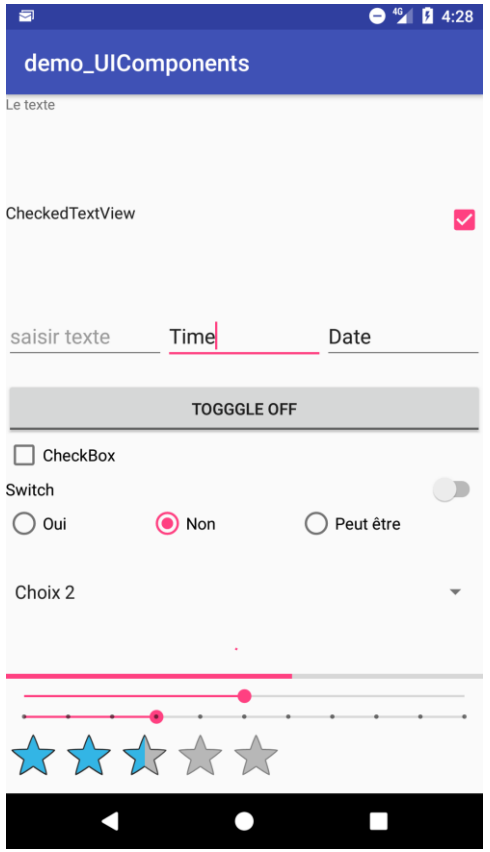
CAI-IP02 SMB116

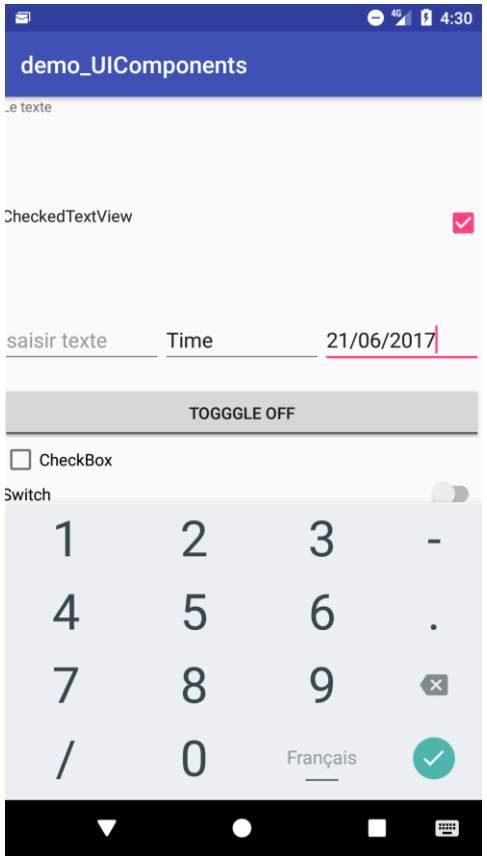
Les **Composants graphiques**

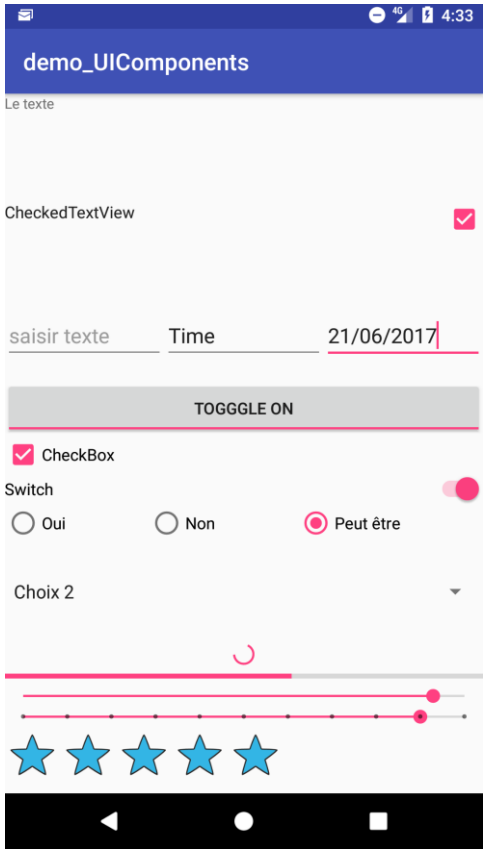
Les composants graphiques

- **Composants** : éléments ajoutés à un **ViewGroup** pour constituer une interface graphique.
- Tous les composants graphiques sont des **android.view.View**
- Un composant occupe une partie de l'écran il est chargé de **se dessiner** et de **gérer ses événements**
- Un composant comporte des **propriétés** déterminant son **apparence** et son **comportement**
- Les utilisateurs des composants peuvent leur affecter les écouteurs (**Listener**) d'évènement pour **être notifié lorsque certains événements se produisent**

Les composants graphiques







TextView

- Composant utilisé pour **afficher un texte**.
- Le texte : **littéral** ou bien référence vers une **chaine d'un fichier de ressource**
- L'apparence du Textview est configurable: Police, Style, etc.
- La propriété Text d'un TextView est du type **CharSequence**

```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:fontFamily="sans-serif"
    android:text="Un TextView"
    android:textSize="24sp" />
```

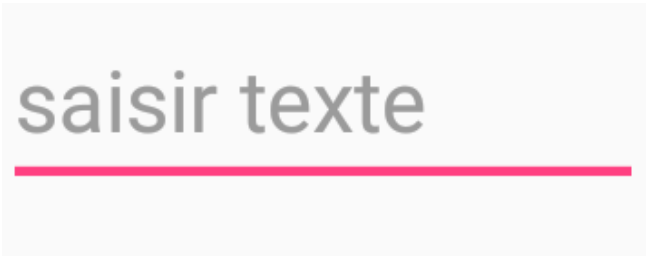
```
// Le texte d'un TextView est une CharSequence
TextView textView = findViewById(R.id.textView);
textView.setText("Le texte");
CharSequence content = textView.getText();
String strContent = textView.getText().toString();
```

- Le **CheckedTextView** est une extension du **TextView** qui réalise l'interface **android.widget.Checkable :**

EditText (1)

- Composant utilisé pour **saisir et modifier du texte**.
- Un **EditText** est un **TextView** éditable (hérite de **TextView)**.
- L'attribut **`android:hint`** est une consigne affichée dans l'EditText quand aucune valeur n'est renseignée.
- L'attribut **`android:inputType`** multivalué décrit le type de donnée acceptée par l'EditText (adaptation du clavier virtuel)
https://developer.android.com/reference/android/R.styleable.html#TextView_inputType

```
<EditText
    android:id="@+id/editText"
    android:hint="saisir texte"
    android:inputType="textPersonName|number" />
```



EditText (2)

- Pour surveiller la saisie un **TextChangedListener** peut être attaché à un EditText
- Le listener doit réaliser l'interface **`android.text.TextWatcher`**

```
editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        //...
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        //...
    }
    @Override
    public void afterTextChanged(Editable s) {
        //...
    }
});
```

ToggleButton

- Bouton à **2 états** : **Coché** / **Non coché** (hérite de **Button**)
- Le bouton est représenté graphiquement « enfoncé » ou « levé »
- Les propriétés **`android:textOn`** et **`android:textOff`** permettent d'associer un texte à chacun de ces deux états (Possibilité d'utiliser des Drawable)

```
<ToggleButton
    android:id="@+id/toggleButton"
    android:checked="false"
    android:text="ToggleButton"
    android:textOff="Togggle OFF"
    android:textOn="Togggle ON"
/>
```

CheckBox

- **Composant à 2 états** : **Coché** / **Non coché** (hérite de **Button**)
- La propriété **`android:text`** permet de définir le libellé associé à la CheckBox

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="CheckBox" />
```

Switch

- Bouton à **2 états** : **On** / **Off** (hérite de **Button**)
- Le bouton est représenté graphiquement « Allumé » ou « Eteint »
- Pour **détecter** le changement d'état d'un **Switch**, **ToggleButton**, **CheckBox utilisateur** Il est possible d'attacher un **`CompoundButton.OnCheckedChangeListener`**

```
Switch switchBtn = findViewById(R.id.switchBtn);
switchBtn.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            //...
        } else {
            //...
        }
    }
});
```

RadioButton

- **Composant à 2 états** : **Coché** / **Non coché** (hérite de **Button**)
- Les **RadioButton** sont utilisés en groupe (**RadioGroup**) **1 seul bouton** du groupe peut être **coché à la fois**.
- Lorsque l'utilisateur clique sur un bouton du groupe:
  - Le bouton passe dans l'état coché
  - Si un bouton du groupe était dans l'état coché, il est décoché.

```
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:orientation="horizontal">
        <RadioButton
            android:id="@+id/rbOui"
          android:text="Oui" />
        <RadioButton
          android:id="@+id/rbNon"
            android:text="Non" />
        <RadioButton
          android:id="@+id/rbPe"
          android:text="Peut être" />
</RadioGroup>
```

Pour détecter les click de l'utilisateur, il faut attacher à chaque **RadioButton** un `View.OnClickListener()`
(possibilité d'utiliser le même Listener pour tous les boutons du groupe)

Spinner (1)

- **Composant** permettant de **sélectionner une valeur dans une liste** (**ComboBox**)
- La liste des **valeurs possibles** peut être **définie** dans le **Layout XML** ou **dans le code**

```
<resources>
    <string-array name="spinerItems">
        <item>Choix 1</item>
        <item>Choix 2</item>
        <item>Choix 3</item>
    </string-array>
</resources>
```

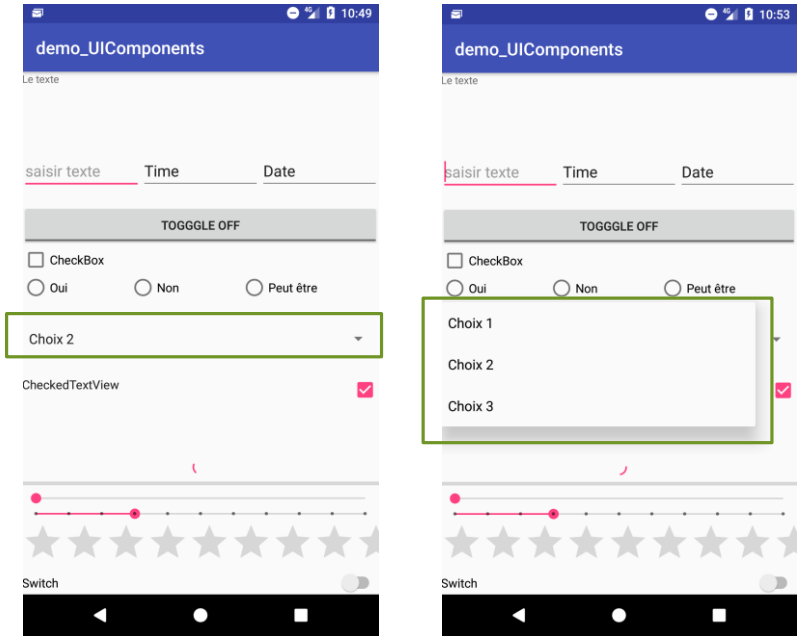
```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    <!--- Liste des valeurs -->
    android:entries="@array/spinerItems"
/>
```

```
// Spinner (ComboBox)
Spinner spinner = findViewById(R.id.spinner);
// Création d'un ArrayAdapter utilisant
// le tableau spinerItems et un layout de spinner
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
    this, R.array.spinerItems, android.R.layout.simple_spinner_item
);
// Définition du layout à utiliser lorsque la liste s'affiche
adapter.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item
);
// Affectation de l'adapter au Spinner
spinner.setAdapter(adapter);
```

Spinner (2)

- Pour **détecter** quand l'**utilisateur sélectionne un élément** de la liste :

```
// Listener de sélection
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        //...
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        //...
    }
});
```



ProgressBar (1)

- **Composant** destiné à montrer visuellement la **progression d'un traitement**
- Il existe **2 modes d'utilisation** des **ProgressBar** : **Determinate** et **Indeterminate**
- **Mode Indeterminate :**

Utilisé lorsqu'il est impossible de déterminer à l'avance la durée du traitement.

```
<ProgressBar
android:id="@+id/indeterminateProgressBar"
    android:layout_width="match_parent"
    android:layout_height="22dp" />
```

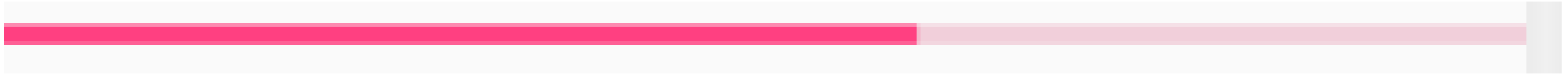
(Animation permanente)

ProgressBar (2)

- **Mode Determinate :** utilisé lorsque l'on veut montrer la progression d'un traitement comme par exemple: le pourcentage effectué par rapport à la totalité.
- Pour utiliser une **ProgressBar** en mode **Determinate** il faut lui affecter le style:

```
?android:attr/progressBarStyleHorizontal
```

```
<ProgressBar
    android:id="@+id/determinateProgressBar"
    style="@android:style/Widget.DeviceDefault.Light.ProgressBar.Horizontal"
    android:progress="60" /> (Progression à 60%)
```



- L'actualisation d'une **ProgressBar** peut être réalisée dans une **AsyncTask**

SeekBar (1)

- **Variante de la Barre de progression (ProgressBar)** présentée sous la forme d'une **barre** comportant un **curseur déplaçable vers la gauche ou la droite** par l'utilisateur.

- **SeekBar basique:**

```
<SeekBar
    android:id="@+id/seekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:progress="50" /> (50%)
```

- **SeekBar discrète :**

```
<SeekBar
    android:id="@+id/seekBarDiscrete"
    style="@style/Widget.AppCompat.SeekBar.Discrete"   (Style: Discrete)
    android:max="10" (graduée de 0 à 10)
    android:progress="3" /> (3 sur 10)
```

SeekBar (2)

- Pour **détecter** quand l'**utilisateur déplace le curseur** de la **SeekBar** Il est possible d'attacher un **SeekBar.OnSeekBarChangeListener**

```
SeekBar seekBar = findViewById(R.id.seekBar);
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
    }
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
    }
});
```

RatingBar

- **Composant issu de la Barre de progression (ProgressBar) et de la SeekBar** destinés à donner une note en nombre d'étoiles.

```
<RatingBar
    android:id="@+id/ratingBar"
    android:numStars="5"  (note maximum)
    android:rating=« 3"     (note attribuée)
    android:isIndicator="true"  (mode lecture seule: vrai/faux)
/>
```

- Pour **détecter** quand l'**utilisateur modifie la note** Il est possible d'attacher un **android.widget.RatingBar.OnRatingBarChangeListener**

```
RatingBar ratingBar = findViewById(R.id.ratingBar);
ratingBar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {
    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {
    }
});
```

Menus (1)

- Les 3 types de menu:
  - Les **Options Menu** pour la navigation dans l'application
  - Les **Menus Contextuels** pour proposer des actions en fonction d'une élément sélectionné
  - Les **Popups Menu** pour les actions secondaires. (menu modal attaché à une View)

Menus (2) - Options Menu

```xml
<!- Menu défini en XML -->
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/opt1"
        android:icon="@android:drawable/ic_lock_lock"
        android:title="Option 1"
        android:visible="true"></item>
    <!- Sous-Menu -->
    <item
        android:id="@+id/subMenu1"
        android:checkable="false"
        android:icon="@android:drawable/ic_lock_power_off"
        android:title="Sous-menu 1">
        <menu>
            <group android:id="@+id/grp1">
                <item
                    android:id="@+id/grp1_opt1"
                    android:title="sous opt 1" />
                <item
                    android:id="@+id/grp1_opt2"
                    android:title="sous opt 2" />
            </group>
        </menu>
    </item>
</menu>
```

Dans l'activité:

```java
// Redéfinition de la méthode onCreateOptionsMenu(Menu menu)
// de l'activité pour créer le menu
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main_menu, menu);
    return true;
}

// Redéfinition de la méthode onOptionsItemSelected(MenuItem item)
// de l'activité pour créer intercepter les click sur les menus
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Toast.makeText(this, item.getTitle(), Toast.LENGTH_LONG).show();
    return super.onOptionsItemSelected(item);
}
```

Menus (3) -Menu Contextuel

```
<!- Menu défini en XML -->
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/opt1"
        android:icon="@android:drawable/ic_lock_lock"
        android:title="Option 1"
        android:visible="true"></item>
    <!- Sous-Menu -->
    <item
        android:id="@+id/subMenu1"
        android:checkable="false"
        android:icon="@android:drawable/ic_lock_power_off"
        android:title="Sous-menu 1">
        <menu>
            <group android:id="@+id/grp1">
                <item
                    android:id="@+id/grp1_opt1"
                    android:title="sous opt 1" />
                <item
                    android:id="@+id/grp1_opt2"
                    android:title="sous opt 2" />
            </group>
        </menu>
    </item>
</menu>
```

Dans l'activité:

```
// Dans le onCreate() on délcare les View sur lequel on souhaite attacher
// un menu contextuel par la méthode: registerForContextMenu(View view)

// Demande d'enregistrement d'un menu contextuel pour la TextView
registerForContextMenu(textView);


// Redéfinition de la méthode onCreateContextMenu()
// de l'activité pour créer le menu contextuel quand
// un appui long sera détecté sur le textview
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(R.menu.main_menu, menu);
}

// Redéfinition de la méthode onContextItemSelected(MenuItem item)
// de l'activité pour intercepter les clicks sur les menus
@Override
public boolean onContextItemSelected(MenuItem item) {
    Toast.makeText(this, item.getTitle(), Toast.LENGTH_LONG).show();
    return super.onOptionsItemSelected(item);
}
```

Un exemple intéressant : https://www.mikeplate.com/2010/01/21/show-a-context-menu-for-long-clicks-in-an-android-listview/

Menus (4) – Popup Menu

```
<!- Menu défini en XML -->
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/opt1"
        android:icon="@android:drawable/ic_lock_lock"
        android:title="Option 1"
        android:visible="true"></item>
    <!- Sous-Menu -->
    <item
        android:id="@+id/subMenu1"
        android:checkable="false"
        android:icon="@android:drawable/ic_lock_power_off"
        android:title="Sous-menu 1">
        <menu>
            <group android:id="@+id/grp1">
                <item
                    android:id="@+id/grp1_opt1"
                    android:title="sous opt 1" />
                <item
                    android:id="@+id/grp1_opt2"
                    android:title="sous opt 2" />
            </group>
        </menu>
    </item>
</menu>
```

Dans onCreate de l'activité création et affichage d'un Popup Menu sur click du ToggleButton:

```
ToggleButton tgButton = findViewById(R.id.toggleButton);
tgButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Création du Menu Popup
        PopupMenu popup = new PopupMenu(DemoWidgetActivity.this, v);
        // Affectation de son Listener au Popup Menu
        popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener()
{
            @Override
            public boolean onMenuItemClick(MenuItem item) {
                Toast.makeText(DemoWidgetActivity.this, item.getTitle(),
Toast.LENGTH_LONG).show();
            }
        });
        // Chargement du Popup Menu
        popup.getMenuInflater().inflate(R.menu.main_menu, popup.getMenu());
        // Affichage du Popup Menu
        popup.show();
    }
});
```

WebView (1)

- **Composant** chargé d'**afficher des pages web** (utilise moteur **WebKit**)
- Délégation au **navigateur par défaut** ou rendu dans la **WebView**
- **Fonctionnalités Configurables :**

```
// Active/Désactive le support du Javascript
webView.getSettings().setJavaScriptEnabled(false);
// Active/Désactive l'utilisation de Content provider locaux
webView.getSettings().setAllowContentAccess(false);
// Active/Désactive le chargement de donnée à partir de fichiers locaux
webView.getSettings().setAllowFileAccess(false);
// Active/Désactive le Zoom
webView.getSettings().setBuiltInZoomControls(true);
// ...
```

- **Chargement et affichage :**

```
// Chargement à partir d'une Url
loadUrl(String url)
// Chargement à partir d'une DateUrl du mimeType et de l'encoding
loadData(String data, String mimeType, String encoding)
```

WebView (2)

- **Comportement personnalisable par création d'un `android.webkit.WebViewClient`**
  https://developer.android.com/reference/android/webkit/WebViewClient.html

```
// Décision d'utliliser la WebView ou le navigateur par défaut
// false: rendu par la WebView
// true: rendu à effectuer autrement
public boolean shouldOverrideUrlLoading(WebView view, String url)
// Notification des erreurs
public void onReceivedError(WebView view, WebResourceRequest request, WebResourceError error)
// ...
webView.setWebViewClient(new CustomWebViewClient());
```

- **Chargement et affichage :**

```
// Chargement à partir d'une Url
loadUrl(String url)
// Chargement à partir d'une DataUrl du mimeType et de l'encoding
loadData(String data, String mimeType, String encoding)
```

Nécessité d'obtenir la permission INTERNET pour utiliser la WebView: `<uses-permission android:name="android.permission.INTERNET" />`

WebView (3)

- **Utilisation d'une WebView:**
  - **Création d'un WebViewClient personnalisé**

```
// WebViewClient personnalisé
public static class CustomWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request) {
        Log.i("URL", request.getUrl().toString());
        // Url autorisée si termine par index.jsf
        if(request.getUrl().toString().contains("index.jsf")) {
            return false;
        } else {
            // Bloque autres pages
            return true;
        }
    }
    @Override
    public void onReceivedError(WebView view, WebResourceRequest request, WebResourceError error) {
        Log.i("WEBVIEW", error.toString());
    }
    @Override
    public void onReceivedHttpError(WebView view, WebResourceRequest request, WebResourceResponse errorResponse) {
        Log.i("WEBVIEW", errorResponse.toString());
    }
}
```

WebView (4)

- **Rendu d'une page Web à partir de son URL**

```
// Affichage d'une page Web dans la WebView
WebView webView = findViewById(R.id.webview);
webView.setWebViewClient(new CustomWebViewClient());
webView.loadUrl("http://www.sysord.com");
```

- **Rendu d'une page à partir d'une DataUrl**


```
// Affichage d'une page Web créée en texte dans la WebView
String htmlContent = "<html><body>Ma<b> page</b> blabla...</body></html>";
webView.loadData(htmlContent, "text/html", null);
```

WebView (5)

- **Interface entre le Javascript d'une page Web et une application**


```
/**
 * Bib accessible par Js à partir de la page Web pour afficher un Toast
 */
public class WebJsBridge {
    @JavascriptInterface
    public void showToast(String msg) {
        Toast.makeText(DemoWebviewActivity.this, msg, Toast.LENGTH_SHORT).show();
    }
}


// Active le support du Javascript
webView.getSettings().setJavaScriptEnabled(true);
// installation de la lib
webView.addJavascriptInterface(new WebJsBridge(), "ToasterLib");
// Affectation de la lib interface de communication
// Page avec bouton
htmlContent = "<html>"
                + "<body>"
                + "Pour tester cliquer sur le bouton "
                + "<input type=\"button\" value=\"...\" onClick=\"ToasterLib.showToast('Ca Marche !!!')\" />"
                + "</body>"
            + "</html>";
webView.loadData(htmlContent, "text/html", null);
```

Les Evénements principaux (1)

- Listener de clic (tactile, trackball, validation sur bouton) :

```
public interface OnClickListener {
    /**
     * Called when a view has been clicked.
     * @param v The view that was clicked.
     */
    void onClick(View v);
}
```

```
EditText editText = findViewById(R.id.editText);
editText.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // traiter le click
    }
});
// avec une lambda
editText.setOnClickListener((View v) -> {
    // traiter le click
});
```

- Listener de clic long (appui > 1 seconde) :

```
public interface OnLongClickListener {
    /**
     * Called when a view has been clicked and held.
     * @param v The view that was clicked and held.
     * @return true if the callback consumed
     * the long click, false otherwise.
     */
    boolean onLongClick(View v);
}
```

```
editText.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        // traiter le click long
        return false;
    }
});
editText.setOnLongClickListener((View v) -> {
    // traiter le click long
    return false;
});
```

Les Evénements principaux (2)

- Listener de Focus (perte ou gain du focus) :

```
public interface OnFocusChangeListener {
    /**
     * Called when the focus state of a view has changed.
     *
     * @param v The view whose state has changed.
     * @param hasFocus The new focus state of v.
     */
    void onFocusChange(View v, boolean hasFocus);
}
```

```
editText.setOnFocusChangeListener(new View.OnFocusChangeListener()
{
    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        //...
    }
});
editText.setOnFocusChangeListener( (v, hasFocus) -> {
    //...
});
```

- Listener de touche (clavier physique uniquement) :

```
public interface OnKeyListener {
    /**
     * Called when a hardware key is dispatched to a view
     * @param v The view the key has been dispatched to.
     * @param keyCode The code for the physical key that was pressed
     * @param event The KeyEvent object containing full information
     *              about the event.
     * @return True if the listener has consumed the event,
     *         false otherwise.
     */
    boolean onKey(View v, int keyCode, KeyEvent event);
}
```

```
editText.setOnKeyListener(new View.OnKeyListener() {
      @Override
      public boolean onKey(View v, int keyCode, KeyEvent event) {
          return false;
      }
});
editText.setOnKeyListener((v, kc, e) -> {
    //..
});
```

TouchListener (1)

- Listener de Touch (placements et déplacements de 1 à plusieurs doigt sur l'écran :

Pour capturer les événement de Touch sur une activité: redéfinition de **OnTouchEvent**

```
public boolean onTouchEvent(MotionEvent event) {
    int action = MotionEventCompat.getActionMasked(event);
    switch(action) {
        case (MotionEvent.ACTION_DOWN) :
            Log.d("onTouchEvent","Premier doigt posé");
            return true;
        case (MotionEvent.ACTION_POINTER_DOWN) :
            Log.d("onTouchEvent","Doigt supplémentaire posé");
            return true;
        case (MotionEvent.ACTION_MOVE) :
            Log.d("onTouchEvent","Au moins 1 doigt bouge");
            return true;
        case (MotionEvent.ACTION_POINTER_UP) :
            Log.d("onTouchEvent","Doigt autre que le dernier retiré");
            return true;
        case (MotionEvent.ACTION_UP) :
            Log.d("onTouchEvent","Dernier doigt retiré (fin de l'action)");
            return true;
        default :
            return super.onTouchEvent(event);
    }
}
```

TouchListener (2)

- Pour capturer les événements de Touch sur une View: ajout **`View.OnTouchListener`**

```
editText.setOnTouchListener(new View.OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        int action = MotionEventCompat.getActionMasked(event);
        switch(action) {
            . . .
        }
        return true;
    }
});
```

Référence des **`MotionEvent`**: https://developer.android.com/reference/android/view/MotionEvent.html

- Le nombre de doigts utilisés et leurs déplacements réalisent des « Figures » ou « Gestes ». Android fournit un outil pour reconnaitre les figures les plus courantes : le **`GestureDetector`**, il existe aussi un **`ScaleGestureDetector`**

GestureDetector (1)

- Utilisation du GestureDetector:

```
public class DemoTouchActivity extends AppCompatActivity
    implements GestureDetector.OnGestureListener,
        GestureDetector.OnDoubleTapListener {

    private GestureDetectorCompat mDetector;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_demo_touch);
        // Instantiate the gesture detector with the
        // application context and an implementation of
        // GestureDetector.OnGestureListener
        mDetector = new GestureDetectorCompat(this,this);
        // Set the gesture detector as the double tap
        // listener.
        mDetector.setOnDoubleTapListener(this);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        // délégation de la détection
        this.mDetector.onTouchEvent(event);
        return super.onTouchEvent(event);
    }

    // Implémentation des interfaces
    @Override
    public void onLongPress(MotionEvent e)
    {
       Log.i("TOUCH","onLongPress: " + e.toString());
    }
    @Override
    public boolean onDoubleTap(MotionEvent e) {
       Log.i("TOUCH","onDoubleTap: " + e.toString());
        return true;
    }
    @Override
    public boolean onDoubleTapEvent(MotionEvent e) {
       Log.i("TOUCH","onDoubleTapEvent: " + e.toString());
        return true;
    }
    @Override
    public boolean onDown(MotionEvent e) {
       Log.i("TOUCH","onDown: " + e.toString());
       Log.i("TOUCH","onDown:");
        return true;
    }
    @Override
    public void onShowPress(MotionEvent e) {
       Log.i("TOUCH", "onShowPress: " + e.toString());
    }
}
```

GestureDetector (2)

```
    @Override // "Fling is a quick swipe action that has no specific target"
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
       Log.i("TOUCH","onFling: " + e1.toString() + " " + e2.toString());
        return true;
    }
    @Override
    public boolean onSingleTapUp(MotionEvent e) {
       Log.i("TOUCH", "onSingleTapUp: " + e.toString());
        return true;
    }
    @Override
    public boolean onSingleTapConfirmed(MotionEvent e) {
       Log.i("TOUCH","onDown: " + e.toString());
        return true;
    }
    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
       Log.i("TOUCH", "onScroll: " + e1.toString() + e2.toString());
        return true;
    }
}
```

Le GestureDetector est assez sommaire, il peut être nécessaire de créer son propre détecteur de « Gestes » en héritant de **`GestureDetector.SimpleOnGestureListener`**

Multitouch

- Comme vu dans exemples précédents: L'écran identifie tous les **doigts utilisés**
- **MotionEvent** contient un tableau avec les coordonnées de chaque doigt
  - Pour connaitre le nombre de doigts: **`Event.getPointerCount()`**
  - Pour chaque index (doigt) des infos de positions sont disponibles:

    `Event.getX(int index)`, **`Event.getY(int index)`** etc.

  - Les infos sur les doigts ne sont pas toujours dans le même ordre (même index) mais à sa première détection un ID est affecté à un doigt et la correspondance entre index et pointerId est fourni par les méthodes :

    `Event.getPointerId(int index)` et **`Event.getPointerIndex(int pointerId)`**

  - Pour les actions **`ACTION_POINTER_DOWN`** et **`ACTION_POINTER_UP`** la méthode

    `Event.getActionIndex()` retourne l'index du doigt ajouté ou retiré.