Corso di laurea in Informatica

Mobile Programming

Prof. De Prisco

Prova scritta del

esempio 5

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

Per ognuno dei seguenti casi scrivere il corrispondente spezzone di codice XML da inserire nel file di layout.

1. Il testo “Ciao”, non editable, largo quanto il contenitore e alto quanto basta a visualizzare il testo con il testo allineato al centro.

```
<TextView
   android:layout_height="wrap_content"
   android:layout_width="match_parent"
   android:gravity="center" />
```

2. Il testo “Android” editable, largo e alto quanto basta a visualizzarlo, che si posiziona sulla destra del suo contenitore.

```
<EditText
   android:layout_height="wrap_content"
   android:layout_width="wrap_content"
   android:gravity="right"/>
```

3. Un pulsante, largo 300dp e alto 60dp, con identificativo “pulsante3”, con testo “Premi” allineato al centro del pulsante, ed avente come listener la funzione “pulsantePremuto()”

```
<Button
   android:layout_width="300dp"
   android:layout_height="60dp"
   andoird:id="@+id/pulsante3"
   android:text="Premi"
   android:gravity="center"
   android:onClick="pulsantePremuto" />
```

4. Una progessBar alta 10dp e larga quanto il suo contenitore con identificativo “pb1”, inizialmente invisibile.

```
<ProgressBar
   android:id="@+id/pb1"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="match_parent"
   android:layout_height="10dp"
   android:visibility="invisible"
   />
```

5. Un gridView largo 400dp e alto 500dp in cui gli elementi sono organizzati su 4 colonne

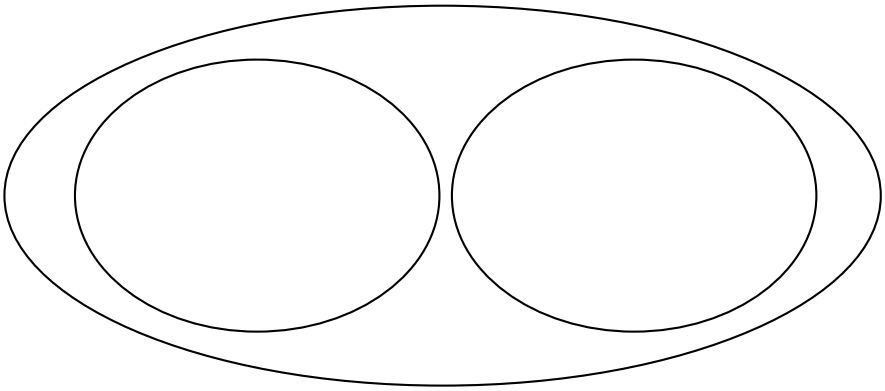
```
<GridView
   android:id="@+id/gridView"
   android:layout_width="400dp"
   android:layout_height="500dp"
   android:numColumns="4"
   />
```

Android permette di specificare le dimensioni degli elementi dell'interfaccia grafica utilizzando varie unità di misura:

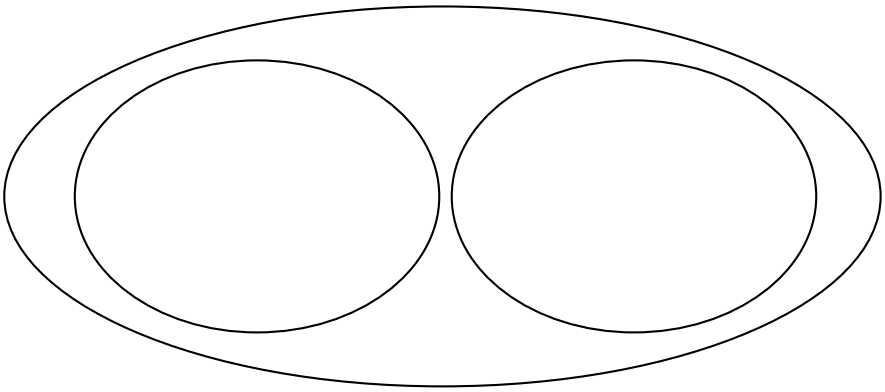
1. px, pixels
2. dp, density-independent pixels
3. sp, scale-independent pixels
4. in, pollici (inches)
5. mm, millimetri
6. pt, punti

Partiziona questo insieme di misure in due sottoinsiemi prima in funzione del fatto che la dimensione fisica sullo schermo è indipendente dalla densità del display e poi in funzione del fatto che la dimensione fisica sullo schermo dipende dalle preferenze dell'utente sulla grandezza dei font.

Partizionamento rispetto all'indipendenza dalla densità dello schermo:



Partizionamento rispetto all'indipendenza dalle preferenze dell'utente sui font:



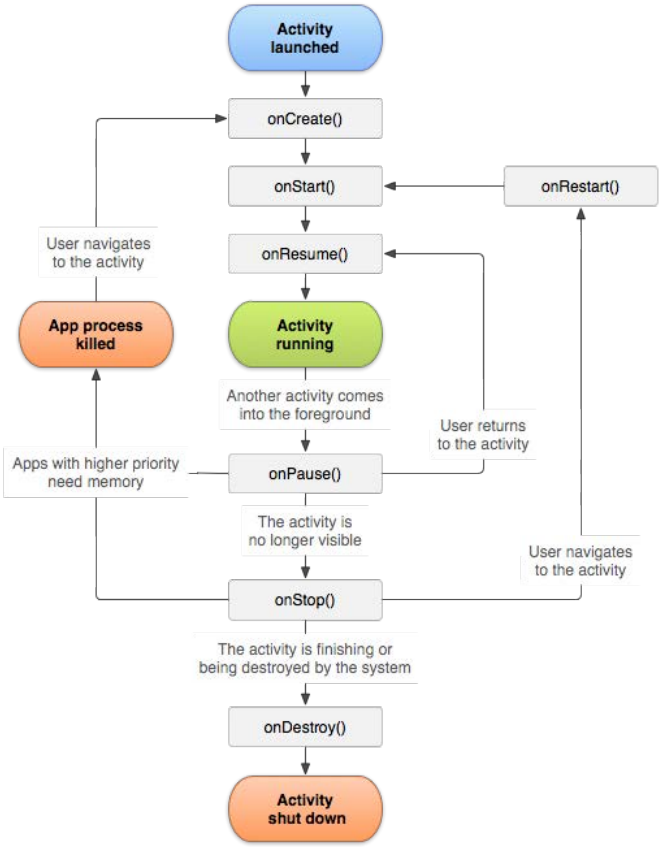
Le unità di misura che sono indipendenti dalla densità del display sono:

dp (density-independent pixels)
sp (scale-independent pixels)
Le unità di misura che dipendono dalle preferenze dell'utente sulla grandezza dei font sono:

sp (scale-independent pixels)
Le unità di misura che non sono influenzate dalla densità del display o dalle preferenze dell'utente sulla grandezza dei font sono:

px (pixels)
in (pollici)
mm (millimetri)
pt (punti)

Disegna il ciclo di vita delle activity e spiega cosa succede a (1) un'activity, che è in esecuzione in foreground e (2) a un'activity che è in esecuzione in background, quando viene ruotato lo schermo.



(1) Quando viene ruotato lo schermo mentre un'activity è in esecuzione in foreground, l'activity viene distrutta e ricreata. Ciò significa che la chiamata all'activity viene interrotta e ripresa quando l'activity viene ricreata.

(2) Quando viene ruotato lo schermo mentre un'activity è in esecuzione in background, l'activity non viene distrutta o ricreata. Tuttavia, può essere influenzata dalla rotazione dello schermo in altri modi, ad esempio se si utilizzano fragmenti o se si salvano e ripristinano dati specifici dell'orientamento dello schermo.

Nota: è possibile disabilitare la distruzione e la ricreazione delle activity durante la rotazione dello schermo utilizzando l'attributo android:configChanges nel file Manifest. Tuttavia, questo può rendere l'app meno flessibile e potrebbe richiedere ulteriore lavoro per assicurarsi che l'interfaccia utente si adatti correttamente alla rotazione dello schermo.

Le due seguenti funzioni fanno parte del codice dell'ActivityA che lancia l'ActivityB per ottenere come risultato un valore booleano e un valore intero. Si completi il codice del metodo `onActivityResult()` in modo tale che alla fine dell'esecuzione dell'AcvtivityB il valore intero sia memorizzato nella variabile **`numero`** e il valore booleano nella variabile **`flag`**. L'activityB resitutisce i valori come dati "extra" associati alle stringhe "VALORE_BOOLEANO" e "VALORE_INTERO". Si facciano gli opportuni controlli per assicurarsi che tutto sia andata per il verso giusto.

```
private boolean flag;
private int numero = 0;


public void lanciaActivityB() {
       Intent i = new Intent();
        i.setClass(getApplicationContext(), ActivityB.class);
        startActivityForResult(i,77);
}
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   if (requestCode != 77) return;
   if (resultCode != Activity.RESULT_OK) return;

   boolean booleanValue = data.getBooleanExtra("VALORE_BOOLEANO", false);
   int intValue = data.getIntExtra("VALORE_INTERO", 0);

   flag = booleanValue;
   numero = intValue;
}
```

Il metodo getBooleanExtra() recupera il valore booleano associato alla stringa "VALORE_BOOLEANO" nell'Intent data, mentre il metodo getIntExtra() recupera il valore intero associato alla stringa "VALORE_INTERO" nell'Intent data. In entrambi i casi, viene fornito un valore predefinito come argomento di backup nel caso in cui il valore non sia presente nell'Intent.

Si consideri la classe `AsyncTask<Integer, Integer, Bitmap>`. A cosa servono i tre parametri che in questo particolare esempio sono istanziati con i tipi `Integer, Integer, Bitmap`? Di quale metodi si deve fare l'ovveride? Per ognuno di tali metodi si spieghi quali sono le operazioni solitamente svolte in esso.

La classe AsyncTask è una classe di supporto che permette di eseguire operazioni in modo asincrono sulla UI thread. Nel caso specifico, i tre parametri AsyncTask<Integer, Integer, Bitmap> indicano i tipi di dati utilizzati nell'esecuzione del task asincrono.

I tre parametri sono:

Integer: il tipo di dati utilizzato per gli argomenti di input del task asincrono.
Integer: il tipo di dati utilizzato per indicare i progressi del task asincrono.
Bitmap: il tipo di dati restituito dal task asincrono una volta completato.
Per utilizzare AsyncTask, è necessario estendere la classe AsyncTask e implementare almeno uno dei seguenti metodi:

doInBackground(Params...): il metodo che esegue il task asincrono. Riceve gli argomenti di input come parametri e restituisce il risultato del task una volta completato.
onPreExecute(): il metodo che viene chiamato prima dell'avvio del task asincrono. Solitamente viene utilizzato per impostare le risorse o le configurazioni necessarie per l'esecuzione del task.
onPostExecute(Result): il metodo che viene chiamato una volta completato il task asincrono. Riceve il risultato del task come parametro e solitamente viene utilizzato per visualizzare il risultato o per effettuare altre operazioni di pulizia.
onProgressUpdate(Progress...): il metodo che viene chiamato per aggiornare i progressi del task asincrono. Riceve i valori di progresso come parametri e solitamente viene utilizzato per aggiornare un'interfaccia utente, ad esempio una barra di avanzamento.

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    ImageView bmImage;

    public DownloadImageTask(ImageView bmImage) {
        this.bmImage = bmImage;
    }

    protected Bitmap doInBackground(String... urls) {
        String url = urls[0];
        Bitmap bitmap = null;
        try {
            InputStream in = new java.net.URL(url).openStream();
            bitmap = BitmapFactory.decodeStream(in);
        } catch (Exception e) {
            Log.e("Error", e.getMessage());
            e.printStackTrace();
        }
        return bitmap;
    }

    protected void onPostExecute(Bitmap result) {
        bmImage.setImageBitmap(result);
    }
}
```

Per eseguire il task asincrono, basta creare un'istanza della classe DownloadImageTask e chiamare il metodo execute() passando l'URL dell'immagine come argomento.

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
      }
```

Per visualizzare le modifiche dopo aver inserito o cancellato una nota, è necessario chiamare il metodo invalidate() della classe View, che causa il ridisegno della vista. Potresti aggiungere questa chiamata al metodo addNote() e al metodo removeNote() per aggiornare l'interfaccia utente ogni volta che viene modificata la lista delle note.

Copy code

```
public void addNote(Nota n) {
   notesList.add(n);
   invalidate();
}

public void removeNote(int index) {
   notesList.remove(index);
   invalidate();
}
```

Ecco un esempio di come utilizzare i metodi addNote() e removeNote() per inserire e cancellare note dal pentagramma:

Copy code

```
Pentagramma pentagramma = new Pentagramma(getApplicationContext(), screen_w, screen_h);

// Crea una nuova nota e la aggiunge al pentagramma
Nota nota = new Nota(0, 0);
pentagramma.addNote(nota);

// Rimuove la nota dal pentagramma
pentagramma.removeNote(0);
```

In questo esempio, viene creata una nuova istanza della classe Pentagramma e quindi viene creata una nuova nota utilizzando il costruttore della classe Nota. La nota viene quindi aggiunta al pentagramma tramite il metodo addNote(). Infine, viene rimossa la nota dal pentagramma chiamando il metodo removeNote() e passando l'indice della nota nell'ArrayList notesList.

Descrivi brevemente gli approcci che un'app può utilizzare per la memorizzazione di informazioni persistenti. Quale di questi metodi conviene utilizzare e in quali casi?

Ci sono diversi modi in cui un'app può memorizzare informazioni persistenti:

Shared Preferences: è un modo semplice per salvare le impostazioni dell'app e altre informazioni di piccole dimensioni in modo che siano disponibili anche dopo il riavvio dell'app.

Database: è possibile utilizzare un database per memorizzare grandi quantità di dati strutturati in modo che siano facilmente accessibili e modificabili. I database possono essere ottimizzati per le prestazioni e possono essere utilizzati per creare relazioni tra i dati.

File: è possibile utilizzare i file per memorizzare grandi quantità di dati non strutturati. Questo può essere utile per memorizzare immagini, audio e altri tipi di file media.

In generale, Shared Preferences è adeguato per memorizzare le impostazioni dell'app e altre informazioni di piccole dimensioni, mentre i database e i file sono più adatti per la memorizzazione di grandi quantità di dati.

Stai sviluppando un'app che permette di visualizzare una serie di scacchiere. Assumi di avere un oggetto `Tris`, 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 9. Supponendo di avere un file di layout che preveda 9 `FrameLayout`, i cui identificatori sono `frame1`, `frame2`, `frame3`, …., `frame9`, scrivi uno spezzone di codice che permetta di inserire in modo programmatico N frammenti, ognuno dei quali è un oggetto `Tris`, in N contenitori `FrameLayout`, a partire dal primo (`frame1`).

```
for (int i = 1; i <= N; i++) {
   n++;
   if (n > 8) break;
   Fragment fragment = new Tris();
   int frameId = getResources().getIdentifier("frame" + i, "id", getPackageName());
   FrameLayout frame = findViewById(frameId);
   getSupportFragmentManager().beginTransaction().add(frame.getId(), fragment).commit();
}
```

In questo codice, viene utilizzato un ciclo for per creare e aggiungere N frammenti di tipo Tris ai FrameLayout con identificatore frame1, frame2, ..., frameN.

Per ottenere l'identificatore del FrameLayout corrente, viene utilizzato il metodo getIdentifier della classe Resources, che restituisce l'identificatore del FrameLayout a partire dal suo nome e dal nome del pacchetto dell'app.

Viene quindi utilizzato il metodo findViewById per ottenere l'oggetto FrameLayout corrispondente e il metodo beginTransaction della classe FragmentManager per avviare una transazione che consente di aggiungere il frammento al FrameLayout. Infine, il metodo commit viene utilizzato per applicare la transazione.

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.

Si spieghi come funziona il meccanismo del Multitouch, spiegando cosa sono i MotionEvent, i Pointer e gli Action_Codes.

Il multitouch consente di rilevare e gestire più punti di contatto con il display di un dispositivo allo stesso tempo. Ciò significa che più persone possono interagire con il dispositivo utilizzando le dita o altri oggetti, ad esempio un pennino.

In Android, il multitouch viene gestito attraverso gli oggetti MotionEvent. Un MotionEvent rappresenta un evento di input che può essere generato dal tocco di un dito o di un pennino sullo schermo. Ogni MotionEvent contiene informazioni sulla posizione del puntatore, sulla pressione esercitata e sulla durata dell'evento.

Un MotionEvent può contenere informazioni su più puntatori, ciascuno dei quali rappresenta un punto di contatto con lo schermo. Ogni punto di contatto è rappresentato da un oggetto Pointer. Un Pointer contiene informazioni come l'ID del punto di contatto, la posizione del punto di contatto e la pressione esercitata.

Gli Action_Codes sono costanti che indicano il tipo di azione associata a un MotionEvent. Ad esempio, ACTION_DOWN indica che un punto di contatto è stato premuto, mentre ACTION_UP indica che un punto di contatto è stato rilasciato. Altri esempi di Action_Codes sono ACTION_MOVE, ACTION_CANCEL e ACTION_OUTSIDE.

Quando si lavora con il multitouch in Android, si utilizzano spesso gli Action_Codes per determinare il tipo di azione che ha generato un MotionEvent e per gestire il comportamento dell'app in risposta a tali azioni.

Fai un esempio di ContentProvider, un esempio di BroadcastReceiver e un esempio di Service. Fornisci qualche dettaglio per ognuno degli esempi.

Ecco un esempio di ContentProvider:

Copy code
```
public class ContactProvider extends ContentProvider {
    private static final int CONTACTS = 1;
    private static final int CONTACT_ID = 2;

    private static final UriMatcher URI_MATCHER;

    static {
        URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
        URI_MATCHER.addURI(ContactsContract.AUTHORITY, "contacts",
CONTACTS);
        URI_MATCHER.addURI(ContactsContract.AUTHORITY, "contacts/#",
CONTACT_ID);
    }

    private ContactDatabaseHelper mDatabaseHelper;

    @Override
    public boolean onCreate() {
        mDatabaseHelper = new ContactDatabaseHelper(getContext());
        return true;
    }

    @Override
    public Cursor query(@NonNull Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();

queryBuilder.setTables(ContactDatabaseHelper.CONTACTS_TABLE_NAME
);

        switch (URI_MATCHER.match(uri)) {
            case CONTACTS:
                break;
            case CONTACT_ID:
                queryBuilder.appendWhere(ContactsContract.Contacts._ID + "=" +
uri.getLastPathSegment());
                break;
            default:
                throw new IllegalArgumentException("Invalid URI: " + uri);
        }

        SQLiteDatabase db = mDatabaseHelper.getReadableDatabase();
        Cursor cursor = queryBuilder.query(db, projection, selection,
selectionArgs, null, null, sortOrder);
        cursor.setNotificationUri(getContext().getContentResolver(), uri);
        return cursor;
    }

    // Altri metodi come insert, update e delete...
}
```
Questo è un esempio di un ContentProvider che gestisce i contatti di un'app. Il ContentProvider espone i dati dei contatti attraverso un'URI e fornisce metodi per eseguire operazioni di lettura, scrittura e cancellazione sui dati dei contatti.

Ecco un esempio di Service:

Copy code
```
public class DownloadService extends Service {
    private static final int NOTIFICATION_ID = 1;
    private NotificationManager mNotificationManager;
    private NotificationCompat.Builder mBuilder;
    private ExecutorService mExecutor;

    @Override
    public void onCreate() {
        super.onCreate();
        mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        mBuilder = new NotificationCompat.Builder(this, "default")
                .setSmallIcon(R.drawable.ic_download)
                .setContentTitle("Download in corso")
                .setProgress(0, 0, true);
        mExecutor = Executors.newSingleThreadExecutor();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        String url = intent.getStringExtra("url");
        mExecutor.execute(new DownloadTask(url));
        startForeground(NOTIFICATION_ID, mBuilder.build());
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        mExecutor.shutdown();
        stopForeground(true);
        super.onDestroy();
    }

    private class DownloadTask implements Runnable {
        private String mUrl;

        public DownloadTask(String url) {
            mUrl = url;
        }

        @Override
        public void run() {
            // Scarica il file...

            // Aggiorna la notifica di avanzamento
            mBuilder.setProgress(100, progress, false);
            mNotificationManager.notify(NOTIFICATION_ID,
mBuilder.build());

            // Quando il download è completato, rimuove la notifica
            mNotificationManager.cancel(NOTIFICATION_ID);
        }
    }
}
```
Questo è un esempio di un Service che scarica un file da Internet e visualizza una notifica di avanzamento durante il download. Il Service viene avviato quando viene inviato un intent che contiene l'URL del file da scaricare. Il Service utilizza un Executor per eseguire il download in un thread separato e visualizza una notifica di avanzamento durante il download. Quando il download è completato, il Service rimuove la notifica.

Ecco un esempio di BroadcastReceiver:

```
Copy code
public class BatteryReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int level = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, 0);
        int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
        boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
                status == BatteryManager.BATTERY_STATUS_FULL;

        String message = "Battery level: " + level + "%";
        if (isCharging) {
            message += " (charging)";
        }
        Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
    }
}
```

Questo è un esempio di un BroadcastReceiver che visualizza un messaggio di toast quando il livello della batteria cambia o quando il dispositivo viene collegato o scollegato dall'alimentazione. Il BroadcastReceiver viene registrato per ricevere gli intent di sistema che segnalano i cambiamenti del livello della batteria o lo stato di ricarica. Quando viene ricevuto un intent, il BroadcastReceiver estrae le informazioni sulla batteria dall'intent e le utilizza per costruire un messaggio da visualizzare nella toast.

Un ContentProvider è un componente di Android che fornisce un meccanismo per la gestione dei dati in modo da renderli disponibili a più app. Un ContentProvider espone i dati attraverso un'URI (Uniform Resource Identifier) e fornisce metodi per eseguire operazioni di lettura, scrittura e cancellazione sui dati. Gli sviluppatori possono utilizzare i ContentProvider per condividere i dati tra le proprie app o per rendere disponibili i dati ad altre app installate sul dispositivo.

Un BroadcastReceiver è un componente di Android che consente di ricevere e gestire gli intent di sistema e di altre app. Un BroadcastReceiver può essere registrato per ricevere determinati tipi di intent e può essere utilizzato per eseguire un'azione in risposta all'intent, ad esempio visualizzare una notifica o avviare un Service.

Un Service è un componente di Android che esegue operazioni in background per conto dell'app. Un Service può essere avviato quando viene inviato un intent e può continuare a eseguire operazioni anche quando l'app che lo ha avviato non è in primo piano. Un Service può essere utilizzato per eseguire operazioni di lunga durata, come il download di file o il caricamento di dati da un server, senza interrompere l'app.

Prova scritta

Pagina aggiuntiva

Prova scritta

Pagina aggiuntiva