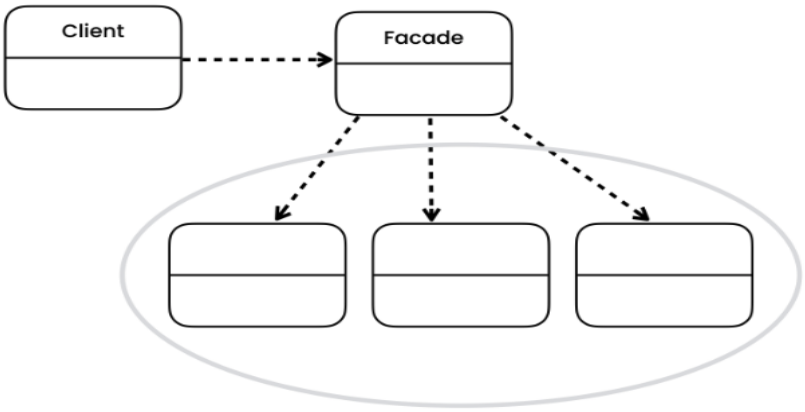
Facade Pattern:

Definition: Provides a simplified, higher-level interface to a subsystem. Clients can talk to the facade (face/front) rather than individual classes in the subsystem.



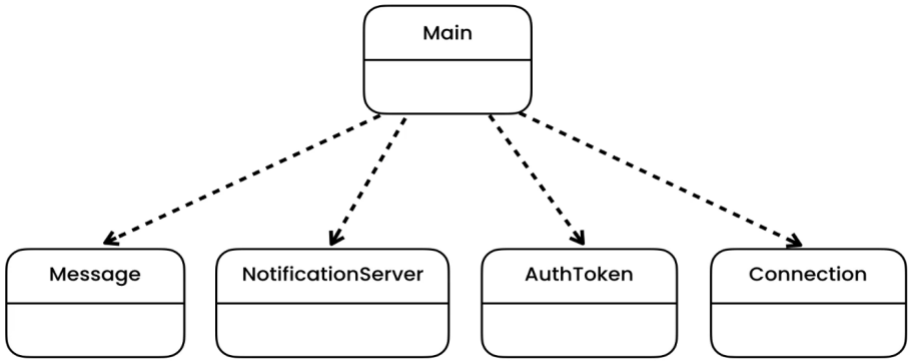
Scenario: We want to implement a notification server which implements:

1. Connection: Connects the client to the server.
2. Authentication: Authorizes the client.
3. Sending: Sends the message to the target machine.
4. Disconnection: Disconnects the client.

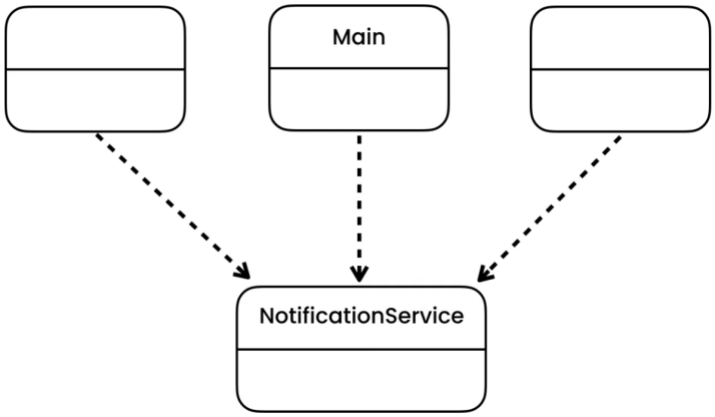
Implementation: In main class, we can make calls like

```
var server = new NotificationServer();
var connection = server.connect("ip");
var authToken = server.authenticate("appId", "key");
var message = new Message("Hello World");
server.send(authToken, message, "target");
connection.disconnect();
```

Problem: We have to follow many steps in a specified order on client. Later if anything changes, it will make breaking changes on client. Secondly, there is too much coupling of classes with client. And client can also make errors by calling authentication before connection.



Solution: Don't give any control to the client. Implement a facade (a new class), who is responsible for all these action and client only talks with that class.




```
public class NotificationService {
    public void Send(String message, String target){
        var server = new NotificationServer();
        var connection = server.connect("ip");
        var authToken = server.authenticate("appId", "key");
        server.send(authToken, new Message(message), target);
        connection.disconnect();
    }
}
```

On client, we can just call

```
var notificationService = new NotificationService();
notificationService.Send("Hello World", "Android");
```