Lezione 10

Programmazione Android

- Ancora sulla UI (ma poi basta!)
  - WebView – una vista tuttofare
  - Drawable
  - Notifiche all'utente
    - Toast
    - Notifications
    - Dialog
  - Fragment
    - DialogFragment

WebView

La View WebView

- Una WebView è una View che incapsula un web browser
  - Fornisce tutte le funzioni di base per caricare una URL e visualizzarla all'utente
    - Compresa l'interazione: scrolling, ecc.
  - Fornisce anche funzioni più specifiche per intercettare funzioni particolari
    - Esecuzione Javascript, gestione cookie, find dialog, ecc.
    - Errori di rete, completamento download, scroll & zoom da programma

WebView vs. Browser

- Eseguire “esternamente” il browser
  - Aggiunge i controlli, la history, i preferiti, ecc.

```
Uri uri = Uri.parse("http://www.di.unipi.it");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

- Incorporare invece la webview in una activity

```
WebView web = getViewById(R.id.web);
web.loadUrl("http://www.di.unipi.it");
```

  - o anche

```
String html = "<html><body><h1>Titolo</h1>Corpo</body></html>";
web.loadData(html, "text/html", null);
```

WebView - chrome

- La WebView standard è del tutto adeguata per **presentare contenuti** in HTML
- Richiede però ulteriore lavoro per regolare impostazioni più di dettaglio
- Elementi di UI custom “intorno” alla WebView
  - Si implementa una sottoclasse di **WebChromeClient**
    - Avanzamento del caricamento: onProgressChanged()
    - Dialog di Javascript: onJsAlert(), onJsConfirm(), onJsPrompt(), ...
    - Elementi della pagina: onReceivedIcon(), onReceivedTitle()
    - Debugging: onConsoleMessage()
    - ecc.

WebView - settings

- La WebView standard è del tutto adeguata per **presentare contenuti** in HTML
- Richiede però ulteriore lavoro per regolare impostazioni più di dettaglio
- Impostazioni della WebView
  - Si invocano metodi di **WebSettings**
    - Gestione delle cache: setAppCacheEnabled(), setAppCacheMaxSize(), setAppCachePath(), ...
    - Permessi: setAllowContentAccess(), setAllowFileAccess(), setJavascriptEnabled(), setPluginEnabled(), setPluginState(), ...
    - Cosa caricare: setBlockNetworkImage(), setBlockNetworkLoads(), ...
    - Aspetto: setDefaultFontSize(), setMinimumFontSize(), setCursiveFontFamily(), ...
    - ecc.

WebView Javascript bridge

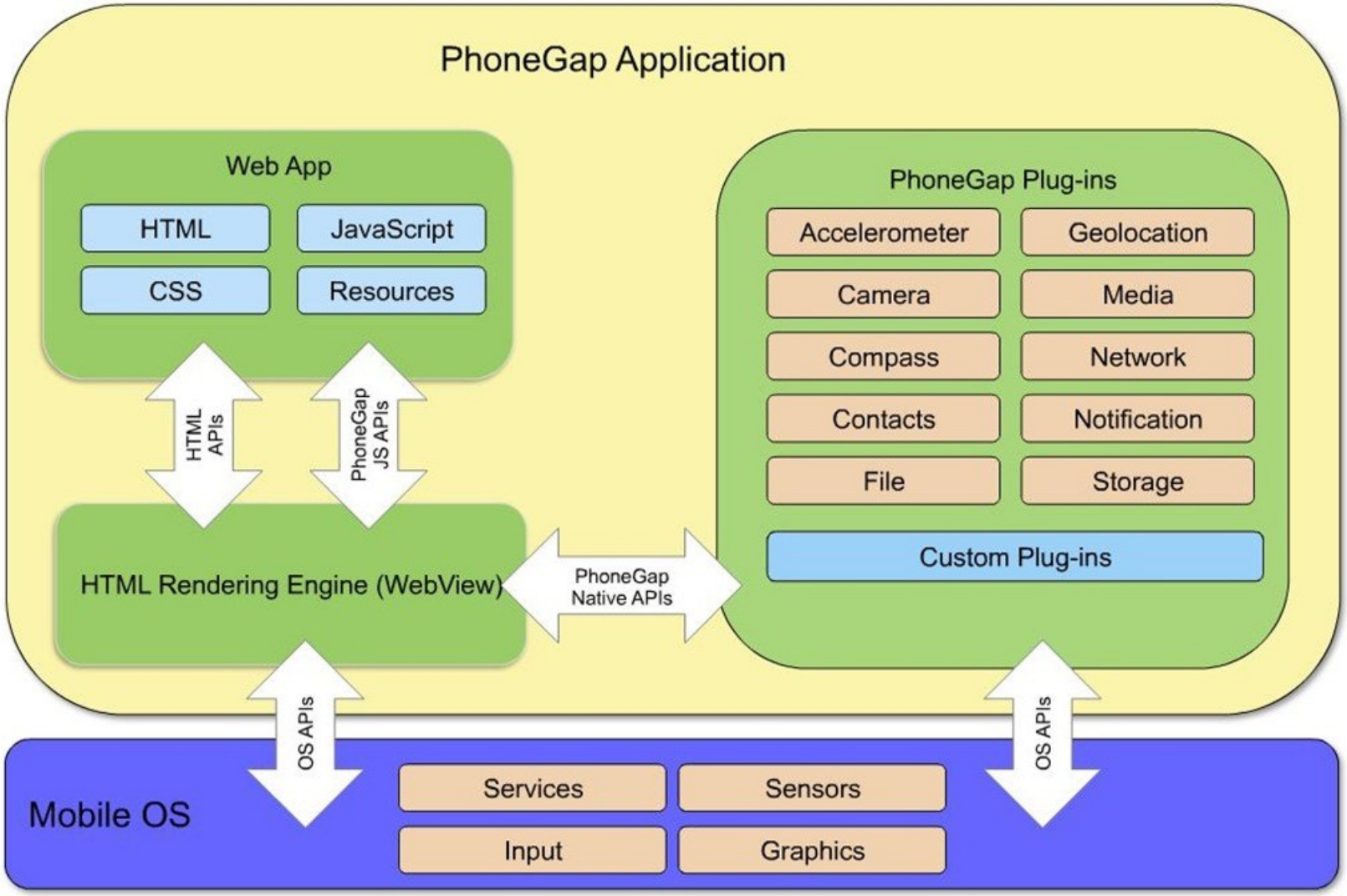
- La WebView esegue nativamente il codice Javascript presente nella pagina
  - Possibile anche passare una URL nella forma javascript:funzione()
- È anche possibile effettuare il **binding** fra oggetti Java (dell'App) e oggetti Javascript (nella pagina)
  - addJavascriptInterface(oggetto, nome)
- L'oggetto Java diventa accessibile agli script in Javascript della pagina, con il nome dato
  - Tecnica un po' rischiosetta in termini di sicurezza!

WebApps

- Questa tecnica è usata da alcuni ambienti di sviluppo cross-platform basati su **tecnologie web**
  - Si scrive una web app “tradizionale”, client-side
    - HTML5, CSS3, Javascript
    - Jquery, Bootstrap, node.js, …
  - La si esegue dentro una WebView “arricchita”
    - L'ambiente aggiunge una serie di oggetti predefiniti, scritti in Java, che fanno da “ponte”
    - Così è possibile chiamare da Javascript i metodi delle varie classi di Android che non sono emulabili da HTML5
  - Tutta la grafica è renderizzata sul DOM

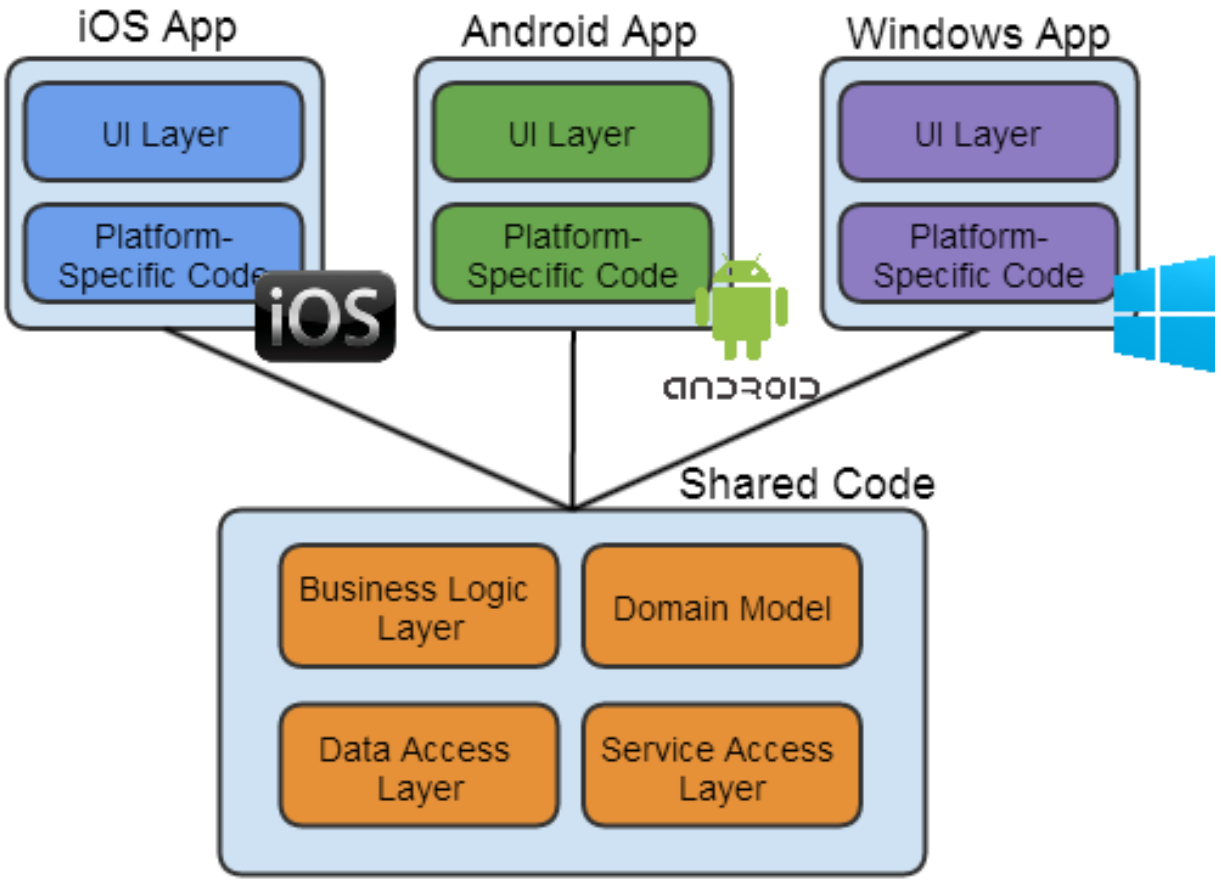
PhoneGap / Cordova
(Adobe) (Apache)

- Tecnologie utili per una sveltina
- È dura realizzare app che vengono prese sul serio
- Caso tipico: **Facebook**
- Dopo anni di tentativi, ha dovuto buttare via la sua app cross-platform HTML5 e scrivere 3 app native



In contrasto: Xamarin

- Esistono altri approcci al cross-platform



- Non basati sul web
- Noi vedremo **Xamarin**:
  - Un ambiente .NET (porting di Mono)
  - "stesso" codice gira sulla VM .NET
  - Ma incoraggiata UI nativa

Drawable

Drawable

- La classe **Drawable** è la superclasse delle “cose che possono essere disegnate”
  - Esistono nel sistema molte sottoclassi specializzate, ed è possibile definire le proprie (ereditando)
    - BitmapDrawable, ClipDrawable, ColorDrawable, DrawableContainer, GradientDrawable, InsetDrawable, LayerDrawable, NinePatchDrawable, PictureDrawable, RotateDrawable, ScaleDrawable, ShapeDrawable
- Metodi setter e getter per molte proprietà grafiche
  - E alcune atipiche, come “level” - utile per le progress bar o altre indicazioni con *modello* numerico

Creare Drawable

- Un Drawable può essere definito in un file XML come risorsa
- Esempio (definisce uno ShapeDrawable): res/drawable/gradient_box.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <corners android:radius="8dp" />
</shape>
```

Creare Drawable

- Il Drawable può poi essere recuperato dal programma con un normale accesso alle risorse:

```
Resources res = getResources();
Drawable shape = res.getDrawable(R.drawable.gradient_box);
```

- … e usato (fra l'altro) per disegnare su un Canvas

```
shape.set<dozzilioni di proprietà>(…);
shape.draw(canvas);
```

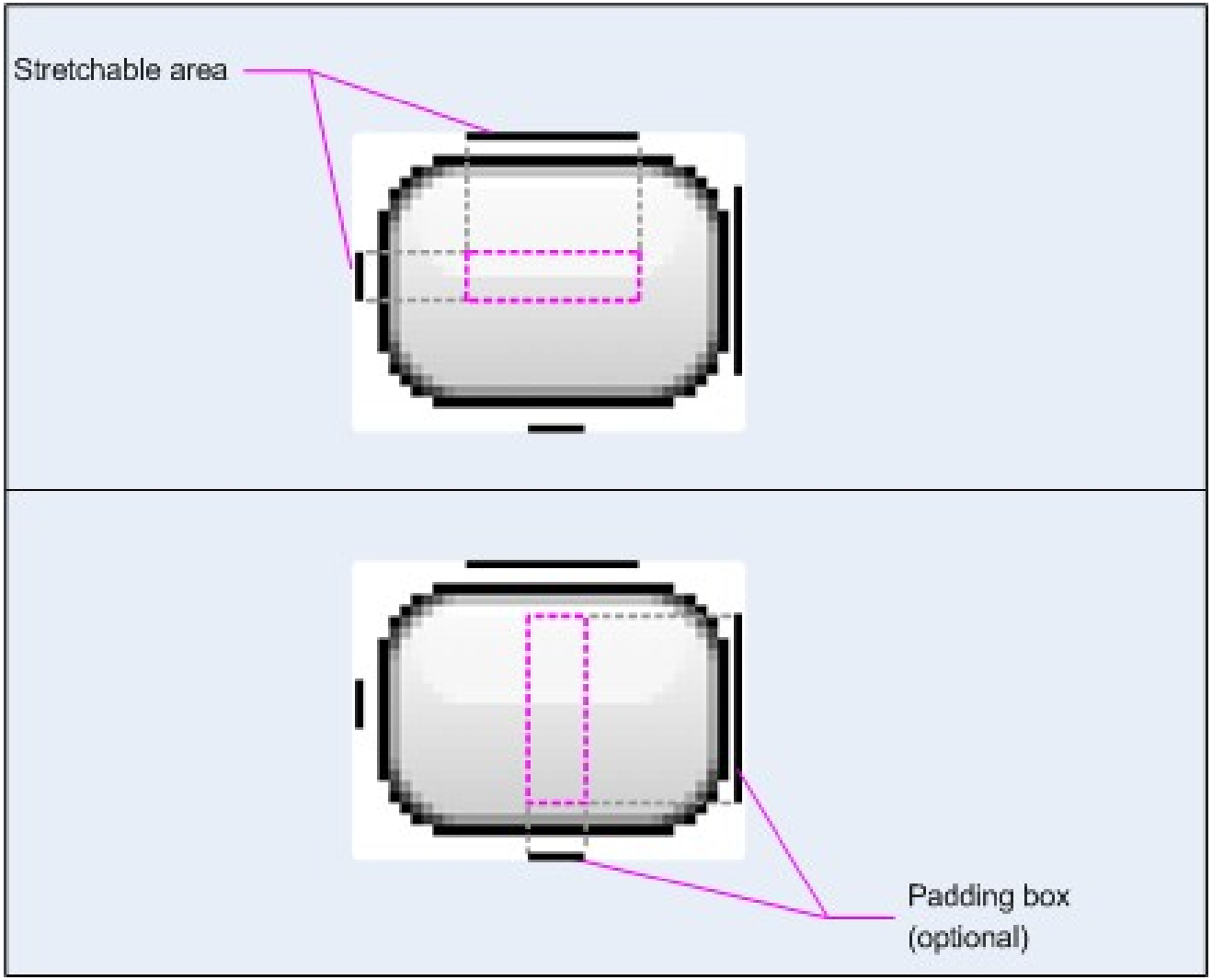
- … o come sfondo di una View

```
TextView tv = (TextView)findViewByID(R.id.textview);
tv.setBackground(shape);
```

Creare Drawable

- Una immagine (.png, .jpg, .gif) salvata fra le risorse è un BitmapDrawable
  - La si può recuperare con resources.getDrawable()
- Una immagine *"nine patch"* salvata fra le risorse è un NinePatchDrawable
  - La si può recuperare con resources.getDrawable()
- Si può istanziare un Drawable (o sottoclasse, anche custom) direttamente a programma
  - Drawable d=new *xxx*Drawable(...);

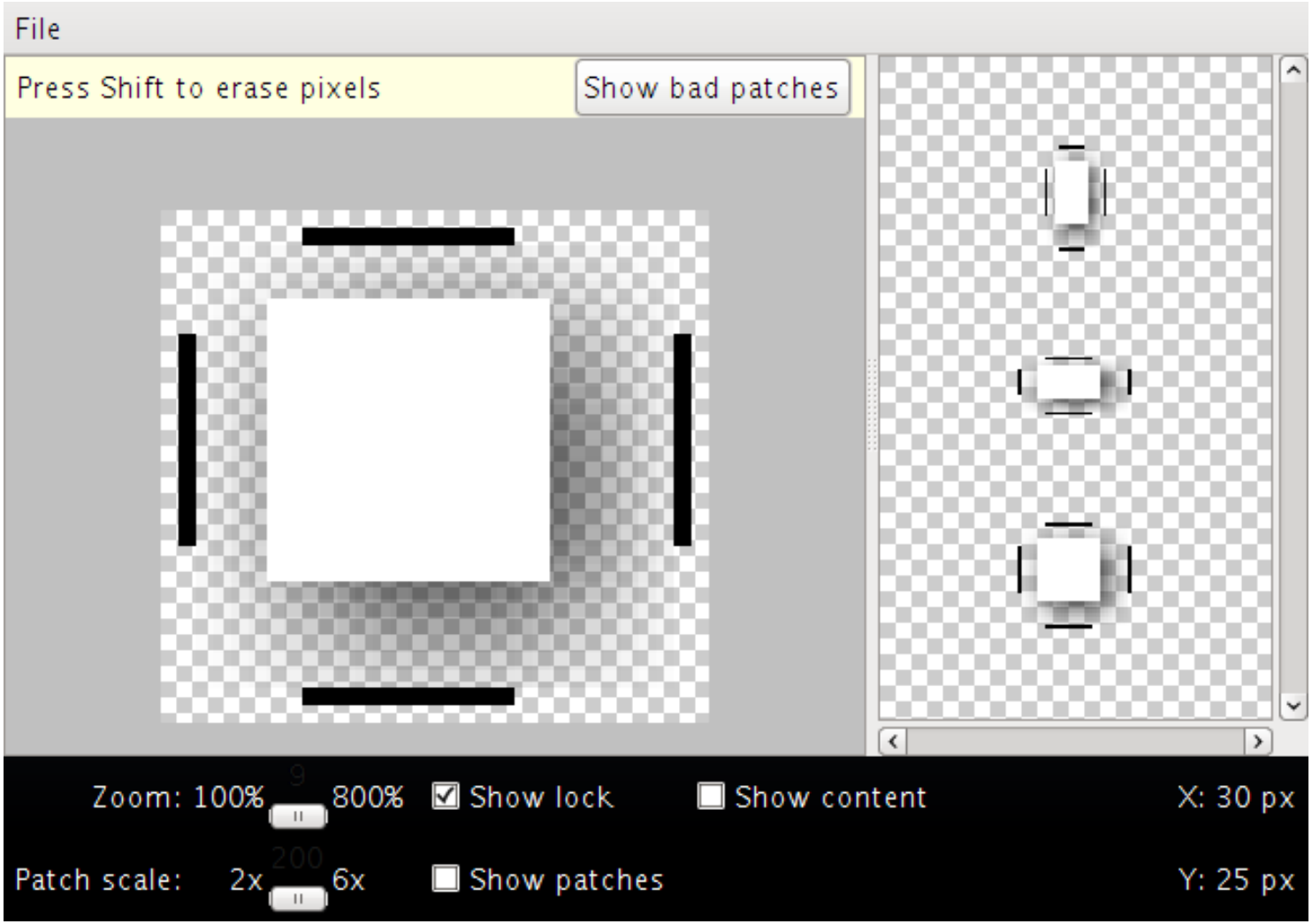
Il formato “nine patch”



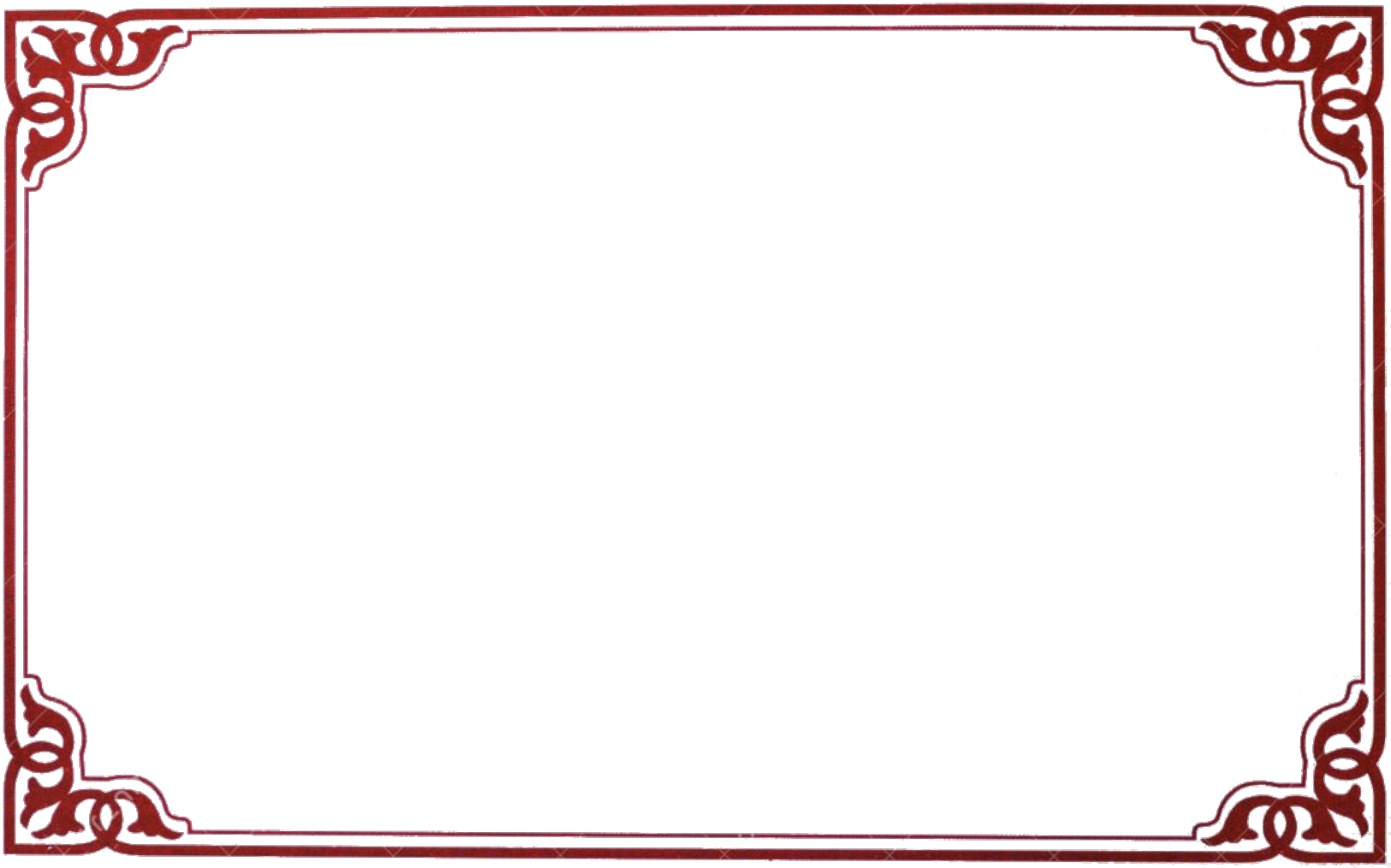
- Un .png con una cornice extra di 1 pixel
- Usata per dividere l'immagine in 9 regioni
- Stretching arbitrario!

Draw-9-patch

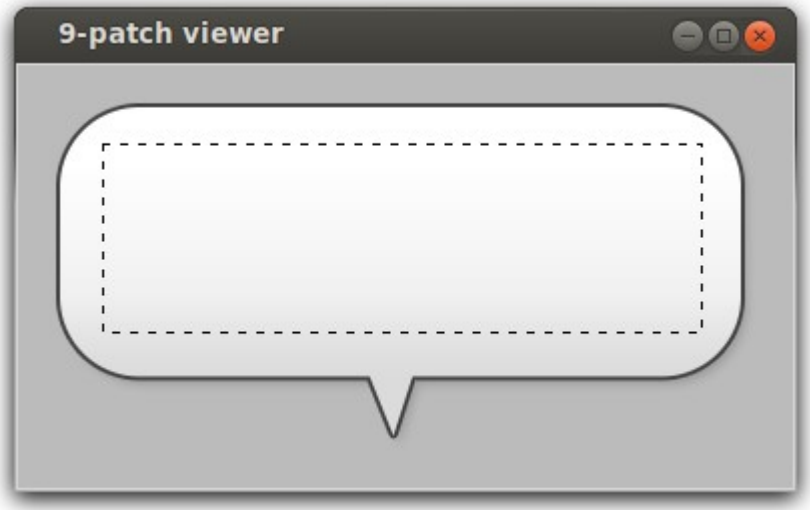
- Tool GUI per trasformare un .png in un .9.png
- Fra i tools dell'SDK
  - draw9patch

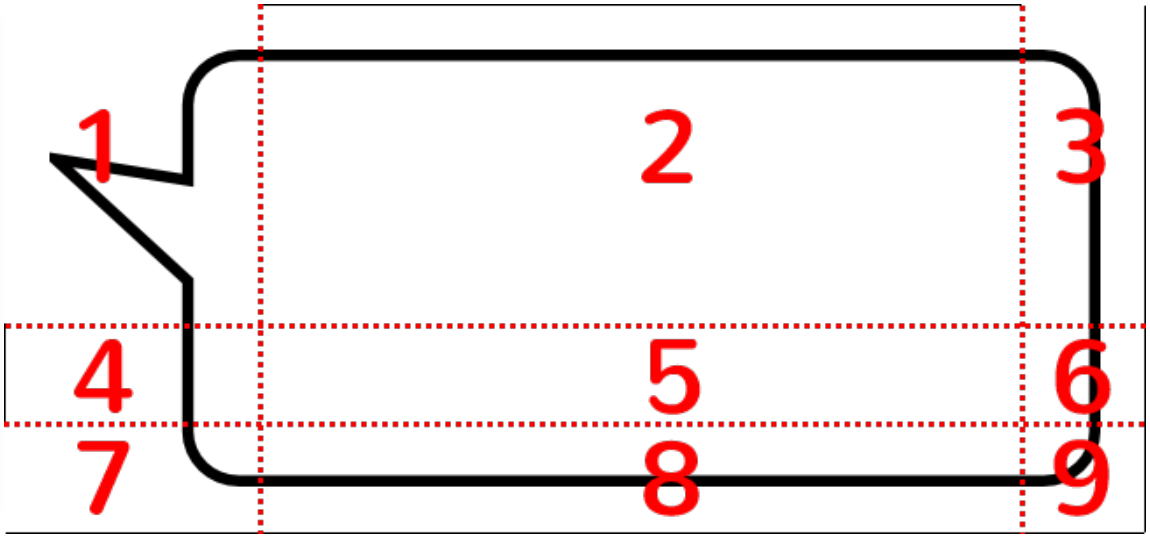


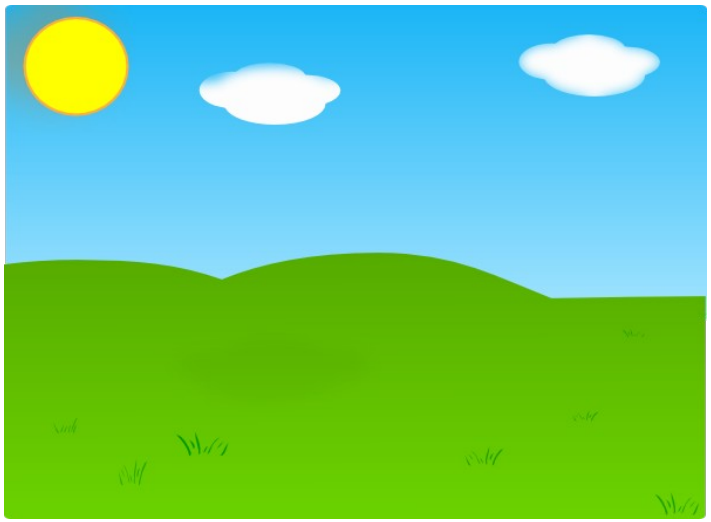
9 patch: Esempi



9 patch: Esempi







Altri Drawable

- **LayerDrawable**
  - Raccolta di altri Drawable disegnati uno sull'altro
- **StateDrawable**
  - Raccolta di altri Drawable, di cui viene disegnato quello corrispondente allo **stato** corrente
    - Es.: vari stati di un pulsante: selezionato, attivo, premuto, disabilitato...
- **LevelDrawable**
  - Raccolta di altri Drawable, di cui viene disegnato quello corrispondente al **livello** corrente
    - Es.: più immagini distinte per carica di batteria o forza del segnale wi-fi
- **ScaleDrawable**
  - Un drawable che ne contiene un altro, che viene disegnato in scala proporzionale al **livello** corrente

Altri Drawable

- **ColorDrawable**
  - Una superficie di colore piatto dato
- **GradientDrawable**
  - Una superficie con un gradiente dato
- **AnimationDrawable**
  - Un drawable animato (con varie tecniche: morphing, tweening, ecc.)
- **TransitionDrawable**
  - Un drawable che mostra una transizione (fade) fra altri due o più drawable (con tempo di transizione dato)

Esempio: TransitionDrawable

- Res/drawable/transition.xml

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/image_plus">
  <item android:drawable="@drawable/image_minus">
</transition>
```

- Caricamento

```
Resources r = context.getResources();
TransitionDrawable trans = (TransitionDrawable)res.getDrawable(R.drawable.transition);
ImageView image = (ImageView) findViewById(R.id.toggle);
image.setImageDrawable(trans);
```

- Avvio

```
trans.startTransition(500); oppure trans.reverseTransition(500);
```

Esempio: AnimationDrawable

- Animazione stile “tweening” (intercalazione)

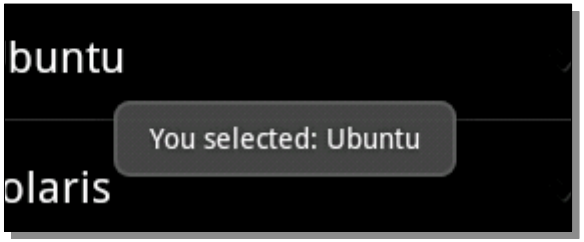
```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/f1" android:duration="200" />
    <item android:drawable="@drawable/f2" android:duration="200" />
    <item android:drawable="@drawable/f3" android:duration="200" />
</animation-list>
```

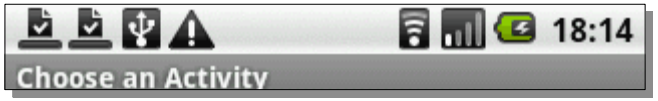
```
fb = (ImageView)findViewById(R.id.fb);
fb.setBackgroundResource(R.drawable.animation);
AnimationDrawable ad = (AnimationDrawable)fb.getBackground();

…
ad.start();
…
ad.stop();
```

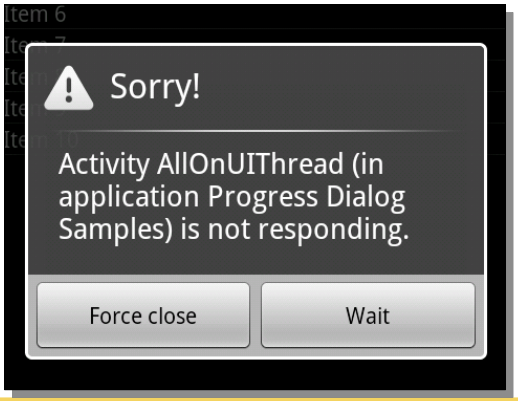
Notifiche all'utente

Forme di notifica

- Android prevede tre forme di notifica per l'utente:
  - **Toast** – brevi popup che appaiono “sopra” altre activity, e scompaiono automaticamente
  - **Status bar** – icone nella barra in alto
  - **Dialog** – finestre tradizionali, con possibilità di interazione dell'utente







Toast

- Semplici, veloci, poco intrusivi
  - Bisogna però essere certi che l'utente sta guardando!
- Caso più tipico:
  Toast.*makeText*(context, text, duration).show();
  - **context** – il Context da cui viene il toast (può essere l'Activity, o l'Application se il toast viene da un Service)
  - **text** – l'ID di risorsa di una stringa, oppure una stringa
  - **duration** – per quanto tempo il toast deve essere mostrato
    - A scelta fra Toast.LENGTH_LONG e Toast.LENGTH_SHORT
    - Il default è SHORT; l'esatta durata potrebbe essere configurabile

Toast

- Anziché visualizzare subito un Toast con show(), lo si può creare, e (ri-)usare in seguito
  - setText() - cambia il testo di un toast esistente
  - cancel() - chiude anzitempo un toast aperto
  - setGravity(), setMargin() - parametri di layout
  - setView() - per visualizzare in un toast una propria View
    - makeText() non fa altro che impostare una view predefinita

Toast – esempio di customizzazione

```
public class CustomToastActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        View layout = getLayoutInflater().inflate(R.layout.toast,
                                       (ViewGroup) findViewById(R.id.toast_root));

        ImageView image = (ImageView) layout.findViewById(R.id.image);
        image.setImageResource(R.drawable.ic_launcher);

        TextView text = (TextView) layout.findViewById(R.id.text);
        text.setText("Sono un toast custom!");

        Toast toast = new Toast(this);
        toast.setGravity(Gravity.CENTER_VERTICAL, 0, -60);
        toast.setDuration(Toast.LENGTH_LONG);
        toast.setView(layout);
        toast.show();
    }
}
```

Toast – esempio di customizzazione

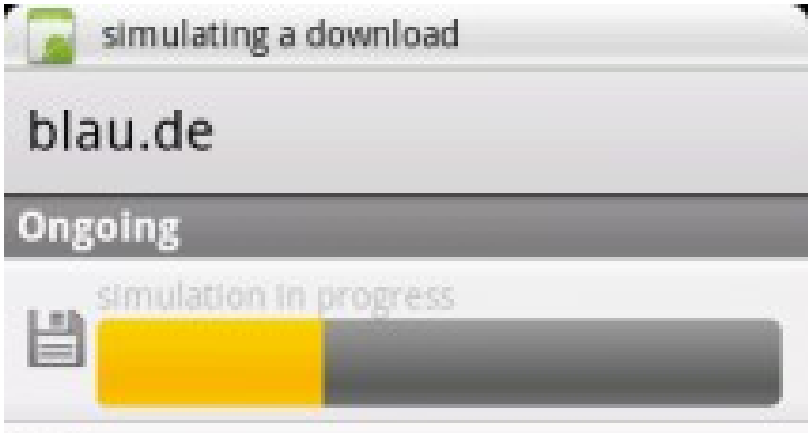
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/toast_root"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="16dp"
        android:background="#D8B8"
        >
   <ImageView android:id="@+id/image"
         android:layout_width="wrap_content"
         android:layout_height="fill_parent"
         android:layout_marginRight="16dp"
         />
   <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:textColor="#EEE"
        />
</LinearLayout>
```

Status bar notification

- Le notifiche su barra di stato sono generalmente utilizzate per segnalare eventi **in background**
- Inizialmente, WP e iOS avevano difficoltà a farlo
  - Successivamente aggiunte forme limitate di esecuzione in background
  - Solitamente utilizzate da servizi di sistema, ovvero dietro particolari contratti con i “big player”
  - Ora si sono allineati anche loro…
- Su Android, possono essere generate da Activity (in foreground) o da Service (in background)

Status bar notification

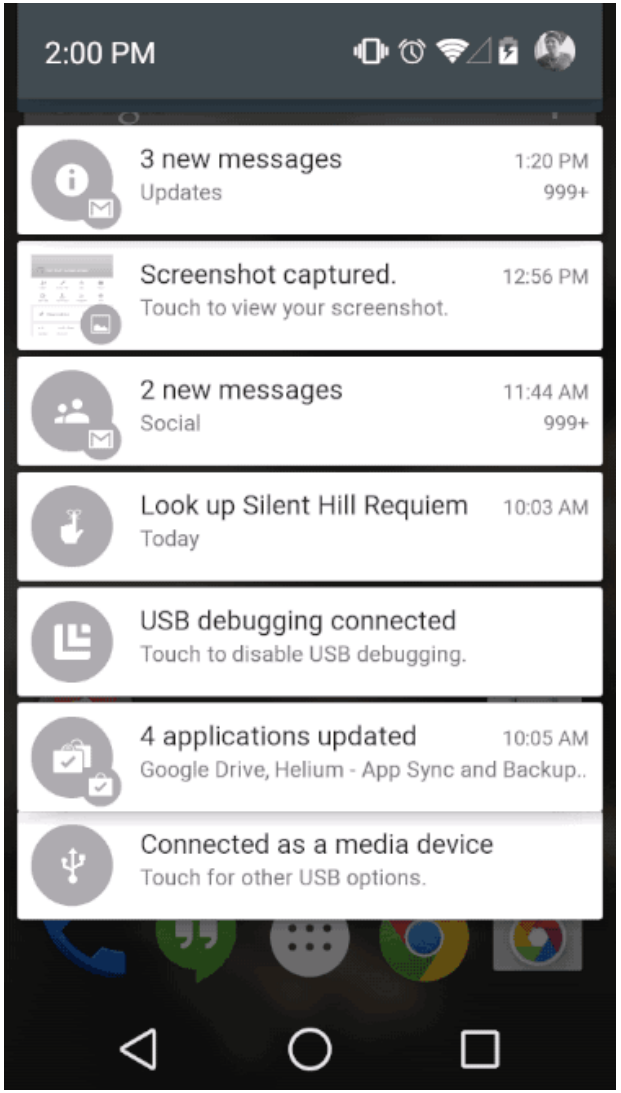
- Le notifiche hanno un ciclo di vita **distinto** da quello del componente che le ha generate
  - In particolare, rimangono attive finché l'utente (o chi le ha generate) non le cancella
    - Anche se nel frattempo l'Activity che le ha create è morta!
  - Possono anche essere animate, o fornire informazioni di progresso
    - Per esempio, il download di una app dal Play Store, una lunga operazione di copia, la riproduzione di un brano audio...



Notification e NotificationManager

- **Notification**
  - Oggetti di questa classe rappresentano una singola notifica
  - Impostiamo qui
    - l'icona
    - Il/i testo/i da mostrare
    - l'istante della notifica
    - cosa fare quando l'utente seleziona la notifica
- **NotificationManager**
  - È un servizio di sistema che gestisce le notifiche
  - Sarà lui a
    - visualizzare la Notification
    - gestire lo swipe-down per i dettagli
    - iniziare l'azione richiesta quando l'utente seleziona la notifica

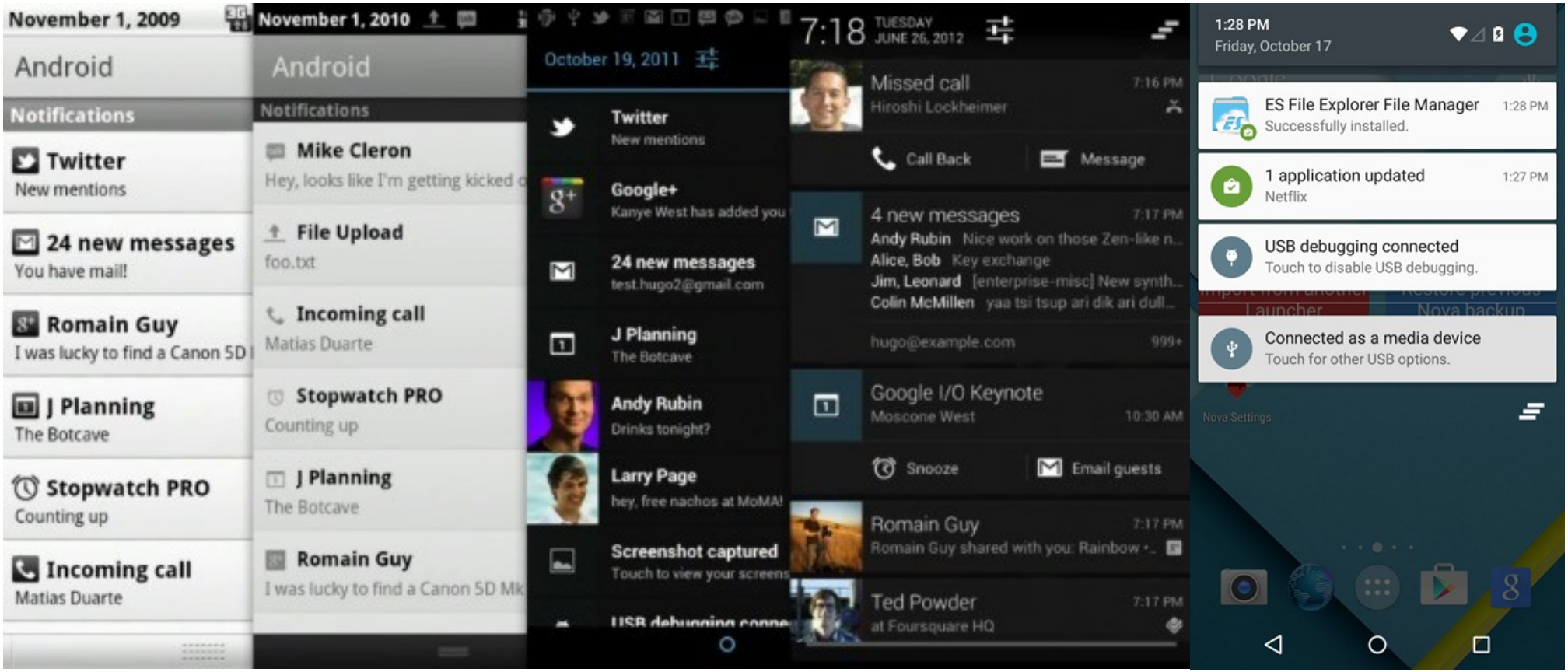
Notification drawer & co.



- Le singole notifiche possono anche fornire layout personalizzati e componenti UI interattivi
- Visualizzate in vari stati
  - Su lock screen
  - In notification drawer
  - Private, compatte, espanse...
  - Su altri dispositivi (Glass, Wear, dashboard, TV)

Evoluzione

- Le notifiche hanno cambiato aspetto *ad ogni versione di Android* – meglio non fare assunzioni



Creare una notifica (pre-Lollipop)

- Per creare l'oggetto:

```
Notification notification =
        new Notification(icon, tickerText, when);
```

- icon – ID di risorsa dell'icona da mostrare
- tickerText – testo che scorrerà brevemente sulla barra di stato nel momento in cui viene emessa la notifica
- when – istante della notifica
  - Spessissimo: System.currentTimeMillis()

Creare una notifica (Lollipop+)

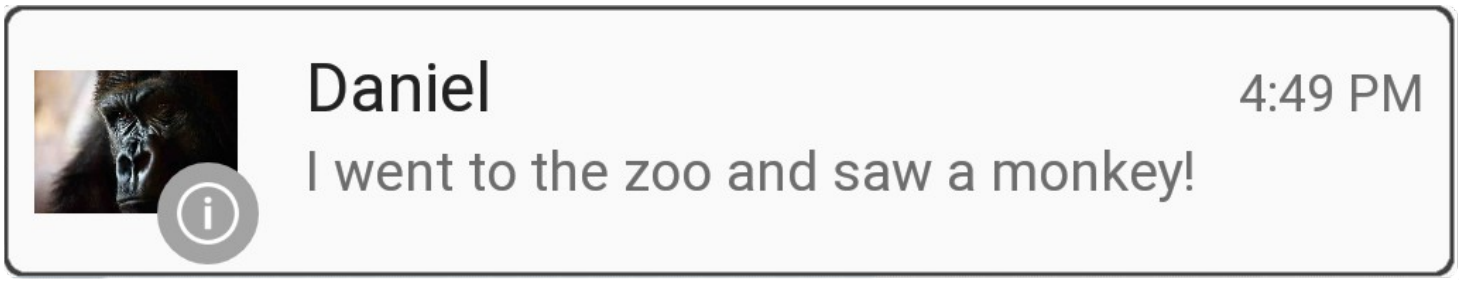
- C'è stato un cambiamento importante
  - Le notifiche ora hanno molti più **metadati** che le collegano al contesto mentale dell'utente
    - Persona (contatto) di riferimento
    - App che l'ha prodotta
    - Classificazione delle informazioni
      - **Sensibili**: da non mostrare sul lock screen
      - **Compatte**: da mostrare nel notification drawer con la notifica collassata
      - **Estese**: da mostrare nel notification drawer con la notifica espansa
  - In più: informazioni sulla propagazione su altri device

Creare una notifica (Lollipop+)

- La classe **NotificationCompat** è una factory per creare notifiche compatibili con i vari sistemi

```
NotificationCompat.Builder builder =
  new NotificationCompat.Builder(this)
  .setSmallIcon(R.drawable.ic_info)
  .setContentTitle("Daniel")
  .setContentText("I went to the zoo and saw a monkey!");
Notification n = builder.build()
```

- Icona (piccola), titolo e testo sono le uniche informazioni *obbligatorie*



Sottomettere una notifica

- Ci serve un riferimento al NotificationManager di sistema:

```
NotificationManager nm =
  getSystemService(Context.NOTIFICATION_SERVICE);
```

  - **Tenere a mente!** getSystemService() si usa anche per accedere ai manager di telefonia, sensoristica, finestre, ecc.

- Poi affidiamo la notifica al manager:

```
nm.notify(id, notification);
```

  - id – un nostro intero che serve a identificare la notifica

Notifiche complesse (pre-Lollipop)

- Una notifica come quella che abbiamo appena creato può anche bastare, ma...

  - Spesso si vogliono dare **più informazioni** quando l'utente “apre” la barra delle notifiche per avere dettagli
  - Potremmo voler **aggiornare** i dati di una notifica già sottomessa
  - Se l'utente seleziona una notifica potremmo voler compiere qualche **azione**
    - Come sempre in Android, se abbiamo l'**intenzione** di fare qualcosa, la esprimiamo tramite un **Intent**

Notifiche complesse (pre-Lollipop)

- Il metodo setLatestEventInfo() consente di aggiornare (e completare) i dati di una notifica

```
Context context = getApplicationContext();
CharSequence titolo = "Titolo (breve)";
CharSequence testo = "Testo (anche lungo)";
Intent i = new Intent(this, MyClass.class);
PendingIntent pi = PendingIntent.getActivity(this, 0, i, 0);

notification.setLatestEventInfo(context, titolo, testo, pi);
```

Intent esplicito

Un **PendingIntent** è un oggetto che custodisce un **Intent** pronto a essere spedito in qualche punto del futuro. L'intent pendente deve essere completamente inizializzato nel momento in cui il PendingIntent viene usato.

Aggiornare una notifica

- Per aggiornare una notifica già esistente:
  - Si chiama setLatestEventInfo() della notifica con i nuovi parametri
  - Si chiama notify() del NotificationManager passando **lo stesso id** della prima notifica
- Esempio:
  - Arriva una nuova email
  - Si aggiorna il conteggio mail “nuove”
  - Si aggiorna l'orario dell'ultimo arrivo

Notifiche complesse (pre-Lollipop)

- Altre possibilità (si noti l'accesso “bruto” ai campi!):
  - Impostare un layout custom della notifica con notification.contentView = *remoteView*
    - RemoteView è una variante di View che incapsula il suo contesto (per poter essere eseguita “fuori” contesto)
    - La incontreremo ancora parlando di Home Screen Widget
  - Impostare pattern di lampeggi, vibrazione, suoni con:
    notification.sound = *suono*;
    notification.vibrate = *pattern di vibrazione*;
    notification.ledARGB = *colore*;
    notification.ledOnMS = *tempo di accensione*;
    notification.ledOffMS = *tempo di spegnimento*;

Notifiche complesse (Lollipop+)

- Una volta ottenuto il builder, è possibile impostare moltissime altre varianti
  - **addPerson()** – la notifica è relativa a uno specifico contatto
  - **setColor()** – imposta il colore primario della vostra app per la notifica
  - **setContent()** – specifica un layout custom per i contenuti
  - **setGroup()**, **setGroupSummary()** – indica che questa notifica è parte di un gruppo di notifiche correlate
  - **SetPriority()** – indica l'importanza dell'evento
  - **SetPublicVersion()** - fornisce una seconda notifica da mostrare “in pubblico”
  - **setLights()**, **setSound()**, **setVibrate()** – colori dei LED, pattern di blink, suono di allarme e vibrazione specifici di questa notifica
  - **SetWhen()**, **setUsesChronometer()** – imposta l'ora a cui si riferisce la notifica (oppure un countdown!)
  - ...

Progress

- È molto comune che una notifica includa una progress bar di qualche tipo
  - Per esempio: download in corso
- Pre-Lollipop, occorre definire un layout custom con una progress bar e aggiornarla “a manina”
- Lollipop+, si può
  - chiamare setProgress() sul builder
  - Aggiornare la notifica con una notify() sul manager

Progress esempio

```
for (i=0;i<=100;i++) {
   builder.setProgress(100, i, false);
   nm.notify(id, builder.build());
   Thread.sleep(1000);
}
```

- Tipicamente, l'aggiornamento però è fatto da un altro thread
  - Servizio in background, AsyncTask, Thread, ecc.
  - Vedremo dopo
- L'esempio sopra **blocca** l'app per 100 secondi!

Notifiche heads-up (Lollipop+)

- Quando un'applicazione è nella modalità **schermo intero**, sovrascrive anche la barra in alto
- Le notifiche non sarebbero visibili affatto in questo caso
  - Rovina un po' l'idea di notifica, no?
- Il sistema le visualizza in forma heads-up, come una finestrella indipendente
- Valutare il compromesso fra utilità e seccatura!