Discovering AOP

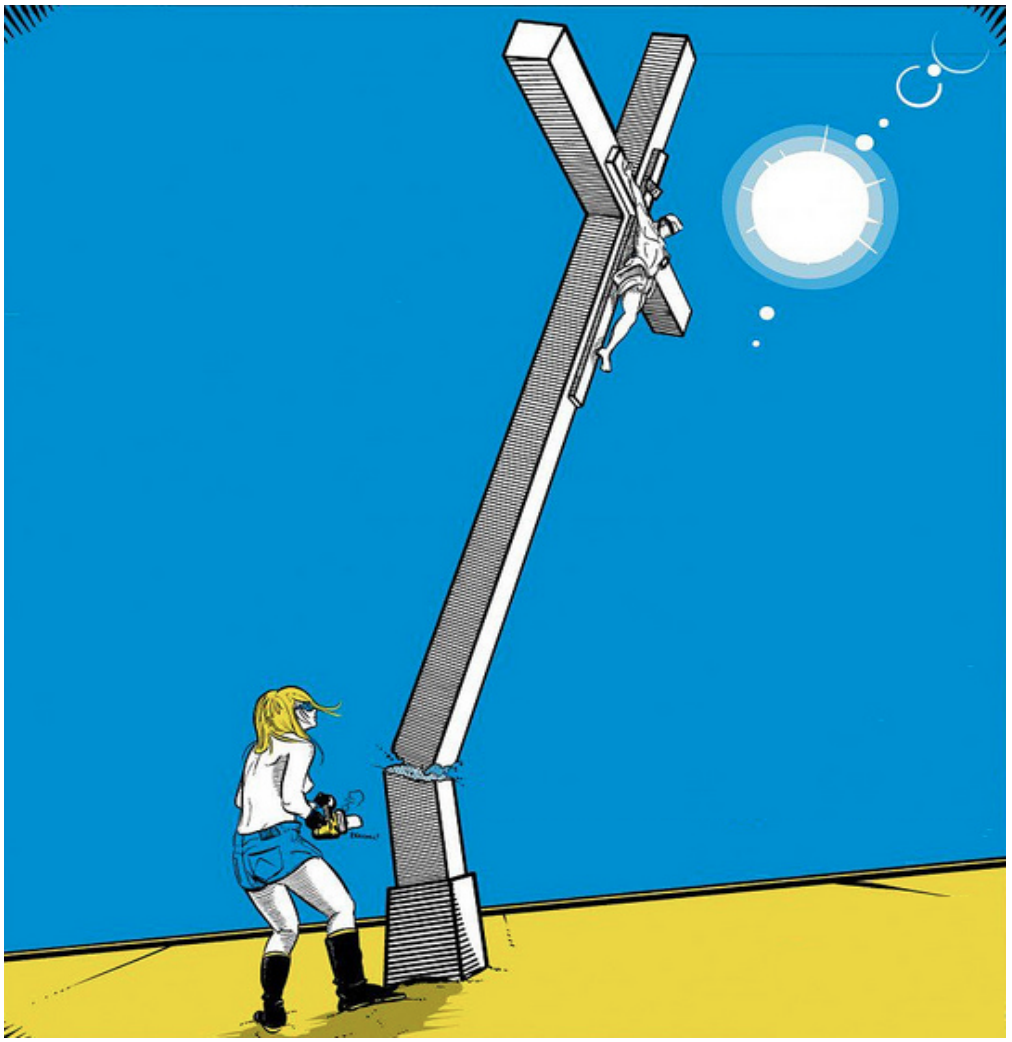
An introduction to AOP and AspectJ

Károly Szántó & Bogdan Solga

Introduction

- Addressing complexity through modularization
- **Core Concerns -** core functionality of the system (business logic, data access, presentation logic)
- **Cross-Cutting Concerns (C3)?**

Introduction



Introduction

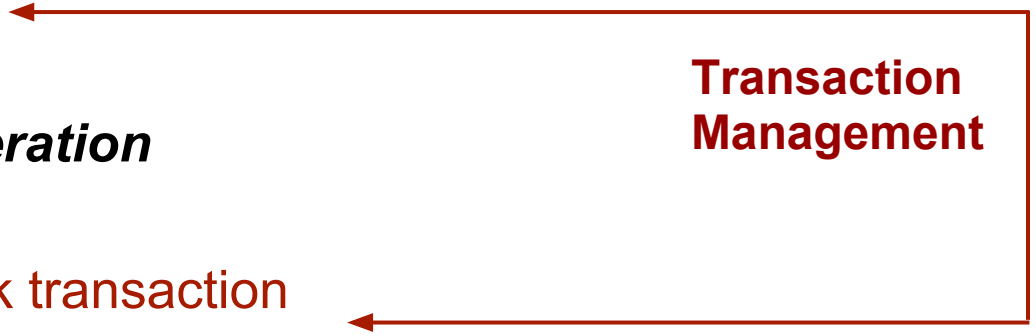
- C3 = Concerns that cut across many modules
  - e.g. security, logging, transaction management, caching
- OOP
  - support to model core concerns (classes)
- AOP
  - new programming methodology
  - support to model c3 (aspect)

Use Case - example

```
public SomeClass {

    public void operation(<args>) {

        ... Perform core operation

    }
}
```

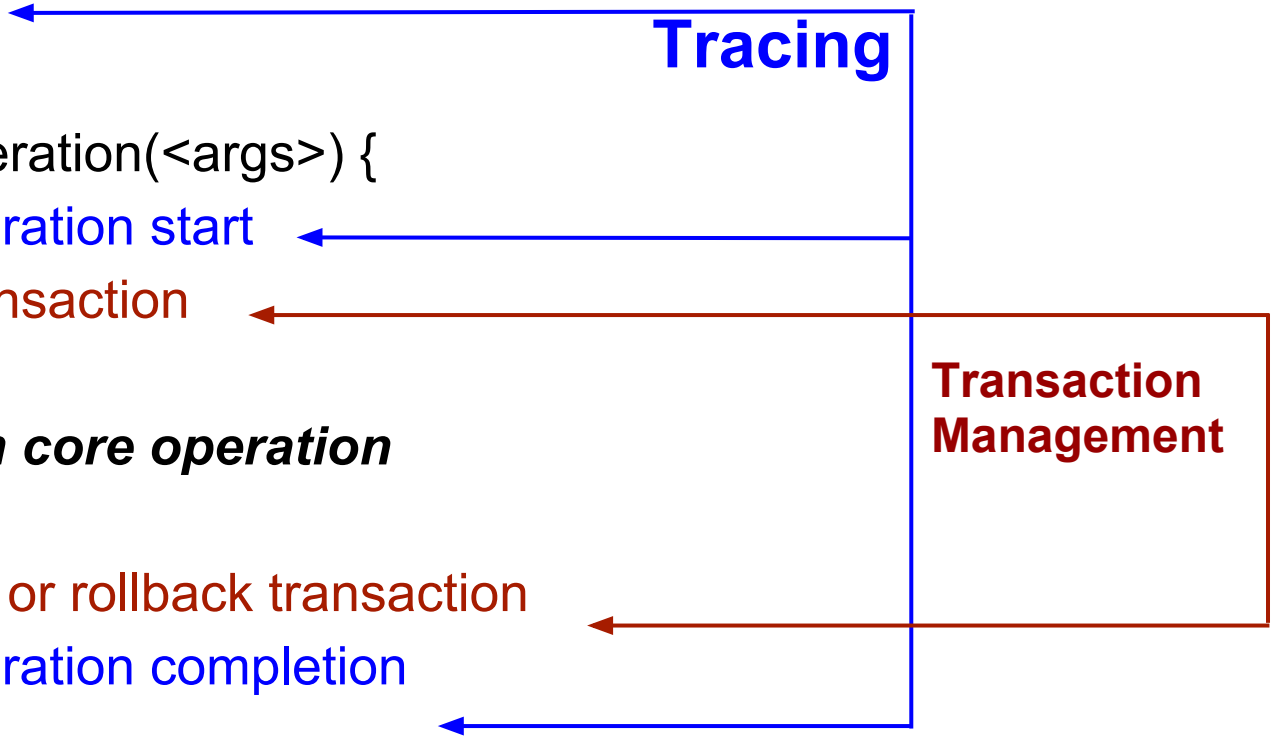
Use Case - example

```
public SomeClass {

    public void operation(<args>) {

        ... Start transaction

        ... Perform core operation

        ... Commit or rollback transaction

    }
}
```



Use Case - example

```
public SomeClass {
    ... Log stream

    public void operation(<args>) {
        ... Log operation start
        ... Start transaction

        ... Perform core operation

        ... Commit or rollback transaction
        ... Log operation completion
    }
}
```



Terminology

- Aspect
  - AOP's unit of modularity
- Join Point
  - an identifiable execution point in a system (e.g. a call to a method, a field access)
- Pointcut
  - mechanism for selecting a join point
- Advice
  - adds behavior before, after or around the selected joint points
- Weaving
  - composing the final system from core modules and aspects

AspectJ

- aspect oriented extension for the Java programming language
- well known AOP implementation

- traditional AspectJ syntax(a set of new keywords) & @ApectJ syntax

- Build-time weaving (AJC) -> Compile-time
- Load-time weaving (JVMTI agent) -> Run-time

Pointcuts in AspectJ

- pointcut_type(signature)
- signature = [access_modifier] [method_return_type] [@annotation_type] ***type*** [.***field*** | ***method(params)***]

- e.g.:
  - execution(* **UserRepository+.*(..)**)
  - call(**public int Calculator.add(int,int)**)
  - set(**private String User.name**)
  - within(**@Transactional* UserRepository+**)

Demo + Q&A

Github - **tmjug-aop-demo**

Spring AOP

An introduction to the Spring AOP implementations

Spring AOP - intro

- implemented in pure Java - no need for a special compiler;
- proxy based implementation (classes / interfaces);
- supported frameworks:
  - AspectJ (limited support for it) - since Spring 2.0;
  - Spring AOP (Spring 1.2);
- currently supports only method join-points;

AspectJ support

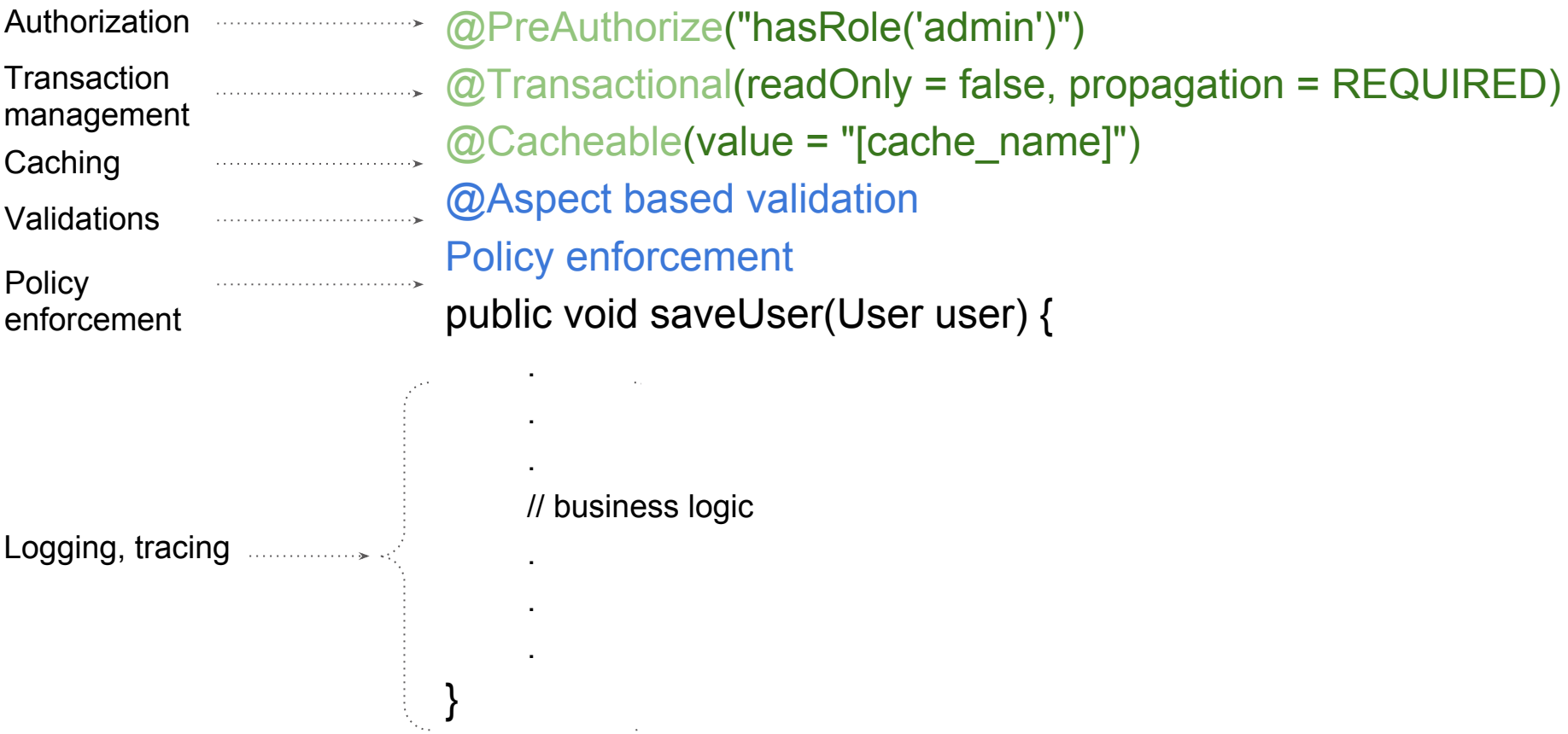
- enabled through the **<aop:aspectj-autoproxy/>** tag;
- autoproxying - Spring generates a proxy for the advised beans;
- aspects:
  - declarative - schema-based approach (aop:* tags in XML);
  - programmatic - @AspectJ annotation style;
- @Aspect annotated classes:
  - classpath scanning (@Component is required);
  - declared as beans;
- supported pointcut designators: *execution, within, this, target, args, @target, @args, @within*

Use cases - overview

Common use cases:

- policy and security enforcement - restrict access to modules;
- authorization [@PreAuthorize, @PostAuthorize, @Pre/PostFilter];
- transactionality [@Transactional];
- caching [@Cacheable, @CacheEvict];
- asynchronous processing [@Async];
- custom validations;
- custom logging / tracing;

Use case - example



Declarative vs programmatic

- declarative:

```
<aop:config>
  <aop:aspect id="aspectId" ref="bean">
    ...
  </aop:aspect>

  <aop:pointcut id="businessService" expression="execution(* com.service.*.*(..))"/>
</aop:config>
```

- programmatic:

```
@Pointcut("execution(* save(..))")
public void onSave() {}

@Before("onSave() && args(userTO)")
public void validateUser(UserTO userTO) throws ValidationException;
```

Usage examples

```
@Pointcut("execution(* get*(..))")
public void onExecution() {}
```

-> matches for the execution of methods which start with 'get'

```
@Pointcut("within(org.tmjug.aop.demo..*)")
public void fromDemoPackage() {}
```

-> matches for the execution of methods from the 'org.tmjug.aop.demo' package

```
@Before("onExecution() && args(name)")
public void traceAccess(String name) {
        //...
}
```

Authorization

- enabled through Spring Security's **<global-method-security pre-post-annotations="enabled">** configuration tag;
- authorize access to classes / methods, through annotations;
- support's Spring Expression Language (SpEL) for configuration;
- used annotations:
  - @PreAuthorize, @PostAuthorize;
  - @PreFilter @PostFilter;

Examples:

```
@PreAuthorize("hasRole('admin') && isAuthenticated()")
public void deleteUser(String userName);

@PreAuthorize("isAuthenticated()")
public void getUser(String userName);
```

Transaction management

- enabled by the **<tx:annotation-driven>** tag;

- supported annotations:
  - Spring's @Transactional;
  - EJB's @TransactionAttribute (if used);

- @Transactional configuration options:
  - **Isolation**;
  - **Propagation**;
  - readOnly;
  - rollbackFor;
  - noRollbackFor;

@Transactional usage examples

```
@Transactional(readOnly = false, propagation = REQUIRED)
public void saveUser(User user);

@Transactional(readOnly = true, propagation = PROPAGATION_SUPPORTS)
public void loadUser(String userName);
```

Cache management

- enabled by the **<cache:annotation-driven>** tag;

- currently defined managers:
  - SimpleCacheManager - basic cache manager from Spring;
  - EhCacheCacheManager - cache manager from EhCache;

Examples:

- adding to the cache:

```
@Cacheable(value = "users")
public void saveUser(User user);
```

- removing from the cache:

```
@CacheEvict(value = "users")
public void deleteUser(String userName);
```

Spring AOP

- **pointcuts** - implementations of the org.springframework.aop.Pointcut interface;

- **advices**:
  - interception around advice - org.aopalliance.intercept.MethodInterceptor;
  - org.springframework.aop.BeforeAdvice;
  - org.springframework.aop.ThrowsAdvice;
  - org.springframework.aop.AfterReturningAdvice;

- **usage** - through AOP proxies, created by the org.springframework.aop.framework.ProxyFactoryBean class;

Demo + Q&A

Source code & PDF

Github - **tmjug-aop-demo**