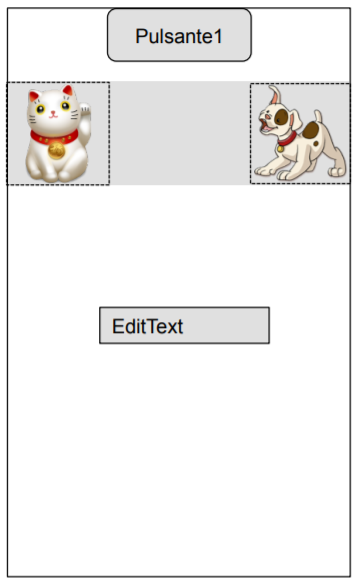
**Prova scritta del 04/2015**

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

Un progetto Android è organizzato nel modo seguente:  
- Manifesto: informazioni generali sull’applicazione, ovvero i permessi, le activity che compongono l’app, il nome dell’app e la sua icona, la versione minima di SDK e altre informazioni generali sull’app.  
- Java: tutti i file sorgenti che compongono l’applicazione.  
- Res, risorse: in questa cartella vi saranno tutte le risorse che compongono la nostra applicazione divise in sottocartelle, come ad esempio la sottocartella layout, nel quale vi sono tutti i file .xml che descrivono in modo statico l’UI dell’activity, la sottocartella drawable, values dove vi saranno le dichiarazioni di alcuni valori che possono essere richiamati nei file sorgenti, ad esempio possiamo salvarci dei valori numerici che possiamo utilizzare come dimensioni standard per alcuni widget, la sottocartella menu, dove vi saranno file xml che descrivono la struttura dei menu che sono implementati all’interno dell’applicazione, etc…

**2 – Si completi il testoXML che descrive l’interfaccia utente specificata nel disegno a destra. Il RelativeLayout già presente si riferisce all’intero schermo.**

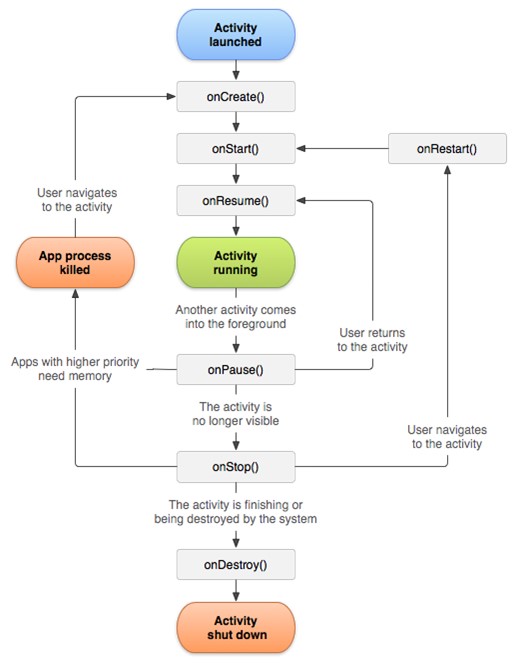
****

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Pulsante1"  
 android:layout\_centerHorizontal="true"  
 android:id="@+id/Pulsante1"/>  
  
 <ImageView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:src="@drawable/faceplaceholder"  
 android:layout\_below="@id/Pulsante1" />  
  
 <ImageView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:src="@drawable/faceplaceholder"  
 android:layout\_below="@id/Pulsante1"  
 android:layout\_alignParentRight="true"/>  
  
 <EditText  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="EditText"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true" />  
   
</RelativeLayout>

**3 – Android prevede un meccanismo per il supporto di lingue diverse che rende molto facile cambiare la lingua utilizzata per i messaggi sullo schermo. Come funziona tale meccanismo?**

Stringe, array, e tanti altri tipi di dati che utilizziamo all’interno dell’applicazione possono essere dichiarate all’interno di una risorsa XML nella cartella res/values/.  
In questo modo possiamo “mappare” qualsiasi stringa che compare nell’applicazione all’interno delle risorse res/values. La cartella values identifica la lingua di default dell’applicazione. Quando un’applicazione viene avviata, la piattaforma Android individua la lingua con cui è impostato il dispositivo ed in base ad essa carica la cartella values-xx dove xx è il suffisso della lingua. Se ad esempio in un’applicazione abbiamo definito una serie di stringhe all’intero della risorsa res/values/string.xml e vogliamo avere il supporto per la lingua italiana, inglese e francese, basta tradurre le stringhe di questo file e inserirle rispettivamente nelle sottocartelle:  
- res/values/strings.xml: default (ad esempio inglese),  
- res/values-**it**/strings.xml: lingua italiana,  
- res/values-**fr**/strings.xml: lingua francese,  
- res/values-**xx**/strings.xml: la lingua corrispondente al suffisso xx.  
Se non viene individuata la sottocartella values-xx relativa alla lingua del dispositivo, allora viene caricata la cartella di default **values.**

**4 – Si disegni il ciclo di vita delle attività. Si descriva una situazione in cui vengono chiamati in sequenza i metodi: onPause, onStop, onDestroy, onCreate, onStart e onResume. Si descriva una situazione in cui vengono chiamati in sequenza i metodi onPause, onStop, onRestart, onStart, onResume.**

****

Una situazione che chiama in sequenza i metodi: onPause, onStop, onDestroy, onCreate, onStart e onResume è una semplice rotazione del dispositivo in modo da modificare il layout visualizzato da orizzontale a verticale o viceversa.

Una situazione in cui vengono chiamati in sequenza i metodi onPause, onStop, onRestart, onStart, onResume è quando dall’applicazione corrente passiamo o ad un’altra applicazione, o torniamo semplicemente all’home screen, senza però chiudere l’applicazione (quindi senza distruggere l’activity).

**5 – Si supponga che un’attività ActivityA debba lanciare un’altra attività ActivityB e che debba passarle come input un intero e una stringa e che inoltre l’attività ActivityB debba restituire un intero come risultato. Mostrare gli snippet di codice in Activity A e in Activity B (indicando anche dove tale snippet debbano essere posizionati) che permettano l’interazione descritta.**

public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 }  
  
 public void pulsanteCliccato(View v) {  
 Intent i = new Intent();  
 i.setClass(getApplicationContext(), Activity2.class);  
 i.putExtra("stringa", "CiaoMondo");  
 i.putExtra("intero", 1);  
 startActivityForResult(i, 0);  
 }  
  
 @Override  
 protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
 if(requestCode != 0) return;  
 if(resultCode != Activity.*RESULT\_OK*) return;  
 if(data == null) return;  
 TextView textView = (TextView) findViewById(R.id.*result*);  
 textView.setText(data.getStringExtra("result"));  
 }  
  
}

public class Activity2 extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity2*);  
 Intent i = getIntent();  
 String string = i.getStringExtra("stringa");  
 int intero = i.getIntExtra("intero", 0);  
 Intent data = new Intent();  
 data.putExtra("result", string + intero);  
 setResult(*RESULT\_OK*, data);  
 onBackPressed();  
 }  
  
}

**6 – Completare il seguente codice per fare 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 – A cosa serve e come funziona la classe *class AsyncTask*?**

Poiché i thread secondari NON possono assolutamente modificare l’interfaccia utente dato che quest’ultima è riservata al thread main, è necessario quindi che i thread di background possano comunicare il loro stato al thread main così da poter modificare l’interfaccia grafica.  
La classe AsyncTask facilita l’interazione fra background thread e main thread. Il background thread esegue il task e notifica il thread main (UI) sul suo stato di avanzamento. Il main thread, usa i risultati forniti dal background thread e li mostra sul display.  
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 metodi 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 – A cosa servono le classi DatabaseOpenHelper e Cursor?**

La classe SQLiteOpenHelper serve a gestire la nascita e l’aggiornamento del database su memoria fisica e a recuperare un riferimento all’oggetto SQLiteDatabase usato come accesso ai dati.

La classe Cursor rappresenta un puntatore ad un set di risultati della query. Somiglia a quell’elemento che in altre tecnologie prende il nome di RecordSet o ResultSet. Un oggetto Cursor può essere spostato per puntare ad una riga differente del set di risultati. Ciò viene fatto con i metodi moveToNext, moveToFirst, moveToLast e così via. Una volta che il cursore ha raggiunto la riga desiderata si può passare alla lettura dei dati con metodi specifici in base al tipo di dato (getString, getLong, etc…) indicando il nome del campo. Ad esempio:  
Cursor.getString(Cursor.getColumnIndex(DatabaseStrings.FIELD\_SUBJ);

**9 – Si descriva il meccanismo di layout di Android, spiegando la fase di misurazione e la fase di posizionamento.**

La classe view rappresenta l’elemento di base per i componenti dell’interfaccia utente. Una view occupa un’area rettangolare sullo schermo ed è responsabile del disegno e della gestione degli eventi.  
View è la classe base per i widget, che vengono utilizzati per creare componenti dell’interfaccia utente interattiva.  
La sottoclasse ViewGroup è la classe base per i layout, che sono contenitori invisibili che contengono altre view (o altre ViewGroup) e definiscono le proprietà di layout.  
Una view possiede due coppie di valori di larghezza e altezza. La prima coppia è nota come larghezza misurata e altezza misurata. Queste dimensioni definiscono la dimensione di una vista all’interno del suo genitore. Le dimensioni misurate possono essere ottenute chiamando getMeasuredWidth() e getMeasuredHeight(). La seconda coppia è semplicemente nota come larghezza e altezza, o talvolta larghezza del disegno e altezza del disegno. Queste dimensioni definiscono la dimensione effettiva della view sullo schermo, al momento del disegno e dopo il layout. Questi valori possono, ma non devono, essere diversi dalla larghezza e dall’altezza misurate. La larghezza e l’altezza possono essere ottenute chiamando getWidth() e getHeight(). Per misurare le sue dimensioni, una vista tiene conto del suo riempimento.  
Durante il processo di layout avviene la fase di misura e la fase di posizionamento.  
La fase di misurazione è implementata in onMeasure(int, int) ed è un attraversamento dall’alto verso il basso dell’albero della view. Ogni view spinge le specifiche delle dimensioni lungo l’albero durante la ricorsione. Alla fine della fase di misurazione, ogni view ha memorizzato le sue misure. Il secondo passaggio avviene nel metodo onLayout(int, int, int, int) ed è anch’esso top-down. Durante questo passaggio ogni genitore è responsabile del posizionamento di tutti i suoi figli utilizzando le dimensioni calcolate nel passaggio di misurazione.

**10 – Che cosa è un Toast? Che si deve fare per creare un Toast customizzato?**

Un toast fornisce un semplice **feedback** a proposito di un’operazione in n piccolo popup. Occupa solo la quantità di spazio richiesta dal messaggio e L’activity corrente rimane visibile ed interattiva. Svaniscono automaticamente dopo un time-out. Si crea attraverso il metodo statico Toast.makeText() che prende tre parametri (context, messaggio e durata) e si mostra schermo attraverso il metodo show().