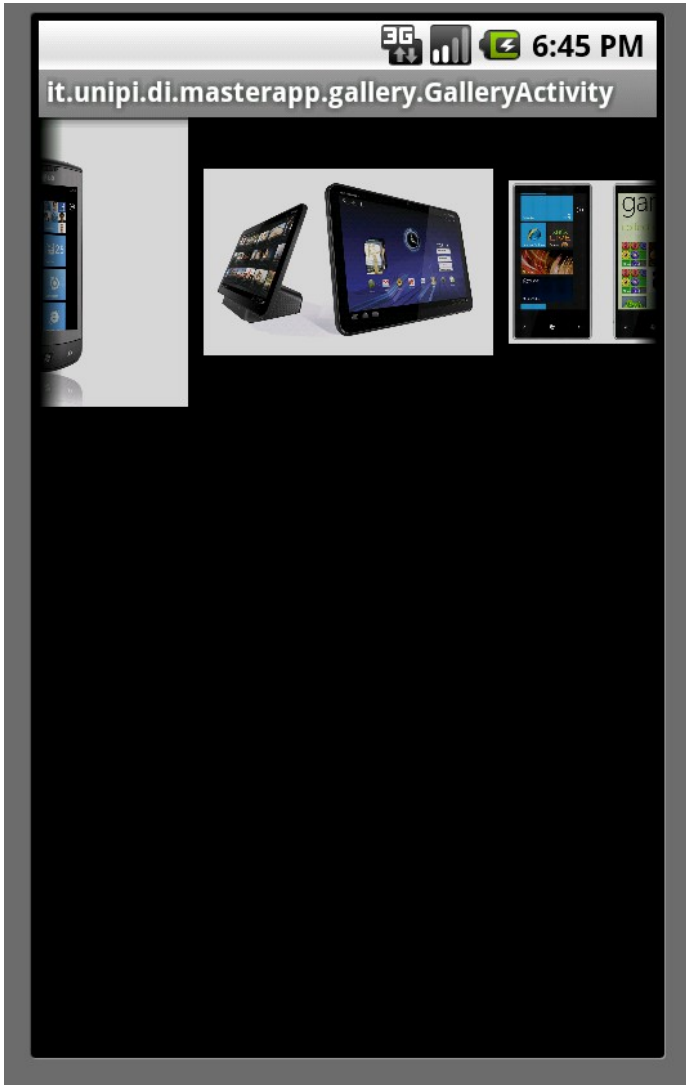
Lezione 9

Programmazione Android

- Ancora sulla UI
  - ListView e data adapter
    - Uso di adapter in altri ViewGroup
  - Drawable
  - WebView
  - Notifiche all'utente
    - Toast
    - Dialog
    - Notification bar
  - Action Bar
  - Stili e temi

Gallery



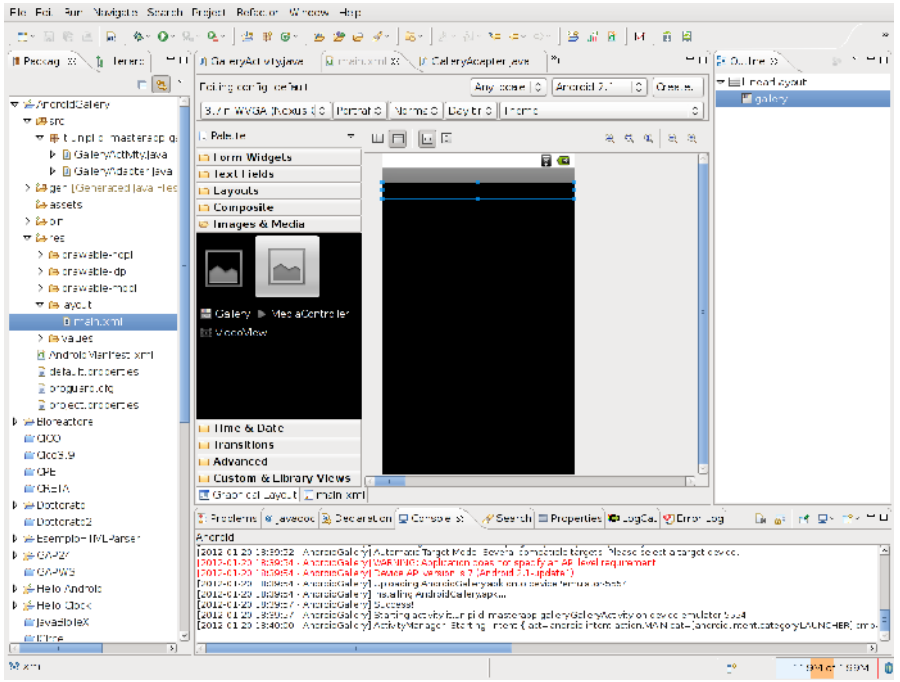
- L'idea generale di usare un **Adapter** per decidere quali **view** visualizzare dentro un **ViewGroup** è usata in altri widget
- **Gallery** mostra una “striscia” orizzontale di view
  - Ciascuna view proviene da un Adapter
- Altri: Spinner, Flipper, …

Gallery – Esempio

- Esempio di **Gallery**
  - Mostriamo una striscia di immagini
  - Il widget gestisce automaticamente lo scorrimento
  - Un doppio tap “seleziona” un elemento
- Esempio di **Adapter** custom
  - Recuperiamo le immagini da rete
  - Creiamo “al volo” la vista per mostrare ogni immagine

Gallery – Esempio
il layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Gallery
        android:id="@+id/gallery"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:gravity="center"
        android:spacing="8dp" />

</LinearLayout>
```



Gallery – Esempio
L'Activity

```
public class GalleryActivity extends Activity {
   private Gallery gallery;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      gallery=(Gallery) findViewById(R.id.gallery);
      gallery.setAdapter(new GalleryAdapter(this));

   }
   @Override
   protected void onResume() {
       super.onResume();
       gallery.setOnItemClickListener(new AdapterView.OnItemClickListener() {
              @Override
              public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
              Toast.makeText(getApplicationContext(), "Immagine "+pos, Toast.LENGTH_SHORT ).show();

              }
       });
   }
}
```

Recuperiamo la Gallery e la colleghiamo al nostro Adapter custom

Impostiamo un semplice OnItemClickListener

(vedremo dopo Toast)

Gallery – Esempio
L'Adapter custom

```
public class GalleryAdapter extends BaseAdapter {

    int galleryItem;
    private Context context;

    public GalleryAdapter(Context context) {
        super();
        this.context = context;
    }

    @Override
    public int getCount() {          // Quanti oggetti abbiamo nella nostra lista
        return 18;
    }

    @Override
    public Object getItem(int position) { // Restituisce l'oggetto alla posizione indicata
        return urlFor(position);
    }

    @Override
    public long getItemId(int position) {// Restituisce l'ID dell'oggetto alla posizione indicata
        return position;
    }
...
```

Salviamo il Context per usarlo dopo (attenzione: questo impedisce di fare garbage collection di **un sacco** di oggetti, finché il nostro Adapter è in memoria. Non raccomandato!)

Brutale!

Gallery – Esempio
L'Adapter custom

```
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView = new ImageView(context);
        imageView.setImageDrawable(loadImageFromURL(urlFor(position)));
        imageView.setLayoutParams(new Gallery.LayoutParams(150, 150));
        imageView.setScaleType(ImageView.ScaleType.CENTER_INSIDE);
        return imageView;
    }

    private String urlFor(int position) {
        return "http://masterapp.di.unipi.it/img/slideshow/a" + (position + 1);
    }

    private Drawable loadImageFromURL(String url) {
        try {
            InputStream is = (InputStream) new URL(url).getContent();
            Drawable d = Drawable.createFromStream(is, "From "+url);
            return d;
        } catch (Exception e) {
            System.out.println(e);
            return null;
        }
    }
}
```

Allochiamo una nuova ImageView per ogni elemento (non consigliabile: vedi dopo)

Privati, di utilità

getView()

public View getView(**int** *position*, View *convertView*, ViewGroup *parent*)

- Deve restituire una View che rappresenta l'oggetto in posizione *position*
- La View verrà inserita come figlia di *parent*
- **Se possibile**, deve modificare *convertView* in modo che essa rappresenti l'oggetto, e restituirla
- **Altrimenti** (meno efficiente), può allocare e restituire una nuova View

Drawable

Drawable

- Abbiamo già visto le due classi principali usate per disegnare su una raster:
  - Canvas: superficie di disegno
  - Paint: “vernice” (= raccolta di parametri) con cui si disegna
- Il Canvas è un contenitore di primitive di disegno
  - Dopo essere passate per la pipeline di rendering 2D, il disegno effettivo va su una Bitmap

SurfaceView, GLSurfaceView

- Fra le view offerte da Android, ne troviamo due destinate esplicitamente al disegno “custom” su un drawable:
  - **SurfaceView**
    - Consente di effettuare il disegno su un Canvas da un thread secondario, anziché dal thread della UI
    - Dispone di un protocollo (callback e locking) per informare il thread secondario di quando disegnare
  - **GLSurfaceView**
    - Consente di disegnare scene 3D in OpenGL ES
    - OpenGL → corso di Fondamenti di Grafica Tridimensionale

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);
```

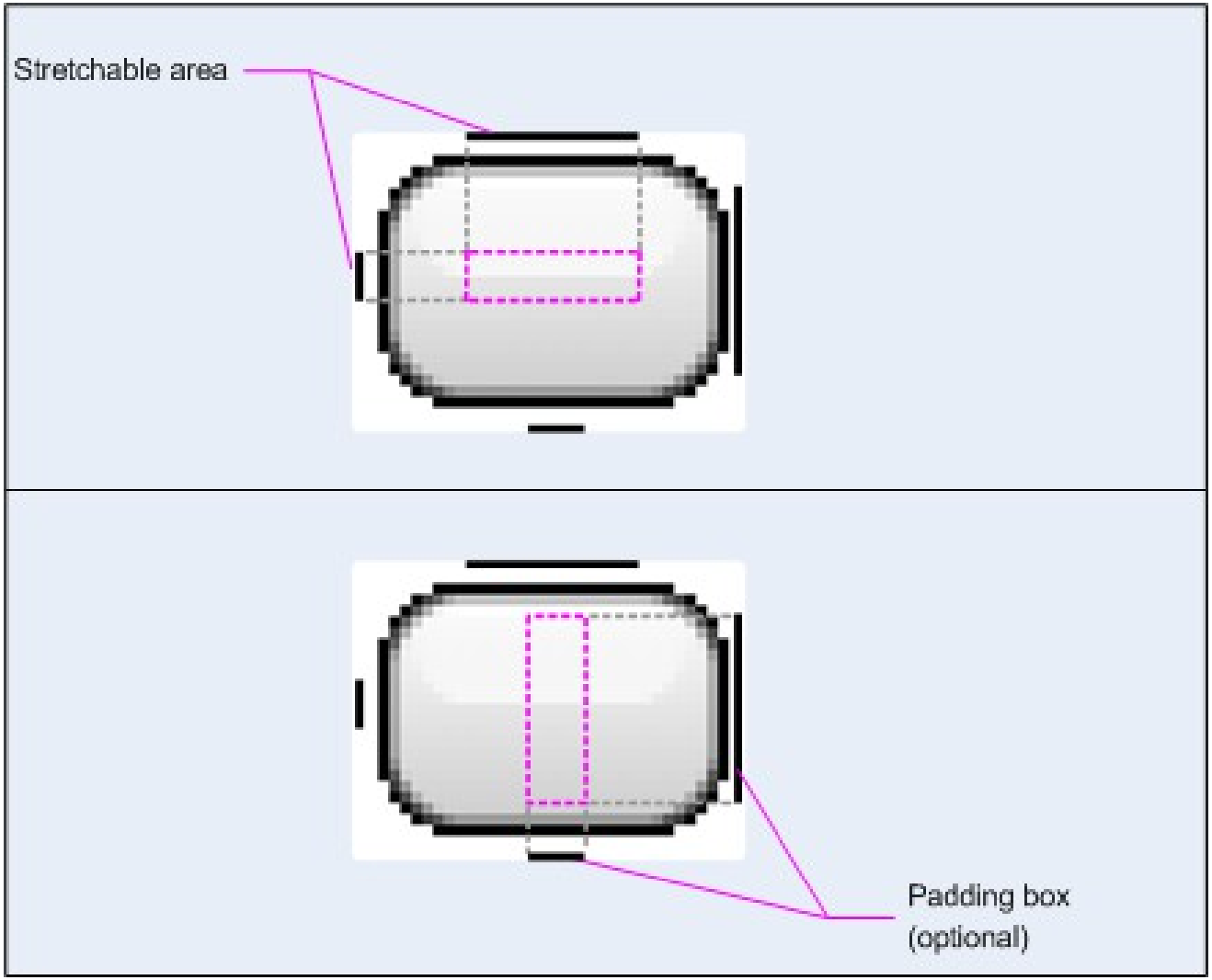
- … o come sfondo di una View

```
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 Drawable(...);

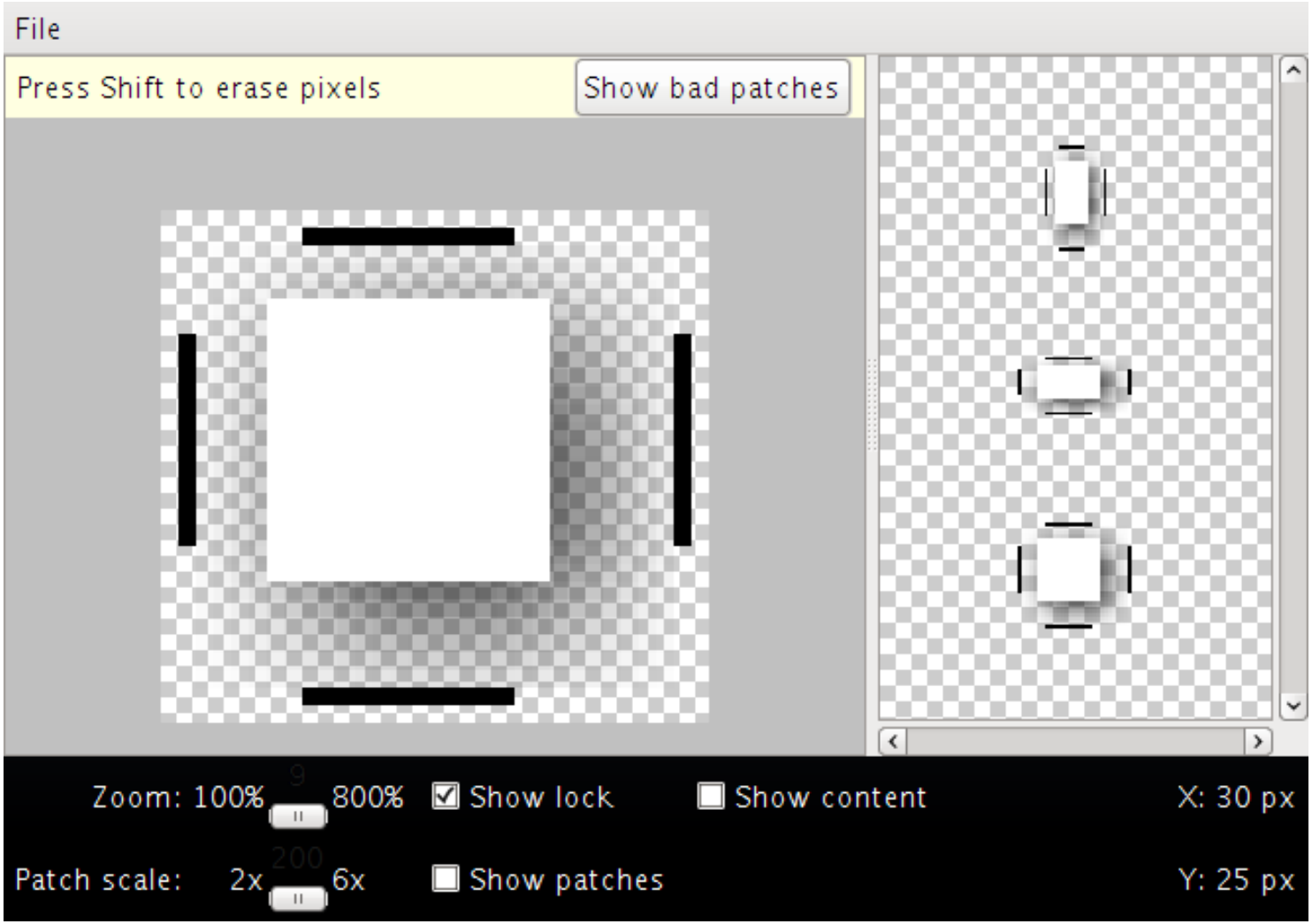
Il formato “nine patch”



- 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



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);
```

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.

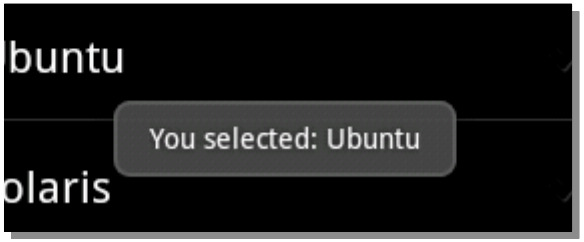
WebView
Javascript bridge

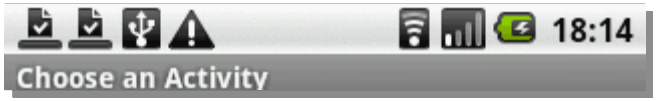
- La WebView esegue nativamente il codice Javascript presente nella pagina
  - Possibile anche passare una URL nella forma javascript:funzione()
- È anche possibile effettuare il **binding** fra oggetti Java (dell'App) e oggetti Javascript (nella pagina)
  - addJavascriptInterface(oggetto, nome)
- L'oggetto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

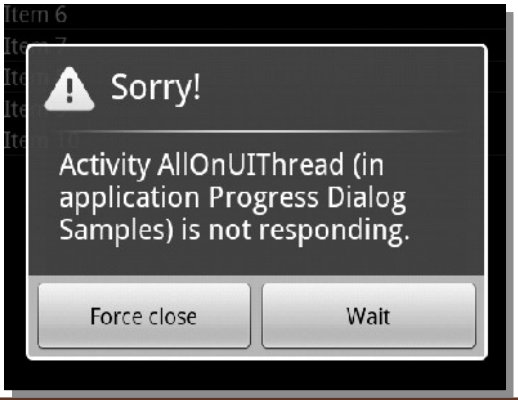
Notifiche all'utente

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 permanenti 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

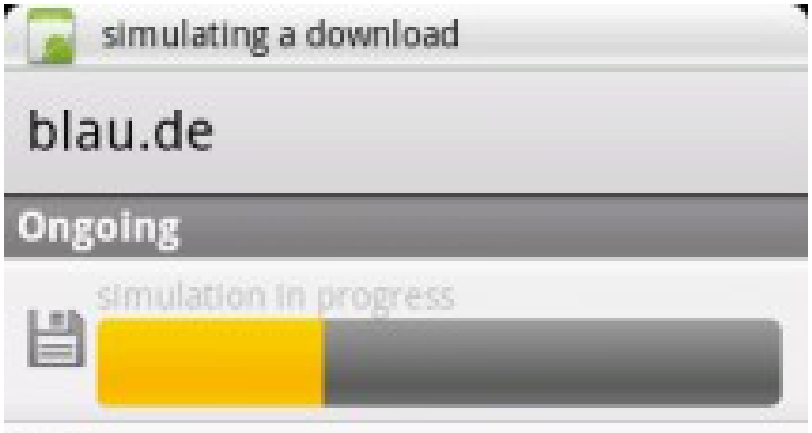
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/toast_root"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="16dp"
        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**
- Questo è esattamente quello che WP7 e iOS hanno difficoltà a fare
  - Solo di recente aggiunte forme limitate di esecuzione in background
  - Solitamente utilizzate da servizi di sistema, ovvero dietro particolari contratti con i “big player”
- 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 Market, 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

Creare una notifica

- 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()

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

- 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

- 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);
```

Intento 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

- 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*;

Dialoghi

- Un **Dialog** è parte dell'interfaccia utente di una Activity, e sempre collegato al suo contesto
- Android mette a disposizione alcuni tipi di dialoghi già pronti:
  - **AlertDialog** – informazioni e richieste di decisione
  - **ProgressDialog** – stato di avanzamento di un task
  - **DatePicker**, **TimePicker** – input di date e orari
- È sempre possibile creare dialoghi custom
  - Come al solito, basta definirsi un layout
  - È utile ereditare lo stile di sistema per i dialoghi!

Interazione tra Dialog e Activity

- Mentre un toast “galleggia” su un'activity e non interferisce con essa, e una notifica su status bar è completamente fuori da ogni activity, un Dialog interagisce con la sua Activity
  - Il Dialog è in effetti implementato come una sotto-activity, in cui gran parte dello schermo è trasparente
  - Però è sempre parte della nostra UI: per esempio, il tasto menu attiva il menu che era attivo per l'activity che ha lanciato il dialogo

Interazione tra Dialog e Activity

- L'activity chiede al sistema di aprire un suo dialogo (passando un int) con **showDialog(id)**
- Se è la prima volta che viene richiesto il particolare dialogo, il sistema chiama **onCreateDialog(id)** dell'activity, che costruisce e restituisce un Dialog
  - Il sistema "conserva" il Dialog restituito per usi futuri
- Prima di aprire il dialog, il sistema chiama **onPrepareDialog(id, dialog)** dell'Activity
  - Qui si possono cambiare i contenuti del dialog
- Come al solito, lo scopo è evitare le **new**!

Creazione di un Dialog

- Il tipo più semplice (e più generale) è l'**AlertDialog**
- Comprende, opzionalmente:
  - Un titolo
  - Un'icona
  - Un testo
  - Da uno a tre pulsanti
    - positive, negative, neutral
  - Una lista di item

Creazione di un Dialog

- La creazione effettiva viene svolta da un **DialogBuilder**

```
protected Dialog onCreateDialog(int id) {
    switch (id) {
    case DIALOG_YES_NO_MESSAGE:
       return new AlertDialog.Builder(this)
          .setIcon(R.drawable.alert_dialog_icon)
          .setTitle(R.string.alert_dialog_two_buttons_title)
          .setPositiveButton(R.string.alert_dialog_ok, new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int whichButton) {

                /* cose da fare se l'utente ha cliccato OK */
             }
          })
          .setNegativeButton(R.string.alert_dialog_cancel, new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int whichButton) {

                /* cose da fare se l'utente ha cliccato Cancel */
             }
          })
          .create();
```

Il Context

Method chaining

Method chaining

Creazione di un Dialog

- I metodi del Builder:
  - **setCancelable()** - se l'utente può chiudere il Dialog con ←
  - **setIcon()**, **setMessage()**, **setTitle()** - contenuti del messaggio
  - **SetPositiveButton()**, **setNegativeButton()**, **setNeutralButton()** - label dei pulsanti
  - **SetAdapter()**, **setCursor()**, **setItems()**, **setSingleChoiceItems()**, **setMultiChoiceItems()** - imposta una lista di scelte
    - Analoghi a quello che abbiamo visto per le ListView
  - **setOnCancelListener()**, **setOnItemSelectedListener()**, **setOnKeyListener()** - listener per i vari eventi
  - **create()** / **show()** - crea / crea e visualizza immediatamente il Dialog

Dialog custom

- Il principio generale è simile, ma creare il layout del Dialog in onCreateDialog() diventa responsabilità del programmatore
  - Non usiamo AlertDialog.Builder
  - Si istanzia direttamente un Dialog e si imposta il suo layout con setContentView()
  - **In alternativa**, si possono aggiungere “pezzi” custom a un AlertDialog con setView()

Dialog custom

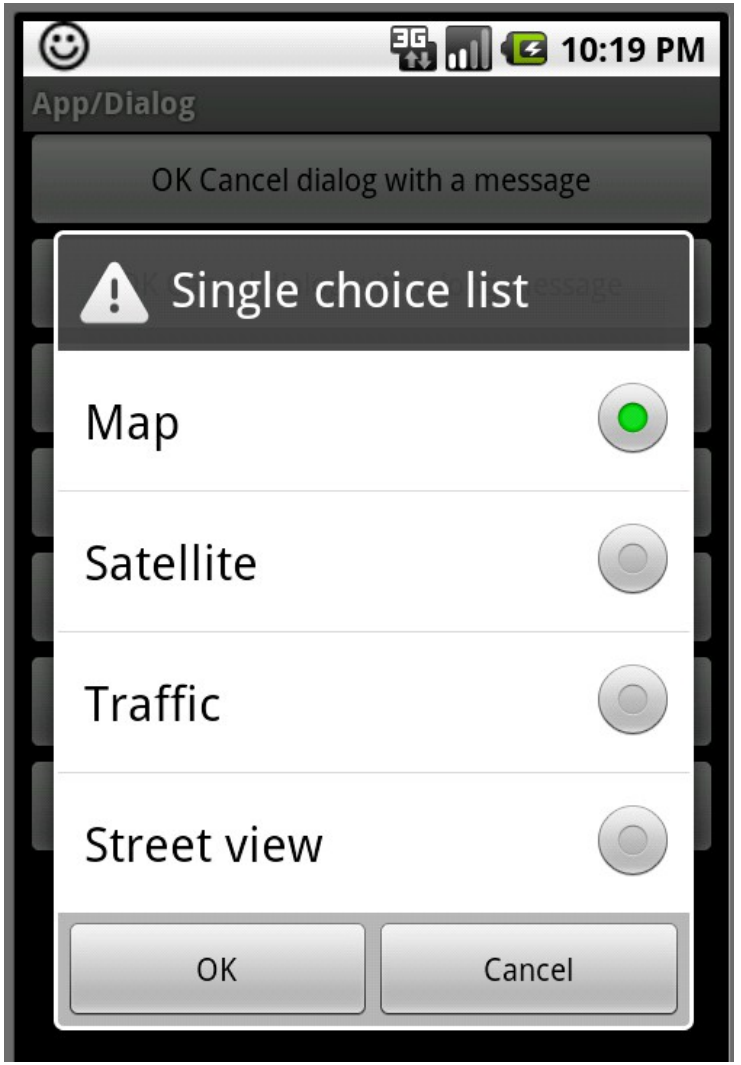
```
case DIALOG_TEXT_ENTRY:
    LayoutInflater factory = LayoutInflater.from(this);
    final View textEntryView = factory.inflate(R.layout.alert_dialog_text_entry, null);
    return new AlertDialog.Builder(AlertDialogSamples.this)
        .setIcon(R.drawable.alert_dialog_icon)
        .setTitle(R.string.alert_dialog_text_entry)
        .setView(textEntryView)
        .setPositiveButton(R.string.alert_dialog_ok, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                /* cose da fare se l'utente ha cliccato OK */
            }
        })
        .setNegativeButton(R.string.alert_dialog_cancel, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                /* cose da fare se l'utente ha cliccato Cancel */
            }
        })
        .create();
```

Dialog custom


```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:id="@+id/username_view"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:text="@string/alert_dialog_username"
        android:gravity="left"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/username_edit"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:scrollHorizontally="true"
        android:autoText="false"
        android:capitalize="none"
        android:gravity="fill_horizontal"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/password_view"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:text="@string/alert_dialog_password"
        android:gravity="left"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/password_edit"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:scrollHorizontally="true"
        android:autoText="false"
        android:capitalize="none"
        android:gravity="fill_horizontal"
        android:password="true"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</LinearLayout>
```

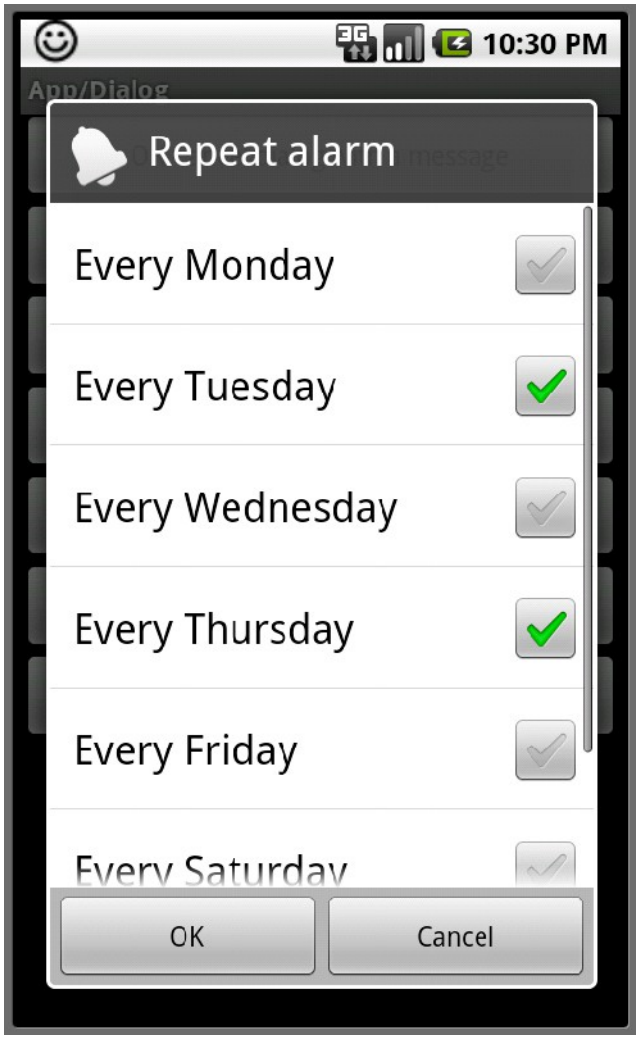
Dialog single-choice

```
case DIALOG_SINGLE_CHOICE:
   return new AlertDialog.Builder(AlertDialogSamples.this)
      .setIcon(R.drawable.alert_dialog_icon)
      .setTitle(R.string.alert_dialog_single_choice)
      .setSingleChoiceItems(R.array.select_dialog_items2, 0,
          new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int whichButton) {
             /* cosa fare se l'utente ha selezionato un item */
          }
      })
      .setPositiveButton(R.string.alert_dialog_ok,
          new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int whichButton) {
             /* cosa fare se l'utente preme OK */
          }
      })
      .setNegativeButton(R.string.alert_dialog_cancel,
          new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int whichButton) {
             /* cosa fare se l'utente preme Cancel */
          }
      })
     .create();
```



Dialog multi-choice

```
case DIALOG_MULTIPLE_CHOICE:
   return new AlertDialog.Builder(AlertDialogSamples.this)
      .setIcon(R.drawable.ic_popup_reminder)
      .setTitle(R.string.alert_dialog_multi_choice)
      .setMultiChoiceItems(R.array.select_dialog_items3,
            new boolean[]{false, true, false, true, false, false, false},
            new DialogInterface.OnMultiChoiceClickListener() {
               public void onClick(DialogInterface dialog,
                     int whichButton, boolean isChecked) {
                        /* cosa fare se l'utente seleziona un item */
               }
            })
      .setPositiveButton(R.string.alert_dialog_ok,
        new DialogInterface.OnClickListener() {
         public void onClick( ... ) { ... }
      })
      .setNegativeButton(R.string.alert_dialog_cancel,
        new DialogInterface.OnClickListener() {
         public void onClick( ... ) { ... }
      })
     .create();
```



Dialog Progress

- Si usa la classe ProgressDialog
- Simile all'AlertDialog, ma in più:
  - SetProgressStyle()
    - istogramma orizzontale o spinner circolare
  - incrementProgressBy(), incrementSecondaryProgressBy()
  - setProgress(), setSecondaryProgress(), setIndeterminate()
- Ha senso in genere solo in presenza di più thread
  - **MAI** svolgere compiti lunghi nel thread dell'UI!

Dialog
DatePicker e TimePicker

- Per fortuna, queste sono facili
  - Perché poco configurabili!

```
@Override
protected Dialog onCreateDialog(int id) {
   switch (id) {
      case TIME_DIALOG_ID:
         return new TimePickerDialog(this, mTimeSetListener, mHour, mMinute, false);

      case DATE_DIALOG_ID:
         return new DatePickerDialog(this, mDateSetListener, mYear, mMonth, mDay);
      }
   return null;
}
```

24 ore o AM/PM

Si può invocare updateTime() o updateDate() nell'onPrepareDialog() per assicurarsi che i dialog siano inizializzati con la data o ora corrente (o desiderata).

Dialog
DatePicker e TimePicker

- I listener vengono chiamati quando l'utente conferma la data/ora
  - void onDateSet(DatePicker view, int anno, int mese, int giorno)
  - void onTimeSet(TimePicker view, int ora, int minuto)
- Nota: il TimePickerDialog non consente di scegliere i secondi!

DatePickerDialog usa il widget DatePicker
TimePickerDialog usa il widget TimePicker

Non è detto che usare un dialog separato sia sempre la cosa migliore... si possono usare i widget direttamente nella activity