UNIVERSITÀ DI SALERNO

Corso di laurea in Informatica

Mobile Programming

Prof. De Prisco

Prova scritta del

esempio 4

NOME: _____

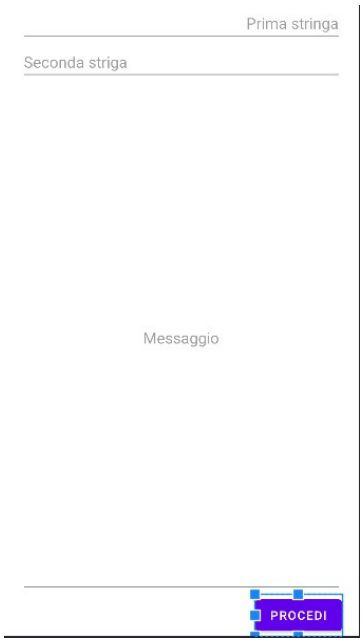
COGNOME:_____

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

Disegnare nell'apposito riquadro, l'interfaccia definita dal seguente file XML. Per gli EditText si utilizzi una liea tratteggiata per indicare l'area occupata.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width=”match_parent"
   android:layout_height=”match_parent"
   android:paddingLeft=”20dp"
   android:paddingRight=”20dp"
   android:orientation="vertical" >
   <EditText
      android:layout_width=”match_parent"
      android:layout_height="wrap_content”
      android:gravity="right"
      android:hint=”Prima stringa" />
   <EditText
      android:layout_width=”match_parent"
      android:layout_height="wrap_content”
      android:layout_gravity="right"
      android:hint=”Seconda striga" />
   <EditText
      android:layout_width=”match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity=”center"
      android:hint=”Messaggio" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
      android:text=”Procedi" />
</LinearLayout>
```



Si consideri un dispositivo A con un display di 5x7cm e densità 1.5 e un dispositivo B con un display di 8x12cm e densità 2 (si assuma che la densità 1 indichi 100 pixel per cm).

Si consideri un'immagine di 300x300pixel. Quanto spazio, misurato in cm, tale immagine occuperà nel display A e quanto spazio nel display B?

Si consideri un'immagine di 254x254dpi. Quanto spazio, misurato in cm, tale immagine occuperà nel display A e quanto spazio nel display B? (Si ricordi che un pollice, inch, equivale 2.54cm)

Densità 1 = 100 pixel per cm
Densità 1.5 = 150 pixel per cm
Densità 2 = 200 pixel per cm

Nel display A l'immagine di 300x300px occuperà uno spazio di 2x2cm (300/150 = 2)
Nel display B l'immagine di 300x300px occuperà uno spazio di 1.5x1.5cm (300/200 = 1.5)

254dpi = 254 * (1/2.54) = 254 * 0.39 = 100px

Nel display A l'immagine di 254x254dpi (100x100px) occuperà uno spazio di 0.66x0.66cm (100/150 = 0.66)
Nel display B l'immagine di 254x254dpi (100x100px) occuperà uno spazio di 0.5x0.5cm (100/200 = 0.5)

Uno stagista di Google ha proposto di semplificare il ciclo di vita delle attività utilizzando un solo metodo per fare passare l'attività da "non esistente" a "in esecuzione" e un solo metodo per passare da "in esecuzione" a "non esistente ("destroyed"). Nel metodo inizioAttività andrebbero eseguite tutte le operazioni che nel sistema attuale si eseguono in onCreate onStart e on Resume, mentre nel metodo fineAttività andrebbero eseguite tutte le operazioni che nel sistema attuale vengono eseguite in onPause, onStop e onDestroy. Quindi al posto di avere la sequenza a sinistra si avrebbe quella a destra.

Attività non esiste

1. onCreate()
2. onStart()
3. onResume()

Attività in esecuzione

4. onPause()
5. onStop()
6. onDestroy()

Attività non esiste

Attività non esiste

1. inizioAttività()

Attività in esecuzione

2. fineAtticità()

Attività non esiste

Questo semplificherebbe il ciclo di vita perché così ci sarebbero meno stati da considerare. Secondo te è una buona idea? Giustificare la risposta.

Questa proposta potrebbe semplificare il ciclo di vita delle attività, ma potrebbero esserci dei problemi nel gestire tutte le operazioni del ciclo di vita in un unico metodo. Ad esempio, le operazioni da eseguire in onResume e onPause potrebbero essere diverse a seconda della situazione, quindi sarebbe difficile gestirle tutte in un unico metodo. Inoltre, il ciclo di vita delle attività è stato progettato in modo tale da consentire una certa flessibilità nella gestione delle operazioni, quindi ridurlo a due metodi potrebbe limitare questa flessibilità.

Inoltre, l'utilizzo di un unico metodo per passare da "non esistente" a "in esecuzione" e un unico metodo per passare da "in esecuzione" a "non esistente" potrebbe rendere il codice meno leggibile e più difficile da comprendere. Inoltre, potrebbe essere più difficile identificare quali operazioni vengono eseguite in ogni fase del ciclo di vita delle attività.

In generale, è importante considerare che il ciclo di vita delle attività è stato progettato in modo tale da garantire una certa flessibilità e facilità di gestione delle operazioni. Ridurre il ciclo di vita a due metodi potrebbe comportare dei problemi a livello di gestione e leggibilità del codice.

Che cosa è o stack delle activity (backstack)? Si descriva il suo funzionamento. In relazione al backstack, le activity ed i frammenti sono trattati allo stesso modo?

Lo stack delle activity (backstack) è una struttura di dati che mantiene traccia delle activity che sono state aperte nell'app e che permette di gestire il loro ciclo di vita. Quando si apre una nuova activity, viene aggiunta allo stack e diventa l'activity corrente. Quando si chiude l'activity corrente, viene rimossa dallo stack e l'activity precedente diventa l'activity corrente.

Il backstack funziona come una pila: l'ultima activity aggiunta è la prima ad essere rimossa. Questo significa che, quando si chiude l'activity corrente, l'activity precedente viene riportata in primo piano.

Le activity ed i frammenti sono trattati allo stesso modo dal backstack, nel senso che entrambi vengono aggiunti allo stack quando vengono aperti e rimossi quando vengono chiusi. Tuttavia, i frammenti devono essere aggiunti manualmente.

Il metodo () della transazione permette di inserire le modifiche apportate ai frammenti nello stack delle attività (backstack). Se chiamiamo () prima di chiamare (), le modifiche apportate ai frammenti verranno memorizzate nello stack delle attività e sarà possibile tornare indietro alle versioni precedenti dei frammenti utilizzando il tasto "indietro" dell'app.
Se non chiamiamo () prima di chiamare (), le modifiche apportate ai frammenti non verranno memorizzate nello stack delle attività. Se l'utente preme il tasto "indietro", verrà visualizzata l'attività precedente senza tenere conto delle modifiche apportate ai frammenti.

Si completi il seguente codice assumendo di avere a disposizione la funzione “partialLoad()” che si occupa di caricare in ogni chiamata un 5% dell’immagine **`img`** (quindi dopo 20chiamate a tale funzione, l’immagine sarà completa). Si renda visibile la ProgressBar all’inizio del caricamento e invisibile alla fine. Si aggiorni la progress bar ad ogni 5% di caricamento e si mostri un Toast di avviso “Caricamento quasi completato” quando si raggiunge l’90% del caricamento. Si mostri l’immagine nell’`imageView` alla fine del caricamento.

```
public class ThreadAsyncTaskActivity extends Activity {
    private ImageView imageView;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_layout);
        imageView = (ImageView)findViewById(R.id.imageView);
        progressBar = (ProgressBar)findViewById(R.id.progressBar);
    }

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

        @Override
        protected void onPreExecute() {
            // Mostra la ProgressBar
            progressBar.setVisibility(View.VISIBLE);
        }

        @Override
        protected Bitmap doInBackground(Integer... ids) {
            Bitmap img = null;
            while (index <= 20) {
                img = partialLoad();
                publishProgress((index * 5));
                index++;
            }
            return img;
        }

        @Override
        protected void onProgressUpdate(Integer... values) {
            // Aggiorna la ProgressBar
            progressBar.setProgress(values[0]);
            // Mostra un Toast di avviso se si è raggiunto il 90% del caricamento
            if (values[0] == 90) {
                Toast.makeText(ThreadAsyncTaskActivity.this, "Caricamento quasi completato", Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        protected void onPostExecute(Bitmap result) {
            // Nascondi la ProgressBar
            progressBar.setVisibility(View.INVISIBLE);
            // Mostra l'immagine nell'imageView
            imageView.setImageBitmap(result);
        }
    }
}
```

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 e e cancellare quelle esistenti (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?

Una nuova nota può essere inserita in una qualsiasi posizione dell'array `notesList`.

Per cancellare una nota occorre passare l'indice dell'array `notesList` in cui la nota è memorizzata.

```
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
      }
```

```
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
   }

   public void addNote(Nota n) {
      // Aggiunge una nuova nota alla lista
      notesList.add(n);
   }

   public void removeNote(int index) {
      // Rimuove la nota all'indice specificato dalla lista
      if (index >= 0 && index < notesList.size()) {
         notesList.remove(index);
      }
   }

   @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 far sì che la modifica venga visualizzata, dopo aver inserito o cancellato una nota, sarà sufficiente chiamare il metodo invalidate() sull'istanza della classe Pentagramma. Questo metodo causerà la chiamata di nuovo del metodo onDraw(), che disegnerà nuovamente il pentagramma con le modifiche apportate.

Stai scrivendo un'app che utlizza 3 activity che devono condividere dei parametri specificati dall'utente come preferenze: la grandezza dei caratteri, misurata in pixels, un indirizzo email, e un colore. Come fai in modo che le 3 activity condividano questi dati? Fornire dei dettagli.

Ci sono diversi modi per condividere i dati tra le activity. Ecco alcune possibili soluzioni:

Usare i SharedPreferences: si può usare questo sistema di memorizzazione per salvare i dati e condividerli tra le diverse activity. I SharedPreferences sono adatti per salvare dati di piccole dimensioni e che devono essere conservati anche quando l'app viene chiusa.

Usare un servizio: si può creare un servizio che gestisca i dati da condividere e fornirne l'accesso alle diverse activity attraverso un'interfaccia di comunicazione, ad esempio una classe AIDL (Android Interface Definition Language). Un servizio è adatto per condividere dati di grandi dimensioni e per gestire operazioni che devono continuare a essere eseguite anche quando l'app non è in primo piano.

Usare un Intent: è possibile passare i dati come extra dell'Intent quando si passa da un'activity all'altra. Tuttavia, questo metodo è adatto solo per passare piccole quantità di dati e non è adatto per conservare i dati quando l'app viene chiusa.

Usare un file: si può salvare i dati in un file e condividerlo tra le activity leggendo e scrivendo il file ogni volta che è necessario accedere ai dati. Tuttavia, questo metodo potrebbe essere inefficiente in termini di prestazioni se si devono gestire grandi quantità di dati.

Usare un database: si può creare un database per memorizzare i dati e condividerli tra le activity utilizzando un provider di contenuti. Un provider di contenuti è un componente di un'app che fornisce una interfaccia per l'accesso ai dati memorizzati in un database. Questo metodo è adatto per condividere grandi quantità di dati e permette anche a più app di accedere ai dati. Tuttavia, richiede una maggiore configurazione e gestione rispetto ad altri metodi.

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`).

```
for (int i = 0; i < N;) {
    i++;
   if (N > 15) break;

   Scacchiera scacchiera = new Scacchiera();
   int frameId = getResources().getIdentifier("frame" + i, "id", getPackageName());
   FrameLayout frameLayout = (FrameLayout) findViewById(frameId);
   getFragmentManager().beginTransaction().add(frameLayout.getId(), scacchiera).commit();
}
```

getIdentifier è un metodo della classe Resources che consente di ottenere l'identificatore di una risorsa a partire dal nome della risorsa e dal nome del tipo di risorsa.

Nel codice che hai fornito, la chiamata a getIdentifier viene utilizzata per ottenere l'identificatore del FrameLayout con nome "frame" + i. Ad esempio, se i vale 1, l'identificatore del FrameLayout sarà frame1.

Il primo parametro di getIdentifier è il nome della risorsa, mentre il secondo parametro è il tipo di risorsa, in questo caso "id". Il terzo parametro è il nome del pacchetto dell'app, che viene ottenuto con getPackageName.

L'identificatore viene quindi assegnato alla variabile frameId.

Che differenza c'è fra permessi normali e permessi pericolosi? Perché alcuni permessi sono raggruppati e quale insidia comporta la gestione dei permessi di gruppo così come viene fatta da Android?

I permessi in Android sono suddivisi in due categorie: permessi normali e permessi pericolosi. I permessi normali sono quelli che non comportano rischi per la sicurezza o la privacy dell'utente, mentre i permessi pericolosi rappresentano un rischio per la sicurezza o la privacy dell'utente se utilizzati in modo improprio.

I permessi pericolosi vengono richiesti all'utente al momento dell'installazione dell'app, mentre i permessi normali vengono concessi automaticamente all'app.

Alcuni permessi sono raggruppati in modo da facilitare la gestione delle autorizzazioni per l'utente. Ad esempio, i permessi di gruppo "Contatti" includono permessi per l'accesso ai contatti dell'utente, mentre il gruppo "SMS" include permessi per l'accesso agli SMS dell'utente.

La gestione dei permessi di gruppo da parte di Android può comportare insidie in quanto l'utente può essere indotto a concedere l'accesso a un intero gruppo di permessi senza rendersi conto che alcuni di essi potrebbero rappresentare un rischio per la sicurezza o la privacy. Ad esempio, l'utente potrebbe concedere l'accesso al gruppo "Contatti" per utilizzare una funzionalità specifica dell'app, senza rendersi conto che alcuni di essi potrebbero rappresentare un rischio per la sicurezza o la privacy

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

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
   <rotate
      android:fromDegrees="0"
      android:toDegrees="1080"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="1000" />
   <translate
      android:fromXDelta="0"
      android:toXDelta="200"
      android:duration="1000"
      android:startOffset="1000" />
   <rotate
      android:fromDegrees="0"
      android:toDegrees="-720"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="1000"
      android:startOffset="2000" />
   <translate
      android:fromXDelta="200"
      android:toXDelta="0"
      android:duration="1000"
      android:startOffset="2000" />
</set>
```

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva