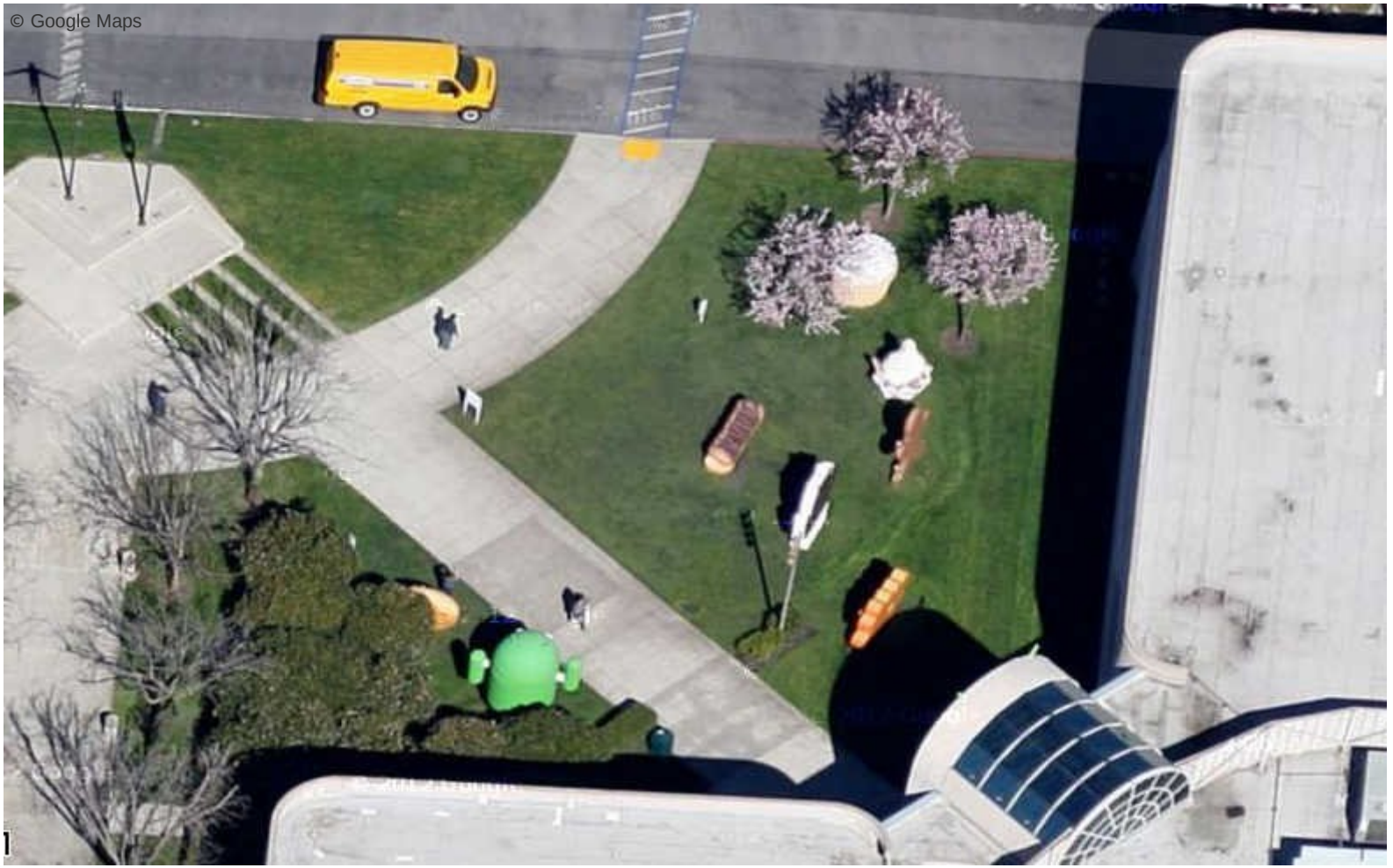
Interface graphique sous Android

Master 2 informatique 2012-2013



Les ressources

- Ressource : élément de vue réutilisable
- Ressources définies dans des fichiers XML (ou binaires) dans res/ (noms de fichiers [a-z0-9_]+) :
  - res/values : déclaration XML (avec i18n) de string, color, dimen, style...
  - res/drawable : fichiers multimédias binaires (images, sons)
  - res/layout : définition XML d'agencements de vues graphiques (sous forme d'arbre)
  - res/anim : fichiers XML de définition d'animations
    - Animation d'interpolation (changement de transparence, échelle, angle de rotation...)
    - Animation pour séquences d'images : liste des images avec leur durée d'affichage
  - res/xml : pour des fichiers XML divers
  - res/raw : pour d'autres ressources sous la forme de fichiers binaires

Références aux ressources

- Référencement de chaque ressource par une constante entière dans des classes internes de R.java
- Utilisation de Resources Context.getResources() (Activity hérite de Context) pour obtenir un getter de ressources
- Quelques exemples :
  - getString(R.string.hello_world)
  - getStringArray(R.stringarray.messages)
  - getColor(R.color.my_nice_color)
  - getLayout(R.layout.activity_layout)
  - getDimension(R.dimen.world_width)
  - getDrawable(R.drawable.card_picture)
  - getIntArray(R.intArray.coordinates)
  - openRawResource(R.raw.bindata) (retourne un InputStream)
- Référencement d'une ressource au sein d'une autre ressource par @[paquetage:]type/nomDeLaRessource
- Il exite des ressources système en utilisant le paquetage android (par exemple : @android:color/blue)
- Accès direct à une vue depuis une activité : View Context.findViewById(int)

Versions de ressources

- Plusieurs versions d'une même ressource peuvent être proposées dans des répertoire distincts suffixés par une spécification de version
  - Par exemple, où stocker les images pour les japanophones français sur un GPS Android à grand écran utilisé la nuit ? Dans le répertoire res/drawable-ja-rFR-xlarge-car-night

Critères de version utilisés

1. Mobile Country Code et Mobile Network Code (MCC et MNC) :
   mcc208-mnc00 (réseau Orange en France), mcc310 (réseau aux USA)...
2. Langue et région : en, fr, fr-rCA...
3. Direction d'agencement : ldltr (direction de gauche à droite), ldrtl (de droite à gauche)
4. Min(largeur, hauteur) : sw<N>dp (N en pixels)
5. Largeur d'écran : w<N>dp
6. Hauteur d'écran : h<N>dp
7. Taille d'écran : parmi smal, normal, large et xlarge
8. Ratio largeur/hauteur : long, notlong
9. Orientation de longueur(peut changer) : port, land
10. Mode d'interface : car, desk, television, appliance
11. Mode nuit : night, notnight
12. Densité de l'écran : ldpi, mdpi, hdpi, xhdpi, nodpi, tvdpi
13. Tactilité : notouch, finger, stylus
14. Disponibilité du clavier : keysexposed, keyshidden, keyssoft
15. Clavier physique : nokeys, qwerty, 12key
16. Touches de navigation : navexposed, navhidden
17. Méthode primaire de navigation non-tactile : nonav, dpad, trackball, wheel
18. Version de plate-forme : vN (e.g. v17 pour Android 4.2)

View

- View : classe ancêtre de tous les composants graphiques
- Gestion de l'affichage et des évènements d'une zone sur l'écran
- ViewGroup : peut contenir des vues enfants (arbre de vues)
- Arbre de vues statique définissable en XML dans une ressource layout
- Package android.widget : vues et groupes de vues prédéfinis pour des usages courants

Implantations élémentaires de View

- Éléments de formulaire
  - *TextView* : affiche une chaîne
  - *EditText* : permet la saisie d'une chaîne (propriété inputType pour le type d'entrée attendu)
  - *Button* : bouton cliquable, variante de type interrupteur avec ToggleButton
  - *CheckBox* : case à cocher
  - *RadioButton* : bouton radio regroupable dans un RadioGroup
  - *CheckedTextView* : chaîne cochable (implante Checkable)
  - *ProgressBar* : barre de progression (horizontale, circulaire), variante avec étoiles de notation avec *RatingBar*
  - *SeekBar* : barre de réglage
  - *SearchView* : champ de recherche avec proposition de suggestions
- Éléments multimédias
  - *ImageView* : affichage d'une ressource image
  - *ImageButton* : bouton avec image
  - *VideoView* : affichage contrôlable de vidéo

Gestion d'évènements

- Rendu graphique dans la thread principale
  - Ne pas appeler des méthodes graphiques depuis une autre thread
  - Ne pas réaliser des calculs longs dans la thread principale (utiliser un Worker)
- Traitement des événements entrants
  - Dispatch de l’événement à la vue responsable qui traite l'évènement et notifie les listeners
  - Si les limites de la vue doivent être changés → *requestLayout()*
  - Si l'apparence de la vue doit être changée → *invalidate()*
  - Recalcul de l'agencement et redessin si nécessaire d'une portion de l'arbre des vues

Listeners d'évènements

- Objectif : associer une action à réaliser lors de la survenue d'un événement
- Moyens possibles :
  - Enregistrement d'un listener avec setOn*Event*Listener(EventListener)
  - Utilisation de la propriété « android:on*Event* » de la vue définie dans le layout XML de l'activité
  - Redéfinition de la méthode on*Event*() du composant (généralement non conseillé)
- Interception globale d'évènements (utile pour débuggage) :
  - *Activity.dispatchXEvent(MotionEvent)* : dispatch de l'évènement de l'activité vers la vue concernée (X={GenericMotion, Key, KeyShortcut, PopulateAccessiblity, Touch, Trackball})
  - *ViewGroup.onInterceptXEvent(MotionEvent)* et *ViewParent.requestDisallowInterceptXEvent(MotionEvent)* : vol d'évènement par la vue parent (X={Touch,Hover})

Évènements courants

- *void onClick(View)* : clic tactile, par trackball ou validation
- *boolean onLongClick(View)* : clic long (1s)
- *void onFocusChange(View v, boolean hasFocus)* : gain ou perte de focus
- *boolean onKey(View v, int keyCode, KeyEvent e)* : appui sur une touche matérielle
- *boolean onTouch(View v, MotionEvent e)* : dispatch d'un événement de touché (appelé avant le transfert de l'évènement à la vue enfant concernée). Les gestures peuvent être composées de plusieurs MotionEvent.
- Valeur de retour boolean : permet d'indiquer si l'évènement a été consommé, i.e. S'il ne doit plus être communiqué à d'autres listeners (de vues enfant) ou si la fin d'un événement composé ne doit pas être envoyée.

Évènements de touché

```
view.setOnTouchListener(new OnTouchListener() {
    @Override public boolean onTouch(View v, MotionEvent e)
    {
        switch (e.getAction() & MotionEvent.ACTION_MASK)
        {
            case MotionEvent.ACTION_DOWN :
                // Starting a new touch action (first finger pressed)
                // Coordinates of the starting point can be fetched with e.getX() and e.getY()
                // The first finger has the index 0
            case MotionEvent.ACTION_POINTER_DOWN :
                // A new finger is pressed
                // Its identifier can be obtained with e.getPointerId(e.getActionIndex())
            case MotionEvent.ACTION_MOVE :
                // One or several fingers are moved
                // Their coordinates can be obtained with e.getX(int index) and e.getY(int index)
                // e.getPointerCount() specifies the number of implied fingers
                // e.getPointerId(int) converts a pointer index to a universal id
                // that can be tracked across events
                // e.findPointerIndex(int) does the reverse operation
            case MotionEvent.ACTION_POINTER_UP :
                // A finger has been raised
                // Its identifier is e.getPointerId(e.getActionIndex())
            case MotionEvent.ACTION_UP :
                // The last finger has been raised
        }
    }
```

Reconnaissance de gestures

- Un détecteur de gestes reçoit les événements de touché (par *onTouchEvent(MotionEvent e)* et appelle les méthodes du listener enregistré lors de la détection de gestes
- Reconnaissance de gestures simples avec *GestureDetector* :
  - onDown, onDoubleTap, onLongPress, onFling, onScroll, onShowPress, onSingleTapConfirmed, onSingleTapUp...
- Reconnaissance de gestures zoom avec *ScaleGestureDetector* :
  - onScaleBegin, onScale, onScaleEnd
- Réception de gestures complexes avec *GestureOverlayView* (vue transparente)
  - onGesturePerformed(GestureOverlayView overlay, Gesture gesture) permet de récupérer le geste et de le comparer à une *GestureLibrary* (méthode recognize()) qui propose des candidats avec score de confiance

Dessin d'une vue

- Lorsqu'une région devient invalide, elle doit être redessinée (peut être forcé avec *View.invalidate()*)
- L'arbre de vue est parcouru en profondeur pour trouver les vues intersectant la région invalide
- Le rendu d'une vue est implanté dans *View.onDraw(Canvas c)* ; le canevas communiqué contient les bornes de la région à redessiner récupérable avec *c.getClipBounds()*

Canvas

- Fournit un API pour le dessin 2D
- Primitives de dessin *draw*()* utilisables directement pour dessiner sur le Bitmap sous-jacent : *drawRect()*, *drawText()*, *drawLine()*, *drawBitmap()*...
- Dernier argument de type *Paint* pour les méthodes *draw*()* : paramètres pour le dessin (couleur, fonte, anti-aliasing...)
- Délégation du dessin à un *Drawable* : appel de *Drawable.draw(Canvas)*

Drawable

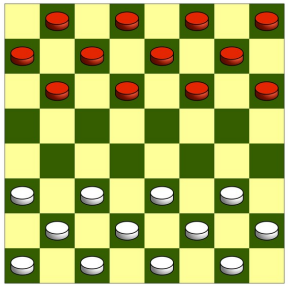
- Définit des données vectorielles (très basiques) ou bitmap pouvant être dessinées : {*Bitmap*, *Layer*, *NinePatch*, *Picture*, *Shape*}*Drawable*
- Méthodes intéressantes :
  - Taille préférée : *getInstrinsic*{*Width*, *Height*}()
  - Avant dessin, fixation des bornes : *setBounds(Rect r)*
  - Dessin avec *Drawable.draw(Canvas)*
  - Récupérable d'une ressource avec *Ressources.getDrawable(int)*
- Support du SVG non-natif (bibliothèque externe nécessaire telle que svg-android)

Exemple : un bel échiquier en XML

- Ajout de *res/drawable/checkersboard.png*
- Ajout des pièces : *res/drawable/{blackpawn, whitepawn}*.png
- Ajout de *res/drawable/completeboard.xml* :

```
<layer-list>
    <item android:drawable="@drawable/chessboard.png" />
    <item android:drawable="@drawable/blackpawn" top="0" left="0" />
    <item android:drawable="@drawable/blackpawn" top="0" left="20" />
            …
</layer-list>
```

- Finalement, il est plus pratique d'ajouter les pièces programmatiquement au sein d'une boucle ;)



Gestion du focus

- Element focusable : *isFocusable()*, *isFocusableInTouchMode()* (changement de l'état avec setter) ; *hasFocusable()* pour test également sur descendants
- Trouver le prochaine vue focusable : *View.focusSearch(View.FOCUS_{UP,DOWN,LEFT,RIGHT})*
- Possibilité de changer l'ordre de focus par défaut avec propriétés XML : *nextFocus*{*Down*, *Left*, *Right*, *Up*}
- Demande dynamique de focus : *View.requestFocus()*, *View.requestFocusFromTouch()*

ViewGroup

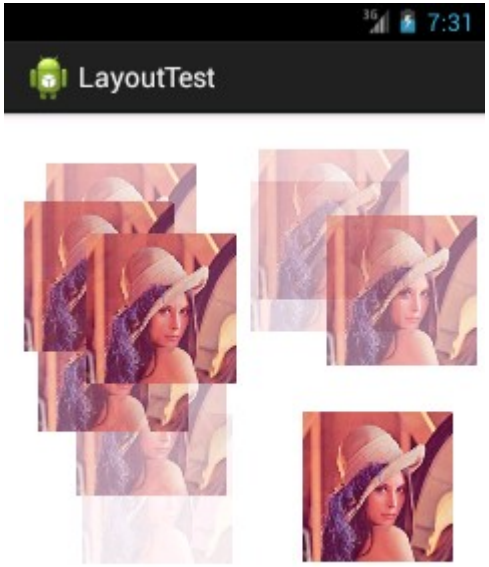
- Container de vues enfants, gère leur agencement
- Création d'un nouveau ViewGroup par héritage
  - Redéfinition de *onLayout()* nécessaire
  - Création d'une classe interne dérivée de *ViewGroup.LayoutParams* pour les paramètres d'agencement de chaque vue enfant (facultatif)
- Deux types de ViewGroup
  - Destinés à l'agencement statique (**Layout*) :
    - Il est conseillé de définir l'arbre de vue par une ressource layout XML
    - Manipulation des enfants possible également à l'exécution :
      - Ajout d'un enfant avec *ViewGroup.addView()*, suppression avec *ViewGroup.removeView()*
      - Parcours de la liste des enfants avec *ViewGroup.getChildAt()* et *ViewGroup.getChildCount()*
  - Destinés à l'agencement dynamique et basés sur un Adapter (modèle) ; efficaces pour afficher un grand nombre d'éléments obtenus à l'exécution avec recyclage des vues

Agencement de vues

- Deux parcours en profondeur de l'arbre des vues
  - Appel récursif de *View.measure(int width, int height)* pour que les vues enfants calculent leurs dimensions souhaitées
    View.measure n'est pas redéfinissable, il faut implanter *onMeasure(int, int)* qui reçoit les dimensions souhaitées par le parent (peuvent être indicatives ou strictes) et déclare les dimensions souhaitées par la vue avec *setMeasuredDimension(int, int)*. Un parent peut appeler plusieurs fois *child.measure* avec différentes hypothèses de dimension pour l'enfant.
  - Appel récursif de *View.layout(int l, int t, int r, int b)* pour positionner les vues enfants dans le cadre spécifié. *View.layout* appelle *View.onLayout(boolean, int, int, int, int)* qui doit être redéfini ; la vue appelle pour chaque enfant *child.set{Left Right, Top, Bottom}(int)* en s'aidant de *child.getMeasured{Width, Height}()*.

Un agencement aléatoire

```
public class RandomLayout extends ViewGroup
{
        public static final int MAX_CHILDREN = 1024;
        public static final String TAG = "RandomLayout";

        private final int[] positions = new int[MAX_CHILDREN * 2];

        public RandomLayout(Context context, Random rng)
        {
                super(context);
                for (int i = 0; i < positions.length; i++) positions[i] = rng.nextInt(Integer.MAX_VALUE);
        }

        public RandomLayout(Context context) { this(context, new Random()); }

        /** Method computing the position of an element */
        private void getPosition(int element, int pl, int pt, int pr, int pb, int[] pos)
        {
                View v = getChildAt(element);
                int pw = pr - pl - v.getMeasuredWidth(), ph = pb - pt - v.getMeasuredHeight();
                int l = (int)(pl + ((long)positions[element * 2] * pw / Integer.MAX_VALUE));
                int t = (int)(pt + ((long)positions[element * 2 + 1] * ph / Integer.MAX_VALUE));
                pos[0] = l; pos[1] = t; pos[2] = l + v.getMeasuredWidth(); pos[3] = t + v.getMeasuredHeight();
        }

        @Override
        protected void onMeasure(int w, int h)
        {
                int width = MeasureSpec.getSize(w); int height = MeasureSpec.getSize(h);
                Log.i(TAG, String.format("Measuring for dimensions %d:%d", width, height));
                final int count = Math.min(getChildCount(), MAX_CHILDREN);
                final double countRoot = Math.sqrt(count);
                final int computedWidth = (int)(width / countRoot), computedHeight = (int)(height / countRoot);
                // Measure the children
                for (int i = 0; i < count; i++)
                {
                        final View v = getChildAt(i);
                        int widthMes = MeasureSpec.makeMeasureSpec(computedWidth, MeasureSpec.AT_MOST);
                        int heightMes = MeasureSpec.makeMeasureSpec(computedHeight, MeasureSpec.AT_MOST);
                        measureChild(v, widthMes, heightMes);
                }
                setMeasuredDimension(width, height);
        }

        @TargetApi(Build.VERSION_CODES.HONEYCOMB)
        @Override
        protected void onLayout(boolean changed, int l, int t, int r, int b)
        {
                Log.i(TAG, String.format("Layout %d:%d:%d:%d", l, t, r, b));
                // Set the position of the current ViewGroup
                setLeft(l); setTop(t); setRight(r); setBottom(b);
                // Layout the children
                final int count = Math.min(getChildCount(), MAX_CHILDREN);
                final int[] pos = new int[4];
                for (int i = 0; i < count; i++)
                {
                        final View v = getChildAt(i);
                        getPosition(i, l, t, r, b, pos);
                        v.layout(pos[0], pos[1], pos[2], pos[3]);
                }
        }
}
```



FrameLayout

- Affichage d'une pile de vues avec gestion basique d'agencement :
  - Paramètre d'agencement : *FrameLayout.LayoutParams(int width, int height, int gravity)*
  - *gravity* définit l'emplacement de la vue enfant (top, bottom, left, right, center, fill...)

LinearLayout

- Agencement des enfants dans une direction unique
- Propriété d'orientation : *LinearLayout.*{*HORIZONTAL*, *VERTICAL*}
- Propriétés de *LinearLayout.LayoutParams* :
  - *layout_height*, *layout_width* : dimensions réclamées pour la vue enfant
  - *layout_gravity* : alignement dans la cellule
  - *layout_weight* : priorité pour l'agrandissement ou le rétrécissement de la vue enfant (distribution de la différence d'espace proportionnelle au poids)

RelativeLayout

- Positionnement des enfants relativement les uns par rapport aux autres
- RelativeLayout.LayoutParams
  - Propriétés d'alignement par rapport au parent : *layout_alignParent*{*Bottom*, *End*, *Left*, *Right*, *Start*, *Top*}, *layout_center*{*Horizontal*, *InParent*, *Vertical*}
  - Propriétés d'alignement relatives à un frère : *layout_align*{*Baseline*, *Bottom*, *End*, *Left*, *Right*, *Start*, *Top*}, to{*End*, *Left*, *Right*, *Start*}*Of*

TableLayout

- Tableau d'agencement de vues ligne *TableRow* (philosophiquement similaire aux <table> <tr> <td> en HTML)
- *TableRow* dérivé de *LinearLayout* avec alignement automatique des colonnes sur chaque ligne
- Propriétés de *TableRow.LayoutParams* :
  - *layout_column* : indice de départ de colonne (à partir de 0)
  - *layout_span* : nombre de colonnes occupées

GridLayout

- Agencement sur une grille rectangulaire de N colonnes ; contrairement au *TableLayout*, les composants sont ajoutés directement avec leur paramètres d'agencement
- Propriétés de *GridLayout.LayoutParams* :
  - *layout_column* et *layout_columnSpan* : colonne de départ et nombre de colonnes occupées
  - *layout_row* et *layout_rowSpan* : ligne de départ et nombre de lignes occupées
  - *layout_gravity* : emplacement de la vue enfant dans la cellule

Vues basés sur modèle

- Classes dérivées de *AdapterView*
- *Adapter<T>* est un modèle permettant d'obtenir les vues enfants à afficher
  - *getView(int i, View convertView, ViewGroup parent)* permet d'obtenir la vue à afficher pour l'enfant #i (implantations par défaut utilisant un *TextView*)
  - Adapteurs pré-définis :
    - *ArrayAdapter<T>* : stockage d'une liste d'éléments
    - *SimpleCursorAdapter* : modèle adaptant les données d'un Cursor de BDD

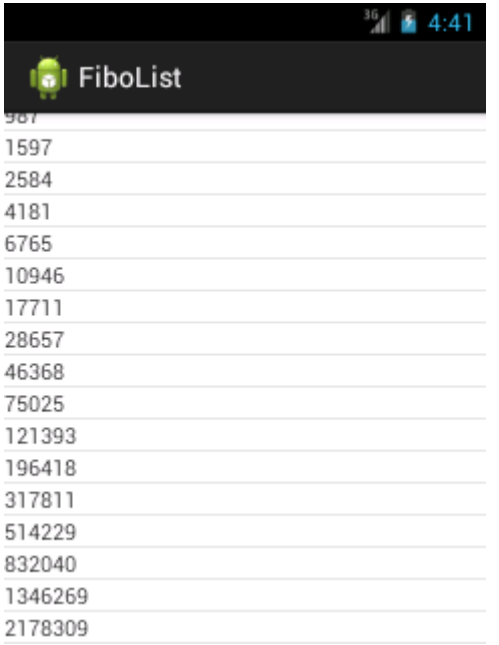
ListView et GridView (AbsListView)

- *ListView* : affichage d'une liste avec défilement vertical puisant ses éléments dans un *ListAdapter* (classe dérivée avec liste à deux niveaux : *ExpandableListView*)
- *GridView* : affichage sur une grille avec défilement vertical Propriétés intéressantes : *columnWidth*, *gravity* (alignement de l'enfant dans la cellule), *numColumns*, *stretchMode*
- Possibilité d'agir sur les évènements de défilement : *setOnScrollListener(ScrollListener)*

ListView : exemple

```
public class FiboList extends Activity
{
	public static final int BATCH_SIZE = 10;

	private ArrayList<Long> list;

	@Override
	protected void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		FrameLayout l = new FrameLayout(this);
		setContentView(l);
		list = new ArrayList<Long>();
		// Create an adapter backed on the list
		ListView listView = new ListView(this);
		l.addView(listView);
		final ArrayAdapter<Long> adapter = new ArrayAdapter<Long>(this, R.layout.simpletextview, list);
		listView.setAdapter(adapter);
		// Create a scroll listener to load the numbers as the user scroll
		listView.setOnScrollListener(new OnScrollListener() {

			@Override
			public void onScrollStateChanged(AbsListView view, int scrollState)
			{
			}

			@Override
			public void onScroll(AbsListView view, int firstVisibleItem,
					int visibleItemCount, int totalItemCount)
			{
				// If we have scroll to the end of the list
				if (firstVisibleItem + visibleItemCount == totalItemCount)
				{
					// We must add new Fibonacci numbers
					generateNumbers(BATCH_SIZE);
					adapter.notifyDataSetChanged(); // The model must inform the view of the change
				}
			}
		});
		generateNumbers(BATCH_SIZE);
	}
...
```

```
private void generateNumbers(int numberToAppend)
{
	int size = list.size();
	for (int i = 0; i < numberToAppend; i++)
	{
		long v = ((size > 0)?list.get(size-1):1)
			+ ((size > 1)?list.get(size-2):1);
		list.add(v);
		size++;
	}
}

}
```



Spinner

- Champ de texte avec menu déroulant pour choix d'un item unique
- Éléments de choix fournis par un *SpinnerAdapter* (interface implantée par *ArrayAdapter*)
- Réaction à l'élément choisi avec un *onItemSelectedListener* :

```
spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {
                // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
         }

public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
}
}
```

ViewPager (version bêta)

- Maintient une liste de vues parcourable par un geste de swipe
- Les vues à afficher sont fournies par un *ViewAdapter* (*ViewPager.setViewAdapter()*) qui est responsable de l'ajout et de la suppression d'éléments dans le *ViewPager* :
  - Object instantiateItem(ViewGroup parent, int position)
  - void destroyItem(ViewGroup parent, int pos, Object o)
  - int getCount()
  - boolean isViewFromObject(View view, Object o)

Widgets composés à usage spécifique

- Pour formulaire : facilite l'entrée de données communes dans des formulaires et permet la validation
  - *TimePicker*, *DatePicker* : choix d'horaire et de date
  - *CalendarView* : affiche un calendrier avec date sélectionnable
  - *NumberPicker* : sélection d'un entier dans un intervalle avec incrémentation et décrémentation
  - *DialerFilter* : permet de saisir des chiffres/lettres avec un clavier numérique de téléphone
- Pour activité multimédia :
  - *MediaController* : offre des boutons de contrôle pour une vidéo (avec *VideoView* par exemple)
  - *ZoomControls* : bouton de zoom/dezoom

WebView

- Vue d'affichage de contenu HTML avec la bibliothèque de rendu WebKit
- Permet d'embarquer un véritable navigateur web dans une application ; bridage possible depuis *WebView.getSettings()* :
  - *setJavaScriptEnabled(false)* pour désactiver JavaScript
  - *setPluginState(PluginState.OFF)* pour désactiver les plugins
  - *setAllowContentAccess(false)* pour désactiver le chargement d'URL
  - *setAllowFileAccess(false)* pour désactiver le chargement de fichiers locaux
- Utilisation :
  - *loadData*(String data, String mimeType, String encoding) : affiche un contenu directement spécifié
  - *loadUrl*(String url) : charge le contenu à l'URL spécifiée
- Utile pour créer une application Android/HTML5/JavaScript hybride : communication d'un objet Java à un script JavaScript avec *addJavaScriptInterface(Object object, String name)*
- Interception des clics sur liens hypertextes en créant une classe dérivée redéfinissant *shouldOverrideUrlLoading()*

Menu

- Types de menu :
  - Menu d'options : obtenu depuis un bouton physique ou la barre d'action ; y résident les actions principales de l'application (navigation, paramétrage, recherche...)
  - Menu contextuel ou popup : obtenu depuis un clic long sur un élément de vue pour proposer des choix spécifiques
- Événement de sélection d'item de menu : *onMenuItemClick(MenuItem item)* ; il faut tester l'item sélectionné avec un switch (en utilisant par exemple *item.getItemId()*)

Ressource XML de menu

Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_search"
          android:icon="@drawable/menu_search"
          android:title="@string/menu_search"
          android:showAsAction="ifRoom|collapseActionView"
          android:actionViewClass="android.widget.SearchView" />
    <!-- We define a menu group -->
    <group android:id="@+id/group_temporal">
        <item android:id="@+id/menu_start"
              android:title="@string/menu_start" />
        <item android:id="@+id/menu_stop"
              android:title="@string/menu_stop" />
    </group>
</menu>
```

ActionBar

- Barre d'action en haut d'écran apparue depuis Android 3.0 (API11), récupérable avec *Activity.getActionBar()*
- Permet la navigation dans une application, son paramétrage ainsi que de lancer des actions
- Barre peuplable en redéfinissant *Activity.onCreateOptionsMenu(Menu menu)*
- Placement des éléments de la barre d'action :
  - Éléments de navigation en haut à gauche
  - Items d'actions en haut à droite (en bas sur petits écrans si propriété d'activité uiOptions="splitActionBarWhenNarrow")
  - Si trop d'items d'action : masquage dans un menu popup overflow

Elements de navigation

- Retour "top" (logo haut gauche)
  remplaçable par une icône haut avec actionBar.setDisplayHomeAsUpEnabled(true);
  - Clic récupérable avec *onOptionsItemSelected()* avec l'ID *android.R.id.home*
  - Utile pour revenir à l'activité principale :
    Intent intent = new Intent(this, HomeActivity.class); intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(intent);
- Onglets
  - Instantiation d'un *ActionBar.Tab*, *setText(...)*, *setIcon(...)*
  - Création d'un *TabListener* (code exécuté lors du choix du tab, typiquement chargement de Fragment) et enregistrement avec *Tab.setTabListener()*
  - Ajout dans la Bar avec *Bar.addTab(Tab tab)*

Items d'action

- Vue pour un item
  - Classiquement : icône + texte
  - Choix d'une vue personnalisée avec *android:actionViewClass*
- Affichage dans l'ActionBar configurable avec *android:showAsAction* (*ifRoom*, *withText*, *never*, *collapseActionView*)
- Certaines vues (comme *SearchView*) proposent deux versions (réduites et étendue) avec *CollapsibleActionView*
- *ActionProvider* : définition d'item avec action prédéfinie (e.g. *ShareActionProvider* pour partager un élément de différentes façons); propriété *android:actionProviderClass*

Menu contextuel

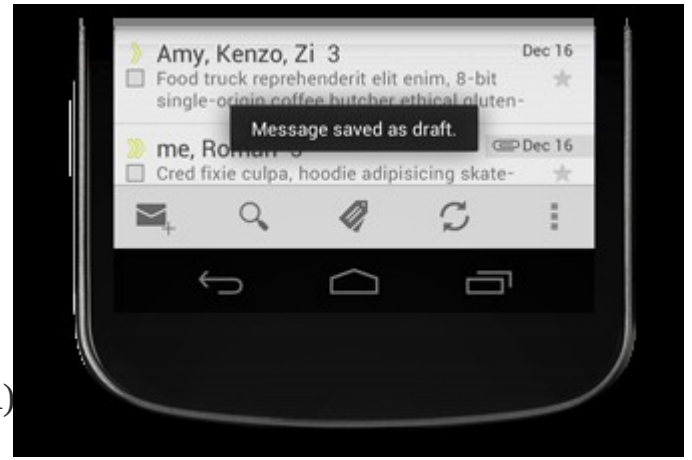
- Propose un choix d'action sur un élément d'UI ; typiquement affiché par un clic long
- Usage classique :
  - Création d'un *ActionMode.CallBack* avec :
    - *boolean onCreationActionMode(ActionMode mode, Menu menu)* : on peuple le menu (par exemple avec un MenuInflater depuis une ressource)
    - *boolean onPrepareActionMode(ActionMode mode, Menu menu)*
    - *boolean onActionItemClicked(ActionMode mode, MenuItem item)* : on peut éventuellement sortir du mode menu avec mode.finish()
    - *void onDestroyActionMode(ActionMode mode)* : lorsque l'on sort du menu
  - Enregistrement d'un listener su un clic long

```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    public boolean onLongClick(View view) {
        if (mActionMode != null) return false ;
        mActionMode = getActivity().startActionMode(callback);
        return true;
    }
});
```

Toast

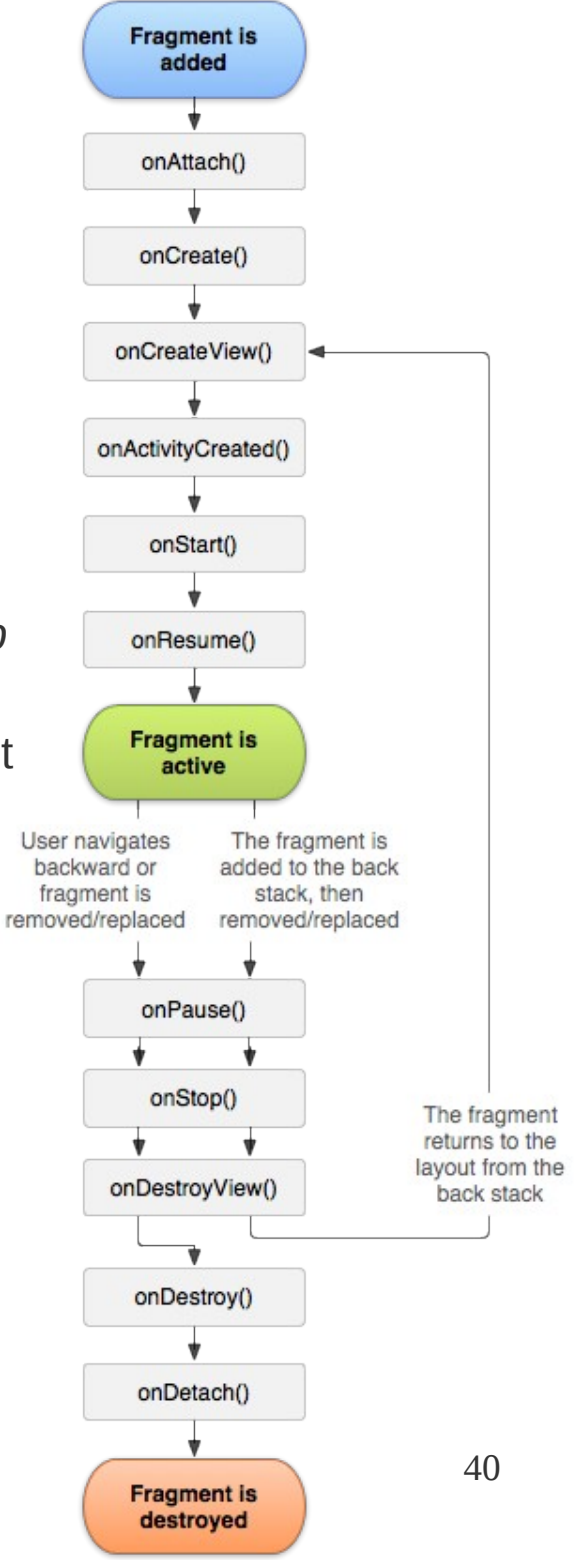
http://developer.android.com/guide/topics/ui/notifiers/toasts.html

- Affiche un popup pour informer l'utilisateur
- Utilisation simple :
  Toast.makeText(Context context, String message, int duration)
  Constantes pour duration :
  Toast.LENGTH_LONG et
  Toast.LENGTH_SHORT



Fragments

- Fragment ~ sous-activité introduite par Android 3.0 (fonctionnement similaire à *Activity* avec cycle de vie)
- Intérêt : découper un écran en plusieurs parties --> flexibilité d'affichage matériel-dépendante
- Redéfinition de *View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)* si le fragment créé une UI
- Gestion avec *FragmentManager* ; fragment ajoutable dynamiquement ou par déclaration dans le layout XML de l'activité parente



FragmentManager

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Création de transaction

Modifications lors d'une transaction :

- *FragmentTransaction add(int containerViewId, Fragment fragment, String tag)*
- *FragmentTransaction remove(Fragment fragment)*
- *FragmentTransaction replace(int containerViewId, Fragment fragment, String tag)*

Gestion de la pile lors d'une transaction (listener *addOnBackStackChangedListener()*) :

- *FragmentTransaction addToBackStack(String shortcut)*
- *FragmentTransaction void popBackStack()*

À la recherche du fragment perdu :

- *Fragment findFragmentById(int id)* : pour un fragment graphique
- *Fragment findFragmentByTag(String tag)* : en utilisant le tag employé pour add

```
// Commit the transaction
transaction.commit();
```

Validation de transaction

Adaptation au matériel avec les Fragments

- Principe :
  - afficher plusieurs fragments sur les grands écrans
  - afficher un seul fragment sur les petits écrans
- On créé plusieurs layouts adaptés au matériel dans *res/layout-X/mylayout.xml* (X=caractéristiques matériel) :
  - un layout avec plusieurs containers de fragments
  - un layout avec un seul container pour un fragment
- On vérifie le layout à l'exécution en testant la présence d'un container : boolean extendedView = getActivity().findViewById(R.id.additionalContainer) != null
- On adapte le comportement d'affichage en fonction de extendedView :
  - extendedView != null : mise à jour du fragment dans additionalContainer
  - extendedView == null : lancement d'une nouvelle activité avec le fragment

OpenGL

- Deux versions de l'API
  - OpenGL ES 1
  - OpenGL ES 2
- Vue OpenGL : GLSurfaceView (à créer dans le onCreate() de l'activité)
- Affectation d'un renderer : GLSurfaceView.setRenderer(GLSurfaceView.Renderer)
  Un nouveau GLSurfaceView.Renderer doit être declaré avec redéfinition des méthodes (le 1er paramètre n'est pas utilisé) :
  - void onSurfaceCreated(GL10 gl, EGLConfig config) : pour l'initialisation de l'environnement
  - void onDrawFrame(GL10 gl) : pour dessiner chaque frame
  - void onSurfaceChanged(GL10 gl, int width, int height) : appelé lorsque la vue est redimensionnée
- Déclaration de l'utilisation de l'API dans AndroidManifest.xml nécessaire