Android HCI

Éric Languénou

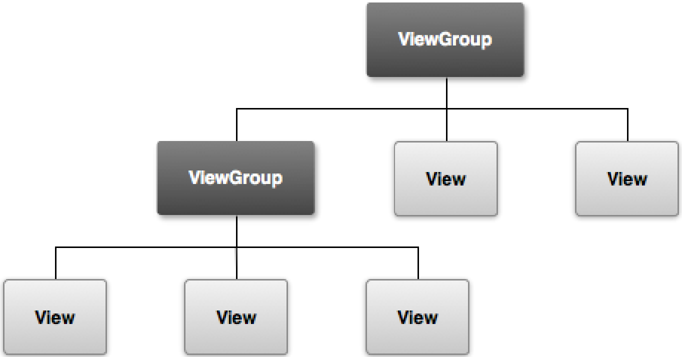
2013-2014

UNIVERSITÉ DE NANTES

Android User Interface

- View :
  - draws something on the screen;
  - the user interacts with it.
- ViewGroup
  - holds Views and ViewGroups;
  - define the UI appearance.
- ressources : `http://developer.android.com/guide/topics/ui/`

User Interface Layout



Layout

UNIVERSITÉ DE NANTES

To declare your layout :

- you can instantiate View objects in code and start building a tree;
- but the easiest and most effective way to define your layout is with an XML file.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Loading the xml file

- each XML layout file is compiled into a View resource;
- in the Activity.onCreate() callback implementation
- calling `setContentView()`

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

More on Layout Objects

- Attributes
  - every View and ViewGroup object supports their own variety of XML attributes (TextView supports the textSize attribute)
  - some are inherited;
- Id
  - Any View object may have an integer ID associated with it, to uniquely identify the View within the tree
  - when compiled, this ID is referenced as an integer,
  - but the ID is typically assigned in the layout XML file as a string, in the id attribute (`android:id="@+id/my_button"`)
  - The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
  - The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file)

How to create and use layouts

to create views and reference them from the application:

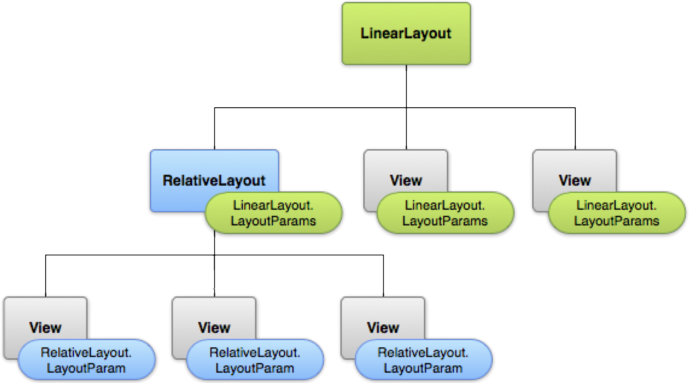
1. Define a view/widget in the layout file and assign it a unique ID:

```
<Button id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

2. Then create an instance of the view object and capture it from the layout (typically in the onCreate() method):

```
    Button myButton = (Button) findViewById(R.id.my_button);
```

Layout Parameters



LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

- the parent view group defines layout parameters for each child view.

Layout Parameters (more)

- All view groups include a width and height (`layout_width` and `layout_height`), and each view is required to define them
- You can specify width and height with exact measurements;
- More often, you will use one of these constants to set the width or height:
  - `wrap_content` tells your view to size itself to the dimensions required by its content
  - `fill_parent` (renamed `match_parent` in API Level 8) tells your view to become as big as its parent view group will allow.
- absolute units such as pixels is not recommended.
- relative measurements such as density-independent pixel units (dp), `wrap_content`, or `fill_parent`, (your application will display properly across a variety of device screen sizes).

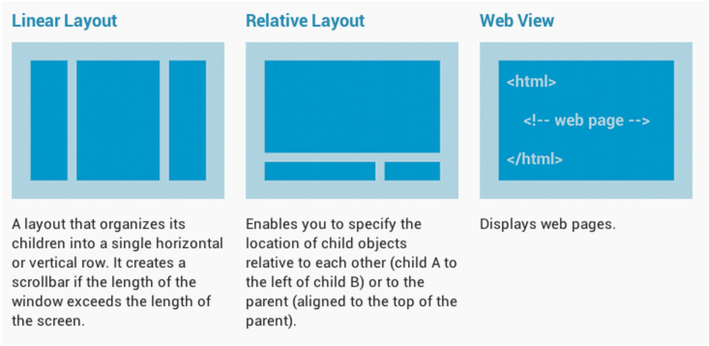
Layout Position

- a view is a rectangle.
- has a **location**, expressed as a pair of left and top coordinates,
  - invoking the methods `getLeft()` and `getTop()`
- and **dimensions** : width and a height.
- the unit is the pixel.
- `getRight()` and `getBottom()`

Size, Padding and Margins

- a view possess two pairs of width an height values;
- width an height
- measuredWidth and measureHeight
- The padding is expressed in pixels for the left, top, right and bottom parts of the view

Common Layouts



Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

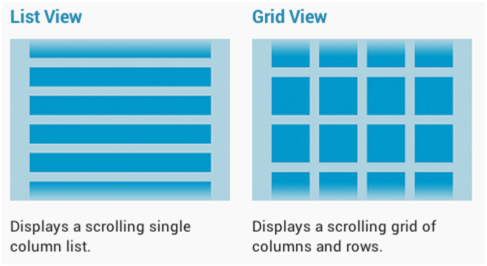
Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

Displays web pages.

Layouts and Adapters



Displays a scrolling single column list.

Displays a scrolling grid of columns and rows.

- if content for your layout is dynamic or not pre-determined
- use a subclasses of AdapterView to populate the layout with views at **runtime**
- A subclass of the AdapterView class uses an Adapter to bind data to its layout.
- The Adapter behaves as a middle-man between the data source and the AdapterView layout

Adapters

UNIVERSITÉ DE NANTES

1. the Adapter retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout.
2. `ArrayAdapter`. If you have an array of strings you want to display in a `ListView`, initialize a new `ArrayAdapter` using a constructor to specify the layout for each string and the string array:
3. `ArrayAdapter adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myStringArray);`
4. Then simply call `setAdapter()` on your ListView:
   ```
   ListView listView = (ListView)
   findViewById(R.id.listview);
   listView.setAdapter(adapter);
   ```

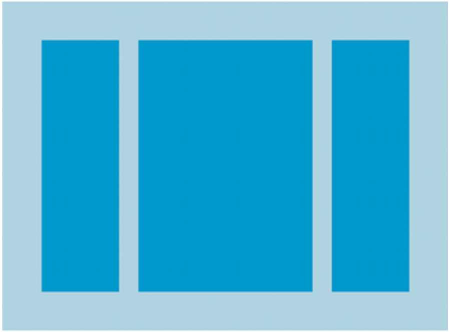
Handling click events

- You can respond to click events on each item in an `AdapterView`
- by implementing the `AdapterView.OnItemClickListener` interface.

```java
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position, long id) {
        // Do something in response to the click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```

Linear Layout

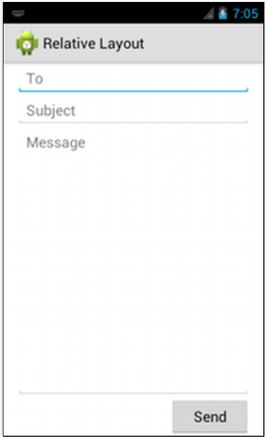


- A LinearLayout respects margins between children
- and the gravity (right, center, or left alignment) of each child
- Layout Weight
  - may assign a weight to individual children with the `android:layout_weight` attribute
  - kind of "importance" value to a view in terms of how much space is should occupy on the screen
  - Default weight is zero.

Linear Layout: Example

UNIVERSITÉ DE NANTES

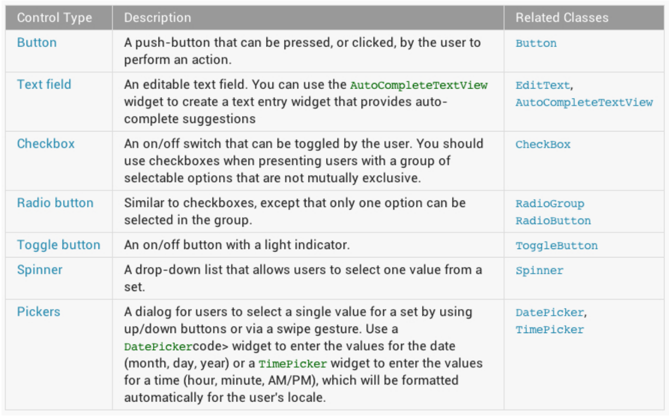
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/r
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```



▶

Input Controls

- interactive components in your app's user interface
- buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons, etc.



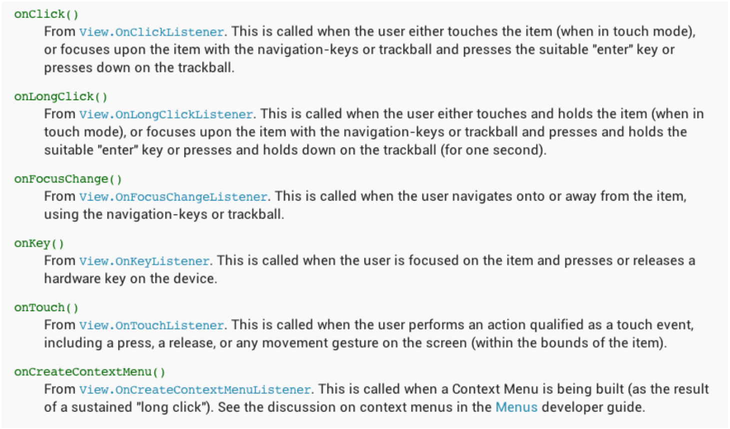
Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	`Button`
Text field	An editable text field. You can use the `AutoCompleteTextView` widget to create a text entry widget that provides auto-complete suggestions	`EditText`, `AutoCompleteTextView`
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	`CheckBox`
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	`RadioGroup` `RadioButton`
Toggle button	An on/off button with a light indicator.	`ToggleButton`
Spinner	A drop-down list that allows users to select one value from a set.	`Spinner`
Pickers	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a `DatePicker`code> widget to enter the values for the date (month, day, year) or a `TimePicker` widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale.	`DatePicker`, `TimePicker`

Input Events

- several ways to intercept the events from a user's interaction with your application;
- events within your user interface : capture the events from the specific View object that the user interacts with;
- when a `View` (such as a `Button`) is touched, the `onTouchEvent()` method is called on that object
- in order to intercept this, you must extend the class and override the method (could be painful);
- the View class also contains a collection of nested interfaces with callbacks that you can much more easily define.
  - These interfaces are called **event listeners**
- to extend a View class, you'll be able to define the default event behaviors for your class using the class **event handlers**

Event Listeners

- is an interface in the View class that contains a single callback method;
- called by the Android framework when the View to which the listener has been registered is triggered by user.



`onClick()`

From `View.OnClickListener`. This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.

`onLongClick()`

From `View.OnLongClickListener`. This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).

`onFocusChange()`

From `View.OnFocusChangeListener`. This is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

`onKey()`

From `View.OnKeyListener`. This is called when the user is focused on the item and presses or releases a hardware key on the device.

`onTouch()`

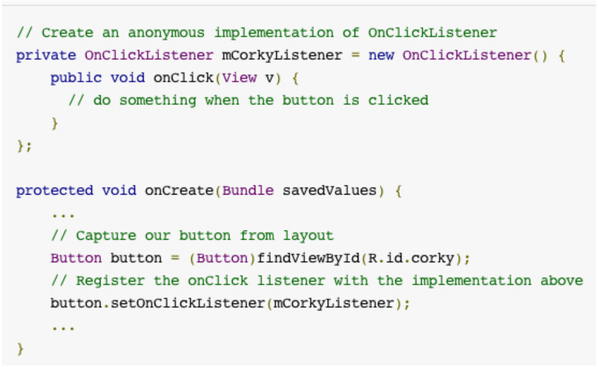
From `View.OnTouchListener`. This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

`onCreateContextMenu()`

From `View.OnCreateContextMenuListener`. This is called when a Context Menu is being built (as the result of a sustained "long click"). See the discussion on context menus in the Menus developer guide.

Event Listeners : HowTo

1. To define one of these methods and handle your events, implement the nested interface in your Activity or define it as an anonymous class.
2. pass an instance of your implementation to the respective `View.set...Listener()` method.
   - (E.g., call `setOnClickListener()` and pass it your implementation of the `OnClickListener`.)

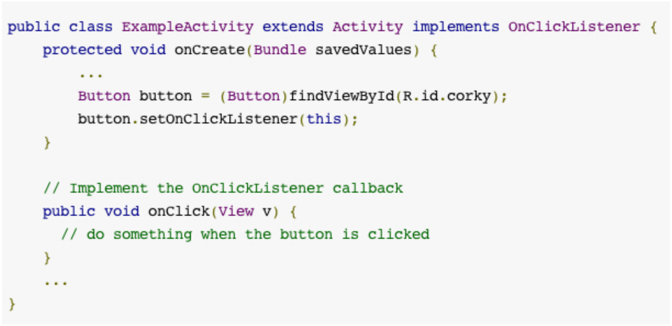


```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

Event Listeners : HowTo

- Or you may implement `OnClickListener` as a part of your `Activity`.



```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

Event Listeners : HowTo

UNIVERSITÉ DE NANTES

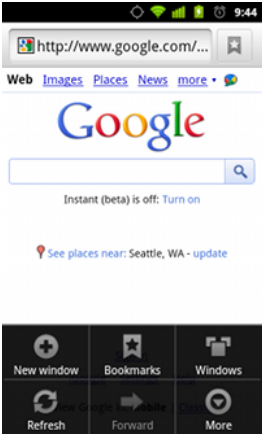
- `onClick()` callback in the previous example has no return value,
- some other event listener methods must return a boolean. The reason depends on the event.
- hardware key events are always delivered to the View currently in focus.
- they are dispatched starting from the top of the View hierarchy, and then down, until they reach the appropriate destination.
- if your `View` (or a child of your View) currently has focus, then you can see the event travel through the `dispatchKeyEvent()` method

Menus

UNIVERSITÉ DE NANTES

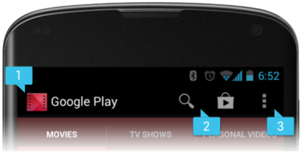
- Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button;
- three fundamental types of menus or action presentations on all versions of Android:
  - Options menu and action bar
  - Context menu and contextual action mode
  - Popup menu

Options menu


9:44
http://www.google.com/...
Web
Images
Places
News
more
Google
Instant (beta) is off: Turn on
See places near: Seattle, WA - update
New window
Bookmarks
Windows
Refresh
Forward
More

Options menu in the Browser, on Android 2.3

Action Bar

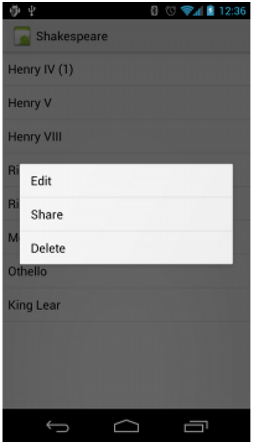


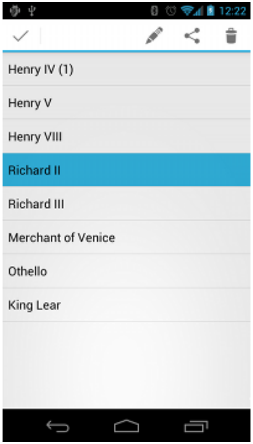
Action bar from the Honeycomb Gallery app, showing navigation tabs and a camera action item (plus the action overflow button).

Options menu and action bar

- The options menu is the primary collection of menu items for an activity.
- place for actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."
- On Android 3.0 and higher, items from the options menu are presented by the **action bar** as a combination of on-screen action items and overflow options.

Context menu

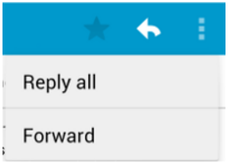




Context Menu and Contextual Action Mode

- a floating menu that appears when the user performs a long-click on an element.
- It provides actions that affect the selected content or context frame.
- when developing for Android 3.0 and higher,
  - you should instead use the contextual action mode to enable actions on selected content.
  - this mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

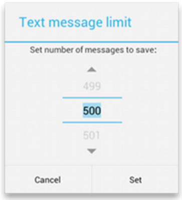
Popup Menu



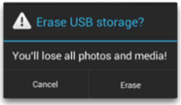
Popup menu

- displays a list of items in a vertical list that's anchored to the view that invoked the menu.
- good for providing
  - an overflow of actions that relate to specific content
  - to provide options for a second part of a command.
- actions in a popup menu should not directly affect the corresponding content (contextual actions are dedicated to this).
- rather, the popup menu is for extended actions that relate to regions of content in your activity.

Dialogs



Set number of messages to save:



You'll lose all photos and media!

- small window that prompts the user to make a decision or enter additional information.
- does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.
- use one of the following subclasses:
  - AlertDialog
    - show a title, up to three buttons, a list of selectable items, or a custom layout.
  - DatePickerDialog or TimePickerDialog
    - dialog with a pre-defined UI that allows the user to select a date or time.

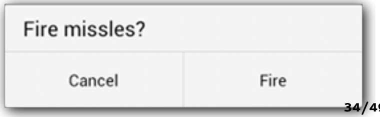
- These classes define the style and structure for your dialog;
- you should use a DialogFragment as a container for your dialog;
- `DialogFragment` class provides :
  - all the controls you need to create your dialog
  - and manage its appearance,
  - instead of calling methods on the Dialog object.
- it ensures that it correctly handles lifecycle events such as when the user presses the Back button or rotates the screen.

Dialog Fragment

a basic AlertDialog that's managed within a DialogFragment:

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // FIRE ZE MISSILES!
                   }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // User cancelled the dialog
                   }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

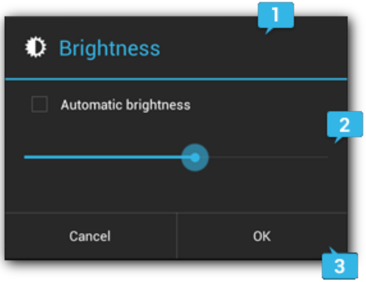
- create an instance of this class
- call `show()` on that object.


Fire missles?

Alert Dialog

The AlertDialog class allows a variety of dialog designs;
There are three regions of an alert dialog:

1. Title
   - optional;
   - should be used only when the content area is occupied by a detailed message, a list, or custom layout.
2. Content area
   - displays a message, a list, or other custom layout.
3. Action buttons
   - no more than three action buttons in a dialog.



Building an AlertDialog

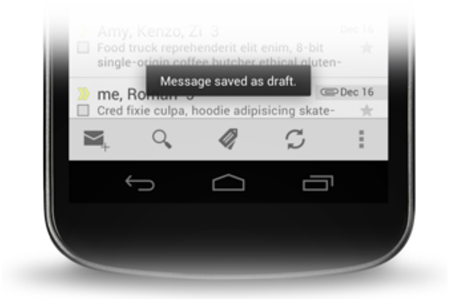
The `AlertDialog.Builder` class provides APIs that allow you to create an AlertDialog with these kinds of content, including a custom layout.

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
       .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

Toasts


Amy, Kenzo, Zi 3
Dec 16
Food truck reprehenderit elit enim, 8-bit
single-origin coffee butcher ethical gluten-
Message saved as draft.
me, Roman 3
Dec 16
Cred fixie culpa, hoodie adipisicing skate-

- provides simple feedback about an operation in a small popup;
- it only fills the amount of space required for the message
  - and the current activity remains visible and interactive.
- for example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later;
- toasts automatically disappear after a timeout.

Toast basics

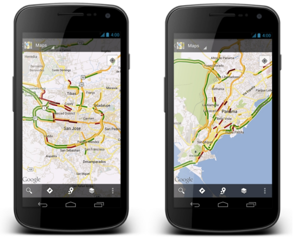
- instantiate a Toast object with one of the `makeText()` methods.
  - this method takes three parameters:
    - the application Context,
    - the text message,
    - the duration for the toast.
  - it returns a properly initialized Toast object.
  - you can display the toast notification with `show()`.

```java
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Location and Maps

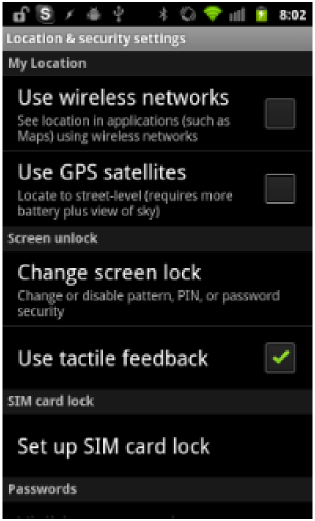
UNIVERSITÉ DE NANTES



- location and maps-based apps offer a compelling experience on mobile devices.
- use the classes of the `android.location package` and the `Google Maps Android API`
- `Google Location Services API` is more recent (provides high-level methods, activity detection, etc.)

Location Services

- the `LocationManager` system service is the central component of the location framework
- it provides APIs to determine location and bearing of the underlying device (if available).
  - send request an instance from the system by calling `getSystemService(Context.`
  - returns a handle to a new `LocationManager` instance.


8:02
Location & security settings
My Location
Use wireless networks
See location in applications (such as Maps) using wireless networks
☐
Use GPS satellites
Locate to street-level (requires more battery plus view of sky)
☐
Screen unlock
Change screen lock
Change or disable pattern, PIN, or password security
Use tactile feedback
✓
SIM card lock
Set up SIM card lock
Passwords

- Once your application has a `LocationManager`
- your application is able to do three things:
  1. query for the list of all `LocationProviders` for the last known user location.
  2. register/unregister for periodic updates of the user's current location from a location provider (specified either by criteria or name).
  3. register/unregister for a given Intent to be fired if the device comes within a given proximity (specified by radius in meters) of a given lat/long.

Google Maps Android API

using the Google Maps Android API [1]

- you can add maps to your app that are based on Google Maps data;
- the API automatically handles
  - access to Google Maps servers,
  - data downloading,
  - map display,
  - and touch gestures on the map.
- You can also use API calls to
  - add markers, polygons and overlays,
  - to change the user's view of a particular map area.

[1] first, install the Google Play services libraries for your Android SDK

MapView

`MapView` is the key class in the Google Maps Android API.

- a MapView displays a map with data obtained from the Google Maps service.
- when the MapView has focus, it will capture keypresses and touch gestures to pan and zoom the map automatically, including handling network requests for additional maps tiles.
- it provides all of the UI elements necessary for users to control the map.
- applications can also use MapView class methods to control the map programmatically and draw a number of overlays on top of the map.

Sensors

- Most Android-powered devices have built-in sensors that measure
  - motion, orientation, and various environmental conditions.
- these sensors are capable of providing raw data with high precision and accuracy
  - to monitor three-dimensional device movement or positioning,
  - to monitor changes in the ambient environment near a device.

Three Categories of Sensors

- Motion sensors
  - acceleration forces and rotational forces along three axes.
  - includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- Environmental sensors
  - various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity;
  - includes barometers, photometers, and thermometers.
- Position sensors
  - physical position of a device;
  - includes orientation sensors and magnetometers.

Android sensor framework

Android sensor framework allows to :

- determine which sensors are available on a device.
- determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- register and unregister sensor event listeners that monitor sensor changes.

Sensor availability :

- varies from device to device,
- vary between Android versions.

Sensor Types

- Hardware-based sensors are **physical components** built into a handset or tablet device.
  - derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change.
- Software-based sensors are **not physical devices**, although they mimic hardware-based sensors.
  - derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors.
  - linear acceleration sensor and the gravity sensor are examples of software-based sensors.
- sensor types can be found at `http://developer.android.com/guide/topics/sensors/sensors_overview.html`

Sensor Framework

`Sensor framework` is part of the `android.hardware package` and includes the following classes and interfaces:

- SensorManager
  - create an instance of the sensor service;
  - provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information
  - provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.
- Sensor
  - create an instance of a specific sensor
  - provides various methods that let you determine a sensor's capabilities.

- SensorEvent
  - the system uses this class to create a sensor event object, which provides information about a sensor event.
  - sensor event object includes the following information:
    - the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.
- SensorEventListener
  - use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.