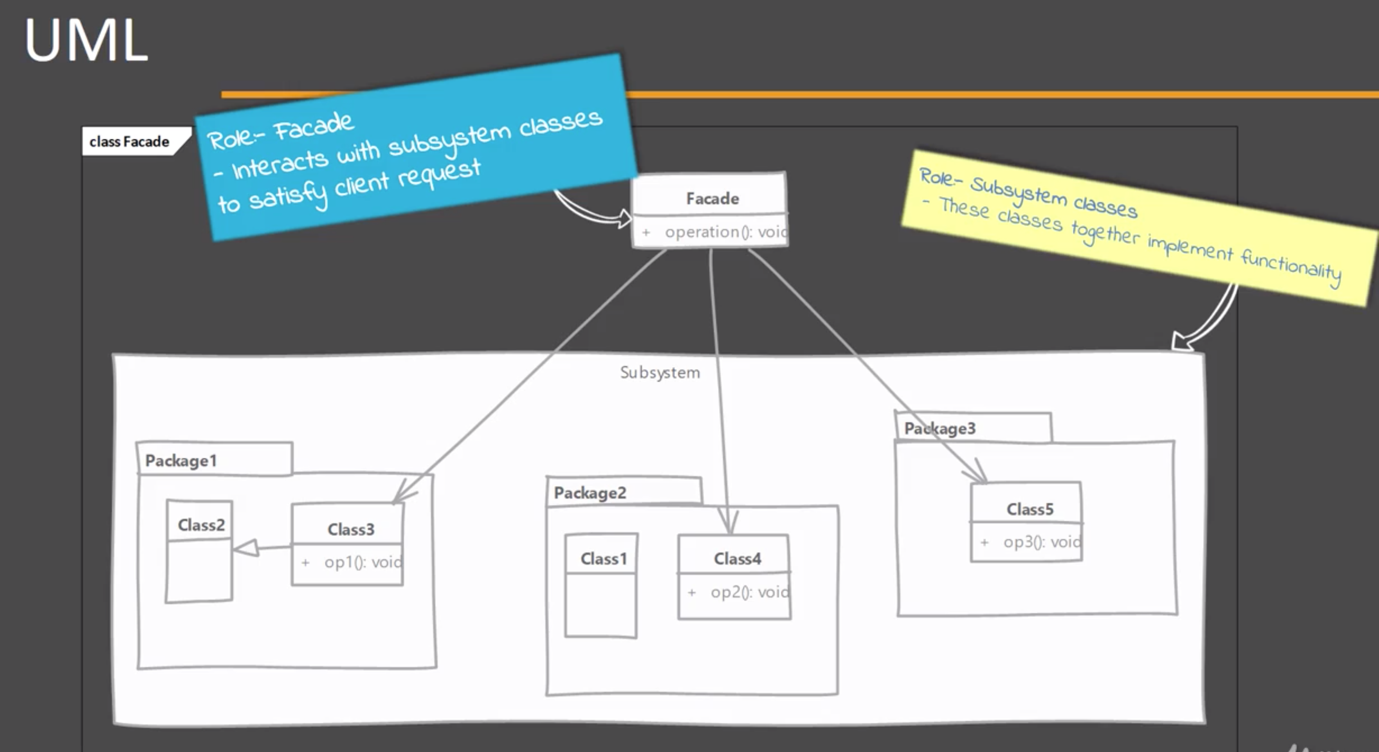
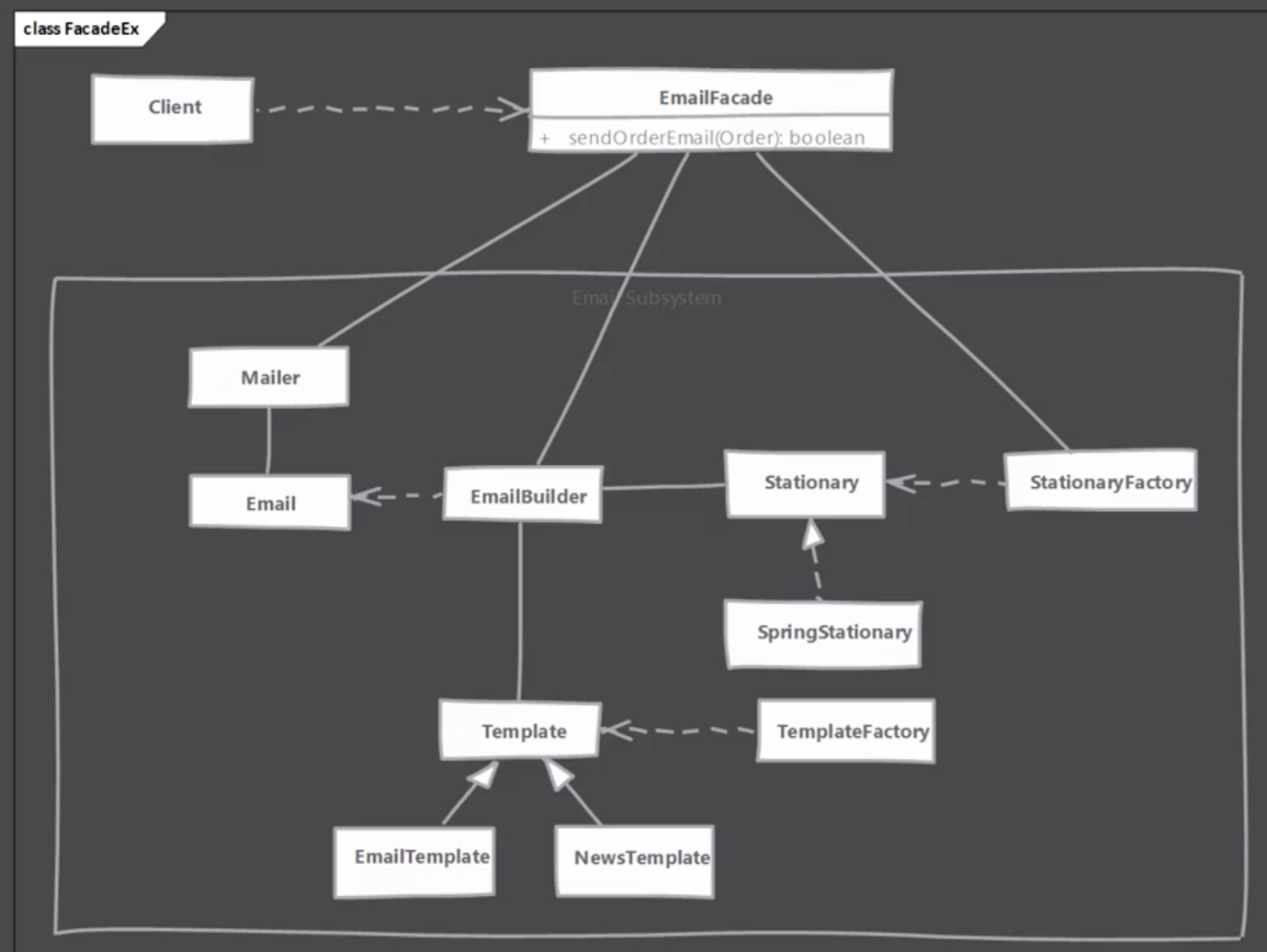
**Facade Pattern**

* **Introduction:**
  + Client has to interact with a large number of interfaces and classes in a sub-system to get result. So client gets tightly coupled with those interfaces and 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:**

****

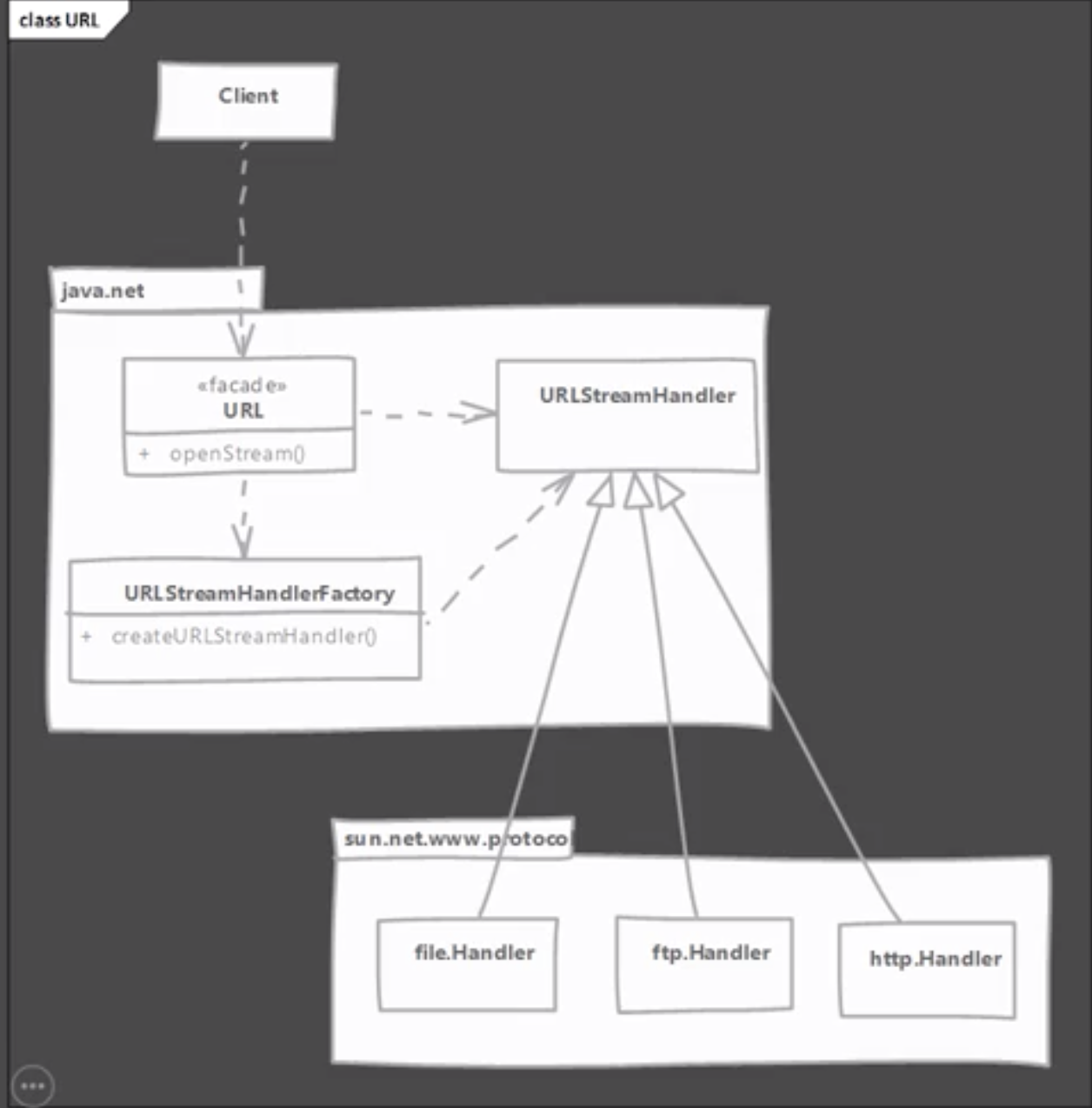
* **Implementation Steps:**
  + We start by creating a class that will serve as façade.
    - We determine the overall “use cases” / tasks that the sub-system is used for.
    - We write a method that exposes each “use case” or task.
    - This method takes care of working with different classes of subsystem.
* **Example UML:**

****

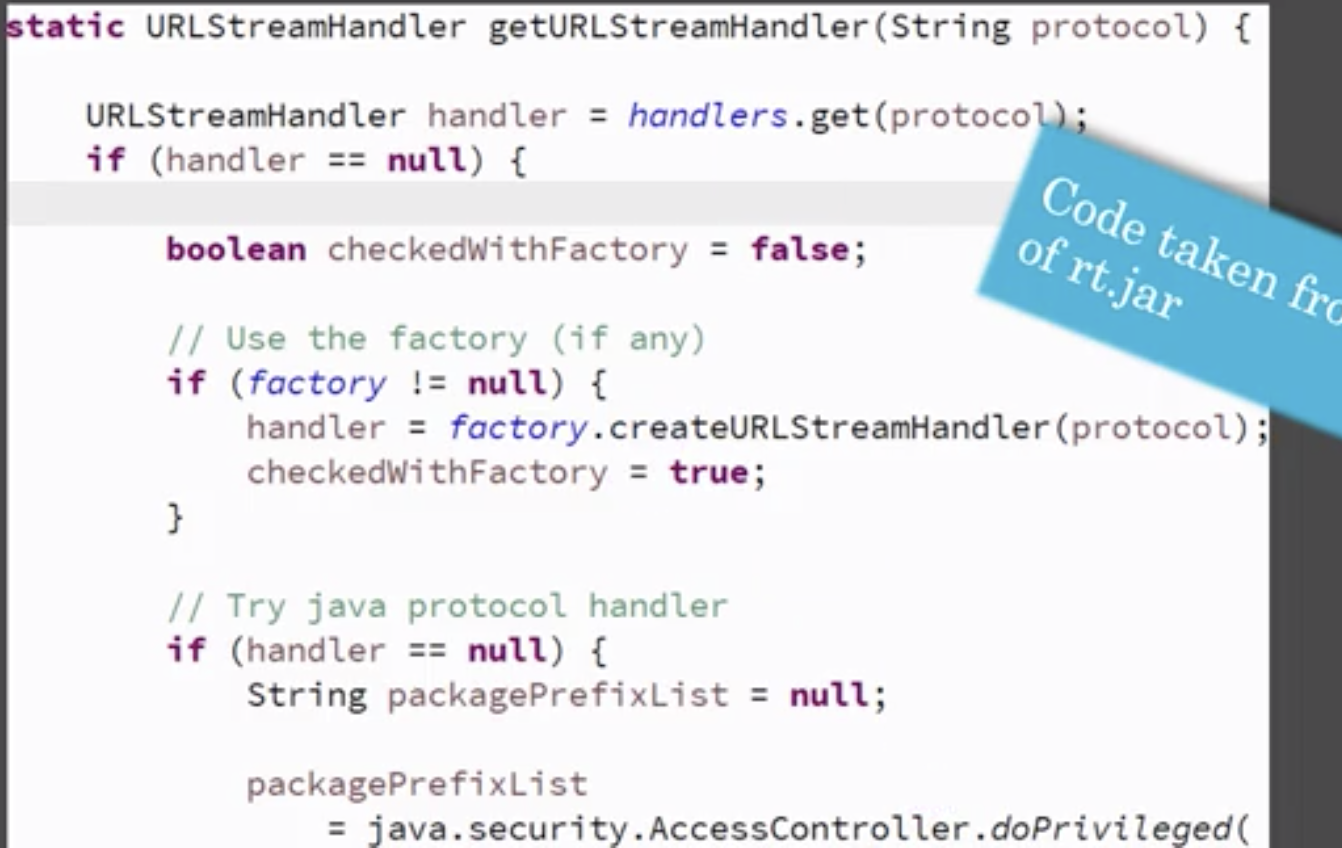
* **Implementation and Design Consideration:**
  + A façade should minimize the complexity of sub-system and provide usable interface.
  + You can have an interface or abstract class for façade and client can use different subclasses to talk to different subsystems.
  + A façade is not a replacement 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 Consideration:**

* + 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 simplifications provided by a façade, then you can use abstract factory pattern in place of façade.
* **Example:**
  + The java.net.URL class is the 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.

****

****

****

* **Comparison with Adapter:**

|  |  |
| --- | --- |
| **Façade** | **Adapter** |
| Intent is to simplify the usage of subsystem from client code. | Adapter is meant to simply adapt an object to different interface. |
| Façade is not restricted by any existing interface. It often defines simple methods which handle complex interactions behind scenes. | 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.
* **Quiz:**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**