26

COUNTING PROXY

This pattern was previously described in GoF95.

DESCRIPTION

The Counting Proxy pattern is useful in designing a set of additional operations such as logging and counting that need to be performed before and/or after a client object invokes a method on a service provider object. Instead of keeping these additional operations' implementation inside the service provider object, the Counting Proxy pattern suggests encapsulating the additional functionality in a separate object referred to as a *counting proxy*. One of the characteristics of a well-designed object is that it offers focused functionality. In other words, an object, ideally, should not do various unrelated things. Encapsulating the logging, counting and other similar functionality into a separate object leaves the service provider object with only the functionality that it is designed to offer. In other words, it allows the service provider object to perform a well-defined, definite task.

A counting proxy is designed to have the same interface as the service provider object that a client accesses. Instead of accessing the service provider object directly, client objects invoke methods on the counting proxy. The proxy performs the required logging and counting and forwards the method call to the service provider object (Figure 26.1).

The following example illustrates how a counting proxy can be used in an application scenario.

EXAMPLE

Let us design an `Order` class hierarchy as in Figure 26.2. The `OrderIF` interface declares a single method `getAllOrders` to read all orders from a database.

```
public interface OrderIF {
  public Vector getAllOrders();
}
```

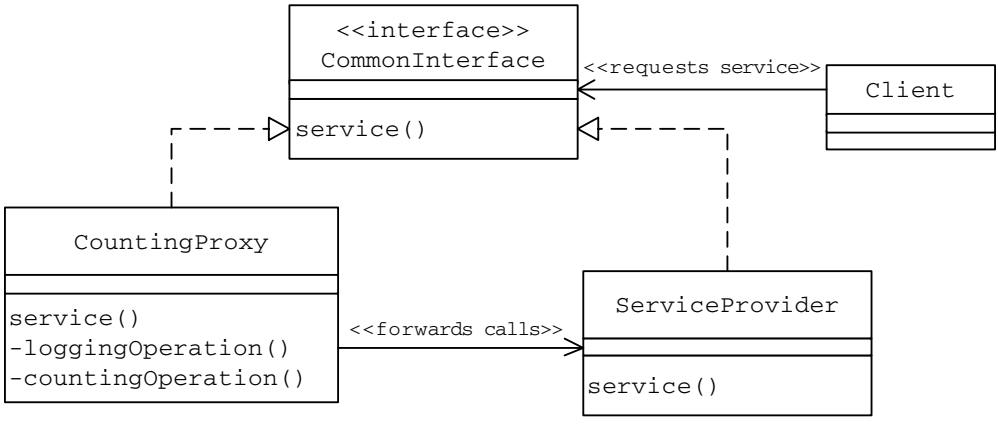

Figure 26.1 Generic Class Association When the Counting Proxy Pattern Is Applied

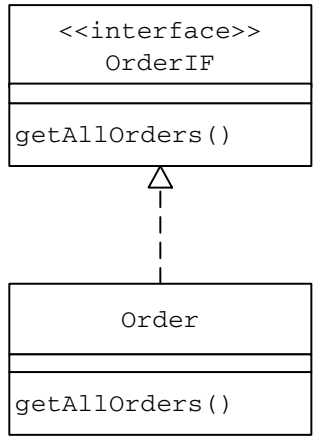

Figure 26.2 Order Class Hierarchy

As part of its implementation of the `getAllOrders` method, the `Order` class makes use of the `FileUtil` utility class to read order items from a data file `orders.txt`.

```
public class Order implements OrderIF {
  public Vector getAllOrders() {
    FileUtil fileUtil = new FileUtil();
    Vector v = fileUtil.fileToVector("orders.txt");
    return v;
  }
}
```

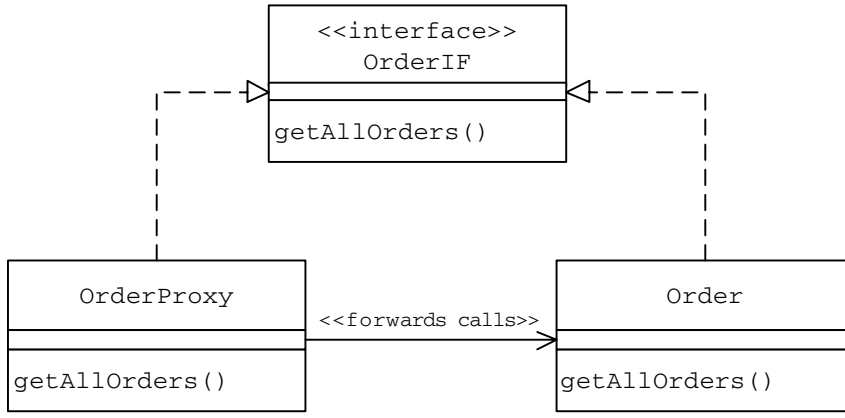

Figure 26.3 Order Class Hierarchy with the Counting Proxy

Let us suppose that the time it takes to read the data file and the number of times the `getAllOrders` operation is invoked need to be logged to a log file.

This additional functionality can be designed as a separate class `OrderProxy` that implements the same `OrderIF` interface as the actual `Order` object. This ensures that the `OrderProxy` offers the same interface to client objects as the actual `Order` object (Figure 26.3).

```
public class OrderProxy implements OrderIF {
  private int counter = 0;
  public Vector getAllOrders() {
    Order order = new Order();
    counter++;
    long t1 = System.currentTimeMillis ();
    Vector v = order.getAllOrders();
    long t2 = System.currentTimeMillis();
    long timeDiff = t2 - t1;
    String msg =
      "Iteration=" + counter + "::Time=" + timeDiff +
      "ms";
    //log the message
    FileUtil fileUtil = new FileUtil();
    fileUtil.writeToFile("log.txt",msg, true, true);
    return v;
  }
}
```

The client object `MainApp` can make use of the `OrderProxy` object as if it is the real `Order` object and invoke the `OrderIF` method `getAllOrders` on

it. The `OrderProxy` forwards the call to the actual `Order` object, calculates the time it takes to read all orders and logs these details to a log file using the `FileUtil` helper class. In this process, the `OrderProxy` plays the role of a counting proxy.

```
public class MainApp {
  public static void main(String[] args) {
    OrderIF order = new OrderProxy();
    Vector v = order.getAllOrders();
    v = order.getAllOrders();
    v = order.getAllOrders();
    v = order.getAllOrders();
  }
}
```

PRACTICE QUESTIONS

1. Design a counting proxy that keeps track of the number of orders created and provides the average order amount.
2. Consider items in a library. Library items can be divided into four categories — magazines, books, videos and DVDs. Design a proxy to keep track of the number of items of each category that are checked out every day.