Exercise 8
Mobile Device Development
Content Providers
Due: 4/23/2020@11:59pm

ContentProvider Overview

A content provider component supplies data from one application to others. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

Each Android applications runs in its own process with its own permissions which keeps an application data private to its own. But it is often required to share data across applications.

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in a SQlite database.

A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
}
```

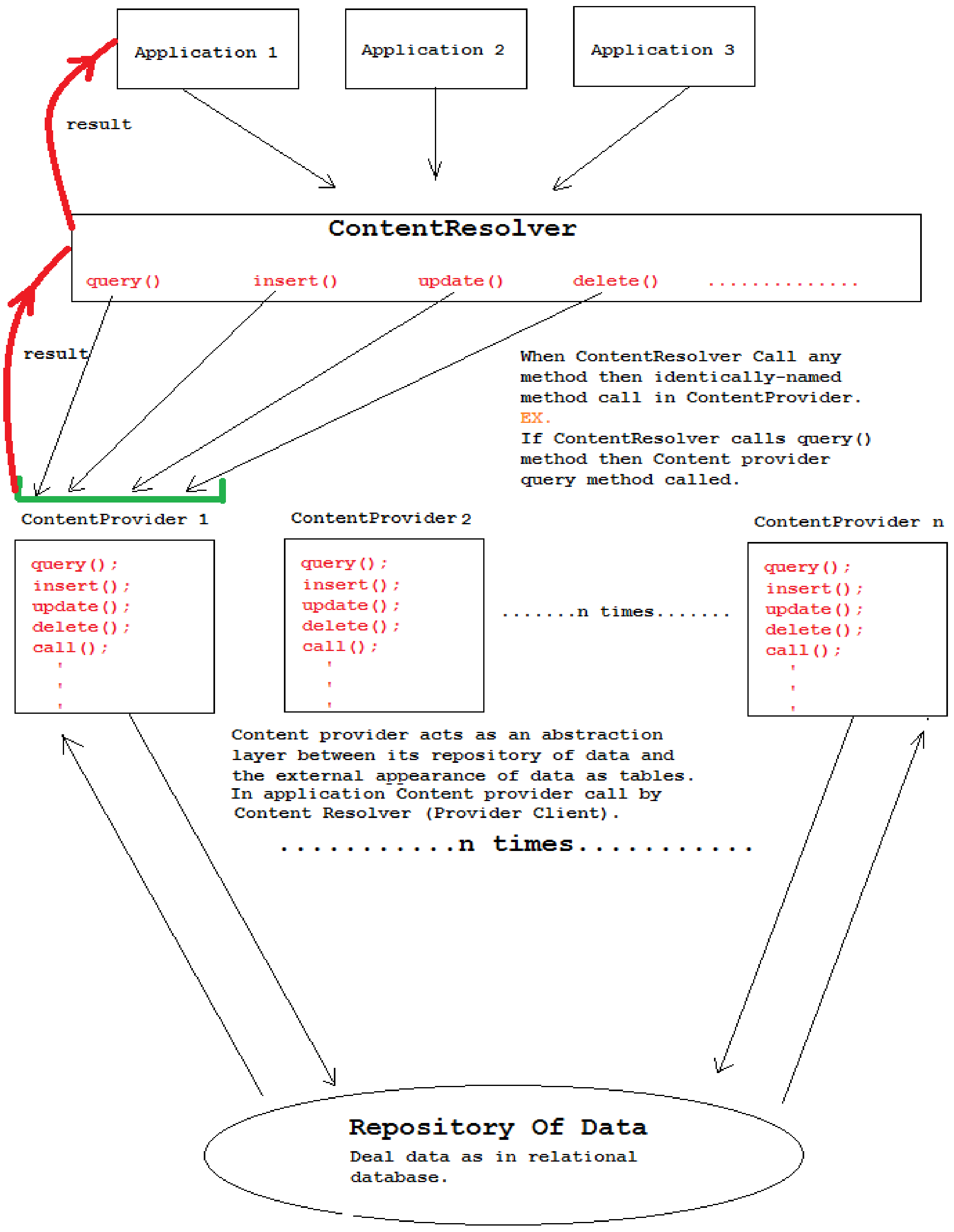
Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format: **`<prefix>://<authority>/<data_type>/<id>`**

Here is the detail of various parts of the URI:

Part	Description
prefix	This is always set to content://
authority	This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
data_type	This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this *content://contacts/people*
id	This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.

Content Provider Workflow:


Application 1
Application 2
Application 3
result
ContentResolver
query()
insert()
update()
delete()
....
result
When ContentResolver Call any method then identically-named method call in ContentProvider.
EX.
If ContentResolver calls query() method then Content provider query method called.
ContentProvider 1
query();
insert();
update();
delete();
call();
ContentProvider 2
query();
insert();
update();
delete();
call();
.......n times.......
ContentProvider n
query();
insert();
update();
delete();
call();
Content provider acts as an abstraction layer between its repository of data and the external appearance of data as tables. In application Content provider call by Content Resolver (Provider Client).
...........n times...........
Repository Of Data
Deal data as in relational database.

Create Content Provider

This involves number of simple steps to create your own content provider.

- Create a Content Provider class that extends the *ContentProvider* base class.
- Define your content provider URI address which will be used to access the content.
- Create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLiteOpenHelper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.
- Implement Content Provider queries to perform different database specific operations.
- Register your Content Provider in your activity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working:

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Exercise 8

This exercise shows how to create your own *ContentProvider*. The steps to complete this exercise are as follows:

1. You will use Android Studio to create an Android application and name it as MyContentProvider under a package edu.sjsu.android.mycontentprovider, with empty Activity.
2. Modify main activity file MainActivity.java to add two new methods onClickAddName() and onClickRetrieveStudents().
3. Create a new java file called StudentsProvider.java under the package edu.sjsu.android to define your actual provider and associated methods.
4. Register your content provider in your AndroidManifest.xml file using <provider.../> tag
5. Modify the default content of res/layout/activity_main.xml file to include a small GUI to add students records.
6. Define required constants in res/values/strings.xml file
7. Run the application to launch Android emulator and verify the result of the changes done in the application.

In the MainActivity.java file add two new methods *onClickAddName()* and *onClickRetrieveStudents()* to handle user interaction with the application.

```
package edu.sjsu.android.mycontentprovider;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

   public void onClickAddName(View view) {
      // Add a new student record
      ContentValues values = new ContentValues();

      values.put(StudentsProvider.NAME,
      ((EditText)findViewById(R.id.txtName)).getText().toString());

      values.put(StudentsProvider.GRADE,
      ((EditText)findViewById(R.id.txtGrade)).getText().toString());

      Uri uri = getContentResolver().insert(
      StudentsProvider.CONTENT_URI, values);

      Toast.makeText(getBaseContext(),
      uri.toString(), Toast.LENGTH_LONG).show();
   }
   public void onClickRetrieveStudents(View view) {
      // Retrieve student records
      String URL = "content://" + getString(R.string.provider) + "/students";
      Uri students = Uri.parse(URL);
      Cursor c = getContentResolver().query(students, null, null, null,
"name");
      if (c.moveToFirst()) {
         do{
            Toast.makeText(this,
            c.getString(c.getColumnIndex(StudentsProvider._ID)) +
            ", " +  c.getString(c.getColumnIndex( StudentsProvider.NAME)) +
            ", " + c.getString(c.getColumnIndex( StudentsProvider.GRADE)),
            Toast.LENGTH_SHORT).show();
         } while (c.moveToNext());
      }
   }
}
```

Create new file StudentsProvider.java under *edu.sjsu.android.mycontentprovider* package. The StudentsProvider.java provides the actual ContentProvider methods that you need to override.

```
package edu.sjsu.android.mycontentprovider;

import java.util.HashMap;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;

public class StudentsProvider extends ContentProvider {

   static final String PROVIDER_NAME = 
"sjsu.edu.android.mycontentprovider.College";
   static final String URL = "content://" + PROVIDER_NAME + "/students";
   static final Uri CONTENT_URI = Uri.parse(URL);

   static final String _ID = "_id";
   static final String NAME = "name";
   static final String GRADE = "grade";

   private static HashMap<String, String> STUDENTS_PROJECTION_MAP;

   static final int STUDENTS = 1;
   static final int STUDENT_ID = 2;

   static final UriMatcher uriMatcher;
   static{
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI(PROVIDER_NAME, "students", STUDENTS);
      uriMatcher.addURI(PROVIDER_NAME, "students/#", STUDENT_ID);
   }

   /**
    * Database specific constant declarations
    */
   private SQLiteDatabase db;
   static final String DATABASE_NAME = "College";
   static final String STUDENTS_TABLE_NAME = "students";
   static final int DATABASE_VERSION = 1;
   static final String CREATE_DB_TABLE = 
      " CREATE TABLE " + STUDENTS_TABLE_NAME +
      " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
      " name TEXT NOT NULL, " +
      " grade TEXT NOT NULL);";
```

```
/**
 * Helper class that actually creates and manages
 * the provider's underlying data repository.
 */
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context){
       super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
       db.execSQL(CREATE_DB_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
                          int newVersion) {
       db.execSQL("DROP TABLE IF EXISTS " +  STUDENTS_TABLE_NAME);
       onCreate(db);
    }
}

@Override
public boolean onCreate() {
   Context context = getContext();
   DatabaseHelper dbHelper = new DatabaseHelper(context);
   /**
    * Create a write able database which will trigger its
    * creation if it doesn't already exist.
    */
   db = dbHelper.getWritableDatabase();
   return (db == null)? false:true;
}

@Override
public Uri insert(Uri uri, ContentValues values) {
   /**
    * Add a new student record
    */
   long rowID = db.insert(  STUDENTS_TABLE_NAME, "", values);
   /**
    * If record is added successfully
    */
   if (rowID > 0)
   {
      Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
      getContext().getContentResolver().notifyChange(_uri, null);
      return _uri;
   }
   throw new SQLException("Failed to add a record into " + uri);
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
```

```
      SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
      qb.setTables(STUDENTS_TABLE_NAME);

      switch (uriMatcher.match(uri)) {
      case STUDENTS:
         qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
         break;
      case STUDENT_ID:
         qb.appendWhere( _ID + "=" + uri.getPathSegments().get(1));
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri);
      }
      if (sortOrder == null || sortOrder == ""){
         /**
          * By default sort on student names
          */
         sortOrder = NAME;
      }
      Cursor c = qb.query(db,  projection,    selection, selectionArgs,
                          null, null, sortOrder);
      /**
       * register to watch a content URI for changes
       */
      c.setNotificationUri(getContext().getContentResolver(), uri);

      return c;
   }

   @Override
   public int delete(Uri uri, String selection, String[] selectionArgs) {
      int count = 0;

      switch (uriMatcher.match(uri)){
      case STUDENTS:
         count = db.delete(STUDENTS_TABLE_NAME, selection, selectionArgs);
         break;
      case STUDENT_ID:
         String id = uri.getPathSegments().get(1);
         count = db.delete( STUDENTS_TABLE_NAME, _ID +  " = " + id +
                (!TextUtils.isEmpty(selection) ? " AND (" +
                selection + ')' : ""), selectionArgs);
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri);
      }

      getContext().getContentResolver().notifyChange(uri, null);
      return count;
   }

   @Override
   public int update(Uri uri, ContentValues values, String selection,
                     String[] selectionArgs) {
      int count = 0;

      switch (uriMatcher.match(uri)){
```

```
      case STUDENTS:
         count = db.update(STUDENTS_TABLE_NAME, values,
                 selection, selectionArgs);
         break;
      case STUDENT_ID:
         count = db.update(STUDENTS_TABLE_NAME, values, _ID +
                 " = " + uri.getPathSegments().get(1) +
                 (!TextUtils.isEmpty(selection) ? " AND (" +
                 selection + ')' : ""), selectionArgs);
         break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri );
      }
      getContext().getContentResolver().notifyChange(uri, null);
      return count;
   }

   @Override
   public String getType(Uri uri) {
      switch (uriMatcher.match(uri)){
      /**
       * Get all student records
       */
      case STUDENTS:
         return "vnd.android.cursor.dir/vnd.example.students";
      /**
       * Get a particular student
       */
      case STUDENT_ID:
         return "vnd.android.cursor.item/vnd.example.students";
      default:
         throw new IllegalArgumentException("Unsupported URI: " + uri);
      }
   }
}
```

Following is the modified content of *AndroidManifest.xml* file. Note the <provider.../> tag that is included below

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.sjsu.android.mycontentprovider">


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name="StudentsProvider"
           android:authorities="@string/provider">
        </provider>
    </application>

</manifest>
```

The layout XML file, **res/layout/activity_main.xml**, is included below. It includes a button to broadcast your custom intent:

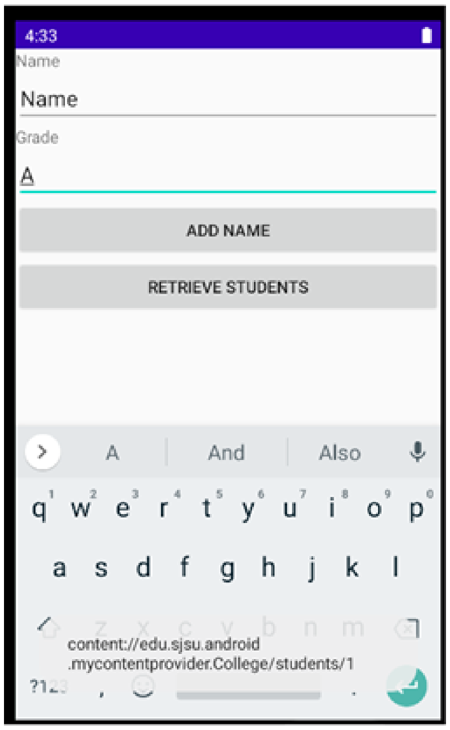
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Name" />
    <EditText
    android:id="@+id/txtName"
    android:layout_height="wrap_content"
    android:layout_width="match_parent" />
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Grade" />
    <EditText
    android:id="@+id/txtGrade"
    android:layout_height="wrap_content"
    android:layout_width="match_parent" />
    <Button
    android:text="Add Name"
    android:id="@+id/btnAdd"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickAddName" />
    <Button
    android:text="Retrieve Students"
    android:id="@+id/btnRetrieve"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickRetrieveStudents" />
</LinearLayout>
```

Make sure you have following content of **res/values/strings.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MyContentProvider</string>
    <string
name="provider">edu.sjsu.android.mycontentprovider.College</string>
</resources>;
```

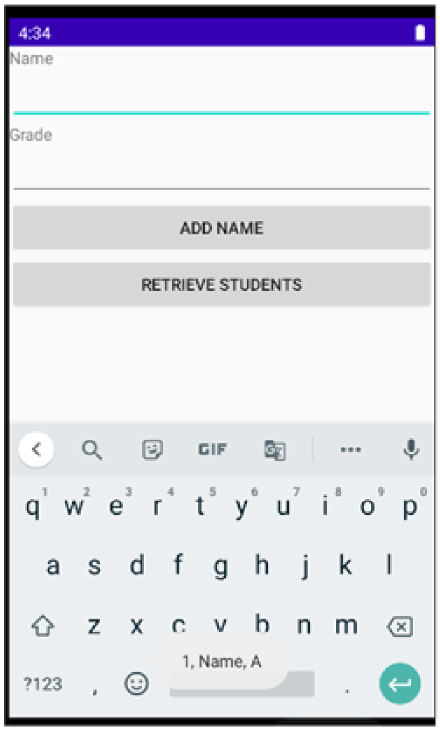
Run your application.

Add name



```
content://edu.sjsu.android
.mycontentprovider.College/students/1
```

Retrieve



```
1, Name, A
```

Submission

1. Push your project directory along with the source to remote bitbucket repository by the due date.
2. Invite and share your project repository the Grader (yan.chen01@sjsu.edu) and Instructor (ramin.moazeni@sjsu.edu).
3. Submit a Readme.pdf to Canvas including your name, repository access link, instructions to run your program (if any), snapshot of your running program
4. Your project directory will be graded according to the state your project directory was in at due time when fetched.