UNIVERSITÀ DI SALERNO

Corso di laurea in Informatica

Mobile Programming

Prof. De Prisco

Prova scritta del

esempio 9

NOME: _____

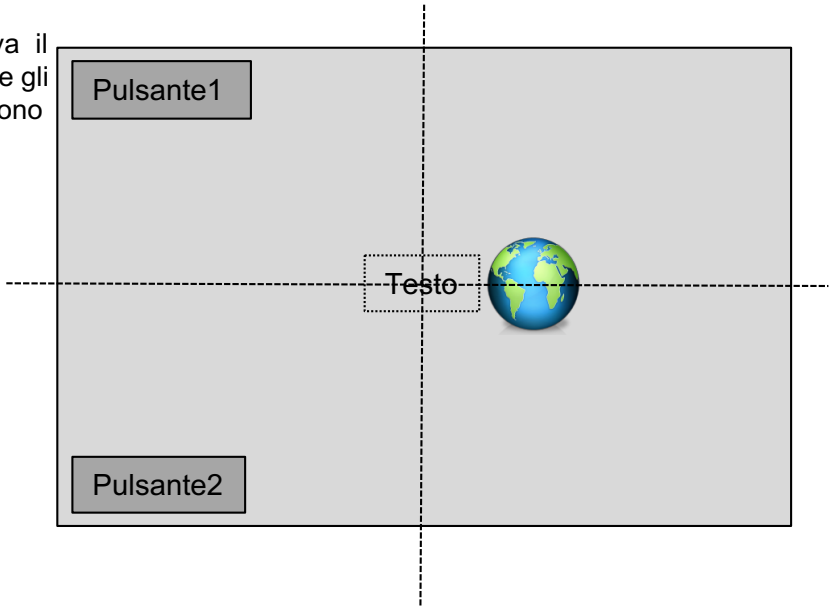
COGNOME:_____

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

Quesito 1

Si scriva un frammento di codice XML che descriva il lavout raffigurato a fianco scegliendo opportunamente gli elementi da utilizzare (le linee punteggiate non sono elementi del layout ma indicano il centro del layout).




```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.
android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/
res-auto"
    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="Pulsante 1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Pulsante 2"
        android:layout_alignParentBottom="true"
        andoird:layout_alignParentLeft="true" />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Testo"
        android:layout_centerInParent="true" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/text"
        android:layout_centerInParent="true"
        android:src="@drawable/ic_launcher_
background" />

</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 un elenco di elementi verticalmente in una schermata. Gli elementi dell'elenco sono solitamente visualizzati con un layout predefinito, ma è possibile personalizzare il layout degli elementi utilizzando un'istanza di una sottoclasse di Adapter.

Un Adapter è un componente che si occupa di creare e gestire gli elementi visualizzati all'interno di un ListView. L'adattatore funziona come un intermediario tra il modello dei dati (che contiene i dati da visualizzare) e la visualizzazione dei dati (il ListView) in modo che l'applicazione non debba gestire direttamente la creazione e l'aggiornamento della visualizzazione dei dati.

Il modo in cui un ListView e un Adapter interagiscono è il seguente:

Si crea un'istanza di un Adapter e si imposta i dati che devono essere visualizzati sull'elenco
si imposta l'adattatore per il ListView tramite setAdapter(adapter)
ListView chiederà all'adattatore di creare visualizzazioni per ogni elemento dell'elenco e di fornire informazioni su come posizionare gli elementi nella visualizzazione.
L'adattatore crea visualizzazioni per ogni elemento dell'elenco e fornisce informazioni su come posizionare gli elementi nella visualizzazione al ListView
Il ListView posiziona gli elementi nella visualizzazione e li visualizza all'utente.
Se ci sono cambiamenti nei dati, l'adattatore notifica al ListView e questo si occuperà di aggiornare la visualizzazione.

Che cosa è il backstack? Supponendo che l'activity A sia l'unica in esecuzione (l'unica presente nel backstack), che la stessa app lancia una nuova activity B che a sua volta lancia l'activity C, che lancia l'activity D, quale è il backstack a questo punto? Cosa succede se dall'activity D si preme il pulsante di "back"?

Cosa si deve fare se si vuole fare in modo che dall'activity D, si torni direttamente ad A quando si preme il pulsante di back?

Il backstack è una pila di activity che tiene traccia dell'ordine in cui le attività vengono lanciate. Quando una nuova attività viene lanciata, viene aggiunta alla cima dello stack. Quando si preme il tasto indietro su un dispositivo Android, l'activity corrente viene terminata e l'activity che si trova immediatamente sotto di essa nello stack viene riportata in primo piano.

In questo caso, si suppone che l'activity A sia l'unica in esecuzione (l'unica presente nello stack), quando l'app lancia una nuova activity B che a sua volta lancia l'activity C, che lancia l'activity D, il backstack a questo punto è composto da A, B, C, D.

Se dall'activity D si preme il pulsante di "indietro", l'activity D viene terminata e l'activity C verrà riportata in primo piano. Quindi, se si continua a premere indietro si tornerà rispettivamente su B,A in sequenza.

Per fare in modo che dall'activity D, si torni direttamente ad A quando si preme il pulsante di back, si può utilizzare il metodo finish() dell'activity D, prima di lanciare la nuova activity A, questo eliminerà dallo stack tutte le attività intermedie (B,C,D) e l'utente tornerà direttamente all'activity A quando premere indietro. Oppure si può utilizzare il metodo popBackStack() dell' FragmentManager per far tornare indietro fino ad un certo punto nello stack e in tal modo saltare tutte le attività intermedie.

La soluzione più semplice è impostare l'attributo noHistorya trueper quello <activity>giorno dopo AndroidManifest.xml:

Un dispositivo Android può funzionare sia in modalità portrait (verticale) landscape (orizzontale). Quando un dispositivo Android viene ruotato si passa dall'una all'altra modalità. Per gestire in maniera appropriata tali passaggi, di cosa si deve preoccupare il programmatore?

Quando un dispositivo Android viene ruotato da modalità portrait (verticale) a landscape (orizzontale) o viceversa, il programmatore deve preoccuparsi di come l'applicazione gestisce questa rotazione per garantire che l'esperienza utente sia ottimale. Ci sono alcune cose che il programmatore deve tenere in considerazione:

Layout: Il programmatore deve assicurarsi che i layout dell'applicazione funzionino correttamente sia in modalità portrait che in modalità landscape. Ad esempio, un layout che funziona bene in modalità portrait potrebbe non adattarsi correttamente alla modalità landscape e viceversa.

Dimensioni e posizioni delle viste: Quando un dispositivo viene ruotato, le dimensioni e le posizioni delle viste possono cambiare, il programmatore deve assicurarsi che le viste siano posizionate e dimensionate in modo appropriato in entrambe le modalità.

Salvataggio e ripristino dello stato: Quando un dispositivo viene ruotato, l'attività corrente viene distrutta e ricreata, il programmatore deve assicurarsi che importanti informazioni dell'applicazione, come ad esempio lo stato dell'interfaccia utente o i dati di input, vengano salvati e ripristinati correttamente in modo che l'utente non perda informazioni importanti durante la rotazione del dispositivo.

Gestione dei fragment: Se l'applicazione utilizza i fragment, il programmatore deve assicurarsi che i fragment vengano gestiti correttamente durante la rotazione del dispositivo, ad esempio, assicurando che i fragment vengano aggiunti o rimossi dalla UI correttamente in base alla modalità.

Gestione delle risorse: Il programmatore deve assicurarsi di utilizzare le risorse appropriate (immagini, stringhe, dimensioni, ecc.) in base alla modalità (portrait o landscape) per evitare di avere problemi di risoluzione o di adattamento dell'interfaccia utente.

Un'activity ActivityA ha bisogno di lanciare un'ActivityB passando a tale activity un intero x ed una stringa s. Cosa si deve fare? Dettagliare la risposta con opportuni frammenti di codice.

Per passare un intero x ed una stringa s da un'activity ActivityA ad un'altra activity ActivityB, è possibile utilizzare il metodo startActivity() e passare i valori come dati extras all'intent che viene utilizzato per avviare l'activity.

Per passare i valori si può utilizzare il metodo putExtra(String key, int value) e putExtra(String key, String value) dell'oggetto Intent.

Ecco un esempio di come passare un intero x ed una stringa s da ActivityA ad ActivityB:

Copy code
```
// Creazione dell'intent per lanciare ActivityB
Intent intent = new Intent(ActivityA.this, ActivityB.class);
// Impostazione dei valori extra
intent.putExtra("x_value", x);
intent.putExtra("s_value", s);
// lanciare ActivityB
startActivity(intent);
```
In ActivityB, si può recuperare i valori passati dall'intent utilizzando i metodi getIntExtra(String key, int defaultValue) e getStringExtra(String key):

Copy code
```
// recupero valore x
int x = getIntent().getIntExtra("x_value", 0);
// recupero valore s
String s = getIntent().getStringExtra("s_value");
```
In questo modo i valori di x e s sono passati dall'ActivityA all'ActivityB e disponibili per essere utilizzati all'interno dell'activity B.

Nota: è anche possibile utilizzare altri metodi di passaggio di parametri come l'uso di Bundle o di un insieme di interfacce da implementare nell'activity richiamante e richiamata.

Il seguente abbozzo di classe NetworkTask mostra nviene utilizzato per ricevere da Internet dei dati nella forma di Stringa. Nella parte di ricezione (non dettagliata) viene eseguito un ciclo per 100 iterazioni e ad ogni itetrazione viene recevuta una stringa str mentre la stringa received_data viene utilizzata per memorizzare tutti i dati ricevuti che alla fine verranno visualizzati nel TextView textResponse. Nel layout c'è una progressBar utilizzata per mostrare in quale iterazione ci troviamo. Questo codice non funziona. Perché? Come possiamo risolvere il problema?

```
class NetworkTask extends AsyncTask<Integer, Integer, String> {

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

  @Override
  protected String doInBackground(Integer...values) {
    //Do something
    String received_data = "";
    for (int i=0; i<100; i++) {
        progressBarDownload.setProgress(i);
        //Ricevi i dati nella stringa str
        received_data += str;
    }
    return received_data;
  }

  @Override
  protected void onPostExecute(String data) {
    progressBarDownload.setVisibility(ProgressBar.INVISIBLE);
    textResponse.setText(data);
  }
}
```

Il codice mostrato non funziona perché il metodo doInBackground(Integer...values) viene eseguito in un thread separato rispetto al thread principale dell'applicazione, quindi non è possibile accedere e modificare la UI, come ad esempio la ProgressBar e il TextView, dal thread doInBackground(...).

Per risolvere questo problema, è possibile utilizzare il metodo publishProgress(Progress...) che permette di passare informazioni dal thread di background al thread principale per aggiornare la UI.

Ecco un esempio di come utilizzare il metodo publishProgress(Progress...) per risolvere il problema:

Copy code

```
class NetworkTask extends AsyncTask<Integer, Integer, String> {
  @Override
  protected void onPreExecute() {
    progressBarDownload.setVisibility(ProgressBar.VISIBLE);
  }
  @Override
  protected String doInBackground(Integer...values) {
    //Do something
    String received_data = "";
    for (int i=0; i<100; i++) {
      //Ricevi i dati nella stringa str
      received_data += str;
      publishProgress(i);
    }
    return received_data;
  }
  @Override
  protected void onProgressUpdate(Integer... values) {
    int progress = values[0];
    progressBarDownload.setProgress(progress);
  }
  @Override
  protected void onPostExecute(String data) {
    progressBarDownload.setVisibility(ProgressBar.INVISIBLE);
    textResponse.setText(data);
  }
}
```

In questo modo, il metodo onProgressUpdate(Integer... values) viene chiamato automaticamente dal framework AsyncTask quando il metodo publishProgress(Integer... values) viene chiamato dal thread di background, e in questo metodo possiamo aggiornare la ProgressBar con il valore corrente del ciclo. Inoltre, il metodo onPostExecute(String data) verrà chiamato una volta completato il lavoro in background e potremmo visualizzare i dati ricevuti nel TextView.

Il seguente snippet di codice esegue prima un'animazione e poi rimuove l'oggetto dalla view parent (gli oggetti image, animation 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?

Il codice mostrato esegue prima l'animazione sull'immagine e poi rimuove immediatamente l'oggetto dalla view parent, causando che l'immagine venga rimossa prima che l'animazione abbia il tempo di essere eseguita.

Questo accade perché il metodo startAnimation(animation) inizia solo l'animazione, ma non la attende perché il suo esecuzione continua immediatamente dopo l'avvio dell'animazione.

Per ovviare al problema, è possibile utilizzare una animazione di tipo AnimationListener che permette di essere avvisati quando l'animazione è completata. In questo modo è possibile rimuovere l'immagine solo dopo che l'animazione è stata completata.

Copy code

```
animation.setAnimationListener(new Animation.AnimationListener() {
    @Override
    public void onAnimationStart(Animation animation) {
    }
    @Override
    public void onAnimationEnd(Animation animation) {
        parentView.removeViewAt(0);
    }
    @Override
    public void onAnimationRepeat(Animation animation) {
    }
});
image.startAnimation(animation);
```

In questo modo, il metodo onAnimationEnd(Animation animation) verrà chiamato una volta che l'animazione è completata e solo in quel momento si rimuoverà l'immagine dalla view parent, garantendo così che l'animazione sia completamente visibile all'utente.

Per cosa sono utili i Frammenti?

I frammenti sono utili per diversi scopi, in particolare per la creazione di interfacce utente più flessibili e riutilizzabili. Alcuni dei motivi per cui i frammenti sono utili sono:

Gestione della UI su più schermi: I frammenti possono essere utilizzati per creare interfacce utente più complesse che possono essere utilizzate su più schermi, come ad esempio su dispositivi tablet o smartphone con diversi formati di schermo. Ciò consente di creare interfacce utente che si adattano automaticamente al formato dello schermo e alla disposizione dei contenuti.

Riutilizzo della logica dell'interfaccia utente: I frammenti possono essere riutilizzati all'interno di diverse attività, consentendo di riutilizzare la logica dell'interfaccia utente senza doverla ripetere ogni volta. Ciò consente di aumentare la manutenibilità del codice e di ridurre la quantità di codice duplicato.

Comunicazione tra frammenti: I frammenti possono comunicare tra loro tramite un'interfaccia o tramite l'attività che li contiene, questo consente di creare interfacce utente più complesse e interattive.

Gestione della vita ciclo dei frammenti: I frammenti hanno un loro ciclo di vita che può essere gestito separatamente rispetto all'attività che li contiene, ciò consente di gestire in maniera più precisa l'esecuzione del codice in background e il ricaricamento dei dati.

Gestione dei fragment con la Navigazione: Utilizzando i frammenti in congiunzione con la navigazione, permette di creare esperienze utente più fluide e intuitive, gestendo i fragment come stack di navigazione, permette di tornare indietro tra le varie schermate e di mantenere una cert a coerenza nell'esperienza utente. Inoltre, utilizzando la navigazione dei frammenti si può gestire in modo efficiente la transizione tra le diverse schermate dell'applicazione, gestendo automaticamente la creazione e la distruzione dei frammenti e il salvataggio e il ripristino dello stato, rendendo la creazione di interfacce utente più semplice e organizzata.

Si descriva l'utilizzo di un sensore spiegando cosa si deve fare per utilizzarlo. Arricchire la spiegazione con frammenti di codice.

Per utilizzare un sensore su un dispositivo Android, è necessario seguire i seguenti passi:

Recuperare un riferimento al SensorManager utilizzando il metodo getSystemService(SENSOR_SERVICE) dell'activity o del fragment.
Copy code
```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```
Recuperare un riferimento al sensore desiderato utilizzando il metodo getDefaultSensor(int type) del SensorManager passando il tipo di sensore desiderato. Ad esempio, per recuperare un riferimento al sensore di accelerazione si può utilizzare:
Copy code
```
Sensor accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```
Creare un'istanza di una classe che implementi l'interfaccia SensorEventListener per gestire i dati del sensore.
Copy code
```
class MySensorEventListener implements SensorEventListener {
    @Override
    public void onSensorChanged(SensorEvent event) {
        // codice per gestire i dati del sensore
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // codice per gestire i cambiamenti di accuratezza del sensore
    }
}
```
Registrare il listener creato utilizzando i metodi registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs) dell'oggetto SensorManager, per iniziare a ricevere gli eventi del sensore.
Copy code
```
sensorManager.registerListener(new MySensorEventListener(), accelerometerSensor, SensorManager.SENSOR_DELAY_NORMAL);
```
Utilizzare i dati del sensore all'interno del metodo onSensorChanged(SensorEvent event) del listener creato.

Non dimenticare di de-registrare l'ascoltatore quando non ne hai più bisogno utilizzando il metodo unregisterListener(SensorEventListener listener) dell'oggetto SensorManager, questo ti aiuterà a gestire al meglio le risorse del dispositivo

Copy code
```
sensorManager.unregisterListener(new MySensorEventListener());
```
Nota: i metodi di registrazione e deregistrazione deve essere chiamati sempre sullo stesso thread, generalmente il main thread, per evitare problemi di concorrenza.

In questo modo si può utilizzare un sensore su un dispositivo Android, gestendo i dati del sensore e utilizzandoli all'interno dell'applicazione.

Si descrivano brevemente i Services, i BrodcastReceiver ed i ContentProvider

I Services sono componenti del sistema Android che consentono di eseguire attività in background senza che l'utente debba interagire con l'applicazione. I services possono essere utilizzati per eseguire operazioni di lunga durata, come ad esempio l'elaborazione di dati o la riproduzione di musica. I services possono anche essere utilizzati per comunicare con altri componenti dell'applicazione o con altre applicazioni.

I BroadcastReceiver sono componenti del sistema Android che consentono di ricevere e gestire i messaggi di sistema e le notifiche di altre applicazioni. I broadcast receiver possono essere utilizzati per ricevere notifiche su eventi di sistema, come ad esempio la ricezione di una chiamata o la modifica dello stato della batteria, e possono anche essere utilizzati per ricevere messaggi da altre applicazioni.

I ContentProvider sono componenti del sistema Android che consentono di condividere e gestire i dati tra diverse applicazioni. I content providers possono essere utilizzati per gestire la persistenza dei dati, come ad esempio i contatti o i messaggi, e possono anche essere utilizzati per condividere i dati con altre applicazioni. In generale i content provider sono la chiave per l'integrazione con altre app e per la condivisione dei dati all'interno di un sistema.