**Prova scritta del 01/2018**

**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.**

Con un file XML (metodo dichiarativo) vengono costruiti layout statici. Sono facili da specificare e separano in modo netto la definizione dell’interfaccia dal codice dell’applicazione. Sono quindi facili da modificare. Il loro vero e proprio svantaggio appunto è che non è possibile costruire layout dinamici.  
In modo programmatico, ossia tramite istruzioni nel programma che saranno eseguite a runtime, è possibile costruire layout dinamici. Il layout è però da gestire nel codice dell’applicazione e quindi di gestione più complicata rispetto a quella con file XML.

**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 sotto elementi (es, una foto, un nome, un numero), il riferimento passato al metodo onItemClick non distingue quale dei sotto elementi è stato selezionato. Come si può fare per reagire in maniera diversa in funzione di quale dei sotto elementi è stato selezionato con il click?**

Per poter ovviare a questo problema, un possibile approccio è il seguente:  
nel file xml dove viene definito il layout di ogni singolo elemento che compone la ListView, inseriamo il tag onClick per ogni elemento che compone il layout, in questo caso, inseriamo il tag onClick per la foto il nome e il numero, assegnandogli rispettivamente 3 metodi diversi, ad esempio onImageClick, onNameClick e onNumberClick.  
Ora nella classe che definisce il nostro customer adapter, precisamente nel metodo getView, oltre ad effettuare le operazioni necessarie, impostiamo un tag per ogni view, dove il tag in questione è proprio la posizione dell’oggetto dentro la ListView.  
In questo modo, quando andremo ad implementare i metodi onImageClick, onNameClick e onNumberClick, non solo riusciamo a riconoscere quale dei 3 elementi è stato cliccato in base al metodo che viene lanciato, ma possiamo anche conoscerne la posizione nella ListView semplicemente ottenendo il tag della view con il metodo view.getTag().

**3 – 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.  
Perché? Non sarebbe stato meglio avere un solo metodo come onEsecuzione() nel quale eseguire tutto ciò che viene fatto nei 3 metodi onCreate, onStart, onResume? Stessa analogia per la terminazione dell’applicazione.**

Non è possibile sostituire i 3 metodi onPause, onStop, onDestroy con un solo metodo semplicemente perché questi 3 metodi non vengono necessariamente chiamati sempre insieme nello stesso momento. Facciamo un esempio: quando l’applicazione viene chiusa, questi 3 metodi vengono chiamati in successione e quindi potremmo pensare di poter fare tutte le varie operazioni divise in questi 3 metodi in un solo metodo, ma supponiamo invece che l’applicazione non viene chiusa ma viene semplicemente portata in background (l’utente torna alla home screen o utilizza un’altra applicazione ma non termina l’applicazione di cui stiamo parlando), in questo caso non possiamo utilizzare un solo metodo onFine() perché così facendo termineremo la applicazione, e così facendo quando l’utente tornerà all’applicazione essa verrà ricostruita perdendo lo stato. Per questo motivo, in questo esempio, vengono eseguiti solo i metodi onPause e onStop, e quindi l’applicazione non viene distrutta ma viene messa nello stato Stopped. Un altro esempio può essere quando l’applicazione è composta da più activity: abbiamo un activity A che chiama l’activity B, così facendo l’activity A chiama solo il metodo onPause(), questo è possibile appunto perché esiste una suddivisione tra i vari stati, ciò non sarebbe possibile con un solo metodo onFine().

**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.**

public class Principale extends Activity {  
 …  
 public void lanciaActivity() {  
 Intent i = new Intent();  
 i.setClass(getApplicationContext(), Secondaria.class);  
 i.putExtra(“intero”, intero);  
 startActivityForResult(i, 0);  
 }  
 …  
}

public class Secondaria extends Activity {  
 …  
 public void ritornaRisultato() {  
 Intent i = new Intent();  
 i.putExtra(“stringa”, risposta);  
 setResult(RESULT\_OK, i);  
 }  
 …  
}

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

I permessi in Android vengono suddivise in tre famigli in base al livello di protezione:  
- permessi normali, che non metto cioè a rischio la privacy dell’utente;  
- permessi pericolosi, potenzialmente più lesive della riservatezza degli utenti;  
- permessi di firma, gestite a livello di installazione ma utilizzabili solo le l’app che le richiede è firmata con lo stesso certificato di quella che ha definito i permessi.

I permessi pericolosi vanno accettati a runtime dall’utente al momento dell’utilizzo della funzionalità che le richiedono per API >= 23, mentre quando si installa l’app per API < 23, mentre per quelle normali è sufficienti dichiararne l’impiego al momento dell’installazione dell’app. Ciò che cambia sul piano della progettazione è che l’app, in prospettiva potrebbe avere più comportamenti: uno in cui tutte i permessi pericolosi sono stati concessi, altri in cui uno o più di essi sono stati negati o revocati.

**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 il 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**

class LoadIconTask extends AsyncTask <Integer, Integer, Bitmap> {  
 private Integer index = 1;

@Override  
 protected void onPreExecute() {  
 ProgressBar.setVisibility(ProgressBar.VISIBLE);  
 }

@Override  
 protected Bitmap doInBackground(Integer… ids) {  
 Bitmap img;  
 img = BitmapFactory.decodeResource(  
 getResources(), img\_ids[0]);

for(int I = 0; I < 10; i++)   
 publishProgress(i);

return img;  
}

@Override  
 protected void onProgressUpdate(Integer… values) {  
 partialLoad();  
 progressBar.setStatus(values[0]);  
 if(values[0] > 75)  
 Toast.newToast(getApplicationContext(),  
 “Caricamento quasi finite”,  
 Toast.LENGHT\_LONG);  
 }

@Override  
 protected void onPostExecute(Bitmap result) {  
 ProgressBar.setStatus(0);  
 ProgressBar.setVisibility(ProgressBar.INVISIBLE);  
 imageView.setImageBitmap(result);  
 }

**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 la comunicazione tra due frammenti bisogna creare dei metodi di callback. Ad esempio, il frammento può definire un’interfaccia che utilizza l’activity principale per passare informazioni all’altro frammento. È buona prassi di programmazione utilizzare dei metodi di callback poiché la comunicazione diretta tra i due frammenti diminuisce la riusabilità.

**8 – Si fornisca un file XML per un’animazione che prima ruota la view di 180\* e poi la sposta orizzontalmente di 200dp.**

<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android"  
 android:shareInterpolator="false">  
  
 <rotate  
 android:startOffset="0"  
 android:duration="2000"  
 android:fromDegrees="0"  
 android:toDegrees="180"  
 android:pivotX="50%"  
 android:pivotY="50%" />  
  
 <translate  
 android:startOffset="2000"  
 android:duration="4000"  
 android:fromXDelta="0"  
 android:toXDelta="200" />  
  
</set>

**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 quali accortezze deve avere il programmatore per un’app che utilizza i sensori.**

Un vantaggio è che avendo una frequenza di campionamento maggiore abbiamo dei dati più precisi, ma allo stesso tempo abbiamo un maggior consumo della batteria.

Le accortezze che il programmatore deve utilizzare sono:  
il programmatore deve controllare se il dispositivo che utilizza l’applicazione possiede i sensori che l’applicazione richiede, inoltre, i listener che andranno a gestire i cambiamenti di stato dei sensori, devono essere registrati e rilasciati nei rispettivi metodi onResume e onPause per evitare di consumare la batteria.

**10 – Che cos’è un Toast customizzato? Si spieghi come implementare un Toast customizzato.**

Un toast customizzato è un breve messaggio temporaneo personalizzato che serve al programmatore per dare un feedback a colui che andrà ad utilizzare l’applicazione.

Public void showCustomToast(View v) {  
 Toast toast = new Toast(getApplicationContext());  
 toast.setGravity(Gravity.CENTER\_VERTICAL, 0 , 0);  
 toast.setDuration(Toast.LENGTH\_LONG);  
 toast.setView(  
 getLayoutInflater().inflate(R.layout.custom\_toast, null));  
 toast.show();  
}