Refactoring with AOP

*Aspect Oriented Programming

About Me

Mert Metin

Software Engineer
@Sompo Sigorta

MSc. in Information Technologies
@Yildiz Technical University

Blog -
https://mertmtn.blogspot.com/

Github -
https://github.com/mertmtn

LinkedIn -
https://www.linkedin.com/in/mrtmtn/

Twitter -
https://twitter.com/_mertmetin

Agenda

1. What is AOP?
2. Some AOP Terminogies
3. Interceptors
4. Refactoring
5. Sample Codes
6. Benefits of AOP

What is AOP - Aspect Oriented Programming

AOP is an approach that separates functional and common non functional codes.

Functional codes perform main business logic of an application.

Non functional codes are cross cutting concerns which can be used everywhere or every layer are meaningful as an individual module.

In other words; increases modularity by allowing the separation of cross-cutting concerns.

Some AOP Terminologies

Cross Cutting Concern

Aspects of a program that affect other concerns. E.g. Validation, Logging, Caching, Exception, etc.

Join point

A candidate point in the program execution where an aspect can be plugged in.
E.g. OnBefore, OnAfter, OnSuccess, OnException

Aspect

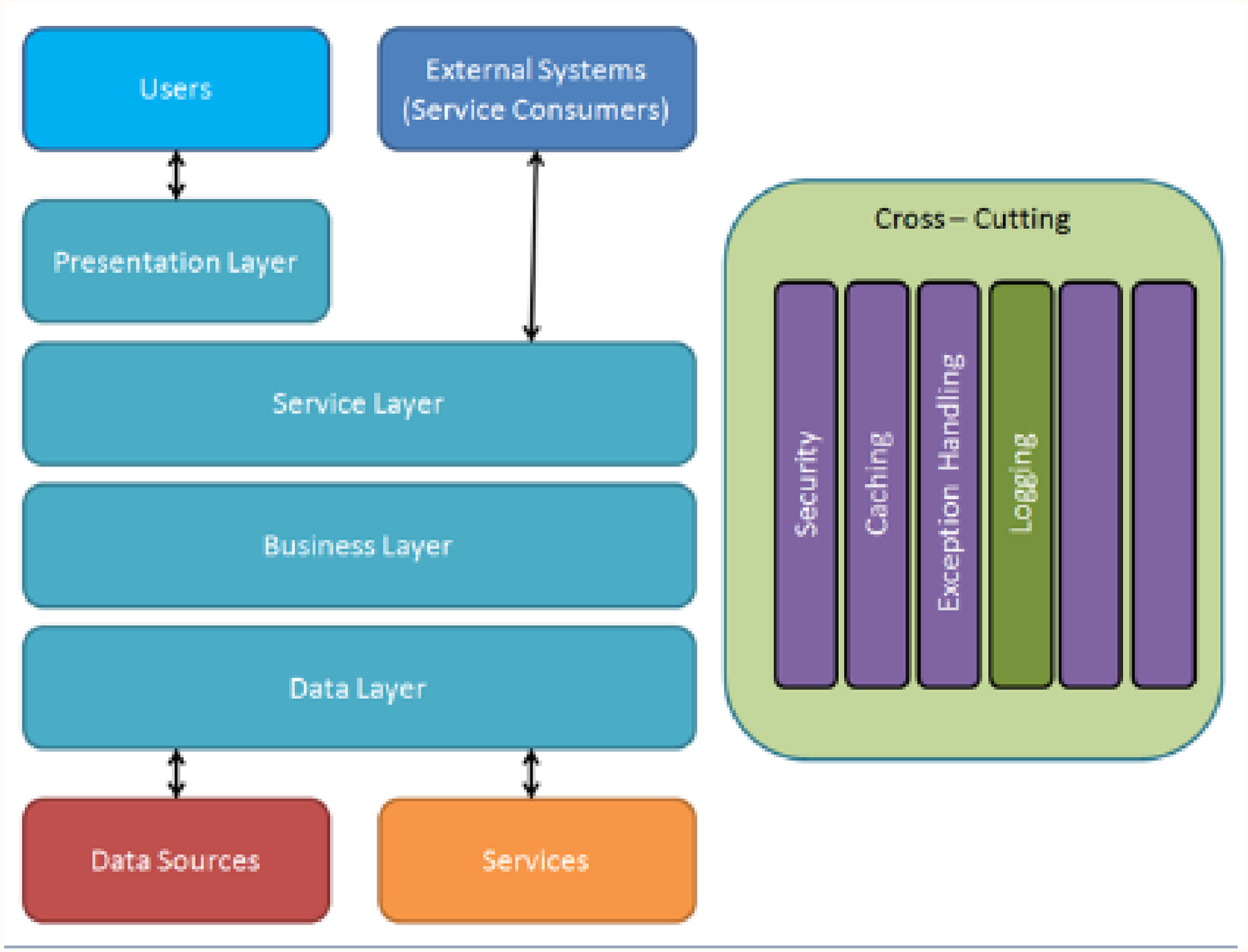
Implemented the concern as programmatically e.g: ValidationAspect, ExceptionAspect

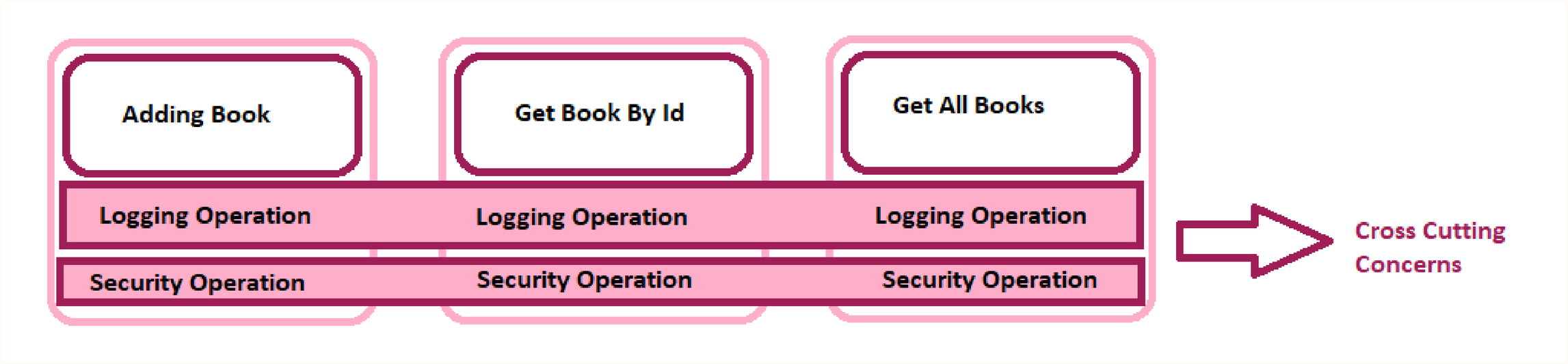
Advice

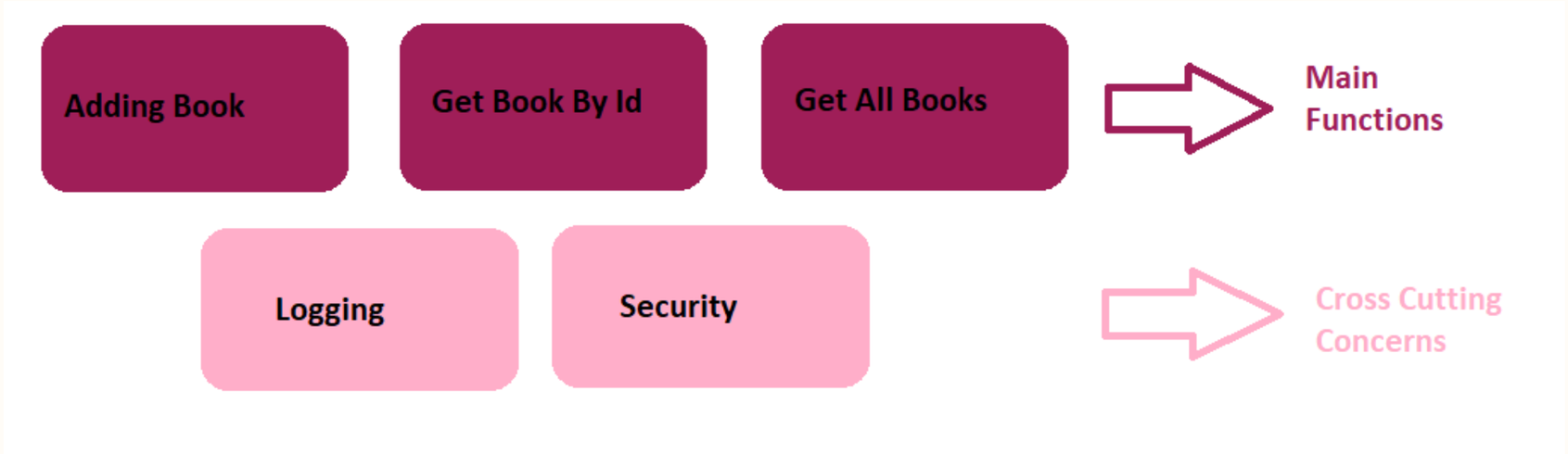
Additional logic or code called from a join point.

Pointcut

Which method does executes on join point? E.g: Logging Method


Users
External Systems
(Service Consumers)
Presentation Layer
Service Layer
Business Layer
Data Layer
Data Sources
Services
Cross – Cutting
Security
Caching
Exception Handling
Logging


Adding Book
Get Book By Id
Get All Books
Logging Operation
Logging Operation
Logging Operation
Security Operation
Security Operation
Security Operation
Cross Cutting Concerns


Adding Book
Get Book By Id
Get All Books
Main Functions
Logging
Security
Cross Cutting Concerns

Interceptors

If you would like to add extra functionality when method execution any where, you need interceptors.

They allow inject logic on before, after, exception, success the method execution.

Most frequently uses by languages/frameworks

- .NET: Postsharp-Autofac
- Java: Spring-AspectJ-JBoss
- PHP: Go AOP

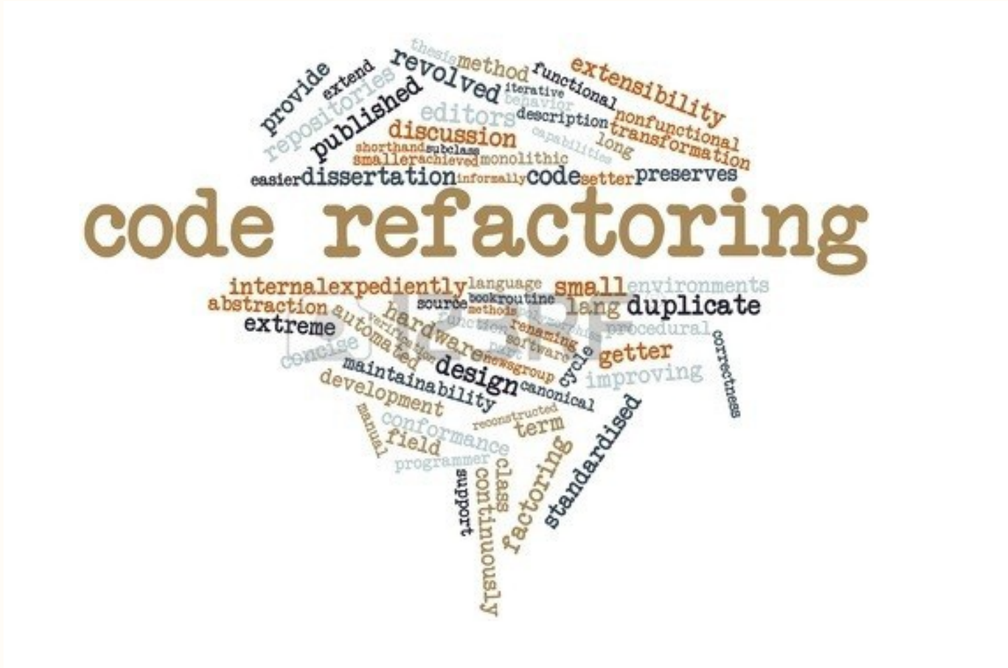
```
public abstract class MethodInterception: MethodInterceptionBaseAttribute
{
    2 references
    protected virtual void OnBefore(IInvocation invocation) { }
    1 reference
    protected virtual void OnAfter(IInvocation invocation) { }
    4 references
    protected virtual void OnException(IInvocation invocation, System.Exception e) { }
    1 reference
    protected virtual void OnSuccess(IInvocation invocation) { }

    4 references
    public override void Intercept(IInvocation invocation)
    {
        var isSuccess = true;
        OnBefore(invocation);
        try
        {
            invocation.Proceed();
        }
        catch (Exception e)
        {
            isSuccess = false;
            OnException(invocation, e);
            throw;
        }
        finally
        {
            if (isSuccess)
            {
                OnSuccess(invocation);
            }
        }
        OnAfter(invocation);
    }
}
```

Each aspect inherits MethodInterception class and override its methods

Refactoring

A systematic process of improving code without creating new functionality that can transform a mess into clean code and simple design.



Before Refactoring

Does this method satisfy

- Single Responsibility Principle
- Separation of Concern

```
public IDataResult<List<Department>> GetAllDepartment()
{
    _logger.LogDebug("Inside GetAllDepartment endpoint");

    const string key = "GetAllDepartment";
    if (_memoryCache.TryGetValue(key, out object list))
    {
        _logger.LogDebug("Inside GetAllDepartment endpoint cached");
        var cachedList = (List<Department>)list;
        return new SuccessDataResult<List<Department>>(cachedList);
    }

    var departmentList = _departmentDal.GetAll();

    _memoryCache.Set(key, departmentList, new MemoryCacheEntryOptions
    {
        AbsoluteExpiration = DateTime.Now.AddSeconds(20),
        Priority = CacheItemPriority.Normal
    });
    return new SuccessDataResult<List<Department>>(_departmentDal.GetAll());
}
```

Before Refactoring

```
public IResult UpdateDepartment(Department department)
{
    _logger.LogDebug("Inside UpdateDepartment");

    var validator = new DepartmentValidator();
    validator.Validate(department);

    _departmentDal.Update(department);
    return new SuccessResult("Update Department Success");
}
```

Sample Codes

After Refactoring

```
[LoggingAspect()]
[ValidationAspect(typeof(DepartmentValidator))]
2 references
public IResult UpdateDepartment(Department department)
{
    _departmentDal.Update(department);
    return new SuccessResult("Update Department Success");
}
```

After Refactoring

```
[LoggingAspect()]
[CachingAspect(60)]
2 references
public IDataResult<List<Department>> GetAllDepartment()
{
    var departmentList = _departmentDal.GetAll();

    return new SuccessDataResult<List<Department>>(_departmentDal.GetAll());
}
```

Benefits of AOP

Adhering to important software principles.

- Single Responsibility Principle (SRP)
- Do Not Repeat Yourself (DRY)
- Separation of Concerns

Readability

Modularity

Maintability

Reusability

```
public IResult UpdateDepartment(Department department)
{
    _logger.LogDebug("Inside UpdateDepartment");

    var validator = new DepartmentValidator();
    validator.Validate(department);

    _departmentDal.Update(department);
    return new SuccessResult("Update Department Success");
}
```

```
[LoggingAspect()]
[ValidationAspect(typeof(DepartmentValidator))]
2 references
public IResult UpdateDepartment(Department department)
{
    _departmentDal.Update(department);
    return new SuccessResult("Update Department Success");
}
```

```
public IDataResult<List<Department>> GetAllDepartment()
{
    _logger.LogDebug("Inside GetAllDepartment endpoint");

    const string key = "GetAllDepartment";
    if (_memoryCache.TryGetValue(key, out object list))
    {
        _logger.LogDebug("Inside GetAllDepartment endpoint cached");
        var cachedList = (List<Department>)list;
        return new SuccessDataResult<List<Department>>(cachedList);
    }

    var departmentList = _departmentDal.GetAll();

    _memoryCache.Set(key, departmentList, new MemoryCacheEntryOptions
    {
        AbsoluteExpiration = DateTime.Now.AddSeconds(20),
        Priority = CacheItemPriority.Normal
    });
    return new SuccessDataResult<List<Department>>(_departmentDal.GetAll());
}
```

```
[LoggingAspect()]
[CachingAspect(60)]
2 references
public IDataResult<List<Department>> GetAllDepartment()
{
    var departmentList = _departmentDal.GetAll();

    return new SuccessDataResult<List<Department>>(_departmentDal.GetAll());
}
```

Thanks all for listening!