**Prova scritta 26/01/2018**

**Quesito 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.

I vantaggi di definire un layout staticamente, cioè tramite un file XML, è sicuramente la più facile “manutenzione” e modellazione, è più semplice da creare. Lo svantaggio è che poiché è definito staticamente, può essere modificato soltanto dinamicamente tramite codice Java.

Al contrario, il layout dinamico ha il vantaggio di poter essere gestito runtime, quindi facilmente adattabile, ma lo svantaggio è che si va a mescolare con la logica del codice Java che c’è dietro l’applicazione. Ovviamente i due metodi non sono mutualmente esclusivi.

Un esempio di quando si deve necessariamente usare un layout dinamico rispetto ad uno statico è quando si utilizzano le listView. Infatti, lo “spazio” della ListView viene definito staticamente nel file XML, ma per inserire gli elementi all’interno del ListView, l’unico modo possibile per farlo è dinamicamente.

**Quesito 2:** Un listview prevede on 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?

Un modo per poter reagire in modo diverso in base ad ogni elemento cliccato è innanzitutto assegnare un diverso metodo “onClick” ad ogni sottoelemento appartenente all’item. Successivamente, nel CustomAdapter che creiamo, oltre a settare il layout di ogni singolo item con i propri dati, setteremo anche un altro attributo ad ogni singolo sottoelemento, un tag, al quale associamo la posizione dell’item. Così, quando clicchiamo sul sottoelemento, potremo riprendere, tramite gli opportuni metodi, la posizione alla quale appartiene il sottoelemento cliccato e potremo fare le nostre operazioni su di esso.

**Quesito 3:** 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. Perché? 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.

Durante il suo ciclo di vita, l’app può passare in diversi stati: Created, Started, Resumed, Paused, Stopped, Destroyed. Ogni stato è importante all’interno del ciclo di vita dell’app, perché ad ogni stato corrisponde un particolare “setup” che l’app utilizza, nel senso che:

* Created: In questo stato l’app viene lanciata e creata, si setta il layout della Main Activity, si caricano le prime informazioni necessarie per l’app;
* Started: in questo stato, l’app viene resa visibile all’utente e viene preparata per essere interattiva con l’utente;
* Resumed: è lo stato in cui la maggior parte delle app passano la maggior parte del tempo, poiché è in questo stato che l’utente può comunicare e utilizzare i servizi dell’app;
* Paused: viene chiamato questo stato quando l’app non è più in foreground, e in questa fase possono esserci salvataggi temporanei di informazioni utili all’app;
* Stopped: in questa fase l’app è in background, e può essere distrutta dal SO nel caso in cui ci sia bisogno di spazio in memoria. Tuttavia, è sempre presente nel caso in cui l’utente voglia utilizzarla nuovamente nel giro di poco tempo;
* Destroyed: l’app viene definitivamente distrutta, e per essere riutilizzata deve essere nuovamente ricreata.

La complessità di tutto questo ciclo di vita, e le varie fasi che la compongono, rispondono al motivo per cui è necessario che ci siano così tanti metodi per gestire il ciclo di vita di ogni singola app, perché un’app in esecuzione e messa in backgroud potrà essere riutilizzata anche nel breve termine, e quindi ricreata non dovendo richiamare il metodo della creazione dell’interfaccia.

**Quesito 4:** Si scrivano degli snippet di codice per lanciare da un’activity “Principale” un’altra activity, “Secondaria”, passando un valore di tipo intero dall’activity principale a quella secondaria e facendo in modo che l’activity secondaria restituisca un valore di tipo stringa all’activity principale.

Main Activity:

Intent i = new Intent();

i.setClass(getApplicationContext(), Activity2.class);

i.putExtra(“INTEGER”, intero);

startActivityForResult(i, 123);

public void onActivityResult(int requestCode, int resourceId, Intent i) {

String stringa = i.getStringExtra(“STRING”);

}

Activity2:

public void onCreate(Bundle onSavedInstanceState) {

super.onCreate(onSavedInstanceState);

Intent i = getIntent();

int i = i.getIntExtra(“INTEGER”, 0);

}

public void return() {

Intent i = new Intent();

i.putExtra(“STRING”, stringa);

setResult(RESULT\_OK, i);

}

**Quesito 5:** 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.

La differenza principale tra i permessi normali e pericolosi è il fatto che i permessi normali non vengono richiesti all’utente, mentre per i permessi pericolosi c’è bisogno dell’autorizzazione dell’utente. Permessi normali possono essere internet, bluetooth, NFC, mentre quelli pericolosi sono chiamate, fotocamera, contatti, GPS…

Ogni permesso ha un “gruppo”, cioè particolari azioni che possono essere fatti su di esso. Infatti, ad esempio, se si dà accesso alla propria lista dei contatti, quindi verrà accettata il permesso di accedere i contatti, verranno accettati una lista di operazioni che quell’app potrà fare con i contatti. Ad esempio, un’app malevola potrebbe anche prelevare i numeri di cellulari dei nostri contatti, mentre noi pensavamo quell’applicazione avrebbe utilizzato i contatti solo per visualizzarli tutti in quella specifica app. Quindi quando si dà il permesso, in realtà si dà l’accesso COMPLETO a tutto il gruppo di permessi associati alla “classe”.

**Quesito 6:** Si completi il seguente codice assumendo di avere a disposizione la funzione “partialLoad()” che si occupa di caricare in ogni chiamata un 10% dell’immagine img (quindi dopo dieci chiamate a tale funzione img sarà completa). Si renda visibile la ProgressBar all’inizio del caricamento e invisibile alla fine. Si aggiorni la progress bar ad ogni 10% di caricamento e si mostri un Toast di avviso “Caricamento quasi completato” quando si raggiunge l’80% del caricamento. Si mostri l’immagine nell’imageView alla fine del caricamento.

Public void onPreExecute() {

progressBar.setVisibility(ProgressBar.VISIBLE);

}

doInBackground(Intege… ids) {

Bitmap img;

Bitmap img = BitmapFactory.decodeResources(getResources(), immagine);

for(int i = 0; i < 10; i++) {

publishProgress(i \* 10);

}

Return img;

}

onProgressUpdate(Integer… values) {

progressBar.setVisibility(ProgressBar.VISIBLE);

progressBar.setProgress(values[0]);

if(values[0] > 80)

Toast.makeText(getApplicationContext(), “Stiamo terminando”, Toast.LENGTH\_LONG).show

}

onPostExecute() {

progressBar.setVisibility(ProgressBar.INVISIBLE);

}

**Quesito 7:** 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 il 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.

Per implementare un tale modo di comunicazione, il frammento che vuole inviare dati ad un altro frammento deve dichiarare un’interfaccia che sarà implementata dall’activity ospitante; nel metodo implementato dall’activity, poi, ci sarà la chiamata all’altro frammento. Questo modo di programmare è molto utile, poiché in questo modo c’è una bassa coesione tra i due frammenti, che nascono proprio per far diventare più dinamica l’applicazione, ma se ci sono chiamate dirette tra di loro, questa dinamicità e bassa coesione non ci sono più.

Quesito 8: Si fornisca un file XML per un’animazione che prima ruota la view di 180°e poi la sposta orizzontalmente di 200dp.

<set>

<rotation

android:startOffset=”0”

android:duration=”1000”

android:fromDegrees=”0”

android:toDegrees=”180”

/>

<traslate

android:startOffset=”1000”

android:duration=”1000”

android:fromXDelta=”0”

android:fromYDelta=”0”

android:toXDelta=”200”

android:toYDelta=”0”

/>

</set>

**Quesito 9:** 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 quale accortezze deve avere il programmatore per un app che utilizza i sensori.

Il vantaggio di utilizzare una velocità di campionamento maggiore è l fatto che i dati ricevuti saranno più precisi e che la latenza sarà minore, e ciò favorirà anche il miglioramento dell’esperienza di utilizzo del device durante il gioco o la visione di un film, ad esempio, mentre lo svantaggio sarà un maggior consumo di batteria, che quindi si scaricherà più velocemente.

Le accortezze del programmatore sono il fatto che innanzitutto si deve capire se il sensore è presente o meno nel dispositivo, e poi sarà cura del programmatore settare la frequenza di aggiornamento dei dati.

**Quesito 10:** Che cosa è un Toast customizzato? Si spieghi come implementare un Toast customizzato.

Un toast è una notifica pop-up temporanea che viene utilizzata molto frequentemente sui dispositivi Android per ricevere notifiche riguardo il click su un item, per un’azione andata a buon fine…

Il toast può essere costumizzato, cioè al suo interno può essere inserita una qualsiasi stringa a scelta del programmatore, e inoltre si può decidere anche la durata della visibilità del toast sul dispositivo: Toast.LENGTH\_LONG oppure Toast.LENGTH\_SHORT.

Per implementare un Toast, si devono usare i seguenti metodi:

Toast.makeText(getApplicationContext(), “Inserire la stringa da visualizzare qui”, Toast.LENGTH\_SHORT).show().