Aspect-Oriented Programming with Spring AOP and AspectJ

Adrian Colyer
Interface21

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Agenda

- Simplifying enterprise application development
  - AOP by example
- The goals of AOP
- Spring AOP and AspectJ
- Introducing AOP into your own projects
  - a roadmap
  - with plenty of examples

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The Spring approach

- Program simply using objects
  - aka "POJOs"
  - non-invasive
  - object-oriented
- Retain architectural choice
  - no environmental assumptions or dependencies in application objects
- Facilitate test-driven development

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The Spring approach

```
            t
   f        e
   l        s
object-oriented
   x        a
  simple    b
   b        l
   l        e
   e
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

The Spring approach

Simple does not mean

simplistic

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

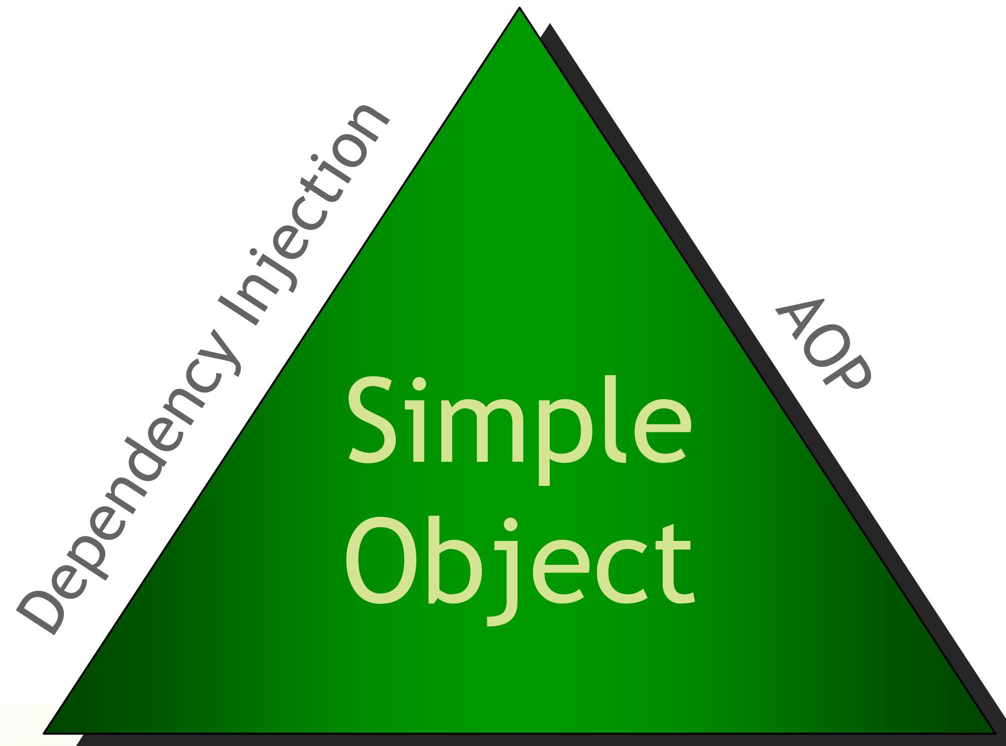
The Spring approach

Simpler can be more

powerful

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Enabling Technologies



Portable Service Abstractions

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

AOP

- Aspect Oriented Programming (AOP)
  - Enables declarative specification of enterprise services
  - Enables modular implementation of requirements that impact multiple points in the application

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Enterprise application vocabulary

the **vocabulary** of enterprise applications

business service

service layer

repository

dao

controller

web layer

data access layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed in terms of this vocabulary
  - the service layer should be transactional
  - when a Hibernate dao operation fails the exception should be translated
  - service layer objects should not call the web layer
  - a business service that fails with a concurrency related failure can be retried

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Meaningful abstractions

It would be simpler…

and more *powerful*

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Meaningful abstractions

if we could use these

abstractions

directly in the

implementation

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

System Architecture

```
@Aspect
public class SystemArchitecture {

  @Pointcut("within(a.b.c.service..*")
  public void inServiceLayer() {}

  @Pointcut("within(a.b.c.dao..*")
  public void inDataAccessLayer() {}

  @Pointcut("execution(* a.b.c.service.*.*(..))")
  public void businessService() {}

  …
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Requirements

- Many requirements are expressed in terms of this vocabulary
  - the service layer should be transactional
  - **when a Hibernate dao operation fails the exception should be translated**
  - service layer objects should not call the web layer
  - a business service that fails with a concurrency related failure can be retried

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Scenario…

- You have your own data access layer written using Hibernate 3
  - not using the Spring HibernateTemplate (yet?)
- In the service layer, you want to insulate yourself from Hibernate exceptions, and take advantage of Spring's fine-grained DataAccessException hierarchy
- After throwing a hibernate exception from a data access operation, convert it into a DataAccessException…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

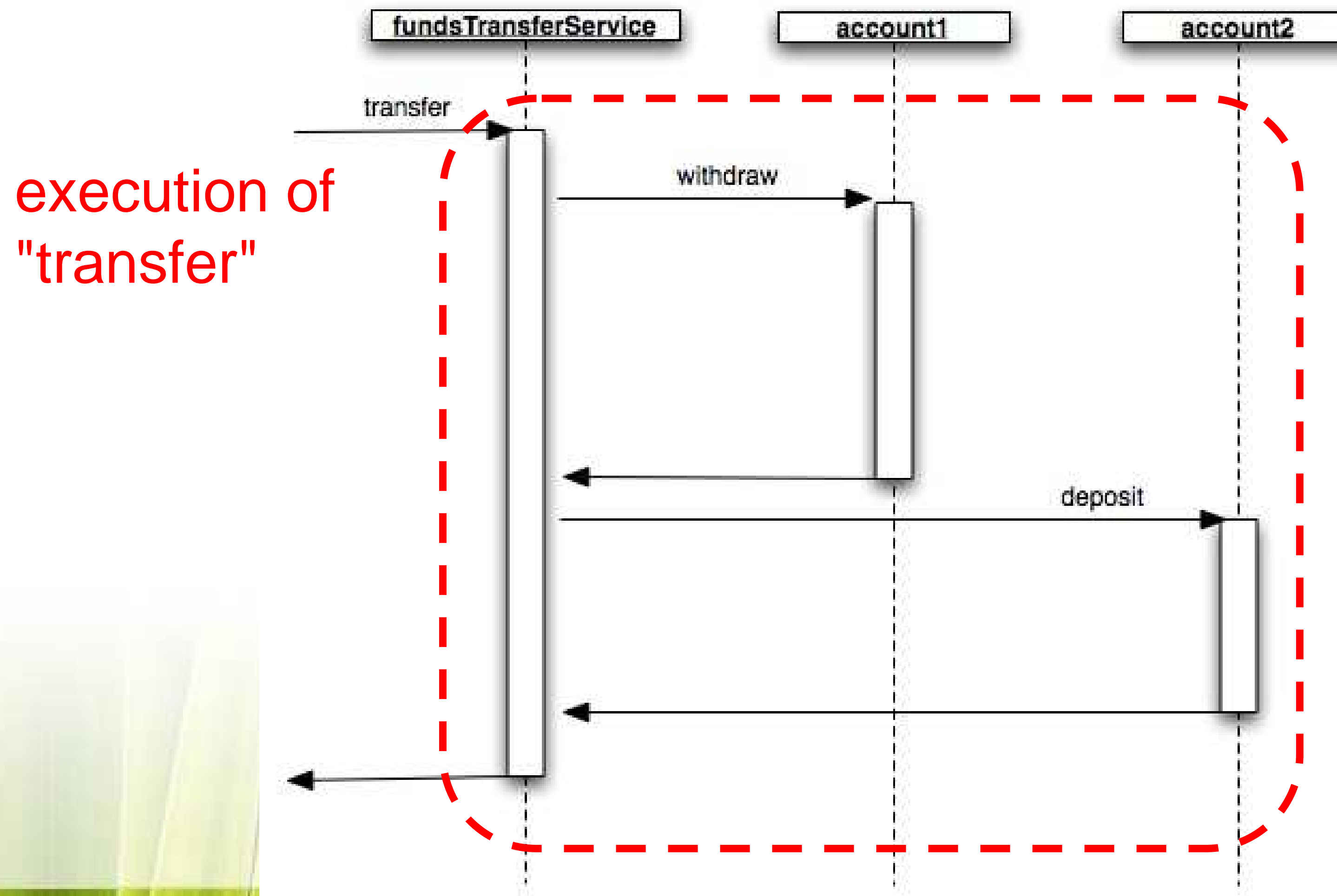
Spring

Step 1: Define "dataAccessOperation"

- The requirements talk about "data access operations"
  - In AOP terminology, the execution of a data access operation is a join point

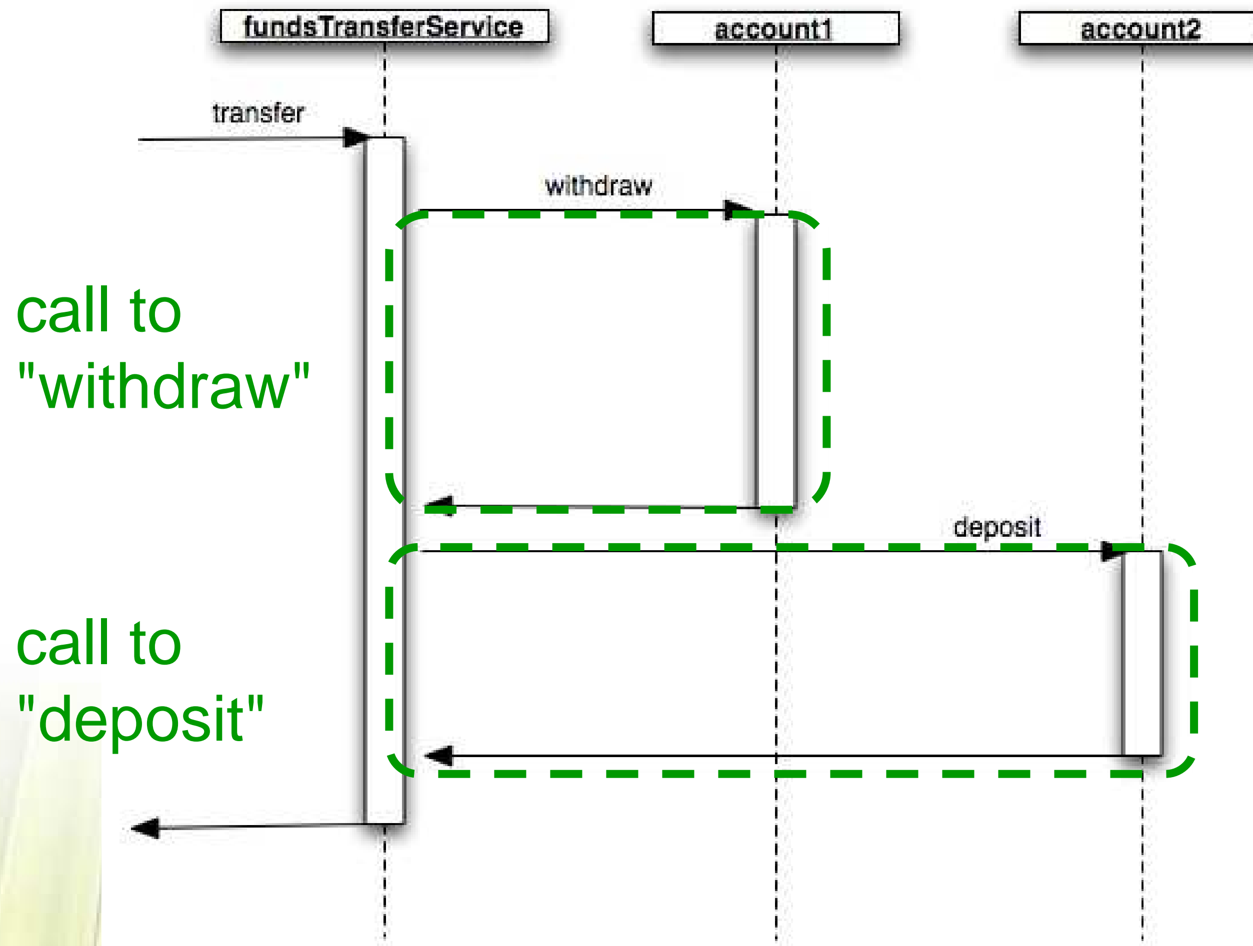
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Understanding Join Points



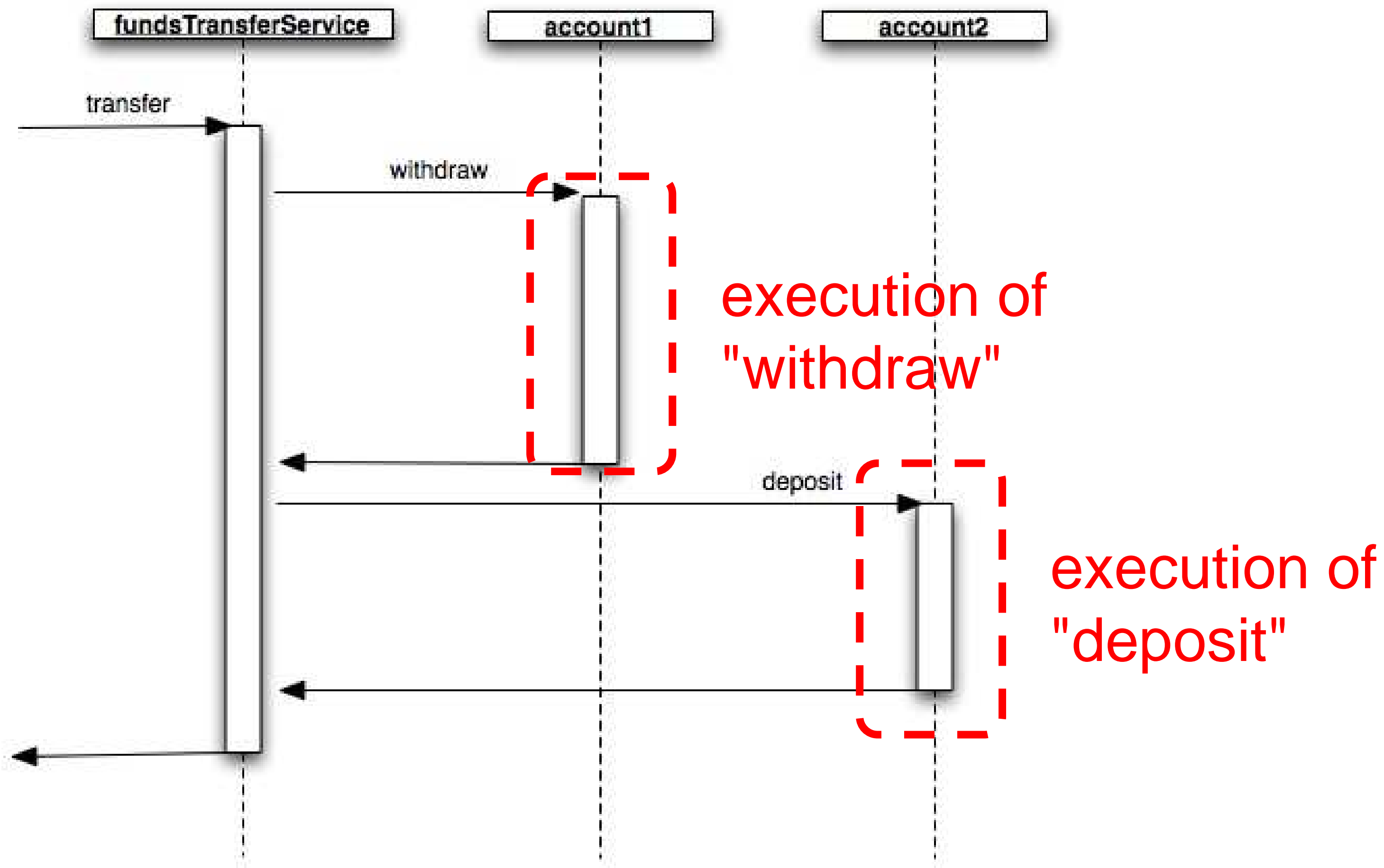
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Understanding Join Points



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Understanding Join Points



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Pointcuts

- The join points that we are interested in are those that represent the execution of a data access operation
- We can write a predicate, known as a pointcut, to match them

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Pointcuts

```
@Pointcut("execution(* a.b.c.dao..*.*(..))")
public void dataAccessOperation() {}
```

```
public pointcut dataAccessOperation() :
  execution(* a.b.c.dao..*.*(..))
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Pointcuts

```
public pointcut dataAccessOperation() :
  execution(* (@Repository *).*(..));
```

- dataAccessOperation is an abstraction we can program against
  - the details of how it is defined are hidden

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Step 2: Use the abstraction

- How do we make use of this dataAccessOperation abstraction?
- Advice!
- Advice is associated with a pointcut expression
- Executes every time a join point matched by the expression occurs

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Advice

- Advice is like a method
  - except that it is never explicitly invoked
- Five kinds of advice:
  - before
  - after returning
  - after throwing
  - after
  - around

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Which advice kind?

- "After throwing a hibernate exception from a data access operation, convert it into a DataAccessException…"

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

After throwing

```
@AfterThrowing(
  throwing="hibernateEx",
  pointcut="SystemArchitecture.dataAccessOperation()"
)
public void rethrowAsDataAccessException(
  HibernateException hibernateEx) {
  // convert exception and rethrow…
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Where does advice live?

- Advice is declared in an aspect
- Aspect are like classes
  - instances
  - state (fields)
  - behaviour (methods)
- Aspects can also have
  - pointcuts
  - advice
  - and a few other things…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Aspect

```
@Aspect
public class HibernateExceptionTranslator {
    // …

    @AfterThrowing(
     throwing="hibernateEx",
     pointcut="SystemArchitecture.dataAccessOperation()"
    )
    public void rethrowAsDataAccessException(
     HibernateException hibernateEx) {
     // convert exception and rethrow…
    }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Step 3: Configuration

```
<aop:aspectj-autoproxy/>

<bean id="hibernateExceptionTranslator"
   class="HibernateExceptionTranslator">
     <property name="hibernateTemplate"
                       ref="hibernateTemplate"/>
</bean>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Schema alternative

- For JDK 1.4 and below
  - and annophobes ;)
- The exact same aspect can be declared in Spring XML, backed by a plain Java class

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Schema-based configuration

```
<aop:config>

  <aop:aspect ref="hibernateExceptionTranslator">
    <aop:after-throwing
       throwing="hibernateEx"
       pointcut="SystemArchitecture.dataAccessOperation()"
       method="rethrowAsDataAccessException"/>
  </aop:aspect>

</aop:config>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Bean Implementation

```
public class HibernateExceptionTranslator {
  private HibernateTemplate hibernateTemplate;

  public void setHibernateTemplate(
    HibernateTemplate aTemplate){
    this.hibernateTemplate = aTemplate;
  }

  public void rethrowAsDataAccessException(
      HibernateException hibernateEx) {
    throw this.hibernateTemplate
        .convertHibernateAccessException(hibernateEx);
  }
}
```

parameter bound in pcut expression

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed in terms of this vocabulary
  - the service layer should be transactional
  - when a Hibernate dao operation fails the exception should be translated
  - service layer objects should not call the web layer
  - **a business service that fails with a concurrency related failure can be retried**

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Retry

- One subtype of DataAccessException is…
  - ConcurrencyFailureException
- Concurrency failures are potentially recoverable
  - If the operation is idempotent, we can retry it*
- We need a ConcurrentOperationExecutor…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Concurrent Operation Recovery

```
public class ConcurrentOperationExecutor {
  private static final int DEFAULT_MAX_RETRIES = 2;
  private int maxRetries = DEFAULT_MAX_RETRIES;

  /** configurable via dependency injection */
  public void setMaxRetries(int numTimesToRetry) {
    this.maxRetries = numTimesToRetry;
  }

 …
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Concurrent Operation Recovery

```
public Object doConcurrentOperation(ProceedingJoinPoint pjp)
  throws Throwable {
   int numAttempts = 0;
   ConcurrencyFailureException failureException;
   do {
     numAttempts++;
     try { return pjp.proceed(); }
     catch(ConcurrencyFailureException ex) {
       failureException = ex;
     }
   }
   while(numAttempts <= this.maxRetries);
   throw failureException;
  }

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring

Concurrent Operation Recovery

```
<aop:config>

  <aop:aspect ref="concurrentOperationExecutor">
    <aop:pointcut id="idempotentOperation"
      expression= "SystemArchitecture.businessService()"/>

    <aop:around
        pointcut-ref="idempotentOperation"
        method="doConcurrentOperation"/>
  </aop:aspect>

</aop:config>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Concurrent Operation Recovery

```
<bean id="concurrentOperationExecutor"
    class="ConcurrentOperationExecutor">
    <property name="maxRetries" value="3"/>
</bean>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Recap:

- Created an abstraction: idempotentOperation
- Used around advice to retry failing idempotentOperations
- Packaged in a ConcurrentOperationExecutor aspect
- Configured using Spring

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Idempotent operations

- It would be nice if all of our service layer operations were idempotent
- But what if some of them aren't?
- We'd need a way to identify and retry only the idempotent subset…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Idempotent operations

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Idempotent {}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Idempotent operations

- Just update the abstraction (pointcut expression)…

```
<aop:pointcut id="idempotentOperation"
    expression=
        "SystemArchitecture.businessService()
         and @annotation(Idempotent)"/>
```

- easier than APT??? ;)

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The goals of AOP

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The DRY Principle

"Every piece of system knowledge should have a single, authoritative, unambiguous representation"

- Dave Thomas

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Separation of Concerns

- “But nothing is gained --on the contrary!-- by tackling these various aspects simultaneously. It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of.”

  - Edsger W. Dijkstra, 1974

  On the role of scientific thought

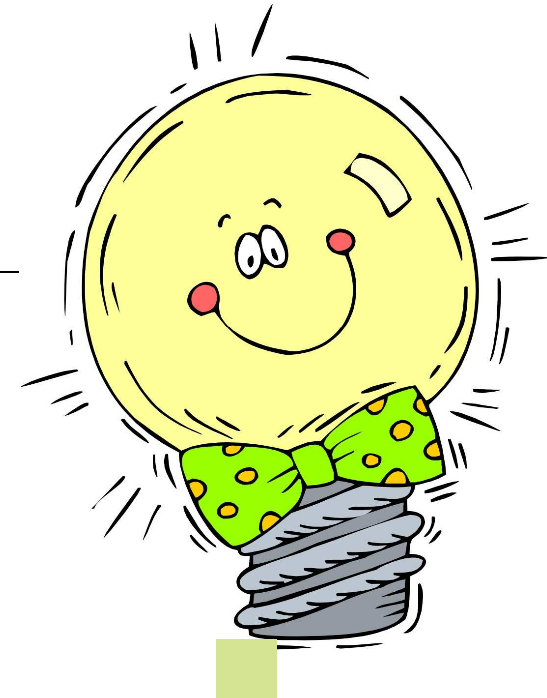
- A software engineering principle, each module
  - does one thing
  - knows one thing
  - keeps the secret of how it does it hidden

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Tip

- Always strive for

DRY SoC(k)s.



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The 1:1 principle

- If you…
  - Don't Repeat Yourself
- and you..
  - maintain a clear Separation of Concerns
- then you end up with…
  - The 1:1 principle

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The 1:1 principle

"There should be a clear, simple, 1:1 correspondence between design-level requirements and the implementation"

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

It's hard to stay dry sometimes

- If the state of a domain object changes, mark it as dirty
- All service layer operations should take place inside a transaction
- Expose usage statistics for all data-access objects
- Initialize resources lazily on first use

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

When dry breaks down…

- The examples all have the following in common:
  - a *single requirement* cannot be represented in a single, authoritative, unambiguous place in the system
  - the *system knowledge* is diffused and scattered throughout the implementation

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Common violations of 1:1

- 1:n
  - a single design requirement is *scattered* across multiple parts of the implementation
- n:1
  - implementation details for multiple requirements are *tangled* up inside the same module
- 1:0
  - it was too hard to implement so we didn't bother
- n:n
  - it's all mixed up… (the normal case!)

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The aspects value-proposition

- AOP is fundamentally a modularity technology
  - from 1:n -> 1:1
- A new kind of module, called an *aspect* encapsulates the implementation of requirements that cut-across many parts of the system
  - these are sometimes called "cross-cutting concerns"
- Improved… agility, testability, maintainability, option value, reuse, flexibility, quality, 100m sprint times, …

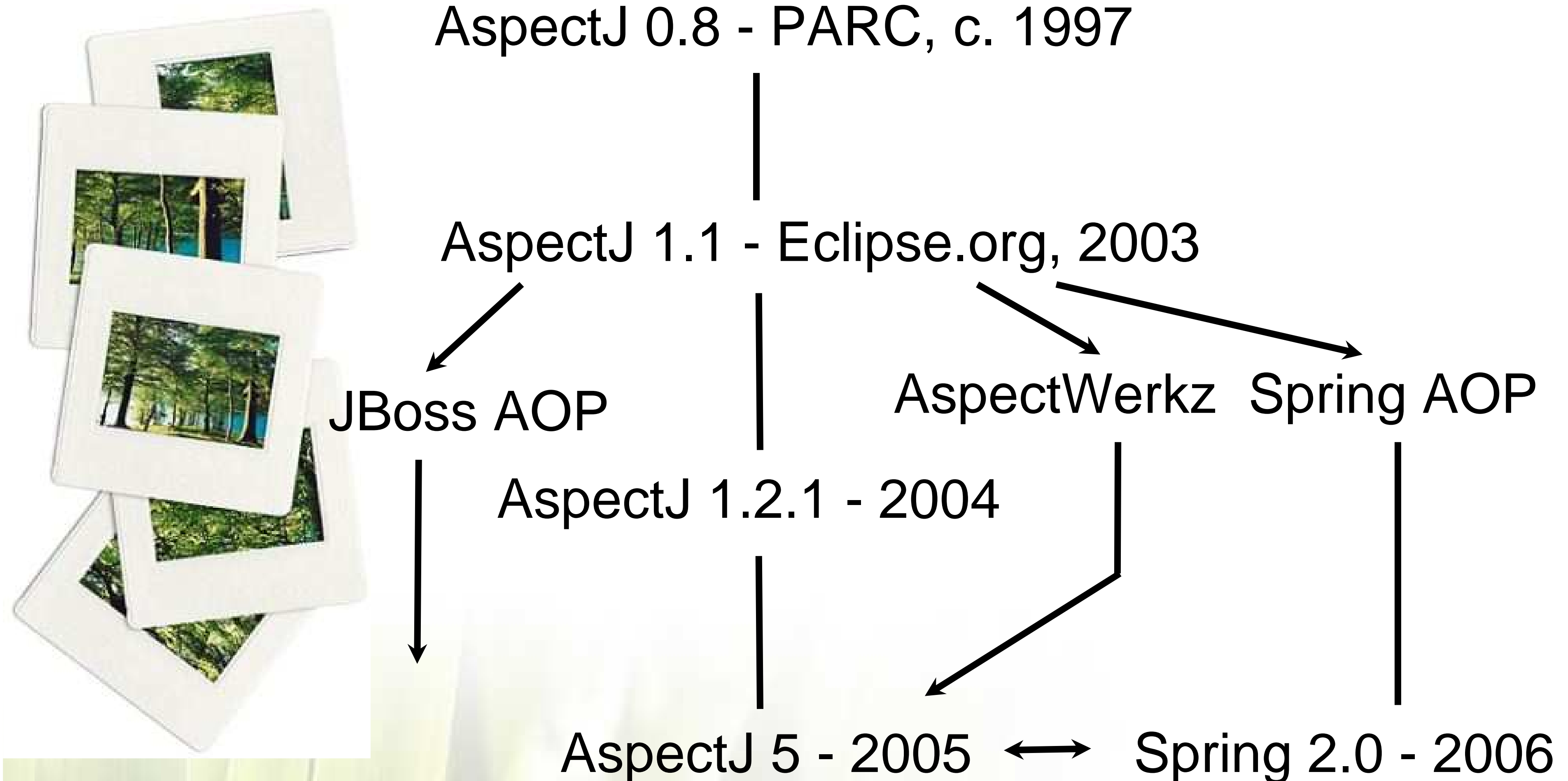
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

AOP is about modularity

- Well defined modules (aspects)
- With well-defined semantics
- Fully type checked
- Not "just another framework"

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

The AOP family tree (industry)



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring AOP and AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

AspectJ: A Programming Language

- Every legal Java program is a legal AspectJ program
- Compilation produces standard class files that can be run on any JVM
- Strongly typed
- The best performing AOP implementation
- Rich pointcut language
- Sophisticated IDE support in Eclipse (AJDT)

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

AspectJ language

- Compilation with ajc

```
public aspect HelloFromAspectJ {

  public pointcut main() :
     execution(* main(..))")

  after() returning : main() {
    System.out.println("Hello from AspectJ!");
  }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

@AspectJ development style

- Supports regular compilation with javac
  - (followed by binary weaving)

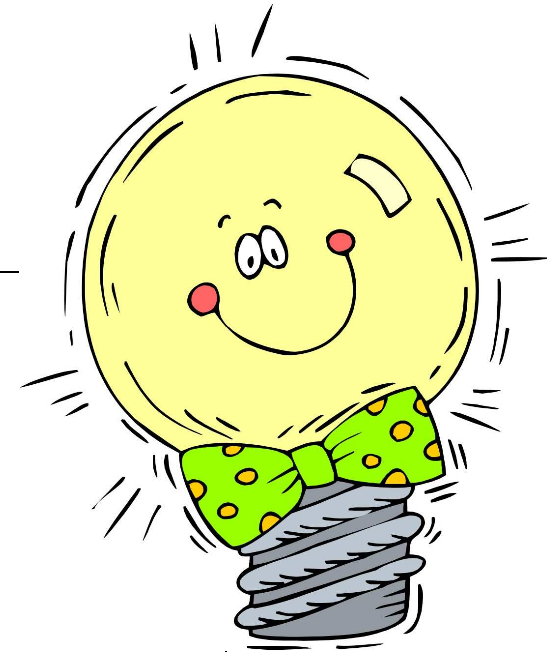
```
@Aspect
public class HelloFromAspectJ {

  @Pointcut("execution(* main(..))")
  public void main() {}

  @AfterReturning("main()")
  public void sayHello { … }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Tip



- If you are developing @AspectJ style aspects you can use AJDT and get *compile-time support* for pointcut expressions and cross-reference information in the IDE.

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

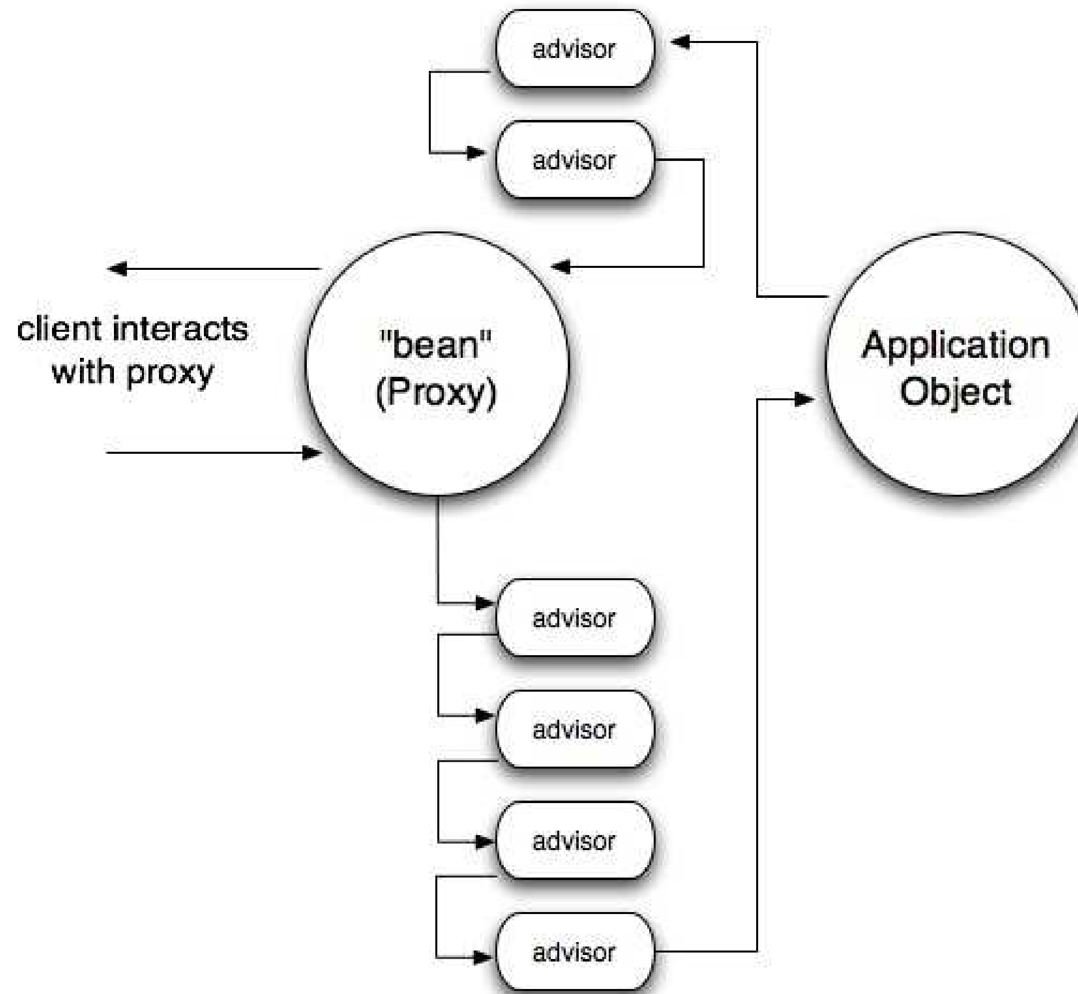
AspectJ weaving

- AspectJ supports:
  - compile-time weaving (source -> .class),
  - binary weaving (.class -> .class),
  - and load-time weaving

- ajc command-line compiler
- ajdoc javadoc tool
- ant task
- maven support also available

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring AOP: Proxy-based system



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring 2.0 AOP

- Aspects are defined in Spring configuration file
  - supports both XML based definition
  - and @AspectJ aspects!
- XML defined aspects are backed by regular classes

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring 2.0 and AspectJ

- Spring and AspectJ are still distinct projects
- Spring just uses the AspectJ pointcut parsing and matching APIs
  - using AspectJ as a library, not as a weaving engine
- Gives the same syntax and semantics across Spring AOP and AspectJ
  - perfect if you are going to use both
  - or start out with Spring AOP, and then want to introduce AspectJ at some point

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

XML based aspect definition

```
<aop:config>
  <aop:aspect ref="helloAspect">
    <aop:pointcut id="main"
      expression="execution(* main(..))"/>
    <aop:after-returning
      pointcut-ref="main"
      method="sayHello"/>
  </aop:aspect>
</aop:config>

<bean id="helloAspect" class="…"/>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

"Aspect" class

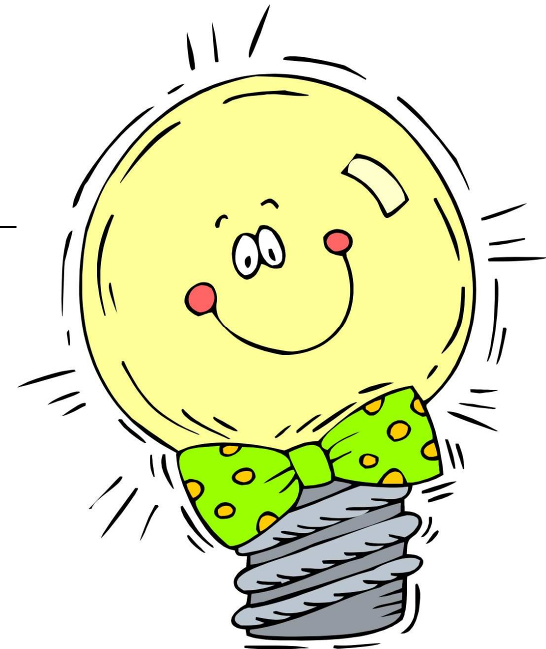
```
public class HelloWorldService {

   public void sayHello() {
     System.out.println("hello");
   }

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Tip



- The pointcut expression used by Spring AOP doesn't just *look like* AspectJ, it *is* AspectJ.

So everything you learn about AspectJ pointcuts applies to Spring AOP too

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Execution Join Points only

- But Spring AOP can only support one join point kind: method-execution join points.

So if you use call, get, set etc. you'll get an 'IllegalArgumentException'

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Using @AspectJ aspects

```
<aop:aspectj-autoproxy/>

<bean id="myAspect"
   class="org.sfw…MyAspect/>
```


Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21

A typical enterprise app…

View

Text Area

Cancel OK

Web Tier Actions

Action A

Action B

Business Services

Service 1

Service 2

Service 3

Data Access

DAOs/ Repositories

ent

Domain model

Class Name
Attributes
Attributes

Class Name
Attributes
Attributes

Class Name
Attributes
Attributes

Class Name
Attributes
Attributes

Copyright 2004
distributing without expressed written permission is prohibited.

Spring

INTERFACE21

Spring managed

View

Text Area

Cancel OK

Web Tier Actions

Action A

Action B

Business Services

Service 1

Service 2

Service 3

Data Access

DAOs/ Repositories

ient

Domain model

Class Name
Attributes
Attributes

Class Name
Attributes
Attributes

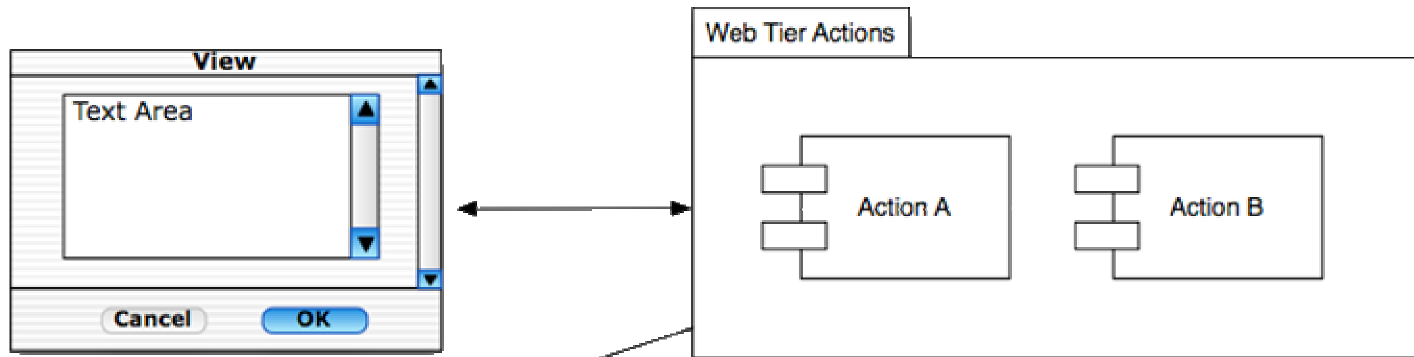
Class Name
Attributes
Attributes

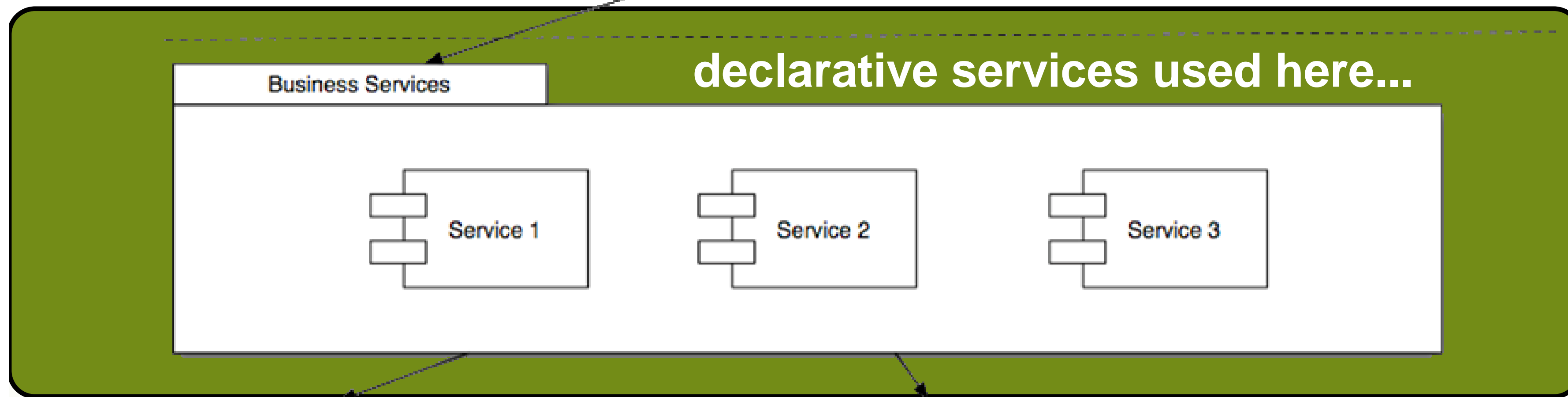
Class Name
Attributes
Attributes

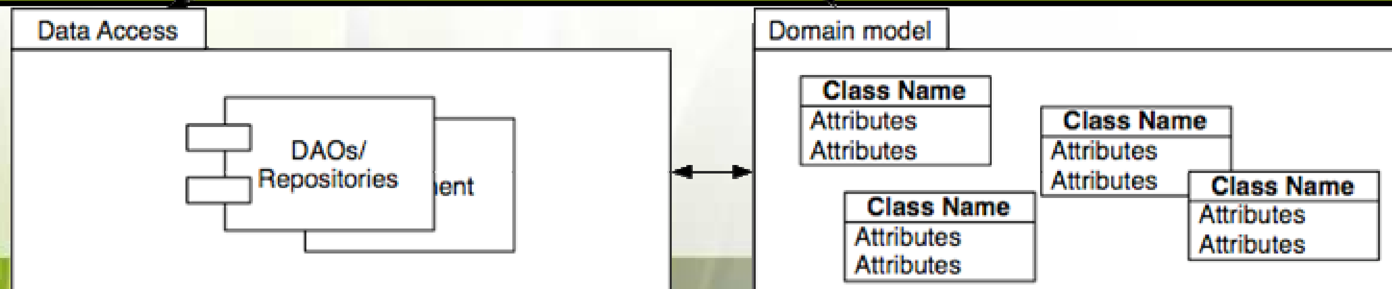
Copyrig[illegible]
distributing without expressed written permission is prohibited.

Spring

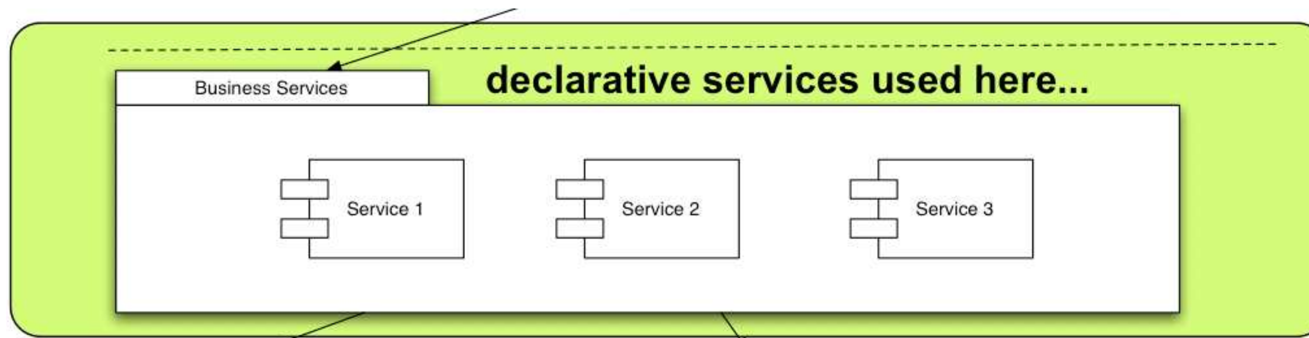
A typical enterprise app...







Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.



- The business services layer is where declarative services are often used
  - transactions
  - security
- Natural fit for Spring AOP
  - Spring already managing instantiation and configuration
- This support is mature and widely deployed
  - use immediately

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Business Services

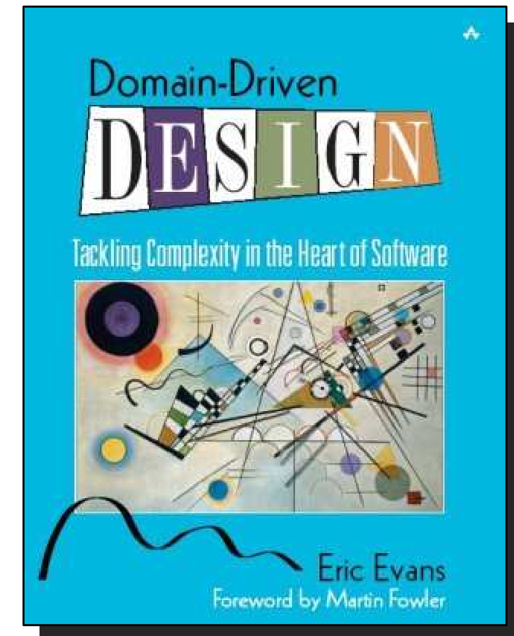
- Other Spring AOP based facilities you may consider using here
  - JMX support
  - Remoting
  - Lazy instantiation
- Spring AOP good fit for:
  - Spring managed beans
  - execution based join points
  - coarser-grained service application

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

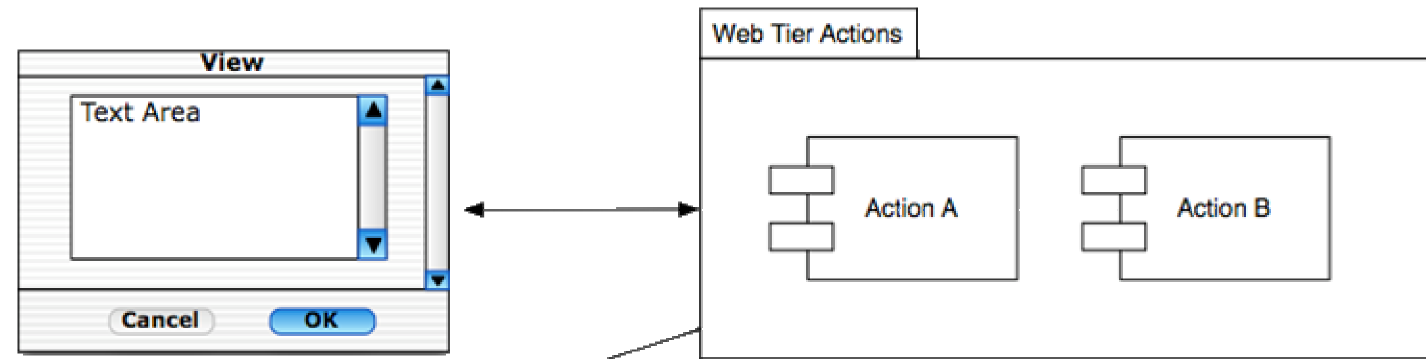
Separation of concerns again

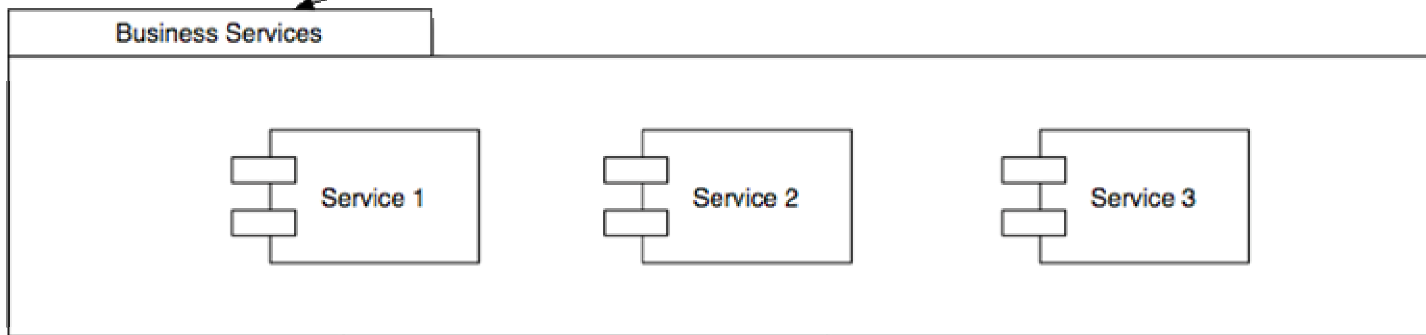
- Here's a way to think about it...
- We want to separate the
  - business concerns
  - from the technical concerns
- Business services, domain objects, and DAO interfaces
  - written in terms of the UBIQUITOUS LANGUAGE

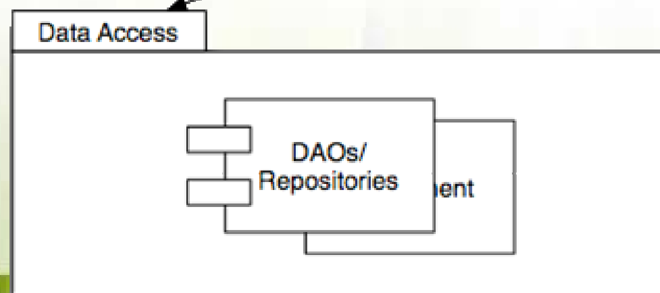


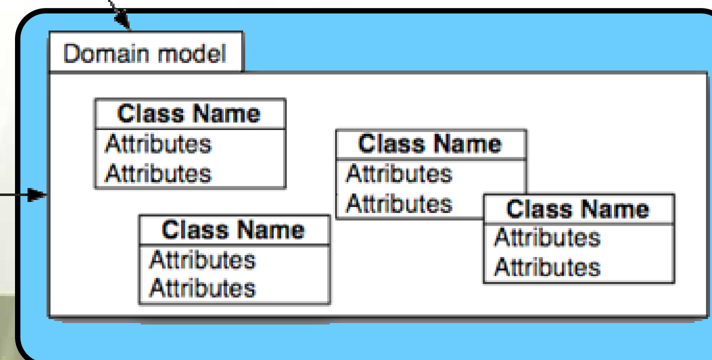
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

A typical enterprise app...









Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring

Domain model

- The domain model is classic territory for AspectJ
  - fine-grained objects
  - created and managed outside of Spring container control
- Also use AspectJ for
  - pervasive aspects across the rest of the system
  - needing richer join point model
    - e.g. *calls* to third-party libraries...
  - or performance sensitive

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Part 1 : Summary

- We want to implement enterprise requirements in as simple and straightforward a manner as possible
  - use the appropriate implementation "vocabulary"
- AOP is about modularity
  - getting closer to the ideal of the "1:1 principle"
- AspectJ and Spring AOP are the leading AOP implementations
  - tightly integrated
- Can use together or independently

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

In Part 2…

- How to start simplifying your own projects using AOP
  - an adoption roadmap
- With more concrete examples…
- Plus:
  - What about EJB3?, and…
  - I hear you teach a course on this stuff?

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Part 2: Adopting AOP

Adrian Colyer
Interface21

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Adoption Roadmap

1. Use the out-of-the-box Spring aspects
   - transactions etc.
2. Introduce your own Spring-AOP based aspects as needed
3. Add enforcement and exploration aspects using AspectJ
4. Add infrastructure aspects using AspectJ
5. Add domain aspects using AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Adopting Spring-AOP

- No tools issues
  - use standard java compiler and language
- No extra compilation/weaving steps
- Runtime proxy framework already well tested in many large production applications
- If Spring-AOP can handle your requirements, it's the natural choice

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Requirements

- Many requirements are expressed in terms of this vocabulary
  - **the service layer should be transactional**
  - when a Hibernate dao operation fails the exception should be translated
  - service layer objects should not call the web layer
  - a business service that fails with a concurrency related failure can be retried

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Transactions

```
<aop:config>
  <aop:advisor
    pointcut="SystemArchitecture.businessService()"
    advice-ref="tx-demarcation"
  />
</aop:config>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Transactions

```
<tx:advice id="tx-demarcation">
  <method name="*"
    propagation="REQUIRED"
    isolation="DEFAULT"/>
</tx:advice>
```

- Gives us TX-REQUIRED semantics for the service layer

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Read-only operations

- Important to distinguish read-only operations for performance reasons
- We could…
  - list them all in a pointcut expression
  - use a naming pattern

    <method name="get*" … />
  - use annotations
    - either @ReadOnly on operations, or…
    - annotation-driven transactions

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Transactional annotation

```
/**
 * default to required, read-write for all
 *  operations
 */
@Transactional
public class AccountService {
  /** this one is read-only… */
  @Transactional(readOnly=true)
  public Account getAccount(AccountNum accNo) {
   …
  }
  …
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Transactional annotation

- Now the configuration just becomes…

<!– tell Spring to perform transaction demarcation on bean operations based on @Transactional annotations -->

<tx:annotation-driven/>

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Adoption Roadmap

1. Use the out-of-the-box Spring aspects
   - transactions etc.
2. **Introduce your own Spring-AOP based aspects as needed**
3. Add enforcement and exploration aspects using AspectJ
4. Add infrastructure aspects using AspectJ
5. Add domain aspects using AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring AOP aspects

- Already seen two examples:
  - Exception translation
  - Concurrent operation retry

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Schema or @AspectJ?

- If you can, use @AspectJ
  - richer support for pointcut composition than schema
  - supports aspect instantiation models
- @AspectJ aspects have the advantage that they can be understood by AspectJ too
  - can also make use of AJDT tools in development

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Example: Security

- Security impacts all architectural layers
- Conceptually…
  – before a securedOperation() we need to authenticate and authorize
  – after returning we may need to filter results or deny access based on credentials of the requesting Principal

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Security Protocol (acegi)

```
@Aspect
public abstract class AbstractAccessControlManager
extends AbstractSecurityInterceptor{
  …

  @Pointcut("")
  protected abstract void securedOperation();

  @Around("securedOperation()")
  public void doSecuredOperation(ProceedingJoinPoint pjp){
    Object result = null;
    InterceptorStatusToken token = super.beforeInvocation(pjp);
    try { result = pjp.proceed(); }
    finally { result = super.afterInvocation(token,result); }
    return result;
  }

}
```


Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Service layer

```
@Aspect
public class ServiceLayerAccessControl extends
   AbstractAccessControlManager {

  @Pointcut("SystemArchitecture.businessService()")
  protected void securedOperation() {}

}
```


Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Configuration

```
<aop:aspectj-autoproxy/>

<bean id="serviceLayerAccessControl"
  class="abc.ServiceLayerAccessControl">
  <property name="authenticationManger"  ref="authenticationManager"/>
  <property name="accessDecisionManager"
            ref="serviceLayerAccessDecisionManager"/>
  <property name="afterInvocationManager"
            ref="resultAccessControlDecisionManager"/>
  <property name="securityAttributes"
            ref="serviceLayerSecurityAttributes"/>
</bean>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Adoption Roadmap

1. Use the out-of-the-box Spring aspects
   - transactions etc.
2. Introduce your own Spring-AOP based aspects as needed
3. **Add enforcement and exploration aspects using AspectJ**
4. Add infrastructure aspects using AspectJ
5. Add domain aspects using AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Adopting AspectJ

- skill & experience
- value derived from AOP

exploration & enforcement

infrastructure aspects

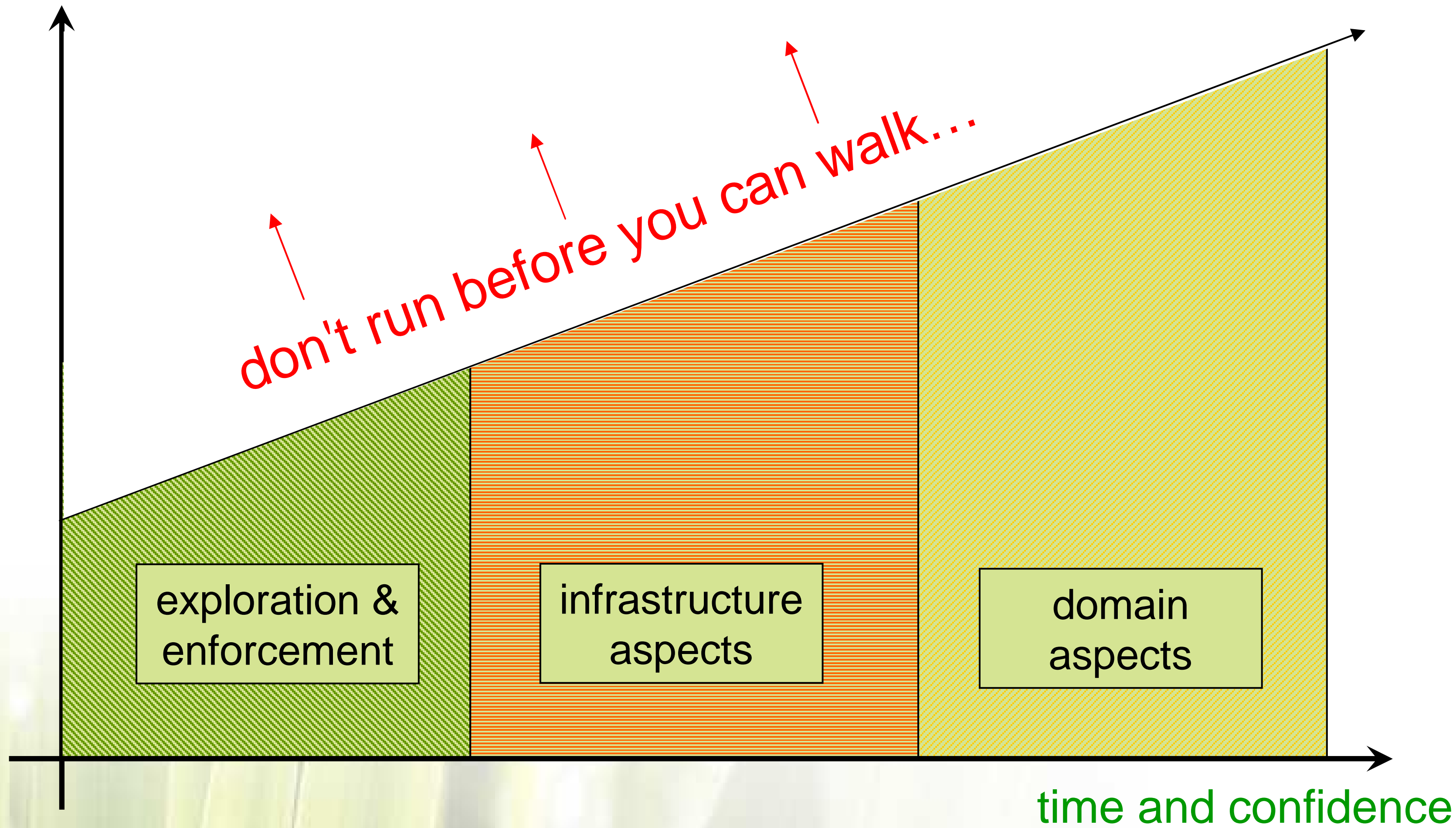
domain aspects

time and confidence

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Adopting AspectJ



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Exploration and Enforcement

- Exploration aspects
  - used to analyze / debug / troubleshoot application
  - developer aids used during the development process
    - not part of the production system
- Enforcement aspects
  - design-level assertions

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Exploration and Enforcement

- What these aspects have in common is…
  - no runtime dependency on AspectJ
  - no need for all developers to switch to using AJDT
- Very low risk way to get started with AspectJ
  - and very beneficial too!

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Permitted component interactions

```
/** … */
public aspect SystemArchitecture {
 …
 /*
  * no other module should depend on the
  * web tier
  */
 declare warning
   : callToWebTier() && !inWebTier()
   : "no external dependencies on web tier";

  …
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Permitted component interactions

```
/** … */
public aspect SystemArchitecture {
 …
 declare warning
   : callToDaoImpl() && !inDaoImpl()
   : "please use Dao interfaces instead";

 pointcut inPermittedDaoClient() :
   inServiceLayer() || inDaoImpl();

 declare warning
   : callToDao() && !inPermittedDaoClient()
   : "only service layer and Dao implementations
      themselves should be calling Daos";
  …
}
```

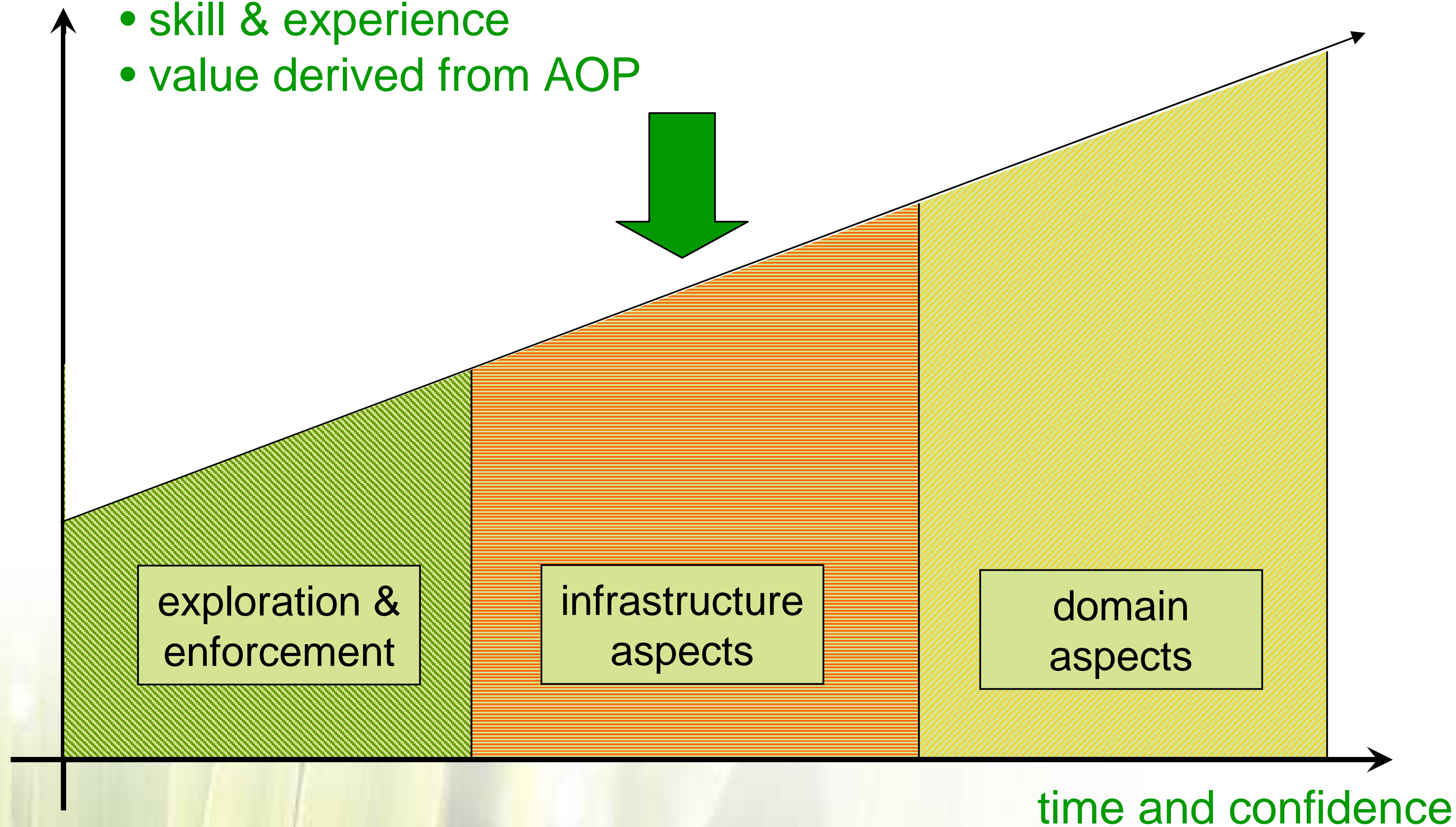
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Demo

- Hibernate usage guidelines…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Infrastructure aspects



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Infrastructure aspects

- At the next stage of adoption, introduce 'infrastructure' aspects
  - not a black-and-white definition
  - in general, an infrastructure aspect is not critical to the basic operation of your application
    - e.g. profiling, tracing, auditing, …
- Not all team members need to be familiar with writing aspects
  - using an aspect is a whole different thing to writing one

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Example: Profiling

- Profile all operations in an application
- Or just one use-case that is not performing as expected
- Spans multiple layers, fine-grained
  - -> AspectJ based solution
  - … with Spring configuration

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Profiling aspect

```
public aspect Profiler {
  private ProfilingStrategy profiler;

  public void
   setProfilingStrategy(ProfilingStrategy p) {
    this.profiler = p;
  }

  …
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Profiling aspect

```
  pointcut profiledOperation() :
    Pointcuts.anyPublicMethod() &&
    SystemArchitecture.inMyApplication();

  Object around() : profiledOperation() {
      Object token =
        this.profiler.start(thisJoinPointStaticPart);
      Object ret = proceed();
      this.profiler.stop(token,thisJoinPointStaticPart);
      return ret;
  }

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Profiling configuration

```
<bean id="profiler" class="Profiler"
  factory-method="aspectOf">
 <property name="profilingStrategy">
  <ref local="jamonProfilingStrategy"/>
 </property>
</bean>
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Use-case based profiling

- Simple to profile only for a given use-case of interest

```
pointcut profiledUseCase() :
  execution(* transfer(..)) &&
  SystemArchitecture.inServiceLayer();

pointcut profiledOperation() :
   Pointcuts.anyPublicMethod() &&
   SystemArchitecture.inMyApplication() &&
   cflow(profiledUseCase());
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

Example: Domain object security

- Security by ACL on individual domain objects
  - Can secure repository (Dao) operations using Spring AOP
  - to secure operations on domain objects themselves, you need to use AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Service layer security (recap)

```
@Aspect
public class ServiceLayerAccessControl extends
   AbstractAccessControlManager {

  @Pointcut("SystemArchitecture.businessService()")
  protected void securedOperation() {}

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Domain model security

```
@Aspect
public class DomainModelAccessControl extends
   AbstractAccessControlManager {

  @Pointcut("Pointcuts.anyPublicMethod() &&
             SystemArchitecture.inDomainModel()")
  public void securedOperation() {}

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring

INTERFACE21

Configuration

```
<!-- <aop:aspectj-autoproxy/> -->
<bean id="domainModelAccessControl"
  class="abc.DomainModelAccessControl" factory-method="aspectOf">
  <property name="authenticationManger"  ref="authenticationManager"/>
  <property name="accessDecisionManager"
            ref="domainModelAccessDecisionManager"/>
  <property name="afterInvocationDecisionManager"
            ref="resultAccessDecisionManager"/>
  <property name="securityAttributes"
            ref="domainModelSecurityAttributes"/>
</bean>
```

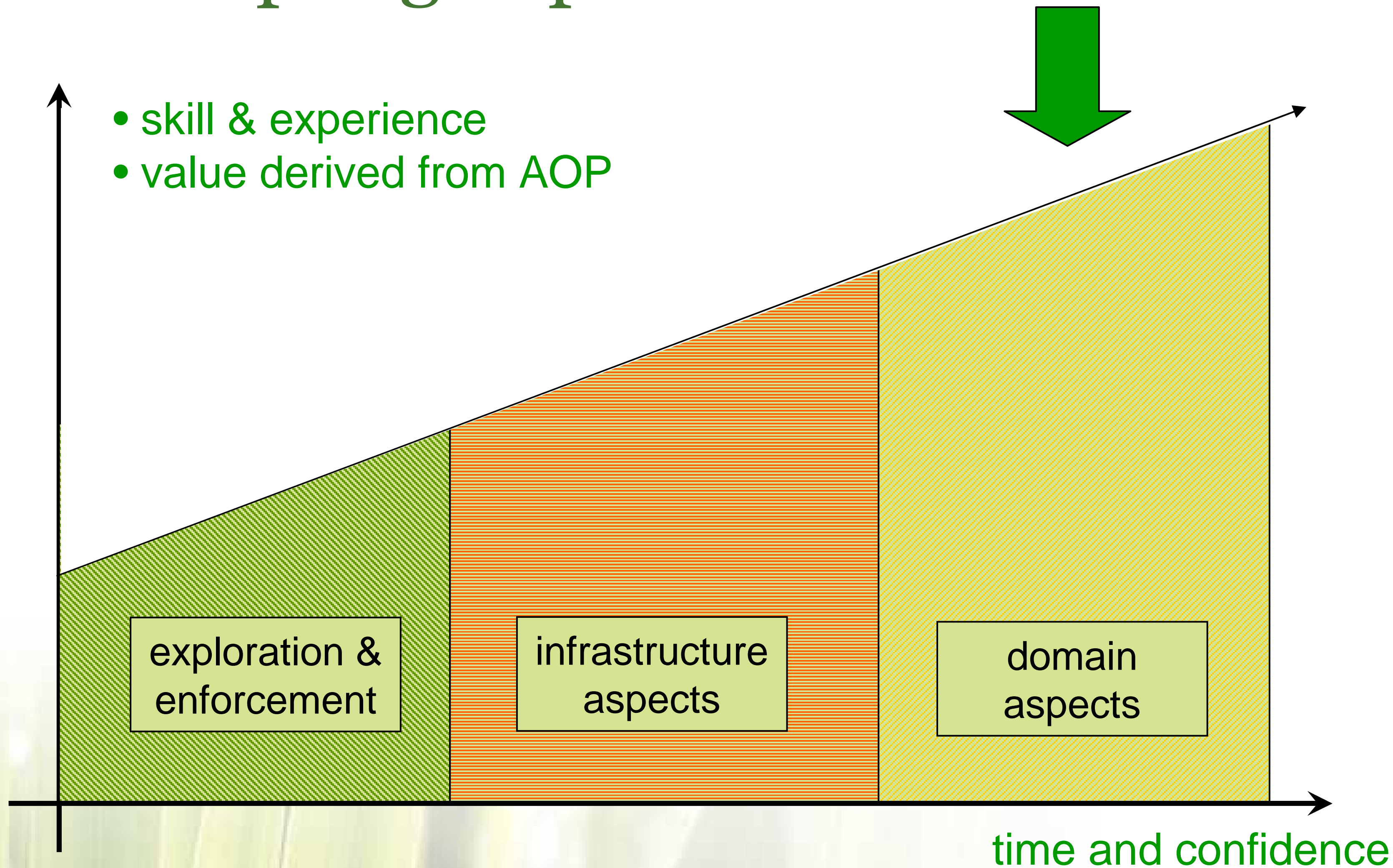
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Demo

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Adopting AspectJ



Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Domain aspects

- Domain aspects implement function critical to the basic functioning of your application
  - not really easy to build and run any part of the system without them
- Cross a threshold here…
  - whole team needs to be comfortable with seeing aspects in the code
  - with debugging and maintaining them
  - whole team really needs to be using aspect-friendly development environment
    - i.e. AJDT

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Examples from AspectJ

- The AspectJ compiler itself is built using AspectJ…
- We extend the JDT compiler
  - want to make our changes as modular as possible
  - separate from main body of JDT code so that it is easy to integrate new drops of the JDT code

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Detecting empty catch blocks

```
/**
 * We get asked for this capability so often,
 * I thought I would simply extend the compiler
 * to support it.
 *
 * @author Adrian
 * @since 1.5.1
 */
public aspect WarnOnSwallowedException {

  pointcut resolvingATryStatement(
   TryStatement tryStatement, BlockScope inScope)
      : execution(* TryStatement.resolve(..)) &&
        this(tryStatement) &&
        args(inScope,..);
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Detecting empty catch blocks

```
    after(TryStatement tryStatement, BlockScope inScope) returning
          : resolvingATryStatement(tryStatement,inScope) {
      if (tryStatement.catchBlocks != null) {
          for (int i = 0; i < tryStatement.catchBlocks.length; i++) {
            Block catchBlock = tryStatement.catchBlocks[i];
            if (catchBlock.isEmptyBlock() ||
              catchBlock.statements.length == 0) {
                   warnOnEmptyCatchBlock(catchBlock,inScope);
            }
          }
      }
    }

    private void warnOnEmptyCatchBlock(
      Block catchBlock, BlockScope inScope) {
        inScope.problemReporter()
             .swallowedException(catchBlock.sourceStart(),
                                 catchBlock.sourceEnd());
    }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Domain aspect examples

- Observing
- Publishing events
- Domain object DI
- Business and application rules
- …

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

What about EJB3 ?

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Example from EJB spec

```
@Stateless
@Interceptors({
com.acme.AccountAudit.class,
com.acme.Metrics.class,
com.acme.CustomSecurity.class
})
public class AccountManagementBean implements AccountManagement {
    public void createAccount(int accountNumber, AccountDetails
    details) { ... }
    public void deleteAccount(int accountNumber) { ... }
    public void activateAccount(int accountNumber) { ... }
    public void deactivateAccount(int accountNumber) { ... }
...
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Aspect: Metrics gathering

```
public class Metrics {
    @AroundInvoke
    public Object profile(InvocationContext inv) throws Exception {
        long time = System.currentTimeMillis();
        try {
            return inv.proceed();
        } finally {
            long endTime = time - System.currentTimeMillis();
            System.out.println(inv.getMethod() +
            " took " + endTime + "milliseconds.");
        }
    }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Aspect: Auditing

- `public void deleteAccount(AccountDetails details)`

```
public class AccountAudit {
   @AroundInvoke
   public Object auditAccountOperation(InvocationContext inv) throws
         Exception {
         try {
                Object result = inv.proceed();
                Auditor.audit(inv.getMethod().getName(),
                              inv.getParameters[0]);
                return result;
         } catch (Exception ex) {
                Auditor.auditFailure(ex);
                throw ex;
         }
   }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

True AOP approach

- No unneccessary annotations in service layer

```
@Stateless
@Interceptors({
com.acme.AccountAudit.class,
com.acme.Metrics.class,
com.acme.CustomSecurity.class
})
public class AccountManagementBean implements AccountManagement {
    public void createAccount(int accountNumber, AccountDetails
    details) { ... }
    public void deleteAccount(int accountNumber) { ... }
    public void activateAccount(int accountNumber) { ... }
    public void deactivateAccount(int accountNumber) { ... }
...
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Use an AOP solution that offers true pointcuts

Ability to reference and *reuse* named pointcuts, as well as define them inline

```
@Aspect
public class MetricsAspect {

    @Around("SystemArchitecture.serviceLayerOperation()")
    public void profileServiceLayerOperation(ProceedingJoinPoint jp) throws
    Throwable {
          long time = System.currentTimeMillis();
          try {
                    return jp.proceed();
          }  finally {
                    long endTime = time - System.currentTimeMillis();
                    System.out.println(jp.getSignature().getName() +
                              " took " + endTime + "milliseconds.");
          }
    }
}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Audit aspect

```
@Aspect
public class AuditAspect {
   @Around("SystemArchitecture.serviceLayerOperation() &&
   args(details, ..)")
   public void auditAccountOperation(ProceedingJoinPoint jp,
   AccountDetails details) throws Throwable {
         try {
                 Object result = jp.proceed();
                 Auditor.audit(inv.getMethod().getName(), details);
                 return result;
         }  catch (Exception ex) {
                 Auditor.auditFailure(ex);
                 throw ex;
         }

}
```

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Simple?

Annotations crosscut multiple classes and methods, or verbose XML in DD

```
@Interceptors(com.acme.AccountAudit.class)
public void createAccount(AccountDetails details)  {

@Interceptors(com.acme.AccountAudit.class)
public void deleteAccount(AccountDetails details)  {


public class AccountAudit {
      @AroundInvoke
      public Object auditAccountOperation(InvocationContext
      inv)
                                                  throws
      Exception {
          try {
      `       Object result = inv.proceed();
              if (inv.getParameters().length > 0 &&
               (inv.getParameters()[0] instanceof
      AccountDetails)){
                          Auditor.audit(
                            inv.getMethod().getName(),
                           (AccountDetails)
      inv.getParameters[0]);
              }
              return result;
          }  catch (Exception ex) {
                  Auditor.auditFailure(ex);
                  throw ex;
            }
      }
}
```

Necessary fix to prevent `ArrayIndexOutOfBoundsException`

- **One single, *simple* module**

```
@Aspect
public class AuditAspect {
      @Around("SystemArchitecture.serviceLayerOperation() &&
      args(details, ..)")
      public void auditAccountOperation(ProceedingJoinPoint jp,
                                    AccountDetails details)
                                          throws Throwable {
              try {
                   Object result = inv.proceed();
                   Auditor.audit(
                              inv.getMethod().getName(), details);
                   return result;
              }  catch (Exception ex) {
                   Auditor.auditFailure(ex);
                   throw ex;
              }
}
```

Type safe
No risk of invocation with incorrect arguments

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Verdict

- Missing the key abstraction that pointcuts give you
- Untyped – easy to make mistakes
- Unproven model

- Tried to make something that was simpler…
  - actually made something that was more complex to use, *and* less powerful

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Summary

- We want to implement enterprise requirements in as simple and straightforward a manner as possible
  - use the appropriate implementation "vocabulary"
- AOP is about modularity
  - getting closer to the ideal of the "1:1 principle"
- AspectJ and Spring AOP are the leading AOP implementations
  - tightly integrated
  - Can use together or independently
- Adoption can proceed in phases
- Simpler and more powerful than EJB3

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

About the course

- 4 days
- Mix of hands-on experience and classroom lectures
- Covers Spring AOP and AspectJ

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring

Day 1

- What is AOP?
  - goals and motivation, how AOP works
  - key terms and concepts introduced
- Spring AOP and AspectJ
  - how to write and build aspects in Spring AOP and AspectJ
- Basic Pointcuts
  - the pointcut language, writing robust pointcuts, style tips
- Design level assertions
  - enforcing layering and other design constraints
  - using the tools – working with AJDT, and integrating into your builds

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Day 2

- Advice types
  - understanding the different kinds of advice
  - exception management with aspects
- Pointcuts with parameters
  - example: clustered caching
- Using an aspect library
  - how to use an existing library, load-time weaving
- Infrastructure aspects
  - transactions, security, tracing, profiling, failure handling, jmx,…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Day 3

- Matching on Java 5 features
  - generics, annotations etc.
- Using annotations (wisely…)
  - asynchronous task execution, auditing
- Mixins and inter-type declarations
  - read-write locking, annotation bridging, dirty tracking, bean-name aware
- Aspects and design patterns

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Day 4

- Aspects
  - abstract aspect and instantiation models
  - generating application events
- Testing
  - tips and techniques for testing aspects
- Aspect libraries
  - writing and packaging reusable aspects
  - creating an aspect library for use by your project / organisation
- Adoption roadmap
  - how to introduce AOP to your project or organisation

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21

The End!

- questions…

Copyright 2004-2006, Interface21 Ltd. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring