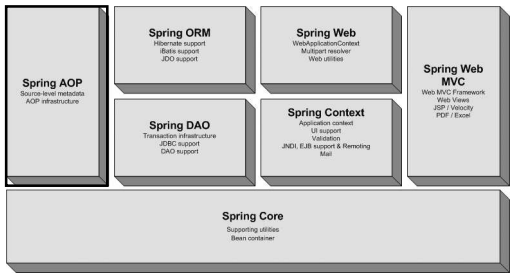
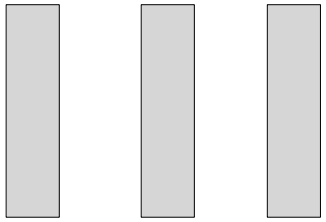
CS520 Web Programming
Spring – Aspect Oriented Programming

Chengyu Sun
California State University, Los Angeles

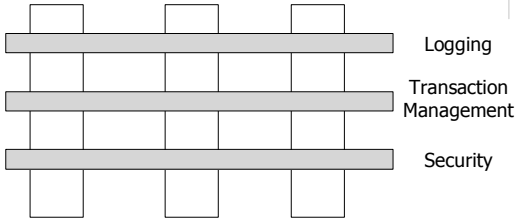
Spring Framework



Concerns



Cross-Cutting Concerns



An Example of Transaction Rollback

```
catch( SQLException e ) {
    System.err.println( e.getMessage() );
    if( c != null ) {
        try {
            c.rollback();
         } catch( SQLException ex ) {
            System.err.println( ex.getMessage() );
        }
    }
}
```

Aspect Oriented Programming

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

Example: Logging

- `LoggedTask1` and `LoggedTask2`
  - Both implement `LoggedTask` interface
  - `LoggedTask1` – mixes application code and logging code
  - `LoggedTask2` – only has application code

How Does it Work?

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

run some code before bar() is called

run some code after bar() is called

Proxy – Subclass

```
class Foo1 extends Foo {

   void bar()
   {
      // some code before super.bar()
      ...
      O o = super.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;
   }

}
```

Java Syntax Issue

- Can we inject an object of *Foo1* or *Foo2* instead of an object of *Foo*??

```
Foo foo;

public void setFoo( Foo foo )
{
   this.foo = foo;
}

...
foo.bar();
```

Java Syntax Issue – Make It Work

```
public Interface FooInterface
{
   void bar();
}
```

```
FooInterface foo;

public void setFoo( FooInterface foo )
{
   this.foo = foo;
}

...
foo.bar();
```

Some AOP Terminology

- Target
- Proxy
- Proxy Interface
- Aspect
- Advice

Create Proxies Automatically - ProxyFactoryBean

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

<bean id="loggedTaskTarget" class="cs520.spring.log.LoggedTask2" />
```

More AOP Terminology

- Join point: a point in the execution of the application where the aspect can be plugged in
- Pointcut: determines what advice(s) should be applied at what 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 configuration file
- Supports only method join points
- Aspects are woven in at runtime
- Advisor = Aspects + Pointcuts

Advice Types

Type	Interface
Around	org.aopalliance.intercept.MethodInterceptor
Before	org.springframework.aop.BeforeAdvice
After	org.springframework.aop.AfterReturningAdvice
Throws	org.springframework.aop.ThrowsAdvice

Use Interceptor

```
public class LoggingInterceptor implements MethodInterceptor {

    public Object invoke( MethodInvocation invocation ) throws Throwable
    {
        // do something before method invocation
        …

        Object result = invocation.proceed();

        // do something after method invocation
        …

        return result;
    }
}
```

Configure Static Pointcuts

- NameMatchMethodPointcutAdvisor
- RegExpPointcutAdvisor

AutoProxying

- BeanNameAutoProxyCreator
- DefaultAdvisorAutoProxyCreator

About AOP

- Good??
- Bad??