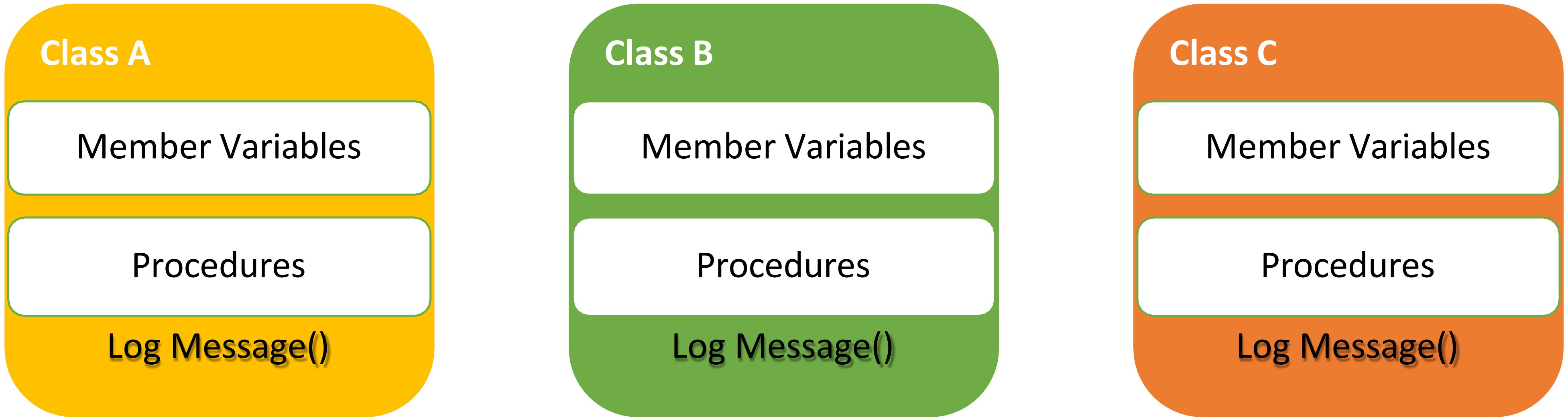
Advance Software Engineering Aspect-Oriented Programming

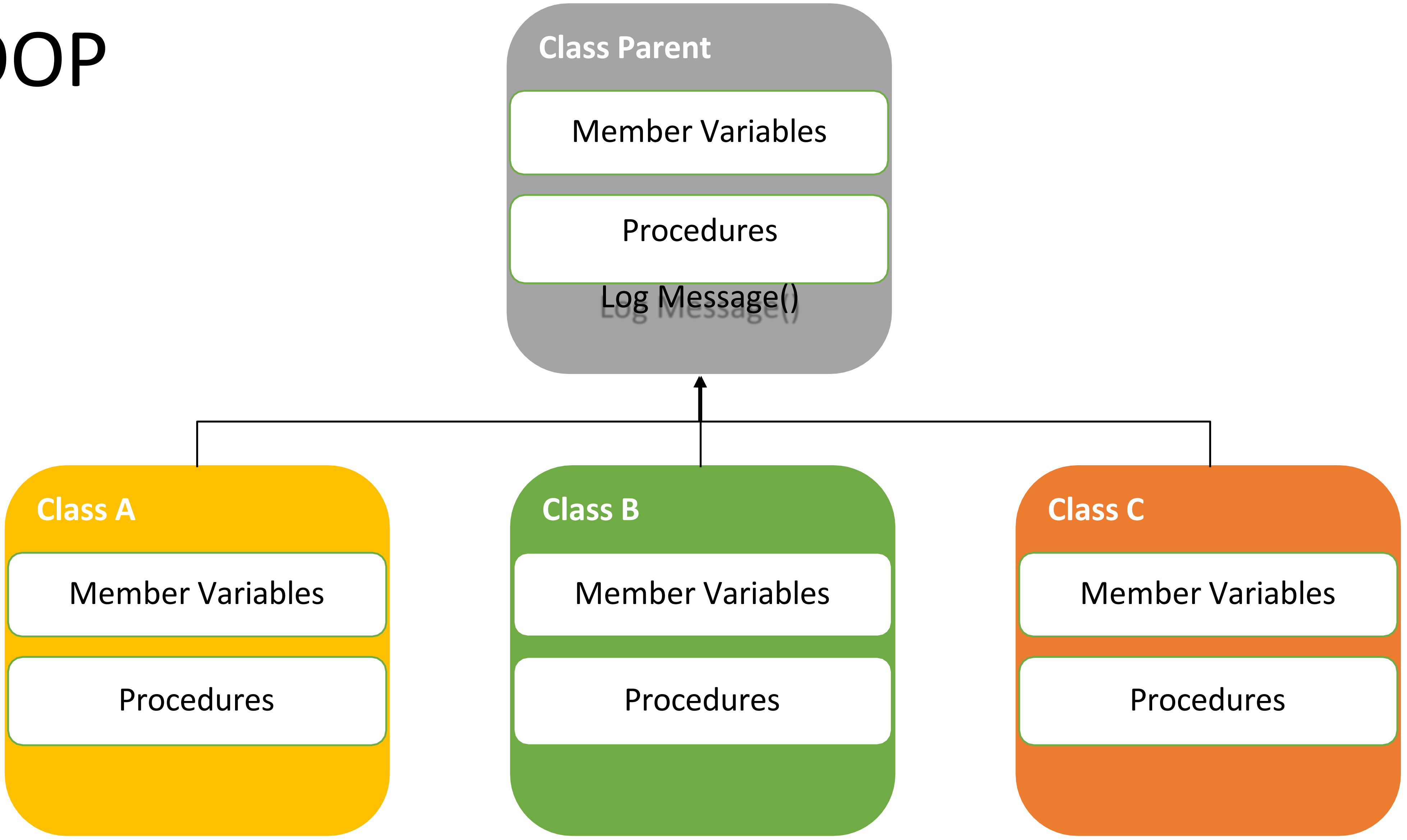
By:

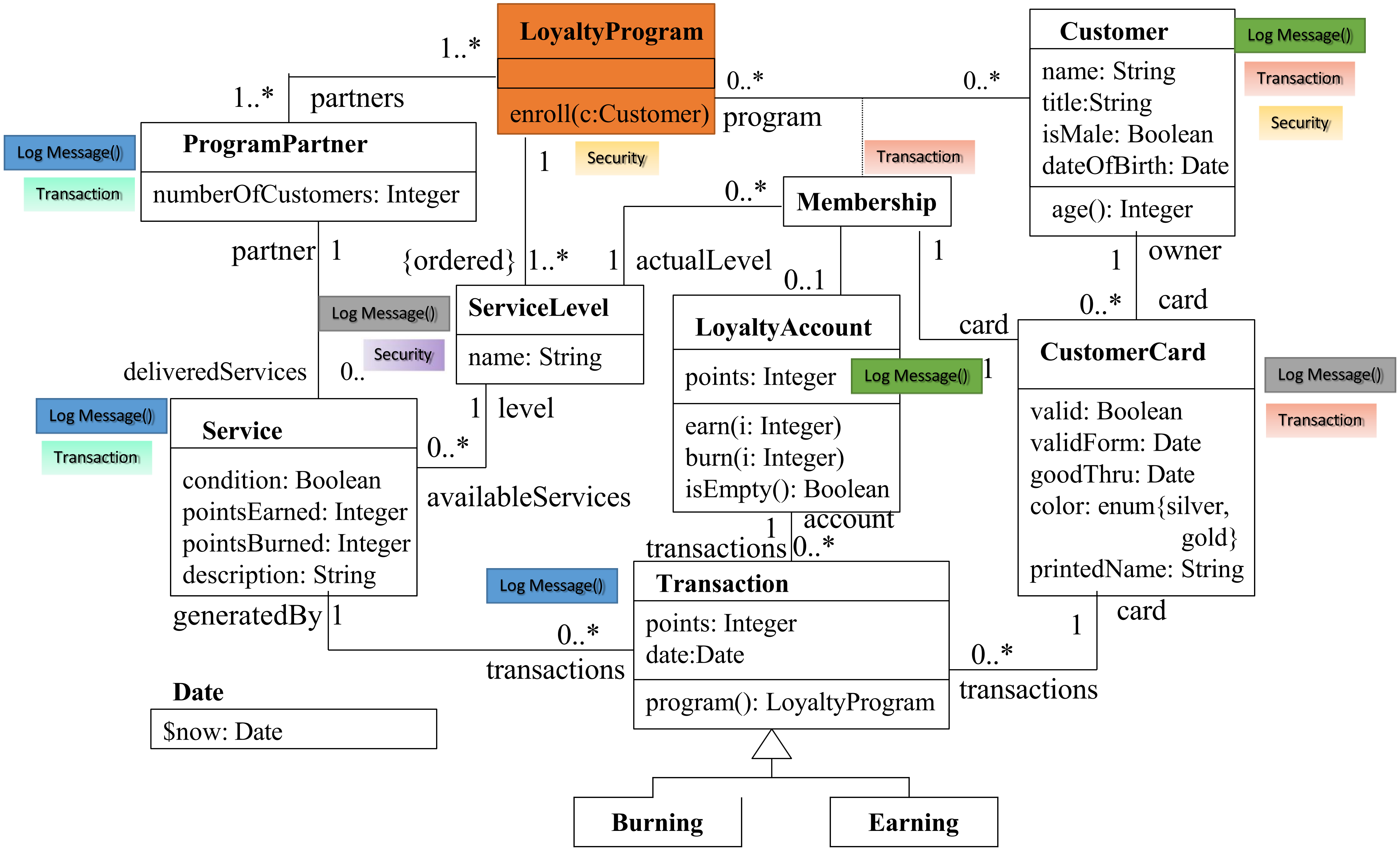
Dr. Salwa Osama

OOP


Class A
Member Variables
Procedures
Log Message()
Class B
Member Variables
Procedures
Log Message()
Class C
Member Variables
Procedures
Log Message()

OOP


Class Parent
Member Variables
Procedures
Log Message()
Class A
Member Variables
Procedures
Class B
Member Variables
Procedures
Class C
Member Variables
Procedures


LoyaltyProgram
enroll(c:Customer)
Security
Customer
name: String
title:String
isMale: Boolean
dateOfBirth: Date
age(): Integer
Log Message()
Transaction
Security
ProgramPartner
numberOfCustomers: Integer
Log Message()
Transaction
partners
1..*
1..*
0..*
0..*
program
Transaction
Membership
0..*
1
{ordered}
1..*
1
actualLevel
0..1
1
owner
1
0..*
card
card
partner
1
Log Message()
Security
ServiceLevel
name: String
deliveredServices
0..
LoyaltyAccount
points: Integer
earn(i: Integer)
burn(i: Integer)
isEmpty(): Boolean
Log Message()
1
CustomerCard
valid: Boolean
validForm: Date
goodThru: Date
color: enum{silver, gold}
printedName: String
Log Message()
Transaction
Log Message()
Transaction
Service
condition: Boolean
pointsEarned: Integer
pointsBurned: Integer
description: String
1
level
0..*
availableServices
1
account
transactions
0..*
Log Message()
Transaction
points: Integer
date:Date
program(): LoyaltyProgram
generatedBy
1
0..*
transactions
card
1
0..*
transactions
Date
$now: Date
Burning
Earning

Privilege

Member Variables

Procedures

Logger

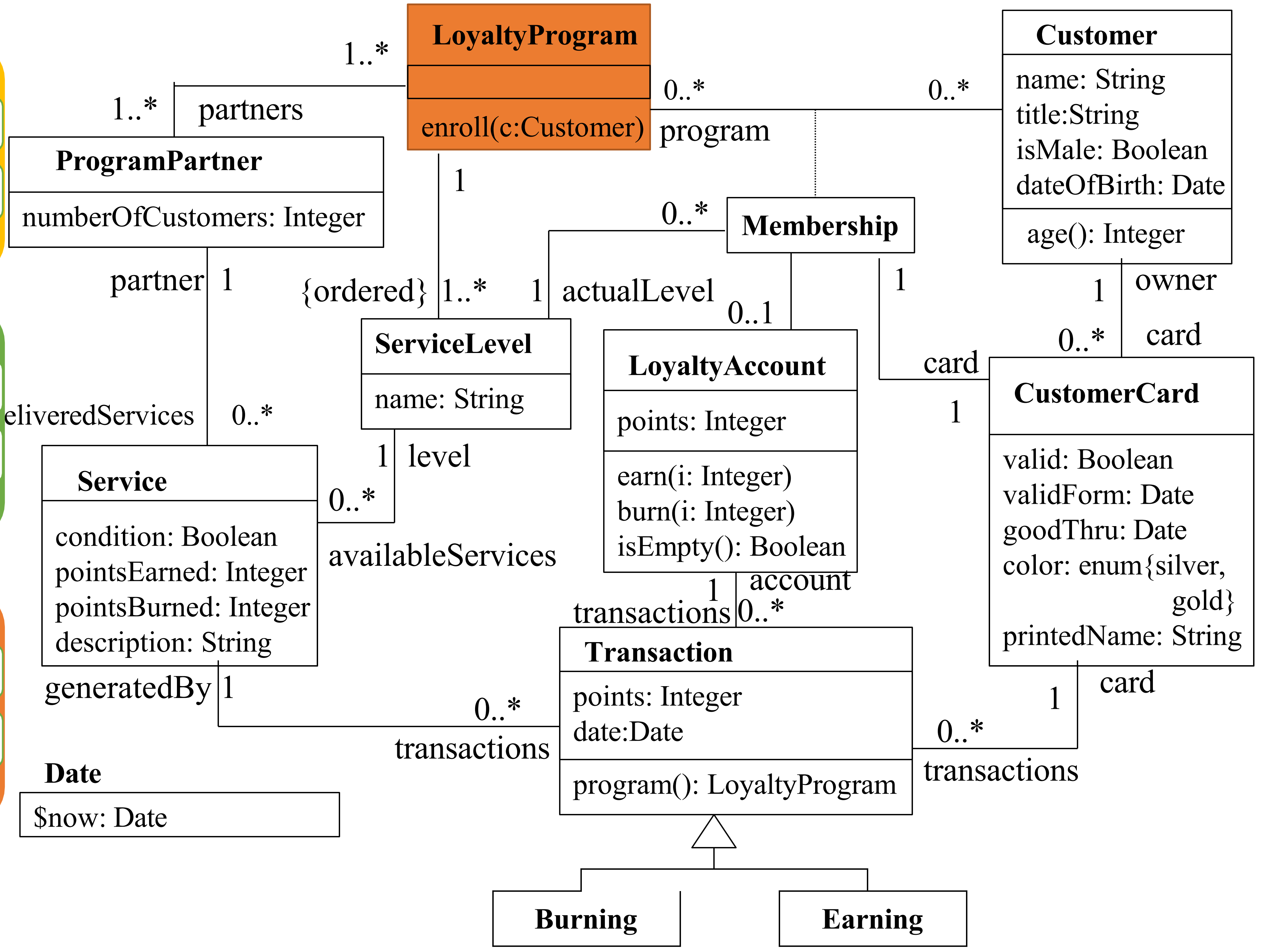
Member Variables

Procedures

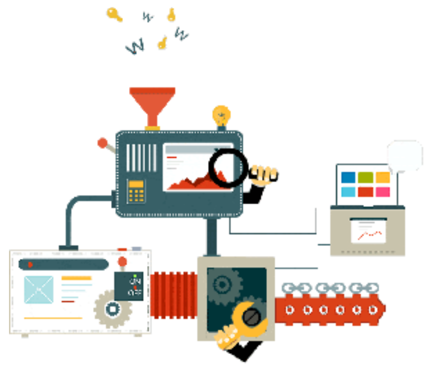
DB

Member Variables

Procedures

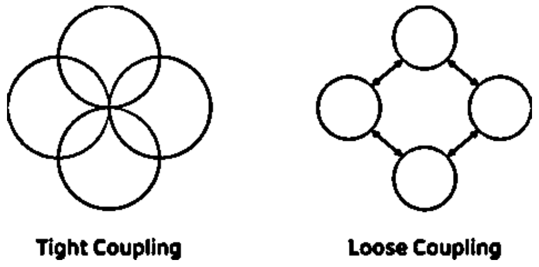


Cuts across multiple abstractions



Boilerplate code

Difficult to decompose



High-coupling

Problems

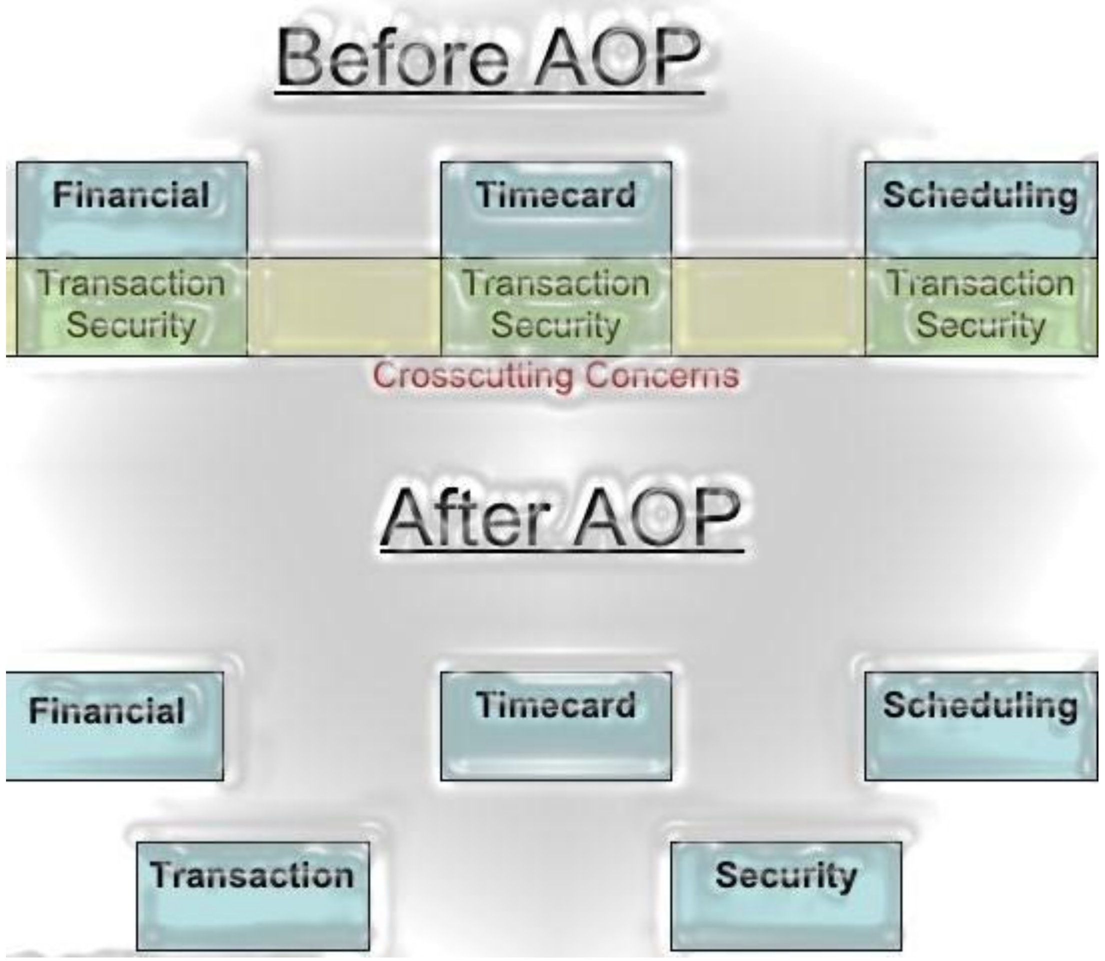
Code tangling

Code scattering

- **Poor traceability** – multiple concerns in the same module breaks linkage between requirement and its implementation, making it harder to understand what a piece of code is doing to address the problem domain.
- **Lower productivity** – developers are spending too much time and attention to peripheral issues rather than the problem domain.
- **Less code reuse** – boilerplate code propagated through cut and paste… no code reuse here!
- **Harder refactoring** – changing requirements means touching many modules for a single concern.

Aspect Oriented Programming

Aspect Oriented Programming (AOP)



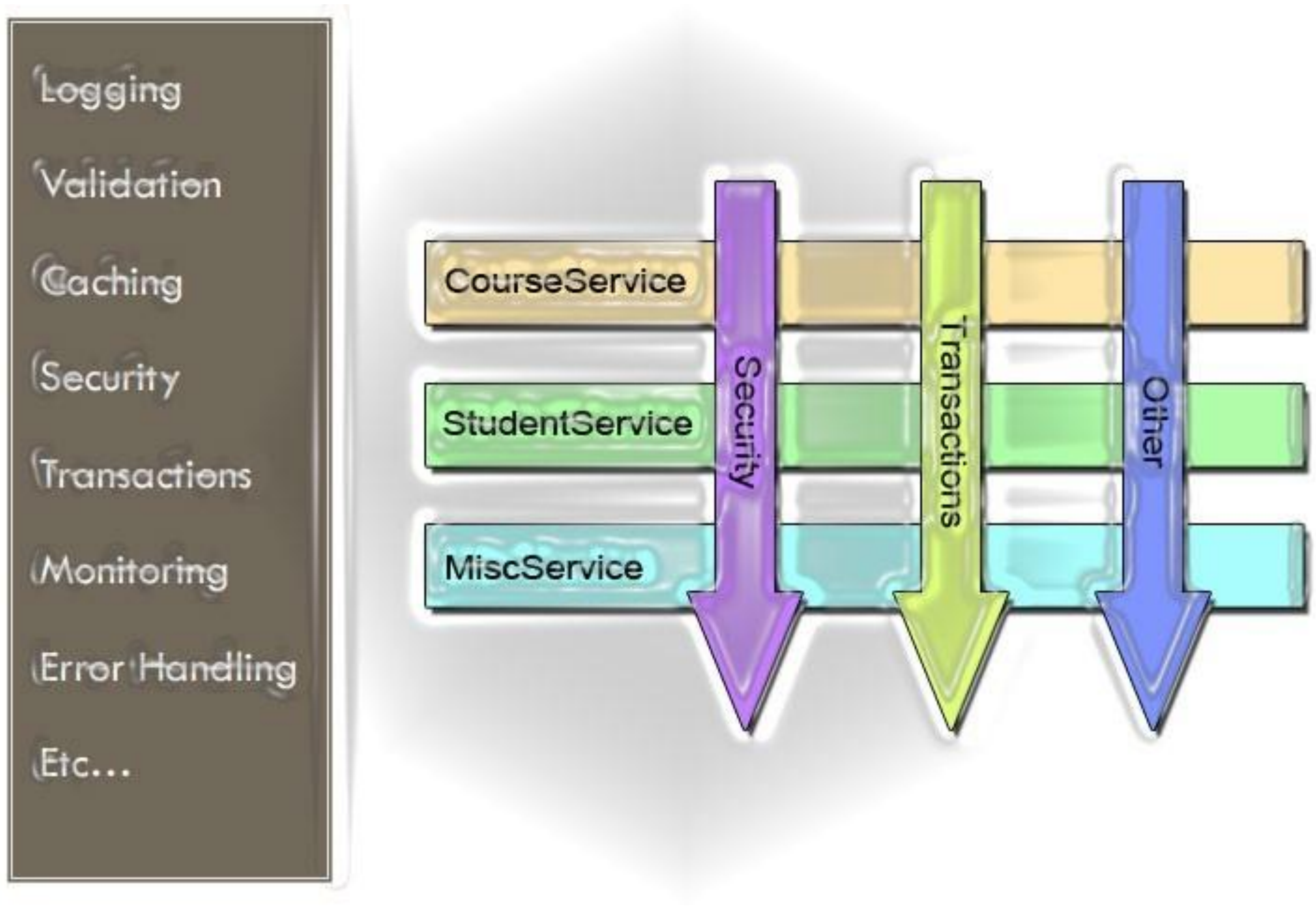
Concerns

- Concerns are not program issues but reflect the system requirements and the priorities of the system stakeholders.
  - Examples of concerns are **performance, security, specific functionality**, etc.
- **By reflecting the separation of concerns** in a program, there is clear traceability from requirements to implementation.
- **Core concerns** are the functional concerns that relate to the primary purpose of a system; **secondary concerns** are functional concerns that reflect non-functional requirements.

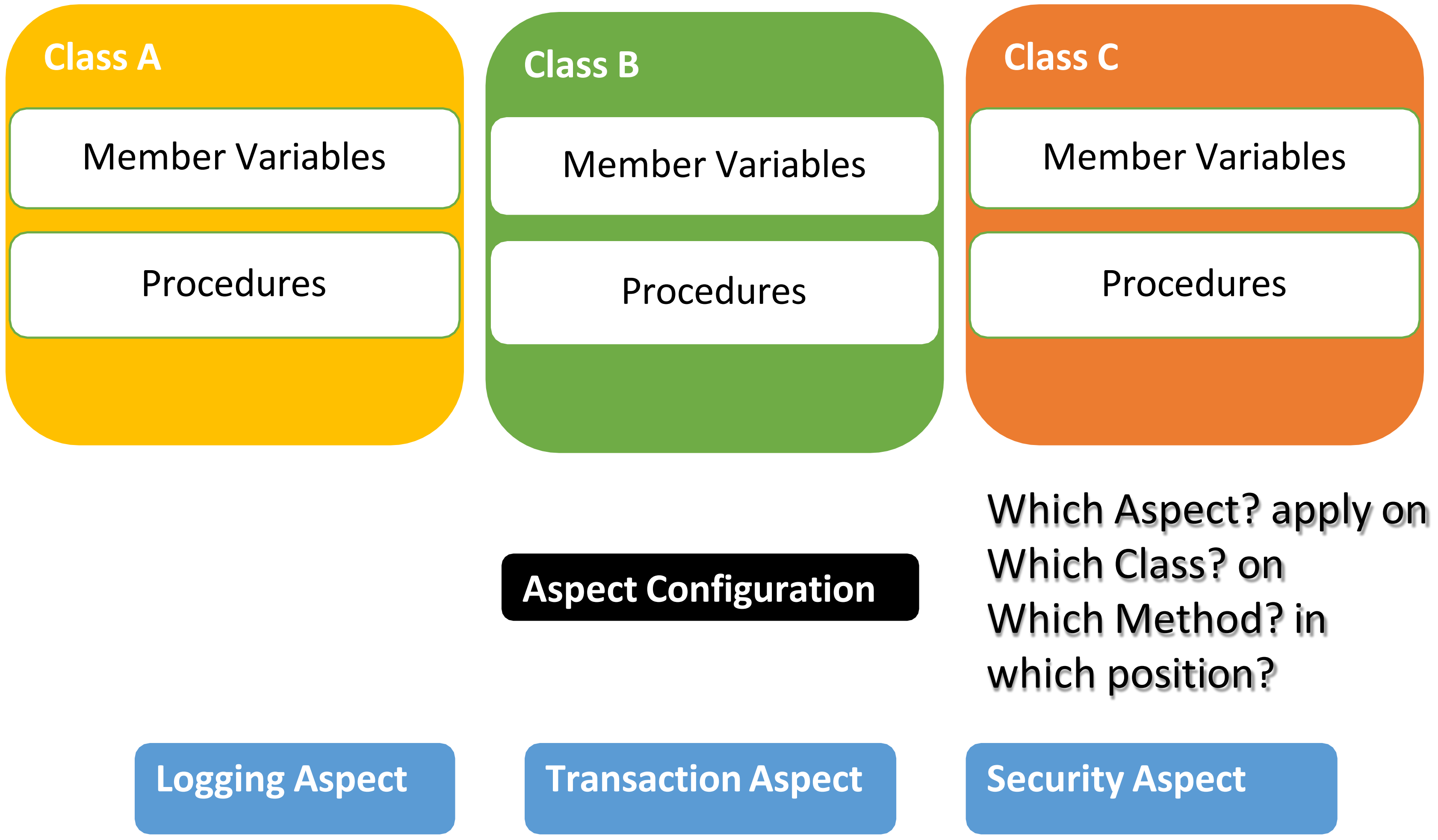
Stakeholder concerns

- **Functional** concerns which are related to **specific functionality** to be included in a system.
- **Quality** of service concerns which are related to the **non-functional behaviour** of a system.
- **Policy** concerns which are related to the **overall policies** that govern the use of the system.
- **System** concerns which are related to attributes of the system as a whole such as its **maintainability** or its **configurability**.
- **Organisational** concerns which are related to **organisational goals and priorities** such as producing a system within budget, making use of existing software assets or maintaining the reputation of an organisation.

Cross-cutting Concerns

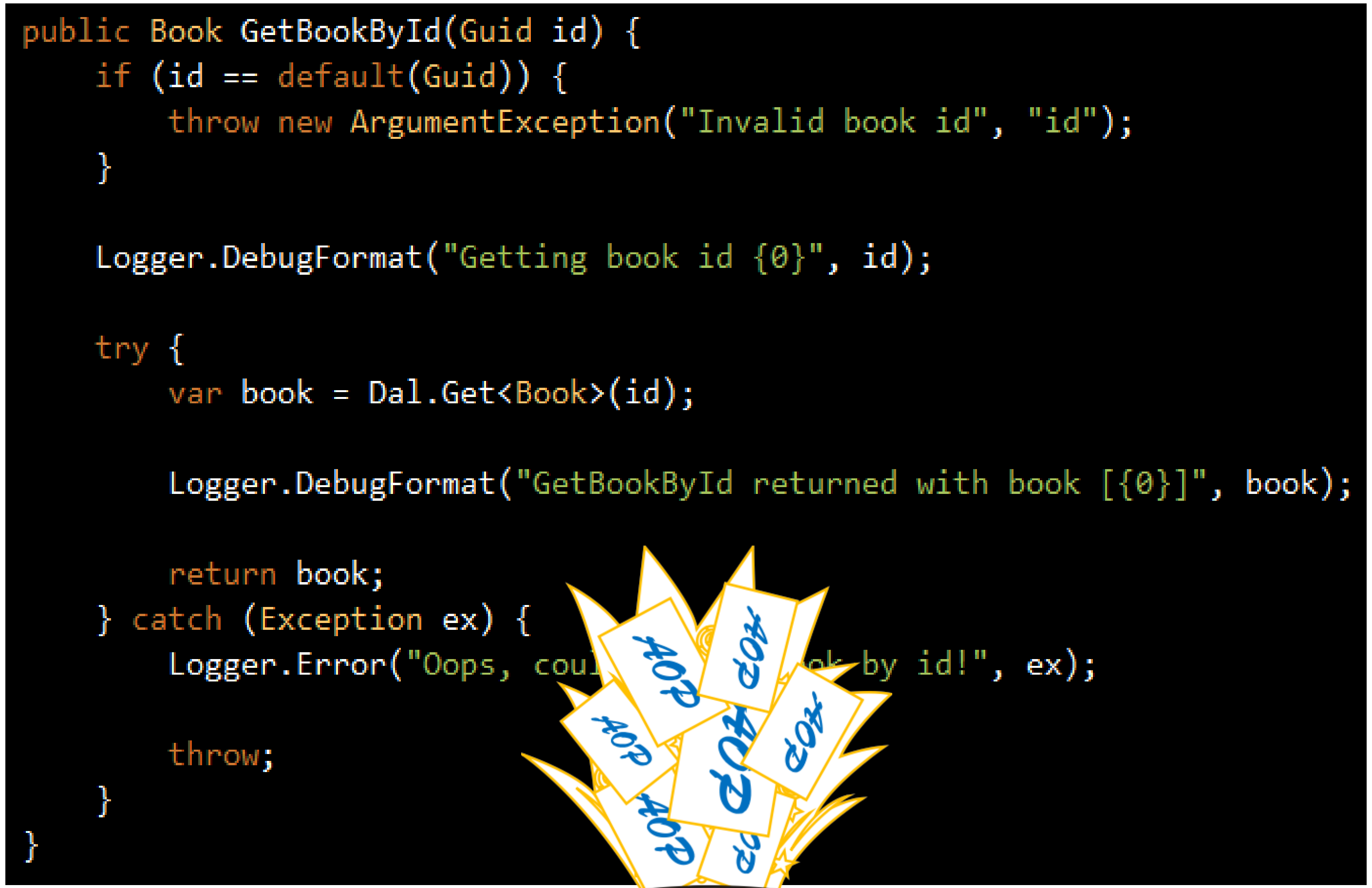


AOP



Which Aspect? apply on
Which Class? on
Which Method? in
which position?

AOP Demo



```
public Book GetBookById(Guid id) {
    if (id == default(Guid)) {
        throw new ArgumentException("Invalid book id", "id");
    }

    Logger.DebugFormat("Getting book id {0}", id);

    try {
        var book = Dal.Get<Book>(id);

        Logger.DebugFormat("GetBookById returned with book [{0}]", book);

        return book;
    } catch (Exception ex) {
        Logger.Error("Oops, cou[illegible]ok by id!", ex);

        throw;
    }
}
```

AOP Demo

```
/// <summary>
/// Simple repository using AOP attributes
/// </summary>
// all public methods starting with 'Get'
[CheckParameters(AttributeTargetElements = MulticastTargets.Method,
                 AttributeTargetMemberAttributes = MulticastAttributes.Public,
                 AttributeTargetMembers = "Get*",
                 AspectPriority = 2)]
// apply to all methods
[Trace(AttributeTargetElements = MulticastTargets.Method,
       AspectPriority = 1)]
public class AopRepository : IRepository {
    public AopRepository(IDataAccessLayer dal) {
        Dal = dal;
    }

    protected IDataAccessLayer Dal { get; set; }

    public Book GetBookById([NotDefaultGuid] Guid id) {
        return Dal.Get<Book>(id);
    }

    public Author GetAuthorById([NotDefaultGuid] Guid id) {
        return Dal.Get<Author>(id);
    }
}
```

Benefits

- Centralize concerns implementation
- Intercept method calls
- Inject new behavior
- More reusable code
- Cleaner code

What's in it for YOU?

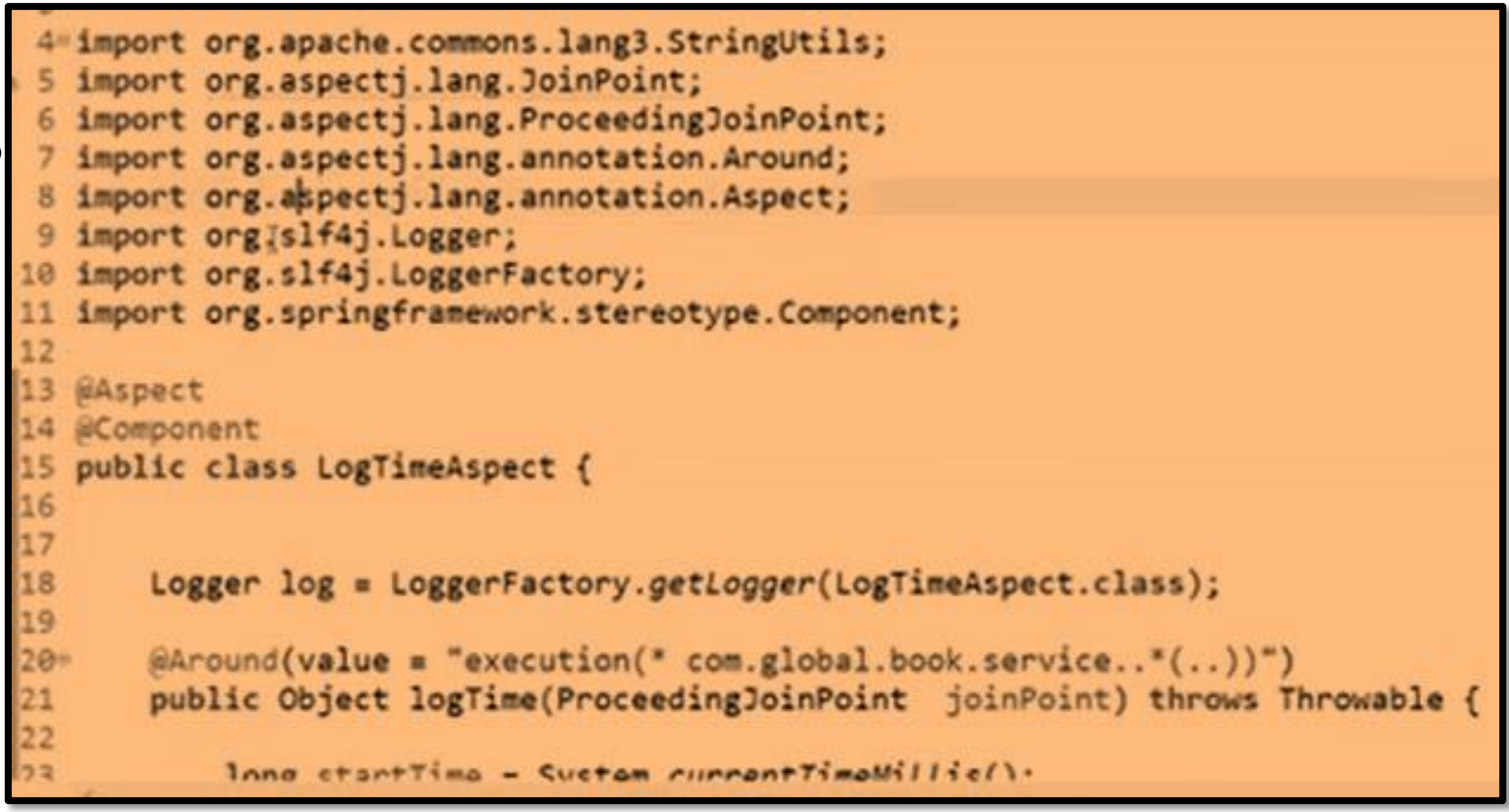
Write less code

Read less code

More maintainable

More concise and easy to understand

AOP Terminologies

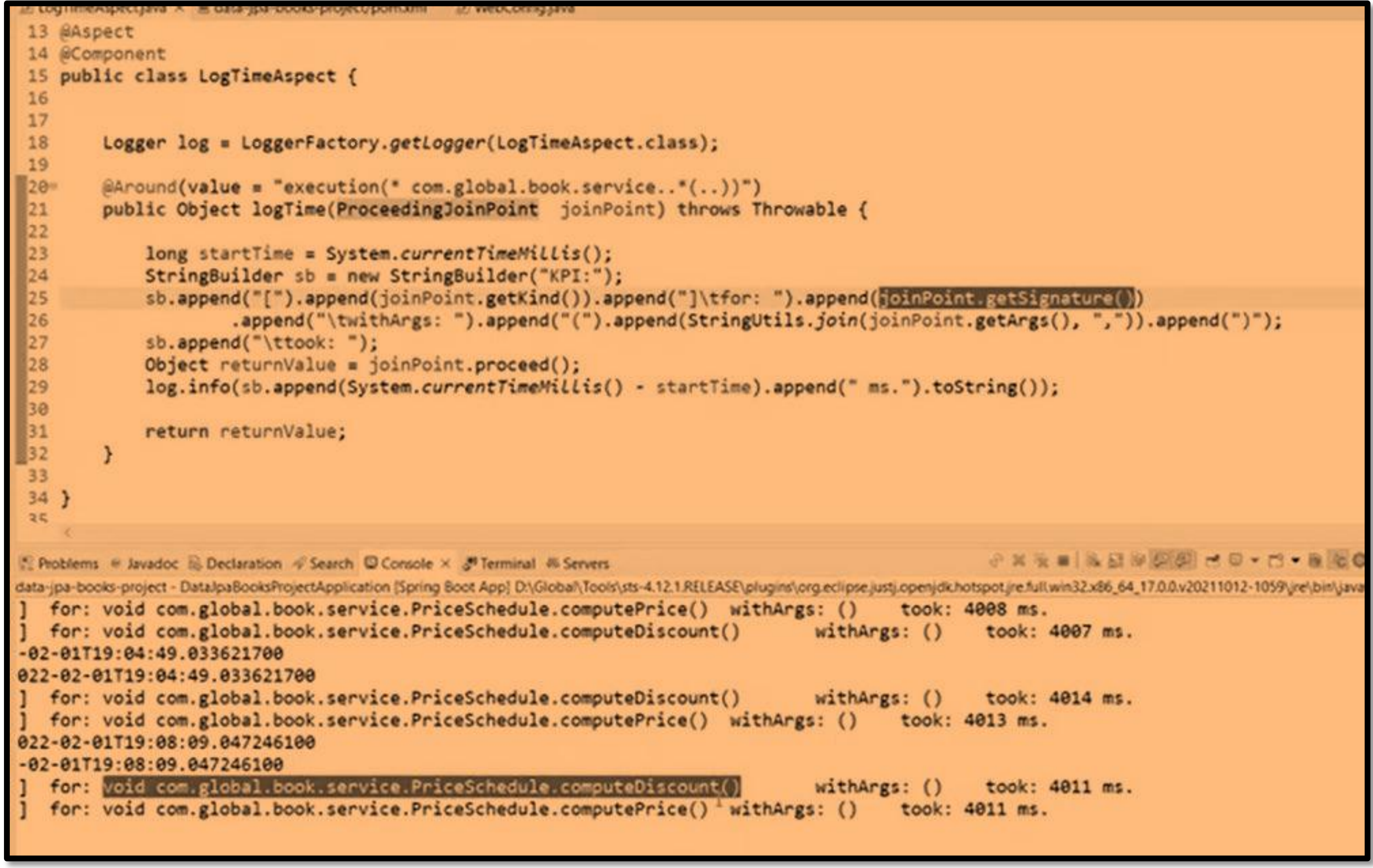


```
import org.apache.commons.lang3.StringUtils;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LogTimeAspect {


    Logger log = LoggerFactory.getLogger(LogTimeAspect.class);

    @Around(value = "execution(* com.global.book.service..*(..))")
    public Object logTime(ProceedingJoinPoint  joinPoint) throws Throwable {

        long startTime = System.currentTimeMillis();
```

Aspect

- It is a class that implements enterprise application concerns that cut across multiple classes such as transaction management.
- Aspects can be a normal class configuration through the spring XML configuration or we can use spring AspectJ integration to define class as Aspect using @aspect annotation

AOP Terminologies



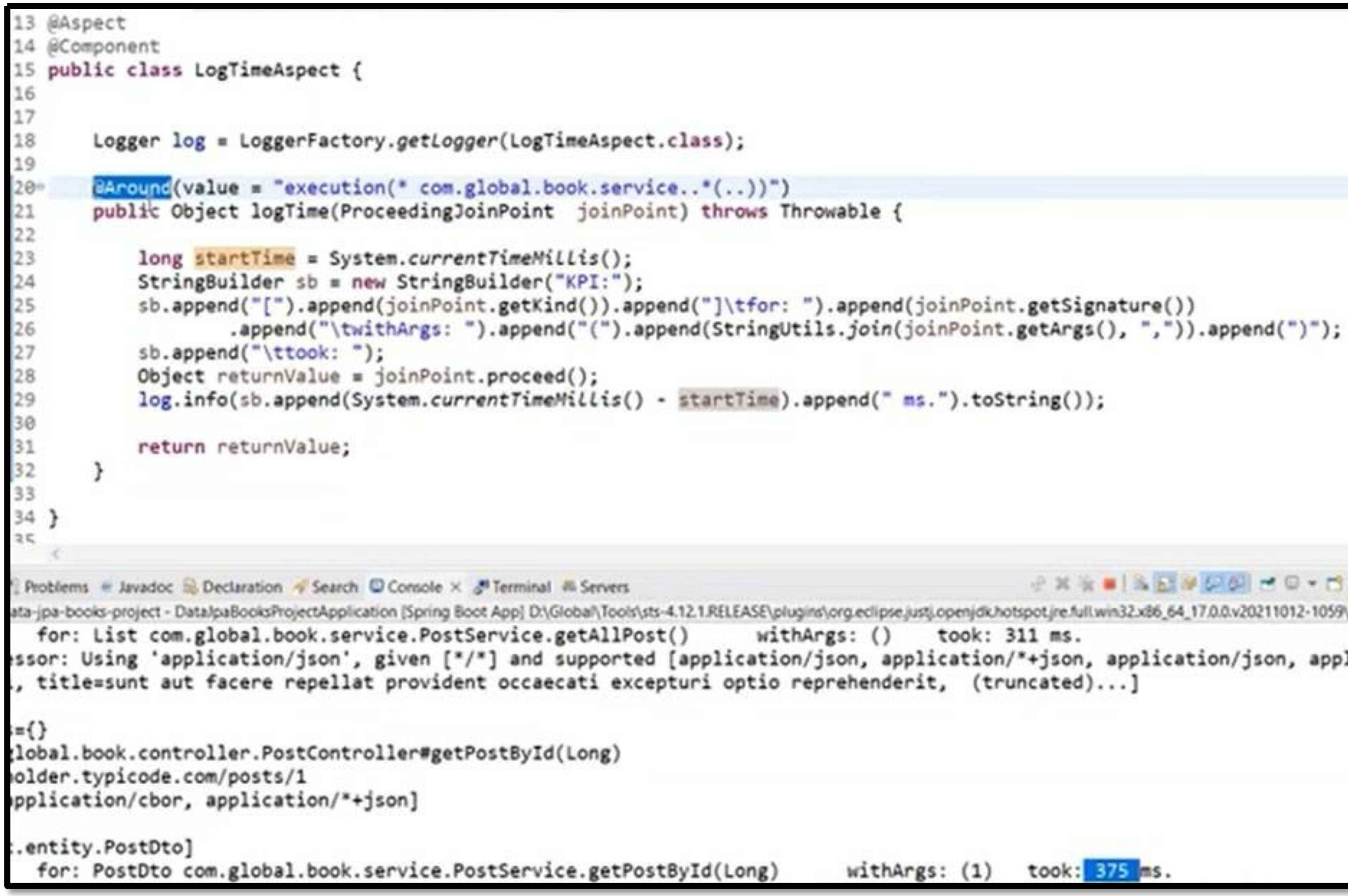
```java
@Aspect
@Component
public class LogTimeAspect {


    Logger log = LoggerFactory.getLogger(LogTimeAspect.class);

    @Around(value = "execution(* com.global.book.service..*(..))")
    public Object logTime(ProceedingJoinPoint  joinPoint) throws Throwable {

        long startTime = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder("KPI:");
        sb.append("[").append(joinPoint.getKind()).append("]\tfor: ").append(joinPoint.getSignature())
                .append("\twithArgs: ").append("(").append(StringUtils.join(joinPoint.getArgs(), ",")).append(")");
        sb.append("\ttook: ");
        Object returnValue = joinPoint.proceed();
        log.info(sb.append(System.currentTimeMillis() - startTime).append(" ms.").toString());

        return returnValue;
    }

}
```

Problems Javadoc Declaration Search Console Terminal Servers

```
data-jpa-books-project - DataJpaBooksProjectApplication [Spring Boot App] D:\Global\Tools\sts-4.12.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.0.v20211012-1059\jre\bin\java
]  for: void com.global.book.service.PriceSchedule.computePrice()  withArgs: ()    took: 4008 ms.
]  for: void com.global.book.service.PriceSchedule.computeDiscount()       withArgs: ()    took: 4007 ms.
-02-01T19:04:49.033621700
022-02-01T19:04:49.033621700
]  for: void com.global.book.service.PriceSchedule.computeDiscount()       withArgs: ()    took: 4014 ms.
]  for: void com.global.book.service.PriceSchedule.computePrice()  withArgs: ()    took: 4013 ms.
022-02-01T19:08:09.047246100
-02-01T19:08:09.047246100
]  for: void com.global.book.service.PriceSchedule.computeDiscount()       withArgs: ()    took: 4011 ms.
]  for: void com.global.book.service.PriceSchedule.computePrice()  withArgs: ()    took: 4011 ms.
```

- Join point:
  - It is a specific point in the application such as method execution, exception handling, changing object variable values. In Spring AOP, a join point is always the execution of the method.

AOP Terminologies



```java
@Aspect
@Component
public class LogTimeAspect {


    Logger log = LoggerFactory.getLogger(LogTimeAspect.class);

    @Around(value = "execution(* com.global.book.service..*(..))")
    public Object logTime(ProceedingJoinPoint  joinPoint) throws Throwable {

        long startTime = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder("KPI:");
        sb.append("[").append(joinPoint.getKind()).append("]\tfor: ").append(joinPoint.getSignature())
                .append("\twithArgs: ").append("(").append(StringUtils.join(joinPoint.getArgs(), ",")).append(")");
        sb.append("\ttook: ");
        Object returnValue = joinPoint.proceed();
        log.info(sb.append(System.currentTimeMillis() - startTime).append(" ms.").toString());

        return returnValue;
    }

}
```

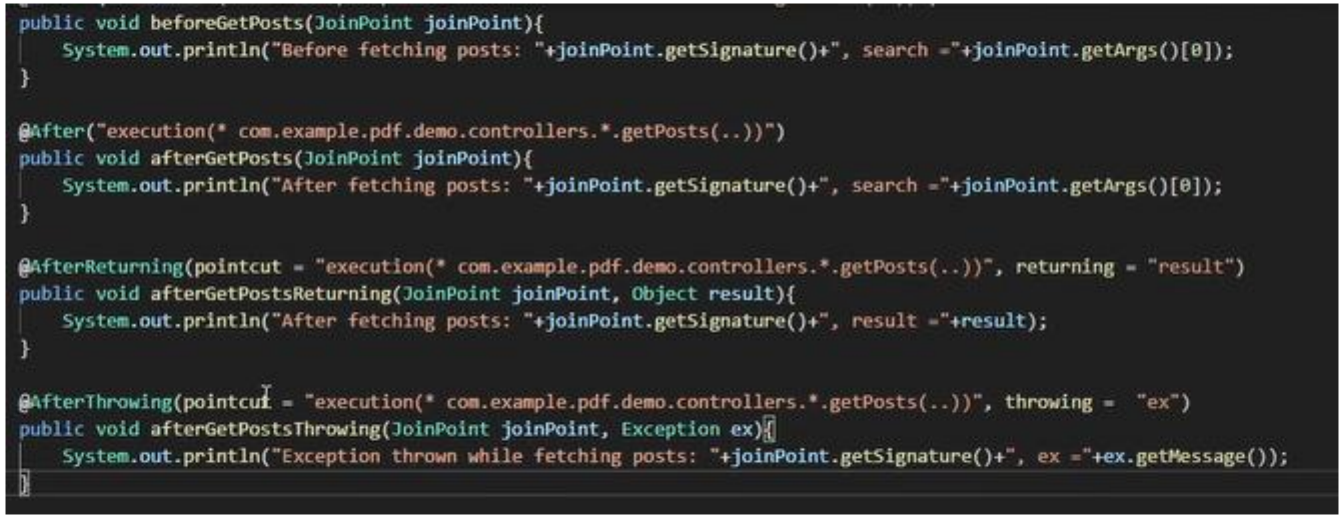
```
ata-jpa-books-project - DataJpaBooksProjectApplication [Spring Boot App] D:\Global\Tools\sts-4.12.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.0.v20211012-1059\
  for: List com.global.book.service.PostService.getAllPost()      withArgs: ()    took: 311 ms.
ssor: Using 'application/json', given [*/*] and supported [application/json, application/*+json, application/json, appl
, title=sunt aut facere repellat provident occaecati excepturi optio reprehenderit,  (truncated)...]

={}
lobal.book.controller.PostController#getPostById(Long)
older.typicode.com/posts/1
pplication/cbor, application/*+json]

.entity.PostDto]
  for: PostDto com.global.book.service.PostService.getPostById(Long)      withArgs: (1)   took: 375 ms.
```

- Advice
  - It are actions taken for a particular joint point

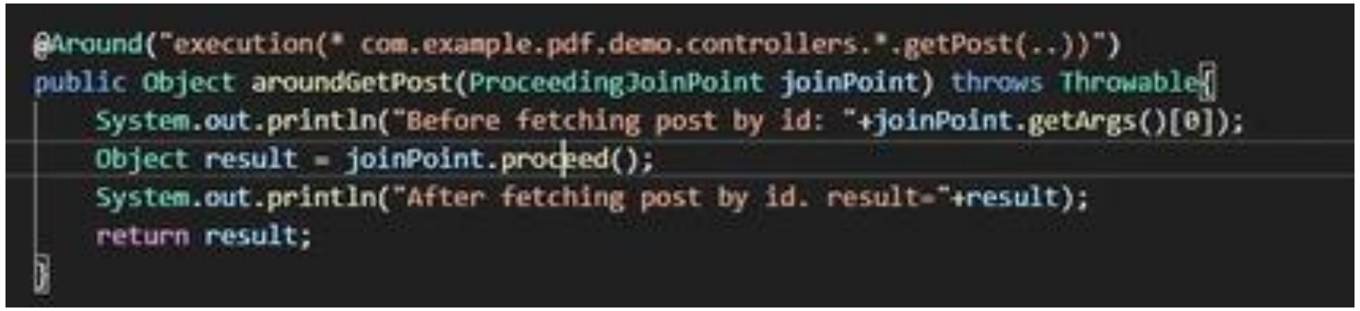
AOP Advice Types

- **Before Advice**: These advices runs before the execution of join point methods. We can use @Before annotation to mark an advice type as Before advice.
- **After (finally) Advice**: An advice that gets executed after the join point method finishes executing, whether normally or by throwing an exception. We can create after advice using @After annotation.
- **After Returning Advice**: Sometimes we want advice methods to execute only if the join point method executes normally. We can use @AfterReturning annotation to mark a method as after returning advice.
- **After Throwing Advice**: This advice gets executed only when join point method throws exception, we can use it to rollback the transaction declaratively. We use @AfterThrowing annotation for this type of advice.
- **Around Advice**: This is the most important and powerful advice. This advice surrounds the join point method and we can also choose whether to execute the join point method or not. We can write advice code that gets executed before and after the execution of the join point method. It is the responsibility of around advice to invoke the join point method and return values if the method is returning something. We use @Around annotation to create around advice methods.

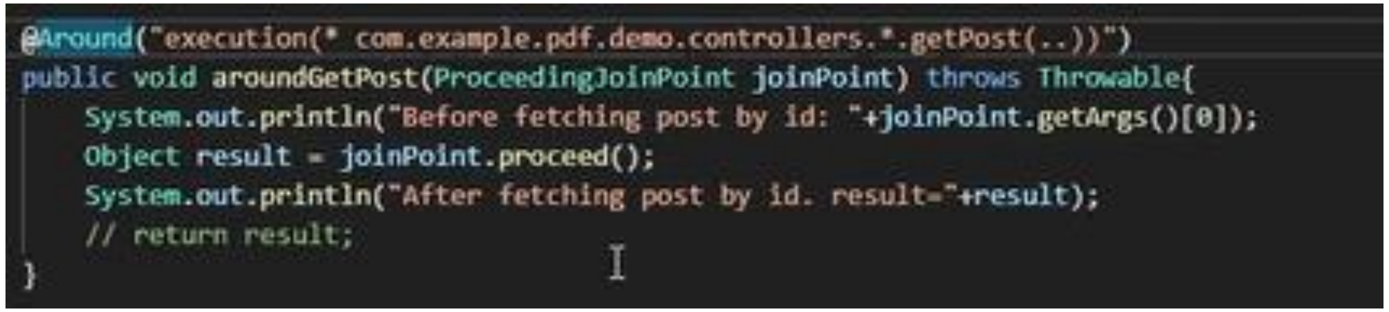
AOP Advice Types



```java
public void beforeGetPosts(JoinPoint joinPoint){
    System.out.println("Before fetching posts: "+joinPoint.getSignature()+", search ="+joinPoint.getArgs()[0]);
}

@After("execution(* com.example.pdf.demo.controllers.*.getPosts(..))")
public void afterGetPosts(JoinPoint joinPoint){
    System.out.println("After fetching posts: "+joinPoint.getSignature()+", search ="+joinPoint.getArgs()[0]);
}

@AfterReturning(pointcut = "execution(* com.example.pdf.demo.controllers.*.getPosts(..))", returning = "result")
public void afterGetPostsReturning(JoinPoint joinPoint, Object result){
    System.out.println("After fetching posts: "+joinPoint.getSignature()+", result ="+result);
}

@AfterThrowing(pointcut = "execution(* com.example.pdf.demo.controllers.*.getPosts(..))", throwing =  "ex")
public void afterGetPostsThrowing(JoinPoint joinPoint, Exception ex){
    System.out.println("Exception thrown while fetching posts: "+joinPoint.getSignature()+", ex ="+ex.getMessage());
}
```



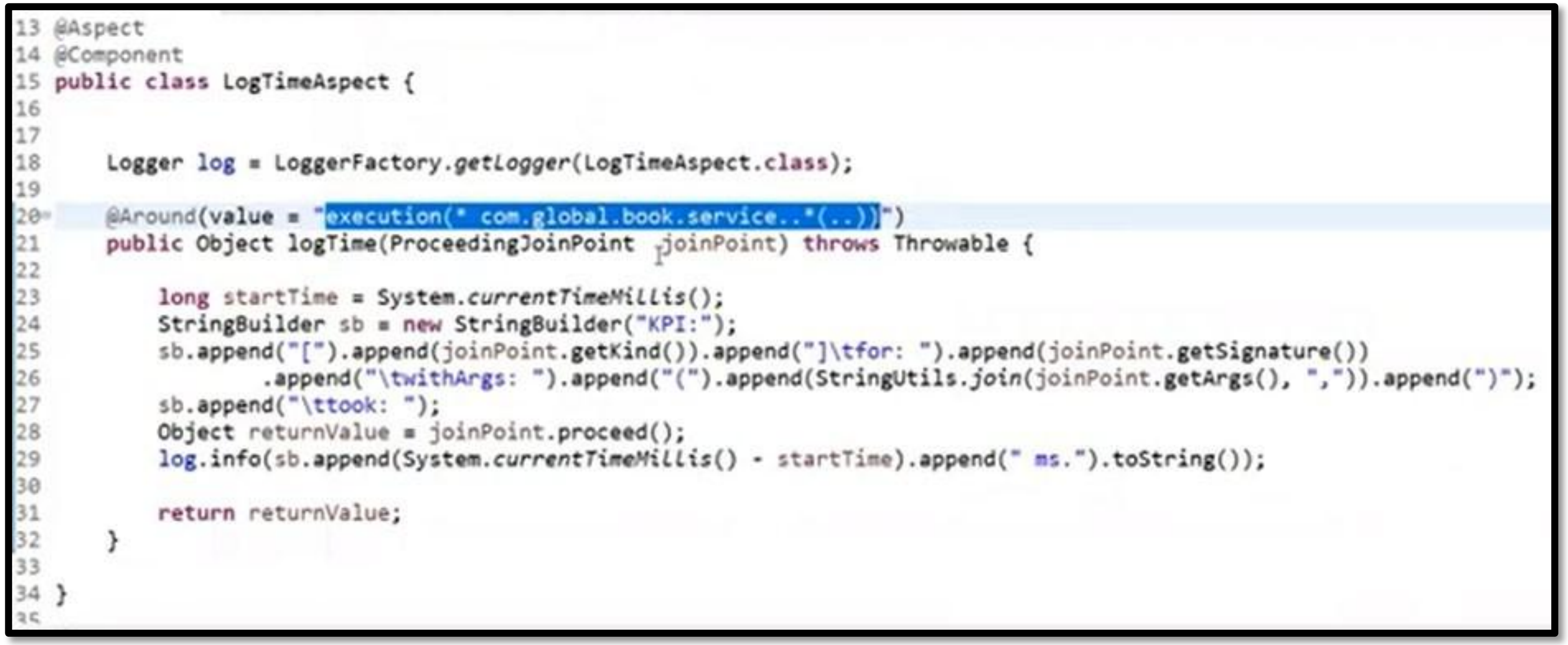
```java
@Around("execution(* com.example.pdf.demo.controllers.*.getPost(..))")
public Object aroundGetPost(ProceedingJoinPoint joinPoint) throws Throwable{
    System.out.println("Before fetching post by id: "+joinPoint.getArgs()[0]);
    Object result = joinPoint.proceed();
    System.out.println("After fetching post by id. result="+result);
    return result;
}
```



```java
@Around("execution(* com.example.pdf.demo.controllers.*.getPost(..))")
public void aroundGetPost(ProceedingJoinPoint joinPoint) throws Throwable{
    System.out.println("Before fetching post by id: "+joinPoint.getArgs()[0]);
    Object result = joinPoint.proceed();
    System.out.println("After fetching post by id. result="+result);
    // return result;
}
```

AOP Terminologies

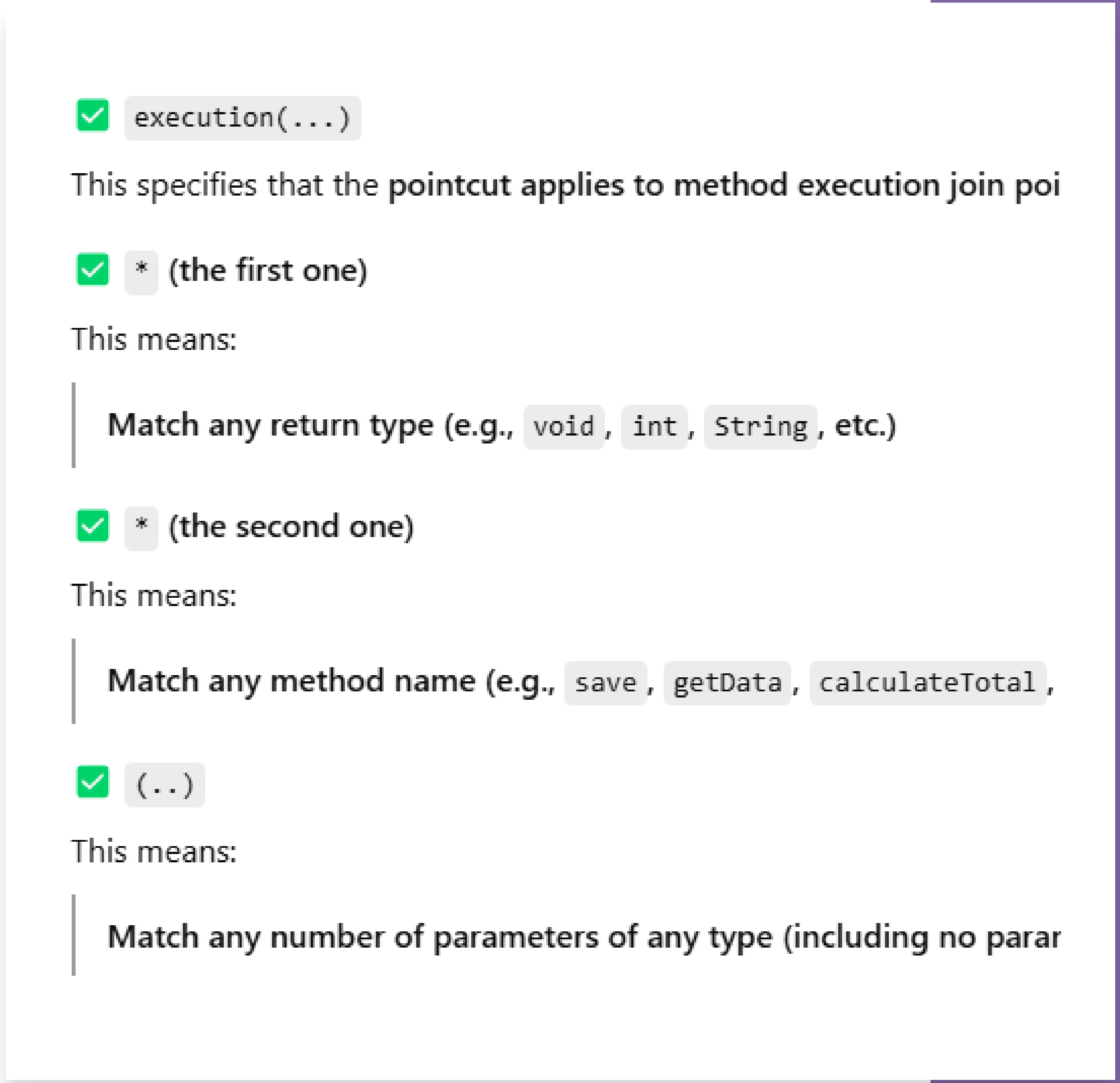
- @Pointcut("execution(* com.tutorialspoint.*.*(..))")
- @Pointcut("execution(* com.tutorialspoint.Student.getName(..))")



```java
@Aspect
@Component
public class LogTimeAspect {


    Logger log = LoggerFactory.getLogger(LogTimeAspect.class);

    @Around(value = "execution(* com.global.book.service..*(..))")
    public Object logTime(ProceedingJoinPoint joinPoint) throws Throwable {

        long startTime = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder("KPI:");
        sb.append("[").append(joinPoint.getKind()).append("]\tfor: ").append(joinPoint.getSignature())
                .append("\twithArgs: ").append("(").append(StringUtils.join(joinPoint.getArgs(), ",")).append(")");
        sb.append("\ttook: ");
        Object returnValue = joinPoint.proceed();
        log.info(sb.append(System.currentTimeMillis() - startTime).append(" ms.").toString());

        return returnValue;
    }

}
```

- Point cut
  - It is expressions that are matched with joint point to determine whether advice needed to be executed or not
  - Identifies the specific events with which advice should be associated.
  - AOP is the ability to multicast the same aspects into many places in your code, this feature is called *Point Cut*

Point Cut



✅ `execution(...)`

This specifies that the **pointcut applies to method execution join poi**

✅ `*` **(the first one)**

This means:

> **Match any return type (e.g.,** `void`, `int`, `String`**, etc.)**

✅ `*` **(the second one)**

This means:

> **Match any method name (e.g.,** `save`, `getData`, `calculateTotal`,

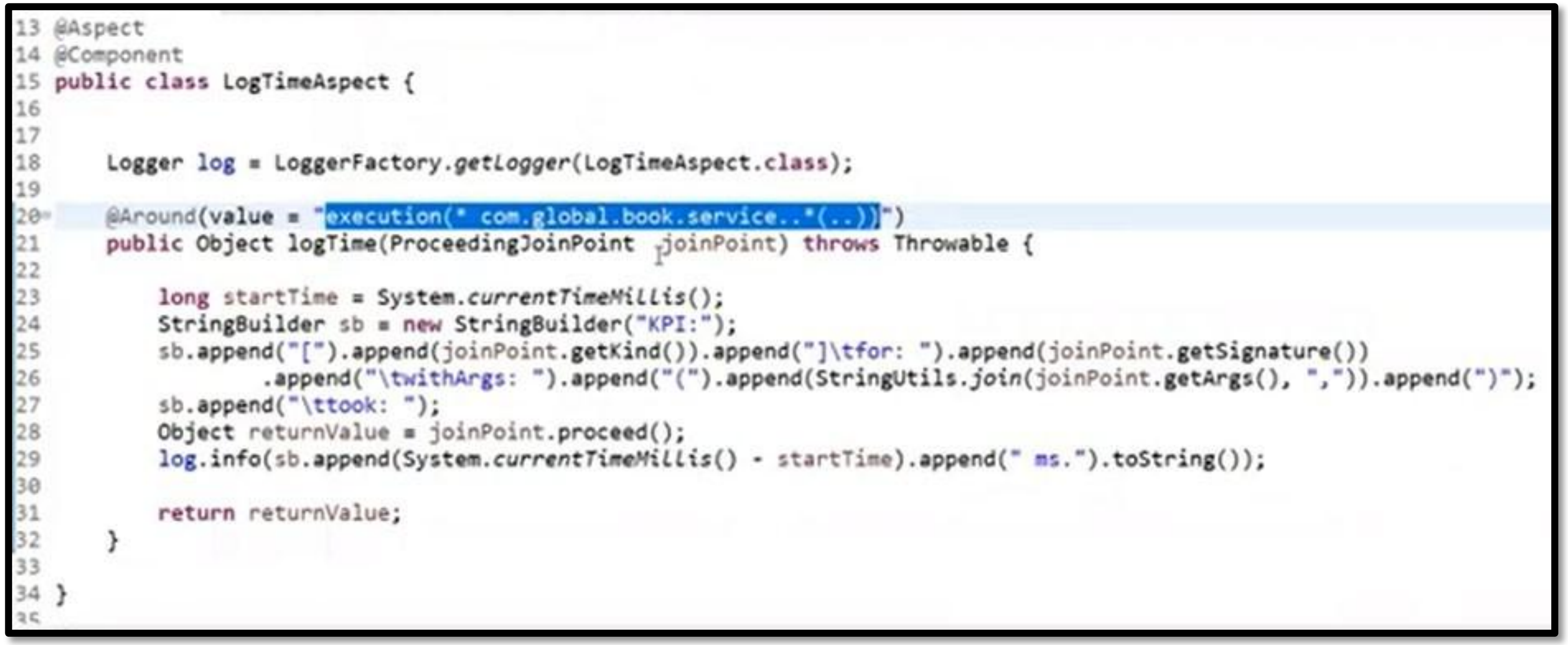
✅ `(..)`

This means:

> **Match any number of parameters of any type (including no parar**

AOP Terminologies

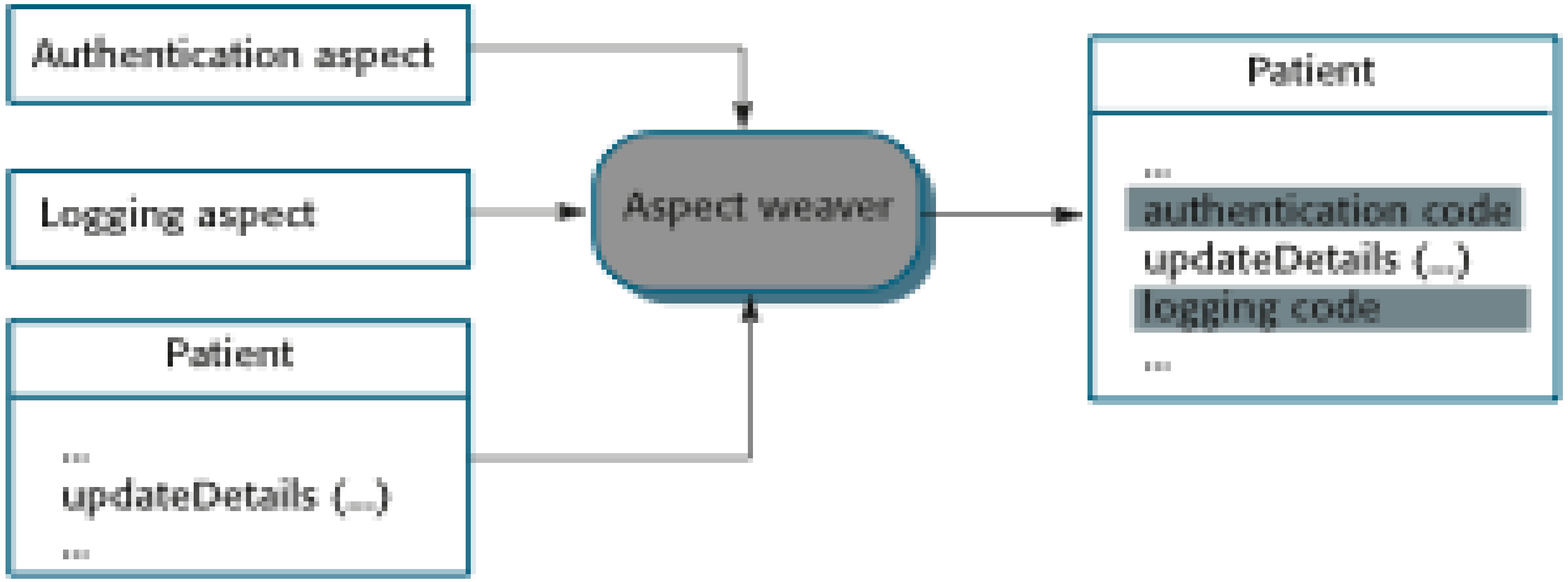
- @Pointcut("execution(* com.tutorialspoint.*.*(..))")
- @Pointcut("execution(* com.tutorialspoint.Student.getName(..))")



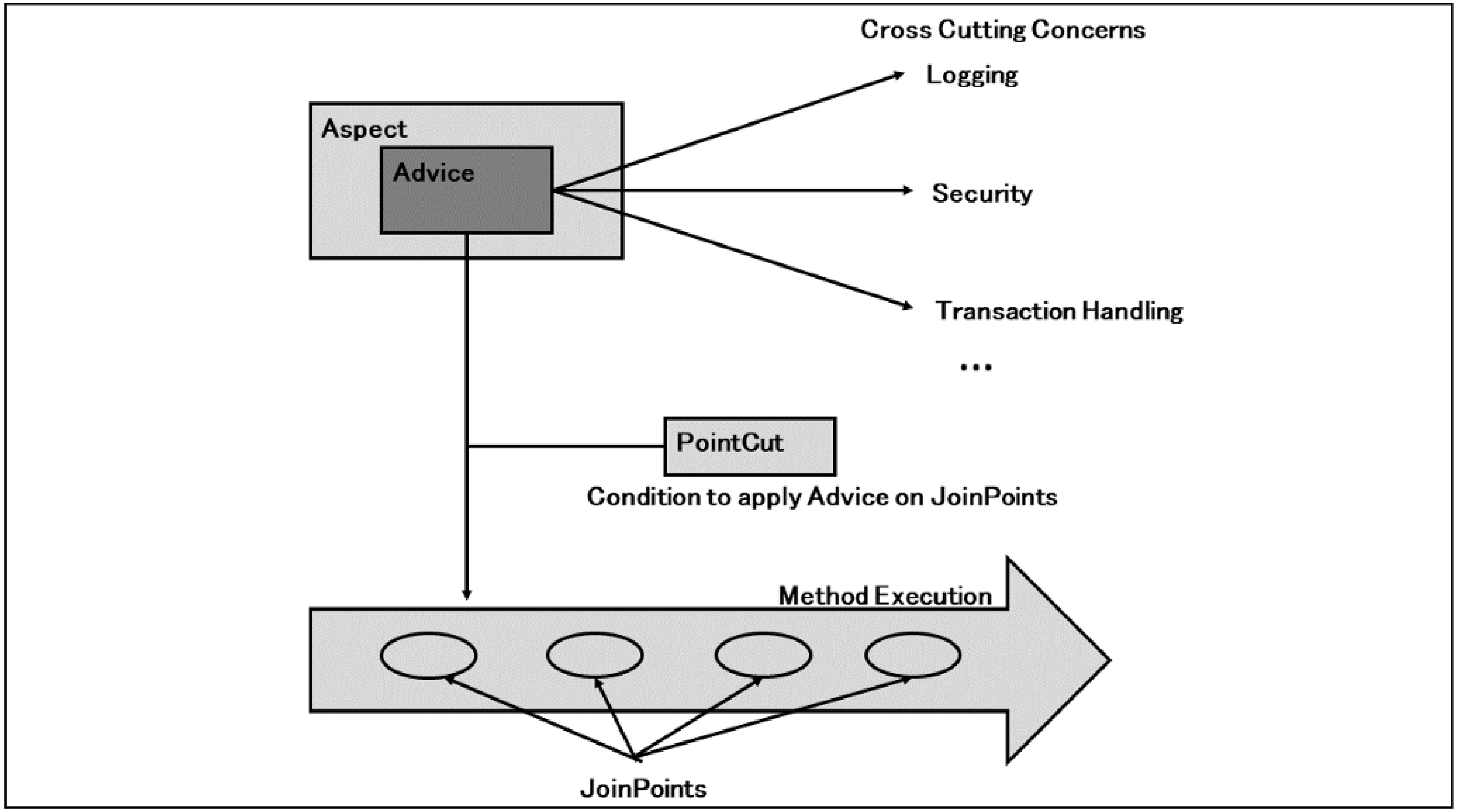
```
@Aspect
@Component
public class LogTimeAspect {


    Logger log = LoggerFactory.getLogger(LogTimeAspect.class);

    @Around(value = "execution(* com.global.book.service..*(..))")
    public Object logTime(ProceedingJoinPoint joinPoint) throws Throwable {

        long startTime = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder("KPI:");
        sb.append("[").append(joinPoint.getKind()).append("]\tfor: ").append(joinPoint.getSignature())
                .append("\twithArgs: ").append("(").append(StringUtils.join(joinPoint.getArgs(), ",")).append(")");
        sb.append("\ttook: ");
        Object returnValue = joinPoint.proceed();
        log.info(sb.append(System.currentTimeMillis() - startTime).append(" ms.").toString());

        return returnValue;
    }

}
```

- Weaving
  - Combines advices with point cuts
  - The incorporation of advice code at the specified join points by an aspect weaver.

Aspect Weaving



AOP Terminologies



TEAM
WORK
!?!

THANK YOU!