Domande Teoria MP

Sommario

Layout

In un progetto Android come sono organizzati i file per lo sviluppo di un’app? Elencare le principali cartelle/file e descrivere brevemente il loro contenuto.

L’organizzazione dei file in un progetto Android è caratterizzata da una struttura ad albero in cui la prima cartella (la radice) è denominata "app" e presenta al suo interno le seguenti sub-directories, il cui ordine è importante:

- **manifests,** che contiene il file AndroidManifest.xml che racchiude le informazioni essenziali dell’app;
- **java,** che contiene il codice sorgente dell’applicazione, organizzato in vari packages ed utilizzato per costruire l’app;
- **res,** che contiene le risorse dell’app, ci sono al suo interno altre directory, come "layout" che fornisce la struttura statica dell’app (in xml), "drawable" all’interno del quale verranno messi immagini e file multimediali, "mipmap" che contiene le varie icone dell’app a seconda della risoluzione del dispositivo, "values" contiene al suo interno file xml che forniscono metadati.

A cosa serve il file Manifest.xml?

Tutte le app Android hanno un file AndroidManifest.xml nella root del sorgente, è un file di risorse che contiene tutti i dettagli necessari dell'applicazione, ci consente di definire: Packages, API, Librerie, elementi base dell'app, come activity e servizi, dettagli sulle autorizzazioni, set di classi necessarie prima del lancio. Ci sono ulteriori informazioni che vengono riportate come:

- Nome dell'applicazione: titolo dell'app (android:label= "nomeApp");
- Icona dell'applicazione: riferimento all'icona visualizzata nella schermata principale di Android per l'app;
- Numero versione: è un singolo numero utilizzato per mostrare una versione dell'app (android:versionCode= "1").

Il layout di un app può essere definito sia staticamente, tramite un file XML, che programmaticamente tramite istruzioni nel programma. Si discuta dei vantaggi e svantaggi e si faccia un esempio di un caso in cui è possibile usare solo uno dei due e non l'altro, motivando la risposta.

Creare un layout tramite file XML consente di specificare facilmente le caratteristiche del layout e porta ad un maggiore disaccoppiamento tra UI e logica di applicazione, ma impone una certa staticità. D'altra parte, creare un layout in modo programmatico rende le cose molto semplici da adattare, ma impone una gestione del tutto all'interno del codice dell'applicazione, che può risultare più difficile e comporta meno leggibilità.

In generale, i due metodi non sono mutualmente esclusivi, ma ci sono alcune situazioni in cui è possibile usarne solo uno, ad esempio per layout completamente statici che possono essere scritti usando soltanto un file XML. È invece necessario usare layout creati soltanto programmaticamente in contesti dinamici in cui bisogna creare interfacce sulla base di un numero variabile di elementi o quando si vuole creare componenti personalizzate sulla base di qualche scelta dell'utente.

Ogni widget del layout prevede degli attributi che determinano parte della visualizzazione dello stesso widget all'interno dello schermo. Non tutti gli attributi possono essere applicati a tutti i widget; tuttavia ci sono degli attributi che si applicano a tutti i widget. Fra questi quali ritieni i più significativi o comunque i più utilizzati? Motivare la risposta.

Ogni elemento (View o ViewGroup) supporta degli attributi che specificano l'aspetto grafico, dove visualizzare l'elemento e fornire ulteriori informazioni. Ci sono degli attributi che sono comuni a tutti gli elementi, ad esempio:

- "**android:id**": permette di assegnare un identificativo univoco agli elementi della UI. In questo modo, in una fase programmatica, si potrà ottenere il riferimento a quell'elemento, per poter modificare in modo dinamico lo stato dell'oggetto.

- "**android:layout_height**"/"**android:layout_width**": specificano altezza/larghezza di base della View, attributi obbligatori per qualsiasi View all'interno del layout.
- "**android:background**": consente di definire il colore di sfondo di un widget (utile per verificare se è stato inserito nel layout).
- "**android:margin**": utilizzato per creare spazio attorno agli elementi, al di fuori di qualsiasi bordo definito; può essere specificato per ogni lato (Top, Bottom, Left, Right).
- "**android:padding**": utilizzato per generare spazio tra il contenuto di un widget ed il suo bordo; anche qui si può specificare per ciascun lato.
- "**android:gravity**": specifica come un oggetto deve posizionare il suo contenuto dentro un componente.

Il posizionamento degli elementi di un layout avviene (anche) specificando delle misure. Quali sono le unità che si possono utilizzare? Si fornisca una breve descrizione di ognuna.

Esistono svariati modi per indicare la grandezza di un elemento:

- dp, ovvero density-independent pixels; si tratta di un'unità astratta basata sulla densità fisica dello schermo;
- sp, ovvero scale-independent pixels; si tratta di un'unità astratta basata sulle preferenze dell'utente in merito alla grandezza del font;
- pt, ovvero points; corrispondono ad 1/72 di pollice in base alle dimensioni fisiche dello schermo;
- px, ovvero real pixels, basati sul numero reale di pixel presenti sullo schermo;
- mm, ovvero millimetri, basati sulle dimensioni fisiche dello schermo;
- in, ovvero inches o pollici, basati sulle dimensioni fisiche dello schermo.

A cosa serve la misurazione degli elementi grafici in "dpi" (density independent pixels). Si discuta del perché è necessaria e di come una cattiva gestione possa influire negativamente sull'aspetto grafico di un'app.

Per preservare la dimensione visibile dell'interfaccia utente su schermi con densità diverse, è necessario progettare l'interfaccia utente utilizzando pixel indipendenti dalla densità (dp) come unità di misura. Android traduce questo valore nel numero appropriato di pixel reali per ogni densità dello schermo.

ES: Si considerino due schermi della stessa dimensione che hanno numero diversi di pixel. Se dovessi definire una vista larga "100px", apparirà molto più grande su uno dei due dispositivi, a seconda dei pixel a disposizione. Quindi si deve invece usare "100dp" per assicurarsi che appaia la stessa dimensione su entrambi gli schermi.

Si consideri un’app il cui layout prevede 4 pulsanti ai 4 angoli del display. Tale layout può essere implementato in vari modi. Descrivine uno (a parole, senza codice XML, spiegando solo come dovrebbe essere fatto il codice XML) mettendone in evidenza vantaggi e svantaggi.

Utilizziamo un RelativeLayout specificando per width e height il valore match_parent per prendere tutto il display a disposizione, mentre per i Button si userà wrap_content sia per width che height. Per il primo Button (in alto a sx), imposteremo "alignParentLeft = true" per posizionarlo in alto a sinistra. Per il secondo Button (in alto a dx), imposteremo in modo analogo "alignParentRight = true". Per il terzo ed il quarto Button, che andranno rispettivamente in basso a sx e in basso a dx, useremo le stesse impostazioni dei primi due, ma aggiungeremo anche "alignParentBottom = true".

Si consideri un dispositivo A con un display di 5x7cm e densità 1.5 e un dispositivo B con un display di 8x12cm e densità 2 (si assuma che la densità 1 indichi 100 pixel per cm). Si consideri un’immagine di 300x300pixel. Quanto spazio, misurato in cm, tale immagine occuperà nel display A e quanto spazio nel display B? Si consideri un’immagine di 254x254dpi. Quanto spazio, misurato in cm, tale immagine occuperà nel display A e quanto spazio nel display B? (Si ricordi che un pollice, inch, equivale 2.54cm).

Il dispositivo A ha una densità in pixel per cm di 1.5 * 100 = 150 pixel per cm.

Il dispositivo B ha una densità in pixel per cm di 2 * 100 = 200 pixel per cm.

Lo spazio occupato dall'immagine in cm è calcolato come il rapporto tra il numero di pixel e la densità pixel/cm:

- Per il dispositivo A si ha 300/150 = 2cm, dunque l'immagine occuperà 2*2 cm su A.
- Per il dispositivo B si ha 300/200 = 1.5cm, dunque l'immagine occuperà 1.5*1.5 cm su B.

Data l'immagine con risoluzione 254*254 dpi, il primo passaggio è convertirne le dimensioni in pixel per cm:

- dpi = pixel/inch => pixel/cm = dpi/2.54, da cui ricaviamo che l'immagine è di 100 pixel/cm
- Per il dispositivo A si ha 254/150 = 1.69cm, dunque l'immagine occuperà 1.69*1.69 cm su A.
- Per il dispositivo B si ha 254/200 = 1.27cm, dunque l'immagine occuperà 1.27*1.27 cm su B.

Per ognuno dei seguenti casi scrivere il corrispondente spezzone di codice XML.

- Il testo “Ciao”, non editable, largo quanto il contenitore e alto quanto basta a visualizzare il testo con il testo allineato al centro.
    - <TextView
    - android:layout_width=”match_parent"

    - android:layout_height="wrap_content”
    - android:gravity=”center"
    - android:text=”Ciao" />
- Il testo “Android” editable, largo e alto quanto basta a visualizzarlo, che si posiziona sulla destra del suo contenitore.
    - <EditText
    - android:layout_width=”wrap_content"
    - android:layout_height="wrap_content”
    - android:layout_gravity=”right"
    - android:text=”Android" />
- Un pulsante, largo 300dp e alto 60dp, con identificativo “pulsante3”, con testo “Premi” allineato al centro del pulsante, ed avente come listener la funzione “pulsantePremuto()”
    - <Button
    - android:id=”@+id/pulsante3"
    - android:layout_width=”300dp"
    - android:layout_height=”60dp”
    - android:onClick=”pulsantePremuto"
    - android:gravity=”center"
    - android:text=”Premi" />
- Una progessBar alta 10dp e larga quanto il suo contenitore con identificativo “pb1”, inizialmente invisibile.
    - <ProgressBar
    - android:id=”@+id/pb1"
    - android:layout_height=”10dp”
    - android:layout_width=”match_parent"
    - android:visibility=”invisible />
- Un gridView largo 400dp e alto 500dp in cui gli elementi sono organizzati su 4 colonne
    - <GridView
    - android:layout_width=”400dp"
    - android:layout_height=”500dp”
    - android:numColumns=”4” />

ListView

Che cosa è un ListView? Che cosa è un Adapter? In che modo ListView e Adapter interagiscono?

Un ListView è un Widget specifico per le liste, raggruppa diversi elementi e li visualizza in un elenco scorrevole verticale. Gli elementi vengono automaticamente inseriti nell'elenco utilizzando un Adapter che estrae il contenuto da un'origine come un array o un DB. Un Adapter è un oggetto, che può essere agganciato a qualsiasi AdapterView (ovvero ListView), che gestisce la visualizzazione degli elementi in un AdapterView: fornisce accesso a tali elementi ed è responsabile della creazione di una View per ogni elemento nel data set. L'Adapter viene assegnato a ListView tramite il metodo setAdapter sull'oggetto ListView: listView.setAdapter(arrayAdapter).

Descrivere schematicamente tutto ciò che serve per far comparire sullo schermo un ListView (semplice) con una lista di nomi (definiti in un array). Si specifichi per ogni passo l'istruzione, il widget o il file XML, necessari per il ListView.

Si definisce nel file xml (activity_main) un widget ListView, specificando l'id (mylistview) e i vari parametri:

- <ListView
  - android:id="@+id/mylistview"
  - android:layout_width="wrap_content"
  - android:layout_width="wrap_content"
- </ListView>

Si crea un ulteriore file xml per rappresentare il singolo elemento della lista, coi relativi parametri:

- <TextView
  - android:id="@+id/textViewList"
  - android:layout_width="wrap_content"
  - android:layout_height="wrap_content"
  - android:text="" android:padding="10dp" android:textSize="22dp"
- />

All'interno del file Java MainActivity si definisce un Array di stringhe contente nomi di persone:

- String [] array = {"Pasquale","Maria","Michele", …}

Si definisce un ArrayAdapter, dove context rappresenta il contesto dell'applicazione, R.layout.list_element rappresenta il file di layout dove viene definito il singolo elemento nella lista, R.id.textViewList che rappresenta l'identificativo del TextView nel file di layout e poi c'è array che rappresenta le informazioni da visualizzare nel ListView:

- ArrayAdapter arrayAdapter = new ArrayAdapter(context, R.layout.list_element, R.id.textViewList, array);

Viene individuato il widget ListView:

- listView = (ListView)findViewById(R.id.mylistview);

Infine si associa l'adapter al widget:

- listView.setAdapter(arrayAdapter);

Che cosa è un Adapter customizzato per un ListView? Si fornisca una spiegazione possibilmente con frammenti di codice.

Un Adapter customizzato nasce dall'esigenza di creare una visualizzazione personalizzata di un ListView. Un Adapter customizzato per un ListView è un Adapter che consente di adattare, ad un ListView, un elemento che contiene dei sottoelementi. L'elemento da adattare dev'essere descritto in un file XML, in cui si inserisce il layout dell'elemento (compreso, pertanto, dei sottoelementi). Per creare un adapter customizzato, occorre creare una classe che estende ArrayAdapter, al cui interno va aggiunto il costruttore (avente come parametri il context, la resource id dell'elemento da adattare, ed una lista di elementi da adattare) ed il metodo getView(): quest'ultimo viene invocato ogniqualvolta è necessario mostrare un elemento del ListView (ad esempio, al lancio dell'applicazione o a seguito di uno scroll), ed è utilizzato anche per eseguire l'inflate del file di layout dell'elemento nel ListView. Per adattare un ListView, nell'activity in cui tale lista è presente occorre seguire i prossimi step:

Creare l'adapter customizzato ed aggiungerlo al ListView:

- CustomAdapter adapter = new CustomAdapter(getApplicationContext(), R.layout.?, new ArrayList);
- ListView listView = (ListView) findViewById(R.id.listView);
- listView.setAdapter(adapter);

Aggiungere i vari oggetti all'adapter:

- for (…) { object = …; adapter.add(object); }

Aggiungere, se richiesto, un listener sul ListView:

- listView.setOnItemClickListener(new OnItemClickListener() { @Override public void onItemClick (AdapterView parent, View view, int position, long id) { … } });

Un listview prevede un OnItemClickListener che gestisce i click sugli elementi della lista chiamando il metodo onItemClick al quale viene passato un riferimento dell'elemento selezionato. Se usiamo un listview customizzato, in cui ogni elemento della lista è composto da vari sottoelementi (es. una foto, un nome, un numero), il riferimento passato al metodo onItemClick non distingue quale dei sottoelementi è stato selezionato. Come si può fare per reagire in maniera diversa in funzione di quale dei sottoelementi è stato selezionato con il click?

Se usiamo un ListView customizzato, per distinguere quale sottoelemento è stato selezionato è opportuno definire dapprima un Adapter customizzato. Supponendo che esso sia stato già creato ed "agganciato" al ListView, allora tale problema deriva dal fatto che è stato passato un OnItemClickListener sull'intero elemento della lista, "creato" dal CustomAdapter. Per risolvere, è opportuno agganciare un onclick listener su ogni sottoelemento del ListView; inoltre, per far sì che si possa risalire alla posizione dell'elemento dell'adapter cliccato, è possibile utilizzare il metodo setTag() su ogni sottoelemento passando la posizione dell'elemento adattato (questo va fatto nel metodo getView() del custom adapter): questa potrà essere ricavata in un altro punto con il metodo getTag() sullo stesso sottoelemento.

Ciclo di Vita

Presenta un disegno che mostra: Lancio activity -> onCreate() ... Running ... Attività terminata; chiede di completare il ciclo di vita. Si descriva un'operazione che è solitamente effettuata in onStart(), con la corrispondente operazione effettuata in onStop(), e un'operazione solitamente effettuata in onResume(), con la corrispondente operazione effettuata in on Pause().

Il ciclo di vita di un'attività viene scandito attraverso le seguenti operazioni:

- onCreate(), in cui si ha l'inizializzazione dell'attività; prevede operazioni che vanno eseguite una sola volta nel ciclo di vita dell'applicazione, come associare l'activity ad un ViewModel.
- onStart(), in cui l'attività è visibile, ma non interagibile; prevede attività che preparano l'attività ad andare in foreground, come l'inizializzazione dell'interfaccia utente.
- onResume(), in cui l'attività è visibile ed interagibile; prevede solitamente l'inizializzazione delle componenti necessarie all'applicazione.
- onPause(), in cui l'attività è parzialmente visibile; prevede il rilascio delle componenti inizializzate con onResume().
- onStop(), in cui l'attività è nascosta; prevede il rilascio di tutte le risorse non necessarie quando l'app non è più in foreground e può essere usata per delle operazioni dispendiose di shutdown, come il salvataggio dei dati.
- onRestart(), quando l'attività viene richiamata dopo essere stata messa in pausa.
- onDestroy(), in cui vengono rilasciate le ultime risorse occupate dall'attività.

Quando l'utente preme il pulsante 'home' vengono chiamati i metodi onPause() e onStop(), mentre quando si torna all'attività vengono richiamati onRestart(), onStart() e onResume(). Quando invece l'utente ruota lo schermo, l'attività viene prima distrutta (onPause(), onStop() e infine onDestroy()) e poi ricreata (onCreate(), onStart() e onResume()); per non perderne lo stato bisogna usare il metodo onSaveInstanceState, in modo da recuperarlo successivamente tramite onCreate.

II ciclo di vita delle activity, riportato schematicamente a sinistra, prevede l'esecuzione in successione di 3 metodi (onCreate, onStart, onResume) per far partire l'esecuzione di un'app. Pechè? Non sarebbe stato meglio avere un solo metodo, come indicato nella figura a destra, nel quale eseguire tutto ciò che viene fatto nei 3 metodi onCreate, onStart, onResume? Analogamente per distruggere un app è prevista l'esecuzione di 3 metodi in successione (onPause, onStop, onDestroy). Non sarebbe stato più semplice avere un solo metodo come indicato nella figura a destra? Motivare la risposta.

La presenza di più metodi per scandire le molteplici fasi del ciclo di vita di un'activity consente una gestione più fine delle risorse impiegate ed una loro liberazione 'a specchio' una volta che l'app smette di essere in foreground. Il metodo onCreate() si occupa di quelle operazioni da eseguire una sola volta nel ciclo di vita, onStart() di quelle che preparano l'attività ad andare in foreground e diventare interattiva ed infine onResume() di quelle inerenti alle altre componenti da usare. Con onPause() liberiamo proprio quest'ultime, in quanto l'app, ora parzialmente visibile, potrebbe non necessitare più di componenti quali, ad esempio, la fotocamera. Successivamente, onStop() libera le risorse non necessarie quando l'app non è più in foreground ed infine, se l'utente non riapre l'app, viene chiamato onDestroy() per rimuovere le ultime risorse. Compattare tutto in due soli metodi, onEsecuzione() e onFine(), non potrebbe garantire nulla di tutto ciò e priverebbe inoltre l'utente della possibilità di ritornare sull'app tramite onRestart() dopo che è stato chiamato onStop(). Un'altro svantaggio sarebbe quello di incorrere in arresti anomali quando l'utente ruota lo schermo o quando, ad esempio, riceve una telefonata e passa ad un app diversa.

Note extra da quattro domande brevi:

Il periodo di visibilità va dalla chiamata onStart() alla chiamata onStop(), mentre il periodo di foreground va da onResume() a onPause().

Le seguenti chiamate vengono effettuate in modo automatico: onStart(), onResume(), onStop(), onDestroy(), onRestart(); richiedono dell'intervento dell'utente o del sistema: onCreate() e onPause().

Sono indicati come stati transienti (in cui l'activity resta pochissimo tempo) Created, Started e Paused, mentre sono indicati come duraturi (l'activity vi può rimanere per molto tempo) Resumed e Stopped.

È possibile che onCreate() venga chiamato più volte di onStart() se l'utente ruota molte volte lo schermo, portando ad una sequenza di cambi di configurazione che distruggono e ricreano l'app: in questo modo, il metodo onStart() potrebbe essere chiamato al più un numero di volte uguale a quello di onCreate().

Backstack

Che cosa è il backstack? Supponendo che l’activity A sia l’unica in esecuzione (l’unica presente nel backstack), che la stessa app lancia una nuova activity B che a sua volta lancia l’activity C, che lancia l’activity D, quale è il backstack a questo punto? Cosa succede se dall’activity D si preme il pulsante di "back"? Cosa si deve fare se si vuole fare in modo che dall’activity D, si torni direttamente ad A quando si preme il pulsante di back?

Un task è una raccolta di attività con cui gli utenti interagiscono durante l'esecuzione di un determinato lavoro, disposte in una pila denominata backstack. Esso ha una struttura LIFO e memorizza le attività nell’ordine della loro apertura. Le attività presenti non vengono mai riorganizzate e la navigazione viene eseguita con il pulsante back. La situazione nel backstack è la seguente: D, C, B , A. Se si preme back, ovvero si esegue un pop dallo stack dell’activity, l’attività D viene distrutta e C viene messa in foreground. Se si vuole far in modo che dall’activity D si torni direttamente ad A quando si preme il pulsante di back, occorre lanciare le activity B e C in modo tale che non vengano memorizzate nel backstack: questo dev’essere effettuato programmaticamente, aggiungendo un FLAG_ACTIVITY_NO_HISTORY all’intent attraverso il metodo intent.setFlags(int flag).

Si spieghi il meccanismo del backstack. In relazione a tale meccanismo che differenza c’è fra una activity e un frammento?

Il backstack è la pila, gestita in modo LIFO, in cui vengono inserite tutte le attività che sono state lanciate ed inizia, generalmente, dall'Home Screen. Un fragment è una porzione dell'interfaccia che viene ospitata da un'activity ed ha un ciclo di vita legato a quello dell'activity. Le activity sono processi pesanti che richiedono molta memoria, mentre i frammenti sono processi leggeri che usano soltanto la memoria necessaria. [Gestione delle activity spiegata sopra]. I fragments vanno invece gestiti manualmente: i cambiamenti vengono inseriti con il metodo addToBackStack() e la comunicazione con le activity viene effettuata con interfacce di callback.

Cambi di configurazione

Un dispositivo Android può funzionare sia in modalità portrait (verticale) che landscape (orizzontale). Quando un dispositivo Android viene ruotato si passa dall’una all’altra modalità. Per gestire in maniera appropriata tali passaggi, di cosa si deve preoccupare il programmatore?

Quando un dispositivo Android viene ruotato, passando da una modalità all’altra, normalmente un’applicazione effettua un "cambio di configurazione", pertanto necessita di esser prima distrutta per poi esser ricreata: questo fa sì che si perda lo stato dell’activity che l’utente utilizzava, se il programmatore non gestisce il cambiamento di stato.

Per gestire tale cambiamento, di recente è stato implementato un attributo di <application>, android:configChanges="orientation | keyboardHidden | screenSize": se lo si specifica, è possibile lasciare al sistema operativo la gestione, così che non venga perso alcun dato; tale attributo fa sì che venga invocato il listener onConfigurationChanged(), in cui è possibile eseguire qualsiasi computazione si desideri.

Un approccio programmatico che consente al programmatore la gestione manuale dei dati consiste nel salvare alcuni dati nell'oggetto Bundle savedInstanceState: quando c'è il cambiamento viene invocato il metodo onSaveInstanceState(Bundle savedInstanceState), al cui interno vengono tipicamente aggiunte informazioni a tale oggetto (usando i setter savedInstanceState.putStringArrayList(String tag, ArrayList<String> list), savedInstanceState.putInt(String tag, int value), …) e, prima della chiusura del metodo, viene invocato super.onSaveInstanceState(savedInstanceState) - metodo di Activity - per rendere effettivo il salvataggio. Questo oggetto viene passato anche al metodo onCreate(), per cui qui è possibile ricavare le informazioni sullo stato memorizzate attraverso i getter savedInstanceState.getStringArrayList(String tag), savedInstanceState.getInt(String tag), …

Si consideri la seguente situazione: un'app viene lanciata e l'utente interagisce con l'app; ad un certo punto l'utente ruota il dispositivo e continua ad interagire con l'app; infine l'utente chiude l'app tramite un apposito pulsante. In quali stati è passata l'app e quali metodi sono stati chiamati dal momento in cui l'app viene lanciata al momento in cui termina?

1. L'app viene lanciata e l'utente interagisce con l'app:
   - onCreate() -> Created -> onStart() -> Started -> onResume() -> Resumed;
2. L'utente ruota il dispositivo e continua ad interagire con l'app:
   - onPause() -> Paused -> onStop() -> Stopped -> onDestroy() -> Destroyed -> [punto 1]
3. L'utente chiude l'app tramite un apposito pulsante:
   - onPause() -> Paused -> onStop() -> Stopped -> onDestroy() -> Destroyed.

Intent

Cosa è un Intent? Si faccia un esempio in cui viene coinvolto un Intent utilizzando opportuni frammenti di codice.

Un intent è un semplice oggetto che viene utilizzato per la comunicazione tra componenti Android, come activity, content provider, receiver e broadcast transmission. Gli intent vengono anche utilizzati per trasferire dati tra attività. Gli intent vengono generalmente utilizzati per iniziare una nuova activity utilizzando startActivity(). Sono caratterizzati da diverse componenti: Action, Data, Category, Type, Component, Extras e Flags.

Spiegare la differenza tra Intent esplicito ed implicito, con esempio di codice.

Intent esplicito: Un intento esplicito è quello che si usa per avviare un componente specifico dell'app, come una particolare activity o servizio nella app. Per creare un intento esplicito, bisogna definire il nome del componente nell'oggetto Intent: tutte le altre proprietà dell'intent sono facoltative.

Un intento implicito specifica un'azione che può invocare qualsiasi app sul dispositivo in grado di eseguire l'azione. L'uso di un intent implicito è utile quando l'app non è in grado di eseguire l'azione, ma altre app probabilmente possono e desideri che l'utente scelga quale app utilizzare. Gli intent impliciti vengono scelti dal sistema operativo in base a tre fattori: action, data e category; ognuno di essi ha un tag specifico nel manifesto, da inserire all'interno di un <intent-filter>. Bisogna implementare una category di default, altrimenti il sistema operativo non riconoscerà nessuno di quegli intent come ideale per risolvere la computazione necessaria. Tramite costruttore o setters, quindi, è necessario specificare almeno uno tra i parametri Action, Data e Category, così che Android possa consultare il manifesto per cercare un'activity adeguata.

Thread

Android differenzia il thread principale dagli altri thread. Quali sono le operazioni non permesse nel thread principale (main) e quali quelle non permesse nei thread secondari (background)? Si motivi la risposta.

Le app Android vengono eseguite per impostazione predefinita sul thread principale, chiamato anche thread dell'interfaccia utente. Gestisce tutti gli input dell'utente e quelli di output, quindi si evitano operazioni che richiedono tempo, a differenza di un Thread secondario, che include chiamate di rete, decodifica di bitmap o lettura e scrittura dal database

Un’app che utilizza una connessione Internet sfrutta la classe AsyncTask. Tale classe prevede i seguenti metodi: onPreExecute(), doInBackground(), onProgressUpdate(), onPostExecute(). Cosa viene tipicamente implementato in questi metodi? Nella risposta specificare per ogni metodo una possibile operazione da implementare nel metodo.

Quando viene eseguita un'attività asincrona, l'attività passa attraverso 4 passaggi:

- onPreExecute(), richiamato sul thread dell'interfaccia utente prima dell'esecuzione dell'attività. Questo passaggio viene normalmente utilizzato per impostare l'attività, ad esempio mostrando una barra di avanzamento nell'interfaccia utente.
- doInBackground(Params ...), richiamato sul thread in background immediatamente dopo che onPreExecute() termina l'esecuzione. Questo passaggio viene utilizzato per eseguire calcoli in background che possono richiedere molto tempo. I parametri dell'attività asincrona vengono passati in questo passaggio ed il risultato del calcolo deve essere restituito da questo passaggio. Si può anche utilizzare publishingProgress(Progress ...) per pubblicare una o più unità di

progresso. Questi valori sono pubblicati sul thread dell'interfaccia utente, nel passaggio onProgressUpdate(Progress ...).

- onProgressUpdate(Progress ...), richiamato sul thread dell'interfaccia utente dopo una chiamata a publishingProgress(Progress ...). I tempi dell'esecuzione non sono definiti. Questo metodo viene utilizzato per visualizzare qualsiasi forma di progresso nell'interfaccia utente mentre il calcolo in background è ancora in esecuzione. Ad esempio, può essere utilizzato per animare una barra di avanzamento.
- onPostExecute(Result), richiamato sul thread dell'interfaccia utente al termine del calcolo in background. Il risultato del calcolo in background viene passato a questo passaggio come parametro.

Fragment

Si descriva il meccanismo dei Frammenti. Si faccia un esempio per illustrarne l'utilità.

I frammenti sono porzioni di interfaccia utente che hanno una vita propria nell'activity che li ospita: per questo, talvolta sono chiamati anche sub-activity. Essi hanno un differente ciclo di vita dall'activity, ma esso dipende da quello dell'activity che lo ospita: se l'activity ospitante viene distrutta, vengono distrutti anche i vari frammenti. Un frammento può essere creato staticamente utilizzando un tag <fragment> da inserire nel layout dell'activity ospitante. Il layout del frammento, tuttavia, dev'essere specificato in un file XML separato. Nel momento in cui viene invocato il metodo onCreateView() del frammento, occorre effettuare l'inflate del file di layout corrispondente nel tag del layout principale: dopo questa operazione, il frammento è Created nell'activity che lo ospita. Dopo che l'activity viene stoppata e prima che essa viene distrutta, viene effettuato il detatch del frammento (metodo onDetatch()). Una classe Fragment cura l'utilizzo dinamico del frammento nel corso dell'interazione con l'utente. Ad esempio, un frammento può essere anche inserito dinamicamente nel layout a runtime. Occorre eseguire il codice posto di seguito:

- FragmentManager fm = getFragmentManager();
- FragmentTransaction ft = fm.beginTransaction();
- ExampleFragment fragment = new ExampleFragment();
- ft.add(R.id.fragment_container, fragment); /* R.id.fragment_container è un ViewGroup nel layout dell'activity che individua la porzione dello schermo da dedicare a questo frammento */

Normalmente, un frammento non viene aggiunto nel backstack: si usa il metodo addToBackstack() per inserire i cambiamenti nel backstack; questo perché il backstack considera solo le activity, pertanto dobbiamo gestire manualmente i frammenti. Se non chiamiamo questo metodo, quando premiamo Back salteremo i cambiamenti fatti con i frammenti: non è quello che l'utente si aspetta. Durante il corso è stato analizzato un esempio in cui occorreva far comunicare due frammenti. In tal senso, è bene specificare che un frammento dovrebbe evitare la comunicazione diretta con un altro frammento della

stessa activity, in quanto ciò riduce la riusabilità; al contrario, è opportuno utilizzare l’activity ospitante come "comunicatore" tra il frammento mittente ed il frammento destinatario.

Se due frammenti di un activity devono comunicare è buona prassi di programmazione implementare tale comunicazione non in modo diretto da frammento a frammento ma passando attraverso l’activity che ospita i frammenti (quindi la frammento che vuole inviare la comunicazione lo fa interagendo con l’activity ospitante e poi questa interagisce con il frammento che deve ricevere la comunicazione). Perché è una buona prassi di programmazione? Si descriva un modo per implementare la comunicazione fra due frammenti attraverso l’activity ospitante.

Usare l'activity come intermediario per la comunicazione tra due frammenti comporta diversi vantaggi:

- Disaccoppiamento; i frammenti non avrebbero una dipendenza diretta e potrebbero dunque restare modulari e facilmente riutilizzabili.
- Manutenibilità; l'activity 'mediatore' facilita modifiche future, riducendo l'impatto su ciascuna delle parti coinvolte.
- Gestione del ciclo di vita, in quanto i frammenti potrebbero non essere attivi allo stesso tempo.
- Chiarezza del flusso di dati, rendendo il codice più leggibile.

Networking

Quando un’app ha bisogno di comunicare via Internet che cosa è necessario fare? Si faccia un esempio usando uno snippet di codice (senza dettagli ma solo con la struttura generale).

Per eseguire operazioni di rete nell'applicazione, il manifest deve includere le seguenti autorizzazioni:

- <uses-permission android:name="android.permission.INTERNET" />
- <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

Per evitare di creare un'interfaccia utente che non risponde, non eseguire operazioni di rete sul thread dell'interfaccia utente. Per impostazione predefinita, da Android 3.0 (livello API 11) si richiede di eseguire operazioni di rete su un thread diverso dal thread dell'interfaccia utente principale; se non lo si fa, viene lanciata una NetworkOnMainThreadException.

Storage

In che modo (o modi) varie activity che fanno parte della stessa app possono condividere dati? Si discuta dei vantaggi e svantaggi di ciascuno dei modi descritti.

Varie activity facenti parte della stessa app possono condividere dati in tre modi:

- Le SharedPreferences sono delle preferenze (localizzate in opportuni file di minime dimensioni) condivise tra più activity, che consentono la memorizzazione, tipicamente, di valori primitivi attraverso setter e getter. Per ricavare le preferenze condivise, è possibile utilizzare il metodo getDefaultSharedPreferences(Context c) oppure getSharedPreferences(String filename). Per salvare informazioni, occorre un oggetto SharedPreferences.Editor, ricavabile attraverso il metodo edit() sulle preferenze ricavate: su quest'oggetto è possibile invocare i vari setter per salvare dati primitivi; dopo aver inserito i vari dati, occorre effettuare il commit() dell'editor.
- I File sono il meccanismo più semplice ed efficiente per salvare informazioni (anche dati non primitivi, cioè oggetti serializzabili). I file utilizzabili con Android possono risiedere in tre memorie differenti: interna, esterna e cache; chiaramente, le directory per utilizzare i file nelle tre categorie precedenti sono diverse, ed esistono metodi che consentono di ricavarle in modo semplice. Il meccanismo di lettura/scrittura su file è, ovviamente, quello classico di Java. Per scrivere su file sulla memoria esterna è necessario richiedere l'opportuno permesso. Ogni applicazione in Android ha la sua directory privata, a cui possono accedere le sole activity della stessa applicazione.
- I Database sono delle strutture che consentono la memorizzazione persistente di informazioni. Android consente un utilizzo semplificato (seppur analogamente complesso da utilizzare) di basi di dati relazionali SQL attraverso SQLite.

Vantaggi e svantaggi di ciascuno dei modi descritti dipendono principalmente da tre fattori: dati memorizzabili, semplicità ed efficienza. Le SharedPreferences possono memorizzare soltanto dati primitivi, sono molto semplici da utilizzare ed efficienti, in quanto trattano dati atomici facilmente ricavabili e memorizzabili in file opportuni di basse dimensioni. I file possono memorizzare qualsiasi oggetto al loro interno (nel caso in cui siano file puramente testuali o binari), sono semplici ed efficienti da utilizzare. I database sono strutture che consentono di memorizzare dati strutturati e relazionati in tabelle (quindi, si parla di dati complessi) persistentemente, sono (non molto) semplici da utilizzare, ma sono spesso inefficienti.

Si descrivano le SharedPreferences. Cosa sono? Come funzionano? Che differenza c'è fra SharedPreferences e Preferences?

Le SharedPreferences sono uno dei tre meccanismi utilizzati per effettuare in modo persistente la memorizzazione di dati condivisi tra più activity della stessa applicazione in un file di piccole dimensioni. Le preferenze condivise permettono la memorizzazione di soli dati primitivi (interi, caratteri, floating points, booleani, …); tuttavia, sono molto efficienti e facili da usare. Per ottenere le preferenze condivise, è possibile usare uno dei seguenti metodi:

- getDefaultSharedPreferences(), che ricava le preferenze condivise di default dell'applicazione;
- getSharedPreferences(String filename), che ricava le preferenze condivise nel file specificato.

Per inserire un valore nelle SharedPreferences, occorre dapprima ricavare uno SharedPreferences.Editor attraverso il metodo edit() sull'oggetto SharedPreferences ricavato; in

seguito, è possibile chiamare uno dei metodi putInt(String tag, int value), putChar(String tag, char value), …, sull’editor; dopo aver inserito i dati, occorre effettuare il commit() dell’editor. Per ricavare un valore dalle SharedPreferences, semplicemente s’invoca uno dei metodi seguenti sull’oggetto SharedPreferences ricavato: getInt(String tag), getChar(String tag), … Bisogna fare attenzione quando si invoca il metodo getSharedPreferences(). Esiste, infatti, un metodo getPreferences() che ricava le preferenze della singola activity, pertanto NON CONDIVISE tra tutte le activity dell’applicazione, a differenza di getSharedPreferences().

Stai scrivendo un’app che utlizza 3 activity che devono condividere dei parametri specificati dall’utente come preferenze: la grandezza dei caratteri, misurata in pixels, un indirizzo email, e un colore. Come fai in modo che le 3 activity condividano questi dati? Fornire dei dettagli.

Possiamo scomporre il problema in diverse fasi:

- Identificare i dati da condividere: in questo caso la grandezza dei caratteri, un indirizzo email ed un colore.
- Scegliere un meccanismo di memorizzazione: poiché i dati devono essere persistenti e accessibili da più Activity, un'opzione efficiente è utilizzare SharedPreferences, un sistema di memorizzazione locale fornito da Android per gestire le impostazioni dell'utente.
- Centralizzare la gestione dei dati: creare un sistema centralizzato per leggere e scrivere i dati in modo coerente, ad esempio una classe helper dedicata che incapsula l'accesso a SharedPreferences.
- Salvataggio e recupero dei dati: quando l'utente modifica una preferenza (es. cambia la dimensione del carattere), l'app aggiorna il valore memorizzato. Quando un’Activity viene avviata, essa recupera i dati salvati e li applica agli elementi dell'interfaccia.
- Garantire una buona esperienza utente: assicurarsi che i cambiamenti si riflettano immediatamente o alla successiva apertura dell’app, in modo che le preferenze siano sempre rispettate senza richiedere configurazioni ripetute.

Animazioni

Cosa è l’albero delle view? Quali sono e come funzionano le due fasi necessarie per la visualizzazione del layout?

L’albero delle view è l’insieme dei widget che formano un determinato layout, strutturato secondo una gerarchia ad albero in quanto ogni file XML ne conserva la struttura. Alla radice dell’albero si trova un singolo ViewGroup, al livello successivo sono posti i suoi figli, al livello successivo sono posti i suoi "nipoti" e così via, fin quando non si raggiungono le foglie dell’albero. Le fasi necessarie per la visualizzazione del layout sono tre:

- **Misurazione:** fase in cui viene fatta una visita top-down dell’albero per misurare width e height di ciascun elemento in modo ricorsivo. In questa fase si effettuano in realtà due misurazioni: una "temporanea" in cui ogni View, tramite LayoutParams, indica al padre quanto grande vorrebbe essere, ed una definitiva in cui il padre, tramite MeasureSpec, indica le dimensioni effettive dell'elemento.
- **Posizionamento:** fase in cui viene fatta una visita top-down dell’albero per posizionare ciascun elemento dell'albero in modo ricorsivo.
- **Disegno:** fase in cui si disegna ogni view nel layout dell’interfaccia.

Si spieghi come avviene la misurazione e il posizionamento delle view di un layout. Perché in alcuni casi i metodi v.getWidth e v.getHeight, dove v è una view del layout, usati in onCreate() restituiscono 0?

Una View è un oggetto che l'utente può vedere e con cui può interagire e fa parte di un ViewGroup, un container invisibile che definisce il layout con cui mostrare tutti gli oggetti (o raggruppamenti di essi) che contiene.

La fase di misurazione ricava la misura che le varie view richiedono e calcola la misura reale che le varie view avranno, in relazione alla view genitore e ad i suoi fratelli (cioè, le view che risiedono nello stesso livello dell’albero): si parte dalla radice e si analizza ognuno dei sottoalberi di ogni figlio ricorsivamente, fin quando tutte le view sono state misurate. La fase di posizionamento prevede il posizionamento delle view del layout secondo le misure calcolate nella fase precedente. In alcuni casi, nel metodo onCreate() non è ancora possibile ricavare le misure (width ed height) delle varie view, in quanto è possibile che esse non siano state ancora posizionate e disegnate nel layout, anche dopo aver invocato il metodo setContentView().

Che cosa è e a cosa serve l’oggetto Canvas? Si illustri, con sufficienti dettagli, un esempio in cui è necessario utilizzare un oggetto Canvas.

L’oggetto Canvas consente di disegnare una view customizzata in un determinato punto del layout. Lo step più impostante per disegnare una view personalizzata è eseguire l’override del metodo onDraw(), che riceve come parametro un oggetto Canvas che la view può usare per disegnare sé stessa. La classe Canvas definisce metodi per disegnare testo, linee, bitmap e molte altre grafiche primitive. Si possono usare i suoi metodi in onDraw() per creare la propria UI customizzata. Prima di poter chiamare qualsiasi metodo di disegno, tuttavia, è necessario creare uno o più oggetti Paint. Ad esempio, un Canvas fornisce un metodo per disegnare una linea, mentre Paint fornisce metodi per definire il colore della linea; Canvas fornisce un metodo per disegnare un rettangolo, mentre Paint definisce se riempire tale rettangolo con un colore o lasciarlo vuoto. Semplicemente, Canvas definisce le forme che si possono disegnare sullo schermo, mentre Paint definisce colore, stile, font e così via, per ogni forma si vuole disegnare. Un esempio che richiede un oggetto Canvas può essere quello di un applicazione che mostra un pentagramma su cui è possibile posizionare e rimuovere delle note tramite click.

Come funzionano le animazioni? Come si può animare un oggetto grafico in Android? Si faccia un esempio scegliendo una specifica animazione.

Le animazioni consentono ad una determinata view di aggiungervi degli effetti speciali come rotazione, traslazione, scaling e trasparenza. Per creare un'animazione, occorre creare un file di layout XML: alla radice dell'albero dev'essere posto un elemento <set> (oppure <objectAnimator> e <valueAnimator>), che indica l'insieme di animazioni da eseguire, al cui interno è possibile porre <animator> ed <objectAnimator>, ossia le singole animazioni, o ulteriori elementi <set>. Ogni file d'animazione può essere memorizzato in un oggetto Animation. Per assegnare un'animazione ad un oggetto animation, occorre invocare il metodo statico AnimationUtils.loadAnimation(Context context, int resource), che ritorna un oggetto Animation. Un'animazione può essere utilizzata sulla maggior parte delle View di un layout, utilizzando il metodo View.startAnimation(Animation animation).

- <rotate
  - android:startOffset="0"
  - android:duration="4000"
  - android:fromDegrees="0"
  - android:toDegrees="180"
  - android:pivotX="50%"
  - android:pivotY="50%"
- />

Nella nostra MainActivity verrà istanziato un oggetto Animation che verrà eguagliato nel seguente modo:

- AnimationUtils.loadAnimation(getApplicationContext(), R.anim.rotazione);

Verrà definito un metodo invocato al click del bottone che starterà l'oggetto con la relativa animazione.

Il seguente snippet di codice esegue prima un'animazione e poi rimuove l'oggetto dalla view parent (gli oggetti image, animation e parentView sono stati in precedenza opportunamente inizializzati):

image.startAnimation(animation);

parentView.removeViewAt(0);

Tuttavia l'effetto è quello di rimuovere immediatamente l'immagine senza dare il tempo all'animazione di essere eseguita. Perché accade ciò? Come si può ovviare al problema?

Il problema si verifica perché l'animazione la seconda riga di codice viene eseguita immediatamente, rimuovendo la vista prima che l'animazione possa essere visibilmente completata. L'animazione viene infatti eseguita in un thread separato rispetto all'esecuzione del codice, quindi il flusso del programma continua senza attendere la fine dell'animazione. Per impedire che avvenga ciò, bisogna usare un Listener sull'animazione in modo da rimuoverla ad animazione terminata.

MultiTouch

Che cosa è un oggetto MotionEvent? Si forniscano dettagli sul suo utilizzo.

Un MotionEvent è un oggetto utilizzato per segnalare eventi di movimento (mouse, penna, dito), rappresenta un singolo pointer e a volte più pointer. Ad esempio, quando l'utente tocca per la prima volta lo schermo, il sistema invia un evento di tocco all'appropriato View con il codice di azione ACTION_DOWN e un insieme di valori degli assi che includono le coordinate X e Y del tocco e informazioni sulla pressione, le dimensioni e l'orientamento dell'area di contatto. La classe MotionEvent fornisce molti metodi per interrogare la posizione e altre proprietà di puntatori, come getX(int), getY(int), getAxisValue(int), getPointerId(int), getToolType(int), e molti altri. La maggior parte di questi metodi accetta l'indice del puntatore come parametro anziché l'id del puntatore. L'indice del puntatore di ciascun puntatore nell'evento varia da 0 a uno in meno del valore restituito da getPointerCount(). Per ottenere un azione di un MotionEvent si dovrebbe sempre usare il metodo getActionMasked(), è progettato per funzionare con più puntatori. Restituisce l'azione mascherata che viene eseguita, senza includere i bit dell'indice del puntatore.

Si descriva il meccanismo del Multitouch. Come vengono rappresentati i movimenti? Che cosa è un "pointer"? Che cosa è un MotionEvent?

Il Multitouch è un meccanismo che permette al dispositivo di reagire opportunamente in base al numero di dita che toccano lo schermo. Il movimento è rappresentato con: ACTION_CODE (cambiamento avvenuto) e ACTION_VALUES (posizione e proprietà del movimento). Ogni evento è rappresentato da un "pointer", mentre MotionEvent rappresenta uno o più pointer. Alcuni eventi sono:

- ACTION_DOWN, che rappresenta la pressione di un singolo dito;

- ACTION_POINTER_DOWN, che rappresenta la pressione di un secondo, terzo, …, dito;
- ACTION_MOVE, che rappresenta il movimento di uno o più dita;
- ACTION_POINTER_UP, che rappresenta il rilascio di un secondo, terzo, …, dito dallo schermo;
- ACTION_UP, che rappresenta il rilascio dell'ultimo dito dallo schermo.

Ogni pointer ha un indice: se un singolo dito effettua la pressione, esso ha indice 0; se un secondo dito effettua la pressione, il primo avrà indice 0 ed il secondo avrà indice 1; e così via. Questi indici vengono memorizzati per individuare quale pointer ha eseguito una determinata operazione. Ogni pointer ha un id univoco per tutta la durata della pressione, fino al momento dopo il rilascio. MotionEvent è la classe che cura la gestione degli eventi elencati in precedenza: essa rappresenta un movimento registrato da una periferica (mouse, trarckboll, penna, dita, …). È possibile invocare numerosi metodi su un oggetto MotionEvent, quali:

- getActionMasked(), che ritorna l'action code catturato;
- getActionIndex(), che ritorna l'indice del pointer che ha scaturito l'evento;
- getPointerId(int pointerIndex), che ritorna l'ID del pointer specificato.

Sensori

Si descriva l'utilizzo di un sensore spiegando cosa si deve fare per utilizzarlo.

I sensori consentono di misurare il movimento, l'orientamento e varie condizioni ambientali, sono in grado di fornire dati grezzi con elevata precisione e accuratezza e sono utili se si desidera monitorare il movimento o il posizionamento tridimensionale del dispositivo o se si desidera monitorare i cambiamenti nell'ambiente circostante ad un dispositivo. Android supporta tre grandi categorie di sensori:

- *Sensori di movimento*, misurano le forze di accelerazione e le forze di rotazione lungo tre assi, categoria che comprende accelerometri, sensori di gravità, giroscopi e sensori vettoriali rotazionali.
- *Sensori ambientali*, misurano vari parametri ambientali, come temperatura e pressione dell'aria ambiente, illuminazione e umidità, categoria che comprende barometri, fotometri e termometri.
- *Sensori di posizione*, misurano la posizione fisica di un dispositivo, categoria che comprende sensori di orientamento.

L'oggetto SensorManager gestisce il tutto, comunicando quali sono i sensori disponibili e quali sono le loro caratteristiche e permettendoci sia di leggere i dati grezzi che usare dei Listeners sui cambiamenti. Come metodi principali troviamo:

- *onSensorChanged()*, per reagire ai cambiamenti del sensore;

- *onResume()* e *onPause()* per registrare e rilasciare il sensore per risparmiare batteria.

Quando si registra il listener di un sensore è possibile selezionare la velocità di campionamento da utilizzare:

- ***SENSOR_DELAY_NORMAL (0.2sec)***
- ***SENSOR_DELAY_GAME (0,02sec)***
- ***SENSOR_DELAY_UI (0,06sec)***
- ***SENSOR_FASTEST (0sec)***

Si discuta dei vantaggi e svantaggi di queste varie possibilità e di quali accortezze deve avere il programmatore per un app che utilizza i sensori.

Un buon programmatore dovrebbe riuscire a trovare un compromesso tra reattività e consumo energetico in base all'applicazione che sta sviluppando. Per decidere come impostare la velocità di campionamento, dovrebbe tenere in conto diversi fattori:

- bilanciare precisione e consumo, usando SENSOR_FASTEST solo se necessario e preferendo SENSOR_DELAY_GAME per giochi o SENSOR_DELAY_UI per UI reattive;
- gestire la concorrenza, elaborando i dati in un thread separato;
- ottimizzare la gestione dei dati, riducendo il numero di aggiornamenti elaborati e filtrando i dati ridondanti;
- disattivare i sensori quando non necessari, interrompendo il listener quando l'attività non è in primo piano;
- gestire le variazioni tra dispositivi, testando su tipologie diverse.

Notifiche

Quali sono i principali modi per fornire notifiche all’utente? Si fornisca una breve descrizione.

- ***Toast:*** una piccola notifica popup che viene utilizzata per visualizzare informazioni sull'operazione che abbiamo eseguito nella nostra app. Un toast mostrerà il messaggio per un breve periodo di tempo e scomparirà automaticamente dopo un timeout. Generalmente, la notifica Toast in Android verrà visualizzata con un testo semplice.
- ***Dialog:*** una piccola finestra che richiede all'utente di prendere una decisione o inserire informazioni aggiuntive. Una finestra di dialogo non riempie lo schermo e viene normalmente utilizzata per eventi modali che richiedono agli utenti di eseguire un'azione prima di poter procedere.

- ***Notification Area (Status Bar):*** è un messaggio che Android visualizza all'esterno dell'interfaccia utente dell'app per fornire all'utente promemoria, comunicazioni di altre persone o altre informazioni tempestive dall'app. Gli utenti possono toccare la notifica per aprire l'app o eseguire un'azione direttamente dalla notifica. Le notifiche possono apparire brevemente in una finestra mobile denominata notifica *heads-up* (da API 21). Questo comportamento è in genere per le notifiche importanti che l'utente dovrebbe conoscere immediatamente e appare solo se il dispositivo è sbloccato.

Che cosa è un Toast customizzato? Si spieghi come implementare un Toast customizzato.

I toast sono una tipologia di notifica usata per mostrare piccoli feedback riguardo una certa operazione: si tratta di popup di breve durata che riempiono soltanto quella porzione di spazio necessaria a mostrare il breve messaggio che portano, lasciando l'activity visibile ed interagibile.

Per crearne uno sono necessari il messaggio da mostrare e la durata del popup e si usano i metodi makeText(context, text, duration) per creare il toast e show() per mostrarlo.

Permessi

Si descriva il meccanismo dei permessi spiegando la differenza fra permessi normali e permessi pericolosi. Si metta in evidenza la gestione dei permessi in gruppi spiegando come vengono gestiti tali gruppi.

I permessi sono meccanismi di protezione per le risorse e i dati, a cui si può accedere solo tramite un consenso esplicito dell'utente. Servono a limitare l'accesso ad informazioni sensibili, risorse di sistema e servizi con costi ed è necessario dichiarare nel manifesto dell'app quali permessi si vuole chiedere. Possiamo dividerli in due categorie: normali, se concessi automaticamente, e pericolosi, se invece vanno approvati al momento dell'installazione o a runtime; la decisione è sempre modificabile dalle impostazioni. I permessi sono rappresentati come delle stringhe ed appartengono a dei macro-gruppi, ad esempio Calendar contiene Read_Calendar e Write_Calendar: quando un app chiede un permesso pericoloso, questo viene concesso automaticamente se essa ha già un permesso per lo stesso gruppo oppure viene richiesto all'utente il permesso per l'intero gruppo.

Broadcast, Content Providers e Services

Le activity sono una delle 4 componenti principale di un’app. Quali sono le altre 3? Si fornisca una breve descrizione.

- Broadcasts: possono essere utilizzate come sistema di messaggistica tra le app e al di fuori del normale flusso di utenti. Tuttavia, è necessario fare attenzione a non abusare dell'opportunità di rispondere alle trasmissioni ed eseguire lavori in background che possono contribuire a

rallentare le prestazioni del sistema. Ad esempio, il sistema Android invia trasmissioni quando si verificano vari eventi di sistema, ad esempio quando il sistema si avvia o il dispositivo inizia a caricarsi.

- Content Providers: fornisce i dati da un'applicazione ad altre su richiesta. Tali richieste sono gestite dai metodi della classe ContentResolver. Un Content Providers può utilizzare diversi modi per archiviare i propri dati ad esempio in un database, in file o persino su una rete.
- Services: è un componente dell'applicazione che può eseguire operazioni di lunga durata in background e non fornisce un'interfaccia utente. Un altro componente dell'applicazione può avviare un servizio e continua a essere eseguito in background anche se l'utente passa a un'altra applicazione. Ad esempio, un servizio può gestire transazioni di rete, riprodurre musica, eseguire operazioni di I / O su file o interagire con un fornitore di contenuti, tutto da sfondo.

Domande ultimo appello

definizione di frame layout

differenze tra listView semplice, custom base e custom con click multiplo

descrizione del backstack

descrizione di startActivityForResult e onResult qualcosa

vita morte e miracoli dei fragments

descrizione audioFocus

broadcast receiver, definizione + quando si dichiarano nel manifest e quando dinamicamente

due esercizi in cui scrivere codice: uno sulle asyncTask e l'altro con un widget personalizzato