Proxy Pattern

Chapter Content

InterBit Training & Consulting Ltd.

- Proxy Pattern Overview
- Proxy Pattern UML Diagram
- Example: synchronization
- RMI Stubs
- Deferred initialization
- Java 1.3 Dynamic proxies
- Dynamic proxy – the proxy

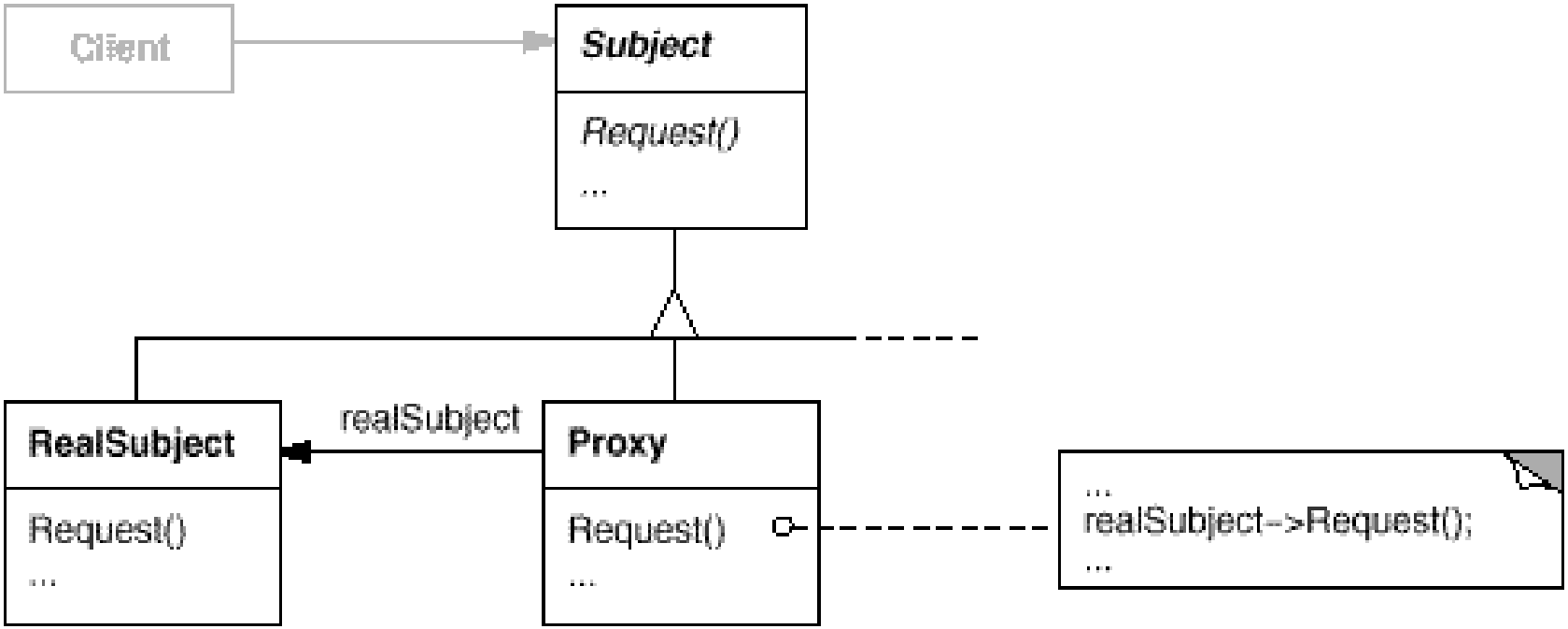
Proxy Pattern Overview

InterBit Training & Consulting Ltd.

- A proxy controls access to another object.
  - Has the same interface as the object it hides.
  - Client communicates with the proxy, which in turn forwards requests to the original object.
- Common usages:
  - Deferred initialization, transparent to client.
  - Validate access permissions.
  - Stubs communicating with remote objects.

Proxy Pattern UML Diagram

InterBit
Training & Consulting Ltd.



Client communicates with RealSubject through a proxy (rather than directly).

No major changes required in client code: Proxy implements the same interface as RealSubject.

Types of Proxies

InterBit
Training & Consulting Ltd.

- ***Remote Proxy*** - Provides a reference to an object located in a different address space on the same or different machine
- ***Virtual Proxy*** - Allows the creation of a memory intensive object on demand. The object will not be created until it is really needed.
- ***Copy-On-Write Proxy*** - Defers copying (cloning) a target object until required by client actions. Really a form of virtual proxy.
- ***Protection (Access) Proxy*** (*Firewall Proxy*) - Provides different clients with different levels of access to a target object

Types of Proxies (cont.)

InterBit Training & Consulting Ltd.

- ***Cache Proxy*** - Provides temporary storage of the results of expensive target operations so that multiple clients can share the results
- ***Synchronization Proxy*** - Provides multiple accesses to a target object
- ***Smart Reference Proxy*** - Provides additional actions whenever a target object is referenced such as counting the number of references to the object

Example: synchronization

InterBit Training & Consulting Ltd.

- Classes to be hidden behind a proxy:

```
// Base interface for data-structure collections:
interface Collection {

    boolean add(Object e);
    int size();
    ...
}

// The following classes are not thread-safe:
class ArrayList implements Collection {
    ...
}

class LinkedList implements Collection {
    ...
}
```

Collection classes.

Those classes are not thread safe.

We shall soon hide them behind a proxy

.)Synchronization (cont

InterBit Training & Consulting Ltd.

- Synchronizing proxy:

```
// Proxy that synchronized the access to a collection
// Note: standard java.util.Collections follow a similar design,
// but it also allows you to configure which mutex to use

class SynchronizedCollection implements Collection {
    private Collection c;

    SynchronizedCollection(Collection c) {
        this.c=c;
    }
    public synchronized boolean add(Object e) {
        return c.add(e);
    }
    public synchronized int size () {
        return c.size();
    }
}
```

Proxy:

Synchronizes the access to a collection

RMI Stubs

InterBit Training & Consulting Ltd.

- Server machine holds and registers an object, e.g. Bank.
- The client machine has a class *Bank Stub*
  - Stub has the same interface as Bank.
  - Stub forwards requests to the remote bank, using object streams.
  - Client's life is now much easier, since it feels as if it talks to a bank; sockets become transparent to him.

Deferred initialization

InterBit Training & Consulting Ltd.

- **Lazy initialization / activation:**
  - With EJB's , client communicates with a proxy rather than with the bean itself.
  - Thus, transparent to the client, beans that are idle may be saved to database and cleared from memory. They'll be re-activated when required.
  - Recall the Double-checked-locking problem (synchronization & lazy initialization, discussed under "Singleton").

Java 1.3 Dynamic proxies

InterBit Training & Consulting Ltd.

- **Creating proxy classes at runtime**
  - When creating a proxy, indicate:
    - Which interfaces it should implement.
    - Which object it wraps (object must implement the afore-mentioned interfaces).
  - There are some limitations on such proxies.
    - E.g. they must implement interfaces rather than inherit from classes.
  - The mechanism relies on reflection.

Dynamic proxy – the proxy

InterBit Training & Consulting Ltd.

```
// A Proxy that intercepts String arguments & converts then to uppercase
// Then, as usual, it will forward method calls to the enclosed object

import java.util.*;
import java.lang.reflect.*;

class UppercaseProxy implements InvocationHandler {

    private Object obj;

    public UppercaseProxy(Object obj) {
        this.obj=obj;
    }

    public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
        if (args!=null){
            for (int i = 0; i < args.length; i++) {
          if (args[i] instanceof String) {
              String s = (String)args[i];
              args[i] = s.toUpperCase();
          }
      }
        }
        return  m.invoke(obj, args);
    }

}
```

Dynamic proxy – test:

InterBit Training & Consulting Ltd.

```
// You can now wrap this proxy around any object (e.g: List),
// provided you only work through interfaces :

public class ProxyTest {

    public static void main(String[] args) throws Exception {
     ArrayList myList=new ArrayList();
     // Create a proxy that wraps myList and implements interface List:
     Object proxy = Proxy.newProxyInstance(
          java.util.List.class.getClassLoader(),
          new Class[] {java.util.List.class},     // interfaces
          new UppercaseProxy(myList));       // wrapped obj

     // Add items to list, through the proxy:
     List pList= (List) proxy;
     pList.add("Aa");
     pList.add("bbb");
     System.out.println(pList);
    }
}
```

Comparison to Related Patterns

- Both the **Adapter** and the **Proxy** constitute a thin layer around an object.

- **The Difference**:
  - **Adapter** provides a different interface for an object.
  - **Proxy** provides the same interface for the object but interposes itself where it can save processing effort.

Comparison (cont.)

InterBit Training & Consulting Ltd.

- **Decorator** also has the same interface as the object it surrounds

- **The Difference**:
  - The **Decorator** purpose is addition of (usually visual) functionality to the original object
  - A **Proxy**, by contrast, controls access to the contained class