**Prova scritta del 06/2015**

**1 – 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.**

L’unita dpi dipende dal rapporto tra risoluzione e dimensione dello schermo. Si tratta quindi del numero di pixel presenti in un pollice. Ad esempio, pensiamo a due display di eguale grandezza, il primo con risoluzione 320\*480 pixel e il secondo 480\*800 pixel; il primo avrà un numero di DPI inferiori al secondo.  
I dp nascono per semplificare la vita agli sviluppatori, in quanto permette di mantenere le proporzioni delle interfacce e non solo, ad esempio:  
prendiamo un’icona di grandezza 50 pixel, e proviamo a inserirla prima su un display con 160 DPI e in seguito su un display di 240 DPI. Nel primo caso l’icona sarà più grande mentre nel secondo più piccola. Questo avviene perché lo schermo con maggiore densità avrà pixel più ravvicinati. Per ovviare a questo problema ed ottenere delle interfacce che siano uguali su tutti i device, si utilizzano i DP, come dice il nome sono “indipendenti”. Questo significa che: 1 DP è uguale ad 1 pixel su uno schermo di 160 DPI. Per tutte le altre densità i pixel verranno scalati seguendo questa formula: 1 pixel = dp \* (dpi / 160); In questa maniera sarà possibile non riscrivere applicazioni per tutti i tipi di device esistenti.  
Questo però non vale per le immagini, le quali non scalano automaticamente.

**2 – Quali sono i widget standard per la costruzione dell’interfaccia utente? E quali sono i principali layout per il posizionamento dei widget? Per ognuno dei layout si descriva brevemente il meccanismo di posizionamento dei widget all’interno del layout.**

I widget standard sono: TextView, pulsanti (normali, toggle, radio), ImageView, input test di vario tipo e widget avanzati (come time picker, media player).

I layout di uso più comune in Android sono:  
- LinearLayout: contiene un insieme di elementi che distribuisce in maniera sequenziale dall’alto verso il basso (se definito con orientamento verticale) o da sinistra a destra (se ha orientamento orizzontale). È un layout molto semplice e piuttosto naturale per i display di smartphone e tablet.  
- GridLayout: altro layout piuttosto semplice, inquadra gli elementi in una tabella e quindi è particolarmente adatto a mostrare strutture regolari suddivise in righe e colonne come form o griglie. Ricorda molto le tabelle HTML nelle pagine web.  
- RelativeLayout: sicuramente il più flessibile e moderno. Adatto a disporre in maniera meno strutturata gli elementi. Adatto a disporre in maniera meno strutturata gli elementi. Gli elementi si posizionano in relazione l’uno all’altro o rispetto al loro contenitore, permettendo un layout fluido che si adatta bene a display diversi. Rispetto ai due precedenti è ricco di attribuiti XML che servono ad allineare e posizionare gli elementi tra loro.  
- ConstraintLayout: permette di specificare la posizione attraverso dei “vincoli” che legano la posizione del nuovo oggetto rispetto a quelli esistenti. Comoda quando si lavora con l’editor grafico.

**3 – Il metodo onCreate prende come parametro un oggetto di tipo Bundle. Perché? Si faccia un esempio di utilizzo di tale oggetto indicando dove (quando) tale oggetto viene scritto.  
@Override  
protected void onCreate(Bundle savedInstanceState) {**

**….**

**}**Quando l’utente ruota il dispositivo, l’attività viene prima eliminata e poi ricreata. L’eliminazione dell’attività, in particolare il metodo onDestroy(), comporta la perdita dello stato, per questo motivo, per evitare la perdita dello stato, si sovrascrive il metodo onSaveInstanceState(Bundle savedInstanceState) nel quale salviamo lo stato dell’app nell’oggetto bundle con una serie di coppie chiave – valore, un esempio può essere:  
savedInstanceState.putInt(“contatore”, counter);  
Una volta inseriti i vari valori all’interno dell’oggetto Bundle, viene chiamato il metodo onSaveInstanceState della superclasse:  
super.onSaveInstanceState(savedInstanceState);  
Lo stato viene recuperato nel metodo onCreate() in questo modo:  
if(savedInstanceState != null) {  
 counter = savedInstanceState.getInt(“contatore”);  
}

**4 – Che cos’è il backstack delle attività? Si faccia un esempio per spiegare il suo funzionamento.**

Un’applicazione normalmente è fatta di più activity. Un’activity può lanciare un’altra activity, anche se quest’ultima appartiene ad un’altra applicazione.  
Se non viene espresso il contrario, ogni qualvolta viene lanciata una nuova activity, quella precedente viene inserita in un backstack. Così facendo, l’utente, cliccando il pulsante back, potrà tornare all’activity precedente.  
Ad esempio, l’utente avvia l’activity A.  
L’activity A lancia l’activity B, e quindi, l’activity A viene messa in stato di stopped e viene inserita nel backstack, e l’activity B viene resa visibile.  
L’activity B lancia l’activity C, e quindi l’activity B viene messa in stato di stopped e viene inserita nel backstack, e l’activity C viene resa visibile.  
L’utente preme il tasto di back, e quindi l’activity C viene distrutta e l’activity B viene riportato nello stato di resumed.

**5 – Si parli della classe Intent. Si spieghi cosa è una risoluzione esplicita e cosa è una risoluzione implicita dell’intent.**

Un componente può attivarne un altro grazie ad apposite invocazioni di sistema che viene codifica con un Intent. Gli intent rappresentano una forma di messaggistica del sistema operativo con cui una componente può richiedere l’esecuzione di un’azione da parte di un’altra componente. Sono uno strumento molto duttile anche se gli utilizzi più comuni ricadono in queste tre casistiche:  
- avviare un’activity.  
- avviare un service.  
- inviare un messaggio in broadcast che può essere ricevuto da ogni applicazione.  
Inoltre gli intent possono essere:  
- espliciti: viene dichiarato quale componente dovrà essere attivato. Particolarmente utili nell’apertura di una nuova activity.  
- impliciti: non specificano una componente da attivare ma quale azione deve essere svolta. La loro invocazione si estrinseca spesso nell’apertura di una finestra di dialogo che chiede all’utente quale app vuole si apra per completare l’azione.

**6 – Completare il seguente codice per far in modo che quando l’utente clicca su un elemento della lista il testo dell’elemento cliccato compaia nel TextView outputTextView.**

@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
  
 String[] array = {"Uno", "Due", "Tre", "Quattro", "Cinque", "Sei", "Sette", "Otto", "Nove"};  
  
 listView = (ListView) findViewById(R.id.mylistview);  
 outputTextView = (TextView) findViewById(R.id.outputTextView);  
 ArrayAdapter <String> arrayAdapter =  
 new ArrayAdapter <String> (this, R.layout.list\_element, R.id.textViewList, array);  
  
 listView.setAdapter(arrayAdapter);  
 listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
  
 @Override  
 public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
  
 String n = listView.getItemAtPosition(position).toString();  
 outputTextView.setText(n);  
  
 }  
 });  
  
}

**7 – Si descriva il funzionamento di AsyncTask.execute, spiegando in che modo i metodi  
1. onPreExecute()  
2. doInBackground(Params… params)  
3. onProgressUpdate(Progress… values)  
4. onPostExecute(Result result)  
interagiscono fra loro e con l’oggetto chiamante.**

La classe Java generica è: class AsyncTask <Params, Progress, Result> { … }  
I vari parametri indicano:  
- Params specifica il tipo (di dati) per il lavoro che deve svolgere il background thread,  
- Progress specifica il tipo (di dato) usato per lo stato di avanzamento,  
- Result specifica il tipo (di dato) per il risultato del task.  
Il main thread esegue i vari metodi per comunicare con il background thread:  
- onPreExecute, che inizializza le operazioni prima che avvenga l’esecuzione del thread background,  
- onPostExecute, che esegue le operazioni da svolgere al termine del thread background,  
- onProgressUpdate, che viene invocato ogni volta che il background thread comunica il suo progresso al thread main.  
Il background thread esegue le sue operazioni all’interno del metodo doInBackground. Ogni qualvolta vuole comunicare con il thread main per notificarlo del suo progresso, viene chiamato il metodo publichProgress, che chiamerà l’esecuzione del metodo onProgressUpdate all’interno del thread main.

**8 – Si descrivano le varie possibilità offerte da Android per la memorizzazione statica (file) dei dati.**

Ogni app ha a disposizione un suo spazio disco, detto Storage interno che risiede in una parte del filesystem e a cui solo l’applicazione dovrebbe accedere. Per accedere allo storage interno si usano per lo più due metodi ovvero openFileInput e openFileOutput.  
- openFileInput apre un file in lettura. Come parametro in input viene passato il nome del file. Non serve specificare il percorso in quanto sarà obbligatoriamente quello messo a disposizione dallo storage interno.  
- openFileOutput apre uno stream in scrittura anche questo nello storage interno. Per il nome del file valo quanto detto per l’input. Come secondo parametro il metodo prende un intero che sta ad indicare la modalità di apertura dello stream. Si può impostare come modalità, ad esempio, la costante Context.MODE\_APPEND per concatenare i nuovi contenuti a quelli già esistenti nel file.  
L’applicazione inoltre può salvare anche file al di fuori dello Storage interno. Per lo Storage esterno Android sceglie una posizione in cui collocare tutto un insieme di risorse che sono di utilità e consultazione generale nel dispositivo (musica, suonerie, e così via).  
Le operazioni per lo Storage esterno avvengono mediante la classe Environment.  
La prima operazione da svolgere è controllare lo stato del supporto. Lo si fa con il metodo statico String Environment.getExternalStorageState();  
La stringa restituita può avere una molteplicità di valori, tutti associati a costanti della classe Environment, ad esempio Environment.MEDIA\_MOUNTED che sta a indicare che il supporto è disponibile in lettura e in scrittura.  
Una volta controllato lo stato del supporto e del relativo filesystem, è arrivato il momento di lavorarci direttamente. L’accesso alla cartella root dello Storage esterno primario si ottiene con il metodo statico File getExternalStorageDirectory(); È consigliabile salvare file direttamente nella cartella principale dello storage esterno, normalmente esso contiene delle cartelle associate alle principali tipologie di contenuti.

**9 – Cosa descriva il seguente file XML? Dare una descrizione dell’effetto risultante.**

<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android"  
 android:shareInterpolator="false">  
  
 <translate  
 android:startOffset="0"  
 android:duration="1500"  
 android:fromXDelta="0"  
 android:fromYDelta="0"  
 android:interpolator="@android:anim/linear\_interpolator"  
 android:toXDelta="150"  
 android:toYDelta="150" />  
  
 <scale  
 android:startOffset="1500"  
 android:duration="3000"  
 android:fromXScale="1"  
 android:fromYScale="1"  
 android:toXScale="0.1"  
 android:toYScale="0.1"  
 android:pivotX="50%"  
 android:pivotY="50%" />  
  
 <rotate  
 android:startOffset="1500"  
 android:duration="3000"  
 android:fromDegrees="0"  
 android:toDegrees="720"  
 android:interpolator="@android:anim/accelerate\_interpolator"  
 android:pivotX="50%"  
 android:pivotY="50%" />  
  
</set>

Nei primi 1,5 secondi l’immagine viene spostata di 150 pixel in basso e 150 pixel a destra contemporaneamente, in modo da formare uno spostamento obbliquo. L’animazione avviene in modo lineare, quindi sempre con la stessa velocità.  
Nei successivi 3 secondi, l’immagine viene scalata, del 90% della sua dimensione mentre ruota su sé stessa per 3 volte. Durante questi 3 secondi l’immagine torna nella sua posizione originale. Le 3 rotazioni non avvengono con la stessa velocità, durante i 3 secondi in cui l’immagine ruota, si ha un effetto di accelerazione, quindi la prima rotazione sarà più lenta rispetto alle altre due.

**10 – Il seguente snippet di codice fa partire un’animazione e poi rimuove l’oggetto dalla view parent (gli oggetti image, animazione 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?**