The X Course: Android

Session 4

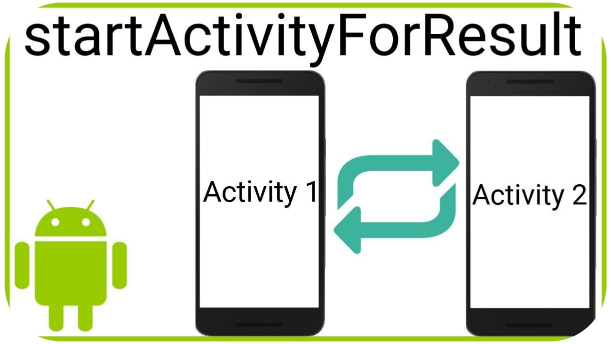
Agenda

- Getting a result back from a child activity.
- Introducing a bug due to Configuration Changes.
- Solution of the Bug: The SaveInstanceState.
- Build The Cheat Aware GeoQuiz.

Getting a result back from a child activity.

- When you want to hear back from the child activity, you call the following Activity method:

```
public void startActivityForResult(Intent intent, int requestCode)
```



The Request Code

- It is a user-defined integer that is sent to the child activity and then received back by the parent.
- It is used when an activity starts more than one type of child activity and needs to know who is reporting back.
- However, **QuizActivity** will only ever start one type of child activity.

Setting the result

```
public final void setResult(int resultCode, Intent data)
```

- Typically, the result code is one of two predefined constants: Activity.RESULT_OK or Activity.RESULT_CANCELED .
- The parent activity would take different action depending on the result code.
- If setResult(...) is not called, then when the user presses the Back button the parent will receive Activity.RESULT_CANCELED .

Sending back an intent

- **Steps:**
  - ❖ You are going to create an Intent.
  - ❖ Put an extra on it.
  - ❖ Call setResult using result code and intent.
- When the user presses the Back button to return to the QuizActivity , the ActivityManager calls:

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data)
```

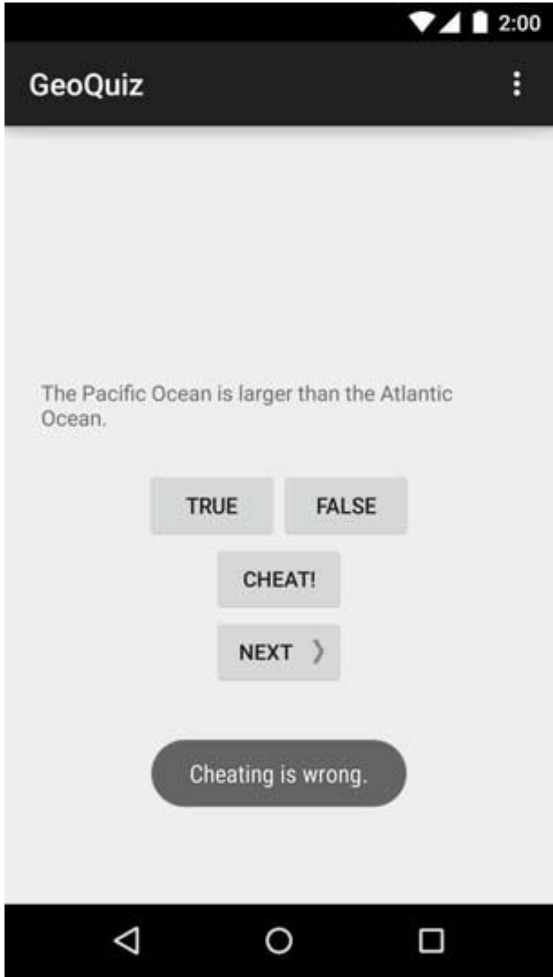
Handling a result

- Override onActivityResult(...) to retrieve the data sent from child.
- Check the request code and result code to be sure they are what you expect.
- This is a best practice to make future maintenance easier.

Now, our App knows who is cheating !

- Apply the changes and steps discussed in code.
- End result is the screen on right, a Cheat Aware geoQuiz !

  (No Next Button for now)



The Pacific Ocean is larger than the Atlantic Ocean.

Cheating is wrong.

Configuration Changes: Introducing Bugs

- There are a number of events that can trigger a configuration change.
- Perhaps the most prominent example is a change between portrait and landscape orientations.
- Other cases that can cause configuration changes include changes to language or input device.

Configuration Changes: Introducing Bugs

- When a configuration change occurs, the activity is destroyed and recreated.
- The original activity instance will have the `onPause()`, `onStop()`, and `onDestroy()` callbacks triggered.
- A new instance of the activity will be created and have the `onCreate()`, `onStart()`, and `onResume()` callbacks triggered.

Configuration Changes: Solutions to Bugs

- A user expects an activity’s UI state to remain the same throughout a configuration change.
- You should preserve the user’s transient UI state using a combination of ViewModel, onSaveInstanceState(), and/or local storage.
- Here, we learn about using the ***InstanceState*** only.

A Solution to Bugs: Instance State

- The saved data that the system uses to restore the previous state is called the *instance state* and is a collection of key-value pairs stored in a `Bundle` object.
- By default, the system uses the `Bundle` instance state to save information about each `View` object in your activity layout.
- However, your activity might have more state information that you'd like to restore, such as member variables that track the user's progress in the activity.

A Solution to Bugs: Instance State

- As your activity begins to stop, the system calls the onSaveInstanceState() method so your activity can save state information to an instance state bundle.
- When your activity is recreated after it was previously destroyed, you can recover your saved instance state from the Bundle that the system passes to your activity.
- Both the onCreate() and onRestoreInstanceState() callback methods receive that Bundle.

A Solution to Bugs: Further Look

- A Bundle object isn't appropriate for preserving more than a trivial amount of data because it requires serialization on the main thread and consumes system-process memory.
- To preserve more than a very small amount of data, you should take a combined approach to preserving data, using persistent local storage, the onSaveInstanceState() method, and the ViewModel class.

Further Readings

1. https://developer.android.com/guide/components/activities/state-changes
2. https://developer.android.com/topic/libraries/architecture/saving-states.html
3. https://developer.android.com/guide/components/activities/activity-lifecycle.html#saras
4. https://developer.android.com/guide/components/activities/parcelables-and-bundles