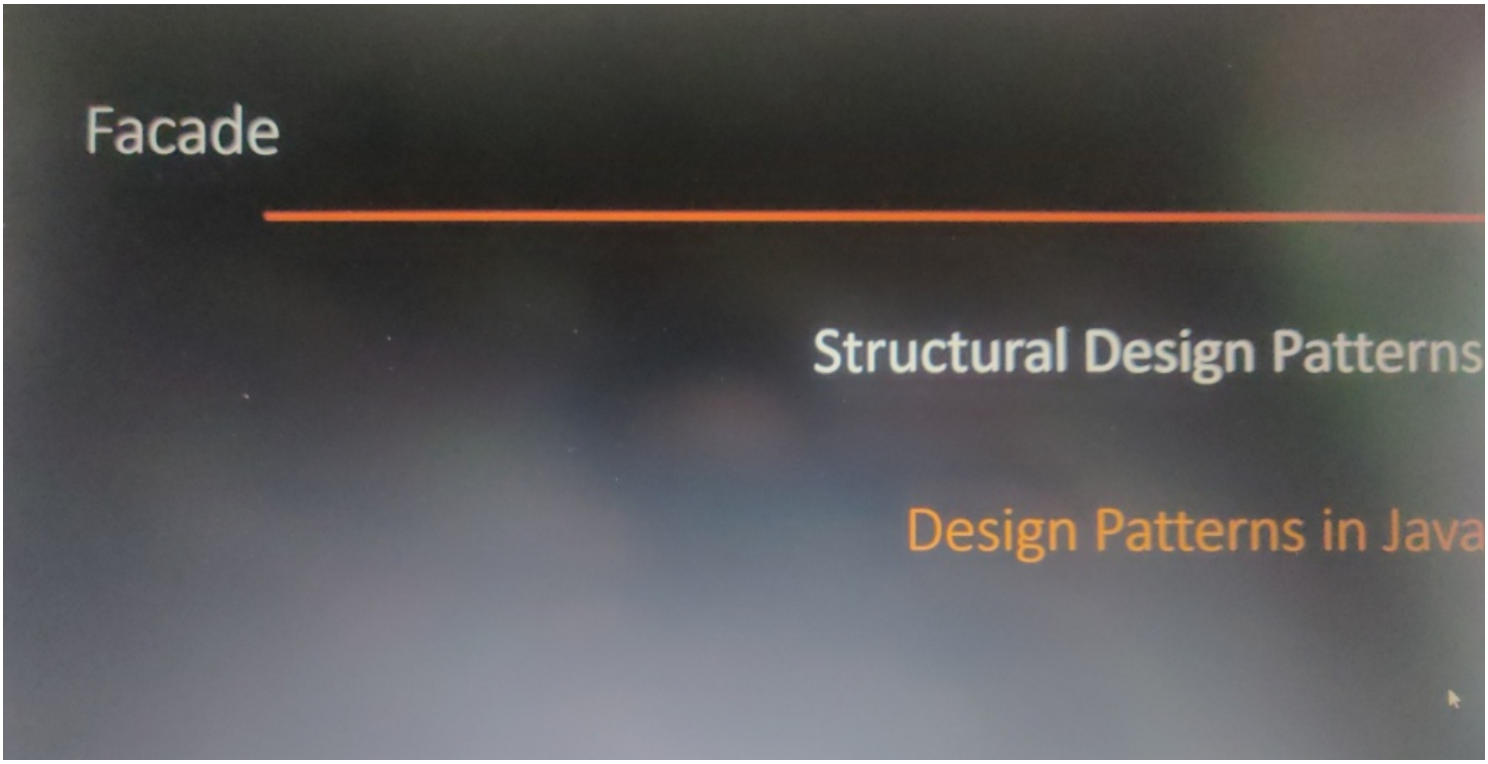
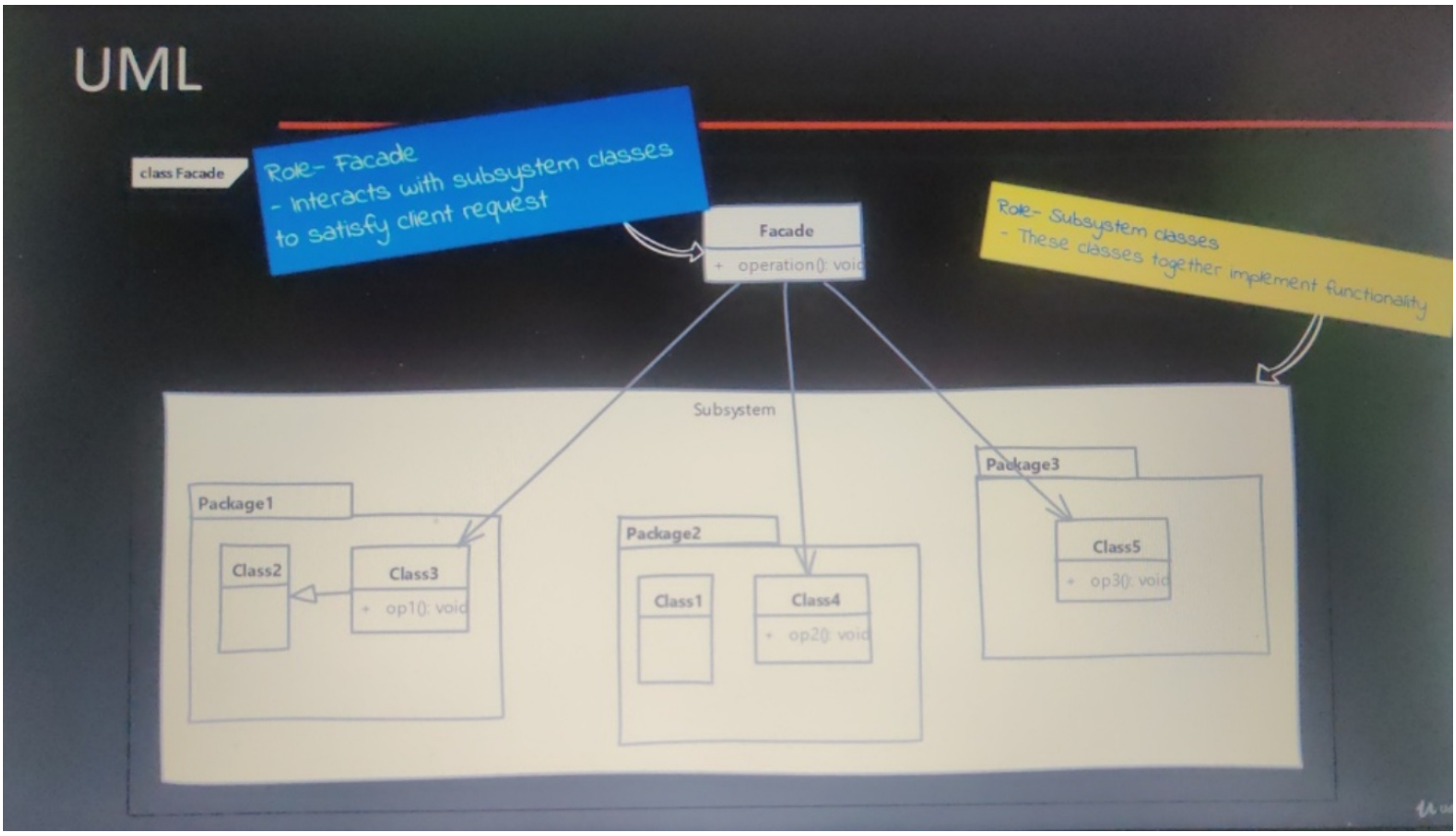

Facade
Structural Design Patterns
Design Patterns in Java

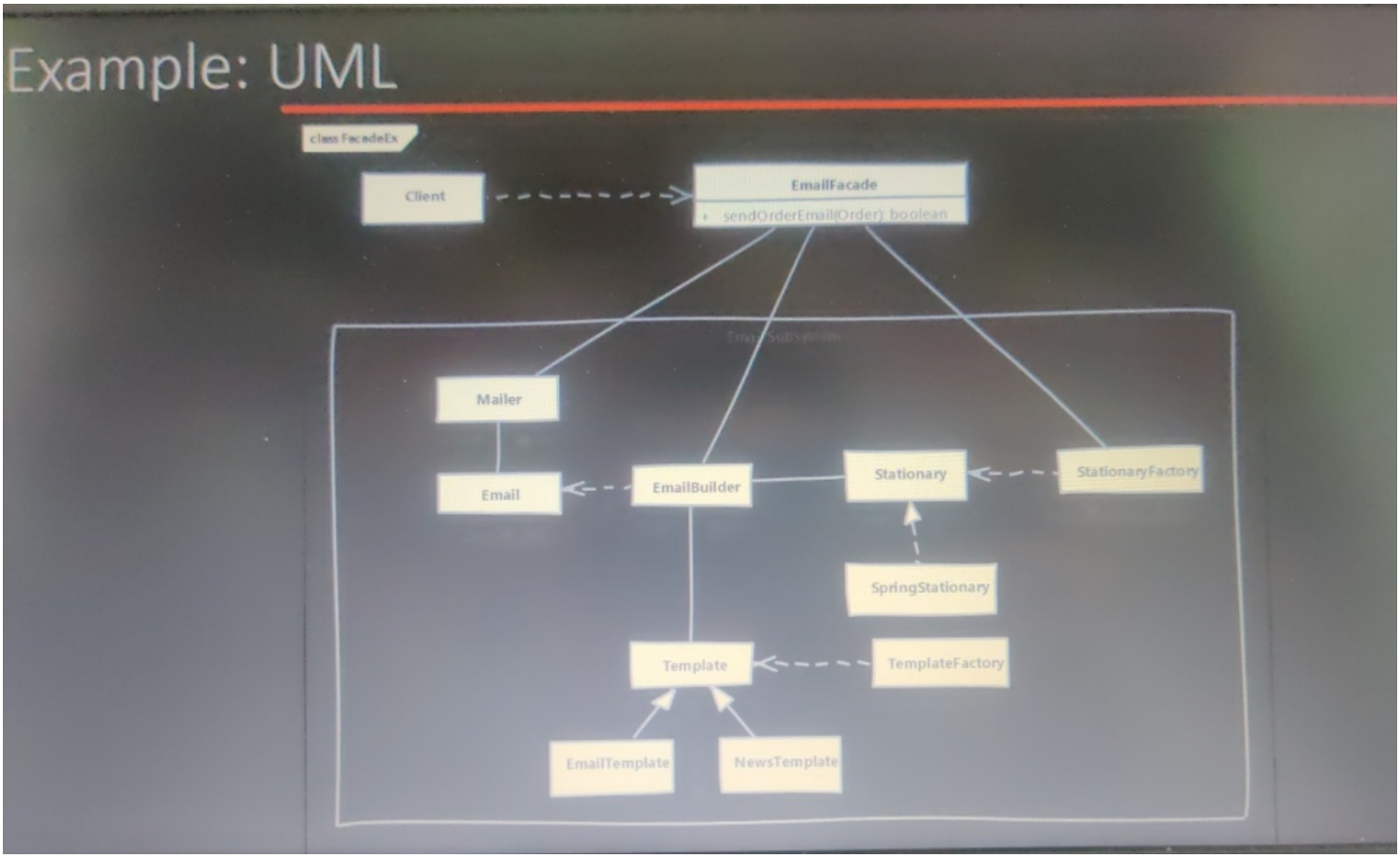
What is Facade?

- Client has to interact with a large number of interfaces and classes in a subsystem to get result. So client gets tightly coupled with those interfaces & classes. Façade solves this problem.
- Façade provides a simple and unified interface to a subsystem. Client interacts with just the façade now to get same result.
- Façade is NOT just a one to one method forwarding to other classes.


UML
class Facade
Role- Facade
- Interacts with subsystem classes to satisfy client request
Facade
+ operation(): void
Role- Subsystem classes
- These classes together implement functionality
Subsystem
Package1
Class2
Class3
+ op1(): void
Package2
Class1
Class4
+ op2(): void
Package3
Class5
+ op3(): void

Implement a Facade

- We start by creating a class that will serve as a facade
  - We determine the overall “use cases”/tasks that the subsystem is used for.
  - We write a method that exposes each “use case” or task.
  - This method takes care of working with different classes of subsystem.


Example: UML
class FacadeEx
Client
EmailFacade
+ sendOrderEmail(Order): boolean
Mailer
Email
EmailBuilder
Stationary
StationaryFactory
SpringStationary
Template
TemplateFactory
EmailTemplate
NewsTemplate

Implementation Considerations

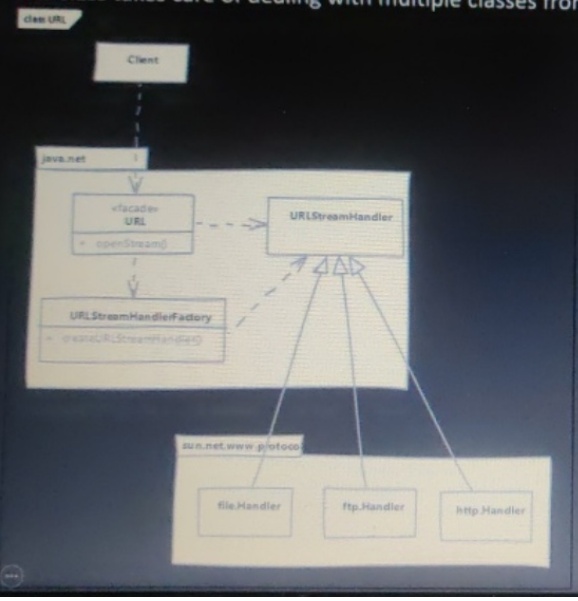
- A façade should minimize the complexity of subsystem and provide usable interface.
- You can have an interface or abstract class for façade and client can use different subclasses to talk to different subsystem implementations.
- A façade is not replacement for regular usage of classes in subsystem. Those can be still used outside of façade. Your subsystem class implementations should not make assumptions of usage of façade by client code.

Design Considerations

- Façade is a great solution to simplify dependencies. It allows you to have a weak coupling between subsystems.
- If your *only* concern is coupling of client code to subsystem specific classes and not worried about simplification provided by a façade, then you can use abstract factory pattern in place of façade.

Example of a Facade

- The java.net.URL class is a great example of façade. This class provides a simple method called as openStream() and we get an input stream to the resource pointed at by the URL object.
- This class takes care of dealing with multiple classes from the java.net package as well as some internal sun packages.

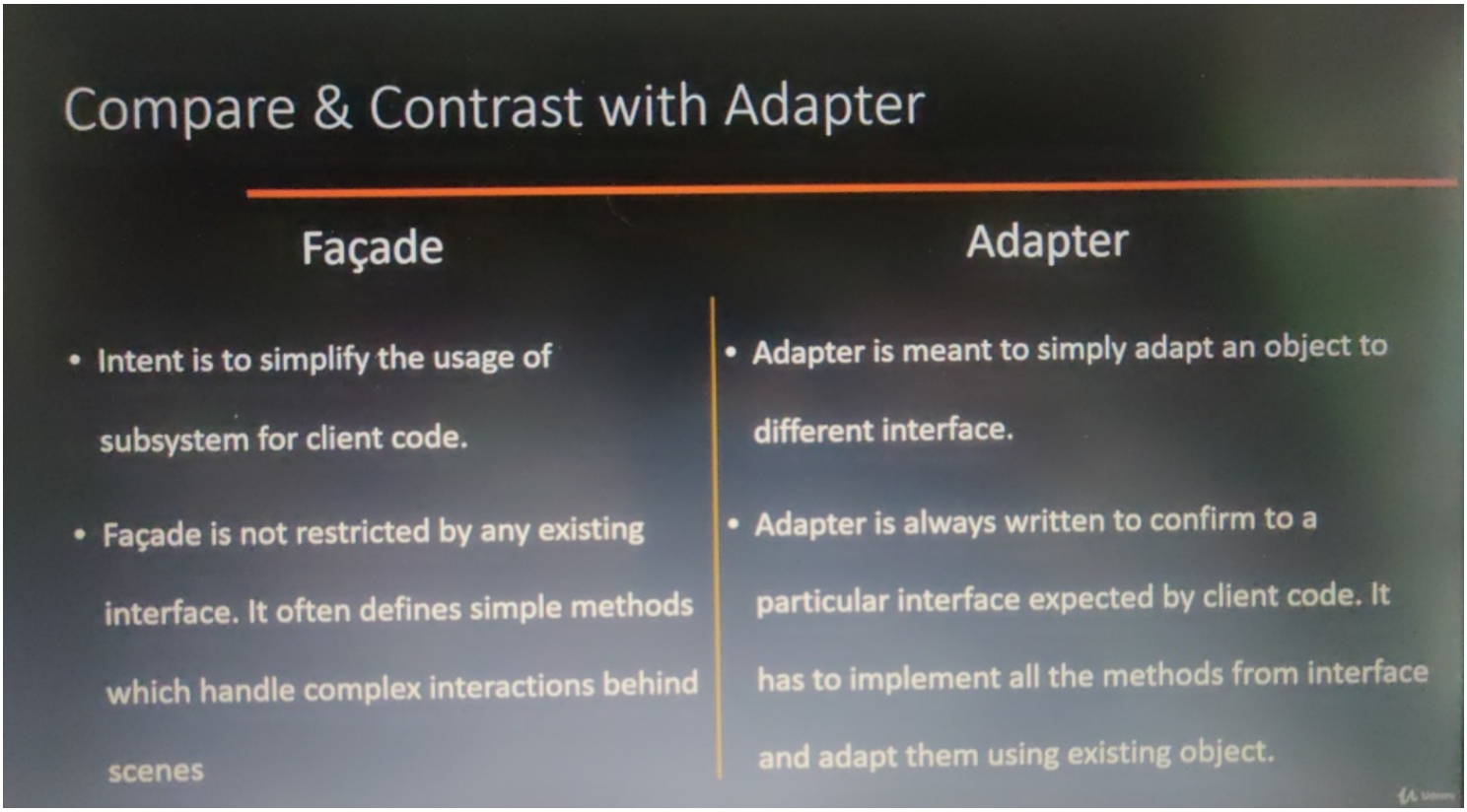



```
//create URL
URL url = new URL("http://google.com");
//open stream
InputStream remoteStream = url.openStream();

//Read from stream
BufferedReader rd = new BufferedReader(new InputStreamReader(remoteStream));
```

URL.class

```
static URLStreamHandler getURLStreamHandler(String protocol) {

    URLStreamHandler handler = handlers.get(protocol);
    if (handler == null) {

        boolean checkedWithFactory = false;

        // Use the factory (if any)
        if (factory != null) {
            handler = factory.createURLStreamHandler(protocol);
            checkedWithFactory = true;
        }

        // Try java protocol handler
        if (handler == null) {
            String packagePrefixList = null;

            packagePrefixList
                = java.security.AccessController.doPrivileged(
```

Code taken from URL.class of rt.jar


Compare & Contrast with Adapter
Façade
Intent is to simplify the usage of subsystem for client code.
Façade is not restricted by any existing interface. It often defines simple methods which handle complex interactions behind scenes
Adapter
Adapter is meant to simply adapt an object to different interface.
Adapter is always written to confirm to a particular interface expected by client code. It has to implement all the methods from interface and adapt them using existing object.

Pitfalls

- Not a pitfall of the pattern itself but needing a façade in a new design should warrant another look at API design.
- It is often overused or misused pattern & can hide improperly designed API. A common misuse is to use them as "containers of related methods". So be on the lookout for such cases during code reviews.

In-A-Hurry Summary

- We use façade when using our subsystem requires dealing with lots of classes & interfaces for client. Using façade we provide a simple interface which provides same functionality.
- Façade is not a simple method forwarding but façade methods encapsulate the subsystem class interactions which otherwise would have been done by client code.
- Facades are often added over existing legacy codes to simplify code usage & reduce coupling of client code to legacy code.