UNIVERSITÀ DI SALERNO

Corso di laurea in Informatica

Mobile Programming

Prof. De Prisco

Prova scritta del

esempio 6

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

Assumi che il contenitore radice nel quale devi posizionare dei widget per il layout di un app corrisponda all'intero schermo del dispositivo. Elenca gli attributi cruciali ed i loro valori, che ti servono per ottenere i seguenti risultati:

1. Posizionare 3 widget allineati orizzontalmente in modo tale che occupino tutta la larghezza dello schermo
2. Posizionare un widget al centro dello schermo
3. Posizionare 4 widget negli angoli dello schermo
4. Posizionare 5 widget allineati verticalmente a destra dello schermo

Nei casi in cui è utile puoi fare assunzioni sul tipo di contenitore radice da utilizzare.

Per posizionare 3 widget allineati orizzontalmente in modo tale che occupino tutta la larghezza dello schermo, puoi utilizzare un LinearLayout con l'attributo android:orientation impostato su horizontal e l'attributo android:layout_width impostato su match_parent per i 3 widget:

Copy code

```
<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="horizontal">

   <View
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1" />

   <View
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1" />

   <View
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1" />
</LinearLayout>
```

Per posizionare un widget al centro dello schermo, puoi utilizzare un FrameLayout con l'attributo android:layout_gravity impostato su center per il widget:

Copy code

```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center" />
</FrameLayout>
```

Per posizionare 4 widget negli angoli dello schermo, puoi utilizzare un FrameLayout con gli attributi android:layout_gravity impostati su top|left, top|right, bottom|left, bottom|right per i 4 widget:

Copy code

```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="top|left" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="top|right" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="bottom|left" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="bottom|right" />
</FrameLayout>
```

Per posizionare 5 widget allineati verticalmente a destra dello schermo, puoi utilizzare un LinearLayout con l'attributo android:orientation impostato su vertical e l'attributo android:layout_gravity impostato su right per il LinearLayout:

Copy code

```
<LinearLayout
   android:layout_width="wrap_content"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:layout_gravity="right">

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

   <View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
</LinearLayout>
```

Scrivere il codice per implementare un ListView semplice in modo tale che quando si seleziona un elemento della lista compaia un Toast che specifichi la posizione dell'elemento nell'array. Si assume che la lista dei nomi sia memorizzata nell'array "nomi" e che il layout del singolo elemento della lista sia specificato nel file "listElement.xml", memorrizzato nella cartella "layout", e che il TextView nel quale visualizzare i nomi della lista abbia identificativo "tvListElement".

Se non ricordi bene le istruzioni da utilizzare spiega a parole cosa si deve fare.

Per implementare un ListView con un Toast che mostra la posizione dell'elemento selezionato nell'array, puoi seguire questi passaggi:

Includi il ListView nel layout dell'activity:

```
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Crea un adapter per il ListView, utilizzando un layout personalizzato per gli elementi della lista:

Copy code

```
ArrayAdapter<String> adapter = new ArrayAdapter<>(this, R.layout.list_element, R.id.tvListElement, nomi);
```

Associa l'adapter al ListView:

Copy code

```
ListView listView = findViewById(R.id.listView);
listView.setAdapter(adapter);
```

Imposta un listener per gli elementi della lista:

Copy code

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(getApplicationContext(), "Posizione: " + position, Toast.LENGTH_SHORT).show();
    }
});
```

In questo modo, quando viene selezionato un elemento della lista, verrà visualizzato un Toast che mostra la posizione dell'elemento nell'array.

Consideriamo i metodi onCreate() e il metodo onStart() del ciclo di vita di un'app.

1. È possibile avere una situazione in cui, durante un ciclo di vita dell'app, onCreate() viene chiamato più volte di onStart()? Se sì, quale?
2. È possibile avere una situazione in cui durante un ciclo di vita dell'app, onStart() viene chiamato più volte di onCreate()? Se sì, quale?

Sì, è possibile che onCreate() venga chiamato più volte di onStart() in un ciclo di vita di un'app. Ad esempio, questo può accadere quando l'app viene chiusa e riaperta rapidamente. In questo caso, onCreate() verrà chiamato nuovamente ma onStart() no, poiché l'app è già stata avviata.

No, non è possibile che onStart() venga chiamato più volte di onCreate() durante un ciclo di vita di un'app. onCreate() viene chiamato solo una volta all'avvio dell'app, mentre onStart() viene chiamato ogni volta che l'app viene resa visibile all'utente (ad esempio, quando viene aperta dalla schermata home o quando viene riportata in primo piano da un'altra app).

Si consideri un'ActivityA che lancia un'ActivityB ed ha la necessità di passare come parametri di input per l'ActivityB due valori interi, memorizzati nelle variabili **`numero1`** e **`numero2`**. Non c'è necessità di ricevere un valore di output. Scrivi uno spezzone di codice per implementare quanto detto.

Intent intent = new Intent(ActivityA.this, ActivityB.class);
intent.putExtra("numero1", numero1);
intent.putExtra("numero2", numero2);
startActivity(intent);
In questo modo, stai creando un'istanza di Intent con l'activity corrente (ActivityA) come contesto e l'activity di destinazione (ActivityB) come classe da avviare. Quindi stai aggiungendo i valori delle variabili numero1 e numero2 come extra all'intent, utilizzando le chiavi "numero1" e "numero2" per identificarli. Infine, stai avviando ActivityB con il metodo startActivity().

In ActivityB, puoi recuperare i valori passati come extra utilizzando il metodo getIntExtra():

Copy code
int numero1 = getIntent().getIntExtra("numero1", 0);
int numero2 = getIntent().getIntExtra("numero2", 0);
Il secondo parametro del metodo getIntExtra() è il valore di default da restituire nel caso in cui l'extra con la chiave specificata non sia stato trovato. In questo esempio, se i valori per "numero1" e "numero2" non sono stati passati all'intent, verranno impostati su 0.

Che cosa è il backstack delle activity? In relazione ad esso, che differenza c'è fra un'activity e un frammento? Come si può modificare il modo in cui Android tratta activity e frammenti in relazione al backstack?

Il backstack delle activity è una pila che tiene traccia delle activity aperte dall'utente durante l'utilizzo dell'app. Ogni volta che viene avviata una nuova activity, viene aggiunta in cima allo stack. Quando l'utente esce dall'activity attuale, torna all'activity precedente nello stack.

Una differenza fondamentale fra un'activity e un frammento è che un'activity rappresenta una schermata intera nell'app, mentre un frammento rappresenta solo una parte di una schermata. Un'activity può contenere più frammenti e i frammenti possono essere utilizzati all'interno di più activity.

Per modificare il modo in cui Android tratta activity e frammenti in relazione al backstack, puoi utilizzare diverse opzioni nell'intent con cui avvii l'activity o il frammento. Ad esempio, puoi impostare l'opzione FLAG_ACTIVITY_NEW_TASK nell'intent per indicare che l'activity o il frammento deve essere avviato in una nuova task, creando così un nuovo backstack separato. Oppure puoi utilizzare l'opzione FLAG_ACTIVITY_CLEAR_TOP per indicare che tutte le activity sopra quella attualmente in cima allo stack devono essere eliminate e che l'activity o il frammento specificato deve essere portato in cima allo stack. Ci sono molte altre opzioni disponibili che puoi utilizzare per modificare il comportamento del backstack.

La seguente classe Tavolozza implementa un widget customizzato che permette di inserire degli oggetti grafici (cerchi colorati) in uno spazio bianco. I cerchi sono memorizzate con degli oggetti Cerchio che specificano la posizione e il colore tramite le coordinate cartesiane e un intero. 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). I cerchi sono identificati dalla loro posizione, più specificamente l'ID di un cerchio è una stringa data alla concatenazione delle coordinate cartesiane e può essere gestito, per un cerchio c, con i metodi c.setID e c.getID.

Completare la classe Tavolozza per permettere di inserire nuovi cerchi e cancellare quelle esistenti, scrivendo due nuovi metodi.

Cosa si deve fare, all'esterno di questa classe, dopo avere inserito o cancellato un cerchio affinché la modifica venga visualizzata?

```
public class Pentagramma extends View {
      HashMap<String,Cerchio> cerchi = new HashMap<>();

      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 i cerchi presenti nell'HashMap cerchi
      }
```

Per inserire un nuovo cerchio nell'oggetto HashMap cerchi della classe Tavolozza, puoi scrivere il seguente metodo:

Copy code
```
public void aggiungiCerchio(Cerchio cerchio) {
   cerchi.put(cerchio.getID(), cerchio);
}
```
Per cancellare un cerchio esistente dall'HashMap, puoi scrivere il seguente metodo:

Copy code
```
public void rimuoviCerchio(Cerchio cerchio) {
   cerchi.remove(cerchio.getID());
}
```
Dopo avere inserito o cancellato un cerchio, è necessario invocare il metodo invalidate() sull'oggetto Tavolozza per forzare il ridisegno della vista e far sì che la modifica venga visualizzata. Ad esempio, puoi scrivere:

Copy code
```
Tavolozza tavolozza = new Tavolozza(getContext(), screen_w, screen_h);
Cerchio cerchio = new Cerchio(x, y, colore);
tavolozza.aggiungiCerchio(cerchio);
tavolozza.invalidate();
```
Questo farà sì che il nuovo cerchio venga aggiunto all'HashMap e che la vista venga ridisegnata con il nuovo cerchio incluso. Allo stesso modo, puoi utilizzare il metodo rimuoviCerchio() e invalidate() per cancellare un cerchio esistente.

Per la memorizzazione di dati strutturati Android permette di usare database SQL. Per il loro utilizzo è fondamentale la classe Cursor. Spiega a cosa serve tale classe e perché è fondamentale.

La classe Cursor è utilizzata in Android per eseguire query su database SQL e per iterare sui risultati delle query. Un oggetto Cursor rappresenta un puntatore a una posizione all'interno di un set di risultati ottenuti da una query al database.

La classe Cursor è fondamentale perché fornisce un modo conveniente per accedere ai dati restituiti da una query al database in una maniera efficiente in termini di memoria. Invece di caricare tutti i risultati della query in memoria, un oggetto Cursor ti permette di accedere ai dati uno alla volta, spostandosi avanti e indietro attraverso il set di risultati utilizzando i metodi moveToNext() e moveToPrevious(). In questo modo, è possibile gestire grandi quantità di dati senza doverli caricare tutti in memoria contemporaneamente, il che riduce la pressione sulla memoria del dispositivo.

Inoltre, la classe Cursor offre numerosi metodi per accedere ai dati restituiti dalla query, come getInt(), getString(), getDouble() e così via, a seconda del tipo di dati presenti nella colonna. Ciò rende semplice estrarre i dati dai risultati della query e utilizzarli nell'app.

In sintesi, la classe Cursor è fondamentale perché fornisce un modo efficiente per accedere ai dati restituiti da una query al database in Android, consentendo di gestire grandi quantità di dati senza doverli caricare tutti in memoria contemporaneamente.

una MainActivity che può ospitare due tipi di frammento, FrammentoA e FrammentoB. I due frammenti possono essere visualizzati contemporaneamente solo quando il dispositivo è in modalità landscape; se è in modalità portrait il FrammentoB dovrà essere visualizzato come nuova actvity.

Progetta uno schema astratto per far comunicare il FrammentoA con il FrammentoB (per semplicità assumi che la comunicazione sia semplicemente un intero memorizzato nella variabile “numero”).

Per far comunicare il FrammentoA con il FrammentoB, potresti utilizzare uno dei seguenti approcci:

Utilizzare un listener: il FrammentoA definisce un'interfaccia con un metodo di callback, ad esempio "onNumeroChanged(int numero)". La MainActivity implementa questa interfaccia e passa un'istanza di se stessa al FrammentoA tramite un metodo setListener(). Quando il numero cambia nel FrammentoA, viene invocato il metodo onNumeroChanged() passando il nuovo valore come parametro. Nel FrammentoB, puoi ottenere il valore del numero utilizzando un metodo getNumero() della MainActivity.

Utilizzare una classe singleton: puoi creare una classe singleton che gestisce lo stato condiviso fra i frammenti. Ad esempio, puoi definire una classe Stato con una variabile numero e un metodo setNumero(). Nel FrammentoA, puoi utilizzare il metodo setNumero() per impostare il nuovo valore del numero. Nel FrammentoB, puoi utilizzare il metodo getNumero() della classe Stato per ottenere il valore corrente del numero.

Utilizzare la classe Bundle: puoi utilizzare la classe Bundle per passare i dati fra i frammenti. Nel FrammentoA, puoi creare un'istanza di Bundle e inserire il valore del numero come extra. Poi, puoi passare l'istanza di Bundle al FrammentoB utilizzando il metodo setArguments() del frammento. Nel FrammentoB, puoi ottenere il Bundle passato utilizzando il metodo getArguments() e quindi estrarre il valore del numero dall'extra.

Quale di questi approcci è più adeguato dipende dalle tue specifiche esigenze. Ad esempio, se hai bisogno di una comunicazione frequente fra i frammenti, potresti trovare più conveniente utilizzare un listener. Se hai bisogno di gestire lo stato condiviso in modo indipendente dai frammenti, potresti trovare più adeguato utilizzare una classe singleton. Se hai bisogno solo di passare dei dati statici fra i frammenti, potresti trovare più conveniente utilizzare la classe Bundle.

Il seguente frammento di codice mostra un OnTouchListener per un MotionEvent. Si completi il codice facendo in modo che la variabile counter (si assuma che tale variabile sia accessibile globalmente) contenga sempre il numero di dita che stanno toccando lo schermo.

```
int counter

public boolean onTouch(View v, MotionEvent event) {

    switch(event.getActionMasked()) {

        case MotionEvent.ACTION_DOWN:

            counter = 1;

            break;

        case MotionEvent.ACTION_POINTER_DOWN:

            counter++;

            break;

        case MotionEvent.ACTION_MOVE:

            // Non fa nulla in questo caso

            break;

        case MotionEvent.ACTION_POINTER_UP:

            counter--;

            break;

        case MotionEvent.ACTION_UP:

            counter = 0;

            break;

    }

    return true;

}
```

Descrivi il meccanismo di layout (fasi di misurazione, posizionamento e disegno dei widget) utilizzato da Android per costruire l'interfaccia utente descritta nel file di layout.

Quando Android costruisce l'interfaccia utente descritta in un file di layout, segue le seguenti fasi:

Misurazione: in questa fase, vengono calcolate le dimensioni dei widget utilizzando le regole di layout specificate nel file di layout. Ad esempio, se un widget ha il layout_width impostato su "match_parent", verrà esteso per riempire tutto lo spazio disponibile del suo parent. Se un widget ha il layout_height impostato su "wrap_content", verrà esteso solo fino a coprire il contenuto interno.

Posizionamento: in questa fase, vengono calcolate le posizioni dei widget utilizzando le regole di layout specificate nel file di layout. Ad esempio, se un widget ha il layout_gravity impostato su "center", verrà centrato all'interno del suo parent. Se un widget ha il layout_margin impostato su un valore positivo, verrà inserito con un margine di quel valore intorno al bordo del suo parent.

Disegno: in questa fase, i widget vengono disegnati sullo schermo utilizzando le dimensioni e le posizioni calcolate nelle fasi precedenti. Ad esempio, se un widget ha uno sfondo colorato, verrà disegnato un rettangolo di quel colore con le dimensioni e la posizione specificate. Se un widget ha un'immagine come sfondo, verrà disegnata l'immagine con le dimensioni e la posizione specificate.

Il meccanismo di layout viene ripetuto ogni volta che l'interfaccia utente viene modificata, ad esempio quando viene aggiunto o rimosso un widget o quando viene modificato un widget esistente. In questo modo, l'interfaccia utente viene costantemente aggiornata per riflettere le modifiche apportate.

Inoltre, il meccanismo di layout viene ripetuto ogni volta che l'orientamento dello schermo viene modificato, ad esempio quando il dispositivo viene ruotato da orizzontale a verticale o viceversa. In questo modo, l'interfaccia utente viene adattata alla nuova orientazione dello schermo.

In sintesi, il meccanismo di layout di Android è responsabile della costruzione dell'interfaccia utente descritta nel file di layout, effettuando le fasi di misurazione, posizionamento e disegno dei widget e ripetendole ogni volta che l'interfaccia utente viene modificata o quando l'orientamento dello schermo viene modificato.

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva