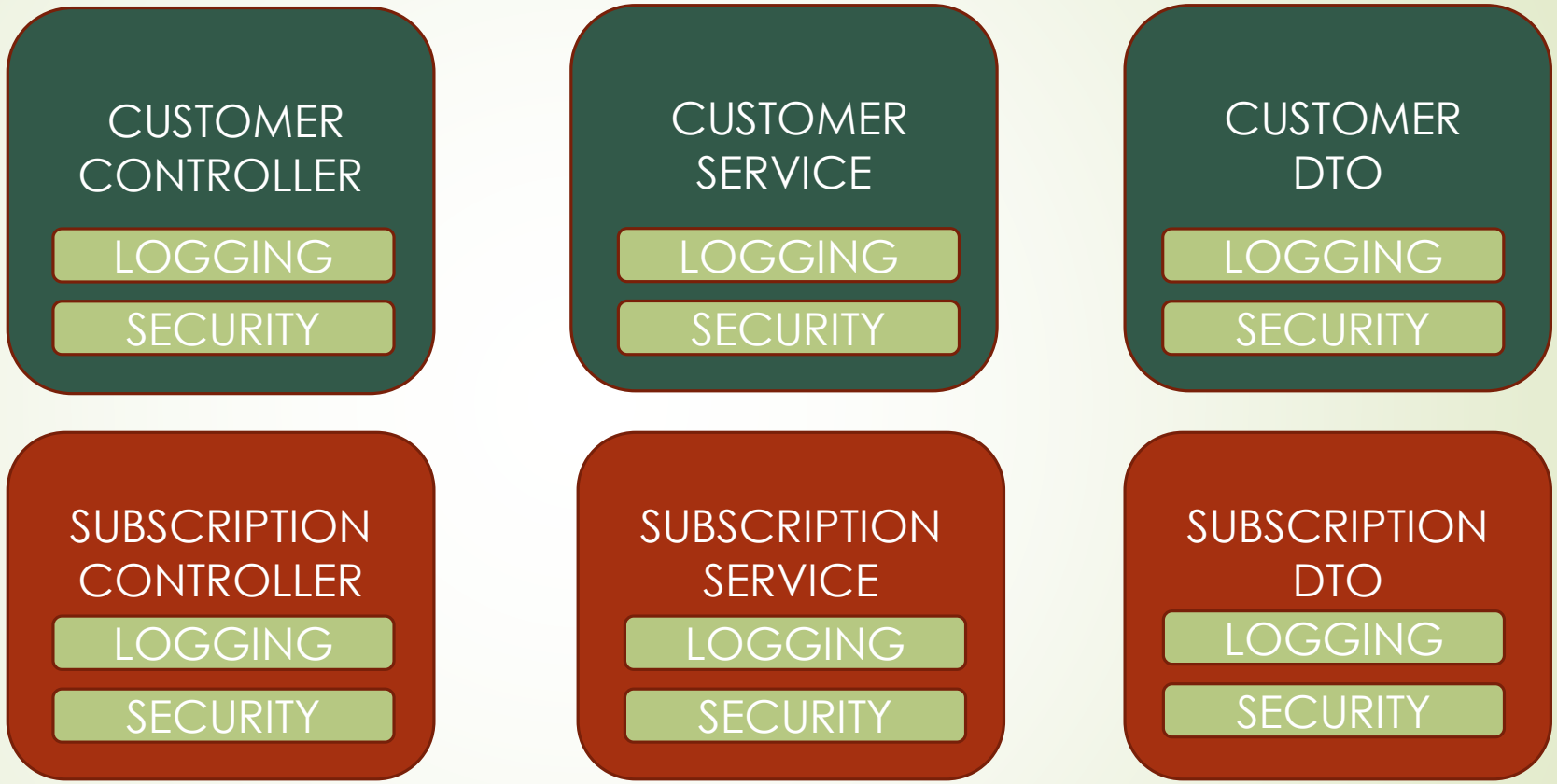
JAVA SPRING CURS 3: AOP

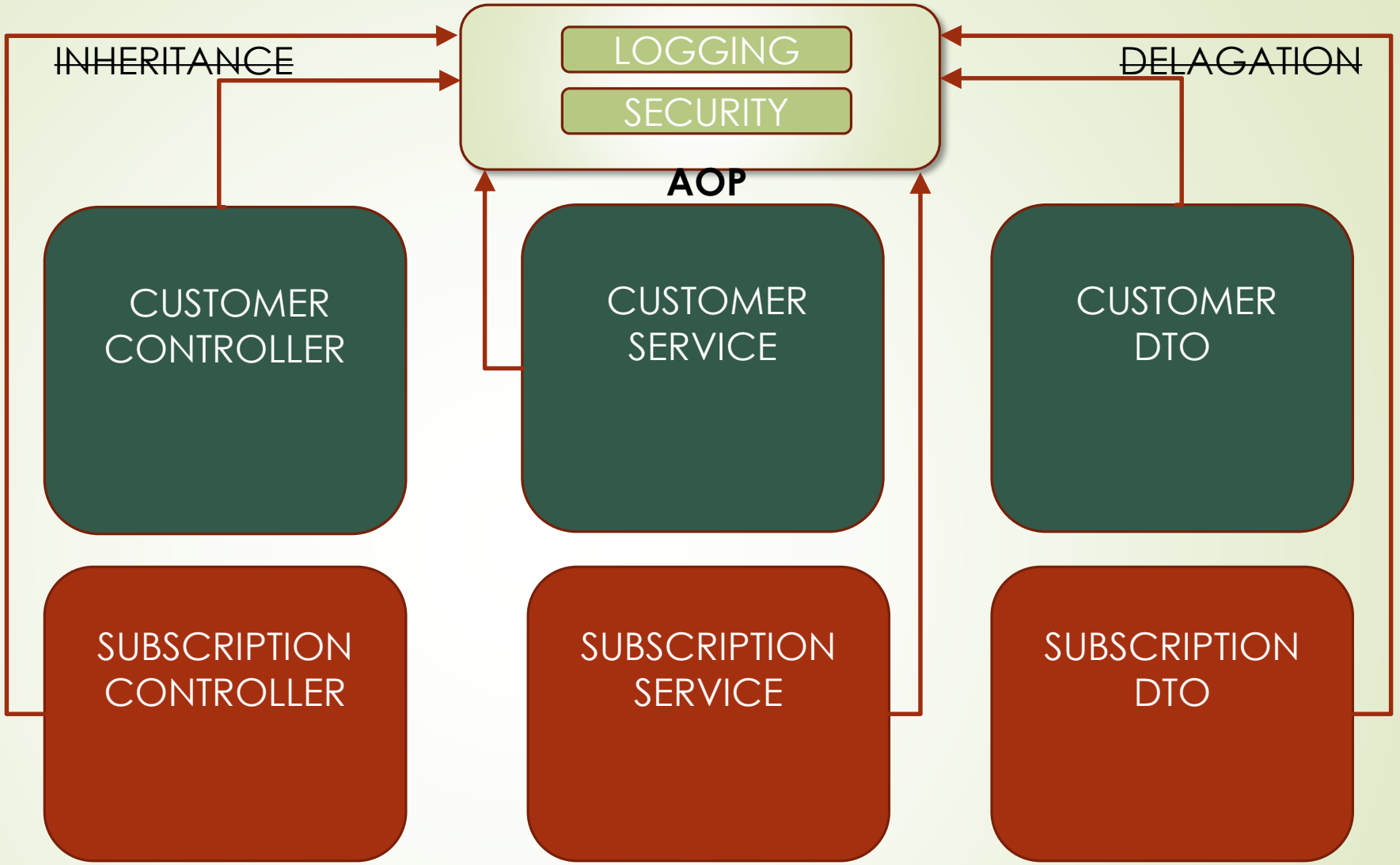
AOP

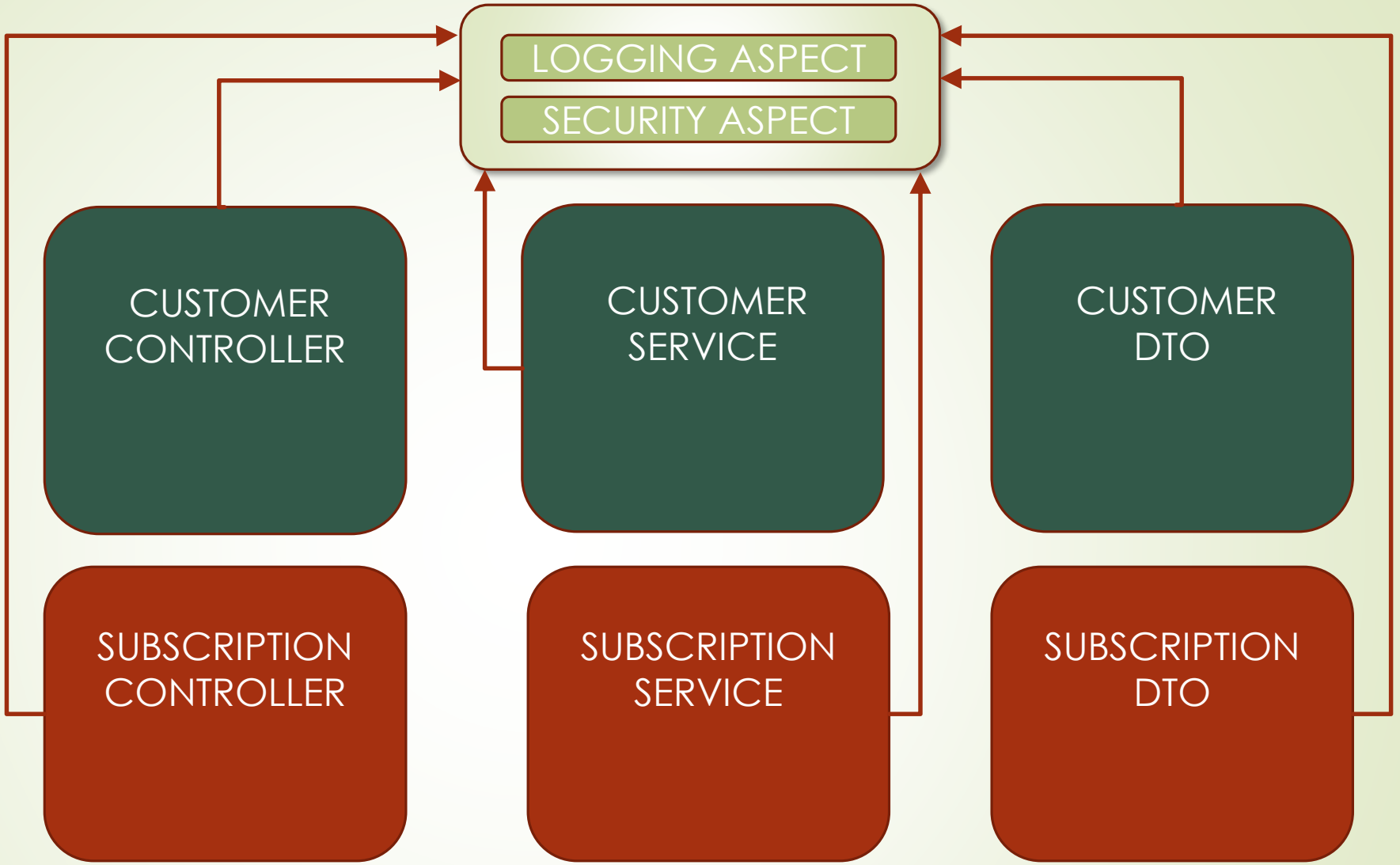
Aspect Oriented Programming basic concepts

Aspect Oriented Programming

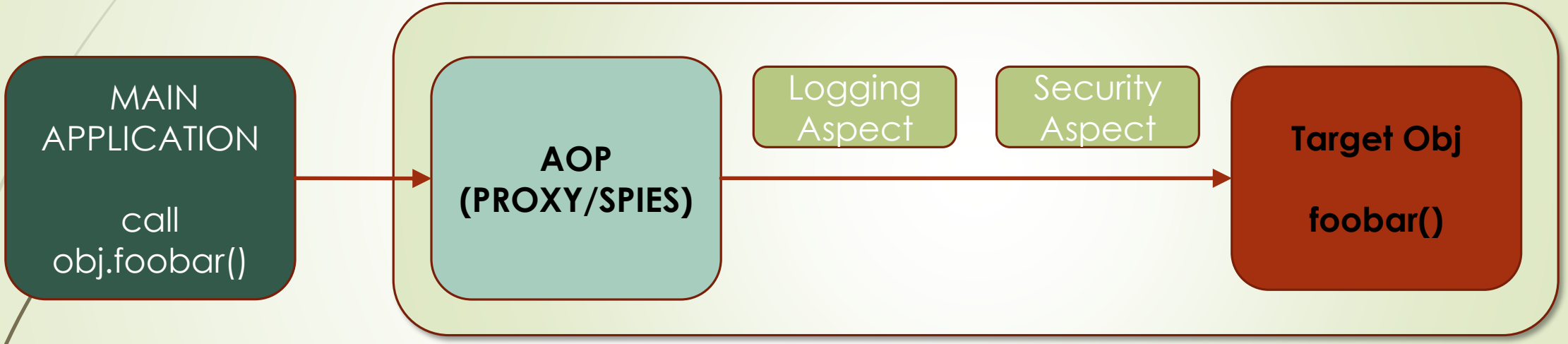
- Technique based on the concept of Aspect
- Add additional behavior without modification of the existing code.
- An aspect encapsulates **cross-cutting logic (corss-cutting concerns).**
- An aspect can be reused in different functionalities.


CUSTOMER CONTROLLER
LOGGING
SECURITY
CUSTOMER SERVICE
LOGGING
SECURITY
CUSTOMER DTO
LOGGING
SECURITY
SUBSCRIPTION CONTROLLER
LOGGING
SECURITY
SUBSCRIPTION SERVICE
LOGGING
SECURITY
SUBSCRIPTION DTO
LOGGING
SECURITY


INHERITANCE
LOGGING
SECURITY
DELAGATION
AOP
CUSTOMER CONTROLLER
CUSTOMER SERVICE
CUSTOMER DTO
SUBSCRIPTION CONTROLLER
SUBSCRIPTION SERVICE
SUBSCRIPTION DTO


LOGGING ASPECT
SECURITY ASPECT
CUSTOMER CONTROLLER
CUSTOMER SERVICE
CUSTOMER DTO
SUBSCRIPTION CONTROLLER
SUBSCRIPTION SERVICE
SUBSCRIPTION DTO

Proxy design pattern



- **Weaving**: connectiong aspects to target objects (compile-time, load-time or run-time).

Aspect Oriented Programming

- Avoids scattered code.
- Makes code easy to reuse or change.
- Based on configurations an ascpect can be selectively applyed to different modules of application.
- Applications: logging, security, audit logging, notifications etc.

- May be hard to follow.
- Performance costs.

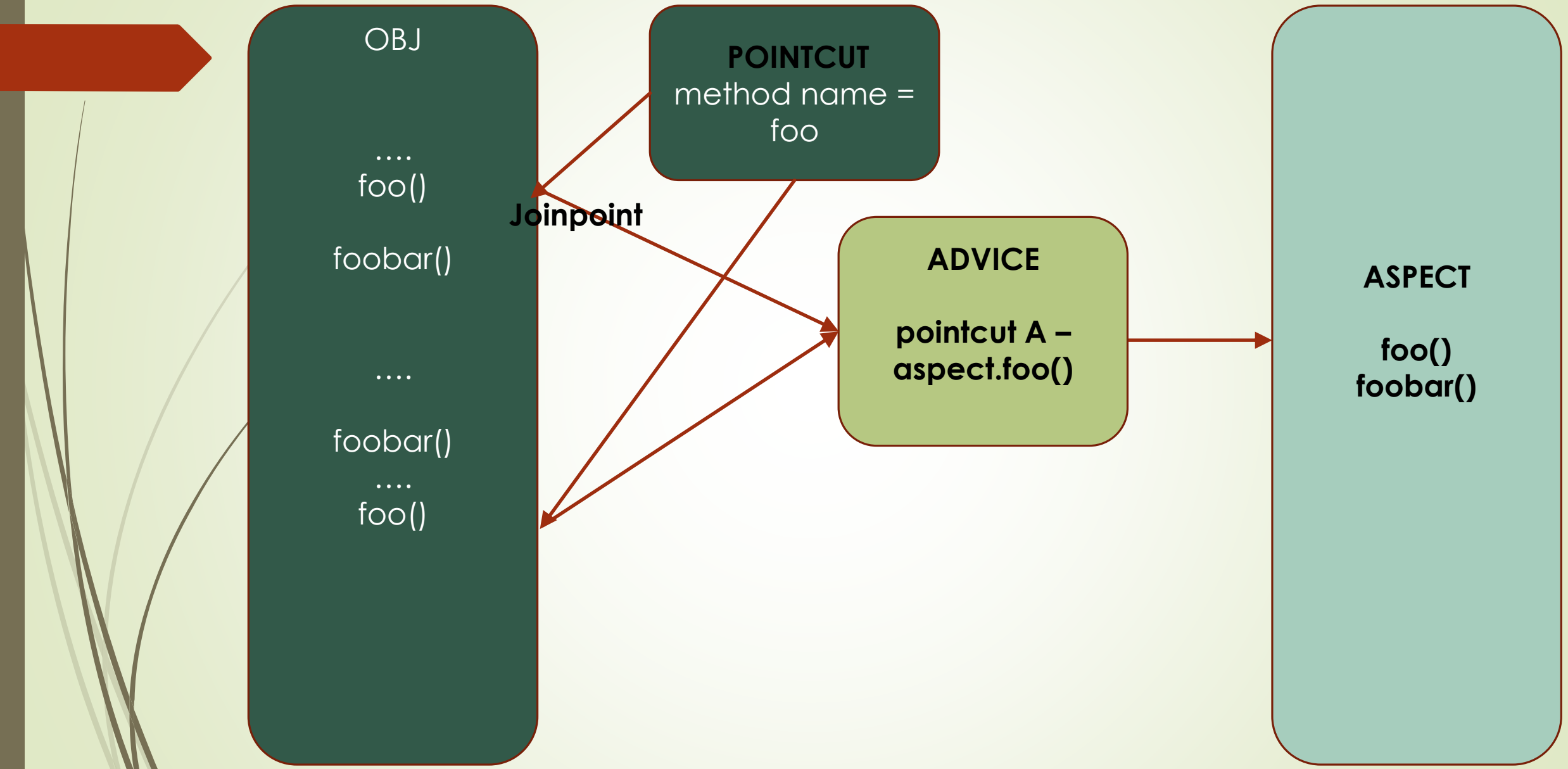
AOP

Aspect Oriented Programming basic terminology

AOP

- **Aspect**: module of code for a cross-cut concern
- **Join point**: a point during the execution of a program
- **Advice**: action taken by an aspect at a particular join point
- **Pointcut**: predicate that matches join points
  example: call a method with a certain name

Advice is associated with a **pointcut expression** and runs at any **join point** matched by the pointcut


OBJ
....
foo()
foobar()
....
foobar()
....
foo()
POINTCUT
method name = foo
Joinpoint
ADVICE
pointcut A – aspect.foo()
ASPECT
foo()
foobar()

Advice types

- **Before**: run the code before the join point
- **After returning**: run the code after a join point completes normally
- **After throwing**: run the code if a method exits by throwing an exception
- **After finally**: run the code after join point exits (normal or by throwing exception)
- **Around advice**: run the code before and after join point

Advice types

- **Before**: run the code before the join point
- **After returning**: run the code after a join point completes normally
- **After throwing**: run the code if a method exits by throwing an exception
- **After finally**: run the code after join point exits (normal or by throwing exception)
- **Around advice**: run the code before and after join point

AOP Java frameworks

AspectJ and Spring

AOP Java frameworks

- **Aspect J**:
  - First AOP Java framework
  - Complete support for AOP
  - Faster than Spring
  - More complex than Spring AOP
  - Apply aspects to POJOs

AOP Java frameworks

- **Spring AOP**:
  - Uses AOP for security, transactions etc
  - Uses run-time weaving
  - Only supports method-level join points
  - Apply aspects to beans only

AOP Java frameworks

- **Spring AOP**:
  - Can use @Aspect annotation
  - Uses proxy pattern
  - Light implementation of AspectJ (easy tot use)

AOP Configuration

AOP Configuration Class

- **@EnableAspectJAutoProxy**

```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages = "com.apbdoo.lab8")
public class AOPConfig {

}
```

@Before advice

Before/After/Around Advice

- **@Before(poincut expression)**

```
@Aspect
@Component
public class LoggingAspect{
    @Before("execution(public void saveCustomer(..))")
    public void beforeSaveCustomerAdvice
}
```

Pointcut expression method designator

- **execution**
- **withincode** **all statements**
- **call** **all calling statements**

- **execution ([modifiers-pattern] [return-type-pattern] [declaring-type-pattern] method-name-pattern(param-pattern) [throws-pattern])**
- **wildcard ***
- **param-pattern: () (*) (..) (declaring-type-pattern)**

JoinPoint

- **@Before(poincut expression)**

```
@Aspect
@Component
public class LoggingAspect{
    @Before("execution(public void saveCustomer(..))")
    public void beforeSaveCustomerAdvice( JoinPoint joinPoint )
}
```

JoinPoint

- **getSignature**
- **getArgs**

- **execution ([modifiers-pattern] [return-type-pattern] [declaring-type-pattern] method-name-pattern(param-pattern) [throws-pattern])**
- **wildcard ***
- **param-pattern: () (*) (..) (declaring-type-pattern)**

Bibliografie

- https://docs.spring.io/spring/docs/2.5.x/reference/aop.html
- https://www.baeldung.com/spring-aop