Observer Design Pattern

Sunday, December 13, 2020 12:24 AM

We make heavy use of the Observer design pattern by using Android Studio to develope our mobile application, because the nature of GUI allow us to take action whenever the user enters something or clicks a button:

For example, in: AttendeeMyEventActivity.java in the folder eventSystem, we have method: onCreate(), which calls setContentView(R.layout.activity_event_attendee);

Each activity instance in our app is acting as an observer to observe if a state has changed when the Activity instances transition through different states in their lifecycle: the system is creating an activity in this case.

We have the above call back method: onCreate(), which is one of the six core callbacks, others are: onStart(), onResume(), onPause(), onStop(), and onDestroy().

We have implemented the onCreate() method in each activity, because this call back fires when the system first creates the activity, on activity creation, the Created state is entered and we add basic application startup logic in this method.

For each activity, we implement onCreate(), which binds data, and associate the activity with a ViewModel, and instantiate some class-scope variables.

setContentView(R.layout.activity_event_attendee);

This line set the UI layout for this activity, the layout file is defined in the project res/layout/activity_event_attendee.xml file.

Note that on line 62:

```
protected void refreshEvents() {
    createEventMenu();
    attendeeMyEventAdapter.notifyDataSetChanged();
    super.refreshEvents();
}
```

The notifyDataSetChanged() method is defined in: androidx.recyclerview.widget.RecyclerView.Adapter: to notify any registered observers that the dataset has changed.

There are two different classes of data change events, item changes and structural changes. Item changes are when a single item has its data updated but no positional changes have occurred. Structural changes are when items are inserted, removed or moved within the data set.

This event does not specify what about the data set has changed, forcing any observers to assume that all existing items and structure may no longer be valid. LayoutManagers will be forced to fully rebind and relayout all visible views.

RecyclerView will attempt to synthesize visible structural change events for adapters that report that they have stable IDs when this method is used. This can help for the purposes of animation and visual object persistence but individual item views will still need to be rebound and relaid out.

https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.Adapter

We use this pattern to notify the adapter about the update of the data, this is an example of using the Observer design pattern, which means it gives us the way to communicate between different classes and objects. Because the cause (i.e. the observable) and the effect (i.e., the observer) belong to different classes, they're not in the same class and therefore we have encapsulated the cause from the effect.

In fact, every clickable button in our project is a subject (a.k.a. observable), and every time the user clicks on it, it changes state, and we have observers that is interested in this change of state, because we would like to provide a response to the user or check & save & update the data entered by the user.

By Jiayi Su, please feel free to contact me if you spot an error, or you think there is something to improve.

I am happy to improve it and make it better:
My email address is: sjy.su@mail.utoronto.ca