selsoft

build better, deliver faster

Enterprise Application Development with Spring

Chapter 9: Spring Aspect-Oriented Programming

Instructor

Akın Kaldıroğlu

Expert for Agile Software Development and Java

Topics

- **Aspect-Oriented Programming**
  - Crosscutting Concerns
  - Definitions
  - Advice
- **Spring AOP**
  - Main Objects
- **AspectJ**
- **Spring's AOP Schema**
- **AspectJ Support**

Aspect-Oriented Programming

selsoft

build better, deliver faster

Aspect-Oriented Programming - I

- **Aspect-Oriented Programming** (**AOP**) tackles complexity in software.
- It is a technique that helps to achieve modularity.
- AOP is an enhancement on object-oriented programming (OOP) and design.
- It is a complementary technique in object-oriented paradigm that emphasizes separation of concerns and the principle of single responsibility (SRP) which lead to highly-cohesive objects.

Aspect-Oriented Programming - II

- AOP allows us to be able to divide a bigger problem into smaller pieces by separating different concerns of the problem.
- It is used to separate core business functionality from peripheral services called crosscutting concerns.
- Aspects are crosscutting concerns.

Crosscutting Concerns

Main and Core Concerns

- Main concern of a software system is its business functionality.
- There are other concerns such as managing data, presenting it to users, integrating with other systems, etc.
  - These are all core concerns that are mandatory to have to implement the main concern.
- Main and core concerns are implemented in different modules.

Crosscutting Concerns - I

- Crosscutting concern is a concern that exists in almost all modules:
  - Logging
  - Auditing
  - Security
  - Concurrency management
  - Resource management
  - Caching
  - Transaction management
  - Performance profiling
  - Event management
  - etc.

Crosscutting Concerns - II

- Crosscutting concerns cut accross many modules.
- Object-oriented programming (OOP) is good at managing modularity for main and core concerns.
  - Modules for different business functionalities and core concerns such as presentation, integration, persistence are all created using OOP.
- But it is not that good at managing crosscutting concerns.
- Using OOP does not lead to having modules for crosscutting concerns.

Crosscutting Concerns - III

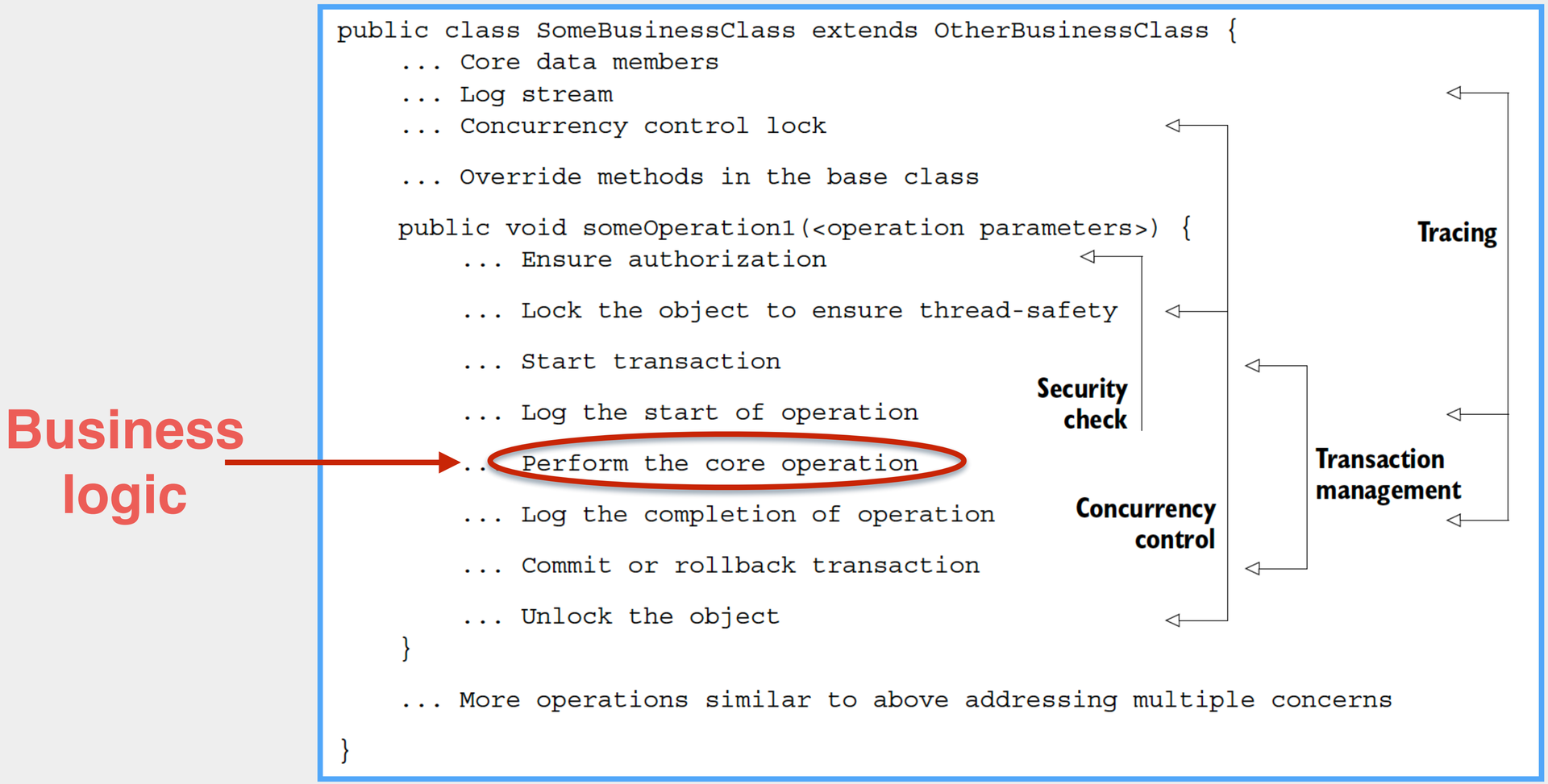
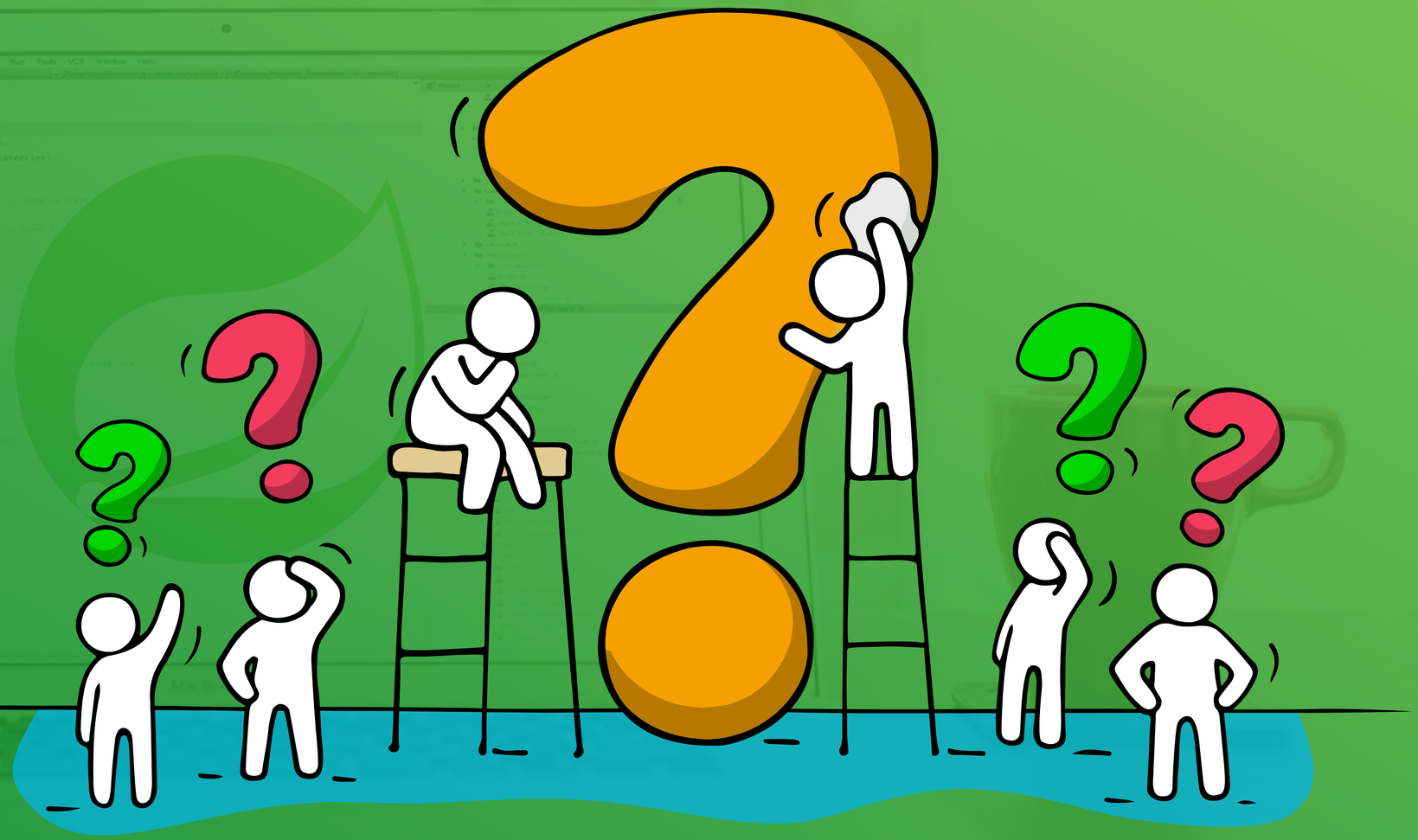


Image taken from the book **AspectJ in Action 2nd Ed.**

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Definitions

Join Point

- **Join point** is a point in the execution of the program.
- It is mostly a method execution.
- Throwing an exception or initializing a field might be a join point too.
- In terms of **Spring** AOP a join point is only method executions on beans.
- As a more powerful AOP framework, AspectJ, supports join points for field access, constructor, etc.

Advice - I

- **Advice** is a specific code executed at a certain join point.
- It is an action taken at a joint point or collectively at a set of join points, which is called pointcut.
- It is the answer to the question *what happens when a certain method is called on a bean?*

Advice - II

- There are different kinds of advice such as **around**, **before** and **after** advice.
  - That means an advice can be applied just before, after and around i.e. before and after a method execution on a bean.
  - For example logging or checking authorization before executing certain methods are applying before advice to those methods.

Advice - III

- Most AOP frameworks implements advices using interceptors.
- Interceptor is a piece of code that interrupts the normal flow of execution to run the code of the advice.
  - Interceptor lets the normal flow to continue after the execution of the advice.
- There can be more than one advice for a join point so interceptors keep track of registered advices in a chain at that join point.

Pointcut - I

- **Pointcut** is a collection of join points on which an advice is to be executed.
  - It is a predicate that matches join points.
- For example all of the methods of the beans (managed by IoC container of course) on which loging as before advice is executed constitute a pointcut.
- Pointcuts represent a certain action i.e. advice to be executed in a set of method executions in a system.

Pointcut - II

- Advice is associated with a pointcut using an expression.
  - It means that all of the method executions expressed in the pointcut are applied associated advice.
- And if there is a match that advice is run on all join points.
  - If a log is to be made before **`@Bean`** methods in **`@Cofiguration`** classes all **`@Bean`** methods together create a pointcut.

Aspect - I

- **Aspect** is a combination of advice and pointcut.
- An aspect is a combination of a certain action i.e. advice and its pointcut or set of join points in a system.
  - Logging before `@Bean` methods in `@Cofiguration` classes is an aspect.
- Aspects help to modularize crosscutting concerns.
- In AOP it is possible to create a module that includes different before advices for logging needs and declare which methods (a pointcut) need to call those advices to implement its logging functionality.

Aspect - II

- Spring AOP implements aspects either by using POJOs in the schema-based approach or classes annotated with the `@Aspect` annotation.

Introduction

- **Introduction** is modifying the structure of an object by introducing additional methods or fields to it.
- Introduction is used to make an object implement a specific interface without needing the object's class to implement that interface explicitly.
- In **Spring** AOP new methods can be introduced through interfaces for the target types.
- For example, beans can implement `IsModified` interface to simplify caching.

Target

- **Target** is an object whose execution flow is modified by AOP.
- Target is also called advised object.
- A target may be advised by one or more advices.
- **Spring** AOP creates runtime proxies of the objects as targets.

AOP Proxy

- **Proxy** is an object created by the AOP framework in order to implement advise methods.
- Proxy is an object that has the same API of the target object and is decorated with the advices applied to the target.

Weaving

- **Weaving** is linking aspects with other application types or objects to create an advised object.
- If AOP uses proxies then it creates proxy for each target object and weaves or attach a piece of advice to that proxy.
- Weaving can be done at compile time or at runtime.
- **Spring** AOP uses JDK dynamic proxy or a CGLIB proxy to create an AOP proxy at runtime.

Advice

Advice - I

- **Spring** has following kinds of advice:
  - **Before advice**: It runs before a join point
    - It can’t prevent execution flow proceeding to the join point unless it throws an exception.
  - **After returning advice**: It runs after a join point and completes normally.
    - If a method throws an exception this advice does not run.
  - **After throwing advice**: It runs if a method throws an exception.

Advice - II

- **After (finally) advice**: It runs regardless of how a join point exits whether by normal or exceptional return.
- **Around advice**: It surrounds a join point.
  - It runs before and after a join point.
  - It starts before the join point and let the method at the join point executes and then completes after returning the method.
  - It is like a combination of before and after returning aspects.

Advice - III

- Around advice is the most powerful kind of advice.
- It can perform custom behavior before and after the method invocation.
- It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.

Croscutting Concerns and Advice - I

- Here are some examples:
  - Logging can be implemented as after advice meaning that logging is always done after the execution of certain methods (point cut).
  - Security check can be implemented as before advice meaning that it always checks if the current user has the priviledge to run a method before the execution of that method.
    - Advice throws an exception if the user does not have the priviledge or it lets the flow continue to the method if the advice successfully authorize the user.

Croscutting Concerns and Advice - II

- Another logging can be implemented as after throw advice so that when an exception is thrown it logs the state of the program.

Spring and AOP

- Any project that uses **Spring** uses AOP.
  - For example **Spring** manages transaction and security using aspects.
- It is nice to start AOP by using Spring AOP.
  - But for this especially bean management and DI of **Spring** should be learned before.

Example

```
@Service
public class UserServiceForDB implements UserService{

   @Log
   @RolesAllowed({"admin"})
   @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
   public boolean create(User user){ … }

   @Log
   @PermitAll
   public User fetch(int userId){ … }

   @Log
   @RolesAllowed({"admin"})
   @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
   public boolean delete(User user){ … }
}
```

Exercise

Exercise

- Which advice is suitable for following use cases?
  - An advice is used to bechmark time for certain code which is critical in terms of performance.
  - An aspect can be used for software license management,
  - An aspect can check if the user has the right to access a web source in a web application.
  - Auditing every single action of the user in a very critical system.

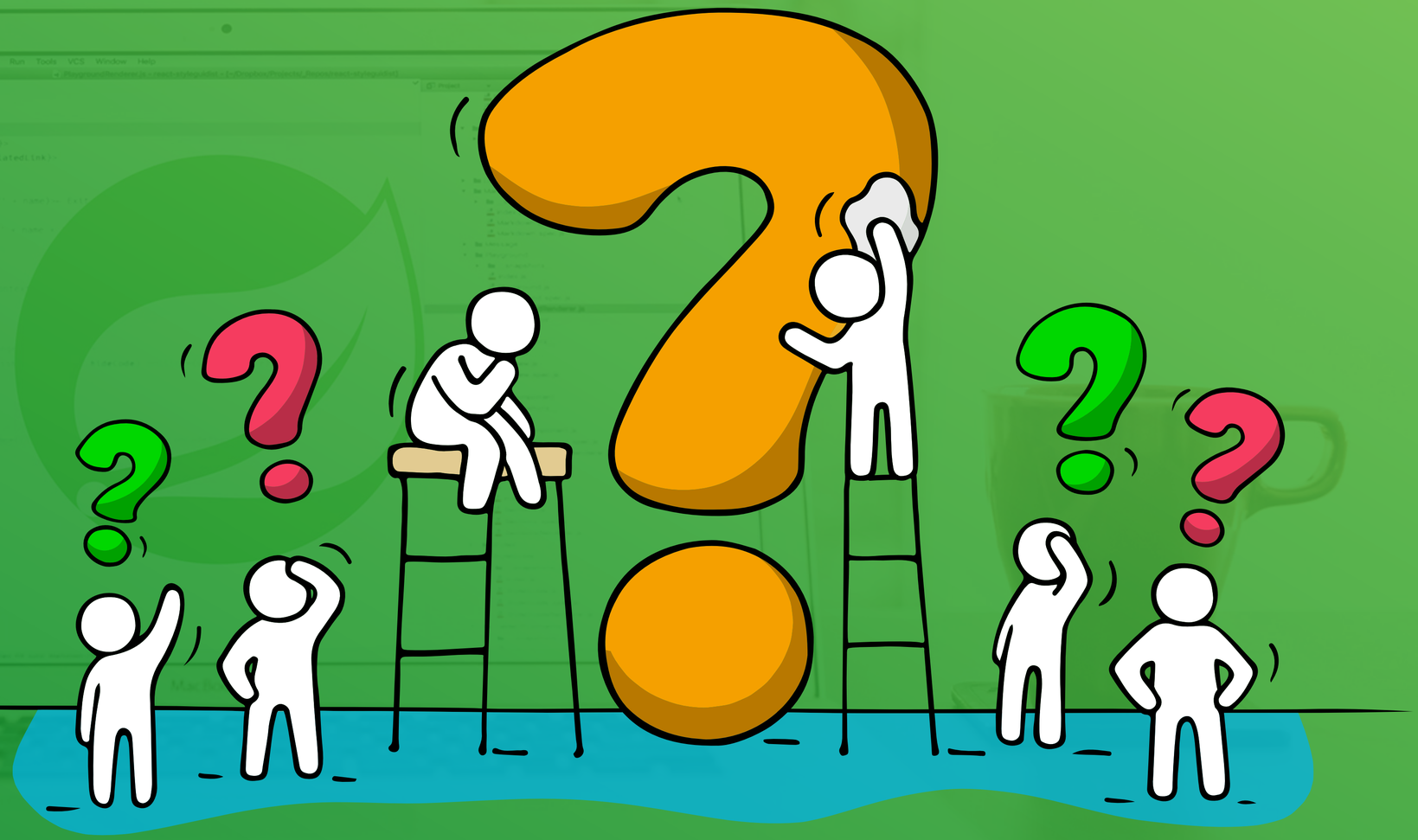
Exercise

- An advice that locks a resource before it is acquired in a method and then releases it after the methods returns.

GreetingExample

- `org.javaturk.spring.aop.ch01.greeting.basic.GreetingExample`

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Main Objects

selsoft
build better, deliver faster

AOP Alliance - I

- AOP Alliance (http://aopalliance.sourceforge.net/) is a group of people formed to provide an AOP standard for Java and Java EE.
- **Spring** AOP adapts its approach to AOP.
- Thus two packages of AOP Alliance's code are part of Spring API:
  - `org.aopalliance.aop` and `org.aopalliance.aop.intercept`
- These two packages defines main interfaces for an AOP system.
- **Spring** includes them as part of its API builds its AOP on it.

AOP Alliance - II

- **`org.aopalliance.aop`** includes only one interface, **`Advice`** which is the main interface that represents all advices and an exception.
- **`org.aopalliance.aop.intercept`** defines nine interfaces including **`Interceptor`** which extends **`Advice`** and represents interceptors.
  - **`MethodInterceptor`** is main interface to implement around advice.
- But **Spring** includes only seven interfaces of this package.

Spring AOP API - I

- Main package of Spring AOP is **`org.springframework.aop`** which has core interfaces, built on interfaces of AOP Alliance.
  - **`BeforeAdvice`**: Marker interface for before advice.
    - Main interface to implement before advice is **`MethodBeforeAdvice`**.
  - **`AfterAdvice`**: Marker interface for after advice.
    - It is parent of **`AfterReturningAdvice`** and **`ThrowsAdvice`**.

Spring AOP API - II

- **`AfterReturningAdvice`**: Interface invoked on normal method return.
  - It is main interface to implement after advice.
- **`ThrowsAdvice`**: Interface invoked when an exception is thrown.
  - It does not have any method.
  - **Spring** AOP finds by reflection the method to be called upon throwing an exception.

Spring AOP API - III

- Implementing these interfaces for different classes of advice and configuring them as beans will be enough to create an AOP application using **Spring**.

SqrtAOPExample

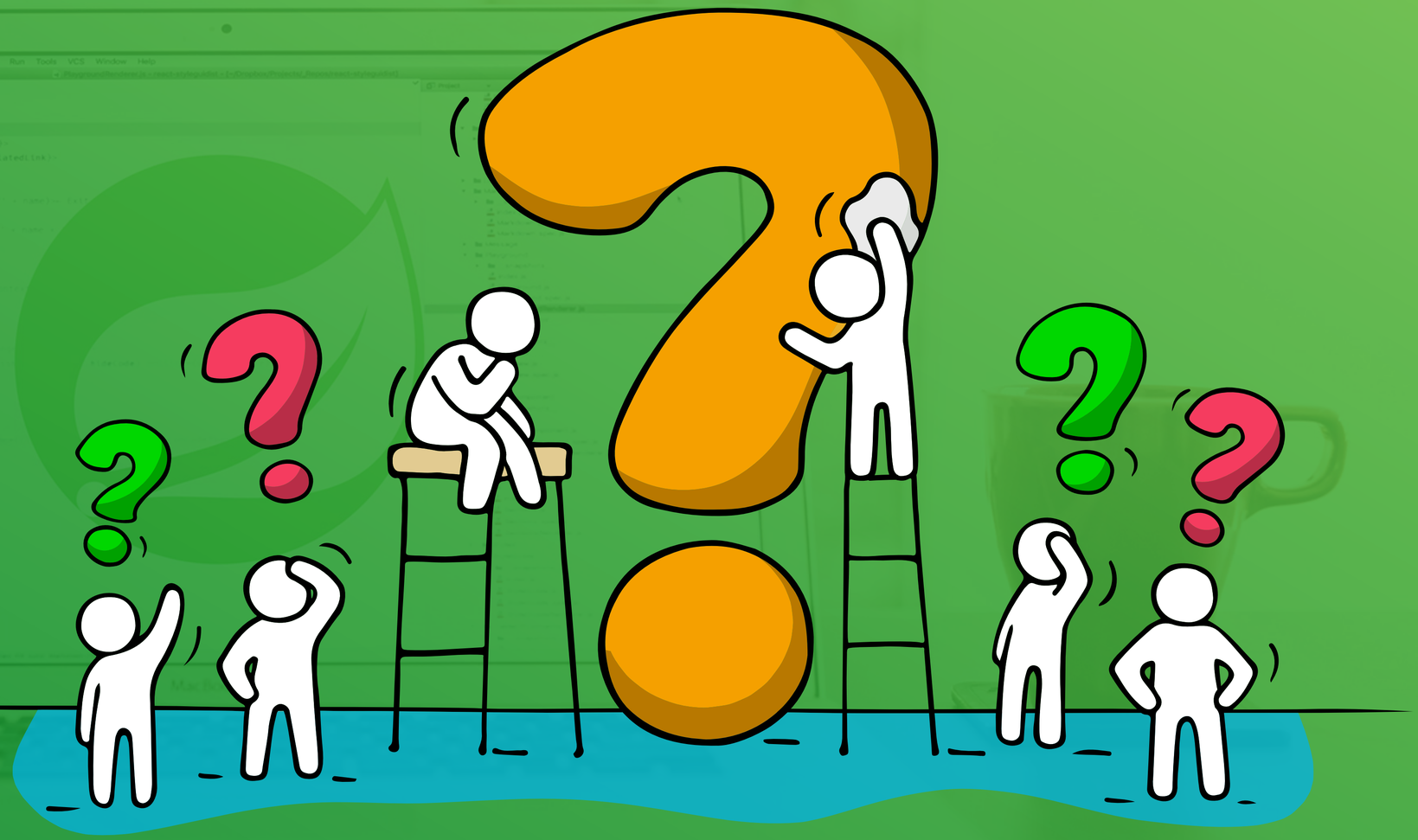
- `org.javaturk.spring.aop.ch01.sqrt.xml.SqrtAOPExample`

Exercise

Exercise

- Create a new version of `org.javaturk.spring.aop.ch01.sqrt.xml.SqrtAOPExample` using annotations and Java configuration.

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

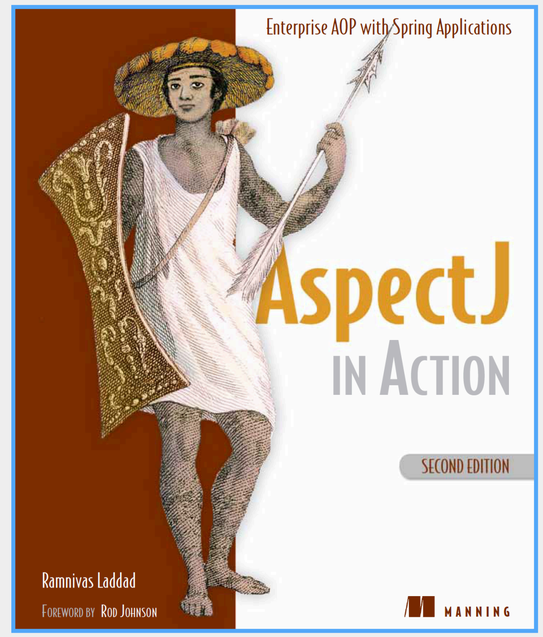
AspectJ

selsoft

build better, deliver faster

AspectJ - I

- **AspectJ** is a full-fledged, open-source AOP framework for Java.
- It is a project of Eclipse Foundation
  - https://www.eclipse.org/aspectj/
- **AspectJ** is a more powerfull framework than **Spring**'s AOP.
- **AspectJ in Action** is a good book both on **AspectJ** and **Spring** AOP.



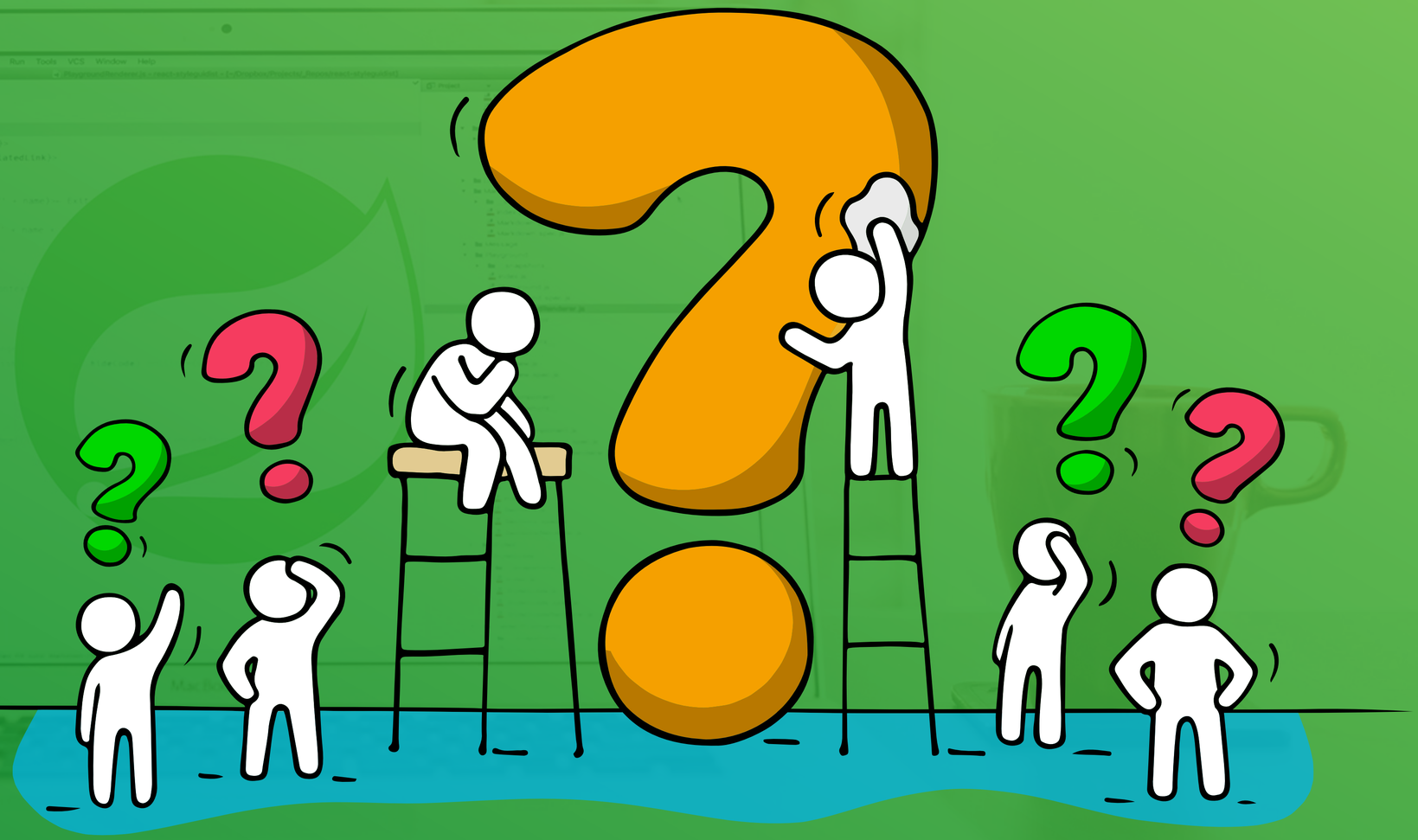
AspectJ - II

- **Spring** follows approach and style of AspectJ for most of the issues.
  - Spring’s developer are still committers for AspectJ.
- **Spring** supports AspectJ configuration and annotations.
  - For example Spring can be configured to work with the objects of AspectJ created outside of IoC container.

Spring and AspectJ

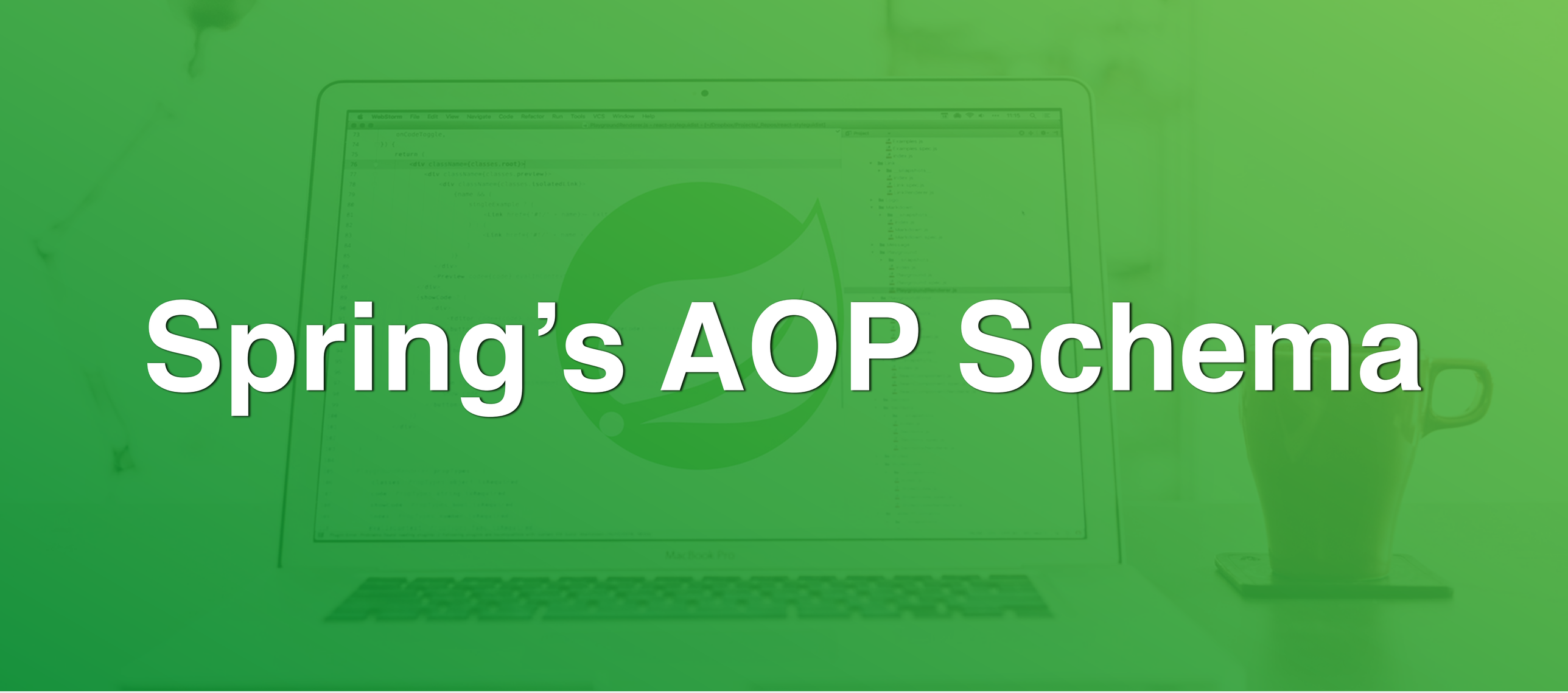
- To learn AOP either Spring AOP or AspectJ itself are good starting points.
- But for developers who already know Spring's DI mechanism its easier to learn Spring AOP.
- Similarly for projects that utilize Spring, Spring AOP would be a natural choice.
- But for advanced AOP functionality AspectJ can be used.

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr



Spring's AOP Schema

selsoft

build better, deliver faster

Spring's AOP Schema - I

- For XML-based configuration Spring offers a richer version of schema using **`<aop>`** tags specifically for AOP.
- AOP schema is at http://www.springframework.org/schema/aop
- Main element of this schema is **`<aop:config>`** element; all AOP elemens must be inside it.
- An aspect can be declared by **`<aop:aspect>`** element.

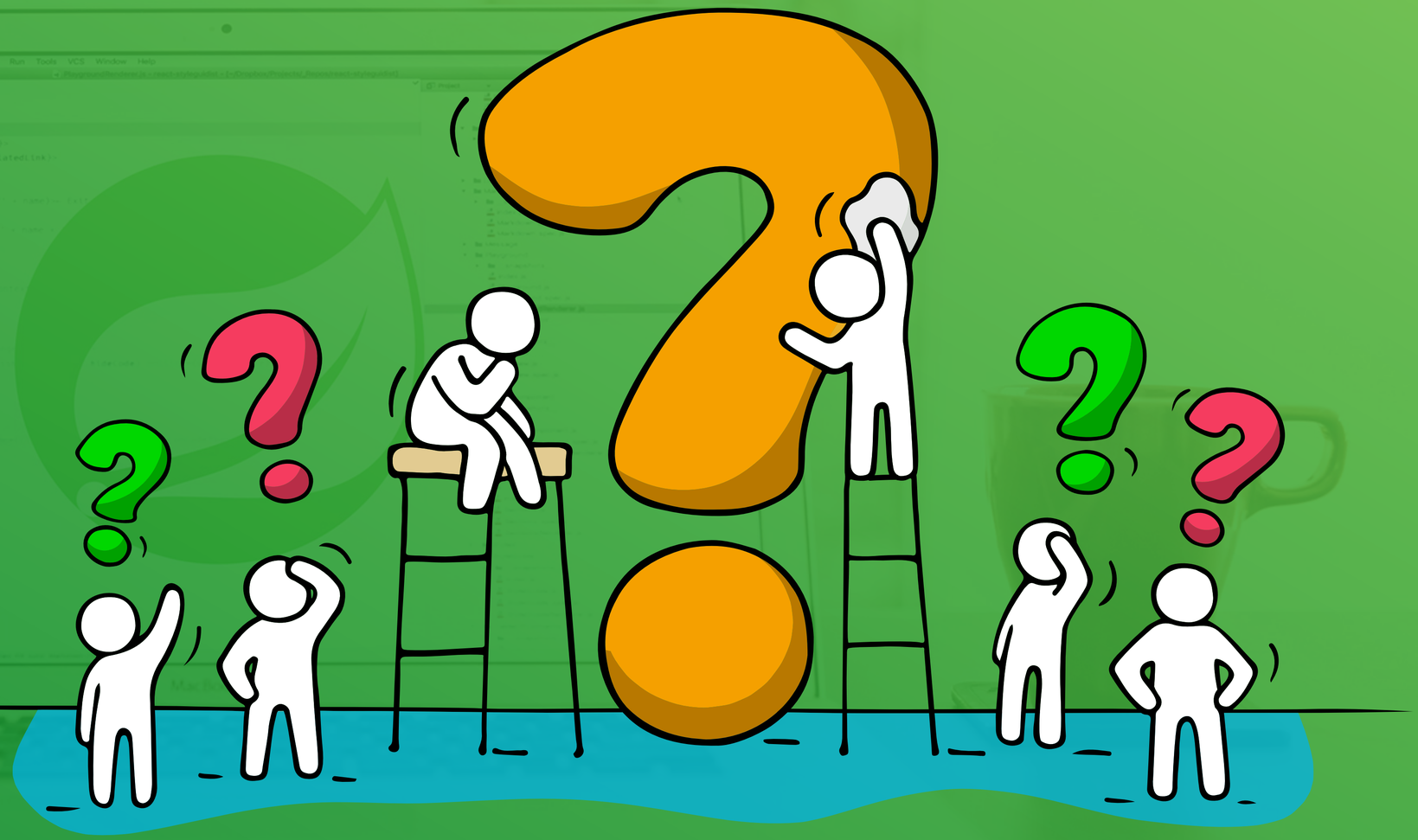
Spring's AOP Schema - II

- A point cut can be declared by **`<aop:pointcut>`** element.
  - The attribute **`expression`** of **`<aop:pointcut>`** element is used to match join points.
- And then advices are defined using **`<aop:before>`**, **`<aop:after-returning>`**, **`<aop:after-throwing>`**, **`<aop:after>`** and **`<aop:around>`**

SqrtAOPExample

- `org.javaturk.spring.aop.ch01.sqrt.aspectJ.SqrtAOPExample`

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

AspectJ Support

selsoft

build better, deliver faster

AspectJ Support - I

- **Spring** supports AspectJ's annotations.
- Main annotations are:
  - `@Aspect`
  - `@Before`
  - `@AfterThrowing`
  - `@After`
  - `@AfterReturning`

AspectJ Support - II

- All aspect objects are singleton in an application context.

SqrtAOPExample

- `org.javaturk.spring.aop.ch01.greeting.aspectJ.SqrtAOPExample`

Order for Advice

Order of Advice - I

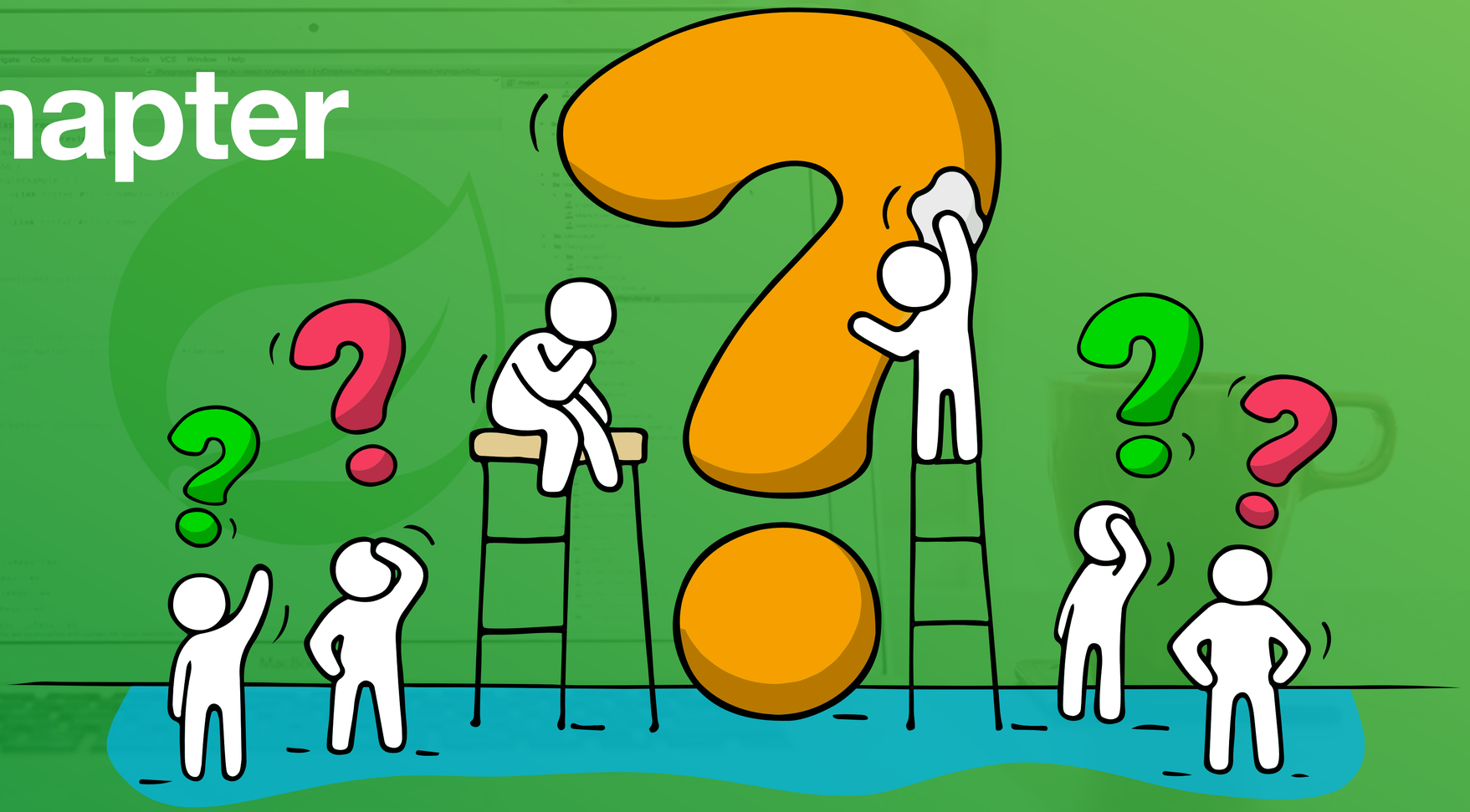
- More than one advice can be applied to a join point.
- Normally the order of execution of two pieces of advice of the same type that need to run at the same join point is undefined.
- `@Order` and `Ordered` interface can be used to decide the order among different pieces of advice to be applied for the same join point.

Order of Advice - II

- As of **Spring** 5.2.7 advice methods defined in the same `@Aspect` class that need to run at the same join point are assigned precedence based on their advice type in the following order, from highest to lowest precedence:
  - `@Around @Before @After @AfterReturning @AfterThrowing`

End of Chapter

Time for Questions!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr