be none the wiser as long as we return compatible responses. We can approximate this flexibility in modern object-oriented languages through the use of interfaces, but that still constrains us to a "physical" coupling to the interface definition. With the logical-only binding, we still need to support expectations from existing clients, but beyond that we are not coupled to any particular implementation detail. This is the same value we see communicating through URIs on the Web, but in locally running software!

Internally, we use the Command Pattern[‡] associated with the method type of the request to implement the accessor. An HTTP GET method is mapped to a `GetResourceCommand` that maintains no state. When the request comes in, we pull the command out of a map and issue the request to it. The REST stateless style ensures that all information needed to answer the request is contained in the request, so we do not need to maintain state in the command instance. We can access that request state through the context instance in the following code. This code looks relatively straightforward to Java developers. We are calling methods on Java objects, catching exceptions, the works. An important thing to note is the use of the IURAspect interface. We are essentially saying that we do not care what form the resource is in. It could be a DOM instance, a JDOM instance, a string, or a byte array; for our purposes it does not matter. The infrastructure will convert it into a bytestream tagged with metadata before responding to the request. If we had wanted it in a particular form supported by the infrastructure, we could have simply asked for it in that form. This declarative, resource-oriented approach helps radically reduce the amount of code that is necessary to manipulate data and allows us to use the right tool for the right job:

```
if(resStorage.resourceExists(context, uriResolver)) {
    IURAspect asp = resStorage.getResource(context, uriResolver);

    // Filter the response if we have a filter
    if (filter!=null) {
        asp = filter.filter(context, asp);
    }

    // Default response code of 200 is fine
    IURRepresentation rep = NKHelper.setResponseCode(context, asp, 200);
    rep = NKHelper.attachGoldenThread(context, "gt:" + path , rep);
    retValue = context.createResponseFrom(rep);
    retValue.setCacheable();
    retValue.setMimeType(NKHelper.MIME_XML);
} else {
    IURRepresentation rep = NKHelper.setResponseCode(context,
            new StringAspect("No such resource: "
                  + uriResolver.getDisplayName(path)), 404);
    retValue = context.createResponseFrom(rep);
    retValue.setMimeType(NKHelper.MIME_TEXT);
}
```

‡ *http://en.wikipedia.org/wiki/Command_pattern*