Dynamic Proxies

Java Reflection

Explained Simply

License

What is a proxy?

- Dictionary definition: “a person authorized to act for another”
  - Example: if you ask a friend to vote on your behalf then you are “voting by proxy”
- In computer terms, a proxy is a delegation object (or process)
- Used in remote procedure call (RPC) mechanisms:
  - Client invokes on a (local) proxy object
  - Proxy object sends request across the network to a server and waits for a reply
- Some companies set up a HTTP proxy server:
  - Firewall prevents outgoing connections to port 80
  - So web browsers cannot connect to remote web sites directly
  - Web browsers are configured to connect via the company’s proxy server
  - Proxy server can be configured to disallow access to eBay, Amazon, …

Dynamic proxies in Java

- Java 1.3 introduced dynamic proxies
  - The API is defined in the `java.lang.reflect` package

```
class Proxy {
    public static Object newProxyInstance(
                    ClassLoader loader,
                    Class[] interfaces,
                    InvocationHandler h) throws ...
    ...
}

interface InvocationHandler {
    Object invoke(Object proxy,
                  Method m,
                  Object[] args) throws Throwable;
}
```

Steps required to create a dynamic proxy

- Step 1:
  - Write a class that implements `InvocationHandler`
  - Your implementation of `invoke()` should:
    - Use `Method.invoke()` to delegate to the target object
    - Provide some “added value” logic
- Step 2:
  - Call `Proxy.newInstance()`, with the following parameters:
    - `targetObj.getClass().getClassLoader()`
    - `targetObj.getClass.getInterfaces()`
    - `InvocationHandler` object “wrapper” around the target object
- Step 3:
  - Typecast the result of `Proxy.newInstance()` to an interface implemented by the target object

How does this work?

- The `Proxy.newProxyInstance()` method:
  - Uses runtime code generation techniques
  - Generates a “hidden” class with a name of the form `$Proxy<int>` (Use of `“$”` prevents namespace pollution)
  - Generated class:
    - Implements the specified interfaces
    - Each method puts parameters into `Object[]` and calls `InvocationHandler.invoke()`
- Can use a dynamic proxy *only if* a class implements 1+ interfaces
  - Use of interfaces is a good programming practice
  - So this requirement is not a problem in practice

Sample code

```
public class Handler implements InvocationHandler {
    private Object target;

    private Handler(Object target) {
        this.target = target;
    }

    public Object invoke(Object proxy, Method m,
                         Object[] args) throws Throwable
    {
        Object result = null;
        try {
            ... // added-value code
            result = m.invoke(target, args);
        } catch(InvocationTargetException ex) {
            ... // added-value code
            throw ex.getCause();
        }
        return result;
    }
    ... // continued on the next slide
```

The `proxy` parameter is usually ignored

Sample code (cont')

```
    ... // continued from the previous slide

    public static Object createProxy(Object target)
    {
        return Proxy.newProxyInstance(
                  target.getClass().getClassLoader(),
                  target.getClass().getInterfaces(),
                  new Handler(target));
    }
}
```

Example uses for dynamic proxies

- Added-value code might:
  - Enforce security checks
  - Begin and commit or rollback a transaction
  - Use reflection & recursion to print details of all parameters (for debugging)
- In a testing system, a proxy might “pretend” to be target object
  - Returns “test” values instead of delegating to a real object
  - EasyMock (www.easymock.org) makes it easy to write tests in this way