CMPE 202

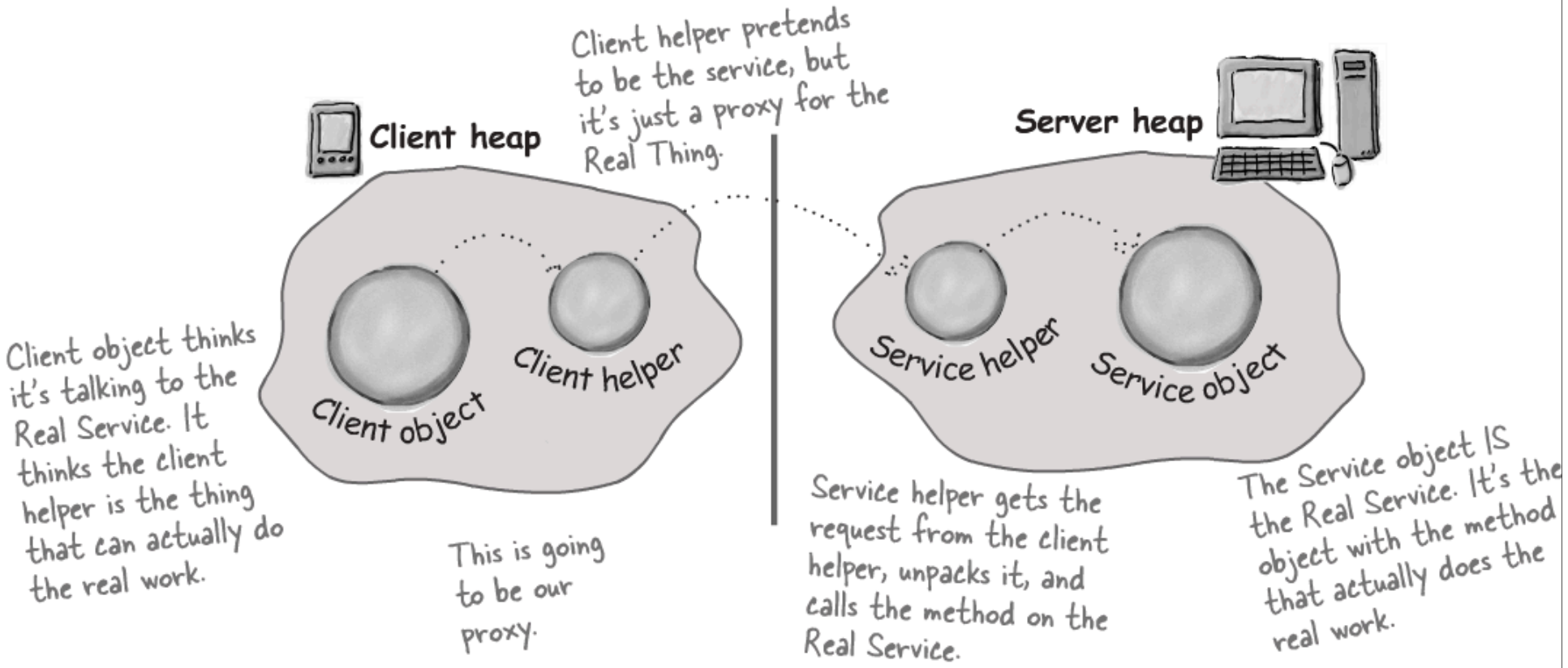
Gang of Four Design Patterns

Proxy

Motivation

- Want to check access to an object before instantiating it
- Want to have a stand-in for an object to mask expensive operations over a network

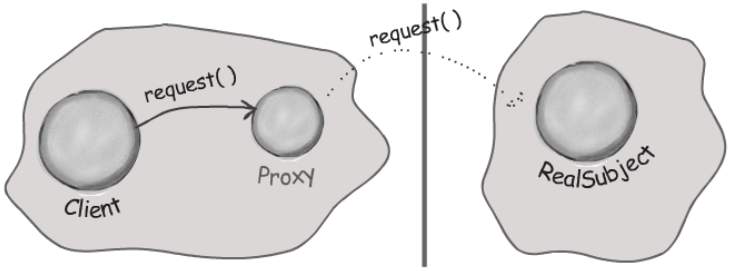
Remote Proxy



Different Types of Proxies

Remote Proxy

With Remote Proxy, the proxy acts as a local representative for an object that lives in a different JVM. A method call on the proxy results in the call being transferred over the wire, invoked remotely, and the result being returned back to the proxy and then to the Client.



Firewall Proxy controls access to a set of network resources, protecting the subject from "bad" clients.

Caching Proxy provides temporary storage for results of operations that are expensive. It can also allow multiple clients to share the results to reduce computation or network latency.

Copy-On-Write Proxy controls the copying of an object by deferring the copying of an object until it is required by a client. This is a variant of the Virtual Proxy.

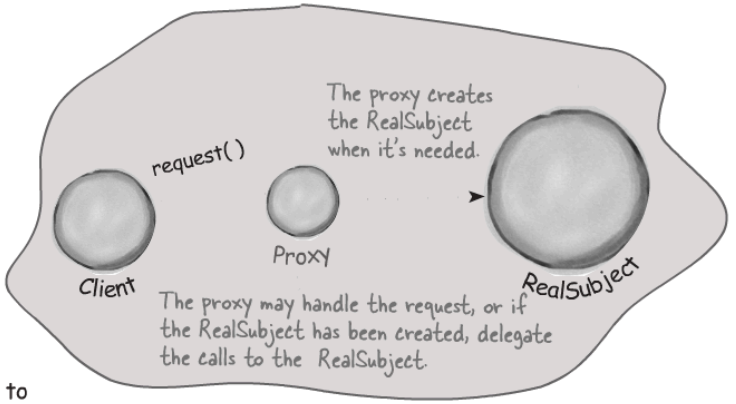
Smart Reference Proxy provides additional actions whenever a subject is referenced, such as counting the number of references to an object.

Synchronization Proxy provides safe access to a subject from multiple threads.

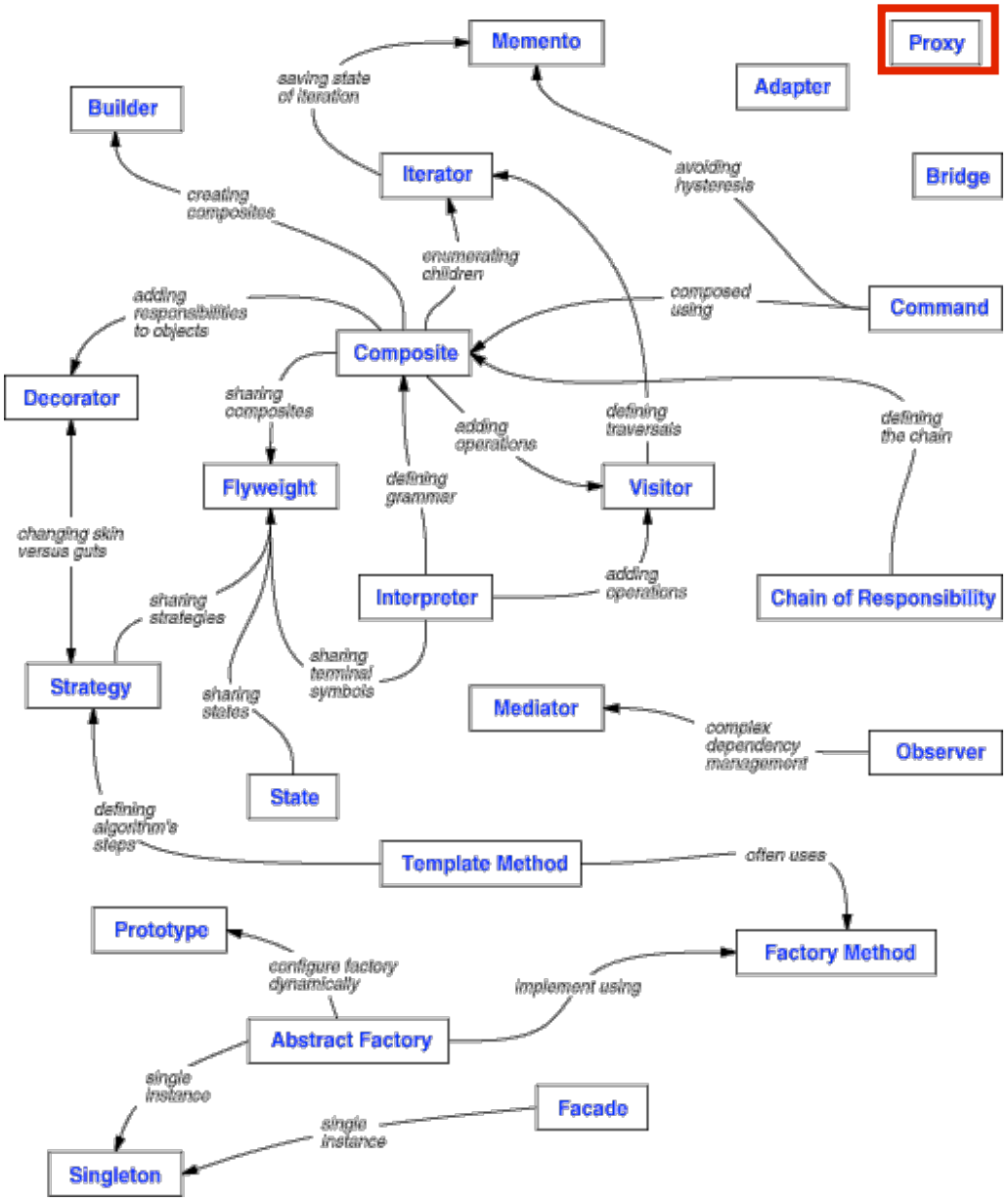
Complexity Hiding Proxy hides the complexity of and controls access to a complex set of classes. This is sometimes called the Facade Proxy for obvious reasons. The Complexity Hiding Proxy differs from the Facade Pattern in that the proxy controls access, while the Facade Pattern just provides an alternative interface.

Virtual Proxy

Virtual Proxy acts as a representative for an object that may be expensive to create. The Virtual Proxy often defers the creation of the object until it is needed; the Virtual Proxy also acts as a surrogate for the object before and while it is being created. After that, the proxy delegates requests directly to the RealSubject.



Proxy



		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

Design Pattern Catalog

Purpose	Design Pattern	Aspect(s) That Can Vary
Creational	Abstract Factory (87)	families of product objects
	Builder (97)	how a composite object gets created
	Factory Method (107)	subclass of object that is instantiated
	Prototype (117)	class of object that is instantiated
	Singleton (127)	the sole instance of a class
Structural	Adapter (139)	interface to an object
	Bridge (151)	implementation of an object
	Composite (163)	structure and composition of an object
	Decorator (175)	responsibilities of an object without subclassing
	Facade (185)	interface to a subsystem
	Flyweight (195)	storage costs of objects
	Proxy (207)	how an object is accessed; its location
Behavioral	Chain of Responsibility (223)	object that can fulfill a request
	Command (233)	when and how a request is fulfilled
	Interpreter (243)	grammar and interpretation of a language
	Iterator (257)	how an aggregate's elements are accessed, traversed
	Mediator (273)	how and which objects interact with each other
	Memento (283)	what private information is stored outside an object, and when
	Observer (293)	number of objects that depend on another object; how the dependent objects stay up to date
	State (305)	states of an object
	Strategy (315)	an algorithm
	Template Method (325)	steps of an algorithm
	Visitor (331)	operations that can be applied to object(s) without changing their class(es)

Intent

Provide a surrogate or placeholder for another object to control access to it.

Applicability

Proxy is applicable whenever there is a need for a more versatile or sophisticated reference to an object than a simple pointer. Here are several common situations in which the Proxy pattern is applicable:

1. A **remote proxy** provides a local representative for an object in a different address space. NEXTSTEP [Add94] uses the class NXProxy for this purpose. Coplien [Cop92] calls this kind of proxy an "Ambassador."

3. A **virtual proxy** creates expensive objects on demand. The ImageProxy described in the Motivation is an example of such a proxy.

5. A **protection proxy** controls access to the original object. Protection proxies are useful when objects should have different access rights. For example, KernelProxies in the Choices operating system [CIRM93] provide protected access to operating system objects.

7. A **smart reference** is a replacement for a bare pointer that performs additional actions when an object is accessed. Typical uses include

    - counting the number of references to the real object so that it can be freed automatically when there are no more references (also called **smart pointers** [Ede92]).
    - loading a persistent object into memory when it's first referenced.
    - checking that the real object is locked before it's accessed to ensure that no other object can change it.

Participants

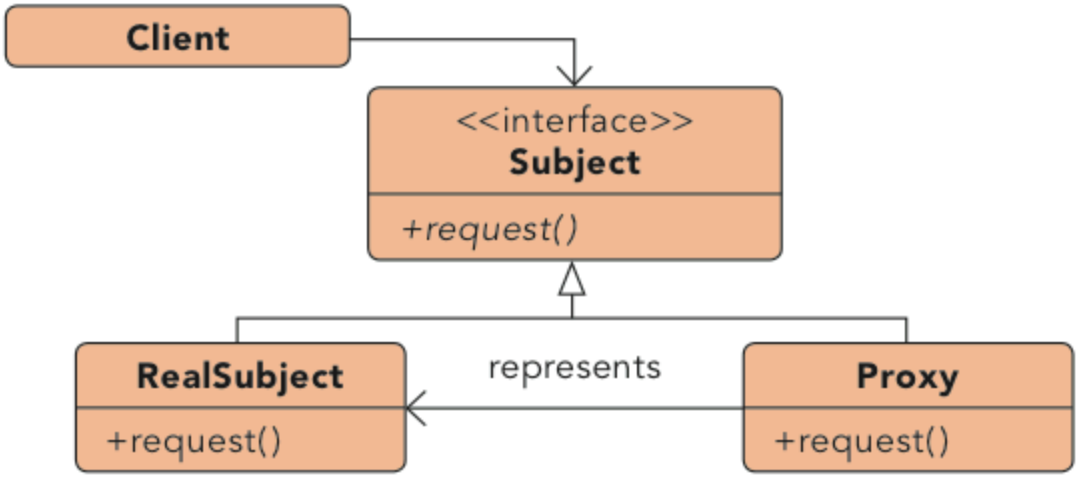
- **Proxy** (ImageProxy)
    - maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same.
    - provides an interface identical to Subject's so that a proxy can by substituted for the real subject.
    - controls access to the real subject and may be responsible for creating and deleting it.
    - other responsibilities depend on the kind of proxy:
        - *remote proxies* are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
        - *virtual proxies* may cache additional information about the real subject so that they can postpone accessing it. For example, the ImageProxy from the Motivation caches the real image's extent.
        - *protection proxies* check that the caller has the access permissions required to perform a request.
- **Subject** (Interface)
    - defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.
- **RealSubject** (Image)
    - defines the real object that the proxy represents.

Collaborations

- Proxy forwards requests to RealSubject when appropriate, depending on the kind of proxy.

PROXY

Object Structural

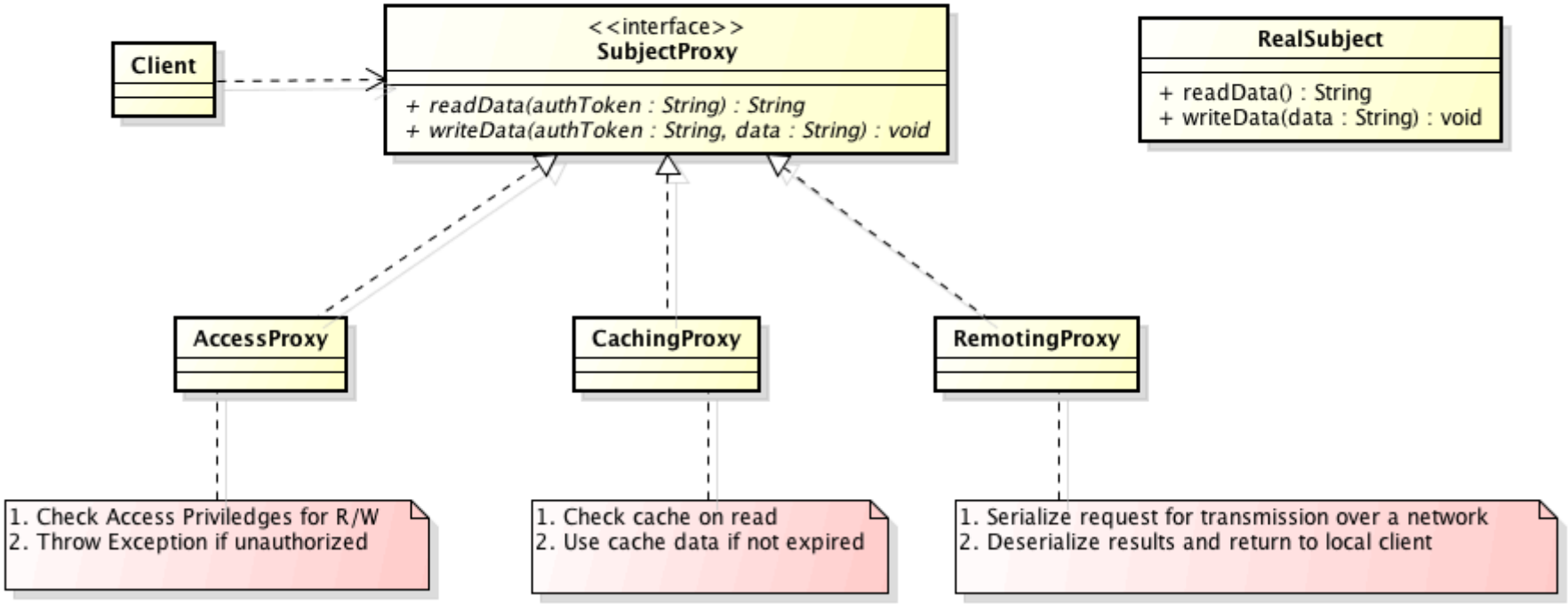


Purpose

Allows for object level access control by acting as a pass through entity or a placeholder object.

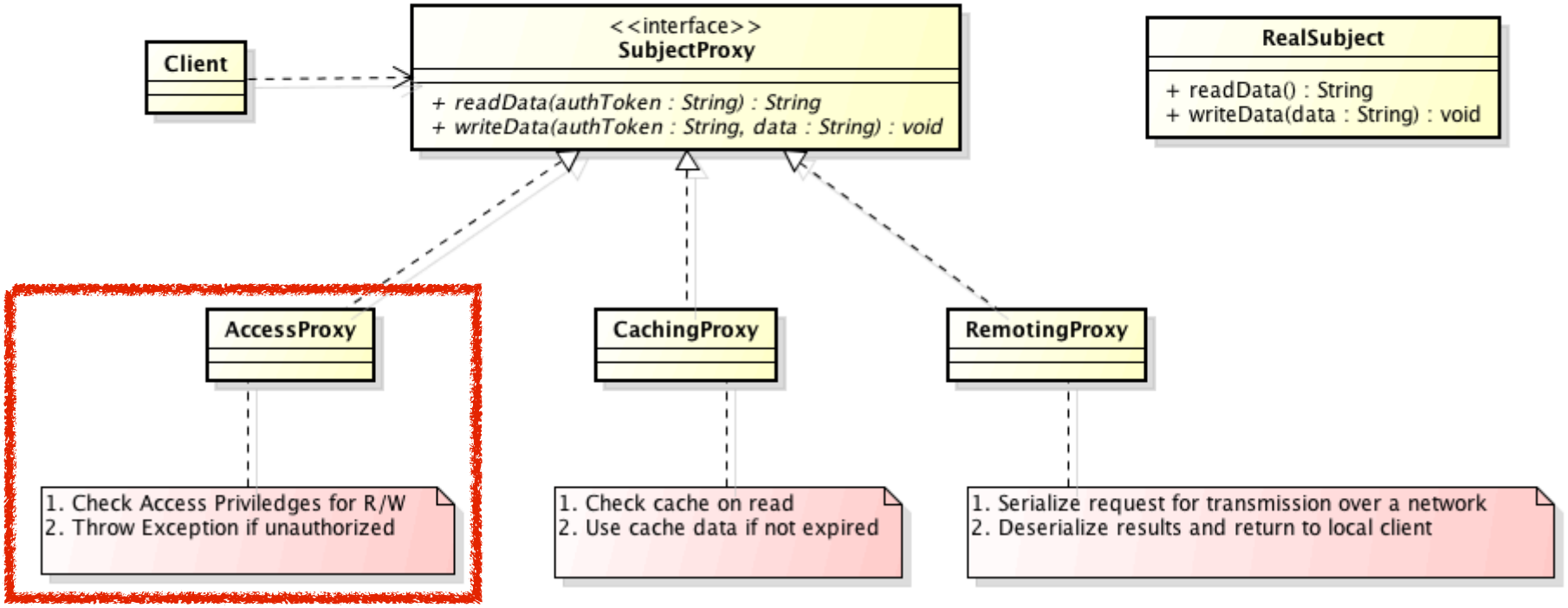
Use When

- The object being represented is external to the system.
- Objects need to be created on demand.
- Access control for the original object is required.
- Added functionality is required when an object is accessed.

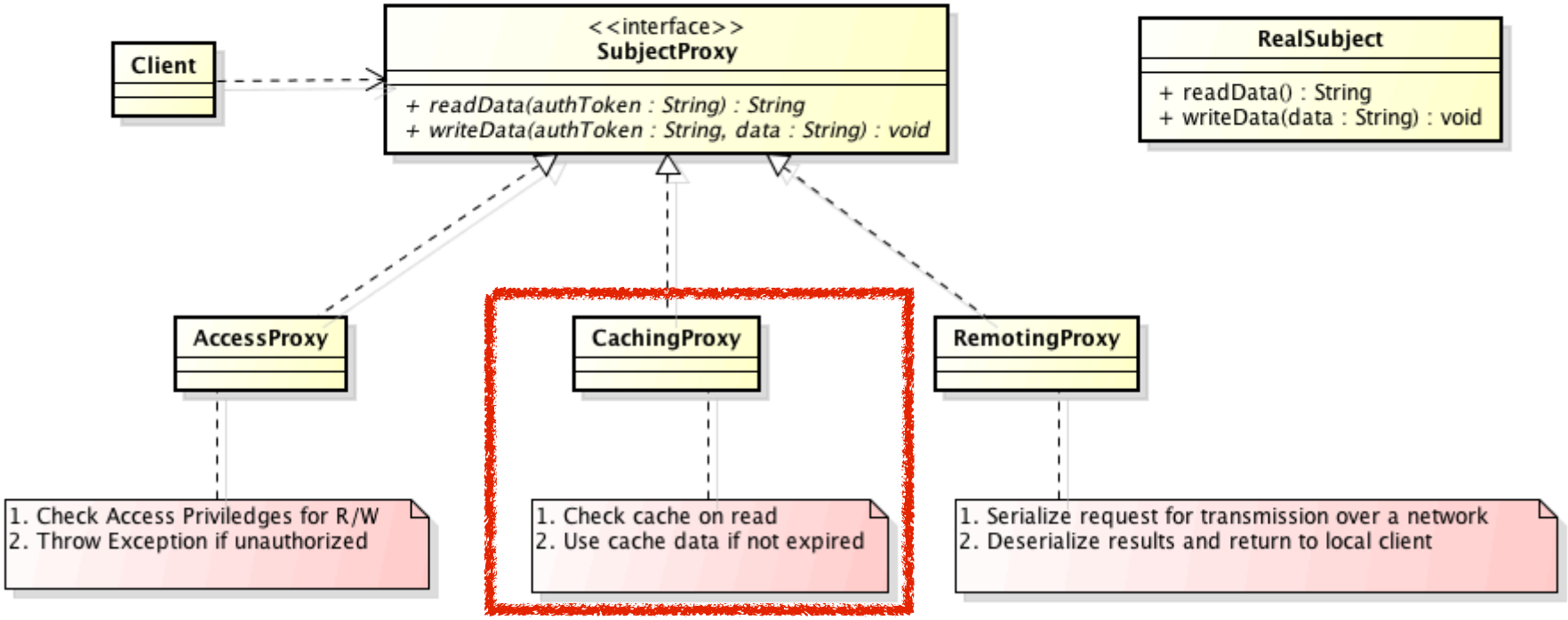



```java
public class Client {

    private SubjectProxy proxy1 = new AccessProxy() ;
    private SubjectProxy proxy2 = new CachingProxy() ;
    private SubjectProxy proxy3 = new RemotingProxy() ;

    public void runTest() {
        try {
            // read
            System.out.println( proxy1.readData("1234567890") );
            System.out.println( proxy2.readData("1234567890") );
            System.out.println( proxy3.readData("1234567890") );
            // cache hit
            System.out.println( proxy2.readData("1234567890") );
            // write
            proxy1.writeData("1234567890", "New Data");
            proxy2.writeData("1234567890", "New Data");
            proxy3.writeData("1234567890", "New Data");
            // access violation
            proxy1.readData("0000000000");
        }
        catch (Exception e)
        {
            System.out.println( e.getMessage() );
        }
    }
}
```

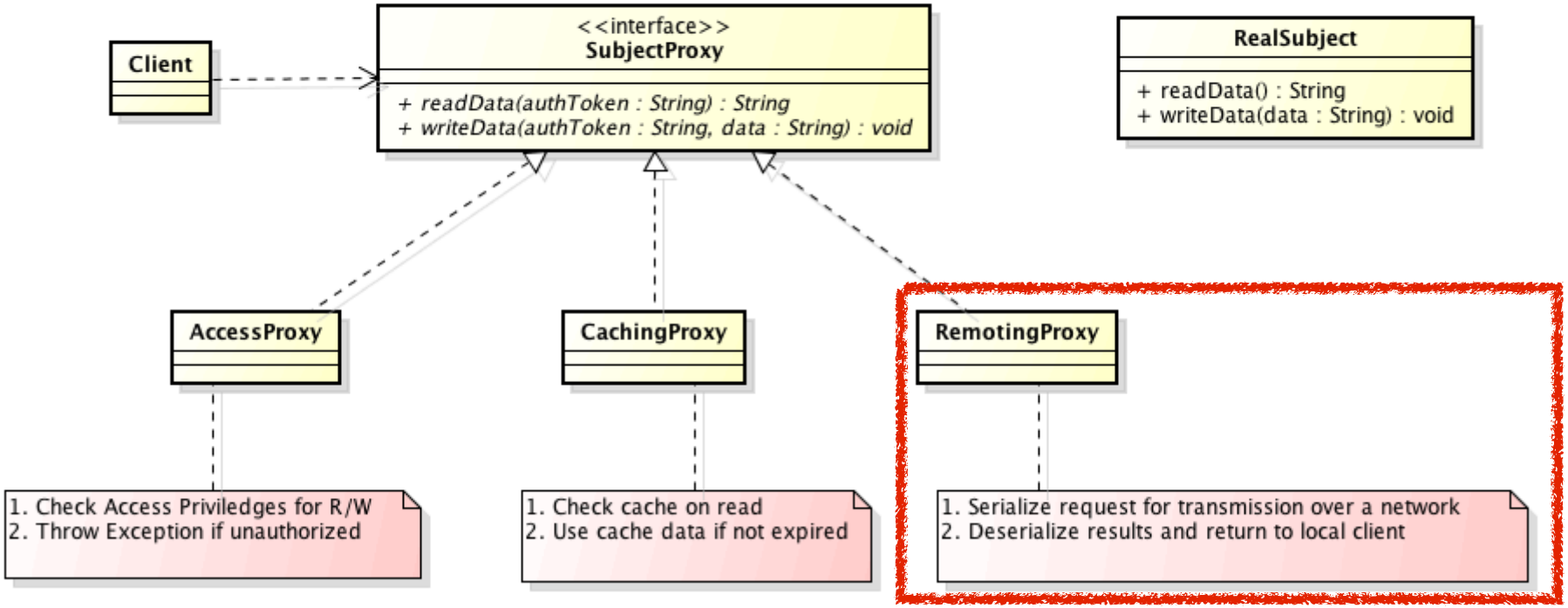
```java
public class RealSubject {

    String theData ;

    public RealSubject()
    {
        theData = "Hello World!" ;
    }

    /**
     * @see proxy.SubjectProxy#readData(java.lang.String)
     */
    public String readData() {
        return theData ;
    }

    /**
     * @see proxy.SubjectProxy#writeData(java.lang.String
     */
    public void writeData(String data) {
        theData = data ;
    }

}
```




```java
public class AccessProxy implements SubjectProxy {

    RealSubject subject = new RealSubject() ;

    /**
     * @see proxy.SubjectProxy#readData(java.lang.String)
     */
    public String readData(String authToken) throws Exception {
        if ( authToken.equalsIgnoreCase("1234567890") )
            return subject.readData() ;
        else
            throw new Exception( "Unauthorized Access" ) ;
    }

    /**
     * @see proxy.SubjectProxy#writeData(java.lang.String, java.lang.String)
     */
    public void writeData(String authToken, String data) throws Exception {
        if ( authToken.equalsIgnoreCase("1234567890") )
            subject.writeData(data);
        else
            throw new Exception( "Unauthorized Access" ) ;
    }

}
```




```java
public class CachingProxy implements SubjectProxy {

    RealSubject subject = new RealSubject();
    String dataCache = null;
    Boolean cacheExpired = false;

    /**
     * @see proxy.SubjectProxy#readData(java.lang.String)
     */
    public String readData(String authToken) throws Exception {
        if (dataCache != null && !cacheExpired) {
            System.out.println("Cache hit! Returning cached data.");
            return dataCache;
        }
        else {
            dataCache = subject.readData();
            cacheExpired = false;
            return dataCache;
        }
    }

    /**
     * @see proxy.SubjectProxy#writeData(java.lang.String, java.lang.String)
     */
    public void writeData(String authToken, String data) throws Exception {
        subject.writeData(data);
        dataCache = data;
        cacheExpired = false;
    }
}
```




```
public class RemotingProxy implements SubjectProxy {

    private class HttpClient {
        public String get( String url ) { return "The Remote Data!" ; }
        public void put( String url, String data ) { System.out.println( "Remote Host Update Processed!");}
    }
    // subject is remote!  access will be via HTTP PUT/GET
    HttpClient httpClient = new HttpClient() ;

    /**
     * @see proxy.SubjectProxy#readData(java.lang.String)
     */
    public String readData(String authToken) throws Exception {
        return httpClient.get( "http://remoteserver/data/id/12345" ) ;
    }

    /**
     * @see proxy.SubjectProxy#writeData(java.lang.String, java.lang.String)
     */
    public void writeData(String authToken, String data) throws Exception {
        httpClient.put( "http://remoteserver/data/id/12345", data);
    }
}
```

Resources for this Tutorial