged	Gère les événements de défilement pour une vue ScrollView ou ListView	view.setOnScrollChangeListener(new View.OnScrollChangeListener() {...});
onFocusChange	Gère les événements de changement de focus pour une vue	view.setOnFocusChangeListener(new View.OnFocusChangeListener() {...});
onEditorAction	Gère les événements de l'action IME pour une vue	view.setOnEditorActionListener(new TextView.OnEditorActionListener() {...});
onMenuClick	Gère les événements de clic pour un élément de menu dans un menu	menuItem.setOnMenuItemClickListener(new MenuItem.OnMenuItemClickListener() {...});
onItemClick	Gère les événements de clic pour les éléments dans une vue ListView ou GridView	listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {...});
onItemLongClick	Gère les événements de clic long pour les éléments dans une vue ListView ou GridView	listView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {...});
onDateSet	Gère les événements de sélection de date dans une boîte de dialogue DatePickerDialog	datePickerDialog.setOnDateSetListener(new DatePickerDialog.OnDateSetListener() {...});

onTimeSet	Gère les événements de sélection de temps dans une boîte de dialogue TimePickerDialog	timePickerDialog.setOnTimeSetListener(new TimePickerDialog.OnTimeSetListener() {...});
onProgressChanged	Gère les événements de changement de progression pour une vue SeekBar	seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {...});
onRatingChanged	Gère les événements de changement de note pour une vue RatingBar	ratingBar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {...});
onSwipe	Gère les événements de swipe pour une vue	view.setOnTouchListener(new OnSwipeTouchListener(context) {...});
onPinch	Gère les événements de pinch pour une vue	view.setOnTouchListener(new OnPinchTouchListener(context) {...});
onDoubleTap	Gère les événements de double clic pour une vue	view.setOnTouchListener(new GestureDetector.OnDoubleTapListener() {...});
onGesturePerformed	Gère les événements de gestes pour une vue	gestureOverlayView.addOnGesturePerformedListener(new GestureOverlayView.OnGesturePerformedListener() {...});