Introduction to Spring

Why Spring? Replaceable Objects

Written By Rick Hightower CTO of ArcMind (http://www.arc-mind.com)

Contributors: Thomas Bridges (Qualcomm Inc.), Alex Redington, Kris Diefenderfer, Mary Basset, Martha Pena, Andrew Barton (eBlox) and Drew Davidson (Ognl).

If you follow the latest developer trends then you've likely heard of Spring, IoC (Inversion of Control) containers and AOP (aspect-oriented programming). Like many developers, architects and development managers, however, you may not see where these technologies fit into your development efforts.

In this paper, we'll begin to remedy that, with a hands-on introduction to Spring to understand the basics of IoC, AOP and Spring Templates. We start with a high-level overview and then later delve into some of the technical details with plenty of code samples. This paper is useful for developers (code examples start on page 10), architects and development managers.

Why should you care?

- AOP can ***drastically reduce code*** that would otherwise be repeated.
- IoC allows you to easily assemble objects with their collaborators **getting rid of so called wiring code that typically occupies 30% or more of many code bases**. IoC also facilitates test driven development and unit testing.
- Spring Templates take common, error prone code and encapsulates the logic for resource management and tricky exception handling so it can be reused, thus ***reducing the possibilities of defects that leak resources and reducing lines of code.***

The great thing about objects is they can be replaced. The great thing about Spring is it helps you replace them. In the book Object-Oriented Analysis and Design with Applications, Grady Booch, one of the founding fathers of object-oriented programming, stated that the great thing about objects is that they can be replaced.

Spring is not prescriptive

One of the biggest benefits of Spring is that it isn't prescriptive. Trying to associate something is a lot easier if you do it piecemeal (ie: "okay guys, let's use this Spring piece" "okay, Now Spring can solve that too", etc.) than trying to do it all as one great big horse pill.

Thus, you can slowly work Spring into your projects.

The ability to replace objects makes the code base more extensible. Thus, the great thing about object-oriented programming is that it supports abstraction. Objects expose functionality through methods and use other objects (conventionally called collaborators) via their interfaces.

In domain driven design, you partition your objects into domain objects that model your applications business rules and data. All domain objects have a role. However domain objects need collaborators objects to perform their role.

Domain objects are connected to objects that help them fulfill their role. The constructs that we create using these abstractions are called object graphs, networks of objects that are navigable from other objects. The object graph defines how one object maintains connections to other objects that collaborate to perform some other, more complex, functionality. In this way complex functionality can be built from simple, easy-to-test parts.

The core of the Spring Framework helps you configure these object graphs. The key concept Spring provides is that objects do not have to know how to locate these collaborators, just use them as provided. The rest of the Spring framework acts with the support of this mechanism to build more sophisticated functionality and provide support for commonly used Java APIs.

Domain objects are objects that deal with your business domain (vertical objects). Domain objects contain logic particular to your business domain. An example of a domain object could be an ***AutomatedTellerMachine*** object or a Library object. The domain object for

an ***AutomatedTellerMachine*** would have the entire business rules particular to ***AutomatedTellerMachine***s. Aspects on the other hand implement crosscutting concerns (horizontal objects). A crosscutting concern is a concern that crosses domain boundaries. For example, both an ***AutomatedTellerMachine*** object and a Library object may use a transaction manager, but an ***AutomatedTellerMachine*** (ATM) has nothing to do with a Library. With Spring, you can extract the code specific to transaction management (or logging, auditing, security, etc.) out of domain objects and use them with the ATM and Library objects as well as other domains. Functionality that is not core to a particular domain is called a crosscutting concern.

Automated Teller Machine is an Automated Banking Machine

In some countries an Automated Teller Machine (ATM) is called an Automated Banking Machine (ABM). If you live in one of those countries, translate accordingly.

IoC and AOP High-level Explanation

With Spring IoC support, you simply inject collaborating objects called dependencies using JavaBeans properties and configuration files. Then it's easy enough to switch out collaborating objects when you need to. You can also define the dependencies such that they are stubs created specifically for unit testing.

The ability to inject collaborating objects is often called Inversion of Control (IoC), also referred to as Dependency Injection (a term coined by Martin Fowler). Thus, Spring is an IoC container. The ability to dynamically add services to objects is called Aspect Oriented Programming (AOP). Spring supports both. Thus, Spring is an IoC/AOP container.

IoC allows you to create an application context where you can construct objects, and then inject those objects into their collaborating objects. As the word inversion implies, IoC is somewhat like JNDI turned inside out. Instead of using a tangle of abstract factories, service locators, singletons, and straight construction, each object is constructed with its collaborating objects provided to it using standard JavaBeans setter methods. The container manages the collaborators and removes the burden of doing lookups via ad-hoc methods.

JNDI turned inside out

Java Naming and Directory Interface (JNDI) provides a unified interface to naming and directory services. In the J2EE model, domain objects look up other objects they need to do their job like datasources. In the Spring model, the Spring container injects objects like datasources that the domain objects need.

Another key focus of the Spring framework is dealing with crosscutting concerns. Spring's AOP support allows you to dynamically add services to objects called aspects. AOP enhances object-oriented programming, it does not replace it. AOP further allows developers to create non-domain concerns, called crosscutting concerns, and insert them in their application code, without the knowledge of the collaborating object. With AOP, common crosscutting concerns like logging, persistence, transactions, and the like can be factored into aspects and applied to domain objects without complicating the object model of the domain objects.

Since all objects are configured using named references to one another we can replace or enhance the object graph at will without affecting the users of these dependencies. Using AOP to create crosscutting concerns where it can be applied to many domain objects is an alternative to adding the crosscutting concern in the domain object where it must be repeated for every method and every other domain object. AOP can drastically reduce the amount of code you need to write for crosscutting concerns. AOP is a cleaner separation of concerns than traditional domain model only programming. Traditional programming of common-concerns in the domain model is fragile and can lead to a lot of redundant code.

AOP can drastically reduce and simplify your code base

Using AOP allows you to move mundane code out of your domain objects into aspects. Then you can apply this aspect dynamically to other objects with no additional coding effort! An aspect is an implementation of a crosscutting concern.

Thus when you get an object from the Spring application context, it is likely to have collaborators (which were injected by the IoC container) and decorators (AOP proxies) associated with it.

AOP Dynamic Event Listener and Trigger Like mechanism

It is often useful to think of AOP as a **dynamic event listener** or if you are more of a database person, you can think of it as a **trigger like mechanism for Java objects**. You can listen to any method invocation.

Vendors like SAP now publish ways to use AOP to listen to method invocation. Do you need to know when a new employee gets added to the system? Write an Aspect! You can extend code in ways never envisioned by the original creators of the code base. **AOP makes OOP (object-oriented programming) more extensible.**

AOP which augments not replaces OOP is still very new. We have not explored all of the possibilities of this technology.

More than IoC and AOP

There are many IoC containers. There are also many AOP frameworks. If Spring was only an IoC/AOP container, it would be worth your attention and interest. What makes Spring different than the other frameworks and containers is that Spring goes beyond just being an IoC container or an AOP framework. The other containers provide good support of IoC and AOP. Spring goes one step further by "eating its own dog food" to build a framework to simplify J2EE development.

AOP and popularity

What is the use of having a great technology if no one uses it? Spring used AOP to create services like declarative transaction management to plain old Java objects. It is the practical application of AOP that made Spring popular and made AOP mainstream. Spring played a large role in making AOP mainstream. Spring was the first AOP framework that actually built useable services with its AOP implementation.

Slang: Eating its own dog food

"A company that eats its own dog food sends the message that it considers its own products the best on the market."

"This slang was popularized during the dotcom craze when some companies did not use their own products and thus could "not even eat their own dog food". An example would've been a software company that created operating systems but used its competitor's software on its corporate computers."

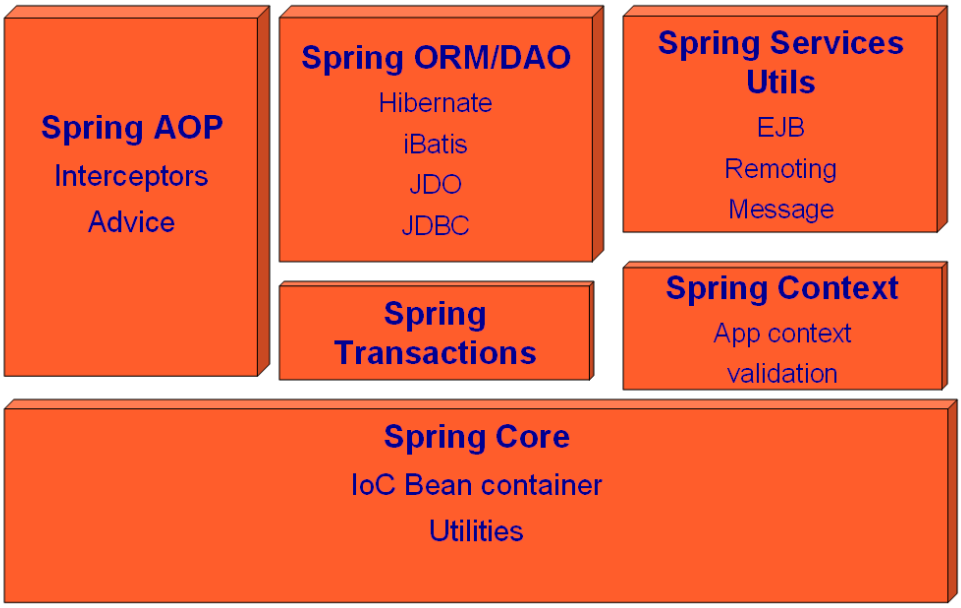
http://www.investopedia.com/terms/e/eatyourowndogfood.asp

The Spring Framework has shown it could "eat its own dog food" by using its implementation of AOP and IoC to build a framework to simplify enterprise development.

Simplify J2EE development

Spring uses IoC and AOP to provide a comprehensive library for simplifying J2EE development. Spring makes using other Java APIs and frameworks very easy to incorporate into your application. This comprehensive library is written with aspects and dependency injection. See figure 1.1.

FIGURE 1-1. BLOCK DIAGRAM OF SPRING FRAMEWORK



Templates

Spring makes J2EE development easier. It does this with a variety of mechanisms. One common mechanism is its use of templates. A template is a cross between a utility class and execution environment. At first glance templates appear to be well written utility classes.

However, Spring Templates provide a lot more than just utility functions. Templates provide an execution environment.

When using a template you first endeavor to use one of its utility methods. If the template doesn't have a utility method you need, you implement a callback object.

The callback object has a method that executes a method in the environment of the template. The template therefore takes care of things like exception handling and resource management in a consistent manner. This means your code base will not be littered with try/catch/finally blocks. Centralizing exception handling helps prevents programming mistakes; thus, improving the quality of your code base and reducing lines of code. A quality code base is a more productive code base.

Advantages of Callback objects

Callback objects reduce your code base and make it less error prone by allowing the Template to do the cleanup.

Spring promotes good programming practices. It does this by providing great examples how to use IoC, AOP and OOP in a consistent manner. It also does this through its ability to build things like templates to manage resources and exceptions in a consistent manner.

Another form of IoC

The callback object that the Spring Templates use is another form of Inversion of Control. IoC is a foundation to OO programming. By using the callback object, you are allowing the Template class to control the flow of the application and allow it to handle exceptions and properly manage resources. This allows you to focus on what you want to do with a particular API (the interesting things) by letting the Template class do the routine things with the API like resource cleanup and location.

Spring and Test Driven Development

IoC capabilities of dependent object injection turns out to be a great mechanism for testing your code. It is easy now to inject mock objects (object for testing) and test your classes in an isolated manner. For example, you can test your business objects without relying on the DAO (Data access objects) talking to the database.

Essentially Spring took back development from the J2EE design pattern hacks that were deemed necessary to program J2EE. Spring puts the OO back in J2EE development. Instead of creating code that is tied to a container, Spring frees your code. This fits the motto "the great thing about objects is that they can be replaced".

Spring and TDD, Much More Productive Than Traditional J2EE

Spring goes hand in hand with test driven development. True developer productivity is driven by the quality of the code base.

The tale of two programmers:

In corner A we have the traditional J2EE developer using EJB. He is dependent on his container to test his code. He must code, deploy, wait (for deployment), test - repeat. He must also reboot his application server every now and then and wait some more.

In corner two we have our lightweight developer using Spring, IOC, Hibernate etc. This developer can test, even against the db without a container. His test cycle is code, test, code, test, etc. He gets more done and its higher quality work.

Higher Level Abstractions for a whole host of projects

Spring provides portability through abstraction of common services. For example, Spring provides a common interface for object relational management (ORM) systems like Hibernate, JDO, Cayenne, Spring JDBC and iBatis. It provides a mechanism for building DAO objects that divorces the client code from the underlying implementation. Spring does this by providing a common set of exceptions like the object not found exception, and making these exceptions runtime exceptions.

Spring provides an easy on-ramp for many industry-standard projects, and also for the de facto industry-standard projects, the projects that people actually use to get their daily work done. For example, Spring provides support for Hibernate, Quartz, Hessian and many more.

Spring simplifies J2EE development. Also, Spring provides utilities to work with all tiers of an n-tier application. For an MVC application, there are utilities for working with View technologies (Struts, Spring MVC, JSF, Tapestry, Swing Rich Client etc.), Model (EJB, AOP based transaction, AOP based security, etc.), etc. You can use Spring to build Swing and SWT applications just as easily.

The rest of this paper is going to give you a bird's eye view of the Spring valley. We won't cover all of the details, but we will show you some of the sample code. In later papers, we will examine the bark of the trees in the

Spring valley.

> **Show me the CODE!**
>
> I dislike the practice of waiting several hundred pages before showing code examples. If the examples don't make sense yet, realize that this series of papers will explain them in more detail in later papers. If this paper opens more question than it answers, so be it. Sometimes it is best to know what questions to ask. The rest of the papers in this series will answer any questions this paper opens for you. For now let's look at the forest in the valley. Later we will examine the bark on the trees.

> **Lab Centric Courseware!**
>
> If you like this paper, you will really like our custom, lab-centric, JSF, Hibernate and Spring Framework training! Our courses are taught by developers with real-world experience. Check our website for onsite training: ArcMind (http://www.arc-mind.com).

This **may** be a good time if you are a manager or architect to skim the rest of the paper and skip to the conclusion of this paper, and leave the low level details to the developers.

Be sure to read on the info boxes and examine the diagrams along the way.

IoC Quick Intro for Developers

IoC allows you to easily assemble objects with their collaborators getting rid of myriads of service locators, abstract factories, and singletons, which are forms of wiring code. This wiring code typically occupies 30% or more of many code bases.

Spring's core functionality serves to provide a central repository for finding objects by name called an ApplicationContext. Your applications will commonly create the ApplicationContext in a centrally-located place where it is accessible to any object that requires its services (for example via a static method in a Rich Client application, or the ServletContext in a web application). Since Spring encourages configuration of objects without their knowing Spring APIs, you will usually only use the Spring API when you want to use the objects in a ApplicationContext, not in the configured objects themselves.

> **"You may never need to use the BeanFactory API"**
>
> The ApplicationContext is a BeanFactory. You could extensibly use Spring and never access the BeanFactory API or ApplicationContext directly. Since Spring integrates with JSF, Tapestry, Struts, WebWork and more, you can write application, which get their dependencies injected and never need to access the BeanFactory or ApplicationContext directly.

Spring can provide services like declarative transactions, declarative security, and more just like EJB. Unlike EJB, beans defined in the Spring container are independent of

Spring; you don't have to implement any special interface or subclass any special base class like you do with EJB. Beans can be configured and managed by Spring without depending on Spring; thus, the beans are decoupled from the container. Spring can injects dependencies, also called collaborative objects. Collaborators are objects that beans needs to complete its role. Spring uses regular JavaBean and Java language constructs to inject dependencies. Beans managed by Spring are plain old Java objects (POJOs).

Beans that are decoupled from the underlying container

Decoupling beans from the underlying container allows those beans to be unit tested outside of the container. Also since Spring is such a light weight container it is easy to run it inside of an IDE without deploying anything to a container, which also facilitates unit testing and test driven development.

One of the biggest slow downs in J2EE development is the deploy, wait, and test cycle with an occasional obligatory reboot the application server (wait a few more minutes).

Developing with Spring can save a lot of time in the development and testing process.

Let's show a short example how a bean is configured in the Spring application context. This example demonstrates an ***AutomatedTellerMachine*** (a banking machine). The ***AutomatedTellerMachine*** needs a ***transport*** object to send credit and debit information back to the bank. Thus, it needs the ***transport*** object to fulfill its role. Here is the ***AutomatedTellerMachine*** which we will declare in Spring's application context:

```
package examples;
    /** @author Richard Hightower from ArcMind, Inc. */
public class AutomatedTellerMachine {
    private ATMTransport transport;

    public void setTransport(ATMTransport transport) {
        this.transport = transport;
    }

    public void withdraw(float amount) {
        transport.connect();
```

```
        transport.send(...);
        ...
        transport.disconnect();
        System.out.println("CREDIT");
    }

    public void deposit(float amount) {
        transport.connect();
        transport.send(...);
        ...
        transport.disconnect();
        System.out.println("DEBIT");
    }

   ...
```

Here is the ***AutomatedTellerMachine*** declared in Spring's application context:

```
<beans>
    <bean id="transport" class="examples.SoapSslATMTransport"/>

    <bean id="atm" class="examples.AutomatedTellerMachine" >
        <property name="transport" ref="transport"/>
    </bean>

</beans>
```

Notice that the ***atm*** (bean id="atm"), gets passed the ***transport*** object (bean id="transport) using the property tag. The ***AutomatedTellerMachine*** Java class has a JavaBean property called ***transport***. Having a setter method called ***setTransport*** is similar to the ***AutomatedTellerMachine*** declaring that it needs the ***transport*** object to fulfill it role (see class diagram figure 1.2). Let's demonstrate this pictorially in the next few figures.

FIGURE 1-2. UML DIAGRAM OF AUTOMATEDTELLERMACHINE AND COLLABORATORS

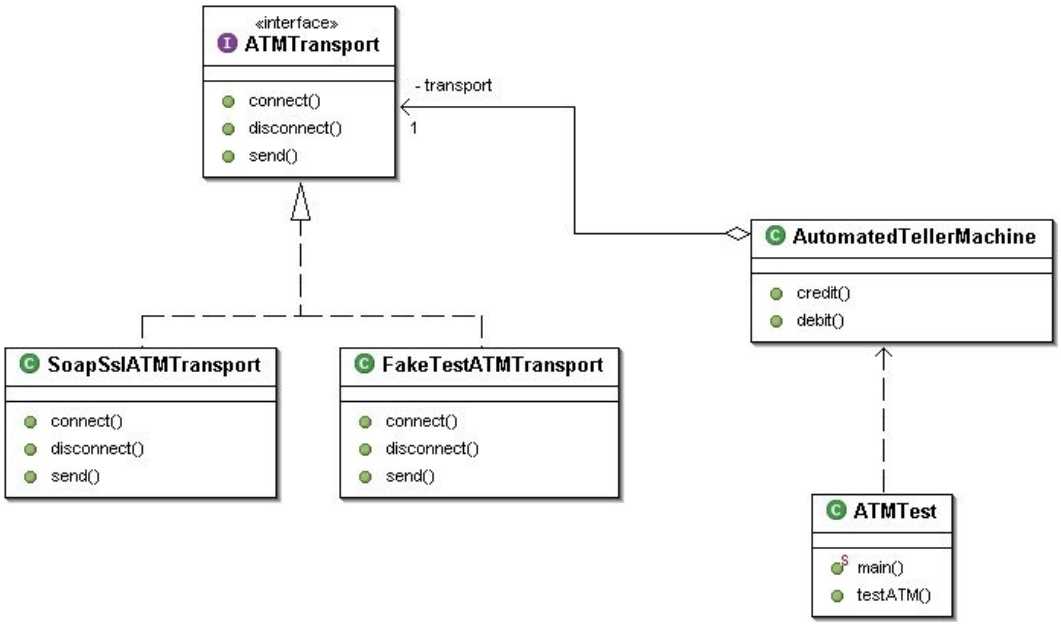


Figure 1-2 is a UML diagram that shows the ***AutomatedTellerMachine*** class contains a ***transport*** reference (the ***transport*** property) that is of type ***ATMTransport*** (an interface). There are two implementations of the ***ATMTransport*** interface (***SoapSslATMTransport*** and ***FakeTestATMTransport***).

FIGURE 1-3. INJECTION BREAKDOWN 1 OF 2

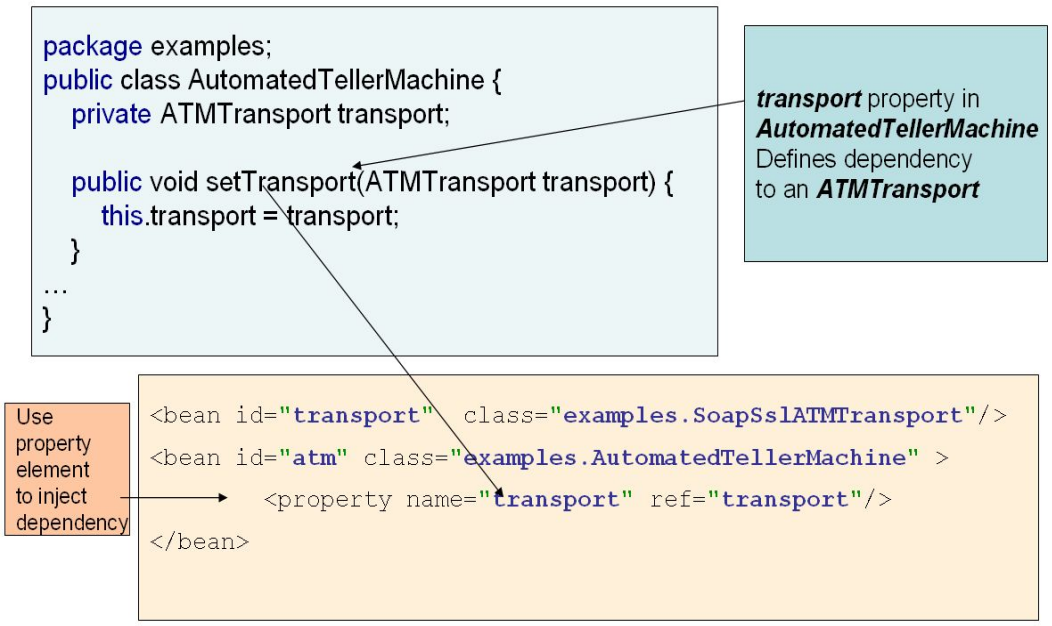


Figure 1-3 shows that the property element of the ***atm*** bean is used to configure the injection of the transport property.

FIGURE 1-4. INJECTION BREAKDOWN 2 OF 2

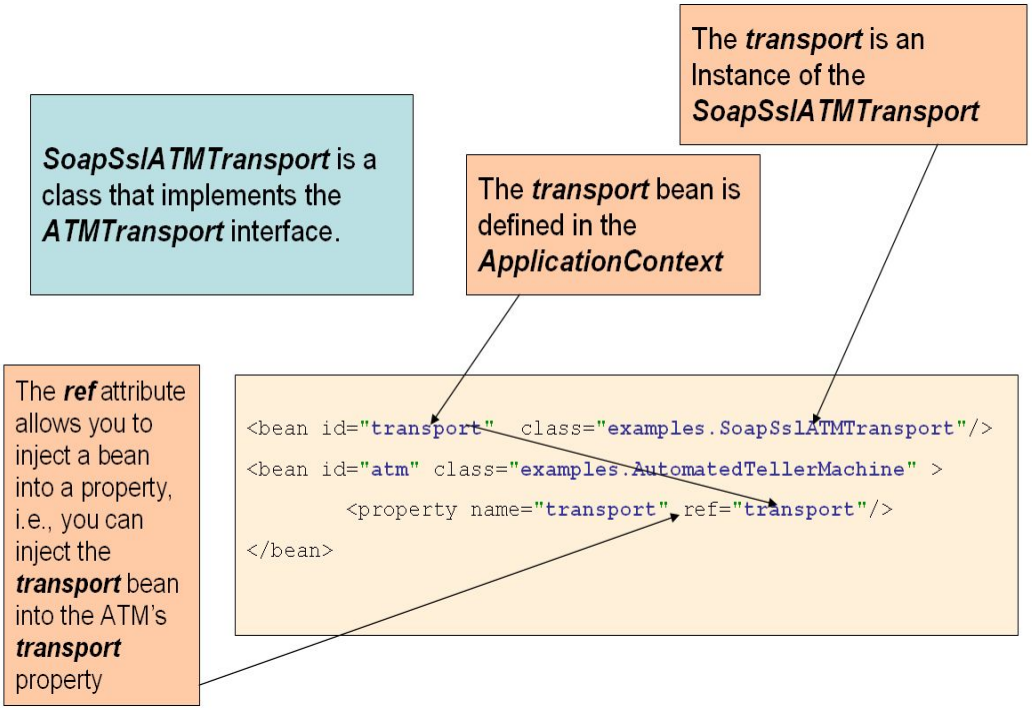


Figure 1-4 shows that the property element of the ***atm*** bean is used to configure the injection of the transport property. An instance of ***SoapSslATMTransport*** will be injected (the setter method will be called passing an instance of the ***SoapSslATMTransport***).

Notice that the ***AutomatedTellerMachine*** is a domain object that uses the ***transport*** object (a collaborator) to complete its role. Likely the ***AutomatedTellerMachine*** implements business rules that you would like to test even when it is not talking to a real transport. Thus, you may swap in a fake transport that you use to test the business rules of the ATM machine demonstrated as follows:

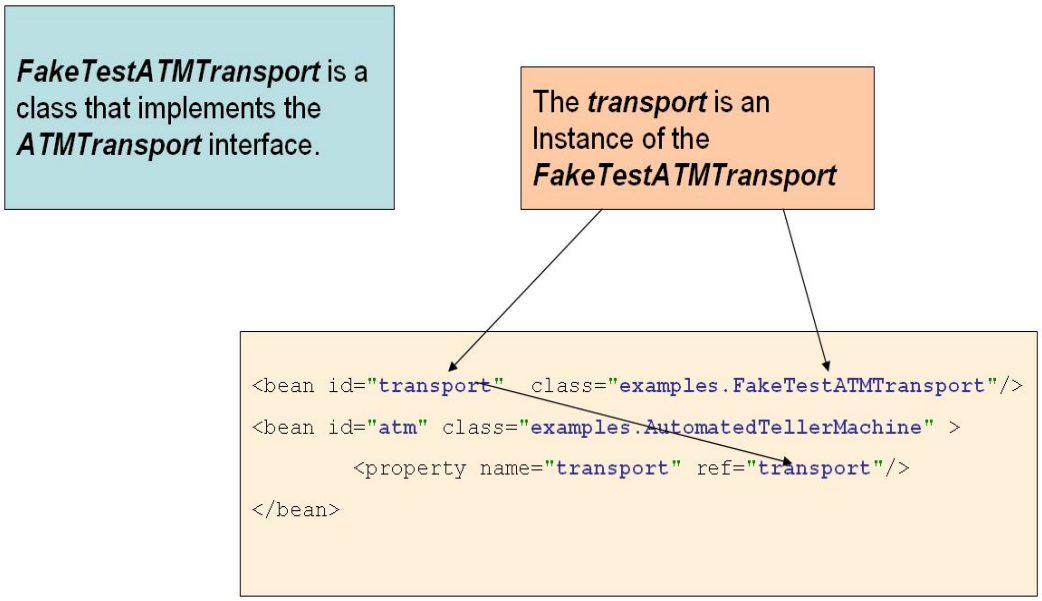
```
<beans>
    <bean id="transport"
              class="examples.FakeTestATMTransport"/>

    <bean id="atm" class="examples.AutomatedTellerMachine" >
        <property name="transport" ref="transport"/>
    </bean>

</beans>
```

Notice here that we swap out the ***SoapSslATMTransport*** with ***FakeTestATMTransport***, now you can write tests that focus on the business rules of the ATM instead of testing the ATM and Transport together (divide and conquer). This is how Spring supports test driven development. It allows collaborative objects to be easily swapped out with test stubs (remember the motto). Figure 1-5 shows this concept pictorially.

FIGURE 1-5. INJECTION OF A TEST OBJECT



Client objects that use the ***atm*** object will look up the ***atm*** object in the application context as follows:

```
import org.springframework.context.ApplicationContext;
...
    ApplicationContext context = ...
    AutomatedTellerMachine atm =
            (AutomatedTellerMachine)context.getBean("atm");
    ...
    atm.deposit(100.0f);
    ...
```

This is just a brief introduction to the IoC container capabilities of Spring. Step by step instructions, and a much fuller coverage of Spring's IoC support will be in the second and third papers in this series.

AOP Quick Intro for Developers

Domain objects are objects that deal with your business domain. Domain objects contain logic particular to your business domain, such as our example dealing with an ATM machine. The domain object would have all of the business rules particular to ATMs. Aspects on the other hand implement crosscutting concerns. A crosscutting concern is a concern that crosses domain boundaries. For example, both an ATM and a Library objects may use a transaction manager, but an ATM machine has nothing to do with a Library. If you are familiar with Design Patterns, an easy way to think about aspects is to think of them as Dynamic Decorator Patterns.

Dynamic decorators are often called AOP proxies in Spring lingo.

Thus, Spring's AOP support allows you to dynamically add services to objects called aspects. This is similar to the Decorator Design Pattern, but does not require you to recompile your code base to apply these services; thus, it is a Dynamic Decorator Pattern. This allows you to replace objects with objects that enhance the originals with additional behavior.

Developer Note: Decorator Design Pattern

The Decorator Design Pattern is a common design pattern that is used quite often to extend functionality of an object without polluting that object with the extension.

"The decorator pattern allows new/additional behavior to be added to an existing method of an object dynamically. This is done by wrapping the new "decorator" object around the original object, which is typically achieved by passing the original object as a parameter to the constructor of the decorator, with the decorator implementing the new functionality. The interface of the original object needs to be maintained by the decorator."

"Decorators are alternatives to subclassing. Subclassing adds behavior at compile time whereas decorators provide a new behavior at runtime."

The above is taken from the Wikipedia
http://en.wikipedia.org/wiki/Decorator_pattern

Spring allows you to create decorator objects on the fly instead of writing code for each decorator. This is a big boon to productivity and reducing the size of your code base (by not repeating yourself).

Aspects are usually services that can be applied to multiple domains. Thus Aspects are application of services like transactions, auditing, security, critical exception handling, etc.

AOP is another way to separate areas of concern in your application. Just like exception and catch blocks allow you to separate your "go code" from your error handling code, AOP allows you to separate your services code (transactions, auditing, security) from your domain logic code. This is a boon for reuse and maintainability of your code base.

Returning to our ATM example, let's say that depending on which country our ATM is in (Canada, U.S.), you have to perform different levels and styles of auditing. Therefore, you want to abstract how auditing is done from the ATM machine. Perhaps, some locations do not need auditing at all. Let's say that the auditing can be used with other domain objects in your business as well. Here is how we would write an Aspect to audit our ATM machine.

Developer Note: If a class ends in FactoryBean

If a class ends in the camel case words FactoryBean it means that Spring will use this object to create another object. A FactoryBean is a bean that knows how to create another bean. Thus, a ***ProxyFactoryBean*** knows how to create an AOP proxy (something we have been calling a dynamic decorator).

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">


<beans>
    <bean id="transport" class="examples.SoapSslATMTransport"/>


    <bean id="atmTarget"
                class="examples.AutomatedTellerMachine" >
        <property name="transport" ref="transport"/>
    </bean>



    <bean id="auditor"
             class="examples.services.AuditingUSAspect"/>


    <bean id="atm"


    class="org.springframework.aop.framework.ProxyFactoryBean">


         <property name="target" ref="atmTarget"/>
         <property name="interceptorNames">
            <list>
               <value>auditor</value>
            </list>
         </property>
    </bean>


</beans>
```

A common complaint in the industry is that all AOP examples deal with logging (and auditing). And that these examples are trivial at best. In other papers we wrote that deal with Spring AOP, we developed a ***SecurityManager*** aspect. Please contact us if you would like our other Spring papers or visit our web site for more details.

http://www.arc-mind.com

Notice that instead of getting the ATM object directly, you get the ***atm*** object through the ***ProxyFactoryBean***. Thus when the client code gets the ***atm*** object, it gets the ***atm*** domain object decorated with the auditor aspects behavior. Figure 1-6 through 1-8 explain this pictorially.

FIGURE 1-6. BEFORE AOP ADDS AUDITOR SUPPORT

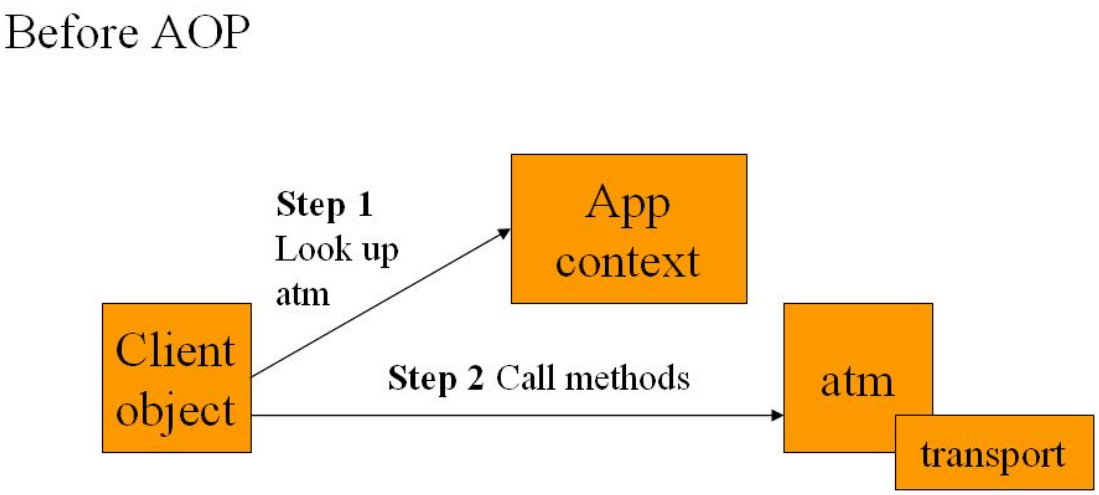


FIGURE 1-7. AFTER AOP ADDS AUDITOR SUPPORT

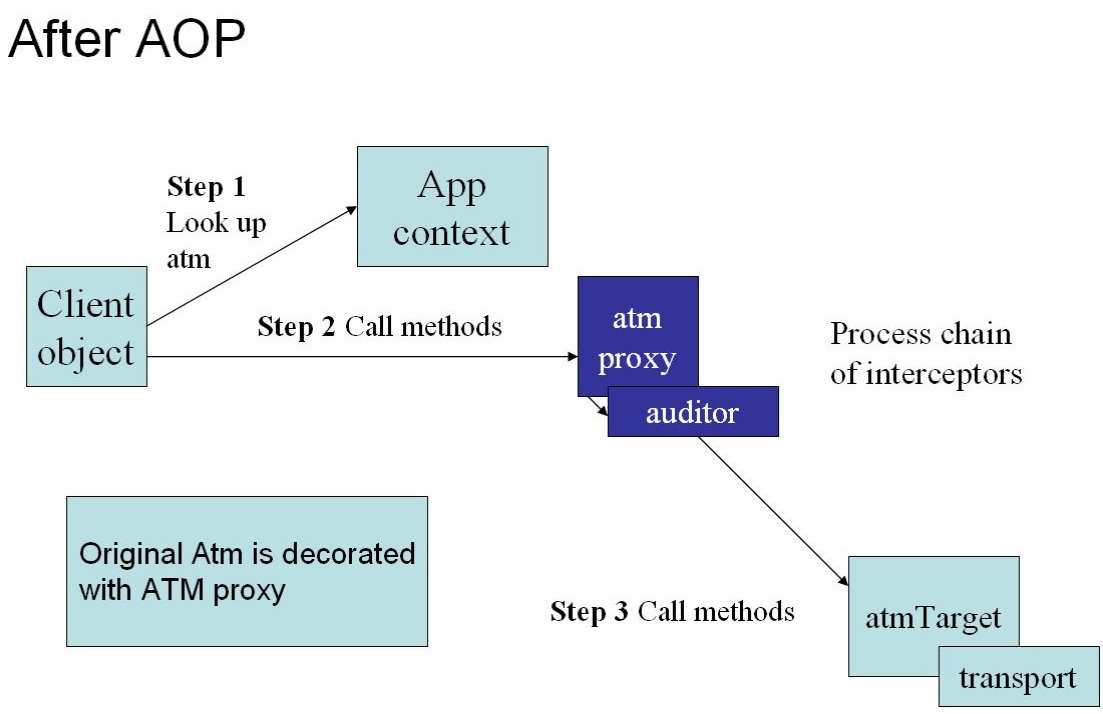
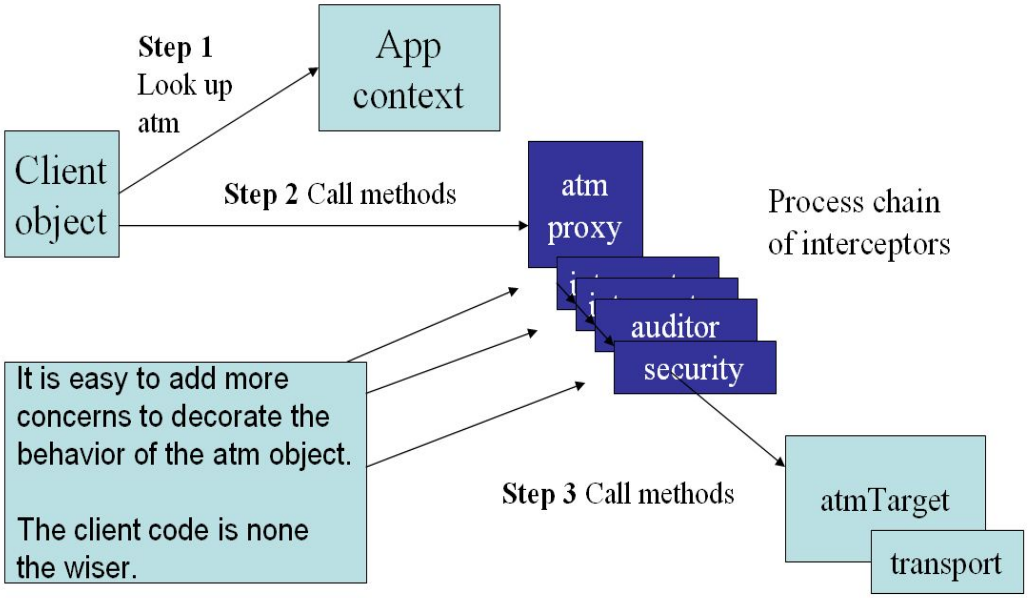


FIGURE 1-8. EASY TO ADD MORE CONCERNS



The client code does not know or care that the atm object is being decorated with new behavior by the aspect. The ***ProxyFactoryBean*** associates the aspect (the auditor interceptor in the ***interceptorNames*** list) with the ***AutomatedTellerMachine*** object (the

atmTarget passed to the target property). The ***ProxyFactoryBean***, therefore, decorates the behavior of the ***AutomatedTellerMachine*** with the aspect (the auditor). The auditor aspect is implemented as follows:

```
package examples.services;

import java.util.ArrayList;
import java.util.List;
import java.text.MessageFormat;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;



/** @author Richard Hightower from ArcMind Inc. */
public class AuditingUSAspect implements MethodInterceptor {

    /**
     * Gets called for every method invocation on target
     * @param methodInvocation
     * @return return code
     */
    public Object invoke(MethodInvocation methodInvocation)
        throws Throwable {
        /* Get the name of the method being invoked. */
        String operationName =
                  methodInvocation.getMethod().getName();

      /* Grab the arguments passed to the method invocation. */
        Object[] arguments = methodInvocation.getArguments();
        List argList = new ArrayList();

        if (arguments != null) {
            for (int index = 0; index < arguments.length;
                                                  index++) {
                argList.add(arguments[index]);
            }
```

```
        }

        /* Invoke the method or next Interceptor in the list */
        Object returnCode = methodInvocation.proceed();

        /* Create the message to log */
        String messageToLog =
            MessageFormat.format(
                 "{0} called with arguments {1} returned {2}",
                new Object[] {operationName,
                                argList, returnCode}
            );

        /* Log the method invocation message,
           return the results */
        ...

        return returnCode;
    }
}
```

Remember; don't sweat the details at this time as we will cover this more fully in other papers. Please notice that the auditing aspect has no real knowledge of the underlying domain. You could use this same auditing object with a different domain object.

Think of the alternative. This is the value proposition of AOP!

The alternative to this approach is to repeat the code in each method of each domain object. Now consider something less trivial like transaction management and imagine that code repeated in every method. It is error prone and repetitive to say the least. AOP is a ticket to reduce your code base and defects. **By modeling a common concern as an aspect instead of embedded in your code base, 10x, 20x, 30x savings is not of unheard of.** Also what if you need to add new concerns that you have not thought about before with AOP you create an aspect and apply it to your domain objects, you don't change the entire code base.

Please check our website for more technical papers like this one: ArcMind (http://www.arc-mind.com).

Now we setup shop in Canada, and we have to follow the Canadian auditing rules. You only have to change one line in our application context file.

```
<bean id="auditor" class="examples.services.AuditingUSAspect"/>
```

Becomes

```
<bean id="auditor" class="examples.services.AuditingCAAspect"/>
```

Basically you pass ***AuditingCAAspect*** instead of ***AuditingUSAspect***. The rest of the code does not change. The client code that uses the atm object does not even know that you changed the auditing implementation. The code that is being decorated does not know that you changed the auditing implementation either. Remember that the great thing about objects is they can be replaced!

Unsullied code

Another huge benefit to doing AOP with Spring is that it is non-destructive. When you write Java classes and compile them, and then deploy the code; your code is unsullied. Classloaders aren't polluted with incompatible versions of the same object. When you want to use AOP, the AOP proxy Java class is generated and has the modifications applied to it, but the original class as it stands is never modified. It is only changed when you want it to be, and only within the context of the Spring container.

Templates Quick Intro for Developers

Templates simplify the use of some API usually to an enterprise service. Spring templates help you to avoid common mistakes. They provide a common template for performing a task like sending a message, or querying a database. You focus on sending the message or performing the query and not handling the common invariant behavior. The invariant behavior is handled by the Spring template. If the workflow is fairly complex, you may have to implement a callback object that will be called at the appropriate time (which is another form of Inversion of Control). Spring templates are similar to the Template Method Design Pattern.

However, code that uses a Spring template may need to implement callback interfaces. Spring clearly defines the contract for dealing with its templates much more formally than the traditional template method from the classical Template Method Design

Pattern.

Spring has templates classes for dealing with Hibernate, JDO, JDBC, JMS, JMX, JNDI, iBatis, transactions and more. The concepts of templates are core to Spring. **Essentially, whenever there is an error prone workflow, Spring creates a template to allocate resources, cleanup resources and properly handle exceptions.** This allows you to write code that is not littered in try/catch/finally blocks each time you want to use an enterprise service. Templates are one of the main ways Spring makes J2EE development easier. Spring templates can greatly improve the quality of your code base by reducing defects.

By way of example, lets demonstrate using Spring JDBC template support. Concepts introduced with ***JDBCTemplate*** class will be applicable to other Spring templates. At a basic level a template can be viewed as a utility class as follows (read the comments):

```
JdbcTemplate template = new JdbcTemplate(dataSource);

//Perform a one line update
template.update(
    "update EMPLOYEE set FIRST_NAME=? where id=?",
    new Object[] {"Rick", new Integer(3)}
);

//Perform a one line query returning an int.
int count = template.queryForInt(
   "select count(*) from EMPLOYEE");
System.out.println(count);

//Perform a one line query returning a name (String)
String name =
    (String) template.queryForObject(
        "select FIRST_NAME from EMPLOYEE where id=3",
        String.class
    );
System.out.println(name);

//Get all of the employees back from the table
List employees =
    template.queryForList(
```

```
        "select id, phone, first_name, last_name from EMPLOYEE"
            );

        for (Iterator iter = employees.iterator();
                                            iter.hasNext();) {
            Map currentRow = (Map) iter.next();
            Long id = (Long) currentRow.get("id");
            String phone = (String) currentRow.get("phone");
            String firstName = (String)
                                currentRow.get("first_name");
            String lastName = (String)
                                currentRow.get("last_name");
            String message =
                MessageFormat.format(
                "Employee [id={0}, phone={1}, name={2} {3}] ",
                 new Object[] {id, phone, firstName, lastName}
                );
            System.out.println(message);
        }
```

As you can see, you can easily work with the Employee table. You can get a count of employees, update employees, query employee data, and even list employees with relative ease. Note that try/catch/finally blocks do not exist and therefore do not litter the code base, making it harder to read. The Spring JDBC template is handling the errors and cleaning up the resources appropriately. The above usage may remind you of working with a high-level scripting language instead of something you would expect with Java.

But what happens, when you want to execute a particular operation not supported by the template. You register a callback object. A callback object runs in the context of the template object. The template object calls the callback object at the appropriate time. It prepares the environment for the callback object, and cleans up after the callback object is done. Here is an example how JDBC allows you to register callback objects with it:

```
/* Get a list of employees */

        //Create callback object
        ResultSetExtractor extractor =
            new ResultSetExtractor() {
                public Object extractData(ResultSet resultSet)
```

```
                    throws SQLException, DataAccessException {
                    List employeeList = new ArrayList();

                    while (resultSet.next()) {
                        Employee employee = new Employee();
                        employeeList.add(employee);

                        employee.setFirstName(
                           resultSet.getString("first_name"));
                        employee.setLastName(
                           resultSet.getString("last_name"));
                        employee.setPhone(
                           resultSet.getString("phone"));
                    }

                    return employeeList;
                }
            };

        /* Execute callback object */
        List employeeList =
            (List) template.query(
      "select id, phone, first_name, last_name from EMPLOYEE",
                extractor
            );

        for (Iterator iter = employeeList.iterator();
                                          iter.hasNext();) {
            Employee employee = (Employee) iter.next();
            System.out.println(employee);
        }
```

The above example creates a special callback object to extract employee objects from the result set from the query. The ***JDBCTemplate*** has many types of callback object. There are even callback objects to create objects like prepared statements as follows:

```
/* Create a prepared statement and use the same extractor */
```

```
PreparedStatementCreator lookUpEmployeeLastNameLikeStatement =
   new PreparedStatementCreator() {
         public PreparedStatement
           createPreparedStatement(Connection connection)
                                          throws SQLException {
             return connection.prepareStatement(
                "select id, phone, first_name, last_name  " +
                " from EMPLOYEE where last_name like 'H%'");
         }
   };

...


/* Execute callback method */
List employeeList2 = (List) template.query(
               lookUpEmployeeLastNameLikeStatement, extractor);

for (Iterator iter = employeeList.iterator(); iter.hasNext();){
            Employee employee = (Employee) iter.next();
            System.out.println(employee);
}
```

Template objects are often constructed and passed to other objects via the Spring IoC container.

Template objects are proof positive that Spring is a lot more than a buzzword compliant framework. Spring uses the best practices of OO, IoC and AOP to simplify your development tasks. It takes design patterns a step further and makes them more applicable to your application development efforts.

Conclusion and Parting shots

Spring is much more than another buzzword compliant framework. Spring simplifies J2EE development and supports test-driven development. Spring does this by improving on OO concepts and augmenting them to make them more adaptable.

The Decorator Design Pattern is good, Spring makes its more flexible and dynamic through AOP.

The Template Method Design Pattern is a good technique for reuse, Spring makes it more adaptable via well defined contracts (callback objects).

Spring simplifies the use of many standard and de facto tools. And it does this by eating its own dog food that is by building the Spring framework on top of its implementation of IoC, AOP, Templates and good OO design principals.

AOP can ***drastically reduce code*** that would otherwise be repeated. AOP is not a replacement of OOP; it is an augmentation of OOP. AOP adds abilities that did not exist before in mainstream programming languages. AOP allows rank and file developers to add services/concerns to any object not just objects managed by a container that implement a certain interface. This allows you the ability to remove mundane code from your domain objects and easily apply future concerns without the upheaval of traditional approaches.

IoC allows you to easily assemble objects with their collaborators ***getting rid of so called wiring code that typically occupies 30% or more of many code bases***. The IoC container allows you to organize applications so that objects are created with their dependencies. This removes the object wiring associated with traditional projects and makes test driven development and unit testing much more feasible.

Spring Templates encapsulate the logic for resource management and tricky exception handling so it can be reused. This encapsulation replaces a lot of common and error prone code, thus **reducing the possibilities of defects that leak resources**.

If you are not using the Spring framework, you should start working it in on your next project or better yet your current project. Since the Spring framework is non-prescriptive, it is easy to work it into the mix.

- For more information about the Spring framework please visit:
  - http://www.springframework.org
- If you would like to read more white papers about the Spring framework, please drop us a line and we will send you other white papers that we have completed. See the page "About ArcMind" for ways to contact us.

- Special thanks to Thomas Bridges, Alex Redington, Kris Diefenderfer, *Mary Basset, Martha Pena, Andrew Barton* and Drew Davidson for their contributions to this paper.
- Special thanks to Rod Johnson, Juergen Hoeller, Colin Sampaleanu, Rob Harrop and the rest of the Spring core team for creating this amazing framework that took IoC and AOP out of the realms of academic discussion and made them mainstream. You have changed the way software will be developed for some time to come. You have made me a better developer.
- Thanks to Tom Dyer whose enthusiasm for Spring turned me on to using Spring, and Paul Tabor who let me use Spring for the first time on a project for his company.

http://www.arc-mind.com

Check our website for onsite training and whitepapers:
ArcMind (http://www.arc-mind.com).

"I attended ArcMind's training class on Spring/Hibernate - I can say without exaggeration that it is the ***best technical training*** I have ever attended! … ***I wish all classes were like this!***"

--Deepa
Senior Software Engineer, eCommerce company, Dallas, TX.

"The instructor delivered what was hands-down the ***best technical training*** I've ever received. (Spring, Hibernate, and JSF 5 day QuickStart)"

--Aaron
Principal Software Engineer, eCommerce company, Los Angeles, CA

Read the next page to learn more about ArcMind, Inc..

http://www.arc-mind.com

Check our website for onsite training and whitepapers: ArcMind (http://www.arc-mind.com).

About ArcMind

ArcMind is a premier provider of advanced training. ***ArcMind*** is a full-service, software development company that will help you and your company succeed.

ArcMind provides systems integration, consulting and mentoring services. Service to Global 1000 companies with a primary focus on J2EE, JSF, Spring, and Hibernate.

Our niche is the application of Agile practices of continuous integration, unit testing and J2EE development (soon JEE 5). Our focus on Agile Methods and TDD allows our customers to deliver business value to clients in the shortest time possible while reducing risk.

Unlike other consulting firms, our objectives do not require your company to hire more consultants. Although ***ArcMind*** is a consulting firm, our focus is on *building your team* through mentoring and training. Our experienced consultants *mentor* your team until outside resources are no longer needed.

We put members on your team "*who have been there and done that*" with J2EE/Struts, .Net and Agile Methods. We share a unified goal of keeping project control within your team and company. We mentor your team through tangible milestones so project satisfaction is assured.

ArcMind trainers are real world developers and instructors who can help you deliver your next project in a timely fashion. Our trainers are experienced developers who have worked with the technologies and have real-world experience with the techniques that they are training.

Our trainers are available to your teams as mentors and consultants saving you valuable ramp up time because they have "been there and done that before".

Developers need answers – not marketecture. ArcMind, Inc. provides an independent voice devoid of marketecture.

Technology is changing fast. Our *specialty workshops* and comprehensive educational experiences compliment your development team with the skills they need to take on tough development challenges.

Our full time **Courseware Development Team** devotes itself to keeping up-to-date curriculum available for you. We tailor courseware to suit your needs. We will build your curriculum, whether product or technology specific, from the ground up.

Web Site & ArcMind News!

Visit our site at www.arc-mind.com for information about our new Spring, Hibernate and JSF courseware. You are invited to download free technical papers on J2EE development.

This site provides detailed information on our special offers, mentoring and consulting programs, current course lists, public training schedule, individual course details, and licensing information.

Our Java Curriculum

- Java Sever Faces (JSF)
- Java Data Objects
- Hibernate (2 or 3)
- Spring (covers IOP, and AOP)
- Tapestry
- Ajax
- EJB 3

ArcMind
Office 520.290.6855
info@Arc-Mind.com
http://www.Arc-Mind.com

What people are saying about ArcMind Training and consulting:

"ArcMind taught a one week long Spring/Hibernate class for us at Sabre. They did a great job keeping the class entertaining while covering a difficult material. ArcMind's class covers in great details the topic at hand and includes numerous labs to help practice each lesson. Highly recommended!"
– Jacques Morel (Chief Architect, Sabre Systems, Dallas TX)

"We hired ArcMind at the beginning of a major Java development initiative. This was some of the best money we have spent. Their consulting on tool selection, organization of our project and other areas laid a foundation that we continue to benefit from. They worked with us onsite, remotely, and through an ongoing relationship."
–Tim Hoyt (CTO of Picture Marketing, Miami, FL).

"(The instructor's) knowledge of J2EE, Java and Open Source technologies is both deep and wide. His training and mentoring skills are the best that I've come across. His passion and proficiency has been inspiring for me and many others in the development community. ArcMind's input and guidance was crucial when my organization needed to determine a strategy for leveraging new technologies a few years ago."
–Tom (Senior Software Developer, Boston MA)

"I've been using them for a while, so when I was sent to ArcMind's Spring/Hibernate class, I thought it'd be easy. Turns out I learned a lot. The coursework was well organized and clearly presented with lots of labs. More importantly, the instructor shared his extensive knowledge and experience in using these products in real-world projects. Highly recommended, even for experienced Spring/Hibernate users!"
– Rong (Senior Software Engineer at an eCommerce company Dallas, TX)

"The instructor is an expert with in-depth knowledge and understanding of the J2EE concepts and technologies. Additionally his expert advice on the integration of Spring and Hibernate was a great help."
--Rohit (Principal Software Engineer at a Manufacturing company, Toronto Canada)

"The instructor was an excellent teacher, thoughtful architect/programmer and absolutely knows his business inside and out. I would highly recommend ArcMind to anyone!"
--Danilo (Principal Software Engineer, CA)

"Well when it come to Hibernate/Spring, I feel (the ArcMind instructor) is someone who is like 'Been There Done That' kind of a guy. I really enjoyed his sessions."
– **Prasanth** (Senior Software Developer)

More endorsements available upon request.