**Lezione 1 (29 Settembre)**

**Introduzione**

Gli oggetti della classe View hanno dei metodi “listeners”, cioè metodi “in ascolto” che vengono chiamati quando viene eseguita un’azione su quell’oggetto, come ad esempio un click.

Nella cartella “app” ci sono 3 cartelle, manifesto, java e res:

* Manifesto: contiene le info relative all’app, come ad esempio permessi, activity, icone;
* Java: contiene il codice (e soprattutto la MainActivity), con alcune cartelle di servizio e testing;
* Res (resources): contiene tutte le risore dell’app, come ad esempio i layouts, i disegni, valori (dimensioni, colori, costanti…), menù, mipmap (utilizzata per gestire le varie risoluzioni dei devices).

setContentView setta il contenuto della view, cioè del layout, ciò che mostriamo all’utente.

I file all’interno della cartella res/layout rappresentano i layout della nostra app.

**Lezione 2 (30 Settembre)**

Gradle è un sistema che mantiene informazioni sulle interdipendenze tra i file coinvolti nella compilazione di un’app (librerie di supporto, compatibilità…).

SDK Manager ci permette di installare e gestire varie versioni di Android, con le varie API.

AVD Manager è il gestore degli emulatori, devices virtuali.

Nel file build.gradle (module: app) andiamo a specificare la versione per la quale vogliamo compilare l’app (compileSdkVersion), ma anche la versione dove parte la compatibilità dell’app (minSdkVersion).

**Layouts**

Il layout è il modo in cui andiamo ad organizzare tutti gli elementi dell’interfaccia grafica per andare a creare l’interfaccia utente. I layout possono essere creati in due modi:

* Modo statico, cioè con un file XML;
* Modo programmatico, cioè attraverso codice Java, che ci permette di rendere il layout più “dinamico”.

I modi non sono esclusivi. Il vantaggio dell’XML è che è facile da specificare, e separa in modo netto la definizione dell’UI dal codice dell’applicazione, ma lo svantaggio è che gli elementi sono statici e possono essere modificati solo tramite codice java. Il vantaggio del modo programmatico è ce è dinamico, quindi si può facilmente adattabile, lo svantaggio è che si va a mescolare con la logica dell’app.

Gli elementi (widget) base sono i TextView e i Button.

Per organizzare i widget all’interno dei layout, possiamo utilizzare i ViewGroup (widget non di base), che organizzano gli elementi per il layout.

Il Linear Layout ci permettono di organizzare i widget in modo orizzontale o verticale.

Il Relative Layout ci permette di organizzare i widget relativamente ad altri widget.

Il Grid Layout organizza i widget come una griglia.

Il Frame è semplicemente un contenitore che ingloba vari widget.

Ogni elemento (semplice o ViewGroup) supporta degli attributi che possono essere obbligatori, opzionali, specifici solo per alcuni elementi.

Per la creazione di un ID di un widget, si deve usare l’attributo android:id=“@+id/…”. Questo attributo è presente in tutti i widget.

Un esempio di attributi obbligatori sono layout\_width e layout\_height.

**Lezione 3 (6 Ottobre)**

Ci sono varie caratteristiche e misure per i device Android, come ad esempio la risoluzione, le dimensioni (fisiche), la densità, i pixel… I px sono i pixel reali, mentre li dp è la densità indipendente dei pixel, una dimensione calcolata su una densità di 160dpi (dot per inch, dove dot è un pixel, quindi significa che in un pollice ci sono 160 pixel), quindi un “dp” ha le dimensioni di un “px” a 160dpi.

LinearLayout ordina i widget in modo orizzontale o verticale (android:orientation=”vertical” o “horizontal”). In ogni figlio si può definire un “peso” (layout\_weight), dove si può assegnare più spazio ad un altro figlio assegnandogli un peso maggiore. Con orientation=”vertical”, i widget vengono inseriti uno sotto l’altro, con “horizontal” vengono inseriti in successione, sulla stessa riga. Nel LinearLayout non viene lasciato spazio vuoto, quindi se lo si vuole si devono aggiungere degli elementi fittizi.

Il RelativeLayout ordina i widget in modo “relativo”, cioè i widget vengono posizionati relativamente ad altri widget o al padre.

Il ConstraintLayout è simile al RelativeLayout, permette di specificare la posizione attraverso “vincoli”, ed è utile per evitare una gerarchia di layout innestati troppo profonda che necessita tempo per essere disegnata.

Il GridView visualizza un insieme di elementi, ha automaticamente lo scroll se ci sono più elementi che non possono essere visualizzati nella schermata.

Il Grid Layout organizza i widget in una griglia (non ha lo scroll automatico).

Il ListView è una sorta di GridView, ma c’è solo una colonna e ha lo scroll automatico. Ha bisogno di un adapter per inserire gli elementi nel List View.

**Lezione 6 (14 Ottobre)**

I messaggi di Log sono dei messaggi che vengono catturati da AndroidStudio e vengono mostrati all’interno del log dell’IDE, e possiamo associare ad ogni messaggio di Log un TAG (per individuare velocemente il messaggio di errore). Equivale ad utilizzare le printf.

**List View**

Gli oggetti List View sono praticamente delle view all’interno della quale c’è una lista, e si può fare lo scroll su di esso. Innanzitutto la List View avrà una specifica area sul telefono, quindi una propria altezza e larghezza, che sarà poi divisa in base a quanti elementi la List View deve contenere. Gli elementi che sono presenti nella List View saranno salvati all’interno di un array perché il numero di elementi che deve mostrare all’interno della view saranno (per la maggior parte delle volte) maggiore dello spazio a disposizione. Per utilizzare una List View c’è bisogno di un Adapter, cioè un oggetto che, lavorando assieme al widget, ci mostrerà correttamente gli elementi successivi a quelli già visualizzati (esempio: stiamo visualizzando gli elementi da 1 a 5, utilizzando l’Adapter con la ListView, i prossimi elementi saranno da 6 a 10).

Per utilizzare la List View si deve innanzitutto inserire la ListView all’interno dell’Activity Main, mentre si dovrà creare un altro documento XML per ogni singolo oggetto che sarà visibile all’interno della ListView. Per quanto riguarda il codice Java, si deve definire innanzitutto un array con tutti gli elementii, poi si definisce un Adapter (o un ArrayAdapter, cioè un array di adapter) alla quale verrà passato il “context” (che contiene le info relative all’app), il layout per il singolo list element (esempio: R.layout.list\_element), il layout per il listView (esempio: R.id.textViewList), e l’array che vogliamo mostrare all’interno del ListView. Infine, deve essere associato al widget ListView l’adapter (esempio: listView.setAdapter(arrayAdapter)).

La ListView ci permette anche di associare un Listener per ogni singolo elemento che appartiene al widget. Per associare un Listener ad ogni elemento della ListView, si deve utilizzare il metodo setOnItemClickListener, al quale dovrà essere passata una funzione onItemClick, che ha come input anche l’oggetto “int position”, che è la posizione dell’item cliccato all’interno della lista ListView.

Per personalizzare ogni singolo item, innanzitutto si può usare il file di layout del singolo elemento, e poi si dovrà personalizzare l’Adapter creando un CustomAdapter. Nel caso in cui vogliamo creare un click multiplo su ogni singolo elemento, dovremo creare del Listeners ad hoc, che però ci farà perdere la posizione dell’item all’interno dell’array. Per risolvere questo problema, si può sfruttare un “trucchetto”, settando ad ogni elemento della ListView un tag.

**Lezione 7 (20 Ottobre)**

**Ciclo di vita di un’app**

Un’app è fatta da una o più activity. Ogni Activity ha un ciclo di vita, cioè passa da uno stato all’altro durante tutta la sua funzione. Possiamo riassumere questi stati usando dei metodi delle app:

* Stato in cui l’app non esiste (non è ancora creata);
* Metodo onCreate();
* Metodo onStart();
* Metodo onResume();
* Stato in cui l’app è in esecuzione (è funzionante e si può interagire con l’activity dell’app);
* Metodo onPause();
* Metodo onStop();
* Metodo onDestroy();
* Stato in cui l’app è stata chiusa.

Quando il metodo onCreate() viene lanciato, l’app passa nello stato Created, da dove viene eseguita onStart(), passando nello stato Started. Adesso, dallo stato Started verrà chiamato il metodo onResume(), passando nello stato Resumed, lo stato in cui l’utente interagisce e usa l’app. Da questo stato si passa allo stato Paused, usando il metodo onPause(). Lo stato Paused ha due possibilità: usare il metodo onResume(), per tornare nello stato Resumedm, oppure usare il metodo onStop() per andare nello stato Stopped. Nello stato Stopped ci sono altre due possibilità: usare il metodo onDestroy(), che distrugge l’app e quindi lo stato diventerà Destroyed, oppure il metodo onRestart(), che riporterà l’app nello stato Started.

Quando un’app si trova negli stati Paused o Stopped, il sistema operativo Android potrebbe anche distruggere l’app per utilizzare la memoria utilizzata da quest’ultima, e quando essa verrà riaperta, verrà richiamato il metodo onCreate(), cioè l’app verrà “ricostruita” daccapo.

Quando viene premuto il pulsante home della device, vengono chiamati i metodi onPause() e onStop(); quando si ritorna all’attività, vengono chiamati i metodi onRestart(), onStart() e onResume(); quando l’utente ruota il dispositivo, l’app viene prima ELIMINATA, cioè vengono eseguiti i metodi onPause(), onStop() e onDestroy(), quindi verranno eliminati tutti dati che erano in memoria (tutti i dati e le variabili salvate in memoria), e poi viene ricreata usando i metodi onCreate(), onStart() e onResume(). Per salvare i dati che si trovano nell’attuale stato dell’app, si deve usare il metodo onSaveInstanceState(Bundle savedInstanceState). L’oggetto di tipo Bundle permette di salvare i tipi di dati primitivi e i suoi array (int, float, array di stringhe). Per salvare dati non primitivi, si deve ricorrere all’uso di file, ad esempio. Quando verrà rieseguito il metodo onCreate(), si passerà all’app l’oggetto Bundle e poi si utilizzeranno i metodi get per recuperare tutto dall’oggetto Bundle.

**Ciclo di vita e cambi di configurazione**

Quando si ruota lo schermo, in android c’è un cambio di configurazione. Altri cambi di configurazione possono essere il cambio di layout, la disponibilità dello schermo, la grandezza dello schermo, schermi rotondi (orologio)… Più in generale, quando c’è un cambio di configurazione Android distrugge e ricrea l’activity per adattarsi più facilmente alla nuova configurazione. Per salvare lo stato è onSaveInstanceState() o i ViewModel, che sono oggetti persistenti. Possiamo anche gestire i cambi di configurazione impedendo ad Android di distruggere l’activity e ricrearla, ma non ci sarà nessun cambio di configurazione (non è consigliato, la documentazione di Android dice così).

**Lezione 8 (21 Ottobre)**

**Backstack**

Un’app è fatta da più activity che ha uno specifico compito. Ad esempio, un’app di posta elettronica può avere un’activity per la scrittura del messaggio, un’altra per inviare (o ricevere) un messaggio, un’altra per gestire la lista dei messaggi, e un’altra per vedere il contenuto dei messaggi.

Un’attività può lanciare un’altra attività, anche un’attività di altre app.

Con la classe Intent si può lanciare e si possono passare dati alla nuova activity che si vuole lanciare. Il task è un insieme di attività con cui l’utente interagisce. Più activity possono coesistere, e vengono organizzate in un backstack (uno stack di activity). Un task (che è un insieme di activity) solitamente parte dall’home screen, cliccando l’icona di un’app, e così l’applicazione vene portata in foreground (in primo piano). Se vengono lanciate nuove app (o l’utente torna nella home del device), l’attività corrente viene inserita nel backstack, e l’utente può tornarci usando il pulsante “back” del device. Se un’attività può essere lanciata da più di un’attività, si possono istanziare più di un’activity della stessa activity (diverse tra di loro).

**Intent**

L’Intent è una classe Java che permette di rappresentare un’operazione da svolgere (l’activity è una parte di Intent) e permette di lanciare una nuova activity, oppure inviare un messaggio broadcast, o comunicare con un servizio in background. Le parti principale di un oggetto Intent sono:

* le azioni, le azioni da svolgere;
* i dati su cui operare, espressi come URI (<schema>:<parte specifica>).

Altri parti di un Intent sono:

* le categorie, che aggiunge altre informazioni sull’azione da eseguire;
* il tipo, che specifica in modo esplicito il tipo (MIME) dei dati;
* la componente, che specifica l’attività da eseguire;
* extras, un bundle di informazioni aggiuntive.

La risoluzione dell’intent può essere esplicita o implicita.

La risoluzione esplicita avviene nel caso in cui si specifica in modo esplicito l’attività (Component) che vogliamo lanciare.

La risoluzione implicita avviene nel caso in cui la component non è specificata, e quindi Android sceglie un’attività appropriata (in base a action, type, URI, category). Le attività dichiarano le action che possono soddisfare nel manifesto. Se c’è la componente, l’azione diventa irrilevante.

**Lezione 10 (28 Ottobre)**

**Permessi**

I permessi sono un meccanismo di protezione che Android offre per proteggere e limitare specifiche risorse o dati di alcune app. Per accedere a specifici dati, c’è bisogno di un consenso esplicito dell’utente.

Per specificare i permessi di cui un’app ha bisogno, si devono aggiungere queste richieste di permesso all’interno del manifesto dell’app.

I permessi sono divisi in due classi:

* i permessi normali, permessi che vengono concessi senza chiedere nulla all’utente;
* i permessi speciali, che devono essere approvati dall’utente (quando si installa l’app, con API < 23, runtime con API altrimenti).

Si può cambiare la decisione su un permesso dato ad un’app andando nelle impostazioni del device e rimuovendo/inserendo il permesso che si vuole dare alla specifica app. Nel codice, si deve sempre controllare che ci sia il permesso per fare quel determinato metodo.

Alcuni permessi normali sono: bluetooth, internet, nfc…

Alcuni permessi pericolosi sono: telefono, microfono, GPS, fotocamera…

**Threads**

I thread permettono di fare una commutazione parallela all’interno di un processo. Ogni thread ha un proprio program counter e il proprio stack, mentre condivide con gli altri thread del processo l’heap e la memoria statica.

In Java ci sono degli oggetti che possono implementare l’interfaccia Runnable, e per usare i thread questa classe deve creare un metodo void run(). I metodi che useremo saranno:

* void start();
* void sleep(long time);
* void wait(), che aspetta che un altro oggetto chiami notify() sull’oggetto che usa questo metodo;
* void notify().

Quindi innanzitutto si crea un oggetto Thread, e poi si chiama il metodo start() del thread, che chiamerà il metodo run().

Android non permette ai thread in background di interagire con l’interfaccia utente, ma è solo il Main Thread che può “comunciare” con l’utente.

Per facilitare l’interazione tra i background thread e i main tread, c’è la classe Async task che permette una più facile comunicazione tra i thread. La classe AsyncTask prende in input tre parametri:

* Params: tipo di dati per il lavoro che deve svolgere il background thread;
* Progress: tipo di dati usato per lo stato di avanzamento;
* Result: tipo di dati per il risultato del task.

Per quanto riguarda i metodi della classe AsyncTask, ci sono 5 metodi che utilizziamo:

* void onPreExecute(): eseguito nel main thread prima di doInBackground();
* Result doInBackground(Params… params): fa tutto il lavoro in background. Prende in input una lista di parametri, e restituisce un oggetto di tipo result. Può chiamare il metodo void publishProgress(Progress… values) che tornerà la percentuale del completamento del thread:
* void onProgressUpdate(Progress… values): usato nel main thread in risposta a publishProgress;
* void onPostExectute(Result result): eseguito nel main thread dopo doInBackground() con il risultato di doInBackground() come parametro.

**Lezione 11**

**Frammenti**

I frammenti rappresentano una porzione dell’interfaccia utente. Possiamo immaginare l’interfaccia utente suddivisa in varie porzioni, e ognuna di questa porzione corrisponde ad un framment. Un’activity può “ospitare vari frammenti, che possono essere” inseriti e rimossi durante l’esecuzione. Un frammento è una sorta di sub-activity, ha un proprio ciclo di vita ed è strettamente legato a quello dell’activity, ad esempio se l’activity ospitante è in pausa saranno in pausa anche tutti i frammenti contenuti in essa. La porzione del frammento può essere definita runtime (dinamicamente). La filosofia di programmazione è per interfacce dinamiche, per adattarsi a schermi grandi e piccoli.

Il ciclo di vita dei frammenti, come detto, è molto legato alle activity, infatti ci sono molti metodi simili a quelli del ciclo di vita delle activity, e anche il flusso è abbastanza simile.

* onAttach(): il frammento viene “attaccato” ad un’activity;
* onCreate(): il frammento viene inizializzato come in un’activity, ma non definiamo il layout (ma viene fatto in onCreateLayout(), facendo un inflate);
* onCreateView(): viene creata la view del frammento attraverso il metodo inflate a partire dal file XML del layout, e non sarà più utilizzata nuovamente durante tutto il ciclo di vita del frammento;
* onPause(): il primo metodo chiamanto quando il frammento viene eliminato. In questa fase si dovrebbero andare a salvare tutti i dati che vogliamo rendere permanenti.

Per creare un frammento dobbiamo innanzitutto creare un file di layout XML per il nostro frammento, poi creare una classe per il frammento che estende la classe “Fragments”.

Per quanto riguarda il file di layout dell’activity, dobbiamo dedicare uno spazio ai frammenti che saranno contenuti all’interno dell’activity con i tag <fragment>. Per farlo in modo dinamico, servono le classi FragmentManager, FragmentTransaction ed ExampleFragment.

I frammenti vengono inseriti nel backstack in modo un po’ diverso per quanto riguarda le activity, infatti i frammenti non vengono inseriti nel backstack, ma possiamo inserirli in modo “manuale”, chiamando il metodo addToBackStack.

Per comunicare con l’activity ospitante, il frammento deve definire un’interfaccia che dovrà essere implementata nell’activity.

**Lezione 14 (11 Novembre)**

**Networking**

Per comunicare in rete, abbiamo bisogno dei socket, e in particolare la classe Java.net, la classe httpURLConnection e formati di dati JSON e XML. La classe IntentAddress permette di gestire gli indirizzi IP; in particolare, il metodogetByName(“…”) restituisce l’indirizzo IP. La classe Socket crea il canale di comunicazione con il server; per creare la connessione al socket utilizziamo il costruttore new Socket(serverAddr, port), per leggere e scrivere utilizzeremo i metodi getInputStream(socket), getOutputStream(socket).

**Lezione 16 (18 Novembre)**

**Data storage**

Ogni app può gestire i propri dati (primitivi o composti) per renderli permanenti attraverso l’utilizzo di files. Esistono vari tipi di files:

* App-specific storage: ogni app ha uno spazio privato per i file, nella memoria interna o anche nella memoria esterna, in directory dedicate;
* Shared storage: file che possono essere condivisi con altre app;
* Preferences: dati chiave-valore privati per ogni singola app (files privati);
* Database: database pri vati SQL.

Per accedere a specifici files di un’app, possiamo utilizzare i metodi getFilesDir() e getCacheDir(). Queste directory sono specifiche dell’app, nel senso che altre app non possono accedere alle directory che sono “esclusive” per la singola app. Le directory specifiche dell’app sono eliminate quando l’app viene disinstallata.

Per quanto riguarda i files condivisi, e quindi accessibili a tutte le app, possiamo chiedere l’accesso a queste directory condivise, ma serve il permesso per accedere alla memoria “esterna”.

* La classe SharedPreferences permette di salvare e recuperare dati all’interno dei files Preferences usando coppie di chiave-valore. Non c’è da confondersi sul fatto che la classe si chiama “shared”, perché questo attributo non si riferisce al fatto che questi files sono condivisibili con altre app ma al fatto che i files Preferences sono condivisi tra tutte le activity della singola app.

Due metodi della classe Activity:

* + getSharedPreferences(“filename”): quando si vogliono creare più file di preferenze per una sola app;
  + getDefaultSharedPreferences(): quando c’è un solo file di preferenza per la singola app;

Per quanto riguarda la lettura dei dati, si utilizzeranno i metodi getInt, getBoolean, o altri get di tipi di base, con la chiave del valore che vogliamo ottenere. Per quanto riguarda la scrittura, serve un “editor”, un oggetto della classe Editor, e su di esso deve essere richiamato il metodo putInt, putBoolean ecc. con la coppia chiave valore.

* Per ogni app il sistema operativo prevede una directory privata, dove solo l’app può accedere a quest’ultima, e se l’app viene disinstallata anche la sua directory verrà eliminata. Per scrivere all’interno dei files servono praticamente tutti i metodi di “default”, quelli studiati anche nei corsi precedenti. Ci sono anche dei files temporanei, ovvero files che si trovano in una directory cache. Attenzione: se al SO servisse spazio, i files salvati nella directory cache potrebbero essere eliminati.
* Ci sono poi i files salvati sulla memoria esterna, tipicamente la SD card. Di solito sulla SD si scrivono files pubblici, cioè files visibili per tutte le app. Per scrivere sulla SD card c’è bisogno del premesso di lettura e scrittura. Nel caso in cui si voglia scrivere sulla SD card, è bene controllare che sia presente nel device.
* Infine ci sono i files per il salvataggio di databases SQL. Per creare un database SQL si deve creare una sottoclasse di SQLiteOpenHelper, sovrascrivendo il metodo onCreate(), e poi si crea un nuovo oggetto Helper ricavando ida quest’ultimo il database. Sull’oggetto creato si utilizzeranno poi alcuni metodi per fare query, aggiornamenti, rimozioni dalle tabelle del database creato.

**Lezione 18 (25 Novembre)**

**Grafica**

Un’immagine (qualsiasi estensione, Android è molto flessibile sotto il punto di vista di estensioni immagini) può essere disegnata in un oggetto View (grafica semplice) o un oggetto Canvas (grafica complessa) o una classe Drawable (per disegnare un’immagine).

L’oggetto Drawable può essere inserito direttamente nel file XML oppure in modo programmatico con Java.

Android, inoltre, permette di definire delle animazioni da applicare alle immagini, come la traslazione, rotazione, scaling della dimensione, trasparenza eccetera. Per gestire queste animazioni si utilizza la classe Antimation che legge le animazioni da fare in un file XML specifico, e poi le applica alle ImageView.

Android ha molti widget, ma per particolari esigenze possiamo anche creare dei widget personalizzati che permettono un maggiore controllo sulla grafica ma saranno più complessi, perché saremo noi programmatori a gestirli.

**Albero delle View**

L’albero delle View è ciò che noi definiamo nell’XML, cioè una serie di View innestate l’una nell’altra partendo dalla View radice, che è la radice dell’albero, e il modo in cui le altre View sono innestate nella View radice, saranno i figli della View radice. Ovviamente, ogni file XML avrà un proprio albero delle View, perché dipende dal modo in cui viene sviluppato il layout.

Per misurare ogni singola View quanto spazio occupa, c’è un meccanismo di layout da “rispettare”, che ha 3 fasi: misurazione, posizionamento, disegno.

Il grosso del lavoro avviene durante la prima fase, perché in quella fase si analizza tutto l’albero delle view e si analizzano tutte le dimensioni di ogni singolo widget.

Nel caso in cui creiamo un widget, il grosso del lavoro sarà fatto, invece, durante la terza fase, quella del disegno.

Analizzando in particolare ogni fase:

* Durante la prima fase, viene chiesto al widget figlio quanto vorrebbe essere grande, ma poi le size saranno ridimensionate rispetto alla reale dimensione disponibile. L’approccio è top-down, cioè ogni view chiede ai suoi figli quando vorrebbe essere grande e il costo di questa analisi è molto dispendiosa a livello di calcolo e tempo. Ogni volta che c’è un cambiamento nel layout, avviene questa analisi. C’è una classe specifica per questa misurazione, chiamata MeasureSpec, e le dimensioni vengono calcolate con i pixel reali.
* Durante la seconda fase, avviene di nuovo una visita top-down dell’albero delle view, ed è in questa fase che avviene l’assegnazione delle reali dimensioni di ogni view.
* Durante la terza fase, c’è di nuovo una visita completa dell’albero delle view, e in questa fase c’è la creazione del layout.

Appena avviene un cambiamento in un layout, viene chiamato il metodo “invalidate” che fa ridisegnare completamente il layout, che sarà dispendioso dal punto di vista computazionale.

**Lezione 20 (2 Dicembre)**

**Multitouch**

MotionEvent rappresenta un movimento registrato da una periferica e posso gestire questi tocchi/multi tocchi, ma focalizziamo la nostra attenzione sul Multitouch. Per gestire multitouch il device deve supportarlo.

Per gestire un multitouch c’è il concetto di “Pointer”, che indica dove è avvenuto il singolo elemento. Un multitouch può gestire più di un pointer, quindi se ad esempio due dita cliccheranno lo schermo, ci saranno due pointer all’interno dell’evento multitouch. Ovviamente, quando le dita non toccheranno più lo schermo il pointer verrà eliminato. I pointer vengono aggiunti all’interno di un array, ma l’ID del pointer darà differente dalla posizione in cui sarà memorizzato nell’array, quindi gli indici possono essere diversi.

Possono esserci vari ACTION\_CODES per quanto riguarda il MotionEvent, come ad esempio se il dito che tocca lo schermo è il primo a cliccarlo, oppure è il secondo, o il dito viene mosso, o non lo tocca più…

Quando c’è un evento, Android notifica il widget sul quale è avvenuto questo evento, perciò se si vuole associare un’azione a quel widget si deve creare un Listener per il widget, e poi gli si deve associare il metodo onTouchListener.

**Gesture Detector**

GestureDetector permette di riconoscere dei gesti fatti sul display, come ad esempio uno o un doppio click, lo scorrimento eccetera.

**Lezione 21 (9 Dicembre)**

**Media player**

L’AudioManager controlla le sorgenti audio e l’output (cioè il volume), mentre il MediaPlayer gestisce i files che voglio riprodurre o utilizzare. Le sorgenti dati che posso usare sono risorse locali, URI interni o URL. Anche se semplice da utilizzare, la classe MediaPlayer è molto complessa. Al termine dell’utilizzo del MediaPlayer, bisogna rilasciare la risorsa.

Poiché c’è un solo canale di output, se più applicazioni vogliono riprodurre un contenuto, ciò potrebbe essere un problema; per risolvere questo problema si può utilizzare l’auto focus che permette di dare priorità all’app, ma se arriva un’app con una priorità più elevata, l’app attuale deve smettere di suonare, o ridurre il volume.

**Sensori**

Molti devices hanno molti sensori al loro interno, come ad esempio bussole, accelerometri, termometri… Tutti questi sensori ci forniscono dati “grezzi”, che derivano dal fatto che queste misurazioni avvengono nel mondo reale, e l’attendibilità dei dati dipende dalla qualità dei sensori. SensorManager ci dice quali sono i sensori disponibili sul device, e per ogni sensore può fornirci il range massimo, l’accuratezza di esso e altro. Inoltre, permette di leggere i dati del sensore, e si possono usare Listeners su quest’ultimo. Per tutte le informazioni che hanno a che fare lo spazio, il sistema di riferimento resta costante, cioè cambieranno solo i valori letti sull’asse delle x, y, z, ma non il modo di leggere i dati.

**Lezione 22**

**Notifiche**

Le notifiche sono informazioni all’utente al di fuori dell’interfaccia grafica dell’app. Esistono 3 tipi di notifiche: toast, dialog box, notifiche.

* Toast: brevi messaggi temporanei che appaiono sullo schermo. Si può decidere la durata della visualizzazione sullo schermo e non si può interagire con essi;
* Dialog: messaggio pop-up che richiede l’interazione con l’utente. Di solito ha due scelte (tipicamente “sì” o “no”), ad ognuna delle quali è associata un’azione;
* Notifiche: le notifiche che noi tutti conosciamo, che si trovano nella barra dello stato e nel cassetto delle notifiche.

Le notifiche possono essere segnalate in vari modi: nella barra di stato come icone, come cerchietto sulle icone, nel cassetto delle notifiche, dove è anche possibile vedere testo aggiuntivo. Le notifiche sono visibili finché l’utente non decide di eliminarle o visualizzarle. Da Android 5 è anche possibile vedere una notifica a comparsa, cioè notifiche che sono visibili per poco tempo mentre si sta utilizzando il device, ma tutto dipende se si è in full screen, se non si è in particolari modalità del device…

Da Android 8 si dà anche priorità alle varie notifiche, assegnando ogni notifica ad uno specifico “canale”, dando preferenze ad alcune applicazioni più importanti. A partire dall’API 26 le icone delle notifiche presenti all’interno della barra di stato deve essere un’immagine bianca con lo sfondo trasparente.

**Alarms**

L’Alarm è una modalità che Android mette a disposizione per eseguire Intent in base a determinati eventi, in questo caso in base al tempo. Per cui, un’applicazione che usa un Alarm riesce ad eseguire una porzione di codice anche quando l’applicazione non è in funzione e quando il device è in sleep. Gli Alarm rimangono attivi finché non viene cancellata oppure il device viene spento. Per creare un alarm si usa la classe AlarmManager.

**Content Providers, Broadcast, Services**

Il sistema Android ha 4 componenti: applicazioni, content providers, broadcast, services. Abbiamo studiato solo le applicazioni, ma anche queste altre 3 componenti sono importanti, e in particolare ci sono cose che possono essere implementate solo con queste ultime tre componenti.

* I broadcast sono dei messaggi che Android genera e vengono inviati a tutte le app Android. In particolare, ogni app che vuole ricevere messaggi broadcast deve usare un oggetto BroadcastManager per “registrarsi” e ricevere messaggi;
* Il Content Provider sono dei contenitori di contenuti, dati. L’esempio più classico è quello della rubrica, perché la rubrica fornisce dei dati ed è pensata come contenitore per fornire dati anche ad altre applicazioni del sistema. Quindi i Content Providers sono fornitori di contenuti con cui altre componenti possono interfacciarsi, ovviamente con un certo livello di sicurezza e permessi;
* I services sono pensati per fornire servizi ad applicazioni che richiedono molto tempo per essere eseguiti e per questo l’esecuzione dell’applicazione viene inserita in background, proprio perché richiede molto tempo.

**Toast**

Il toast è un modo per dare delle informazioni all’utente. Il toast mostra delle notifiche pop-up temporanee, e al metodo statico Toast.makeText si passano il context dell’applicazione, il messaggio che vogliamo mostrare sullo schermo e la durata della visualizzazione dei Toast che può avere due durate: Toast.LENGTH\_LONG oppure Toast.LENGTH\_SHORT.