TP: Optimisation des performances de UI

Expérience Utilisateur

Utilisabilité

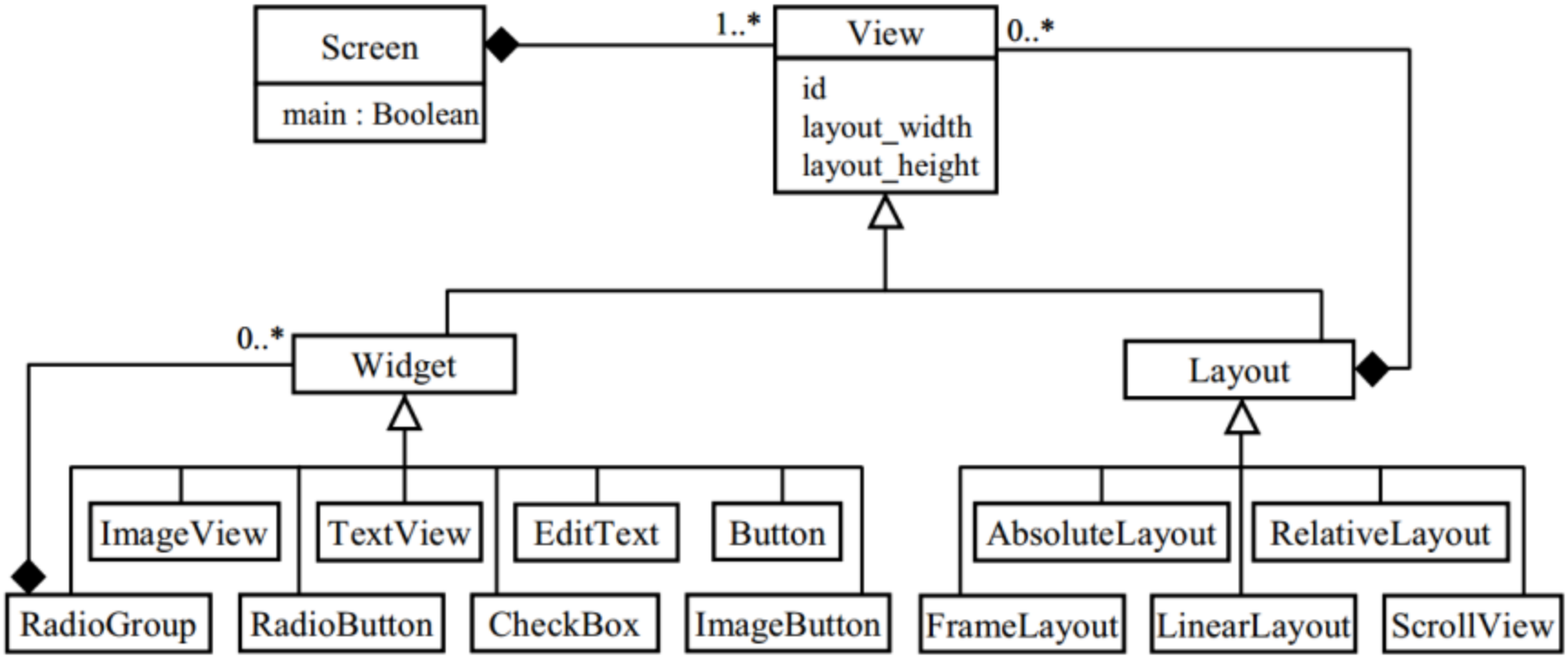
Test utilisateur

Technologie et inclusivité

Android et les Design Patterns

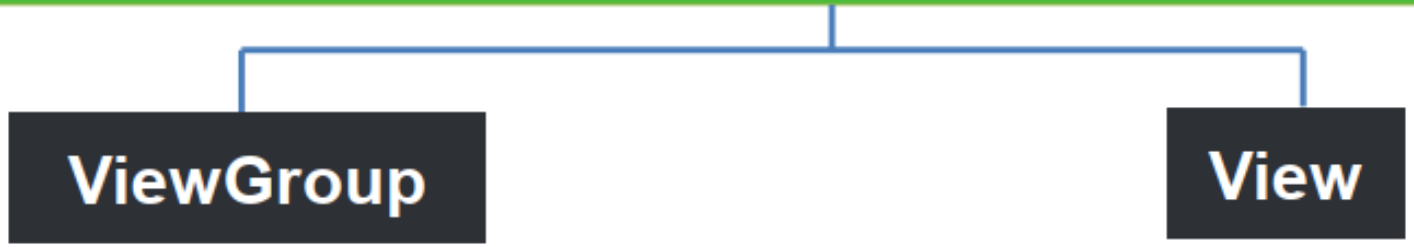
Optimisation des performances de UI

Un extrait du métamodèle de l'application Android



Android: Views & Layout

Composants de **l'interface utilisateur (UI)** d'une activité



- **Conteneur** de View.
- Responsable du **placement** d'autres View sur l'écran
- Chaque layout doit étendre un **ViewGroup**

- **Composant de base (UI)**
- Peut gérer/produire des **événements**
- Nouveau composant: extension de **View**

Fichier XML ayant LinearLayout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="This is a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="This is a Button" />

   <!-- More GUI components go here  -->

</LinearLayout>
```

Views

View Group

Une fois votre mise en page (**layout**) est créée, vous pouvez charger **ressource** de mise en page (**layout**) à partir du code de votre application, dans votre implémentation « **Activity.onCreate()** »

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

View & Events

Les vues/Widgets sont des composants **interactifs** → Lors de certaines actions de l'utilisateur, un événement approprié sera déclenché

clic, clic long, focus, éléments sélectionnés, éléments cochés, glisser,…

Comment gérer les événements générés par une View ?

1. Directement depuis **XML**
2. Via **Event Listeners** (général, recommandé)

View and Events

Pour un ensemble limité de composants, il est possible de gérer les événements via des **rappels** indiqués dans la mise en page XML.

```
<Button
android:text="@string/textButton”
android:id="@+id/idButton”
android:onClick=”doSomething”
/>
```

XML Layout File

Java code

```
public void doSomething(View w) {
   // Code to manage the click
event
}
```

View & Events

Les vues/Widgets sont des composants **interactifs** → Lors de certaines actions de l'utilisateur, un événement approprié sera déclenché

clic, clic long, focus, éléments sélectionnés, éléments cochés, glisser,…

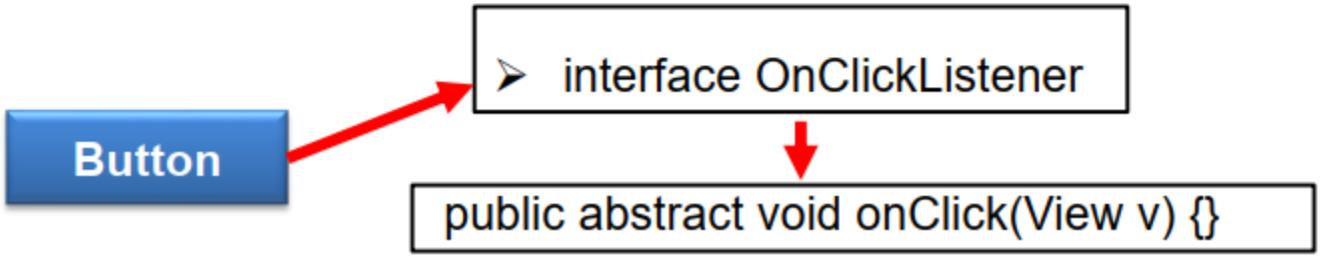
Comment gérer les événements générés par une View ?

1. Directement depuis **XML**
2. Via **Event Listeners** (général, recommandé)

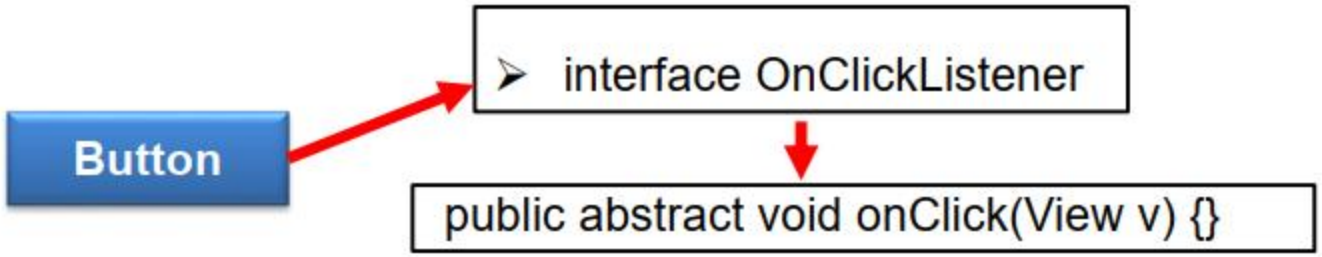
View & Events

Chaque View contient une collection d'interfaces imbriquées (listeners).

- Chaque listener gère un seul **event** …
- Chaque listener contient une seule méthode de rappel…
- Le rappel est invoqué lors de l'apparition de l'événement.



View & Events




```
…
Button button=(Button)findViewById(R.id.buttonNext);
button.performClick();
…
```

```
// Callback method
public void onClick(View v) {
        ….
}
```

View & Events: ActionListener

LISTE DES INTERFACES ACTIONLISTENER

- interface OnClickListener
  abstract method: *onClick()*
- interface OnLongClickListener
  abstract method: *onLongClick()*
- interface OnFocusChangeListener
  abstract method: *onFocusChange()*
- interface OnKeyListener
  abstract method: *onKey()*
- interface OnCheckedChangeListener
  abstract method: *onCheckedChanged()*

- interface OnItemSelectedListener
  abstract method: *onItemSelected()*
- interface OnCheckedChangeListener
  abstract method: *onCheckedChanged()*
- interface OnItemSelectedListener
  abstract method: *onItemSelected()*
- interface OnTouchListener
  abstract method: *onTouch()*
- interface OnCreateContextMenuListener
  abstract method: *onCreateContextMenu()*

Android: ViewGroups

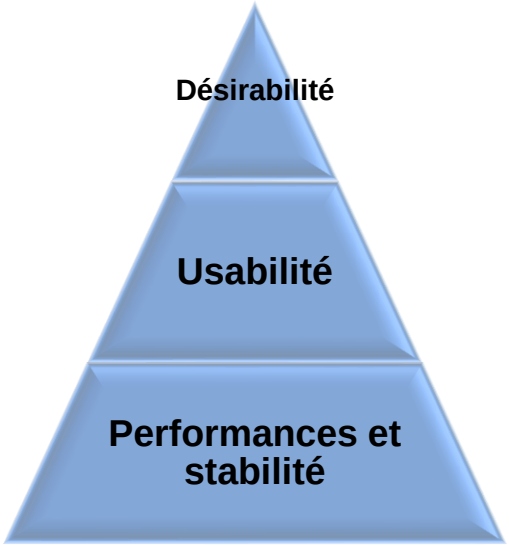
Liste des **Layouts** les plus courantes fournies par Android

Nom	XML Tag	Description
LinearLayout	<LinearLayout> </LinearLayout>	arrange Views by aligning them on a single row or column
RelativeLayout	<RelativeLayout> </RelativeLayout>	arrange Views through relative positions
TableLayout	<TableLayout> </TableLayout>	arrange Views into rows and columns
FrameLayout	<FrameLayout> </FrameLayout>	arrange a single View within a Layout
AbsoluteLayout	<AbsoluteLayout> </AbsoluteLayout>	arrange Views through absolute positions
ConstraintLayout	<ConstraintLayout> </ConstraintLayout>	arrange views through constraints in ConstraintLayout:

- Un Layout peut être déclaré dans un autre layout

Améliorer les performances de l'interface utilisateur

- Vous êtes responsable de l'UX.
- **La performance** est le fondement de l'UX sur mobile



Pour la meilleure expérience utilisateur :

1. Utiliser un langage natif
2. Créer une API spécifique aux mobiles
3. Penser bien au chargement

Améliorer les performances de l'interface utilisateur

- **Durant 16 millisecondes, Android rafraîchit l'écran du mobile**
  - ❑ Ce rafraîchissement dépend des événements
    - ➢ **par ex**. la suppression de composant, Visible/Invisible

⇒ Changer UI

- ❑ Implémentation de l'algorithme de rafraîchissement: C,C++, autres langues: Rust

Améliorer les performances de l'interface utilisateur

- **Objectif** : Maintenir 60 FPS pour une expérience utilisateur fluide.
- **Cadence de Rendu :** Chaque frame doit être rendue en 16 ms ou moins

Frames : cette section présente le thread UI et les événements de trace

RenderThread dans votre application.

Risque: Les événements **de plus de 16 ms** sont colorés **en rouge**, car ils dépassent le délai d'affichage de 60 frames par seconde (FPS).

Outils de Profilage pour Visualiser les Frames

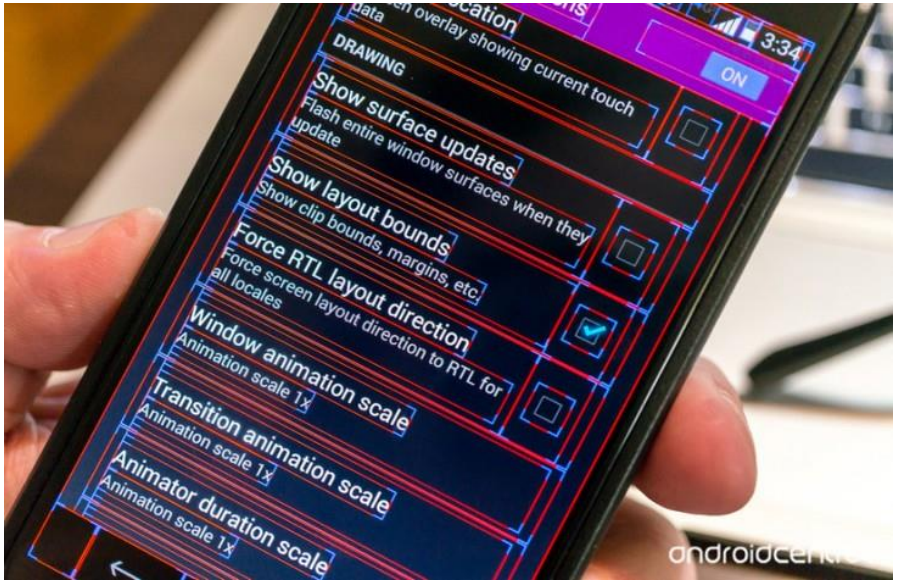
- **Profiler de Performance d'Android Studio**
  - Visualise les threads UI et RenderThread.
  - **Code couleur:** Événements > 16 ms en rouge.
- **Systrace**
  - Capture les traces système et application.
  - Met en évidence les événements de longue durée.
- **GPU Profiler**
  - Visualise la charge GPU pour chaque frame.
  - Repère les frames qui dépassent la limite de 16 ms.

Améliorer les performances de l'interface utilisateur

Option de développeurs:

Settings > Developer options. **Advanced**.

- Show **surface updates**
- Show **Layout Bounds (Afficher les contours)**



Améliorer les performances de l'interface utilisateur

Pour améliorer les performances → Diminuer les trois opérations:

- Re-calculate « Espace »
- Re-layout « Positionnement»
- Re-drawn « Redessiner »

Améliorer les performances de l'interface utilisateur

- Eviter « Nesting Linear Layout »
- Utiliser ContraintLayout au lieu de LinearLayout
- Varier les appareils mobiles
  - Consulter « Firebase Monitoring Service (SAS) »
- Utiliser “Multi threading system”
  - Main Threading (3 Op), Utiliser “Background thread”
    - **Network on main thread Exception:** The error is due to executing long running operations in main thread
- Utiliser « Explicit Invalidate »
- Suppression de l'arrière-plan inutile dans les layoutes
- Réduire la transparence
- Voir l’animation
- ...

Gestion des Threads dans Android : Main Thread, Background Thread et Exécution Parallèle

- **Main Thread (Thread Principal) :** Le thread principal gère les opérations de l'interface utilisateur (UI). Toute manipulation visuelle (comme modifier du texte ou la couleur d'un bouton) doit s'y faire. Des opérations lourdes (calculs complexes, accès réseau) sur ce thread peuvent bloquer l'UI et réduire la réactivité de l'application.

- **Background Thread :** Thread secondaire pour les tâches sans interaction directe avec l'UI. Utiliser un "Background Thread" permet de décharger le thread principal, en exécutant des tâches intensives (accès aux données, requêtes réseau, traitement d'images) en arrière-plan, évitant ainsi de ralentir l'UI.

- **Exécuter plusieurs threads en parallèle (3 Opérations) :** Pour lancer trois opérations en parallèle, Android propose des outils comme les coroutines **Kotlin** (Dispatchers.IO pour I/O, Dispatchers.Default pour les calculs) ou les classes AsyncTask et ExecutorService.

Améliorer les performances de l'interface utilisateur

- LinearLayaut **vs** ConstraintLayaut
- Rotation ??
- Utiliser d’autres optimisation
- Utiliser « Android monitoring Tool »

  pour le test de performance

Thank you !

Questions ? abdelkader.ouared@univ-tiaret.dz

www. .univ-tiaret.dz