Legond-Aubry Fabrice
fabrice.legond-aubry@u-paris10.fr

PPM(A)

Programmation sur Plateformes Mobiles (Android)

Section:

PLAN

Fondamentaux & API

Buts

- Favoriser la réutilisation d'applications
  - Ne pas réinventer la roue ...
  - Ré - utiliser des morceaux d'application ...
- Types de composants :
  - Activity : Interface compréhensible pour un utilisateur et son code associé
  - Fragment (android >3.x): Souplesse de manipulation des interfaces (ce sont des morceaux d'interfaces)
  - Service : Tâche de fond (musique, serveur, synchronisation ...). Pas pour la MainActivity.
  - Widget : Programmes « visuels » (ie gadget windows)

Partage de données / Communications / Persistance

- Intra-Application (VOIR PLUS LOIN)
  - Bus de communication (Intent)
  - Singleton, variables statiques
  - La plus part des techniques inter-applications (LENTES !!!)
- Inter-Application (VOIR PLUS LOIN)
  - Bus de communication (Intent)
  - Fichiers
  - Bases SQLite (partagées)
  - Serveurs externes
- Persistance
  - Fichiers
  - Base de données SQL
  - Préférences

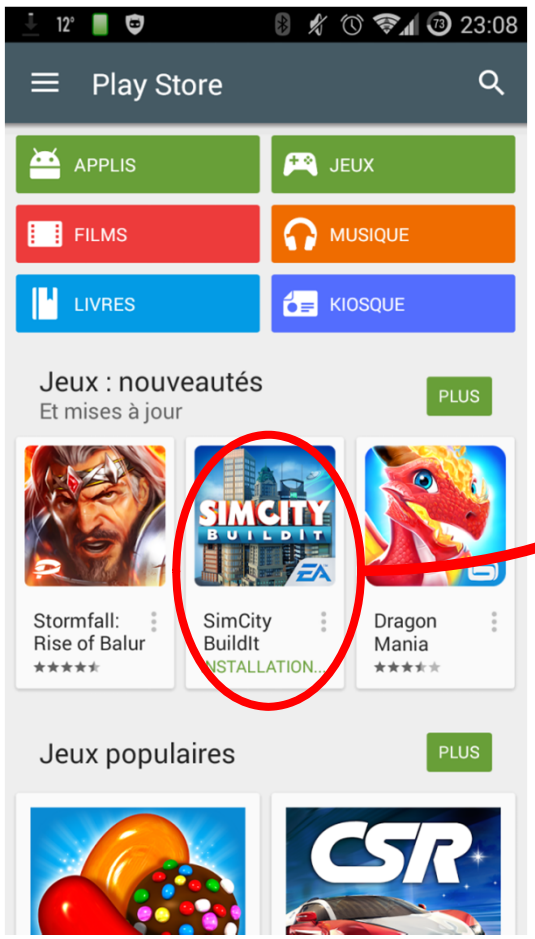
Communications

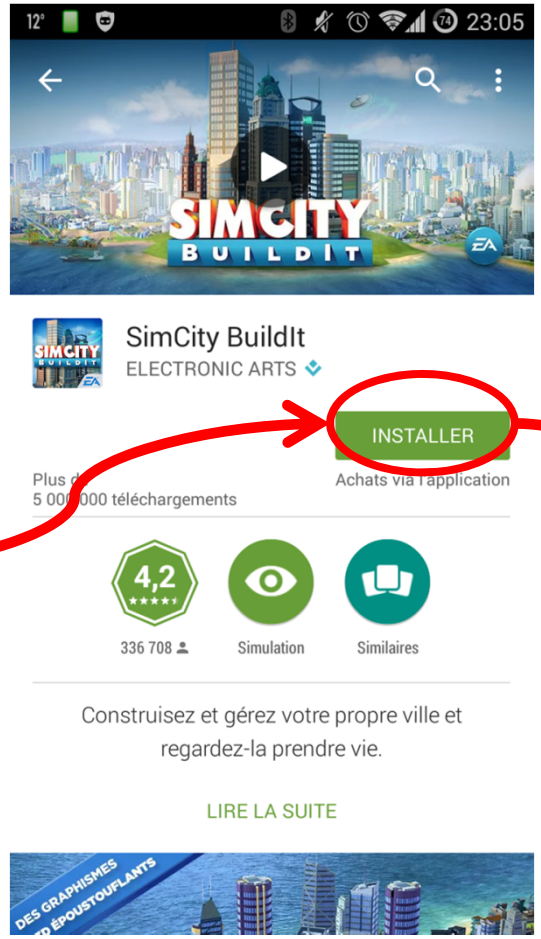
- Faciliter la communication inter-application via les Intents ...
  - Bus de communication. Publish / Subscribe.
  - Intent (le message) : Envoie de messages (action).
  - sendBroadcast(envoie)
  - BroadcastReceiver (réception) : Gestion des annonces (batterie, fin de téléchargement ...) générées par l'OS.
- Pour la communication intra-application …
  - LocalBroadcastReceiver (réception) : messages du développeur uniquement
- Faciliter la communication C/S (problèmes de confidentialité)
  - Internet (Gestion automatique 2G/3G/4G/Wifi)
  - Notifications Push - Google Cloud Messaging / MQTT

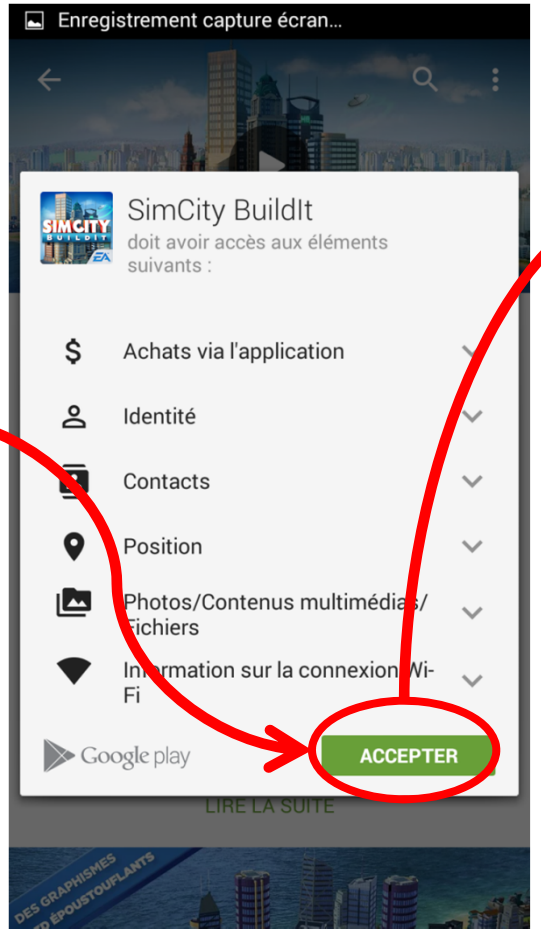
    https://developer.android.com/google/gcm/index.html
  - Authentification aux services Google

Un Exemple

Programmation Evenemmentielle







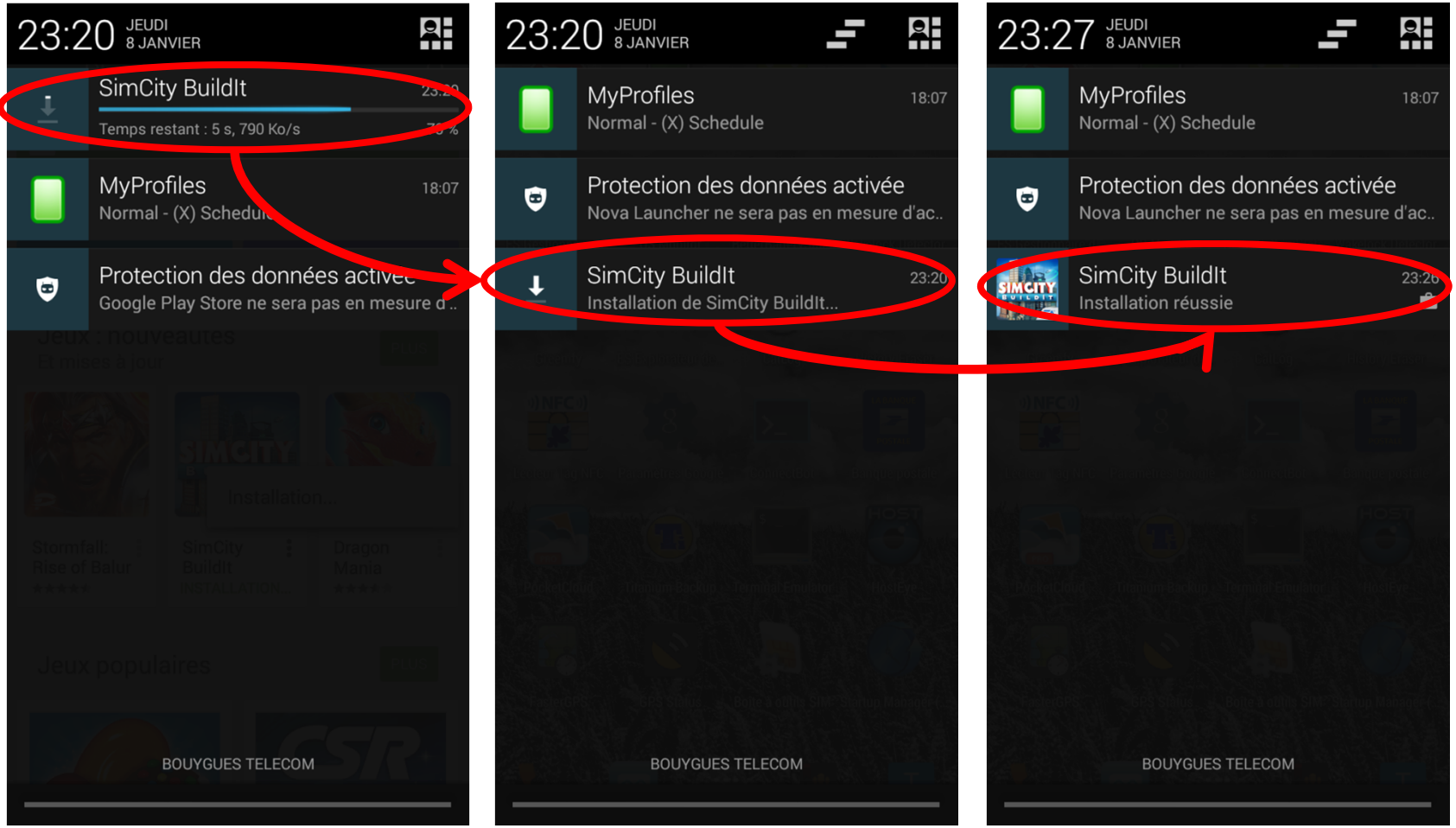
1 Activity

1 Activity

1 Dialog

Un Exemple (suite)

Programmation Evenemmentielle



1 Service | 1 Service | 1 Notification

Un Exemple

- Point de vue du développeur de l'application
  - Activity (Activité) :
    - 1 Affichage de la liste des application
    - 1 Affichage du détail d'une application
  - 1 boite de dialogue (dialog box)
  - BroadcastReceiver
    - Détecter la fin d'un téléchargement d'application
  - Service :
    - 1 Service pour télécharger l'application
    - 1 Service pour installer l'application
  - Gestion
    - des notifications

Définition

Cycle de vie

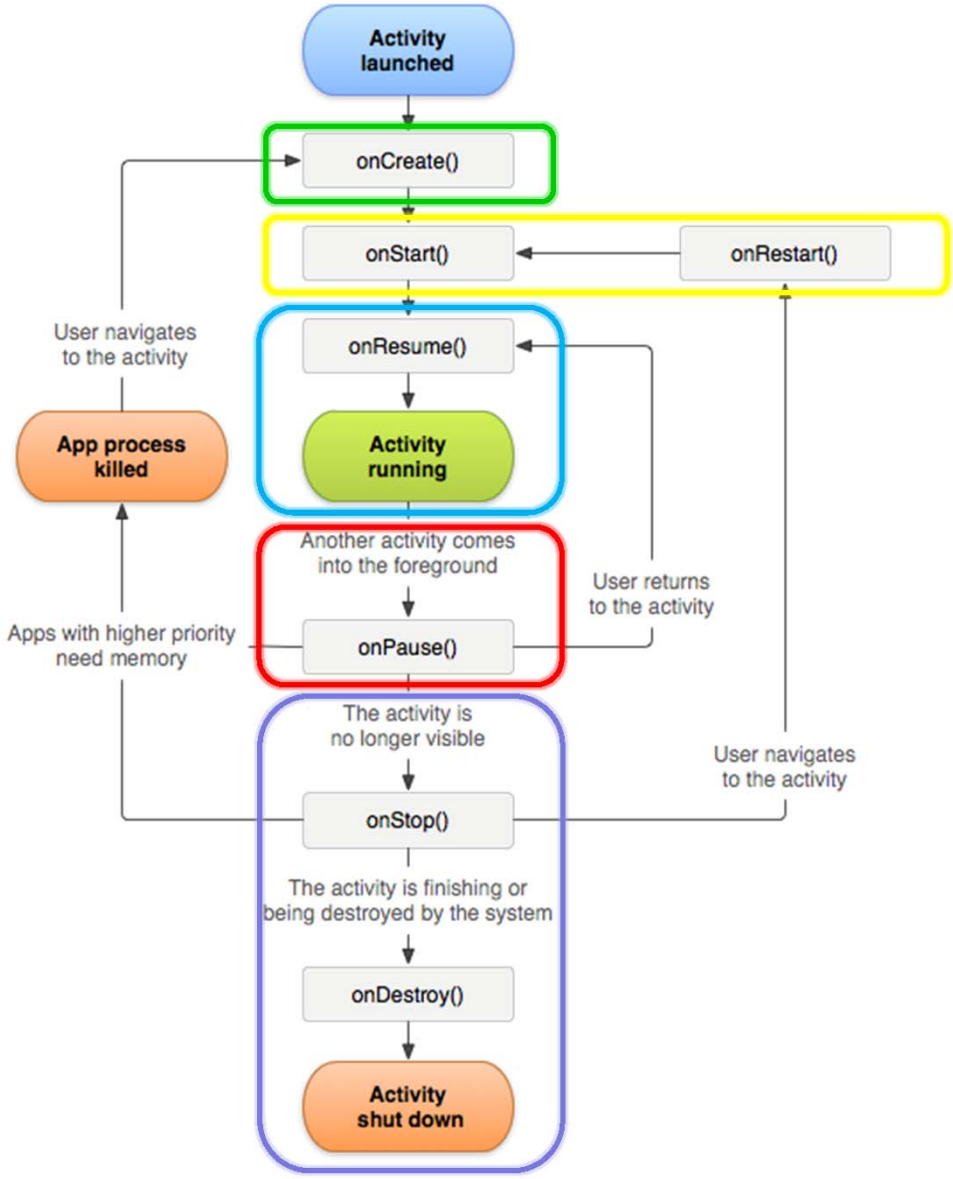
Evèments / Callback

Activity (Activité) : Définition

- Elément visuel principal d'une application
  - Une entité visuelle complète pour un « utilisateur » : un écran
  - Divisée en deux parties distinctes :
    - La vue : ensemble des éléments graphiques à l'écran
    - La gestion de la vue et des événements
      - partie Contrôleur du MVC
  - Une Application est un ensemble d'activités
  - Une activité peut lancer une ou plusieurs autre(s) activité(s)
  - On peut changer les propriétés d'une activité en modifiant le fichier xml de l'activité
    https://developer.android.com/guide/topics/manifest/activity-element.html
- 1 Thread UI :
  - Gestion du rafraîchissement
  - Gestion des événements
  - SEUL LE THREAD UI PEUT MODIFIER LES ELEMENTS GRAPHIQUES

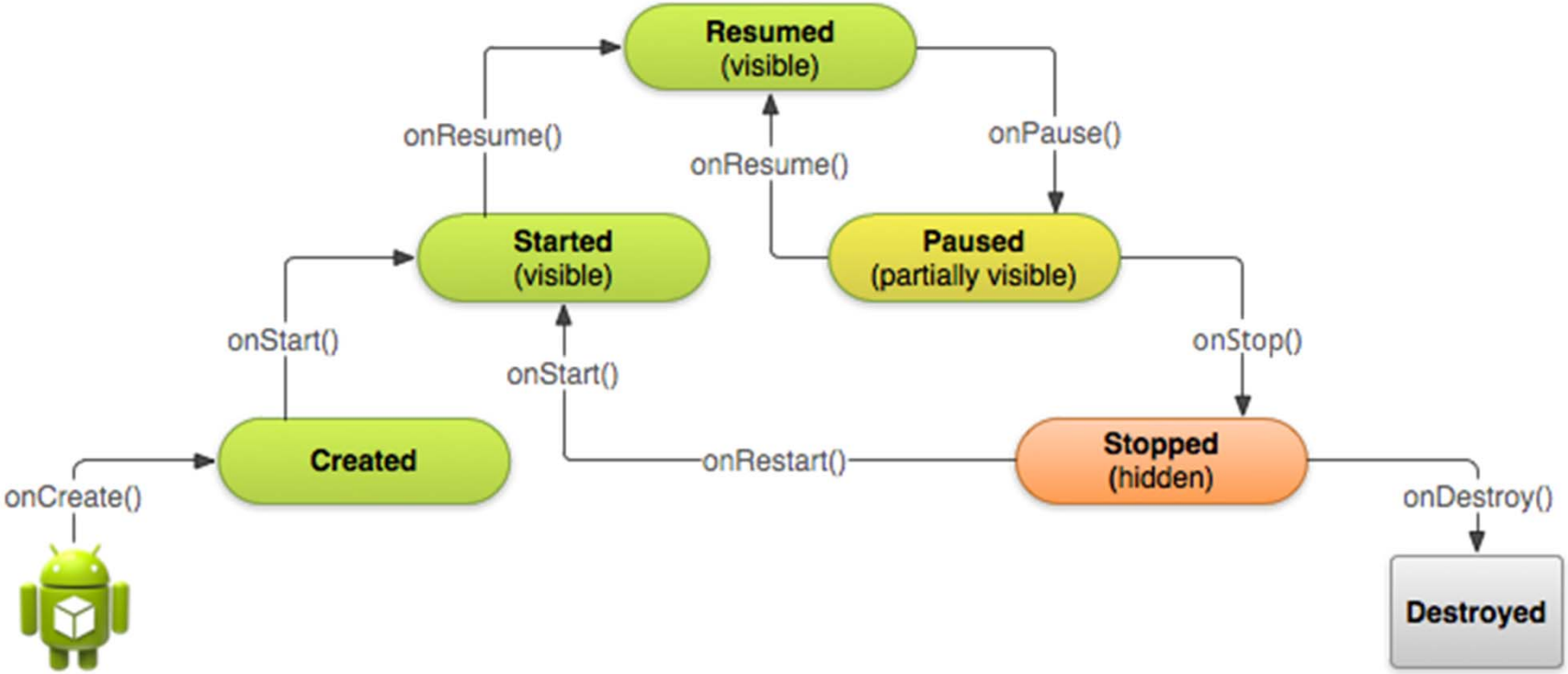
Activity : Cycle de vie / Evènements

Détails : Activity



Activity : Cycle de vie / Evènements

Détails : Activity



Activity : Cycle de vie / Evènements

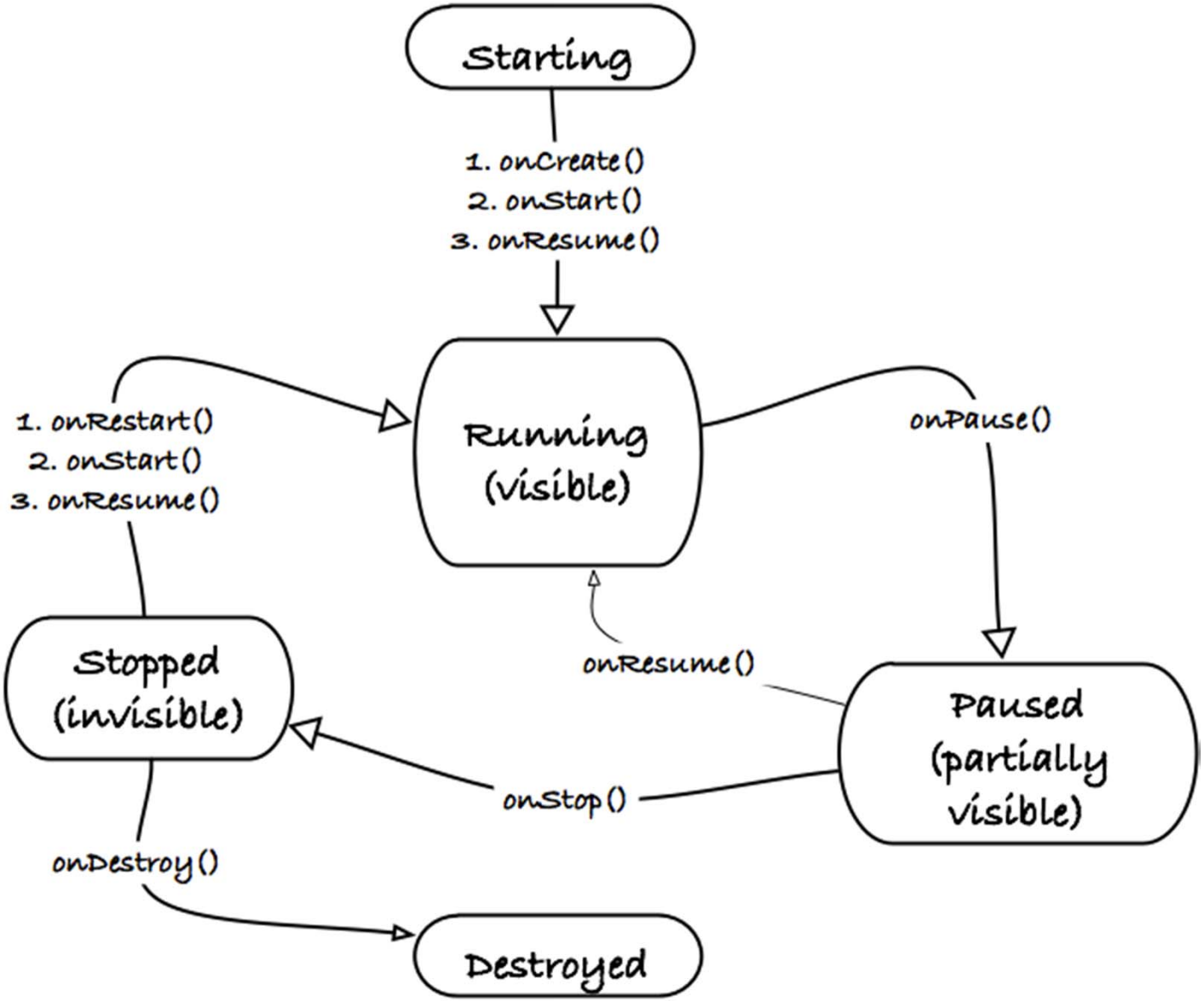
Détails : Activity

Evènement	Description	Visible	Killable after ?
onCreate	Instanciation + binding evt	Non	Non
onStart onRestart	Initialisation / Réinitialisation Chargement données persistantes Redémarrer les capteurs	Non	Non
onResume	Affichage du GUI	Oui	Non
onPause	Sauvegarde - Persistance Stop animation ... Stop service (GPS , ...)	Non	Oui
onStop	Libération des ressources (RAM)	Non	Oui
onDestroy	Destruction GUI Libération des autres ressources	Non	Oui

- Visible : L'activité est-elle visible (ie en premier plan)
- Killable after : Le système peut-il tuer le processus juste après la sortie de l'évènement ?

Activity : Cycle de vie / Evènements

Détails : Activity



Activity : Opération du cycle de vie

- Toute activité hérite de Activity

https://developer.android.com/reference/android/app/Activity.html

```
public class HelloActivity extends Activity implements .... {

    String TAG_LOG;
    public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                TAG_LOG=this.getClass().getCanonicalName();
                setContentView(R.layout. activity_main);
                 Log.i(TAG_LOG, "onCreate message");
    }
    protected void onDestroy() {
        Log.i(TAG_LOG, "onDestroy message");
        super.onDestroy();
    }
    protected void onStart() {
        Log.iTAG_LOG, « onRestart/onStart message");
        super.onRestart();
    }
```

Remplace
System.out.println

Implementer

Les GUI

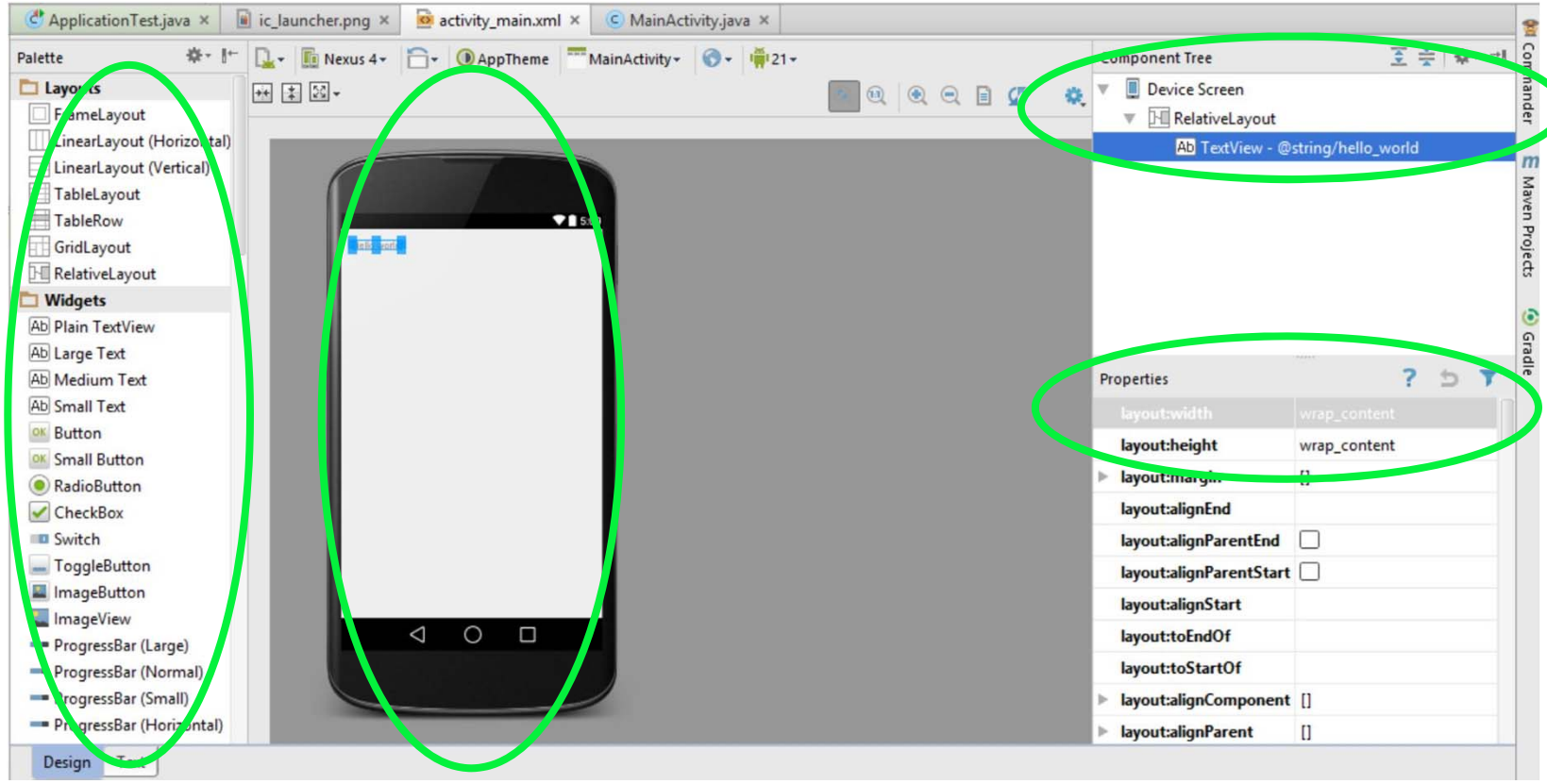
Activity : Création des vues

- 2 possibilités de conception
  - Pure Java : Programmatiquement (ie un SWING Java android)
  - XML « à la main »
  - XML en utilisant une GUI (VBA) dans un IDE
  - Possibilité de créer de multiples vues
  - Avantage: isolation code/vue en XML (comme en iOS)

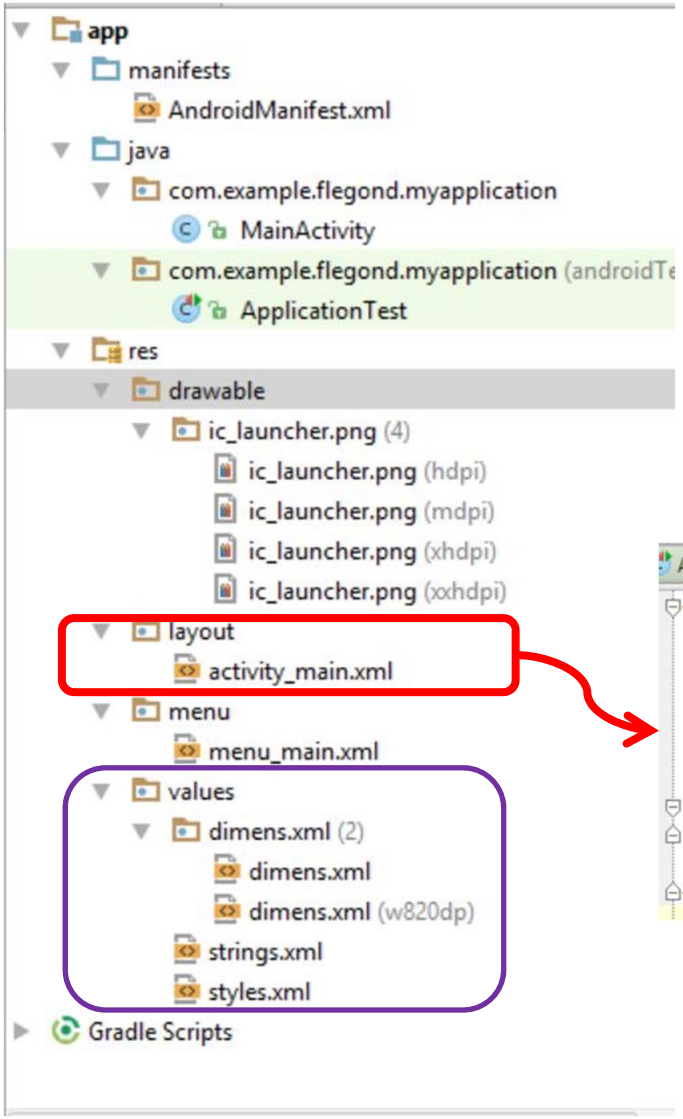
- Environnement / IDE :
  - Répertoire « res » (voir les slides manip)
    - Layout : définition des éléments des vues
    - Menu : définition des éléments de menus
    - Values: définition des constantes
    - Etc ...

Activity : Création des vues à la VBA

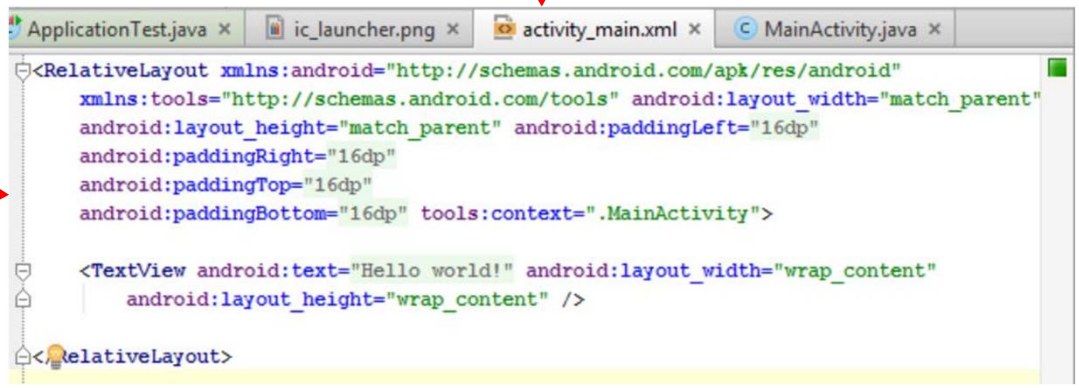
Détails : Activité



Activity : Création des vues XML



Fichier XML Brut !



Activity : Création des vues XML

- Une vue se caractérise par :
  - Type (Balise XML) et des propriétés
    - TextView, Button, ImageView, EditView ...
  - **Un identifiant de l'objet dans la vue (utilisé par R.id)**
    - Défini par la balise XML : android:id="@+id/mon_id"
    - Génère une classe virtuelle R.java qui contient les ids.
    - Si id vide, impossible de résoudre par R.java
  - Un layout
    - Accès aux identifiants des GUI XML via R.layout
    - **R.layout utilise le nom des fichiers XML**
- Les identifiants (id)
  - Le @+id est obligatoire dans les fichiers xml de layout
  - Le + force la création de la référence
  - @id partout ailleurs pour utiliser la référence
  - Pour l'instant le compilateur Android accepte @+id tout le temps mais c'est une tolérance
  - LIEN 1 to 1 entre une référence et un élément
  - Création d'une référence programmatique automatique : R.id.xxxxxxx

Activity : Création des vues XML

- Instanciation et Création
  - setContentView(int layout_id): dans la méthode onCreate() à mettre en priorité (instanciation)
- Lien entre XML et Objet(s) JAVA
  - findViewById(int id): récupère une référence d'objet java décrit par un XML (ELLE DOIT DÉJÀ ETRE INSTANCIEE)
  - Utilisation de la classe « android.view.LayoutInflater » pour instancier
  - Utilisation de R.java qui contient tous les éléments GUI
  - R.ressource_type.id
    - ressource_type: id (Retrouver un élément par son id), string, anim, layout, …
    - Id: l'identifiant

Activity : Liens GUI / Objets Java

- La classe R
  - Classe virtuelle (pas au sens langage c)
  - Générée à la compilation du projet
  - Souvent générée à la volée dans les IDE
  - R.java contient tous les éléments GUI qui ont une « id »
- Permet le lien entre XML et Objet JAVA
  - findViewById(int id)
    - ✔ récupère l'objet java décrit par un XML
    - ✔ **"No resource found that matches the given name"**
      si vous avez oublié le + dans le champ id de la description de l'élément
  - R.ressource_type.id
    - ✔ ressource_type: id, string, anim, layout, …
    - ✔ Retrouver un élément par son Id → R.id.xxxxxx
    - ✔ Id fonctionne sur n'importe quelle type de ressource nommée
    - ✔ Nécessite un cast … par ex

```
Button b = (Button) findViewById (R.id.my_only_button)
```

Activity : Liens GUI / Objets Java

- ATTENTION LES ELEMENTS PREDEFINIS DE L'OS sont dans « android.R.* ». Ne pas confondre R.* (applicatif) et android.R.* (OS)
- La méthode findViewById est couteuse
- Il est souvent conseillé de créer une « inner » static classe ViewHolder pour éviter les répétitions d'appels

```
static class ViewHolder {
    public TextView text;
    public TextView timestamp;
    public ImageView icon;
    public ProgressBar progress;
}
```

Activity : Un petit résumé (lien XML/GUI/R.java)

Détails : Activity

Génération automatique

Class R

Création / Instanciation

Accès

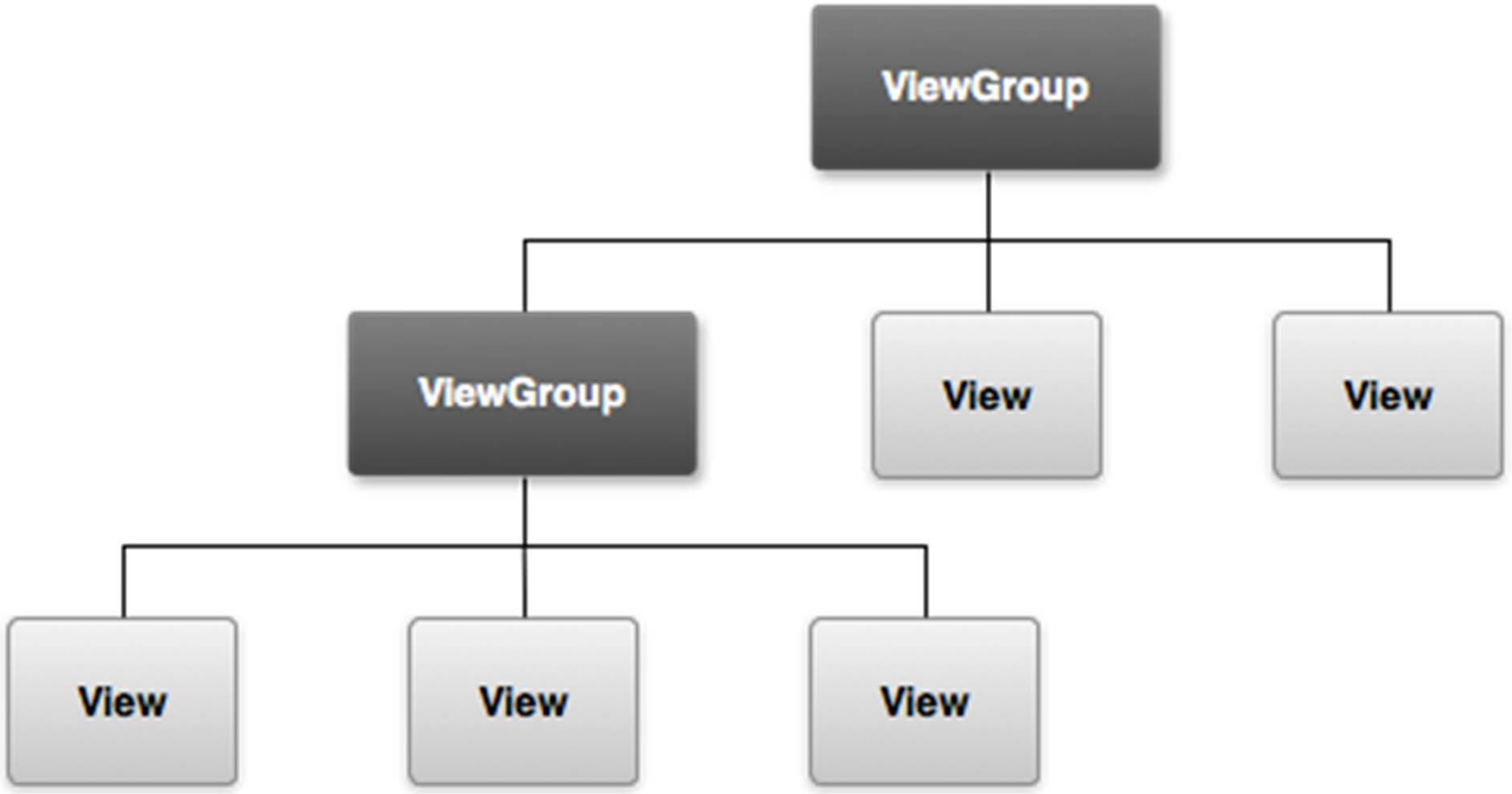
Source

Activity : View, ViewGroup

- Les vues sont composées d'éléments
- Il y a deux type d'éléments pour les vues
  - Les éléments complexes (ViewGroup) qui sont composés d'éléments
    - ✔ conteneur de vues (Layout, Grille, Liste)
  - Les éléments terminaux (View) – Input Control
    - ✔ Bouton, CalendarView, CheckBox, RadioButton
- Les vues forment une Hiérarchie / un arbre
- Chaque élément (ViewGroup et View) peut (ou non) implémenter des interfaces évènementielles
  - par ex. View.OnClickListener

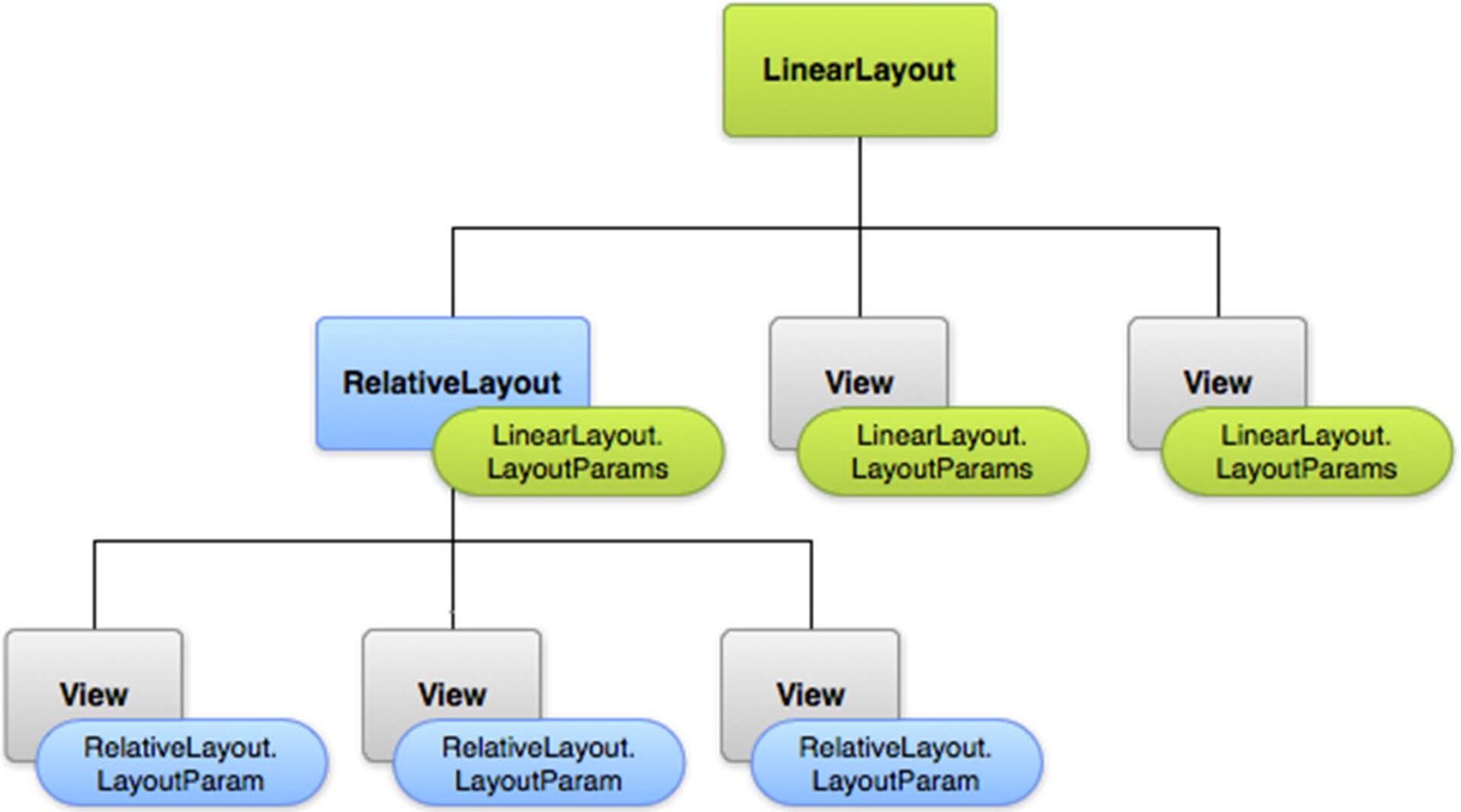
Activity : View, ViewGroup

Détails : Activité



Activity : GUI. Vue et éléments

Détails : Activité



Activity : GUI. Eléments ViewGroup

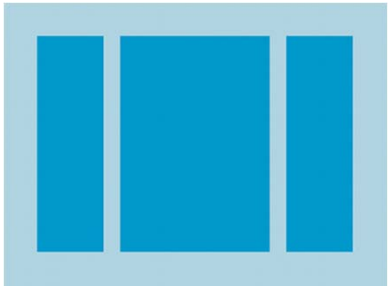
Détails : Activité

Layout basiques

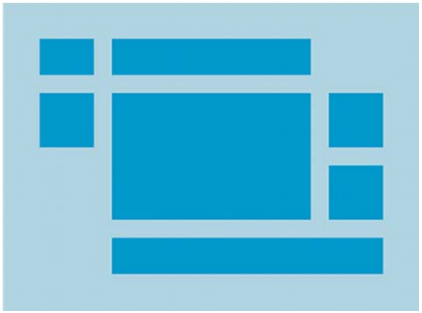
- FrameLayout
  - Le plus basique
  - Contient un seul élément
  - Utile pour faire un conteneur à Fragment
  - Réserver un espace
- RelativeLayout
  - Positionner les éléments les uns par rapport aux autres et par rapport aux bords du RelativeLayout
- LinearLayout (vertical/horizontal)
  - Elements les uns à la suite de autres
  - Un LinearLayout peut être soit vertical soit horizontal
- TableLayout
  - Un layout découpé en lignes horizontales
  - Chaque ligne peut être découpé ensuite en colonne

FrameLayout

RelativeLayout



LinearLayout



TableLayout

Activity : GUI. Eléments ViewGroup

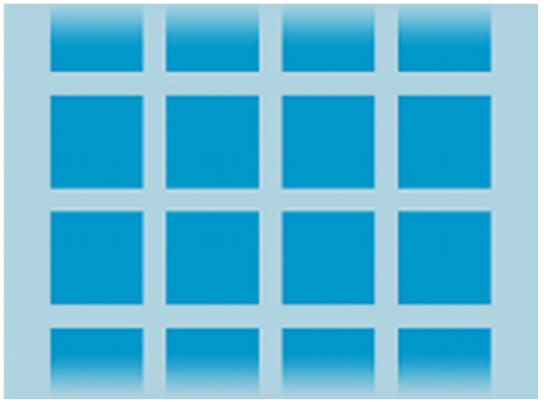
Layout avec conteneur

- Nécessite un adapteur (voir la section sur les ListView)
- ListView
  - Tous les éléments de la liste sont empilés les uns en dessous des autres.
  - Chaque élément de la liste est un layout
  - En général, on applique le même layout pour l'ensemble de la liste
- GridView
  - Identique à la ListView mais sous forme de grille

Layout avec contenu

- WebView

ListView



GridView

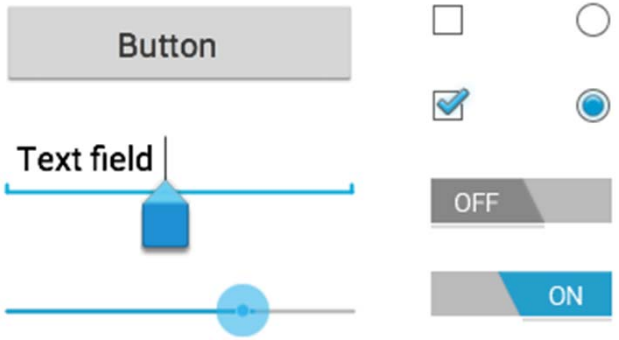
Activity : GUI. Eléments ViewGroup

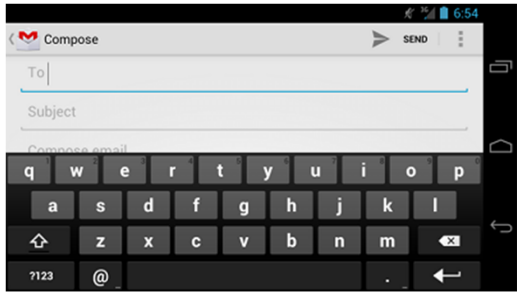
Les Layouts : quelques conseils

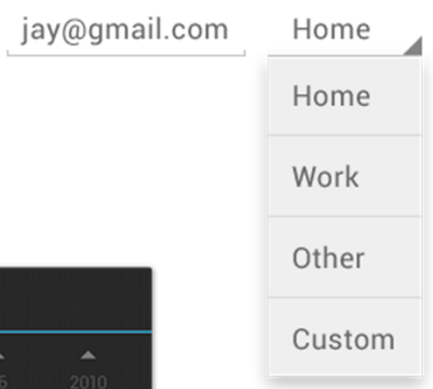
- Eviter l'empilement des éléments
  - Préférer les structures les plus à plat possibles
  - Empilement = Ralentissement
- Taille des éléments:
  - warp_content : la taille de l'élément sera définie par son contenu
  - match_parent : la taille de l'élément sera fixée par les bordures du conteneur
  - fill_parent : DEPRECATED. Identique à match_parent
  - Une taille fixe : la taille donnée

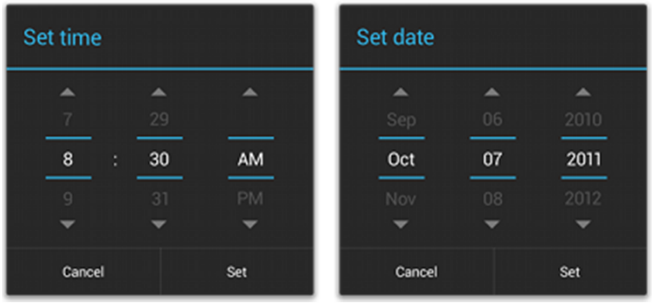
Activity : GUI. Eléments View

- View / Input Controls
  - AutoCompleteText
  - Spinner
  - CheckBox
  - RadioButton
  - EditText
  - ToggleButton
  - ImageView
  - …









Menu

Menu Contextuel

ActionBar

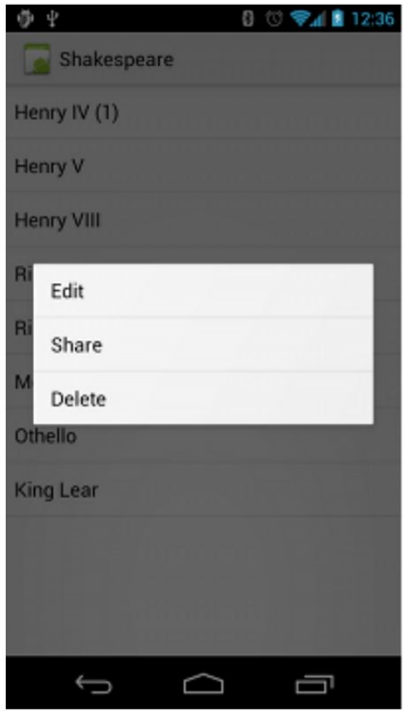
Activity : Menu d'options / Menu d'activité

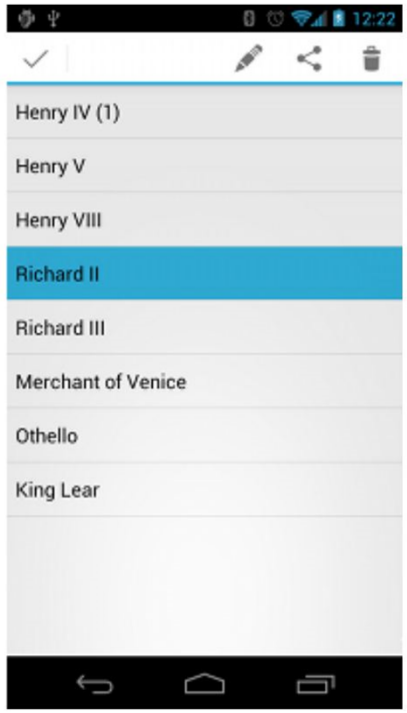
- OptionsMenu :
  - Icône + « MenuItem »
  - Pas de sous menu
  - Activable par pression du «Menu button»
  - 1 Menu / Activity
- Jusque Android 2.3.x :
  - Pas plus de 6 items, les autres sont passé dans «more»
  - Apparition dans le bas de la fenêtre
- A partir d'Android 3.x :
  - Utilisation d'un espace dédié «Action Bar»
  - Possibilité de définir son propre layout
- Une application peut aussi avoir une ActionBar
  - 1 ActionBar / Activity

Activity : Menu contextuel

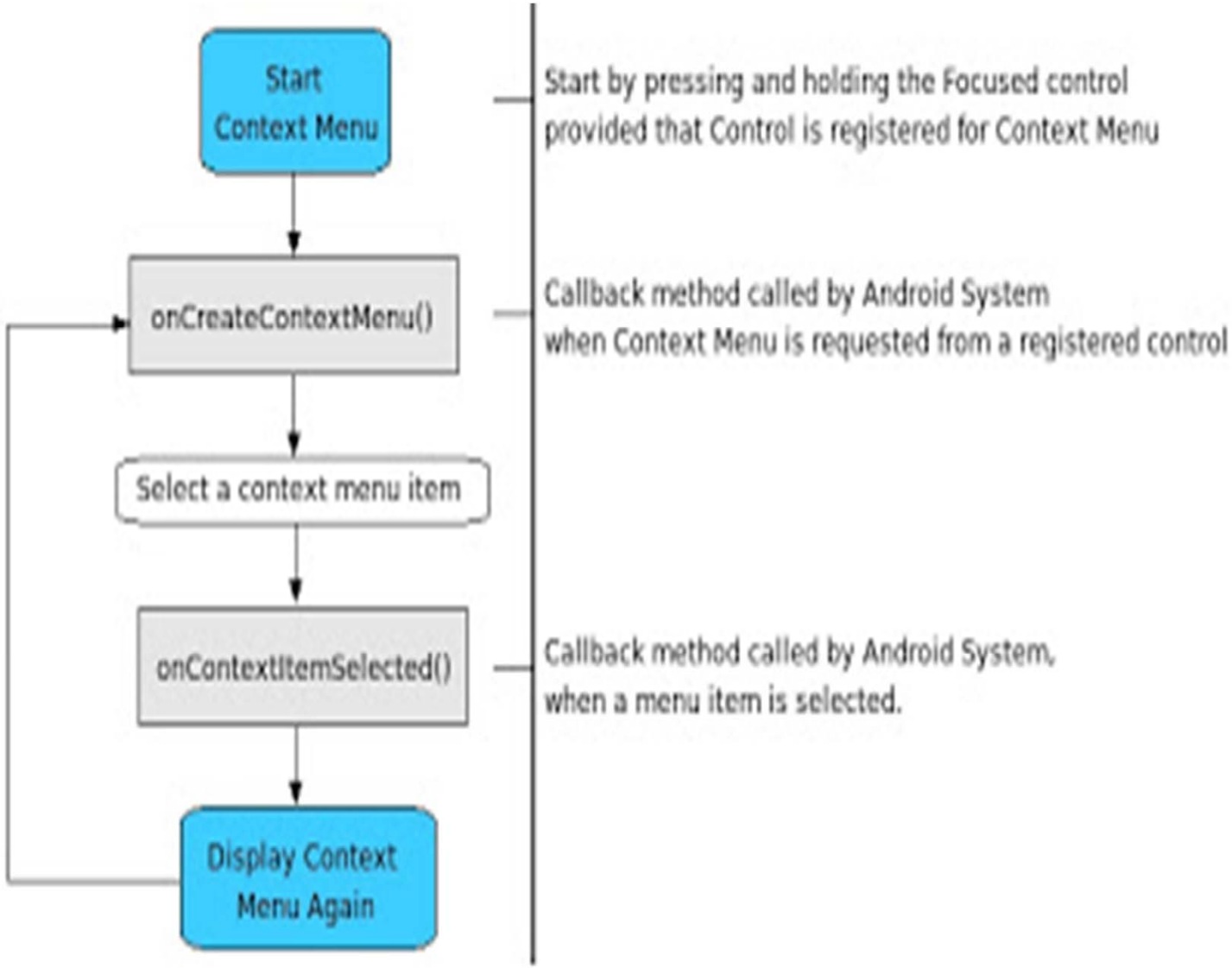
Context Menu :

- Menu contextuel
  - Possibilité de sous menus
  - Activable par pression longue sur un élément
- La vue doit supporter le menu contextuel
  - registerForContextMenu()
- 2 types :
  - floating context menu
  - contextual action mode
- Plusieurs Context Menu / Activity

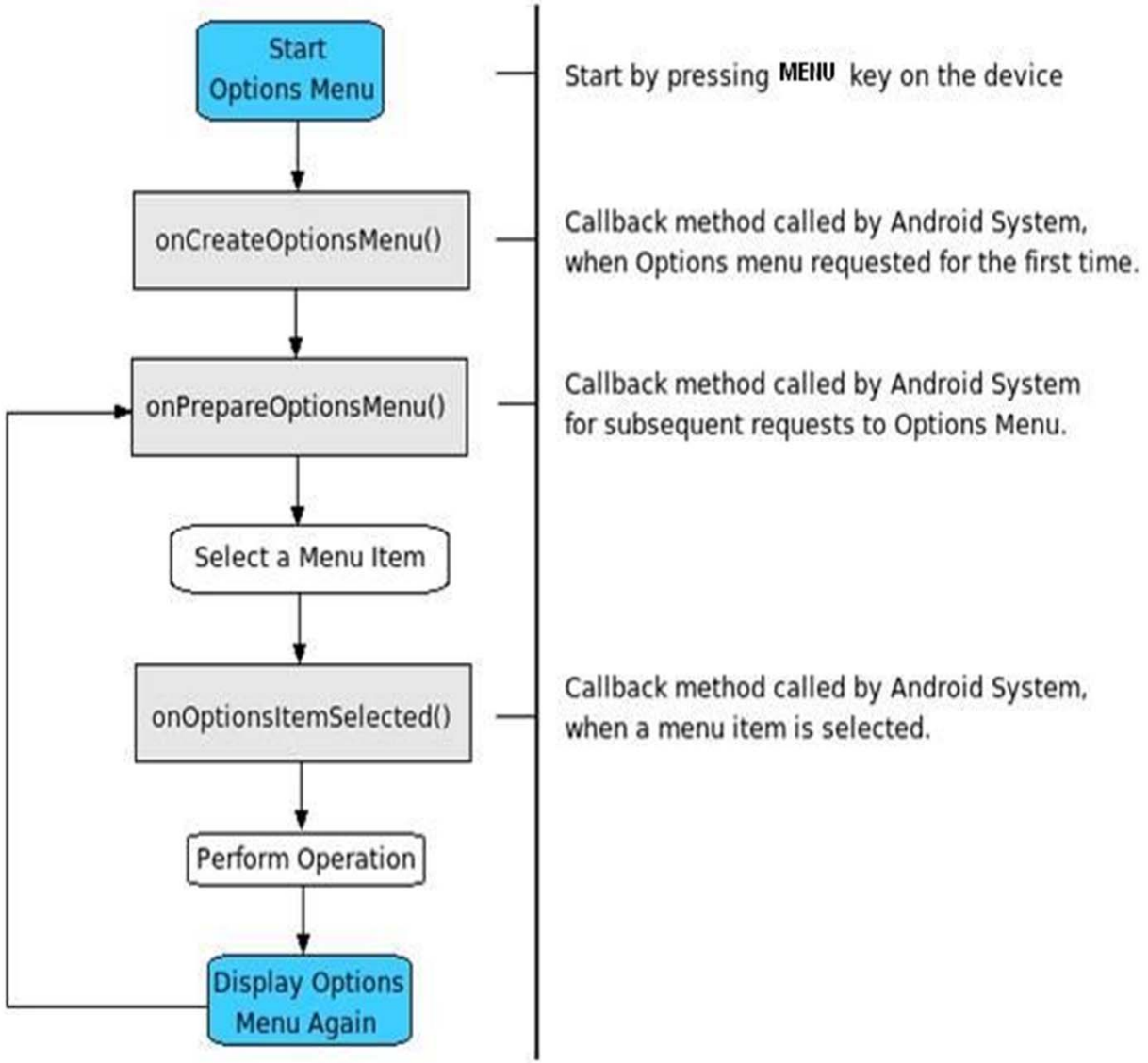




Activity : Cycle de vue des menus

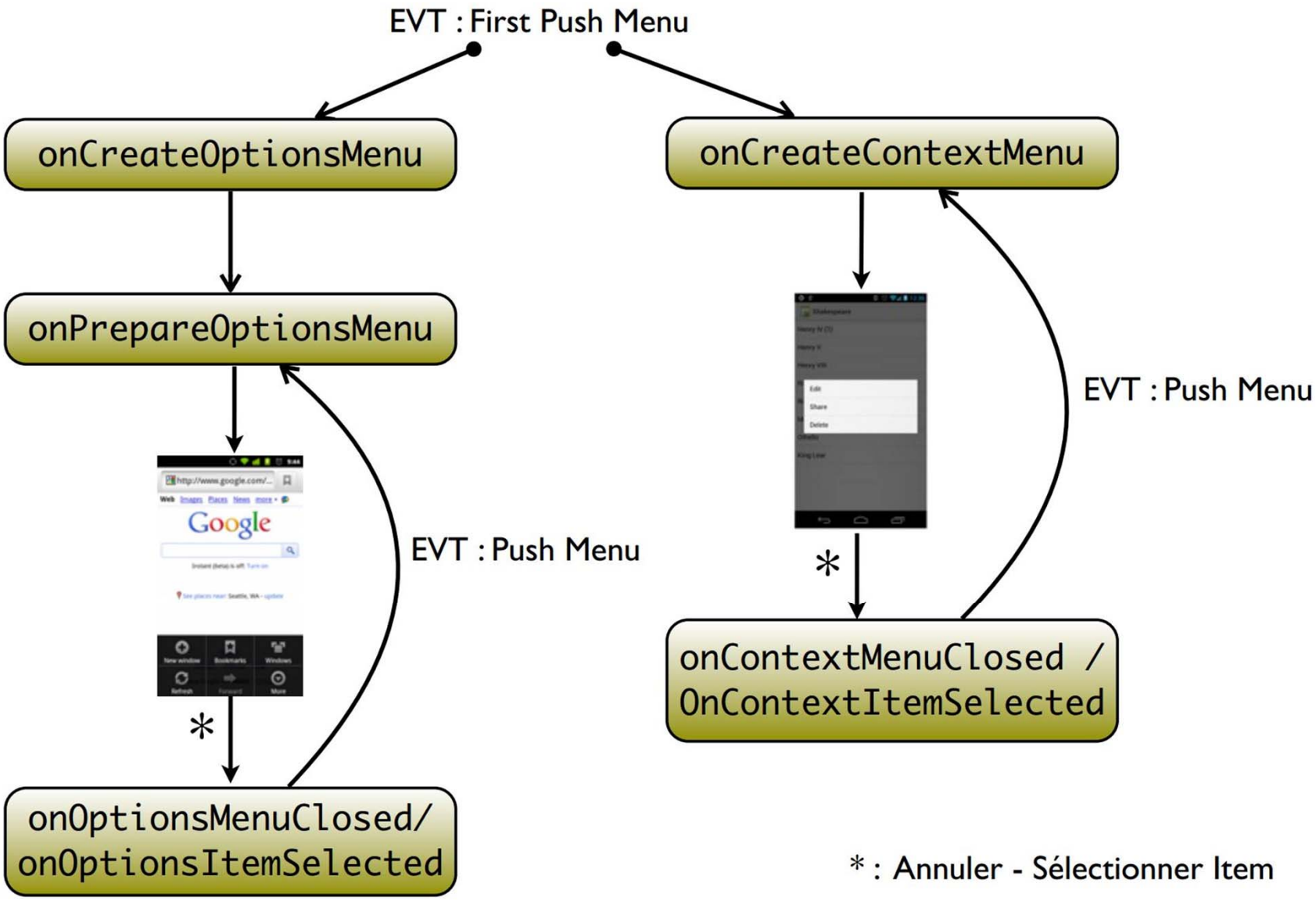


Activity : Cycle de vue des menus



Activity : Activity : Cycle de vue des menus

Détails : Activity



Notifications:

Toast

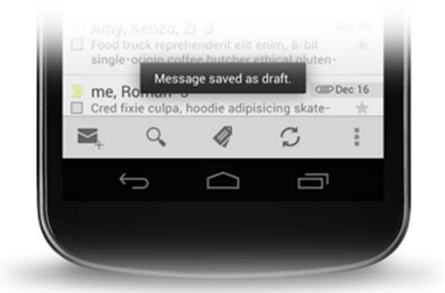
Status bar

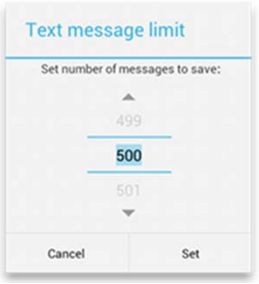
Dialog

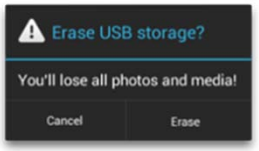
Activity : Notifications

Différentes types de notifications :

- Toast Notification
  - Message immédiat (bulle)
  - Simple d'utilisation
- Status Bar Notification
  - Message + icône (barre des notifications)
  - Utiliser dans les tâches de fond (Service)
- Dialog Notification
  - Boîte de dialogue ...
  - AlertDialog, ProgressBar ...







Activity : Notifications Toast

- Toast Notification

```
//display in short period of time
Toast.makeText(getApplicationContext(), "msg msg",Toast.LENGTH_SHORT).show();

//display in long period of time
Toast.makeText(getApplicationContext(), "msg msg", Toast.LENGTH_LONG).show();
```

- Les notifications Toast peuvent être « Skinées » (personnalisées)

```
// get your custom_toast.xml layout
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
        (ViewGroup) findViewById(R.id.custom_toast_layout_id));
// Fill the view with …
// set a dummy image
ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.ic_launcher);
// set a message
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Button is clicked!");
// make a Toast... !!!
Toast toast = new Toast(getApplicationContext());   toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG); toast.setView(layout); toast.show();
```

Activity : Notifications Status Bar

Détails : Activity

- Toutes les options des notifications de status ne sont pas disponibles en fonction des versions Android
  - Vérifier en fonction de vos API !!!
  - Une notification « statusbar » est associé à un context
  - NotificationCompat (pour une version plus compatible) et Notification (pour la dernière version)
- Création de la notification

```
NotificationCompat.Builder builder =
                new NotificationCompat.Builder(getApplicationContext());
Notification notification = builder
        .setSmallIcon(R.drawable.me)
        .setContentTitle(getString(R.string.notif_title))
        .setContentText(txt)
        .build();
```

- Puis display Notification ()
  - A vous de gérer les id de chaque message
  - Par ex id (4) permet de mettre à jour, plus tard, la notification

```
NotificationManager notificationManager =
      (NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
      notificationManager.notify (4 , notification);
```

Activity : Notifications Status Bar

- Associer le démarrage d'une activité à la notification
  - revenez dessus lorsque nous aurons vu les intents

```
Intent intent = new Intent(getApplicationContext(),
        MainActivity.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
        Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(
        getApplicationContext(), 0, intent,
        PendingIntent.FLAG_UPDATE_CURRENT);
```

- Puis l'ajouter avec setContentIntent(notificationIntent)
- Quelques autres options possibles (RTFM, la vibration requière autorisation) sur une instance de Notification :

```
.setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.me))
.setContentInfo(getString(R.string.notif_contentInfo))
.setWhen(Calendar.getInstance().getTimeInMillis() + 1000*60+60)
.setVibrate(vibrationPattern)
...
```

Avec long[] vibrationPattern = { 0, 200, 800, ...};

Activity : Notifications Status Bar

- On peut changer le style de la notification
  - Utilisation de .setStyle avec une instance de InboxStyle de NotificationCompat/Notification

- **Rappel : ATTENTION EVITER LE MELANGE**
  - **android.app.* (version récente)**
    - ✔ android.app.Notification
  - **android.support.v4.app.* (version de compatibilité – support library)**
    - ✔ android.support.v4.app.NotificationCompat

Activity : Notifications Dialog

- Une notification « Dialog » est associé à une activité
- Code similaire aux notification de statusbar

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
builder.setMessage(« my alert message »);
builder.setTitle (« my alert windows title »);
AlertDialog ad = builder.create();
ad.show();
```

- Il existe différents types de dialog
  - AlertDialog, CharacterPickerDialog, MediaRouteChooserDialog, MediaRouteControllerDialog, Presentation
  - DatePickerDialog, ProgressDialog, TimePickerDialog

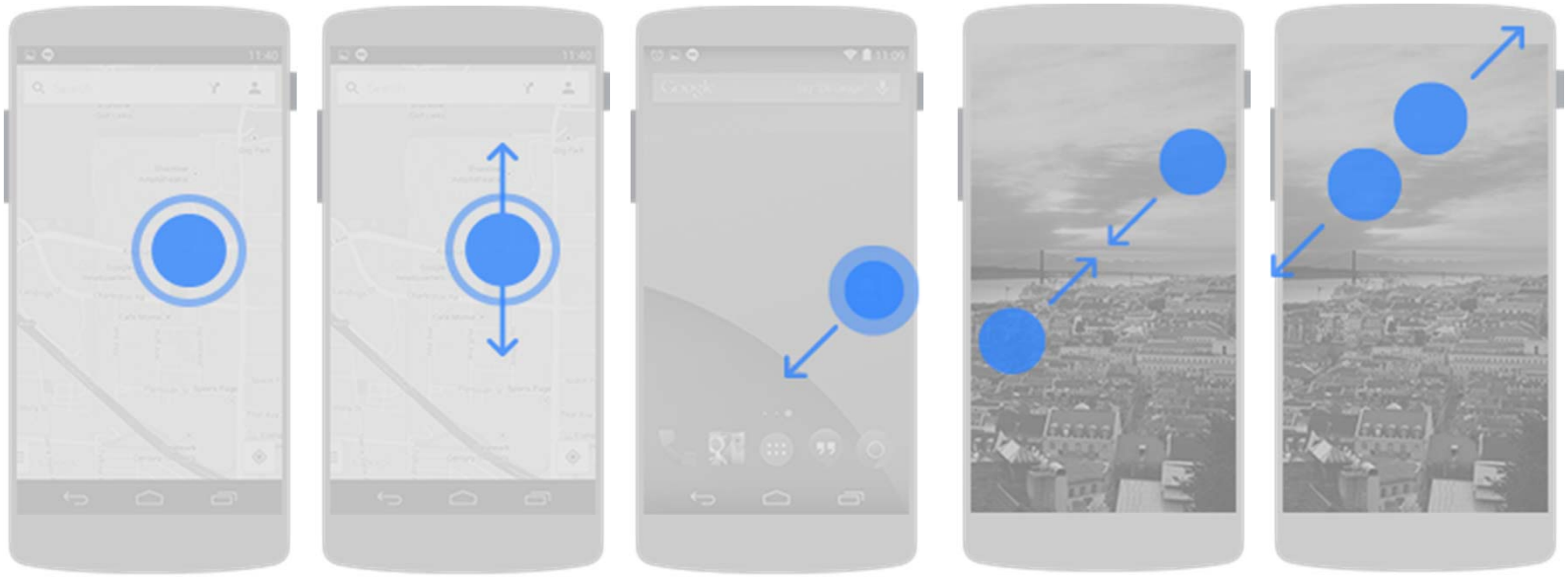
Evènements Utilisateurs sur une activité

Activités : Evènements de touché

- le multi-touch
  - implémenté depuis Android 2.0 / 2.1 (Eclair)
    - Ne pouvait gérer que 3 « Touch » simultanément
  - Depuis Android 2.2 (Froyo)
    - plus de limite
- Action d'un « Touch » (down/move/up)
- Gestions avancées ?
  1. Tap (Click, LongClick)
  2. Drag and Drop (>3.x)
  3. Zoom (>2.x)
- Développer son API …

Activités : Evènements de touché

Détails : Activité



Activités : différents types d'évènements

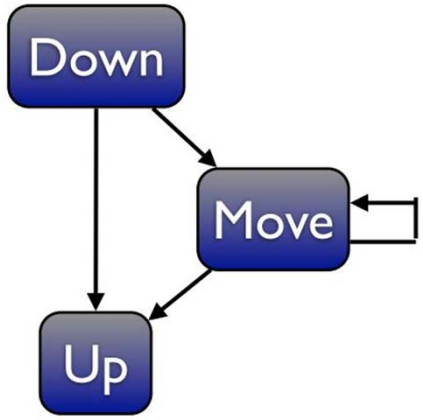
- Actions prédéfinies. Utilisation intensive de listeners de la classe View (interfaces d'évèments associées aux View)
  - View.On*: Interface pour de nombreux évènements
    - View.OnDragListener: gestion du drag and drop
    - View.OnTouchListener: tap
  - GestureDetector.OnGestureListener: événements prédéfinis
  - ScaleGestureDetector.OnScaleGestureListener: gestion du zoom
- Pas de Shake Motion (mais de nombreux exemples d'implémentations sur le net)
- Multitouch pour traitements particuliers
  - Notion de action_pointer_down et action_pointer_up
- Le multitouch est plus complexe à gérer que sous iOS

Activités : Evènements de touché

- Utilisation via l'interface View.OnTouchListener

  public boolean onTouch(View v, MotionEvent event);

  – Cycle de vie d'un « Touch »



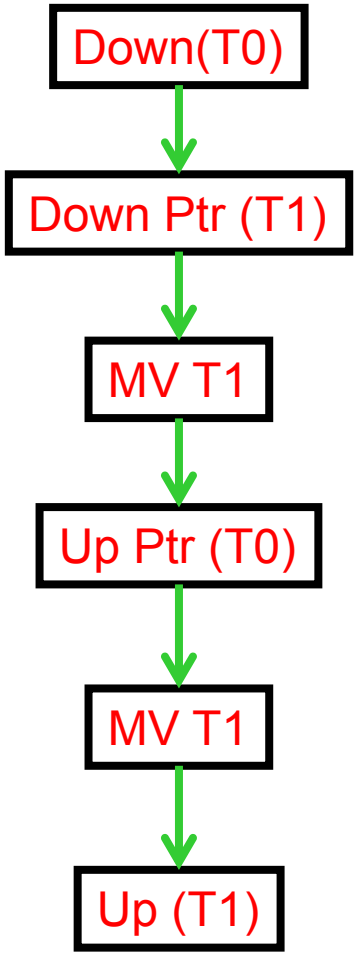
- Reçu par la méthode : MotionEvent
  - Vecteur de « Touch »
    - ✔ Nombre de « Touch », Actions
    - ✔ Position (x,y), Temps
    - ✔ Historiques ...
  - Les événements sont « recyclables »
    - ✔ Ne pas sauvegarder les instances directement

Activités : Evènements de touché

Détails : Activité

Down / Up != Down Pointer / Up Pointer

- Up/Down sont le premier et dernier évènement de contact
- Up/Down pointer sont tous les évènements de contact entre
- DOWN / UP
  - Dans le cas du MONO-TOUCH
  - SI event.getPointerCount() <= 1
- DOWN / UP POINTER
  - tous les autres « touch »
  - Dans le cas du MULTI-TOUCH
  - SI event.getPointerCount() >= 2
- Exemple
  - Premier touch Down (T0)
  - Deuxième touch Down (T1) → Pointer
  - Déplacement de (T1)
  - Touch Up (T0) → pointer
  - Déplacement de (T1)
  - Touch Up (T1)



Activités : évènements préprogrammés

- Événements de « View »
  - onClick: au clic sur un item (Button ...)
  - onLongClick: au maintient d'un clic durant un certain temps
  - onCreateContextMenu: création du contexte menu après un LongClick (ListView)
  - onFocusChange: lorsque la sélection change (avec le trac-ball)
  - onKey: au clic sur un bouton du clavier
  - onTouch: à la détection d'une « gesture » UP/DOWN/MOVE
- Recevoir les événements. 3 méthodes :
  - Pattern Observer : implémenter l'interface
  - Récupérer la vue
  - S'abonner à l'événement

Activités : exemple de capture de click

- Ajout du listener par le fichier XML (onClick de certaines View)
- Ajout des listeners (onClick et tous les autres) par du code

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    /* Bind the XML view */
    /* INSTANCIATE all sub elements / views */
    setContentView(R.layout. main);

    /* Link onClick operation */
    Button button = findViewById(R.id. Button01);
    button.setOnClickListener(this);
    /* Create a SECOND listener : a long click listener ... */
    button.setOnLongClickListener( ……. );
}


public void onClick(View v){
    /* Do something */
}
```

Activité : la Mort

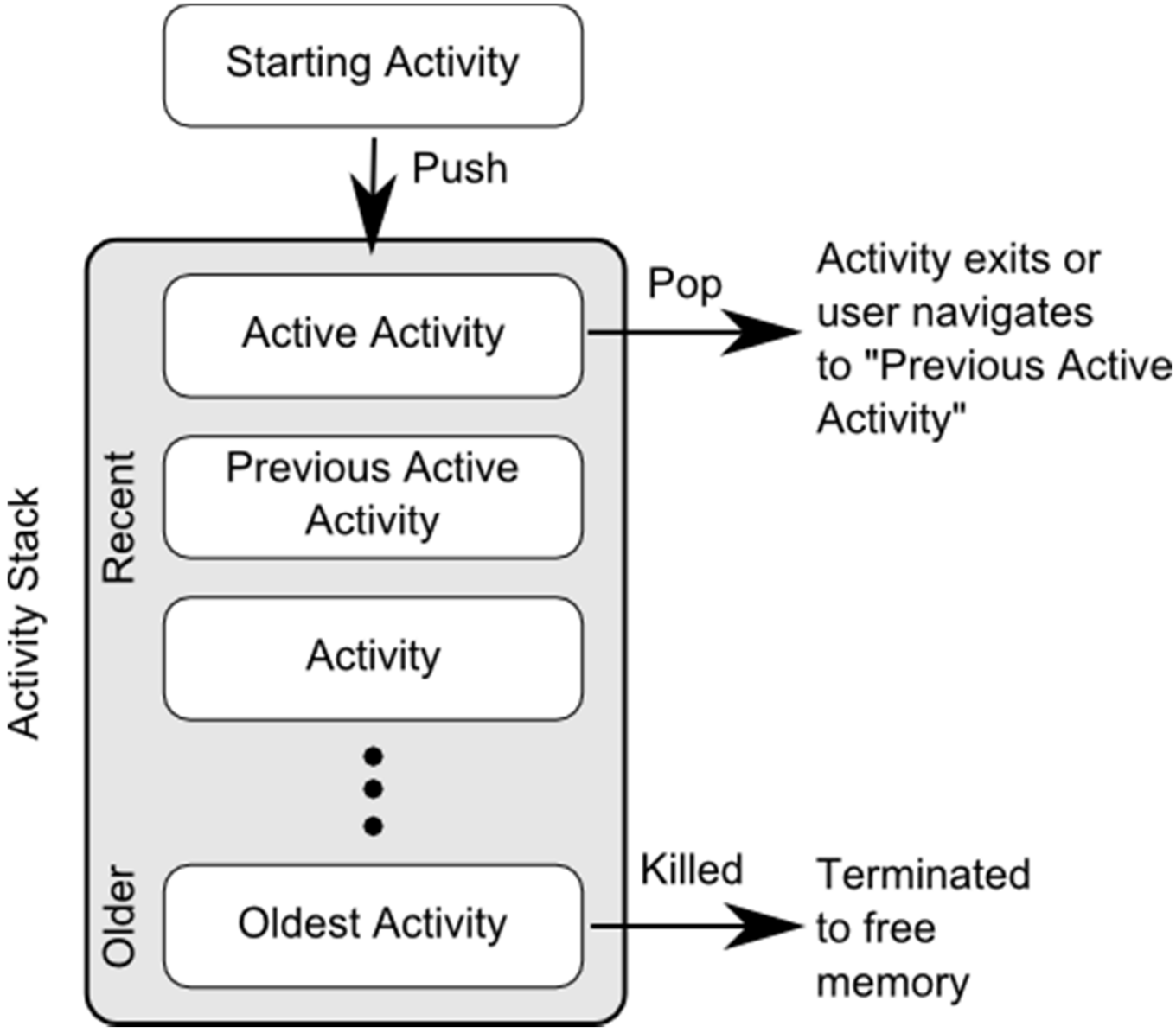
Le système gère une liste d'Activités

- La tête de liste : « en cours d'affichage » (onResume)
- Les autres : « passives » (onStop)
- A la Création
  - Activity pas dans la liste
    - Création et ajout en tête de liste => onCreate / onStart / onResume
  - Activity dans la liste
    - Mise en tête de liste => onRestart / onStart / onResume
- Mise en pause
  - Lancement d'une autre Activité...
    - onPause / onStop
  - Terminaison
    - L'utilisateur appuie sur « Annuler »
    - Cas particuliers (cf « Activity : Killed »)

Détails : Activity

Activité : la Mort

Détails : Activité



Activité : la Mort

- Terminaison d'une Activité ?
  - Cas nominal : create-start-resume-pause-stop-restart
  - Changement d'orientation (voir la variable Bundle)
  - Tué par le système
    - Mémoire - Changement de l'orientation (si pas géré) – Bug
    - Peut arriver n'importe quand !
  - Statut tué (« killed ») ?
    - Application détruite (onDestroy)
    - Mais qui sera notée comme re-exécutable
      (Toujours dans la liste des activités)
  - Si pas de Persistance … perte des données
  - Solution ?
    - Persistance intermédiaire ? durable ?
    - Mécanisme offert par Android

Activité : la Mort

- En résumé, il faut des moyens pour:
  - Echanger des données entre les éléments
    Voir le partage des données
  - Eviter la perte de données ?
    Voir Utilisation de la persistance

Que nous découvrirons un peu plus tard ...

Détails : Activité

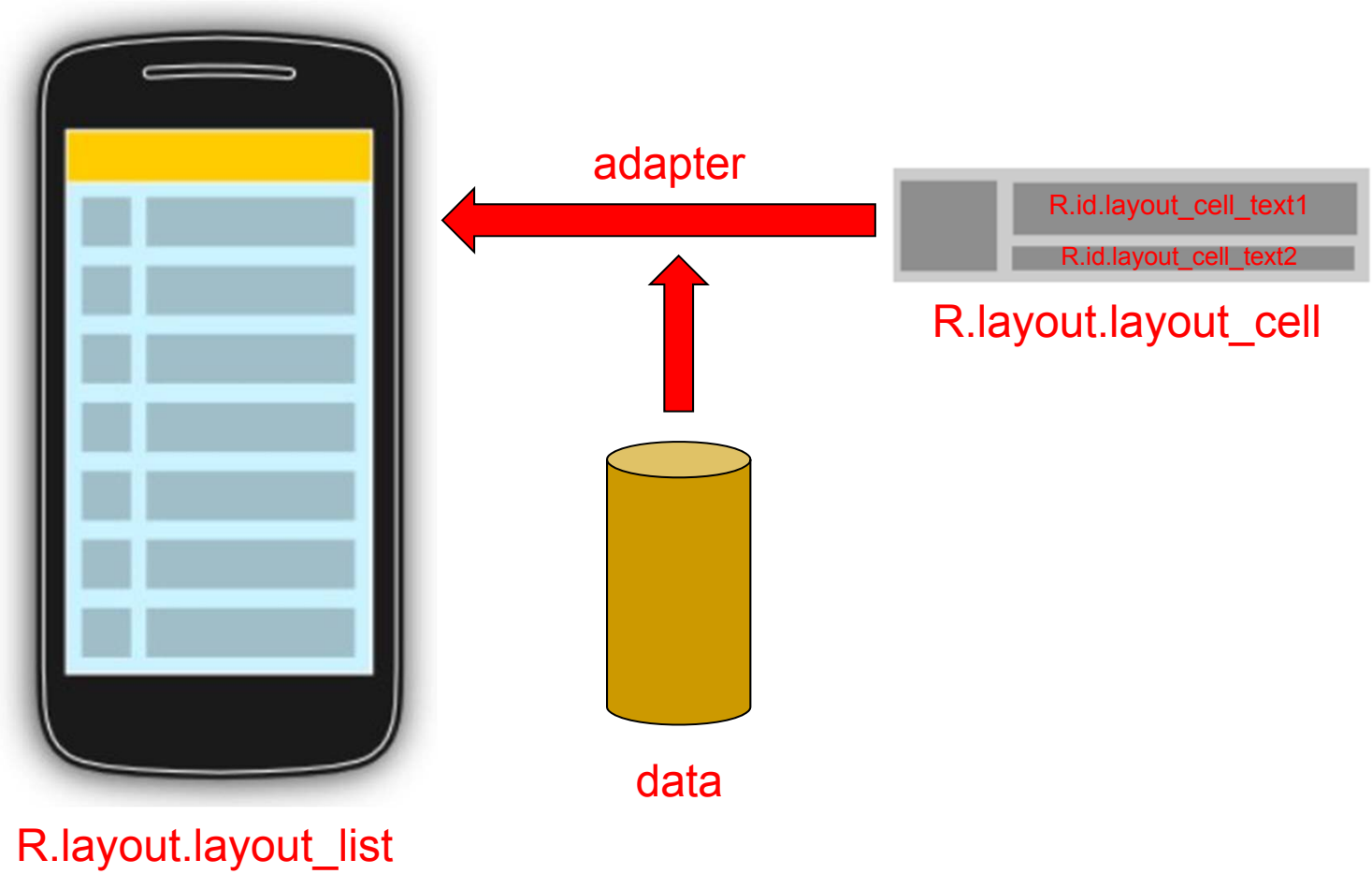
Eléments GUI :

Les listes

Liste : différents types de listes

- Plusieurs façons d'utiliser des listes : ListActivity, ListFragment, ListView et ExpendableListView
- ListActivity est une activité constituée uniquement d'une ListView, pas besoin de faire un fichier xml
- Une liste se gère par une classe adaptateur (Adapter)
- Une classe Adapter se connecte à une source de données et génère chacune des cellules de la liste
  - On doit définir un layout pour chaque TYPE de cellule de la ListView
  - Des types de cellules prédéfinies existent
    - android.R.simple_list_item_1, android.R.simple_list_item_checked
  - Il y a donc au minimum deux layouts
    - un pour la liste, un pour la description d'une cellule (qui peut être prédéfini)

Liste : différents types de listes



Liste : différents types de listes

- On fixe en général l'adaptateur dans le onCreate()
- Pour fixer un listener sur chaque item (le même pour tous) : setOnItemClickListener
- Utilisation de notifyDataSetChanged() et notifyDataSetInvalidated() pour mettre à jour la liste
- Il existe donc différents adaptateurs
  - Chacun prenant un type de source de donnée
  - Rempli le tableau en mettant les données dans les cellules
- L'adaptateur permet de gérer la mémoire (réutilisation de cellules déjà instanciées)

Liste : différents types de listes

- Adaptateurs:
  - SimpleAdapter: permet de placer plusieurs éléments dans plusieurs éléments d'une vue en utilisant un hashtable de description des associations
  - ArrayAdapter: permet de placer chacun des élément d'un tableau dans un élément d'une vue
  - SimpleCursorAdapter et CursorAdapter: permet de faire un accès aux base de donnée
  - SpinnerAdapter: fait le lien entre les listes déroulantes et leurs données
- Pour CHACUN des adaptateurs, il est possible de surdéfinir (@Override) la méthode getView pour sélectionner le layout de la cellule en fonction d'un algorithme particulier
- Il existe d'autres méthodes Overridable …

Liste : ArrayAdapter

- Plus simple que le SimpleAdapter
- Permet de faire de listes basiques contenant un seul élément (ex: string, int, images)
- Un constructeur de ArrayAdapter (il existe plusieurs constructeurs)
  - public ArrayAdapter (Context context, int resource, int textViewResourceId , T[] objects)
  - Context : le contexte de l'application (ex: this)
  - int ressource: le layout d'un élément de la liste (peut être modif. par getView)

    R.layout.layout_cell
  - int textViewResourceId: La vue de l'élément à remplir (TextView)
  - T[] objects: listes des éléments à ajouter (type unique, String)

Liste la plus basique : ArrayAdapter

- La ArrayList peut être vide ou non.
- Pour ajouter dynamiquement un élément, il faut invoquer notifyDataSetChanger sur l'adapter.
- Exemple de code :

```
ArrayList<String> als = new ArrayList<String>();
ArrayAdapter<String> sa =
    new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,als);
ListView lv = (ListView) findViewById(R.id.listView);
lv.setAdapter(sa);
als.add("test1");
als.add("test2");
sa.notifyDataSetChanged();
```

Liste assez basique : SimpleAdapter

- Le constructeur de SimpleAdapter
  - Context : le contexte de l'application (ex: this)
  - List<? extends Map<String, ?>>
    - Les données sous forme d'une liste d'éléments
    - Chaque élément est une (Hash)Map associant le nom de la donnée et la donnée
    - Un exemple d'une liste de 3 éléments

      ArrayList <

      HashMap [(« data1 » , valeur1) (« data2 » , valeur2)]

      HashMap [(« data1 » , valeur3) (« data2 » , valeur4)]

      HashMap [(« data1 » , valeur5) (« data2 » , valeur6)]

      >
  - int ressource: le layout d'un élément de la liste (peut être modif. par getView)

    R.layout.layout_cell
  - String[] fromColumnKey : le nom des champs de la liste de données

    [« data1 » , « data2 »]
  - int[] toTextView : la liste de champs (TextView) du layout de la cellule / élément

    [R.id.layout_cell_text1 , R.id. layout_cell_text2]
  - String[] et int[] doivent avoir la même taille !

Liste : Gestion de liste complexe

- Les classes adapter peuvent être étendues
  - Pour des listes complexes (ie des cellules / éléments de types différents)
  - De préférence étendre un ArrayAdapter (voir SimpleAdapter)
  - Utiliser un ArrayList d'objets complexes (classe contenant les données)
  - Surdéfinir la méthode getView de l'adapter
- Surcharge de getView()
  - Si la convertView est non nul, c'est qu'android fourni un objet recyclé à utiliser (peu utile pour des listes complexes)
  - On peut remplir les champs de chaque élément de la ListView dans la méthode getView
- Rappel: pensez à notifyDataSetChanged() et notifyDataSetInvalidated() sur l'adapter
- Rappel : On associe ensuite l'adaptateur au layout liste
  - Par « setListAdapter(Adapter a) »

Liste : Gestion de liste complexe


```
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater inflater = (LayoutInflater) parent.getContext().
    getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    View rowView;
    if (null == convertView) {
        rowView = inflater.inflate(R.layout.activity_host_list_element, parent, false);
    } else {
        // PAS TOUJOURS ADAPTE AUX BESOINS
        // IL FAUDRA PARFOIS TOUJOURS JETER LE convertView FOURNI
        rowView = convertView;
    }
    if(position%2==0)
        rowView.setBackgroundColor(Color.LTGRAY);
        else rowView.setBackgroundColor(Color.WHITE);
    TextView tv_fqdn = (TextView) rowView.findViewById(R.id.hle_textView_fqdn);
    TextView tv_ip = (TextView) rowView.findViewById(R.id.hle_textView_ip);
    TextView tv_yesno = (TextView) rowView.findViewById(R.id.hle_textView_yesno);
    Host h = getItem(position);
    tv_fqdn.setText(h.fqdn);
    tv_ip.setText(h.IP);
    if (h.isSVC)
        tv_yesno.setText("Yes");
        else tv_yesno.setText("No");
    return rowView;
}
```

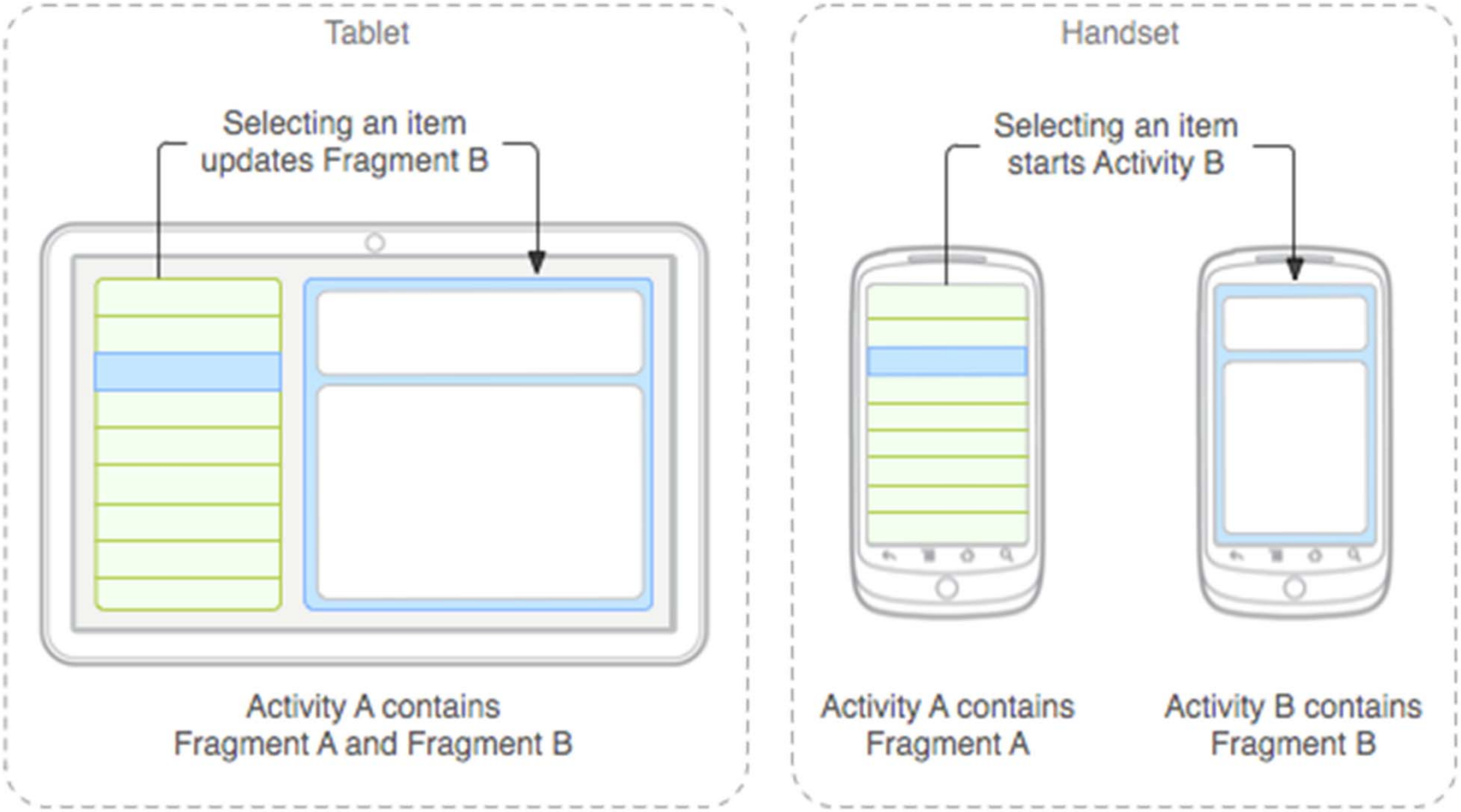
Eléments GUI:

Les fragments

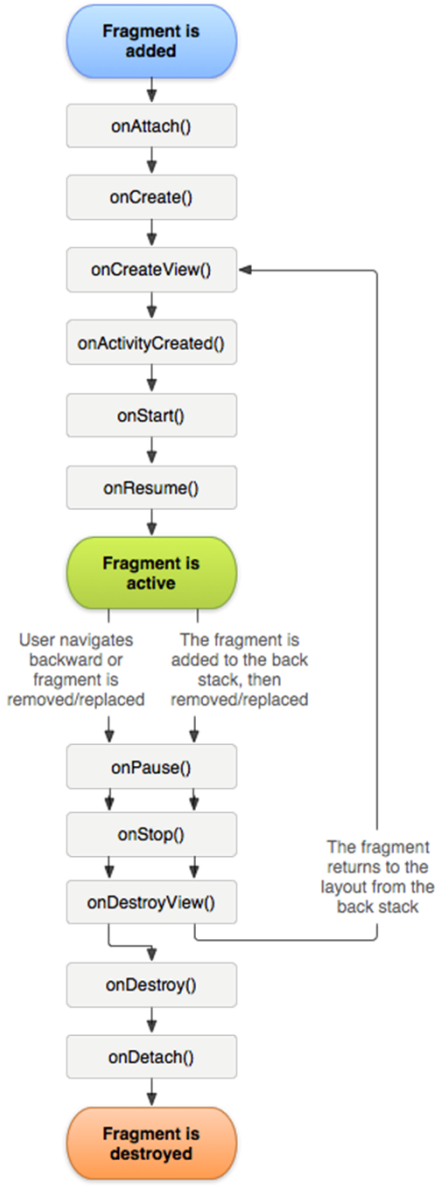
Fragment : Définition

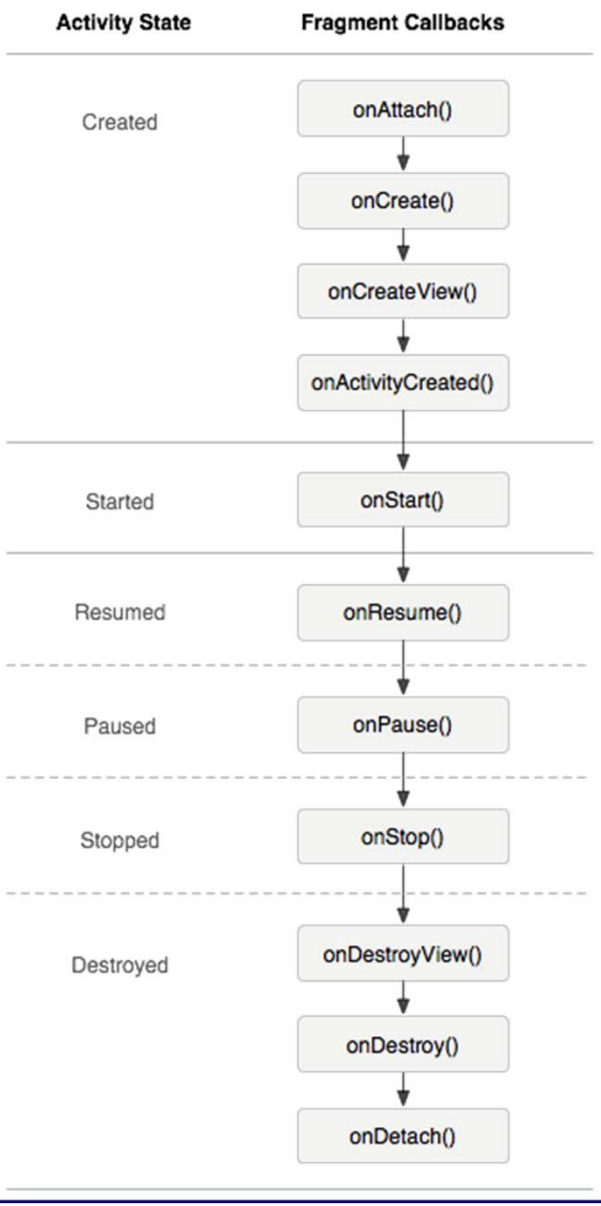
- Le probable successeur des Activités
  - Dans Android Studio (KITKAT), un fragment est associé à chaque activité pour la vue principale
  - Permet de faire des activités modulables et dynamiques
- Partie modulaire d'une Activité
  - Composant réutilisable
  - Construction de panels de vues
- Possède son propre cycle de vie mais suit le cycle de vie de l'activité
- Ajout ou retrait dynamique
  - N'utilise pas le bus de communication
  - Permet d'éviter les commutations entre activités

Fragment : Exemple

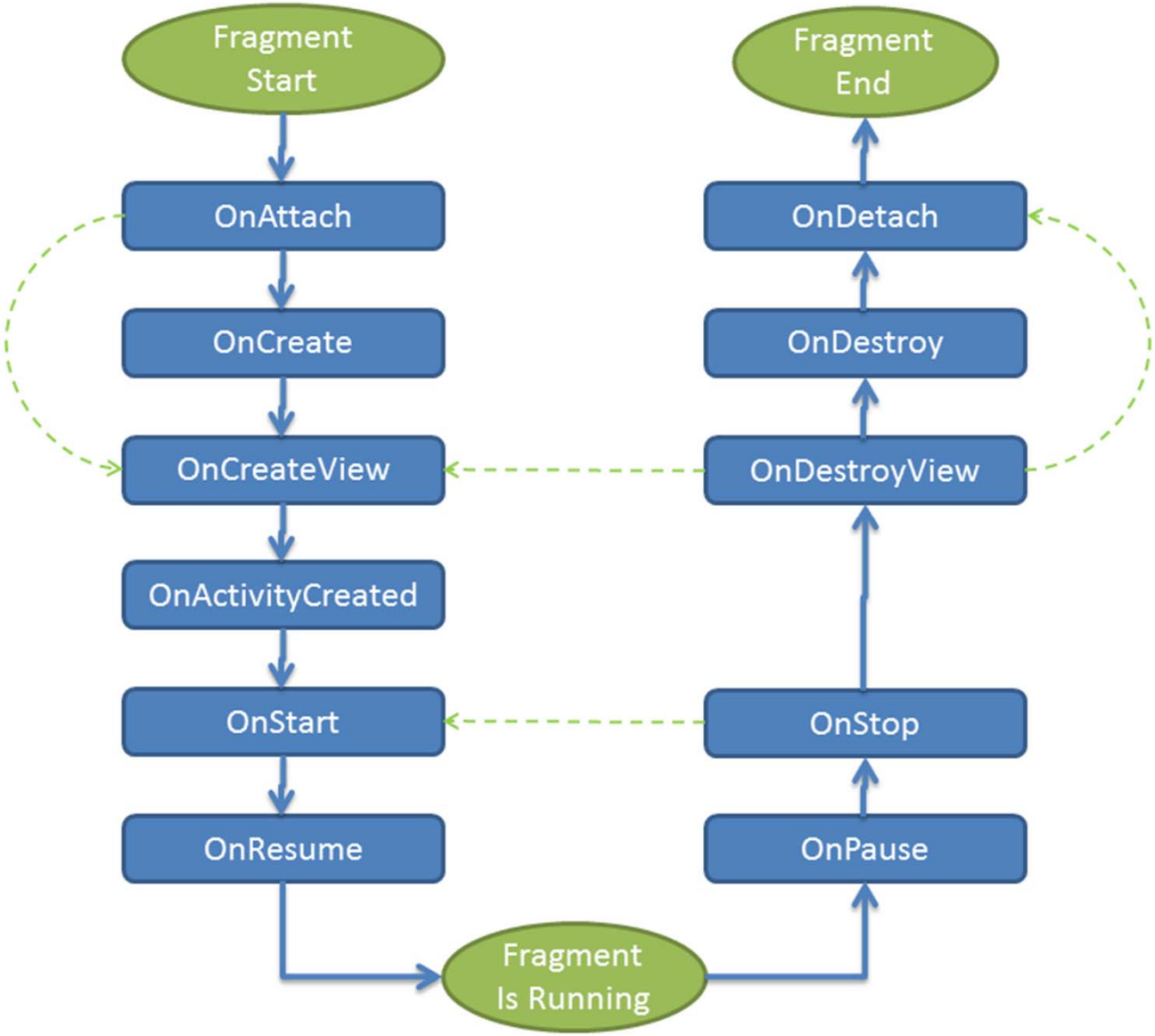


Fragment : Cycle de vie





Fragment : Cycle de vie



Fragment : Définition

- **ATTENTION EVITER LE MELANGE**
  - **android.app.* (version récente)**
    - ✔ getFragmentManager
  - **android.support.v4.app.* (version de compatibilité – support library)**
    - ✔ getSupportFragmentManager
- Nécessite un container
  - Le plus basique: FrameLayout
- On peut définir un id et un tag pour le fragment
  - Soit par son id (ne pas oublier de définir android:id)
  - Soit par son tag (ne pas oublier de définir android:tag)

Fragment : Création

- Fichier XML de layout … As usual …
  - Par exemple création d'un R.layout.fragment_3_main
- Création de la classe Java (étend « Fragment »)
  - Méthode onCreate, onCreateView
  - Méthode onAttach, onDetach
- Méthode onCreateView(), utilisation de « android.view.LayoutInflater »

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_3_main, container, false);
}
```

Fragment : Gestion & Manipulations

- Lors de la création du Main Layout, il est FORTEMENT conseillé d'instancier un fragment par défaut dans les container à fragment !
- EVITEZ DE CRÉER A TOUT VA DES FRAGMENTS
  - Préférer des structures singletons pour les fragments sauf nécessité contraire (ex: multiples instances d'un même type de fragment !)
- Puis pour modifier les fragments de l'activité, on utilise le manager

```
FragmentManager fm = getFragmentManager();
FragmentTransaction ft = fm.beginTransaction();

...

ft.commit();
```

- Méthode sur l'instance de FragmentTransaction
  - replace, add/remove, attach/detach, show/hide
    - ✔ Lors de l'ajout (add), on peut donner un tag particulier à l'instance du fragment
  - setCustomTransition() → transitions graphiques
  - addToBackStack → comportement vis-à-vis du bouton « back » (permettre le retour sur l'ancien Fragment ou non)

Fragment : Gestion & Manipulations

- Add
  - Ajoute le fragment au FragmentManager
  - Rend visible le fragment instancié
  - Durant la transaction isAdded/isRemoved va être MAJ
  - Un fragment ajouté résiste à un changement de conf
- Remove
  - Un fragment enlevé ne sera toujours pas accessible après un changement de conf
  - Un fragment enlevé ne sera plus visible, plus acessible
  - Il sera détruit
- Replace
  - Enlève (remove) TOUS les fragments d'une vue (view)
  - Ajoute (add) le nouveau fragment

Fragment : Gestion & Manipulations

- Attach
  - Rend visible le fragment instancié
  - Durant la transaction isAdded/isRemoved va être MAJ
  - Un fragment attaché résiste à un changement de conf
- Detach
  - Un fragment détaché existera toujours après un changement de conf
  - sa configuration sera préservée pendant un changement de conf
  - Il sera ré-attachable
  - Même détaché, un fragment continue de réagir aux callback (ex: onSaveInstanceState() sera invoqué)
- Show / Hide
  - Change l'état visible/caché d'un fragment
  - L'état visible/caché n'est pas préservé entre les changements de configurations

Fragment : Gestion & Manipulations

- Le Fragment Manager gère les fragments
  - Ne pas oublier de faire le getFragmentManager() sur l'activité
  - Pour les fragments incluant des fragments méthodes getParentFragment(), getChildFragmentManager()
- Savoir si le layout a été ajouté statiquement: isInLayout()
- Récupérer une référence sur un fragment dans un container
  - Mémoriser la référence lors de la création du fragment
    - méthode getId() sur le fragment
    - utiliser putFragment/getFragment sur un Bundle
  - Soit par son id (ne pas oublier de définir android:id)
    Soit par l'id du container (en cherchant l'id du conteneur)

    ```
    getFragmentManager().findFragmentById(R.id.container);
    ```
  - Soit par son tag (ne pas oublier de définir android:tag, ou lors du add)

    ```
    getFragmentManager().findFragmentByTag();
    ```

Eléments GUI:

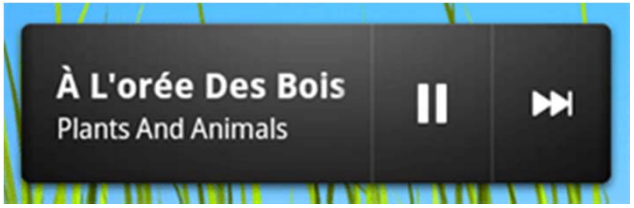
Les Widgets

Définition

- Un plus pour une application !
  - Beaucoup d'applications « modernes » en possède
  - Facebook, SMS, Gmail, etc.
- Permet à l'utilisateur de visualiser d'un coup d'œil les derniers évènements de l'application
  - Pas de badge sur les icônes en Android
  - Les widgets offrent une alternative aux notifications

Définition

- Un plus pour une application !
  - Beaucoup d'applications « modernes » en possède
  - Facebook, SMS, Gmail, etc.
- Permet à l'utilisateur de visualiser d'un coup d'œil les derniers évènements de l'application
  - Pas de badge sur les icônes en Android
  - Les widgets offrent une alternative aux notification
- Tournent en permanence
  - Consomme du CPU → ATTENTION aux ressources !
  - Séparées des applications
  - Nécessité un moyen de communication

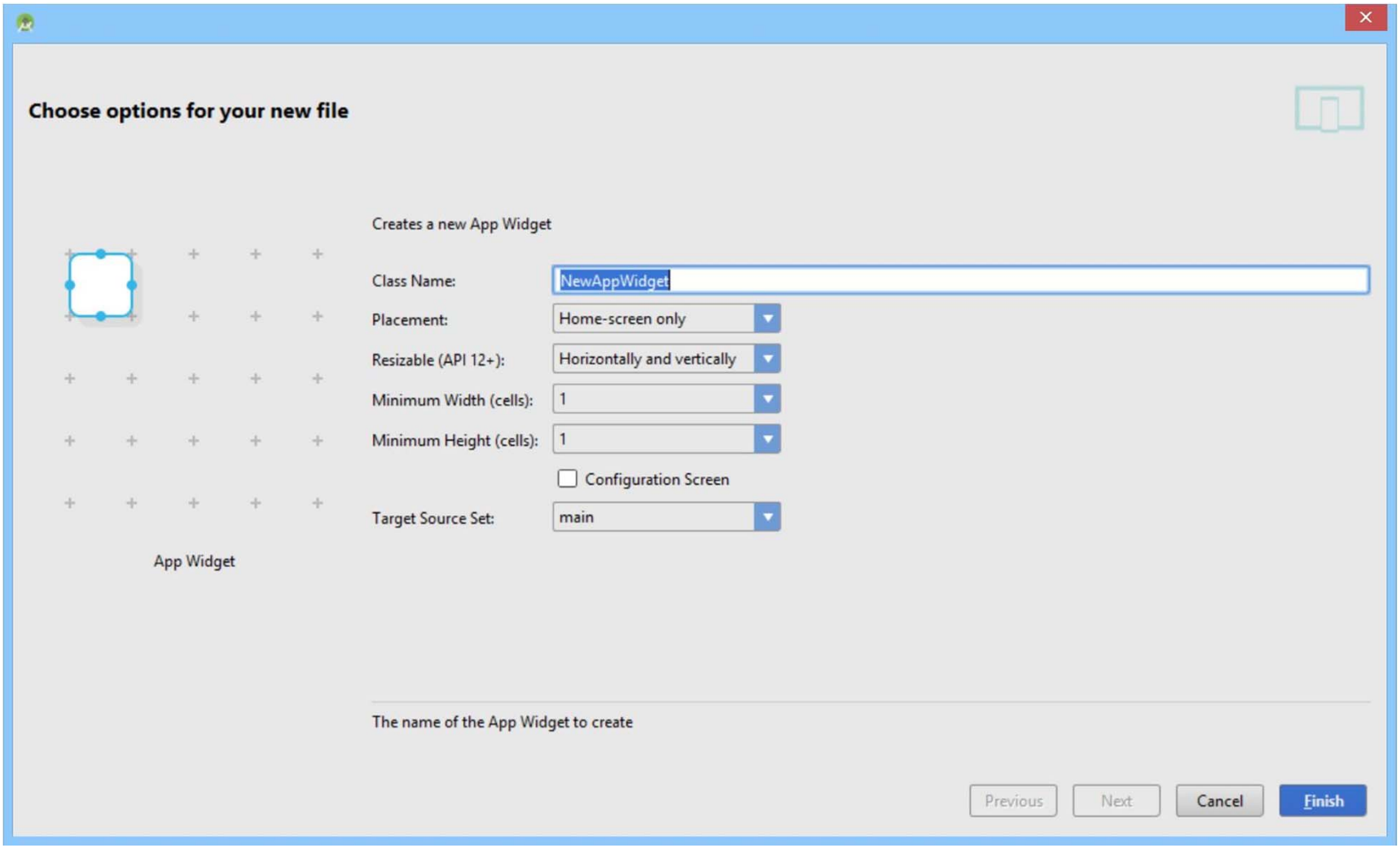


Définition

- Ce sont des applications miniatures qui :
  - reçoivent des mises à jour périodiques
  - peuvent être embarqués dans d'autres applications (HomeScreen)
  - Utilise une surcouche des broadcast receiver
    (Voir les intents)
- Les éléments des widgets :
  - AppWidgetProviderInfo (xml)
    description de la fréquence de MAJ, layout, TAILLE (en dpi) ...
    - ✔ android:minHeight, android:minWidth
  - AppWidgetProvider (class)
    definition des méthodes pour interfacer AppWidget
  - View layout (xml)
    définition du layout du widget
- Généralement on associe aussi une application permettant de configurer le widget

Android-Studio : Nouveau Widget

- Doit cliquer sur Configuration Screen pour créer une activité de configuration



Nouveau widget

- Un widget doit avoir la bonne dimension ! Sinon il est inutilisable…
- AndroidManifest.xml

```
<receiver android:name=".NastySmsWidget" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/nasty_sms_widget_info" />
</receiver>
```

- Fichiers XML (*_info.xml, layout xml)

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="110dp" android:minHeight="110dp" android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/example_appwidget_preview"
    android:initialLayout="@layout/nasty_sms_widget"
    android:configure="legond.fabrice.nastysms.NastySmsWidgetConfigureActivity"
    android:resizeMode="horizontal|vertical" android:widgetCategory="home_screen|keyguard"
    android:initialKeyguardLayout="@layout/nasty_sms_widget">
</appwidget-provider>
```

Nouveau widget

- Fichier de layout classique ...
- Fichier de classe
  - Gère TOUTES les instances de Widget
    - Tableau de int (Widget Id) dans certaines méthodes de callback
  - Doit étendre « AppWidgetProvider »
  - Doit définir les méthodes
    - onEnable → appelée lors de l'ajout sur l'écran d'accueil (homescreen). SEULEMENT la première instance !
    - onDelete → appelé lors de la suppression d'une des instances
    - onDisable → appelée lors de la suppression de la dernière instance du widget sur l'écran d'accueil (HomeScreen)
    - onUpdate → appelée pour chaque MAJ requise. Un id spécifie si besoin l'id de la widget à mettre à jour
- **Les Widgets forment un champ de mines !!! Quelques conseils.**
  - Gérer les bugs avec onDisable/onEnable (Ghost widgets, bug API>4)
  - Gérer le bug du onDelete pas toujours invoqué (API<16). Préférer le onReceived avec les Intents.
  - Gérer le bug des onUpdate sur des widget déjà mort (API<16)
  - Gérer le bug de onUpdate appelé sur un widget non encore configuré
  - Les évènements d'alarme sont délivrés même après la mort d'un Widget
  - Utiliser un thread indépendant et un service
  - Penser à arrêter votre service associé à la Widget

MAJ de l'apparence des widgets

- C'est au programmeur de maintenir les infos spécifiques à chaque WidGet
- Pour récupérer le manager de Widget
  - Singleton / getInstance() sur le context

```
AppWidgetManager awm = AppWidgetManager.getInstance(context);
```

- Pour mettre à jour les widgets (le plus simple):
  - Il faut obtenir la liste des ids des instances des Widgets

```
ComponentName cn = new ComponentName(context, MyWidget.class);
int [ ] awIDs=awm.getAppWidgetIds(cn);
```

  - Ensuite boucler sur toutes les widgets

```
for (int aWid = 0; aWid < awIDs.length; aWid++) {
    RemoteViews view =
                new RemoteViews(context.getPackageName(), R.layout.my_first_widget);
    appWidgetManager.updateAppWidget(aWid, view);
}
```

  - Attention un widget peut mourir ou apparaître pendant la MAJ !!! « synchronized » ?

Widget

Utilité des widgets

- Si la méthode « onReceive » est définie, elle intercepte tous les évènements qui provoque l'invocation des autres méthodes.
  - En résumé, il existe deux méthodes en EXCLUSION
  - Soit on surdéfini onEnable, onDelete, onDisable, onUpdate
  - Soit on utilise onReceive (Context c, Intent i) et les INTENTS
- Deux moyens d'informer un utilisateur d'évènements applicatifs sans le forcer à ouvrir les applications
  - Les widgets, Les notifications
- Très utiles pour reporter les notifications systèmes
- Pour les autres notifications (mails, messages social, …), il faut coupler ces techniques aux services
- Classes utiles
  - AppWidgetManager (getInstance, constantes)
  - AppWidgetProvider
  - RemoteViews / RemoteViewsFactory