16. Multiple object instances of different types need to be able to communicate with each other

Imagine that you have a container object that contains other objects of various types. To make it clearer, imagine that the container object is the background of a user interface, while other objects, such as text boxes, buttons, labels, etc. are the components of it.

There are certain events that certain objects need to emit that should be able to affect other objects. For example, clicking a particular button should be able to change text on specific labels.

The last thing you would want to do in this situation is connect the components directly to each other. If you do that, your code will become excessively complex. It's especially true when you have many elements inside the placeholder and they need to be connected together in arbitrary many-to-many relationships.

Suitable design patterns

Mediator

Mediator is a class that acts as a communication medium between objects when they don't have a way of communicating with each other directly.

An analogy would be an air traffic control. Airplanes that take off and land at an airport don't communicate directly with each other. But air traffic control sees them all and issues appropriate instructions to each of them based on what other airplanes are doing.

In the example with the user interface, the background layout itself may act as a mediator. Once any of the controls on it emits a specific type of event, the layout class will decide whether this event is something that any other element needs to be aware of and it will notify that element accordingly.

So, none of the elements care about what happens after the event has been emitted. Deciding what to do with the event is the sole responsibility of a single Mediator class. And if any other elements are affected, the Mediator will issue instructions to them accordingly.

Why would you use Mediator

- The sole responsibility of facilitating communication between different objects is taken up by a single class.
- Individual components are much easier to re-use.
- Open-closed principle is easy to enforce.

Observer

The Observer design pattern utilizes the concept of publishing and subscribing. Basically, any elements that emit a specific event are publishers and any elements that need to react to the event are subscribers.

So, if you want a specific object to be affected by specific event types or events with specific data values, you can subscribe that object to those events. If you no longer need to get that object to react to those events, you can unsubscribe.

And an object doesn't have to be restricted to being either a publisher or a subscriber. It can be both. Likewise, an object can be a publisher and/or a subscriber for more than one event type.

Why would you use Observer

- You can easily establish relationships between objects at runtime.
- Easy to enforce open-closed principle.
- You can implement it across application boundaries.