CMR Institute of Technology, Bangalore
Common to all Branches (Open Elective)
II - INTERNAL ASSESSMENT

Semester: 6-CBCS 2018 Date: 24 Jun 2021
Subject: MOBILE APPLICATION DEVELOPMENT (18CS651/17CS661/15CS661)
Faculty: Dr. Sudhakar K N
Time: 01:00 PM – 02:30 PM Max Marks: 50

Scheme & Solution
ANSWER_ANY 5 Question(s)

Marks CO PO BT/CL

1a. If I want to download a large file in my android app, can I run in UI thread. If not why? How to create a background process in Android? Explain usage of AsyncTask to push download to background with code snippet.

[6.0] 1 [1, 2, 3] [3]

Scheme: No option 1M + Process 3M + AsyncTask 2M

Solution

Downloading file is depends on the size of the file, internet connection and many external factors. Hence running on the main thread may crash the application if the response is not received in 6 Seconds.
For the said reason, its preferable to push such tasks to background by forking a new independent thread to handle the task.
All Android apps use a main thread to handle UI operations. Calling long-running operations from this main thread can lead to freezes and unresponsiveness. For example, if your app makes a network request from the main thread, your app's UI is frozen until it receives the network response. You can create additional background threads to handle long-running operations while the main thread continues to handle UI updates.
There are two ways to do background processing in Android: using the AsyncTask class, or the Loader framework, which includes an AsyncTaskLoader class that uses AsyncTask. In most situations you'll choose the Loader framework, but it's important to know how AsyncTask works so you can make a good choice.

AsyncTask

Use the AsyncTask class to implement an asynchronous, long-running task on a worker . (A worker thread is any thread which is not the main or UI thread.) AsyncTask allows to perform background operations and publish results on the UI thread without manipulating threads or handlers.
When AsyncTask is executed, it goes through four steps:

i. onPreExecute() is invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the UI.
ii. doInBackground(Params...) is invoked on the background thread immediately after onPreExecute() finishes. This step performs a background computation, returns a result, and passes the result to onPostExecute(). The doInBackground() method can also call publishProgress(Progress...) to publish one or more units of progress.
iii. onProgressUpdate(Progress...) runs on the UI thread after publishProgress(Progress...) is invoked. Use onProgressUpdate() to report any form of progress to the UI thread while the background computation is executing. For instance, you can use it to pass the data to animate a progress bar or show logs in a text field.
iv. onPostExecute(Result) runs on the UI thread after the background computation has finished.

1b. How to create broadcast receivers? Explain.

[4.0] 1 [1, 2, 3] [2]

Scheme: Description 1M + Process 3M

Solution

Broadcast Receivers:

Explicit intents are used to start specific, fully qualified activities, as well as to pass information between activities in your app. Implicit intents are used to start activities based on registered components that the system is aware of, for example general functionality.
Broadcast intents, which don't start activities but instead are delivered to broadcast receivers.

Example: Create a broadcast receiver:
In this example, the AlarmReceiver subclass of BroadcastReceiver shows a Toast message if the incoming broadcast intent has the action ACTION_SHOW_TOAST :

```
private class AlarmReceiver extends BroadcastReceiver {
	@Override
	public void onReceive(Context context, Intent intent) {
		if (intent.getAction().equals(ACTION_SHOW_TOAST)) {
			CharSequence text = "Broadcast Received!";
			int duration = Toast.LENGTH_SHORT;
			Toast toast = Toast.makeText(context, text, duration);
			toast.show();
		}
	}
}
```

2a. Discuss the limitations of AsyncTask. Where and when can AsyncTask can be used.

[5.0] 1 [1, 2, 3] [2]

Scheme: Limitations 2.5M + Usage 2.5M

Solution

Limitations of AsyncTask

AsyncTask is impractical for some use cases:

1. Changes to device configuration cause problems.

When device configuration changes while an AsyncTask is running, for example if the user changes the screen orientation, the activity that created the AsyncTask is destroyed and re-created. The AsyncTask is unable to access the newly created activity, and the results of the AsyncTask aren't published.

2. Old AsyncTask objects stay around, and your app may run out of memory or crash.

If the activity that created the AsyncTask is destroyed, the AsyncTask is not destroyed along with it. For example, if your user exits the application after the AsyncTask has started, the AsyncTask keeps using resources unless you call cancel( ).

When to use AsyncTask :

- Short or interruptible tasks.
- Tasks that don't need to report back to UI or user.
- Low-priority tasks that can be left unfinished.
- For all other situations, use AsyncTaskLoader, which is part of the Loader framework

2b. How can we overcome the AsyncTask limitations? With a code snippet explain AsyncTask loader [5]

[5.0] 1 [1, 2, 3] [3]

Scheme: Description 5M

Solution

There are two ways to do background processing in Android: using the AsyncTask class, or using the Loader framework, which includes an AsyncTaskLoader class that uses AsyncTask. In most situations you'll choose the Loader framework, but it's important to know how AsyncTask works so you can make a good choice.

Background tasks are commonly used to load data such as forecast reports or movie reviews. Loading data can be memory intensive, and you want the data to be available even if the device configuration changes. For these situations, use loaders, which are a set of classes that facilitate loading data into an activity.

Loaders use the LoaderManager class to manage one or more loaders. LoaderManager includes a set of callbacks for when the loader is created, when it's done loading data, and when it resets.

3a. How can we create the raised buttons and flat buttons? Explain with example.

[7.0] 1 [1, 2, 3] [2]

Scheme: Explanation 4M + Process 3M

Solution

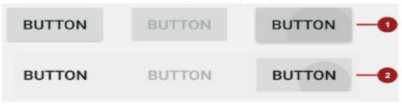
Using buttons:

People like to press buttons. Show someone a big red button with a message that says "Do not press" and the person will likely press it for the sheer pleasure of pressing a big red button (that the button is forbidden is also a factor).When touched or clicked, a button performs an action. It is also referred to as a "push-button". A button is a rectangle or rounded rectangle.

You can make a Button using:

- Only text, as shown on the left side of the figure below.
- Only an icon, as shown in the center of the figure below.
- Both text and an icon, as shown on the right side of the figure below.

Android offers several types of buttons, including raised buttons and flat buttons as shown inthe figure below. These buttons have three states: normal, disabled, and pressed.



Designing raised buttons:
To use raised buttons that conform to the Material Design Specification, follow these steps:
1. In your build.gradle (Module: app) file, add the newest appcompat library to the dependencies section:
compile 'com.android.support:appcompat-v7:x.x.x.'
In the above, x.x.x. is the version number. If the version number you specified is lower than the currently available library version number, Android Studio will warn you ("a newer version is available"). Update the version number to the one Android Studio tells you to use.
2. Make your activity extend android.support.v7.app.AppCompatActivity :
public class MainActivity extends AppCompatActivity {
...
}
3. Use the Button element in the layout file. There is no need for an additional attribute, as a raised button is the default style.
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
... />

3b. What are the three states of a button? Explain.

[3.0] 1 [2] [1]

Scheme: 3 states 3M

Solution

Buttons are in three states: normal, disabled, and pressed.
Normal state: In its normal state, the button looks like a raised button.
Disabled state: When the button is disabled, it is grayed out and it's not active in the app's context. In most cases you would hide an inactive button, but there may be times when you would want to show it as disabled.
Pressed state: The pressed state, with a larger background shadow, indicates that the button is being touched or clicked. When you attach a callback to the button (such as the OnClick attribute), the callback is called when the button is in this state.

4a. What is spinner? explain.

[3.0] 1 [1, 2, 3] [2]

Scheme: Description 3M

Solution

Spinner: provides a quick way to select one value from a set. Touching the spinner displays a drop-down list with all available values, from which the user can select one.
If you have a long list of choices, a spinner may extend beyond your layout, forcing the user to scroll it. A spinner scrolls automatically, with no extra code needed.

To create a spinner, use the Spinner class, which creates a view that displays individual spinner values as child views, and lets the user pick one. Follow these steps:
1. Create a Spinner element in your XML layout, and specify its values using an array and an ArrayAdapter.
2. Create the spinner and its adapter using the SpinnerAdapter class.
3. To define the selection callback for the spinner, update the Activity that uses the spinner to implement the AdapterView.OnItemSelectedListener interface.

4b. How to create the spinner and its adopter? Create a spinner to wait for the background

process to complete.

[7.0] 1 [1, 2, 3] [3]

Scheme: Implementation 5M + Description 2M

Solution

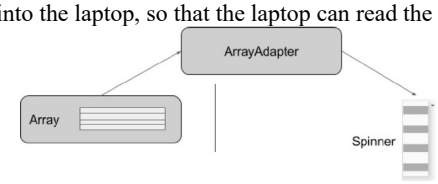
Create the spinner UI element:

To create a spinner in your XML layout, add a Spinner element, which provides the drop-down list:

```
<Spinner
android:id="@+id/label_spinner"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
</Spinner>
```

Specify the values for the spinner:

Use the adapter to fill the values for Spinner. An adapter is like a bridge, or intermediary, between two incompatible interfaces. For example, a **memory card reader** acts as an adapter between the memory card and a laptop. You plug the memory card into the card reader, and plug the card reader into the laptop, so that the laptop can read the memory card.



The SpinnerAdapter class, which implements the Adapter class, allows you to define two different views: one that shows the data values in the spinner itself, and one that shows the data in the drop-down list when the spinner is touched or clicked.

The values you provide for the spinner can come from any source, but must be provided through a SpinnerAdapter, such as an ArrayAdapter if the values are available in an array. The following shows a simple array called labels_array of predetermined values in the strings.xml file:

```
<string-array name="labels_array">
<item>Home</item>
<item>Work</item>
<item>Mobile</item>
<item>Other</item>
</string-array>
```

Create the spinner and its adapter:

Create the spinner, and set its listener to the activity that implements the callback methods. The best place to do this is when the view is created in the onCreate() method. Follow these steps (refer to the full onCreate() method at the end of the steps):

1. Add the code below to the onCreate() method, which does the following:
2. Gets the spinner object you added to the layout using findViewById() to find it by its id (label_spinner).
3. Sets the onItemSelectedListener to whichever activity implements the callbacks (this) using the setOnItemSelectedListener() method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// Create the spinner.
Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
if (spinner != null) {
spinner.setOnItemSelectedListener(this);
}
}
```

4. Also in the onCreate() method, add a statement that creates the ArrayAdapter with the string array:

```
//Create ArrayAdapter using the string array and default spinner layout.
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this, R.array.labels_array,
android.R.layout.simple_spinner_item);
```

5. Specify the layout the adapter should use to display the list of spinner choices by calling the setDropDownViewResource() method of the ArrayAdapter class. For example, you can use simple_spinner_dropdown_item as your layout:

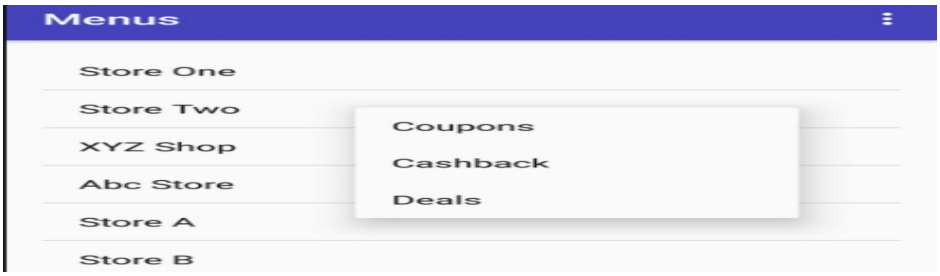
```
// Specify the layout to use when the list of choices appears.
adapter.setDropDownViewResource
(android.R.layout.simple_spinner_dropdown_item);
```

6. Use setAdapter() to apply the adapter to the spinner:

```
// Apply the adapter to the spinner.
```

```
spinner.setAdapter(adapter);
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// Create the spinner.
Spinner spinner = (Spinner) findViewById(R.id.label_spinner);
if (spinner != null) {
setOnItemSelectedListener(this);
}
// Create ArrayAdapter using the string array and default spinner layout.
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
R.array.labels_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears.
adapter.setDropDownViewResource
(android.R.layout.simple_spinner_dropdown_item);
// Apply the adapter to the spinner.
if (spinner != null) {
spinner.setAdapter(adapter);
}
}
```

5a. What are the steps required for floating context menu? Explain how to create the menu show in figure with necessary coding snippet.



[6.0] 1 [1, 2, 3] [3]

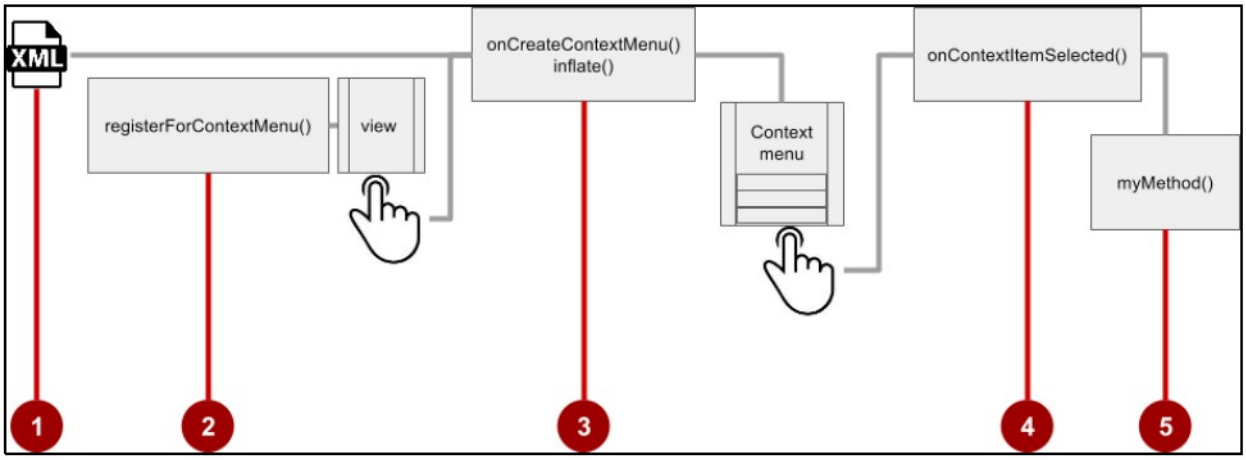
Scheme: Implementation 4M + Description 2M

Solution

Floating context menu:

Follow these steps to create a floating context menu for one or more view elements (refer to figure above):

1. Create an XML menu resource file for the menu items, and assign appearance and position attributes (as described in the previous section).



2. Register a view to the context menu using the registerForContextMenu() method of the Activity class.
3. Implement the onCreateContextMenu() method in your activity or fragment to inflate the menu.
4. Implement the onContextItemSelected() method in your activity or fragment to handle menu item clicks.

5. Create a method to perform an action for each context menu item.

5b. Distinguish between started services and bounded services

[4.0] 1 [1, 2, 3] [2]

Scheme: 2M each

Solution

Services:

A service is an application component that performs long-running operations, usually in the background. A service doesn't provide a user interface (UI). (An activity, on the other hand, provides a UI.) A service can be started, bound, or both.

A started service is a service that an application component starts by calling startService().

Use started services for tasks that run in the background to perform long-running operations. Also use started services for tasks that perform work for processes.

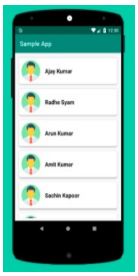
A bound service is a service that an application component binds to itself by calling bindService().

Use bound services for tasks that another app component interacts with to perform interprocess communication (IPC).

For example, a bound service might handle network transactions, perform file I/O, play music, or interact with a content provider.

Note: A service runs in the main thread of its hosting process—the service doesn't create its own thread and doesn't run in a separate process unless you specify that it should.

6. Explain Recycler View components in detail. Explain how to we create the view show in the image.



[10.0] 1 [1, 2, 3] [3]

Scheme: Description 6M + Implementation 4M

Solution

RecyclerView

When you display a large number of items in a scrollable list, most items are not visible. For example, in a long list of words or many news headlines, the user only sees a small number of list items at a time. The RecyclerView class is a more advanced and flexible version of ListView. It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.

RecyclerView components

To display your data in a RecyclerView, you need the following parts:

- Data: It doesn't matter where the data comes from. You can create the data locally, as you do in the practical, get if from a database on the device as you will do in a later practical, or pull if from the cloud.
- RecyclerView: The scrolling list that contains the list items. An instance of RecyclerView as defined in your activity's layout file to act as the container for the views.
- Layouts for one item of data. All list items look the same, so you can use the same layout for all of them. item layout has to be created separately from the activity's layout, so that one item view at a time can be created and filled with data.
- Layout manager. The layout manager handles the organization (layout) of user interface components in a view. All view groups have layout managers. For the LinearLayout, the Android system handles the layout for you. RecyclerView requires an explicit layout manager to manage the arrangement of list items contained within it. This layout could be vertical, horizontal, or a grid. The layout manager is an instance of Recyclerview. LayoutManager to organize the layout of the items in the RecyclerView.
- An adapter. The adapter connects your data to the RecyclerView. It prepares the data and how will be displayed in a view holder. When the data changes, the adapter updates the contents of the respective list item view in the RecyclerView. And an adapter is an extension of RecyclerView.Adapter. The adapter uses a ViewHolder to hold the views that constitute each item in the RecyclerView, and to bind the data to be displayed into the views that display it.
- A view holder. The view holder extends the ViewHolder class. It contains the view information for displaying one item from the item's layout.

A view holder used by the adapter to supply data, which is an extension of RecyclerView.ViewHolder.