



Gestion des évènements sous android



Plan

I.	INTRODUCTION	1
II.	CARACTERISTIQUES D'UN EVENEMENT	1
III.	SOLUTIONS D'IMPLEMENTATION DES ECOUTEURS	1
III.1.	L'ACTIVITE ELLE-MEME IMPLEMENTE L'ECOUTEUR (SOLUTION1)	2
III.2.	UNE CLASSE ANONYME IMPLEMENTE L'ECOUTEUR (SOLUTION2)	2
III.3.	EXEMPLE	2
III.3.1.	Le code de la classe MainActivity en utilisant solution1	2
III.3.2.	Le code de la classe MainActivity en utilisant la solution2	3
IV.	ECOUTEURS A USAGE COURANT DE LA CLASSE VIEW [3]	4
IV.1.	ONCLICKLISTENER	5
IV.2.	ONFOCUSCHANGELISTENER	5
IV.3.	ONKEYLISTENER	5
IV.4.	ONLONGCLICKLISTENER	5
IV.5.	ONTOUCHLISTENER	5
IV.6.	ONDRAGLISTENER	5

I. Introduction

Sous Android, toutes les actions de l'utilisateur sont perçues comme un événement, que ce soit le clic sur un bouton d'une interface, le maintien du clic, l'effleurement d'un élément de l'interface, etc. Ces événements peuvent être interceptés par les éléments de votre interface pour exécuter des actions en conséquence. Le mécanisme d'interception repose sur la notion d'écouteurs, aussi appelés écouteurs (listeners) dans la documentation Java. Il permet d'associer un événement à une méthode à appeler en cas d'apparition de cet événement. Si un écouteur est défini pour un élément graphique et un événement précis, la plate-forme Android appellera la méthode associée dès que l'événement sera produit sur cet élément. Par exemple, on pourra définir un écouteur sur l'événement clic d'un bouton pour afficher un message «Bouton cliqué !». C'est justement ce que nous allons faire ci-après. Notez que les événements qui peuvent être interceptés ainsi que les méthodes associées sont imposés. Pour un événement `OnClick` (élément cliqué), la méthode associée sera `OnClick()` : il nous suffira alors de définir cette méthode pour notre écouteur pour qu'elle soit appelée lorsque l'utilisateur cliquera sur l'élément graphique associé. Ainsi, pour que l'interface réagisse à un événement sur un élément, il faudra choisir l'élément et l'événement à intercepter, définir un écouteur sur cet événement et définir la méthode associée [1].

Notez que les événements qui peuvent être interceptés ainsi que les méthodes associées sont imposés. Pour un événement `OnClick` (élément cliqué), la méthode associée sera `OnClick()` : il nous suffira alors de définir cette méthode pour notre écouteur pour qu'elle soit appelée lorsque l'utilisateur cliquera sur l'élément graphique associé. Ainsi, pour que l'interface réagisse à un événement sur un élément, il faudra choisir l'élément et l'événement à intercepter, définir un écouteur sur cet événement et définir la méthode associée [1].

II. Caractéristiques d'un événement

Pour gérer convenablement des événements sous android, il faut bien tenir compte des caractéristiques d'un événement qui sont :

- **La source d'évènement** : c'est l'élément c.-à-d. la View qui va subir l'évènement. Exemple : un Button, un Checkbox, un EditText, un Spinner, un élément d'un Spinner...
- **L'évènement** : c'est l'action que va subir la source d'évènement. Exemple : Click, LongClick, Touch, Key, FocusChanged...
- **L'écouteur** : c'est une interface qui vous oblige à redéfinir des méthodes de callback et chaque méthode sera appelée au moment où se produira l'évènement associé [2]. Le nom d'un écouteur est de la forme : `OnEventListener` (`OnClickListener`, `OnLongClickListener`, `OnTouchListener`...).
- **L'abonnement de l'écouteur à l'évènement** : la source d'évènement appelle une méthode de la forme `setEcouteur(ecouteur)` pour que «écouteur» intercepte les événements qui se produisent sur la source d'évènement. Exemple : `btnValider.setOnClickListener(ecouteur1)`, `edSaisie.setOnKeyListener(ecouteur2)`...
- **Les méthodes de l'écouteur à redéfinir** : chaque écouteur possède au moins une méthode abstraite qui sera redéfinie par l'utilisateur pour contenir l'action à exécuter lorsque l'évènement se produit. Le nom de ces méthodes peut être de la forme `onEvènement` (`onClick(...)`, `onLongClick(...)`, `onTouch(...)`), mais il peut aussi être sous une autre forme (L'écouteur `OnItemSelectedListener` possède deux méthodes abstraites : `onItemSelected(...)` et `onNothingSelected(...)`).
- **Les paramètres des méthodes** : chaque méthode possède au moins un paramètre. Le premier paramètre est de type View et il représente la source de l'évènement. Les autres paramètres dépendent de la nature de l'évènement et ils contiennent un ensemble de données sur l'évènement qui s'est produit. Par exemple pour la méthode `onClick(View view, int keyCode, KeyEvent event)` : le premier paramètre représente la source d'évènement, le deuxième représente le code du caractère tapé et le troisième représente un événement qui contient toutes les autres caractéristiques de la tape sur le clavier tels que le nombre de répétition, l'état de la touche CTRL ou SHIFT...
- **L'action** : c'est le traitement à exécuter lorsque l'écouteur intercepte l'évènement sur la source d'évènement. Il faut appeler cette action dans la méthode adéquate de l'écouteur. Dans certains cas il faut accéder aux paramètres de la méthode de l'écouteur pour extraire des propriétés de l'évènement utile dans le traitement (code de la touche du clavier, coordonnées du pointeur pour une touchée...).

III. Solutions d'implémentation des écouteurs

Il existe quatre solutions pour implémenter les écouteurs sous java qui sont donc possible sous android :

- La classe elle-même (l'activité pour android) implémente l'écouteur
- Une classe anonyme implémente l'écouteur
- Une classe indépendante implémente l'écouteur
- Une classe interne implémente l'écouteur

Bien que ces quatre solutions sont équivalentes et produisent le même résultat, la solution1 et la solution2 paraissent plus simple à utiliser. Pour cela on détaille ici seulement ces deux solutions et lors de la programmation d'un problème, il faut choisir celle qui s'adapte le mieux à la nature du problème.

III.1. L'activité elle-même implémente l'écouteur (Solution1)

Dans cette solution l'activité implémente l'écouteur, elle doit donc redéfinir toutes ses méthodes abstraites. Dans ce cas la source d'évènement s'inscrit à l'écouteur en passant « this » dans le paramètre de la méthode setEcouteur(...). Exemple `btnValider.setOnClickListener(this); edSaisie.setOnKeyListener(this); imgFruit.setOnTouchListener(this);`

Dans cette solution, deux sources d'évènement peuvent inscrire l'activité pour intercepter le même type d'évènement. Pour traiter convenablement les évènements, il faut pouvoir identifier la source d'évènement qui a subi l'évènement à traiter. Ceci est possible en accédant au premier paramètre de la méthode de traitement de l'évènement qui est « View view ». `view.getId()` retourne l'identifiant de la View qui a subi l'évènement. Il faut donc utiliser un if ou un switch pour comparer `view.getId()` aux différents `R.id.nomView` pour identifier la View et appeler l'action adéquate.

Une classe anonyme implémente l'écouteur (Solution2)

Dans cette solution l'instanciation de l'écouteur se fait au moment de son abonnement à l'évènement en utilisant « new » et en ajoutant la redéfinition des méthodes abstraites. La classe résultante de cette technique produit une instance d'une classe qui ne possède pas de nom, d'où le nom classe anonyme. Dans cette solution chaque source d'évènement possède ses propres écouteur donc a priori il n'y a pas un besoin d'un test sur l'identifiant de la View.

Exemple

Dans cet exemple nous avons deux Button (`btnRemplir` et `btnVider`) et un EditText (`edSaisie`). Le click sur un bouton affiche un message qui indique « Click sur Remplir ! » ou « Click sur Vider ! » suivant le bouton cliqué. Lorsqu'un caractère est tapé dans « `edSaisie` » son code est affiché.

III.1.1. Le code de la classe MainActivity en utilisant solution1

```
public class MainActivity extends Activity implements OnClickListener, OnKeyListener {
    private Button btnRemplir;
    private Button btnVider;
    private EditText edSaisie;
    ...
    private void ajouterEcouteur() {
        btnRemplir.setOnClickListener(this);
        btnVider.setOnClickListener(this);
        edSaisie.setOnKeyListener(this);
    }
    @Override
    public void onClick(View arg0) {
        switch(arg0.getId()){
            case R.id.btnRemplir:
                remplir();
                break;
            case R.id.btnVider:
                vider();
                break;
        }
    }
    @Override
    public boolean onKey(View arg0, int arg1, KeyEvent arg2) {
        afficher(arg1);
        return true;
    }
    public void remplir () {
        Toast t=Toast.makeText(this, « Click sur Remplir ! », Toast.LENGTH_LONG);
        t.show();
    }
    public void vider() {
        Toast t=Toast.makeText(this, « Click sur Vider ! », Toast.LENGTH_LONG);
        t.show();
    }
    public void afficher(int keyCode) {
        Toast t=Toast.makeText(this, « Code du caractère : » + keyCode, Toast.LENGTH_LONG);
        t.show();
    }
}
```

III.1.2. Le code de la classe MainActivity en utilisant la solution2

```
public class MainActivity extends Activity {
    private Button btnRemplir;
    private Button btnVider;
    private EditText edSaisie;
    ...
    private void ajouterEcouleur() {
        btnRemplir.setOnClickListener( new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                remplir();
            }
        });
        btnVider.setOnClickListener( new OnClickListener(){
            @Override
            public void onClick(View arg0) {
                vider();
            }
        });
        edSaisie.setOnKeyListener( new OnKeyListener(){
            @Override
            public boolean onKeyDown(View arg0, int arg1, KeyEvent arg2) {
                afficher(arg1);
                return true;
            }
        });
    }
    public void remplir () {
        Toast t=Toast.makeText(this, « Click sur Remplir ! », Toast.LENGTH_LONG);
        t.show();
    }
    public void vider() {
        Toast t=Toast.makeText(this, « Click sur Vider ! », Toast.LENGTH_LONG);
        t.show();
    }
    public void afficher(int keyCode) {
        Toast t=Toast.makeText(this, « Code du caractère : » + keyCode, Toast.LENGTH_LONG);
        t.show();
    }
}
```

IV. Utilisation des expressions Lambda dans les évènements

Les expressions lambda permettent d'écrire du code plus concis, donc plus rapide à écrire, à relire et à maintenir [5].

```
//sans expression lambda
btnValider.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        valider();
    }
});
//avec expressions lambda
btnValider.setOnClickListener( v -> valider() );
```

V. Ecouteurs à usage courant de la classe View [3]

Package	Listeners	Methods	Parameters
android.view.View	OnClickListener Interface definition for a callback to be invoked when a view is clicked.	public abstract void onClick (View v)	v : The view that was clicked.
	OnFocusChangeListener Interface definition for a callback to be invoked when the focus state of a view changed.	public abstract void onFocusChange (View v, boolean hasFocus) Called when the focus state of a view has changed.	v : The view that was clicked. hasFocus : The new focus state of v.
	OnKeyListener Interface definition for a callback to be invoked when a hardware key event is dispatched to this view. The callback will be invoked before the key event is given to the view. This is only useful for hardware keyboards; a software input method has no obligation to trigger this listener.	public abstract boolean onKey (View v, int keyCode, KeyEvent event) Called when a hardware key is dispatched to a view. This allows listeners to get a chance to respond before the target view. Key presses in software keyboards will generally NOT trigger this method, although some may elect to do so in some situations. Do not assume a software input method has to be key-based; even if it is, it may use key presses in a different way than you expect, so there is no way to reliably catch soft input key presses. Returns true if the listener has consumed the event, false otherwise.	v : The view that was clicked. keyCode : The code for the physical key that was pressed event : The KeyEvent object containing full information about the event.
	OnLongClickListener Interface definition for a callback to be invoked when a view has been clicked and held.	public abstract boolean onLongClick (View v) Called when a view has been clicked and held. Returns true if the callback consumed the long click, false otherwise	v : The view that was clicked.
	OnTouchListener Interface definition for a callback to be invoked when a touch event is dispatched to this view. The callback will be invoked before the touch event is given to the view.	public abstract boolean onTouch (View v, MotionEvent event) Called when a touch event is dispatched to a view. This allows listeners to get a chance to respond before the target view. Returns true if the callback consumed the long click, false otherwise	v : The view that was clicked. event : The MotionEvent object containing full information about the event.
	OnDragListener Interface definition for a callback to be invoked when a drag is being dispatched to this view. The callback will be invoked before the hosting view's own onDrag(event) method. If the listener wants to fall back to the hosting view's onDrag(event) behavior, it should return 'false' from this callback.	public abstract boolean onDrag (View v, DragEvent event) Called when a drag event is dispatched to a view. This allows listeners to get a chance to override base View behavior. Returns true if the drag event was handled successfully, or false if the drag event was not handled. Note that false will trigger the View to call its onDragEvent() handler.	v : The View that received the drag event. event : The DragEvent object for the drag event.

V.1.1. OnClickListener

Définit les écouteurs d'événements de type clic. La méthode à surcharger pour traiter les événements est : `onClick(View)`.

V.1.2. OnFocusChangeListener

Définit les écouteurs d'événements de type clic long. La méthode à surcharger pour traiter les événements est : `onFocusChange(View,boolean)`. Le deuxième paramètre a pour valeur `true` si la View a gagné le focus et `false` sinon.

V.1.3. OnKeyListener

Définit les écouteurs d'événements de type clavier. La méthode à surcharger pour traiter les événements est : `onKey(View, int, KeyEvent)`.

Le deuxième paramètre est le code de la touche tapée, le troisième est l'événement clavier.

Le code de la touche tapée.

La classe `KeyEvent` (`android.view.KeyEvent`) est associée aux événements clavier, parmi ses méthodes:

- `getAction()` qui renvoie un entier pouvant prendre les valeurs `ACTION_DOWN` ou `ACTION_UP` qui indique l'action faite sur la touche (appuyée, lâchée).
- `getRepeatCount()` renvoie le nombre de répétitions lorsque la touche est maintenue
- `getKeyCode()` renvoie le code de la touche (même valeur que le dernier paramètre de `onKey`)
- Les méthodes `isAltPressed()`, `isShiftPressed()`, `isCtrlPressed()`, `isCapsLockOn()`, `isNumLockOn()`, `isScrollLockOn()` permettent de tester l'état des touches de modification.

V.1.4. OnLongClickListener

Définit les écouteurs d'événements de type clic long. La méthode à surcharger pour traiter les événements est : `onLongClick(View)`.

V.1.5. onTouchListener

Définit les écouteurs d'événements de type touché. La méthode à surcharger pour traiter les événements est : `onTouchEvent(MotionEvent)`

La classe `MotionEvent` (`android.view.MotionEvent`) est associée aux événements touch, parmi ses méthodes:

- `getAction()` qui renvoie un entier pouvant prendre les valeurs `ACTION_DOWN`, `ACTION_UP`, `ACTION_MOVE` ou `ACTION_OUTSIDE`
- `getPressure()` renvoie un réel entre 0 et 1 indiquant la force de pression du touché sur l'écran
- `getX()` et `getY()` indiquent les coordonnées (resp en x et en y).
- `getXPrecision()` et `getYPrecision()` renvoient un réel indiquant la précision des coordonnées (resp en x et en y).

V.1.6. OnDragListener

Définit les écouteurs d'événements de type drag. Cet événement être déclenché par la méthode `startDrag(...)` de la classe `View`.

La classe `DragEvent` (`android.view.DragEvent`) est associée aux événements drag, parmi ses méthodes:

- `getAction()` qui renvoie un entier pouvant prendre les valeurs `ACTION_DRAG_STARTED`, `ACTION_DRAG_ENTERED`, `ACTION_DRAG_EXITED` ou `ACTION_DROP`.
- `getX()` et `getY()` qui retournent les coordonnées de l'objet glissé.

Références

[1] Damien Guignard, Julien Chabre, Emmanuel Robles, Programmation Android de la conception au déploiement avec le SDK Google Android 2, Eyrolles, ISBN : 978-2-212-12587-0

[2] <https://openclassrooms.com/courses/creez-des-applications-pour-android/les-widgets-les-plus-simples>

[3] <http://developer.android.com/reference/android/view/View.html>

[4] Développement d'applications pour Android, M. Dalmau, IUT de Bayonne-Pays Basque

[5] <https://www.jmdoudoux.fr/java/dej/chap-lambdas.htm#lambdas-2>