Aspect Oriented Programming

Separate out cross-cutting concern code into their own classes or modules, called aspects.
Example: Logging

- LogTask1 and LogTask2
  - Both implement LogTask interface
  - LogTask1 – mixes application code and logging code
  - LogTask2 – only has application code

How Does it Work?

```java
class Foo {
    Object bar() {...}
}
```

```
class Foo1 extends Foo {
    void bar() {
        // some code before super.bar()
        ...
        Object o = super.bar();
        // some code after super.bar();
        ...
        return o;
    }
}
```

Proxy – Subclass

```
class Foo2 {
    private Foo foo;
    void bar() {
        // some code before super.bar()
        ...
        Object o = foo.bar();
        // some code after super.bar();
        ...
        return o;
    }
}
```

Proxy – Wrapper Class

```
class Foo2 {
    private Foo foo;
    void bar() {
        // some code before super.bar()
        ...
        Object o = foo.bar();
        // some code after super.bar();
        ...
        return o;
    }
}
```

Create Proxies Automatically - ProxyFactoryBean

```
<bean id="loggedTask2" class="cs520.spring.aop.LoggedTask2" />
<bean id="loggedTask2WithAdvice" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
        <list>
            <value>cs520.spring.log.LoggedTask</value>
        </list>
    </property>
    <property name="target" ref="loggedTask2" />
    <property name="interceptorNames">
        <list>
            <value>loggingAdvice</value>
        </list>
    </property>
</bean>
```

Some AOP Terminology

- Target
- Proxy
- Proxy Interface
- Advice
More AOP Terminology

- Join point: a point in the execution of the application where the advice can be plugged in
- Pointcut: A predicate that determines join points
- Introduction: adding new methods and/or fields to existing classes
- Weaving
  - Compile time, class load time, or runtime

Spring AOP

- Advices are written in Java
- Pointcuts are defined in XML or annotation
- Supports only method join points
- Aspects are woven in at runtime
- Advisor = Advice + Pointcuts

Advice Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Around</td>
<td>org.aopalliance.intercept.MethodInterceptor</td>
</tr>
<tr>
<td>Before</td>
<td>org.springframework.aop.BeforeAdvice</td>
</tr>
<tr>
<td>After</td>
<td>org.springframework.aop.AfterReturningAdvice</td>
</tr>
<tr>
<td>Throws</td>
<td>org.springframework.aop.ThrowsAdvice</td>
</tr>
</tbody>
</table>

Use Interceptor

```java
public class LoggingInterceptor implements MethodInterceptor {
    public Object invoke( MethodInvocation invocation ) throws Throwable {
        return result;
        ...
        Object result = invocation.proceed();
        ...
        return result;
    }
}
```

Configure Pointcuts

- NameMatchMethodPointcutAdvisor
- RegExpPointcutAdvisor

```
<bean id="loggingAdvisor"
     class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
    <property name="mappedName" value="\dSomething\" />
    <property name="advice" ref="loggingInterceptor" />
</bean>
```

AOP Annotations

- Spring Framework Reference Documentation, Chapter 8.2
About AOP

- Good??
- Bad??