Intro to Spring AOP

For Spring Framework 4.3.x

Aspect-Oriented Programming

- Aspect-Oriented Programming is a programming paradigm like OOP is a programming paradigm.
- Spring implements its own AOP support using Spring-AOP, but it also supports AOP using AspectJ, a popular Java extension that provides AOP.
- We'll be using Spring-AOP -- it's built in and a little simpler.

Aspect-Oriented Programming

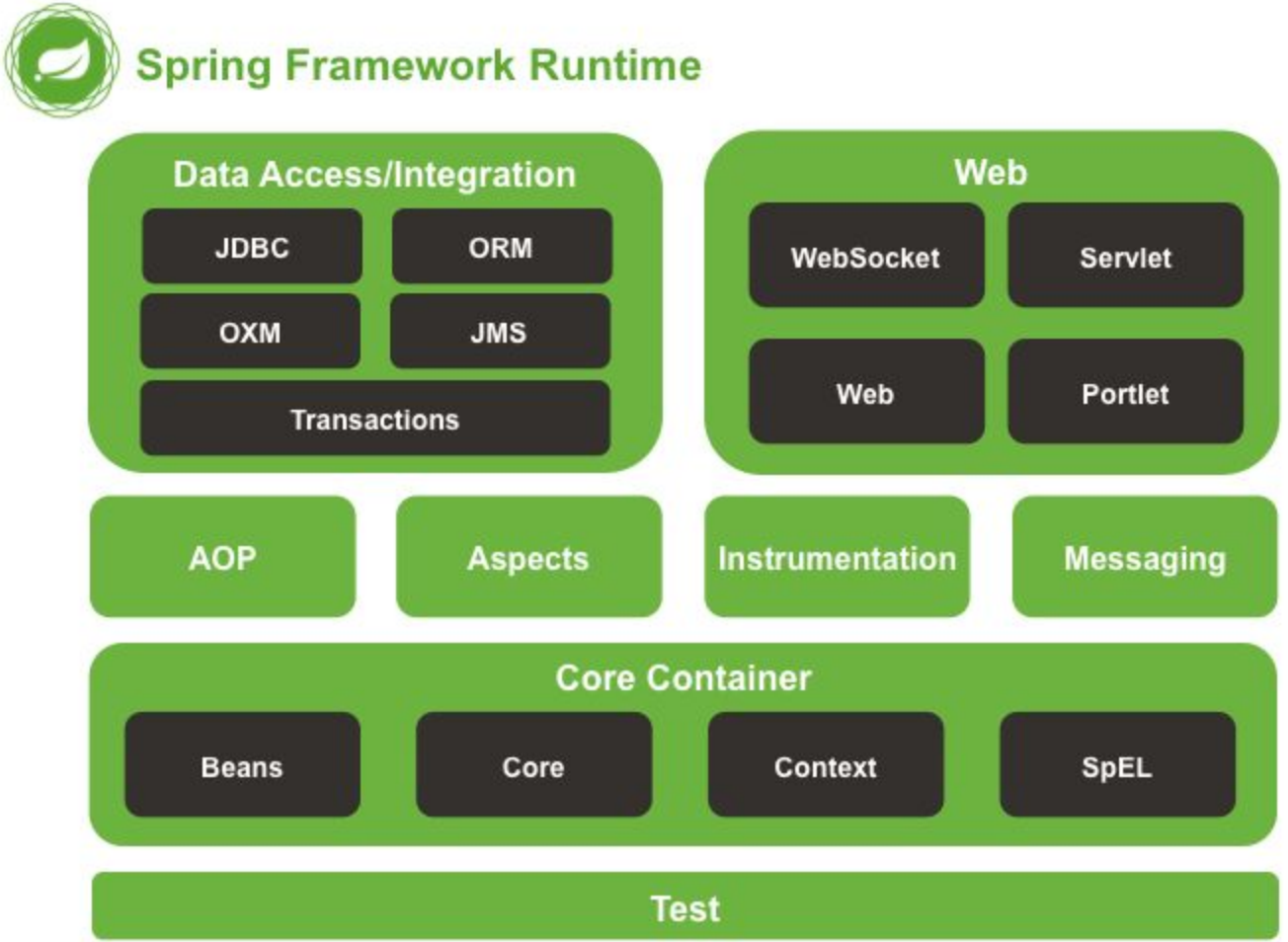
- AOP complements OOP by providing another way of thinking about program structure.
- Aspects handle “crosscutting concerns”: concerns that span multiple types and objects.
- AOP allows you to deal with these crosscutting concerns modularly

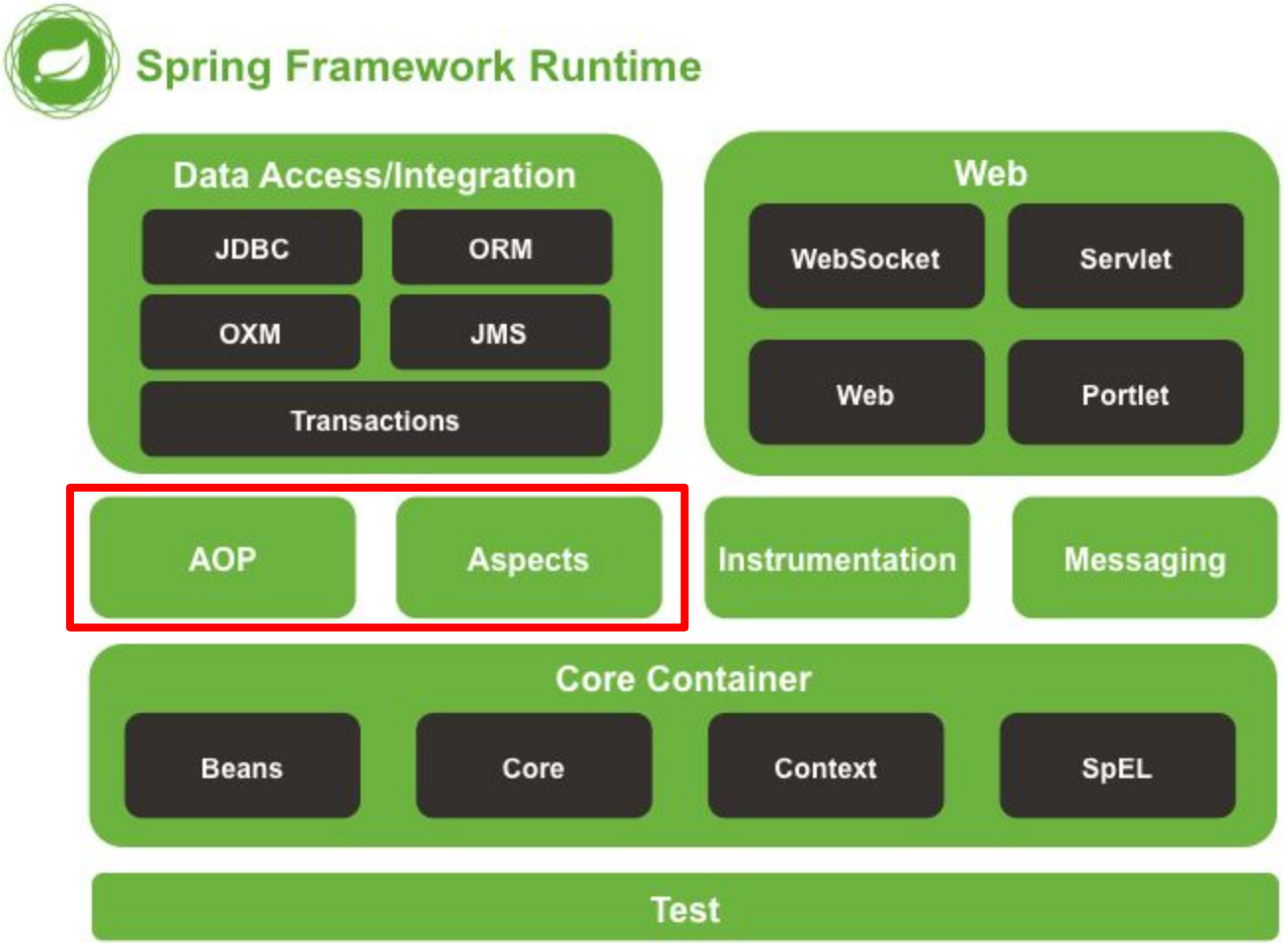
Aspect-Oriented Programming

- We have some terminology to learn, but I like to start with the motivation of AOP, which we'll check out in a Wikipedia example:

```
void transfer(Account fromAcc, Account toAcc, int amount) throws Exception {
  if (fromAcc.getBalance() < amount)
      throw new InsufficientFundsException();

  fromAcc.withdraw(amount);
  toAcc.deposit(amount);
}
```

```
void transfer(Account fromAcc, Account toAcc, int amount, User user,
    Logger logger, Database database) throws Exception {
  logger.info("Transferring money...");

  if (!isUserAuthorised(user, fromAcc)) {
    logger.info("User has no permission.");
    throw new UnauthorisedUserException();
  }

  if (fromAcc.getBalance() < amount) {
    logger.info("Insufficient funds.");
    throw new InsufficientFundsException();
  }

  fromAcc.withdraw(amount);
  toAcc.deposit(amount);

  database.commitChanges();  // Atomic operation.

  logger.info("Transaction successful.");
}
```


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

AOP in Spring

- Spring-AOP is the module we'll be using
- Aspects is a module that includes support for AspectJ, another way of doing AOP in Java
- Spring uses AOP under-the-hood for transaction management, security, and remote access

Aspects

- The modularization of a feature or concern that touches many parts of a program, but isn’t related to the primary function of the program.

- The combination of Advice and a Pointcut

- Best understood through examples:
  - Logging
  - Database Transactions
  - Security
  - File compression and decompression

Join Point

- A point during your program when a method is called or an exception is handled. These points are where it is possible to define Aspects that apply their Advice.

- In Spring AOP, we can only define Join Points for method execution, not exceptions

Advice

- Advice is the action taken / method ran by an Aspect when an associated Join Point runs.
- As an example, if we're using AOP to do our logging, the Advice is the method that actually writes to the logs.
- There are five types of Advice in Spring-AOP:
  - Before : executes before join point; can't interrupt flow unless it throws an exception
  - After Returning : executes after the join point returns successfully
  - After Throwing : executes after the join point throws an exception
  - After (finally) : executes after the join point runs, regardless
  - Around : surrounds a join point, allowing you to execute methods before or after the join point, and allowing you to interrupt flow to skip or modify the execution of the join point

Pointcut

- A pointcut is a set of join points in your program. We define pointcuts using the Pointcut Expression Language. Pointcuts are associated with Advice that runs each time any of the join points are run.

- As an example, if we’re using AOP to do our logging, the Pointcut defines where in our code we want to write to the logs.

Pointcut Expression Language

- A way of programmatically defining pointcuts used by default in Spring-AOP
- Some templates and examples:
- execution(ret-type-pattern name-pattern(param-pattern)
  - execution(* *(..)) Any method execution
  - execution(String *(..)) Any method execution that returns a string
  - execution(* set*(..)) Method that begins with “set” execution
  - within(com.revature.*) Any join point in com.revature package
  - within(com.revature..* Any join point in com.revature package or sub-packages
  - bean(userService) Any join point on a bean named “userService”
  - bean(*Service) Any join point on beans whose names end with “Service”
- You can add modifiers like:
  - execution(public * *(..)) Any public method execution

Weaving (and Proxies!)

- Weaving is when we actually produce the “advised object” -- when the business logic code from your objects is combined with the advice from your aspects.
- In Spring AOP Weaving occurs at runtime via AOP-Proxies:
  - An AOP-Proxy Object is created that includes both the public advised methods of your advised object and the appropriate advice methods.
  - Method calls on a proxied object are first invoked on the proxy, then on the object itself
  - Advice is only applied to method calls on the proxy, so any self-invocation inside your objects WILL NOT apply advice.

Terminology Overview

- Aspect: a modularization of a concern that cuts across multiple classes.
- Join point: a point during the execution of a program, such as the execution of a method or the handling of an exception.
- Advice: action taken by an aspect at a particular join point.
- Pointcut: a predicate that matches join points.
- Pointcut Expression Language: A way to describe pointcuts programmatically
- Weaving: linking aspects with other application types or objects to create an advised object. Occurs in Spring AOP using proxies.

Spring AOP v. AspectJ

- Spring AOP doesn’t replace all the functionality of AspectJ
- Spring AOP was built to be less complex in design, configuration, and implementation
- Some differences:
  - Spring AOP uses proxies for its implementation
  - Spring AOP implements advice at runtime, using reflection (runtime weaving)
  - Spring AOP only has join points on method execution
  - Spring AOP makes use of the IoC container

AspectJ

- AspectJ is not what we use!
- Can be integrated into Spring using the spring-aspects module
- AspectJ is more powerful than Spring-AOP, and is a separate extension of Java
- Has its own separate compiler
- Can do both compile-time and load-time weaving
- Widely used, and is the standard for enterprise AOP
- More complicated to configure and use than Spring-AOP
- The source of @AspectJ style, so it uses the same annotations we use in Spring