

Module 11 sample

Managing components with Modeler

11.1 Understanding JMX.....	1
11.2 Say hello to Modeler.....	20
11.3 Summary.....	31
Index.....	32

Imagine that you've just built the next killer Java application, and it's selling like hotcakes. However, even though (being a good developer) you built in enough debug and tracing information to monitor the state of your live application units, you're missing the ability to monitor each and every class and their attributes. Although you could build monitoring services as part of your application, it takes the focus away from your core business, the application itself. It would be nice if you could monitor your application for the state of its classes, attributes, and operations. Java Management Extensions (JMX) let you do just that. However, this chapter isn't about JMX. It's about how to easily configure JMX using the Modeler component. Specifically, it's about how to use the Modeler component to create Model MBeans (a special kind of management bean), which are used to monitor resources in your application.

It would be difficult to understand Modeler without first JMX. Therefore, this chapter starts with a simple tutorial of the JMX API and its components. We'll pay special attention to Model MBeans and how they interact with their environment. We'll then introduce Modeler using several examples. We'll round out the chapter with examples of how to integrate Modeler with the Ant build tool.

11.1 Understanding JMX

JMX is a toolset that provides guidelines to manage and monitor applications in a distributed environment. But what does that mean? It means that using JMX, you can view the state of your application, make changes to variables, invoke operations on your objects, and gather any other information that is exposed by your application, without making wholesale changes to your application's main functions or architecture.

Let's try to understand this in the context of an application with diagrams and code. In this process, we'll introduce the various layers of a JMX-managed application. The first layer is called the Instrumentation layer.

11.1.1 Introducing the Instrumentation layer

To begin, consider figure 11.1. It shows an application as a standalone component that's deployed and functional in a production environment.

How do you introduce JMX in this picture? You start by introducing another component called a managed bean (MBean). An MBean is a Java object that acts as a wrapper for your application components and exposes them for management. Figure 11.2 shows an MBean introduced into figure 11.1.

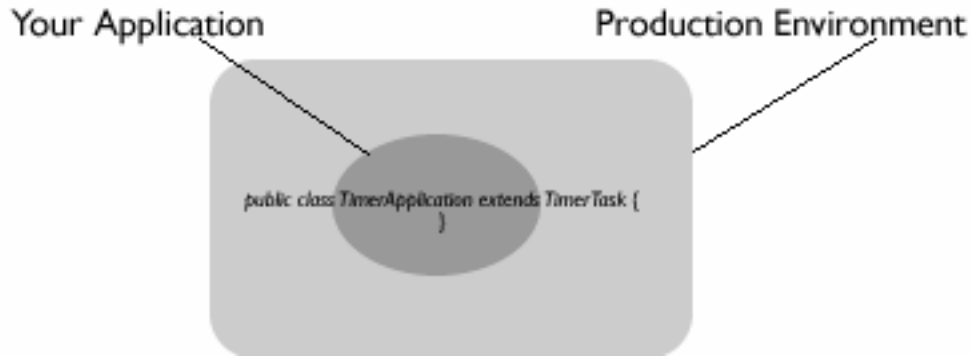


Figure 11.1 An application in a production environment. How do you manage this application?

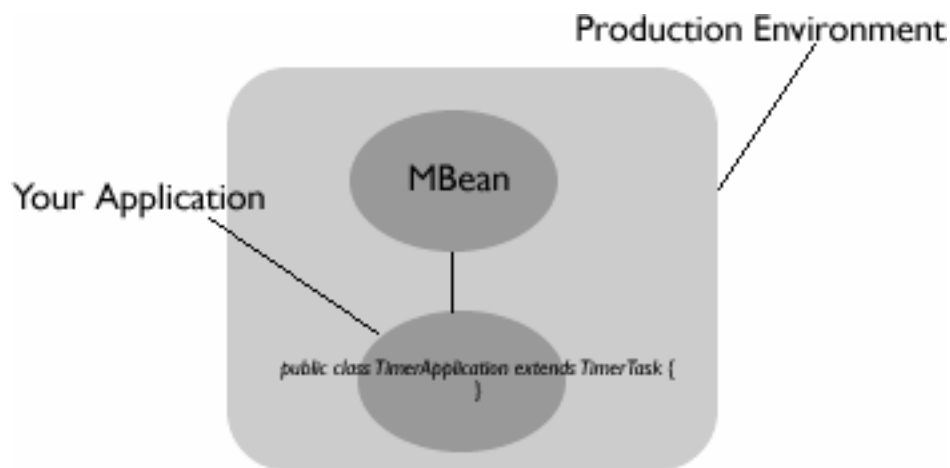


Figure 11.2 Introducing an MBean to expose your application for management

The MBeans and your application's components together form what is called the *Instrumentation layer* for JMX-managed applications. The Instrumentation layer interfaces with the outside world to expose these components for management. But how do the MBeans in this layer expose these components? Well, MBeans are interface descriptions of your components. This means that the MBeans closely resemble the components they expose and that the components must implement these interfaces themselves. These interface definitions are read by the outside components using Java Reflection.

Let's look at some code to clarify things before proceeding further. Listing 11.1 shows a simple application that prints the current date and time after a specified delay. Consider this listing an application component that will be managed using JMX.

Listing 11.1 Simple application that prints the current date and time after a delay

```
package com.manning.common.chapter11;

import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;

public class TimerApplication extends TimerTask {

    private long delay;
```