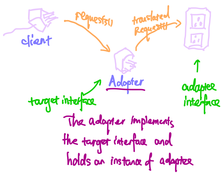
Adapter Design Pattern

Sunday, December 13, 2020 8:36 PM

From the lecture, we learned that the Adapter Design Pattern solves the problem:

We would like to reuse a class from a previous program (the Adaptee), but it doesn't work in the current situation.



In this case, we can create an adapter class which contains an instance of the Adaptee class. The adapter class implements the methods that are required for the rest of the program to interact with the class, ie. we can create an Object Adapter.

How the client uses the Adapter:

① The client makes a request to the adapter by calling a method on it using the target interface

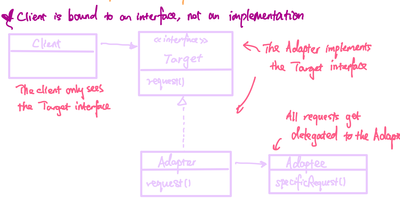
② The adapter translates the request into one or more calls on the adaptee using the adaptee interface

③ The client receives the result of the call and never knows there is an adapter doing the translation.

Note: Client and Adaptee are decoupled - neither knows about the other

The Adapter Pattern converts the interface of a class into another interface the client expect. Adapter classes work together that couldn't otherwise because of incompatible interfaces.

★ Client is bound to an interface, not an implementation



Adapter is composed with the Adaptee

★ Added advantage: we can use an adapter with any subclass of the adaptee

In our project, we make heavy use of RecyclerView library,

https://developer.android.com/guide/topics/ui/layout/recyclerview

↳ from this website, we have learned that we can supply our lists of event data and define how each event item looks, then, the RecyclerView library can dynamically creates the event items when they're needed.

It "recycles" individual element: When an item scrolls off the screen, it doesn't destroy its view! Instead, it reuses the view for new items that are scrolled onscreen currently.

Benefit: This reuse improves performance a lot, the app is more responsive and the power consumption can be reduced.

Note that each individual element in the list is defined by a view holder object. When it's created, it doesn't have any data associated with it. Later, the RecyclerView binds it to its data

& The RecyclerView requests those views, and binds the views to their data, by calling methods in the adapter

In our project, we make use of Android's RecyclerView and create many adapter classes for their corresponding activity classes.

For example, in eventSystem package, we have EventActivity.java and EventAdaptor.java.

EventActivity is a subclass of BaseActivity, and it is the superclass of, for example, AttendeeEventActivity, in it we have stored a data field of type SwipeRefreshLayout, this is used so that the user can refresh the contents of a view via an vertical swipe gesture.

EventAdapter is an abstract class extending RecyclerView.Adapter It does not know what is Event, but it simply handles the data and sends the configuration to the correct ViewHolder to display.

In line 116 We define a static class VHEvent extends RecyclerView.ViewHolder, this is our ViewHolder class

These two classes: Adapter and ViewHolder, work together to define how the event's data is displayed.

The ViewHolder is a wrapper around a View that contains the layout for an individual item in the list.

line 116... in EventAdapter.java:

```
static class VHEvent extends RecyclerView.ViewHolder {
    CardView cardView;
    ImageView eventImage;
    TextView eventTitle;

    public VHEvent(View v) {
        super(v);
        cardView = (CardView) v;
        eventImage = v.findViewById(R.id.event_imagey);
        eventTitle = v.findViewById(R.id.event_title);
    }
}
```

This is the reason why we can implement the Adapter Design Pattern for different classes, it'll work in all situations because it's not interact directly with the thing we want to adapt, in this case, Event. Instead, it's interacting with the adapter, and the adapter will make sure the thing that it's adapting will work.

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