Domande Teoria MP

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 in cui vengono inserite tutte quelle attività che non sono attualmente in foreground ed inizia, generalmente, dall'Home Screen. Un fragment è una porzione dell'interfaccia che viene ospitata da un'activity ed ha un proprio ciclo di vita. Quando l'utente clicca un'icona della Home, l'applicazione viene portata in foreground; se vengono lanciate nuove activity da quella corrente, questa viene messa nel backstack e la nuova passa in foreground: vi si può tornare indietro con il pulsante Back. 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.

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 cilco 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 impedire ciò bisogna usare onSaveInstanceState e si recupera succesivamente lo stato con onCreate().

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.

Esistono diverse opzioni quando si tratta l'archiviazione dei dati in Android: gli app-specific files sono file privati ad uso esclusivo dell'app, così come sono privati sia le preferences che i database in SQLite, mentre nello shared storage è possibile inserire file da condividire previa autorizzazione, come media e documenti.

Per condividere dati tra più activities possiamo dichiarare statiche le risorse che vogliamo condividere, per poi accedervi tramite getters e setters, ma un modo più comune è quello di ricorrere agli Intent, che offrono più flessibilità. Possiamo creare un nuovo Intent a cui forniamo come primo parametro il contesto dell'attività chiamante e come secondo l'activity che vogliamo chiamare e a cui vogliamo passare i dati. Dopo aver inserito i valori tramite putExtra(name, value), chiamiamo start(activity). Nella seconda activity invece usiamo getIntent() per creare l'intent e poi getExtra(name) per ottenere il valore inserito. Il metodo spiegato è detto 'Direct Intent', in quanto viene usato l'Intent come contenitore dei dati, ma è possibile seguire una seconda strada, quella dei Bundle: l'Intent conterrà al suo interno soltanto un oggetto Bundle, che si occuperà di mantenere tutti i dati al suo interno. L'uso di un Bundle rende il codice leggermente più pesante, ma lo rende anche più facile da leggere e gestire; inoltre, i Bundle consentono una serializzazione più pulita dei dati prima del loro invio, se necessario.

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. L'albero delle View viene costruito nel seguente modo: i figli comunicano al padre, tramite LayoutParams, quanto vorrebbero essere grandi e dove vorrebbero essere posizionati; successivamente, il padre, tramite MeasureSpec, comunica le dimensioni effettive. È possibile dichiarare elementi del layout sia staticamente tramite XML che dinamicamente in modo programmatico: il primo consente di avere una separazione più netta tra UI e logica sacrificando la flessibilità, mentre il secondo è facile da adattare, ma impone la gestione del layout all'interno del codice. Con il seconodo metodo è possibile, inoltre, andare in contro ad alcuni problemi, come i metodi getWidth e/o getHeight che restituiscono 0 se usati nel metodo onCreate(). Questo tipo di problemi si riconduce al fatto che bisogna esplorare l'albero delle View ed eseguire tutti i calcoli per l'inserimento di nuovi elementi nella UI e questo richiede tempo. In genere, la fase di calcolo dura fino alla onResume(), dunque bisogna prevedere un modo per aspettare fino a quel momento: un metodo potrebbe essere quello di definire un ViewTreeObserver in modo da ascoltare eventi Draw e Layout.

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.

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.

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?

Nel caso in cui un ListView sia personalizzato e presenti più sottoelementi, non è possibile usare un Listener semplice per distinguere cos'è stato selezionato nello specifico, in quanto il riferimento sarà sempre quello dell'intero elemento. Questo implica che è necessario scrivere dei Listener ad-hoc per ciascun sottoelemento e per capire in che posizione dell'array ci troviamo dobbiamo usare getTag().

Il 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 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().

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.

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.

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 per ognuna di esse.

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.