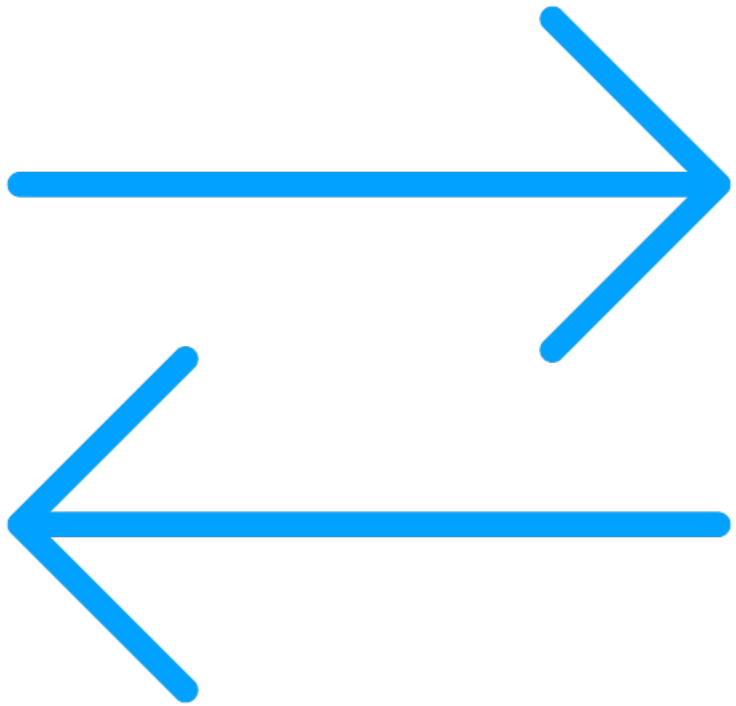
Proxy Design Pattern

Bryan Hansen

@bh5k

Proxy



Concepts

Interface by wrapping

Can add functionality

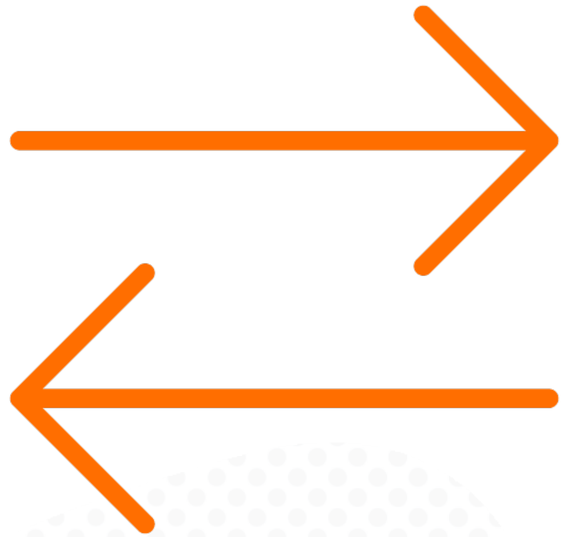
Security, Simplicity, Remote, Cost

Proxy called to access real object

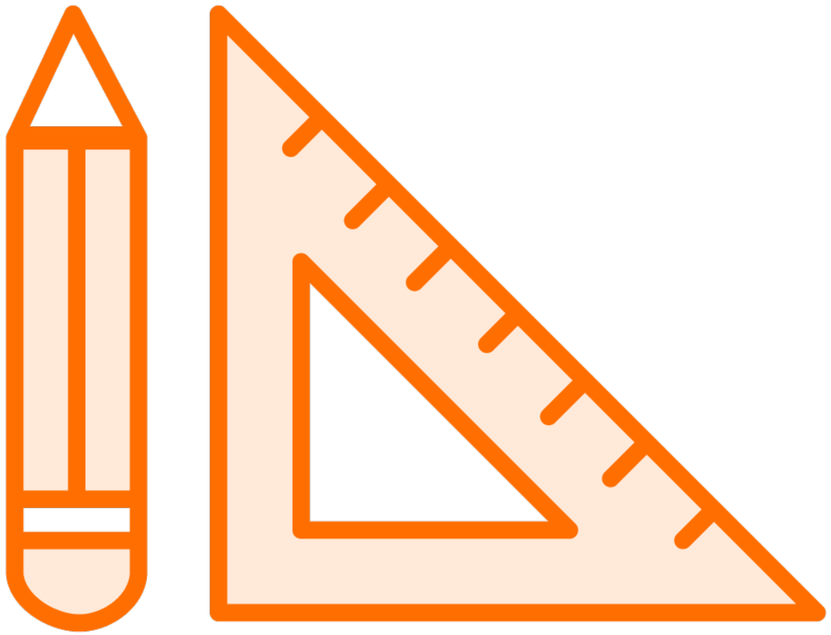
Examples:

java.lang.reflect.Proxy

java.rmi.*



Design



Interface based

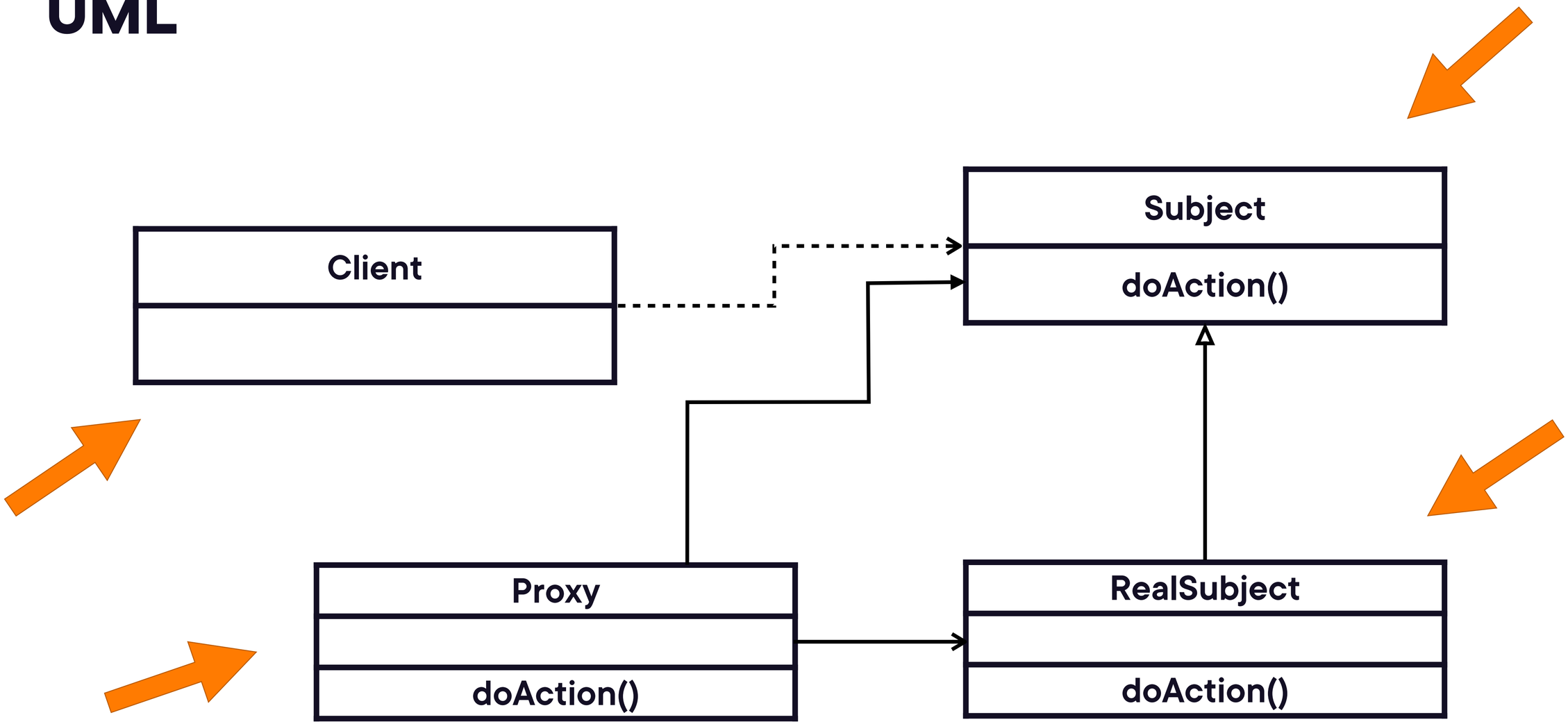
Interface and Implementation Class

java.lang.reflect.InvocationHandler

java.lang.reflect.Proxy

Client, Interface, InvocationHandler, Proxy, Implementation

UML


Client
Subject
doAction()
Proxy
doAction()
RealSubject
doAction()

Everyday Example - Integer

```
CustomService proxyService = (CustomService)
Proxy.newProxyInstance(
        customService.getClass().getClassLoader(),
        new Class[] {CustomService.class},
        customHandler);

proxyService.doServiceCall();
```

Exercise Proxy

Image Proxy

Security Proxy

Pitfalls

Only one Proxy

Another Abstraction

Similar to other patterns

Contrast

Proxy	VS	Decorator
Can add functionality, but not its main purpose		Dynamically add functionality
Can only have one		Chained
Compile time		Decorator points to its own type
		Runtime

Proxy Summary

Great utilities built into Java API

Only one instance

Used by DIJ/IoC Frameworks