UNIVERSITÀ DI SALERNO

Corso di laurea in Informatica

Mobile Programming

Prof. De Prisco

Prova scritta del

esempio 8

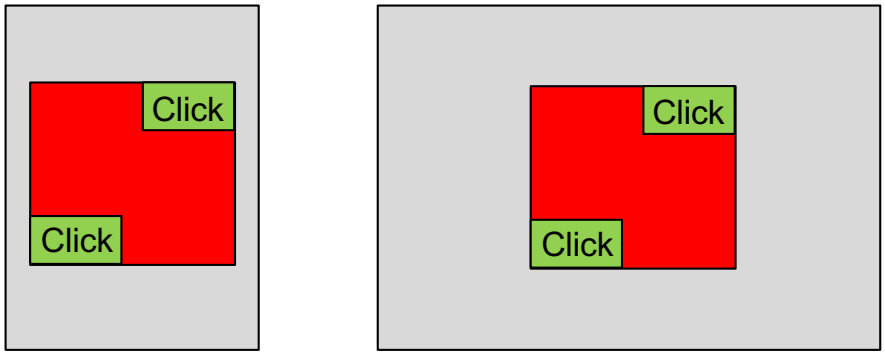
NOME: _____

COGNOME:_____

MATRICOLA: _____

Domande	Punti
1	/10
2	/10
3	/10
4	/10
5	/10
6	/10
7	/10
8	/10
9	/10
10	/10
TOTALE	/100

Si scriva il codice XML che implementa il layout riportato in figura, per uno smartphone in portrait e per un tablet in landscape. Il codice deve essere uno solo e deve implementare il layout riportato per entrambi i casi. Il frame interno è un quadrato di 200dp. I pulsanti invece hanno la dimensione che serve a contenere il testo.




```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">


   <RelativeLayout
      android:layout_width="200dp"
      android:layout_height="200dp"
      android:background="#FF0000"
      android:layout_centerInParent="true">

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Click"
         android:layout_alignParentTop="true"
         android:layout_alignParentLeft="true"/>

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Click"
         android:layout_alignParentBottom="true"
         android:layout_alignParentRight="true"/>

   </RelativeLayout>

</RelativeLayout>
```

Che cosa è un ListView? Che cosa è un Adapter? In che modo ListView e Adapter interagiscono?

In Android, un ListView è un widget che consente di visualizzare una serie di elementi all'interno di una lista. Gli elementi della lista possono essere visualizzati in un layout predefinito (come una riga di testo) oppure in un layout personalizzato che si può creare.

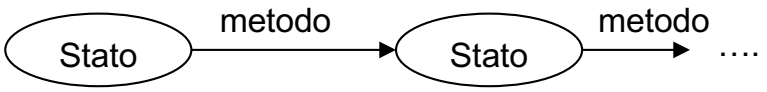
Un Adapter è un componente che serve a collegare i dati da visualizzare all'interno di un ListView. L'adapter si occupa di prendere i dati, creare gli elementi visuali per la lista (ad esempio, viste View per ogni riga della lista) e infine associare queste viste al ListView.

L'interazione tra un ListView e un Adapter avviene nel seguente modo:

Il ListView crea un'istanza dell'Adapter e gli passa i dati da visualizzare nella lista
L'Adapter utilizza questi dati per creare gli elementi visuali per ogni riga della lista (ad esempio, viste View)
L'Adapter notifica al ListView che sono state create le viste e che possono essere visualizzate nella lista
Il ListView visualizza gli elementi della lista utilizzando le viste create dall'Adapter
In questo modo, l'Adapter gestisce i dati e la creazione delle viste per il ListView, mentre il ListView si occupa di visualizzare gli elementi della lista all'interno dell'interfaccia utente.

Si consideri la seguente situazione: un'app viene lanciata e l'utente interagisce con l'app (questa interazione non fa cambiare stato); ad un certo punto l'utente ruota il dispositivo e continua ad interagire con l'app (anche questa interazione non fa cambiare stato); infine l'utente chiude l'app tramite un apposito pulsante previsto dall'app stessa la cui pressione fa chiamare la funzione `finish()`. In quali stati è passata l'app e quali metodi sono stati chiamati dal momento in cui l'app viene lanciata al momento in cui termina?

Si risponda usando una sequenza del tipo mostrato in figura:



In questa situazione, l'app passa attraverso i seguenti stati:

"created" o "launched": quando l'app viene lanciata per la prima volta, viene creata una nuova attività e viene chiamato il metodo onCreate() per impostare la configurazione iniziale dell'app.

"running": quando l'utente inizia a interagire con l'app, essa entra nello stato "running". Durante questo stato, l'app è in esecuzione e l'utente può interagire con essa. Nel caso specifico, l'utente interagisce con l'app senza che ciò faccia cambiare stato.

"paused": quando l'utente ruota il dispositivo, l'app viene messa in pausa e il metodo onPause() viene chiamato. In questo stato, l'app non riceve input dell'utente e non viene eseguita alcuna attività di elaborazione, ma i suoi dati e la configurazione attuali vengono mantenuti in memoria.

"stopped": quando l'utente chiude l'app tramite il pulsante di chiusura, l'app viene fermata e il metodo finish() viene chiamato, che termina l'attività corrente e fa in modo che l'app vada nello stato "stopped". In questo stato, i dati e la configurazione dell'app vengono eliminati dalla memoria, ma possono essere salvati nel caso in cui l'utente decida di riaprire l'app in un secondo momento.

In sintesi:

Quando l'app è lanciata per la prima volta: onCreate()
Quando l'utente interagisce con l'app senza cambiare stato: onResume()
Quando l'utente ruota il dispositivo: onPause()
Quando l'utente chiude l'app tramite il pulsante di chiusura: finish()

Si forniscano degli spezzoni di codice per il lancio di una nuova activity con un Intent esplicito e con un Intent implicito. Si spieghi cosa è necessario per lanciare l'Intent implicito.

Ecco un esempio di codice per lanciare una nuova attività con un Intent esplicito:

Copy code
Intent intent = new Intent(this, NewActivity.class);
startActivity(intent);
In questo caso, l'Intent esplicito specifica esplicitamente l'attività che deve essere avviata, ovvero NewActivity.class.

Ecco un esempio di codice per lanciare una nuova attività con un Intent implicito:

Copy code
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.example.com"));
startActivity(intent);
In questo caso, l'Intent implicito specifica un'azione (in questo caso ACTION_VIEW) e un set di dati (in questo caso un URL) che devono essere utilizzati per determinare quale attività deve essere avviata.

Per lanciare un Intent implicito è necessario specificare l'azione e i dati che devono essere utilizzati per determinare quale attività deve essere avviata

State sviluppando un app “DoppioTris“ che utilizza un frammento Scacchiera che implementa opportunamente una scacchiera del gioco. Il file di layout prevede due contenitori per i due frammenti i cui id sono rispettivamente `idContenitore1` e `idContenitore2`. Completate il metodo onCreate creando i due frammenti Scacchiera e posizionandoli nei rispettivi contenitori.

L’oggetto Scacchiera prevede un metodo setAltra(Scacchiera s) che prende in input una Scacchiera s. Tale metodo serve perché ogni modifica alla scacchiera verrà fatta anche alla scacchiera s. In altre parole se si chiama s1.setAltra(s2), ogni modifica fatta a s1 verrà fatta anche a s2 – questo è codificato nel frammento stesso (non bisogna scrivere codice).

Dopo aver creato e posizionato i frammenti, fare in modo che ognuna delle scacchiere sia la copia dell’altra, quindi quando l’utente modifica una scacchiera viene modificata anche l’altra.

```
public class Scacchiera extends Fragment implements View.OnClickListener {
  //PER QUESTA CLASSE NON BISOGNA SCRIVERE NIENTE
  //occorre solo utilizzarla nel metodo onCreate
}

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //creazione dei frammenti
        Scacchiera scacchiera1 = new Scacchiera();
        Scacchiera scacchiera2 = new Scacchiera();

        //posizionamento dei frammenti nei rispettivi contenitori
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.add(R.id.idContenitore1, scacchiera1);
        ft.add(R.id.idContenitore2, scacchiera2);
        ft.commit();

        //impostazione della relazione tra i due frammenti
        scacchiera1.setAltra(scacchiera2);
        scacchiera2.setAltra(scacchiera1);
        }
        }
```

Con questo codice, quando si crea l'activity, vengono creati anche i due frammenti Scacchiera e poi posizionati nei rispettivi contenitori. In seguito, utilizzando il metodo setAltra(Scacchiera s) per impostare la relazione tra i due frammenti, si garantisce che ogni modifica effettuata su una scacchiera venga riflessa sull'altra.

Descrivere i tipi utilizzati per i parametri nella classe `AsyncTask<Type1,Type2,Type3>`
Quale è il ruolo di `Type1,Type2,Type3`? Fornire un esempio per rendere chiara la risposta

La classe AsyncTask di Android è utilizzata per eseguire operazioni in background e comunicare i risultati con l'interfaccia utente. Essa utilizza tre tipi generics per specificare i tipi dei parametri utilizzati nelle sue varie fasi: Type1, Type2 e Type3.

Type1: rappresenta il tipo del parametro passato all'esecuzione dell'AsyncTask. Questo parametro viene passato alla chiamata del metodo execute(). E' utilizzato per passare i parametri per l'esecuzione del task asincrono.
Esempio :
Copy code

```
new MyTask().execute(URL_to_download);
```

Type2: rappresenta il tipo del parametro utilizzato per comunicare la progressione dell'operazione in background. Questo parametro viene passato al metodo onProgressUpdate() che permette di indicare lo stato di avanzamento dell'operazione.
Esempio :
Copy code

```
private class MyTask extends AsyncTask<String, Integer, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
    int count = urls.length;
    for (int i = 0; i < count; i++) {
        publishProgress((int) ((i / (float) count) * 100));
        // download the image
    }
    return bitmap;
    }
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
}
```

Type3: rappresenta il tipo del parametro che rappresenta il risultato dell'operazione in background. Questo parametro viene passato al metodo onPostExecute() che permette di utilizzare i risultati dell'operazione per l'interfaccia utente
Esempio :
Copy code

```
private class MyTask extends AsyncTask<String, Integer, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
    int count = urls.length;
    for (int i = 0; i < count; i++) {
        publishProgress((int) ((i / (float) count) * 100));
        // download the image
    }
    return bitmap;
    }
    protected void onPostExecute(Bitmap result) {
        setBitmap(result);
    }
}
```

In sintesi Type1 è il tipo dei parametri passati all'inizio del task, Type2 è il tipo dei parametri utilizzati per segnalare l'avanzamento del task e Type3 è il tipo dei parametri passati come risultato del task asincrono.

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?

La misurazione e il posizionamento delle view di un layout avviene in modo simile a come ho descritto in precedenza. Il processo è iniziato dalla view radice del layout e poi ricorsivamente passa attraverso ogni view figlia. Il layout manager utilizza il metodo onMeasure() per determinare le dimensioni richieste da ogni view e il metodo onLayout() per stabilire la posizione esatta sullo schermo.

Il metodo onMeasure() è chiamato su ogni view del layout e viene utilizzato per stabilire le dimensioni richieste dalla view stessa. Il metodo utilizza le proprietà come layout_width e layout_height per determinare le dimensioni desiderate. La misurazione delle dimensioni di ogni view è importante per assicurare che tutte le view siano posizionate correttamente all'interno del layout.

Una volta che le dimensioni sono state stabilite per ogni view, il layout manager chiama il metodo onLayout() per stabilire la posizione esatta sullo schermo. Il metodo onLayout() utilizza le proprietà come left, top, right and bottom per stabilire la posizione.

Per quanto riguarda i metodi getWidth() e getHeight(), essi restituiscono la dimensione corrente della view. Tuttavia, durante la fase di creazione di un'activity, le dimensioni delle view potrebbero non essere ancora state stabilite dal sistema, il che significa che i metodi getWidth() e getHeight() restituiranno 0.
Per questo è consigliabile non utilizzare questi metodi nella fase di creazione dell'activity,ma invece in onStart, onResume, onDraw e simili.

In generale, per avere accesso alle dimensioni corrette delle view, è meglio utilizzare i metodi onLayout, onSizeChanged e simili.

La seguente classe Pentagramma implementa un widget customizzato che permette di inserire delle note in un pentagramma. Le note sono memorizzate con degli oggetti Nota che specificano la posizione nel pentagramma tramite delle coordinate cartesiane. Completare la classe Pentagramma per permettere di inserire nuove note (il costruttore, la funzione `onLayout` e `onDraw` già ci sono anche se non sono riportati i dettagli, quindi non serve aggiungere niente per queste funzioni). Cosa si deve fare, all'esterno di questa classe, dopo avere inserito o cancellato una nota affinché la modifica venga visualizzata?

```
public class Pentagramma extends View {
      ArrayList<Nota> notesList = new ArrayList<Nota>();

      public Pentagramma(Context c, int screen_w, int screen_h) {
         super(c);
         //Costruttore, inizializza l'oggetto
      }

     @Override
          protected void onLayout(boolean b, int x1, int y1, int x2, int y2) {
          //Questa funzione specifica le dimensioni
      }

      @Override
      protected void onDraw(Canvas canvas) {
           //Questa funzione disegna il pentagramma e le note in notesList
      }
```

Per completare la classe Pentagramma e permettere di inserire nuove note, è possibile aggiungere i seguenti metodi:

Copy code

```
public void addNote(Nota note) {
   notesList.add(note);
   invalidate(); //richiamando invalidate() verrà chiamato onDraw() automaticamente per ri-disegnare il pentagramma con la nuova nota
}

public void removeNote(Nota note) {
   notesList.remove(note);
   invalidate(); //richiamando invalidate() verrà chiamato onDraw() automaticamente per ri-disegnare il pentagramma senza la nota cancellata
}
```

Per quanto riguarda ciò che si deve fare, all'esterno di questa classe, dopo avere inserito o cancellato una nota affinché la modifica venga visualizzata, si può utilizzare il metodo addNote per inserire una nuova nota nella lista notesList e il metodo removeNote per cancellare una nota dalla lista. Entrambi questi metodi chiamano invalidate() che richiama automaticamente onDraw() per ri-disegnare il pentagramma con la modifica apportata.

In generale chiamare invalidate() su una view fa in modo che venga chiamato onDraw() automaticamente e così visualizzare l'effetto delle modifiche apportate.

Si scriva un file xml per la seguente animazione di un oggetto drawable:

1. rotazione di 3 giri completi a sinistra, dal tempo 0 al tempo 1 sec
2. traslazione di 200px a destra, dal tempo 1 sec al tempo 2 sec
3. rotazione di 2 giri completi a destra e contemporaneamente traslazione di 200px a sinistra, dal tempo 2 al tempo 3 sec

L'animazione descritta può essere realizzata utilizzando la classe ObjectAnimator insieme a un file XML di animazione. Ecco un esempio di un file XML che implementa l'animazione descritta:

Copy code

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
   android:ordering="sequentially">

   <objectAnimator
      android:propertyName="rotation"
      android:duration="1000"
      android:valueFrom="0"
      android:valueTo="-1080"
      android:repeatCount="1"
      android:repeatMode="restart" />

   <objectAnimator
      android:propertyName="translationX"
      android:duration="1000"
      android:valueFrom="0"
      android:valueTo="200"
      android:startOffset="1000"
      android:repeatCount="1"
      android:repeatMode="restart" />

   <objectAnimator
      android:propertyName="rotation"
      android:duration="1000"
      android:valueFrom="0"
      android:valueTo="720"
      android:startOffset="2000"
      android:repeatCount="1"
      android:repeatMode="restart" />
   <objectAnimator
      android:propertyName="translationX"
      android:duration="1000"
      android:valueFrom="0"
      android:valueTo="-200"
      android:startOffset="2000"
      android:repeatCount="1"
      android:repeatMode="restart" />

</set>
```

Questo file xml utilizza un set che contiene 4 objectAnimator, uno per ogni animazione da fare, che vengono eseguiti in sequenza uno dopo l'altro.
il primo objectAnimator anima la proprietà rotation del drawable per una durata di 1 secondo, facendo una rotazione di 3 giri completi a sinistra.
il secondo objectAnimator anima la proprietà translationX del drawable per una durata di 1 secondo, spostando di 200px verso destra.
il terzo objectAnimator anima la proprietà rotation del drawable per una durata di 1 secondo, facendo una rotazione di 2 giri completi a destra
il quarto objectAnimator anima la proprietà translationX del drawable per una durata di 1 secondo, spostando di 200px verso sinistra.

per utilizzare questo file xml è necessario caricarlo come un oggetto Animation e utilizzarlo per animare il drawable desiderato.

Copy code

```
Animation animation = AnimationUtils.loadAnimation(this, R.anim.animazione_xml);
imageView.startAnimation(animation);
```

Stai sviluppando un'app che permette di visualizzare una serie di scacchiere. Assumi di avere un oggetto `Scacchiera`, che estende la classe `Fragment`, e che implementa tutte le funzionalità della singola scacchiera. Il numero di scacchiere da visualizzare, che indicheremo con N, lo specifica l'utente, quindi non lo conosciamo a priori, ma sappiamo che sarà al massimo 15. Supponendo di avere un file di layout che preveda un numero 15 `FrameLayout`, i cui identificatori sono `frame1`, `frame2`, `frame3`, …., `frame15`, scrivi uno spezzone di codice che permetta di inserire in modo programmatico N frammenti, ognuno dei quali è un oggetto `Scacchiera`, in N contenitori `FrameLayout`, a partire dal primo (`frame1`).

Puoi utilizzare il FragmentManager per inserire i frammenti in modo programmatico, in questo caso l'oggetto Scacchiera, in un FrameLayout. Ecco un esempio di codice che mostra come inserire N frammenti Scacchiera in N FrameLayout in modo programmatico:

Copy code

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

for (int i = 1; i <= N; i++) {
   Scacchiera scacchiera = new Scacchiera();
   fragmentTransaction.add(getResources().getIdentifier("frame"+ i, "id", getPackageName()), scacchiera);
}

fragmentTransaction.commit();
```

In questo codice, utilizziamo getFragmentManager() per ottenere un riferimento al FragmentManager e beginTransaction() per iniziare una transazione. Poi, creiamo un ciclo for che itera fino a N, in cui creiamo un oggetto Scacchiera per ogni iterazione. Utilizziamo getResources().getIdentifier("frame"+ i, "id", getPackageName()) per ottenere l'id del FrameLayout, e quindi utilizziamo il metodo add(int containerViewId, Fragment fragment) dell'oggetto FragmentTransaction per inserire il frammento Scacchiera in questo FrameLayout.
Infine, per completare la transazione, chiamiamo commit().

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva