# WZORZEC DEKORATOR

MICHAŁ ŻDANUK
INFORMATYKA STOSOWANA, WYDZIAŁ ELEKTRYCZNY, POLITECHNIKA WARSZAWSKA

Prezentacja zrealizowana na zajęcia z przedmiotu „Projektowanie oprogramowania z użyciem wzorców”

*14.10.2024*

# PROBLEMY PRZY ROZWIJANIU OPROGRAMOWANIA

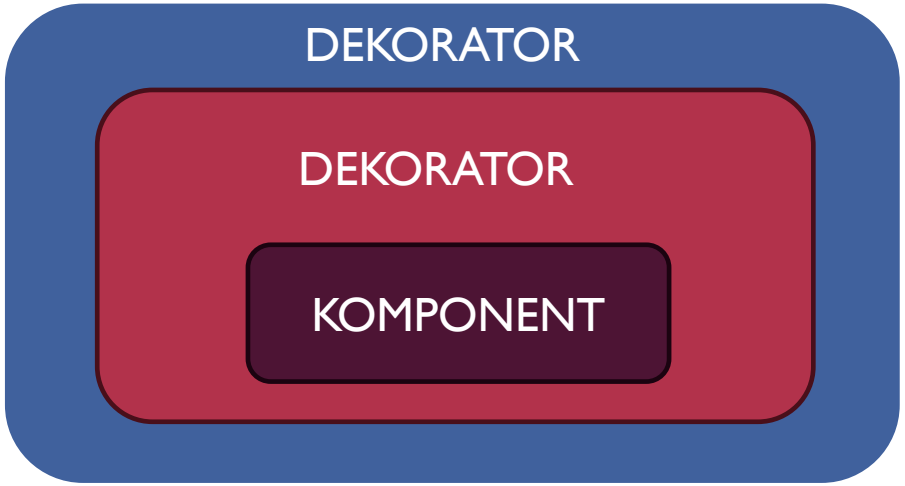
Posiadamy już zaimplementowaną klasę/serwis, która jest w pełni funkcjonalna, ale:

- przychodzącą nam nowe wymagania i musimy rozszerzyć zachowanie (**klasa zaczyna rosnąć i puchnąć, łamiemy Open-Close principle – kod otwarty na rozszerzenia, ale zamknięty na modyfikacje**)

# POMYSŁ ROZWIĄZANIA PROBLEMU

**Wzorzec dekorator** – strukturalny wzorzec pozwalający na dodawanie obiektom nowych obowiązków w sposób dynamiczny poprzez „opakowanie" istniejących obiektów w specjalne obiekty spełniającą inną funkcjonalność.

Dekorować możemy wielokrotnie (układając je łańcuchowo) i kolejność wywołań jest odwrotna niż ich rejestracja (*ang. LIFO – Last In First Out*).



# PRZYKŁAD

Mamy gotowy, prosty serwis, pobierający wideo z bazy danych:

```
2 references
public class MovieService : IMovieService
{
    private readonly IMovieRepository _movieRepository;
    private readonly IMapper _mapper;

    0 references
    public MovieService(IMovieRepository movieRepository,
        IMapper mapper)
    {
        _movieRepository = movieRepository;
        _mapper = mapper;
    }

    2 references
    public async Task<Guid> CreateAsync(CreateMovieDto dto)...

    2 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        var movie = await _movieRepository.GetAsync(id);

        var movieDto = _mapper.Map<MovieDto>(movie);

        return movieDto;
    }
}
```

```
namespace MoviesAPI.Services
{
    4 references
    public interface IMovieService
    {
        2 references
        Task<Guid> CreateAsync(CreateMovieDto dto);
        2 references
        Task<MovieDto?> GetByIdAsync(Guid id);
    }
}
```

# SERWIS PO ROZSZERZENIU GO O CACHE'OWANIE

```
2 references
public class MovieService : IMovieService
{
    private readonly IMovieRepository _movieRepository;
    private readonly IMapper _mapper;
    private readonly IMemoryCache _cache;
    private readonly TimeSpan _cacheDuration = TimeSpan.FromMinutes(5);

    0 references
    public MovieService(IMovieRepository movieRepository,
        IMapper mapper,
        IMemoryCache cache)
    {
        _movieRepository = movieRepository;
        _mapper = mapper;
        _cache = cache;
    }

    2 references
    public async Task<Guid> CreateAsync(CreateMovieDto dto)...

    2 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        var dto = _cache.Get<MovieDto>($"movies:{id}");

        if(dto is not null)
        {
            return dto;
        }

        var movie = await _movieRepository.GetAsync(id);

        if(movie is null)
        {
            return null;
        }

        var movieDto = _mapper.Map<MovieDto>(movie);

        _cache.Set($"movies:{id}", movieDto, _cacheDuration);

        return movieDto;
    }
}
```

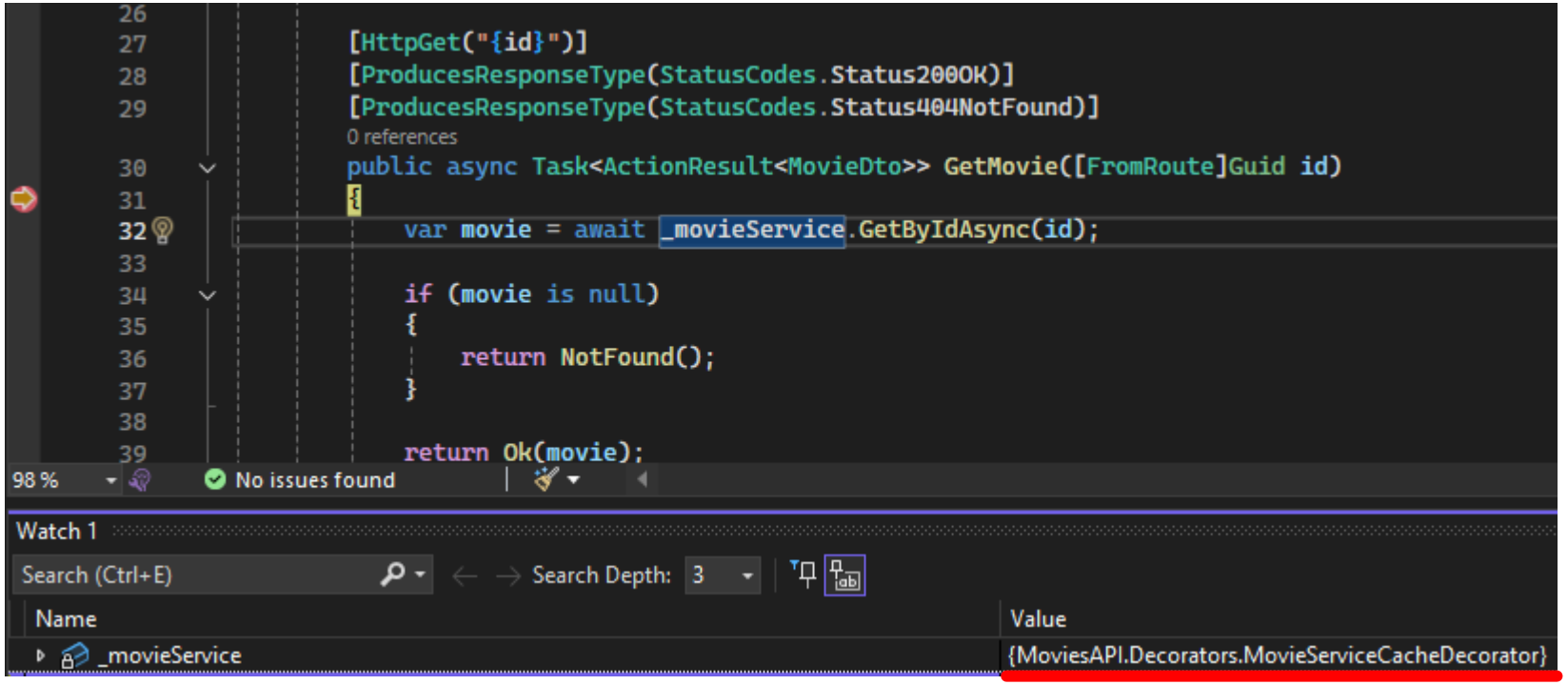
# DODANIE DEKORATORA CACHE

```
public class MovieServiceCacheDecorator : IMovieService
{
    private readonly MovieService _movieService;
    private readonly IMemoryCache _cache;

    0 references
    public MovieServiceCacheDecorator(MovieService movieService,
        IMemoryCache cache)
    {
        _movieService = movieService;
        _cache = cache;
    }

    2 references
    public Task<Guid> CreateAsync(CreateMovieDto dto) => _movieService.CreateAsync(dto);

    2 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        var cacheKey = $"movies:{id}";
        var movie = _cache.Get<MovieDto>(cacheKey);
        if(movie is null)
        {
            movie = await _movieService.GetByIdAsync(id);

            if( movie is not null)
            {
                _cache.Set(cacheKey, movie);
            }
        }

        return movie;
    }
}
```

```
public class MovieService : IMovieService
{
    private readonly IMovieRepository _movieRepository;
    private readonly IMapper _mapper;

    0 references
    public MovieService(IMovieRepository movieRepository,
        IMapper mapper)
    {
        _movieRepository = movieRepository;
        _mapper = mapper;
    }

    3 references
    public async Task<Guid> CreateAsync(CreateMovieDto dto)...

    3 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        var movie = await _movieRepository.GetAsync(id);

        var movieDto = _mapper.Map<MovieDto>(movie);

        return movieDto;
    }
}
```

```
builder.Services.AddScoped<MovieService>();
builder.Services.AddScoped<IMovieService, MovieServiceCacheDecorator>();
```

# DODANIE DEKORATORA CACHE (2)



# DODANIE DEKORATORA CACHE (3)

```
7   public class MovieServiceCacheDecorator : IMovieService
8   {
9       private readonly MovieService _movieService;
10      private readonly IMemoryCache _cache;
11
        0 references
12      public MovieServiceCacheDecorator(MovieService movieService,
13          IMemoryCache cache)
14      {
15          _movieService = movieService;
16          _cache = cache;
17      }
18
        2 references
19      public Task<Guid> CreateAsync(CreateMovieDto dto) => _movieService.CreateAsync(dto);
20
        2 references
21      public async Task<MovieDto?> GetByIdAsync(Guid id)
22      {  ≤ 2ms elapsed
23          var cacheKey = $"movies:{id}";
24          var movie = _cache.Get<MovieDto>(cacheKey);
25          if(movie is null)
26          {
27              movie = await _movieService.GetByIdAsync(id);
28
29              if( movie is not null)
30              {
31                  _cache.Set(cacheKey, movie);
32              }
33          }
34
35          return movie;
36      }
37  }
38  }
```

98 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

| Name | Value |
| --- | --- |
| _movieService | {MoviesAPI.Services.MovieService} |

```
public class MovieService : IMovieService
{
    private readonly IMovieRepository _movieRepository;
    private readonly IMapper _mapper;

    0 references
    public MovieService(IMovieRepository movieRepository,
        IMapper mapper)
    {
        _movieRepository = movieRepository;
        _mapper = mapper;
    }

    3 references
    public async Task<Guid> CreateAsync(CreateMovieDto dto)...

    3 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {  ≤ 2ms elapsed
        var movie = await _movieRepository.GetAsync(id);

        var movieDto = _mapper.Map<MovieDto>(movie);

        return movieDto;
    }
}
```

# DODANIE DRUGIEGO DEKORATORA (LOGUJĄCEGO)

```
public class MovieServiceLoggingDecorator : IMovieService
{
    private readonly MovieServiceCacheDecorator _service;
    private readonly ILogger<MovieServiceLoggingDecorator> _logger;

    0 references
    public MovieServiceLoggingDecorator(MovieServiceCacheDecorator service,
        ILogger<MovieServiceLoggingDecorator> logger)
    {
        _service = service;
        _logger = logger;
    }

    2 references
    public Task<Guid> CreateAsync(CreateMovieDto dto) => _service.CreateAsync(dto);

    2 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        _logger.LogInformation($"Fetching movie with id: {id}...");
        var movie = await _service.GetByIdAsync(id);
        _logger.LogInformation($"Movie with id: {id} was {(movie is null ? "not" : "")} found.");

        return movie;
    }
}
```

```
builder.Services.AddScoped<MovieService>();
builder.Services.AddScoped<MovieServiceCacheDecorator>();
builder.Services.AddScoped<IMovieService, MovieServiceLoggingDecorator>();
```

# REJESTRACJA ZALEŻNOŚCI

- Manualna (trzeba pamiętać o odpowiedniej kolejności i pilnować co gdzie wstrzykujemy w konstruktorach)

```
builder.Services.AddScoped<MovieService>();
builder.Services.AddScoped<MovieServiceCacheDecorator>();
builder.Services.AddScoped<IMovieService, MovieServiceLoggingDecorator>();
```

- Automatyczna (przykładowo wykorzystany pakiet Scrutor) – w tym przypadku we wszystkich implementacjach serwisu wstrzykujemy zawsze abstrakcję (interfejs), a kontener IoC wstrzyknie nam automatycznie po kolei implementacje
  kolejność wywołań jest odwrotna do dekoracji

```
builder.Services.AddScoped<IMovieService, MovieService>();
builder.Services.Decorate<IMovieService, MovieServiceCacheDecorator>();
builder.Services.Decorate<IMovieService, MovieServiceLoggingDecorator>();
```

# DEKORATORY PRZY WYKORZYSTANIU AUTOMATYCZNEGO WSTRZYKIWANIA

```csharp
public class MovieServiceCacheDecorator : IMovieService
{
    private readonly IMovieService _movieService;
    private readonly IMemoryCache _cache;

    0 references
    public MovieServiceCacheDecorator(IMovieService movieService,
        IMemoryCache cache)
    {
        _movieService = movieService;
        _cache = cache;
    }

    4 references
    public Task<Guid> CreateAsync(CreateMovieDto dto) => _movieService.CreateAsync(dto);

    4 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        var cacheKey = $"movies:{id}";
        var movie = _cache.Get<MovieDto>(cacheKey);
        if(movie is null)
        {
            movie = await _movieService.GetByIdAsync(id);

            if( movie is not null)
            {
                _cache.Set(cacheKey, movie);
            }
        }

        return movie;
    }
}
```

```csharp
public class MovieServiceLoggingDecorator : IMovieService
{
    private readonly IMovieService _service;
    private readonly ILogger<MovieServiceLoggingDecorator> _logger;

    0 references
    public MovieServiceLoggingDecorator(IMovieService service,
        ILogger<MovieServiceLoggingDecorator> logger)
    {
        _service = service;
        _logger = logger;
    }

    4 references
    public Task<Guid> CreateAsync(CreateMovieDto dto) => _service.CreateAsync(dto);

    4 references
    public async Task<MovieDto?> GetByIdAsync(Guid id)
    {
        _logger.LogInformation($"Fetching movie with id: {id}...");
        var movie = await _service.GetByIdAsync(id);
        _logger.LogInformation($"Movie with id: {id} was {(movie is null ? "not" : "")} found.");

        return movie;
    }
}
```

# DZIĘKUJĘ ZA UWAGĘ