Proxy

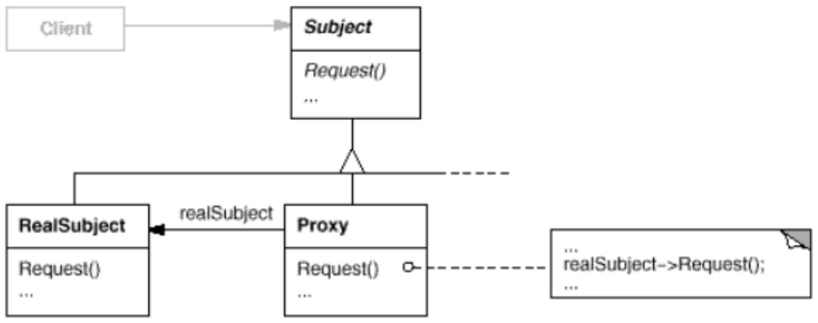

Figure 1: “UML of Proxy Design Pattern”

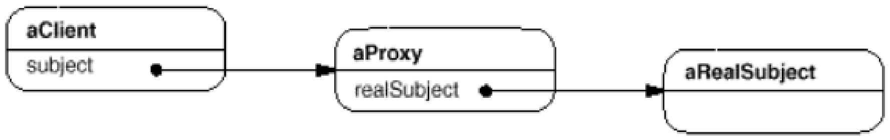

Figure 2: “Example of a UML instance diagram of the Proxy Design Pattern”

Intent

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

Applicability

1. **remote proxy**: used when we want to provide a representative for an object in a different address space.
2. **virtual proxy**: used when we want to create expensive objects on demand.
3. **protection proxy**: controls access to the original object, which is useful when objects should have different access rights.
4. **smart reference**:
   - a replacement for a bare object pointer, that does additional actions when an object is accessed
   - operations include:
     1. 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).
     2. loading a persistent object into memory when it’s first referenced.

3. checking that the real object is locked before it's accessed to ensure that no other object can change it.

Participants

1. `Proxy` (*concrete class*)
    - maintains a reference that lets it access the `RealSubject`.
    - may refer to `Subject` if `Subject` and `RealSubject` share the same interface.
    - provides an identical interface to to that of `Subject` so that `Proxy` can be substituted for `RealSubject`.
    - controls access to `RealSubject` and may be responsible for creating/deleting it.
    - depending on its kind, other responsibilities may incur:
        1. **remote proxies**: responsible for encoding a request and its arguments and sending the encoded request to `RealSubject` residing in a different address space.
        2. **virtual proxies**: may cache additional information about `RealSubject` so that they can postpone accessing it.
        3. **protection proxies**: check that the caller of `RealSubject` has the required permissions to access it.
2. `Subject` (*abstract class/interface*): defines the common interface for `RealSubject` and `Proxy` so that `Proxy` can be used anywhere `RealSubject` is expected.
3. `RealSubject` (*concrete class*): defines the real object that `Proxy` represents.

Collaborations and UML interaction diagram

1. `Proxy` forwards requests to `RealSubject` when appropriate, depending on the kind of `Proxy`.

Pros

Indirection when accessing an object, depending on the kind of `Proxy`

1. **remote proxies**: can hide the fact that an object resides in a different address space.
2. **virtual proxies**: can perform optimizations such as creating an object on demand.
3. **protections proxies** and **smart references**: can perform additional operations on an object when it's accessed.

Copy-on-write

1. **fact**: copying a large object can be expensive.

2. **observation**: if the object is not modified, then there's no need to incur this cost.
3. **consequence**:
    - using this pattern postpones the copying process to ensure that we pay the price for copying the object only if it's modified.
    - this reduces the cost of copying heavyweight subjects significantly.
4. **implementation**:
    - the subject must be reference counted.
    - copying the proxy will only increment the references counter.
    - only when the client requests an operation that modifies the subject, does the proxy actually copy it, and it decrements the reference counter afterwards.
    - when the reference count goes to zero, the subject gets deleted.

Implementation issues

(*read the book*)

Example

```java
package structural.proxy;

/**
 * an Internet interface that plays the role of Subject
 * in the Proxy design pattern.<br/>
 * It provides an interface for connecting to the Internet
 * that we want to limit access to using a proxy.
 * @author anonbnr
 */
public interface Internet {

    /* METHODS */
    /**
     * Connects to serverHost
     * @param serverHost An Internet host to which we wish to connect
     */
    void connectTo(String serverHost);
}
```

```java
package structural.proxy;

/**
 * a RealInternet concrete class that plays the role of RealSubject
 * in the Proxy design pattern.<br/>
 * It implements the Internet interface to allow access to the Internet.
 * @author anonbnr
 *
 */
public class RealInternet implements Internet {

    /* METHODS */
    @Override
    public void connectTo(String serverHost) {
        System.out.println("Connecting to " + serverHost);
    }
}

package structural.proxy;

import java.util.ArrayList;
import java.util.List;

/**
 * a ProxyInternet concrete class that plays the role of Proxy
 * in the Proxy design pattern.<br/>
 * It provides a proxy to classes implementing Internet, particularly
 * to ban Internet connections to some hosts.
 * @author anonbnr
 *
 */
public class ProxyInternet implements Internet {

    /* ATTRIBUTES */
    /**
     * The proxied Internet connection
     */
    private Internet internet;

    /**
     * The list of banned sites
     */
    private static List<String> bannedSites;

    static {
        bannedSites = new ArrayList<>();
        bannedSites.add("abc.com");
        bannedSites.add("def.com");
        bannedSites.add("ijk.com");
        bannedSites.add("lnm.com");
    }

    /* METHODS */
```

```java
    /**
     * Only allows the proxied Internet connection to connect to hosts
     * that are not in the banned sites, otherwise denies access to the host.
     * It also creates the Internet connection, only if it hasn't already
     * been created
     */
    @Override
    public void connectTo(String serverHost) {
        if (bannedSites.contains(serverHost))
            System.err.println("Access Denied to " + serverHost);

        else {
            if (internet == null)
                internet = new RealInternet();

            internet.connectTo(serverHost);
        }
    }
}

package structural.proxy;

/**
 * a Test class for the Proxy Design pattern
 * @author anonbnr
 *
 */
public class Test {
    public static void main(String[] args) {
        Internet internet = new ProxyInternet();
        internet.connectTo("abc.com"); // Access Denied to abc.com
        internet.connectTo("google.com"); // Connecting to google.com
    }
}
```