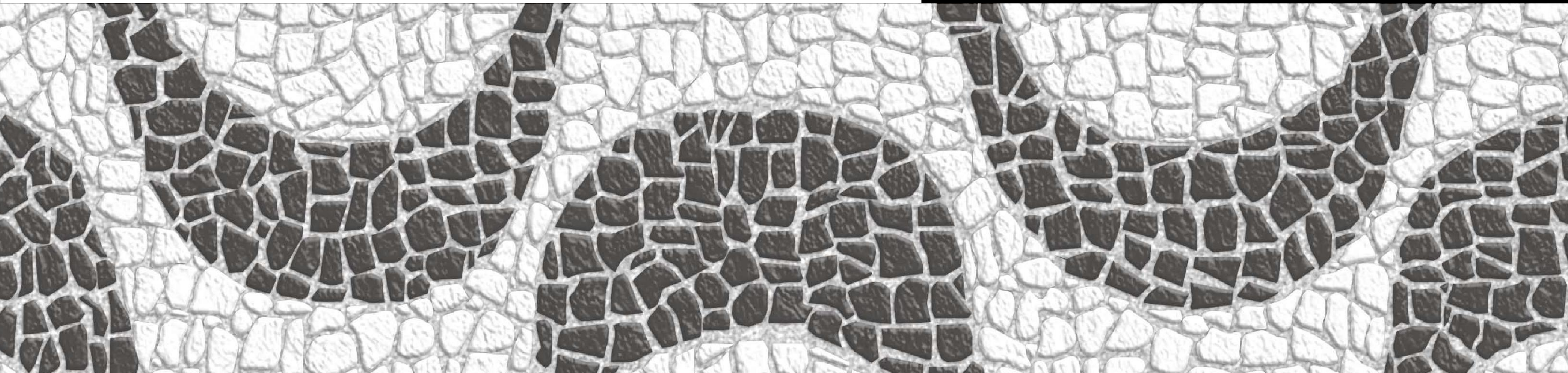
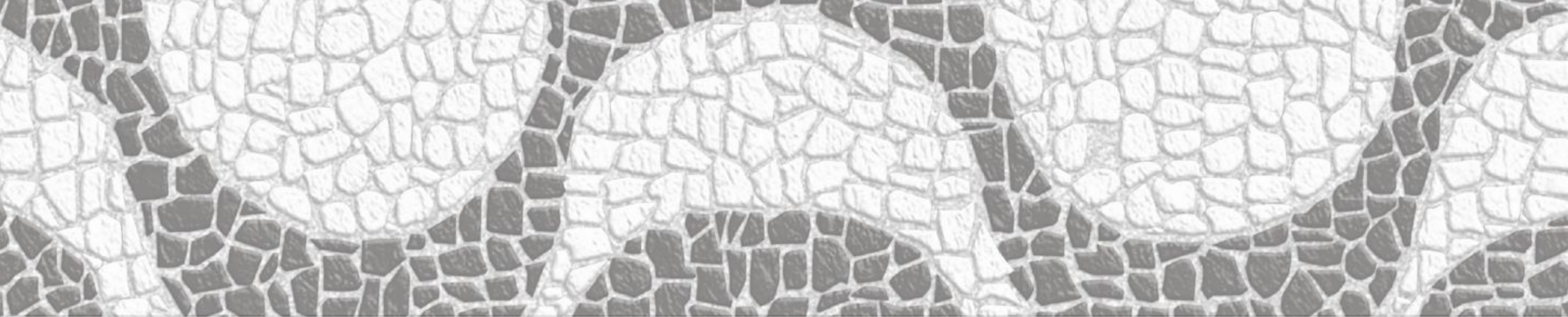


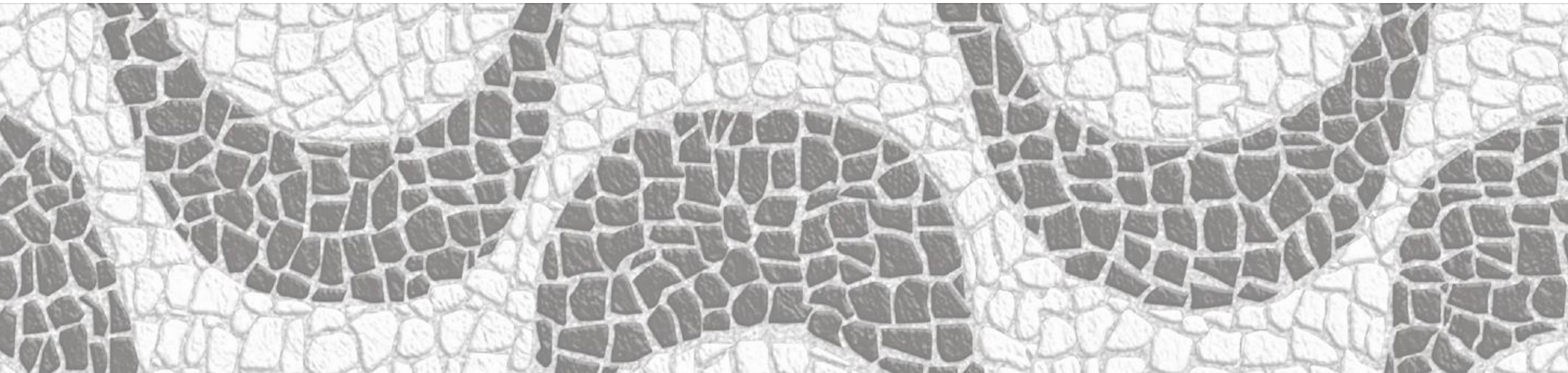
<Coders> in

</Rio>





Design Patterns no dia a dia



PALESTRANTE



Vitor Fitzner

- Bacharel e pós graduado em engenharia de software.
- Desenvolvedor Web há 8 anos.
- Apaixonado por leitura e códigos.
- Começando um blog para compartilhar ideias e conhecimento com a comunidade de desenvolvedores.
- Violão, Bateria e Dota > Lol

O que são padrões de projeto?

- Soluções reutilizáveis para problemas comuns no projeto de software orientados a objetos.
- Foco no relacionamento entre classes.
- Independentes de tecnologia.
- Não são a solução final.
- Não são inventados, são descobertos.
- Não são algoritmos.
- Não são implementações de classes específicas.

Características dos padrões de projeto

- O nome do padrão: Ter um vocabulário nos permite falar sobre eles em um nível mais alto.
- O problema: Descreve a situação em que se aplica o padrão.
- A solução: Descreve os elementos que compõem o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações.
- As consequências: Resultados e análises das vantagens e desvantagens (trade-offs) da aplicação do padrão.

Organização dos padrões de projeto

- Creational Patterns:
Simple Factory, Abstract Factory, Builder, Factory Method, Prototype, Singleton,
Lazy Instantiation, Utility Pattern.
- Classificações:
Creational, Structural, Behavioral, Security, Concurrency, UI, Distributed, etc.

História dos padrões de projeto

- A Pattern Language: Towns, Buildings, Construction, 1977
Christopher Alexander, arquiteto, mas não de software.
- Kent Back e Ward Cunningham apresentaram padrões na conferência OOPSLA em 1987
- Design-Patterns: Elements of Reusable Object-Oriented Software, 1994.
Gang of four: Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

... padrões de projeto?

- Cara, cadê o código?

[TestMethod]

✓ | 0 references

```
public void ObterValorDoPedido()
{
    var pedido = new Pedido();
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 2 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });

    var valorEsperado = 500;

    Assert.AreEqual(valorEsperado, pedido.Valor());
}
```


[TestMethod]

✓ | 0 references

```
public void ObterValorDoPedidoComDescontoDe10PorCentoParaFuncionario()
{
    var pedido = new Pedido();

    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 2 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });

    pedido.Desconto = Desconto.Funcionario;

    var valorEsperado = 450;

    Assert.AreEqual(valorEsperado, pedido.Valor());
}
```


[TestMethod]

✓ | 0 references

```
public void ObterValorDoPedidoComDescontoDe20PorCento()
{
    var pedido = new Pedido();

    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 2 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });

    pedido.Desconto = Desconto.VintePorCento;

    var valorEsperado = 400;

    Assert.AreEqual(valorEsperado, pedido.Valor());
}
```


0 references

```
public class Pedido
{
    1 reference
    public IList<ItemDePedido> Itens { get; set; } = new List<ItemDePedido>();
    1 reference
    public Desconto Desconto { get; set; } = Desconto.None;
    0 references
    public virtual decimal Valor()
    {
        var valor = Itens.Sum(x => x.Valor());

        switch (Desconto)
        {
            case Desconto.Funcionario: return valor * 0.9m;
            case Desconto.None: return valor;
            case Desconto.VintePorCento: return valor * 0.8m;
        }

        return valor;
    }
}
```


Dia a dia na vida do programador

```
2 references
public class ItemDePedido
{
    0 references
    public string Produto { get; set; }
    1 reference
    public int Quantidade { get; set; }
    1 reference
    public decimal Preco { get; set; }

    1 reference
    public virtual decimal Valor()
    {
        return Preco * Quantidade;
    }
}
```


Dia a dia na vida do programador

```
5 references  
public enum Desconto  
{  
    None,  
    Funcionario,  
    VintePorCento  
}
```


Aí vem uma mudança

- Boss: Jovem, precisamos adicionar um novo tipo de desconto. Estamos começando com uma campanha que será lançada na sexta-feira da semana que vem. (não existe isso na vida real)
- Você: Tá blz! Como vai ser?
- Boss: Vamos calcular o desconto baseado em pontuação. Para cada ponto, será aplicado um desconto acumulativo de R\$ 0,50.
- Você: Ok!

Aí vem uma mudança

Quem já pensou em como seria uma solução para isso?

Aí vem a mudança

```
7 references  
public enum Desconto  
{  
    None,  
    Funcionario,  
    VintePorCento,  
    PorPontos  
}
```


Aí vem a mudança

```
7 references  
public enum Desconto  
{  
    None,  
    Funcionario,  
    VintePorCento,  
    PorPontos  
}
```



[TestMethod]

✓ | 0 references

```
public void ObterValorDoPedidoComDescontoPorPontos()
{
    var pedido = new Pedido();

    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 2 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });

    pedido.Desconto = Desconto.PorPontos;
    pedido.PontosDeDesconto = 10;

    var valorEsperado = 495;

    Assert.AreEqual(valorEsperado, pedido.Valor());
}
```


4 references | 4/4 passing

```
public virtual decimal Valor()
{
    var valor = Itens.Sum(x => x.Valor());

    switch (Desconto)
    {
        case Desconto.None: return valor;
        case Desconto.Funcionario: return valor * 0.9m;
        case Desconto.VintePorCento: return valor * 0.8m;
        case Desconto.PorPontos: return valor - (PontosDeDesconto * 0.50m);
    }

    return valor;
}
```

Algumas provocações

- Quanto a classe Produto vai crescer em função dos algoritmos de calculo de desconto?

Algumas provocações

- Quanto a classe Produto vai crescer em função dos algoritmos de calculo de desconto?
- Por que a classe Produto tem que ser alterada por causa do algoritmo de desconto?

Algumas provocações

- Quanto a classe Produto vai crescer em função dos algoritmos de calculo de desconto?
- Por que a classe Produto tem que ser alterada por causa do algoritmo de desconto?
- Estamos ferindo o OCP?

Algumas provocações

- Quanto a classe Produto vai crescer em função dos algoritmos de calculo de desconto?
- Por que a classe Produto tem que ser alterada por causa do algoritmo de desconto?
- Estamos ferindo o OCP?
- Se o requisito é adicionar um novo cálculo por que estou alterando o código do que está funcionando?

Algumas provocações

- Quanto a classe Produto vai crescer em função dos algoritmos de calculo de desconto?
- Por que a classe Produto tem que ser alterada por causa do algoritmo de desconto?
- Estamos ferindo o OCP?
- Se o requisito é adicionar um novo cálculo por que estou alterando o código do que está funcionando?
- Algo sobre Merge, Deploy, Mappers, Testes?

Refatorando para um padrão

- Em qual classificação de padrão (GoF) esse problema estaria classificado?
Criação, comportamental, estrutural?

Refatorando para um padrão

Strategy Pattern

- Definir uma família de algoritmos, encapsular e torná-las intercambiáveis.

Aplicabilidade (by the book):

- Você necessita de variantes de um algoritmo.
- Um algoritmo usa dados dos quais os clientes não deveriam ter conhecimento.
- Uma classe define muitos comportamentos e estes aparecem em suas operações como múltiplos comandos condicionais da linguagem.

Aplicabilidade (by blogs, papers, etc):

- Viu um switch pra alterar algoritmo?

Refatorando para um padrão

5 references

```
public interface ICalculadoraDeDesconto
```

```
{
```

5 references

```
    decimal CalcularDesconto(decimal valorBruto);
```

```
}
```

Refatorando para um padrão

5 references

```
public class Pedido
{
    17 references | 4/4 passing
    public IList<ItemDePedido> Itens { get; set; } = new List<ItemDePedido>();
    5 references | 4/4 passing
    public ICalculadoraDeDesconto CalculadoraDeDesconto { get; set; } = new SemDesconto();
    5 references | 5/5 passing
    public virtual decimal Valor()
    {
        var valorBruto = Itens.Sum(x => x.Valor());
        return CalculadoraDeDesconto.CalcularDesconto(valorBruto);
    }
}
```


[TestMethod]

✓ | 0 references

```
public void ObterValorDoPedido()
{
    var pedido = new Pedido();
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 2 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });

    var valorEsperado = 500;

    Assert.AreEqual(valorEsperado, pedido.Valor());
}
```

[TestMethod]

✓ | 0 references

```
public void ObterValorDoPedidoComDescontoPorPontos()
{
    var pedido = new Pedido();

    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 2 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });
    pedido.Itens.Add(new ItemDePedido { Preco = 100, Quantidade = 1 });

    pedido.Desconto = new DescontoPorPontos(10);

    var valorEsperado = 495;

    Assert.AreEqual(valorEsperado, pedido.Valor());
}
```

2 references

```
public class DescontoPorPontos : ICalculadoraDeDesconto
{
    private int _pontos;

    1 reference | 1/1 passing
    public DescontoPorPontos(int pontos)
    {
        _pontos = pontos;
    }

    5 references
    public decimal CalcularDesconto(decimal valorBruto)
    {
        return valorBruto - (_pontos * 0.50m);
    }
}
```


Consequências

- Estratégias podem ser 'imitadas' ao testar classes que as utilizam.
- Estratégias podem ser testadas isoladamente.
- Adicionar um novo Strategy não modifica a classe que a utiliza.

[TestMethod, TestCategory("Iteração")]

✓ | 0 references

```
public void ObterValorDoPedidoIterageComCalculadoraDeDesconto()
{
    var pedido = new Pedido();

    var mockCalculadoraDeDesconto = new Mock<ICalculadoraDeDesconto>();

    pedido.CalculadoraDeDesconto = mockCalculadoraDeDesconto.Object;

    pedido.Valor();

    mockCalculadoraDeDesconto.Verify(x => x.CalcularDesconto(It.IsAny<decimal>()), Times.Once);
}
```

[TestMethod]

✓ | 0 references

```
public void CalcularDescontoSobreValorBrutoPorPontos()
{
    var calculadoraDeDesconto = new DescontoPorPontos(10);
    var valorEsperado = 495;
    Assert.AreEqual(valorEsperado, calculadoraDeDesconto.CalcularDesconto(500));
}
```

Consequências

```
[TestMethod, TestCategory("Iteração")]
✓ | 0 references
public void ObterValorDoPedidoIterageComCalculadoraDeDesconto()
{
    var pedido = new Pedido();

    var mockCalculadoraDeDesconto = new Mock<ICalculadoraDeDesconto>();

    pedido.CalculadoraDeDesconto = mockCalculadoraDeDesconto.Object;

    pedido.Valor();

    mockCalculadoraDeDesconto.Verify(x => x.CalcularDesconto(It.IsAny<decimal>()), Times.Once);
}
```

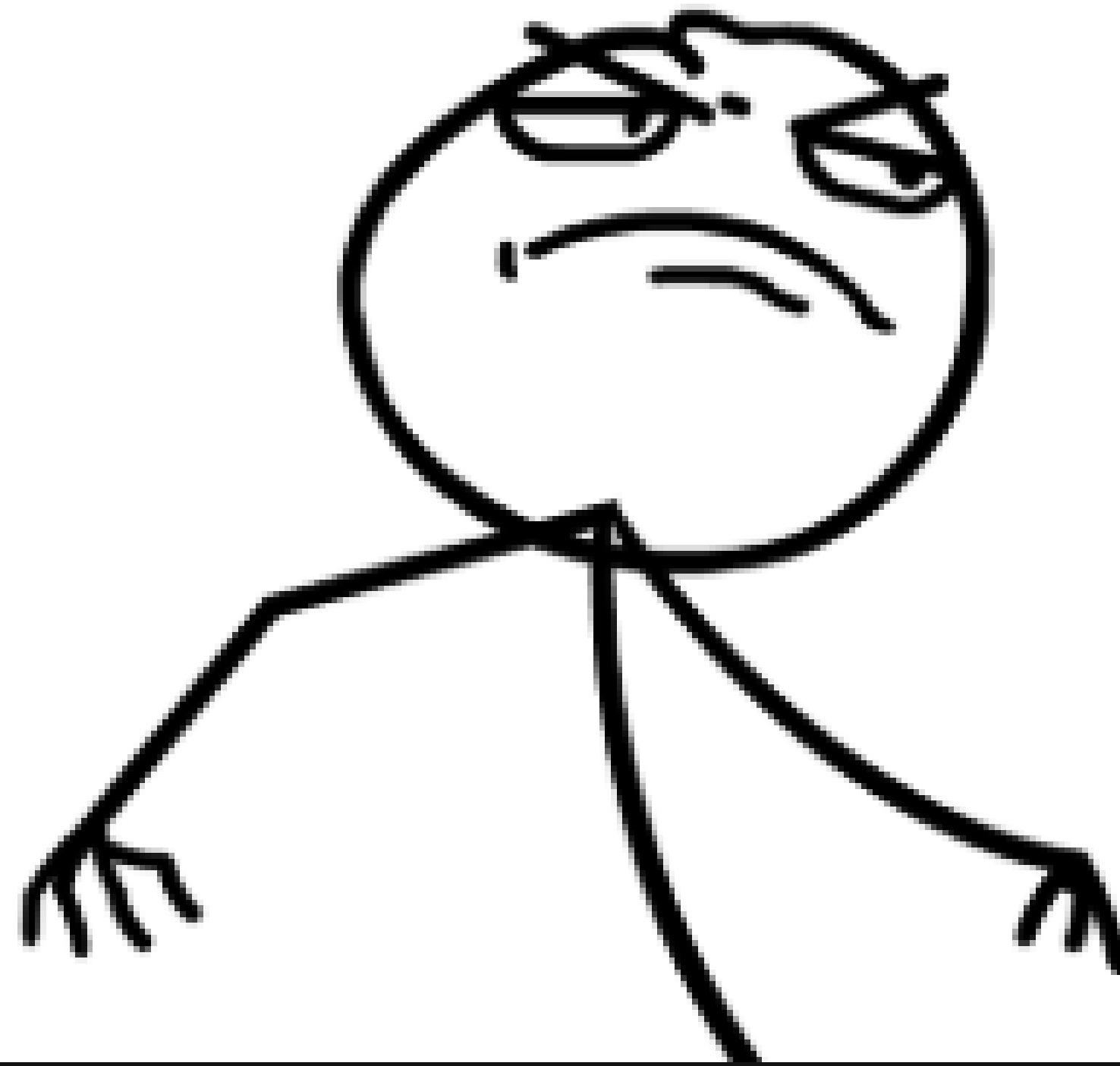
```
[TestMethod]
✓ | 0 references
public void CalcularDescontoSobreValorBrutoPorPontos()
{
    var calculadoraDeDesconto = new DescontoPorPontos(10);
    var valorEsperado = 495;
    Assert.AreEqual(valorEsperado, calculadoraDeDesconto.CalcularDesconto(500));
}
```



Um dia como qualquer outro...

- Boss: Jovem, temos que fazer uma modificação no serviço que altera senha. Agora após alterar a senha, o usuário vai receber um e-mail avisando sobre a troca de senha.
- Você: Blz! Já tenho um serviço que dispara e-mails, só vou reutilizar ele.

Um dia como qualquer outro...



2 references | 0 changes | 0 authors, 0 changes

```
public class ServicoDeSeguranca
```

```
{
```

```
    readonly IUserarioRepository _usuarioRepository;
```

1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes

```
public ServicoDeSeguranca(IUserarioRepository usuarioRepository)
```

```
{
```

```
    | _usuarioRepository = usuarioRepository;
```

```
}
```

1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes

```
public void AlterarSenha(string login, string senha, string novaSenha)
```

```
{
```

```
    var usuario = _usuarioRepository.ObterUsuario(login, senha);
```

```
    usuario.SetSenha(novaSenha);
```

```
    _usuarioRepository.Salvar(usuario);
```

```
}
```

```
}
```


[TestMethod]

✓ | 0 references | 0 changes | 0 authors, 0 changes

```
public void AlterarSenhaDoUsuario()
```

```
{
```

```
    string login = "vitor";
```

```
    string senha = "password";
```

```
    string novaSenha = "password@123";
```

```
    mockUsuarioRepository.Setup(x => x.ObterUsuario(login, senha)).Returns(mockUsuario.Object).Verifiable();
```

```
    mockUsuario.Setup(x => x.SetSenha(novaSenha)).Verifiable();
```

```
    mockUsuarioRepository.Setup(x => x.Salvar(mockUsuario.Object)).Verifiable();
```

```
    var servicoDeseguranca = new ServicoDeSeguranca(mockUsuarioRepository.Object);
```

```
    servicoDeseguranca.AlterarSenha(login, senha, novaSenha);
```

```
    mockUsuarioRepository.Verify(x => x.ObterUsuario(login, senha), Times.Once);
```

```
    mockUsuario.Verify(x => x.SetSenha(novaSenha), Times.Once);
```

```
    mockUsuarioRepository.Verify(x => x.Salvar(mockUsuario.Object), Times.Once);
```

```
}
```

2 references | 0 changes | 0 authors, 0 changes

```
public class ServicoDeSeguranca
```

```
{
```

```
    readonly IUserarioRepository _usuarioRepository;
```

```
    readonly IEmailService _emailService;
```

1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes

```
public ServicoDeSeguranca(IUserarioRepository usuarioRepository, IEmailService emailService)
```

```
{
```

```
    _usuarioRepository = usuarioRepository;
```

```
    _emailService = emailService;
```

```
}
```

1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes

```
public void AlterarSenha(string login, string senha, string novaSenha)
```

```
{
```

```
    var usuario = _usuarioRepository.ObterUsuario(login, senha);
```

```
    usuario.SetSenha(novaSenha);
```

```
    _usuarioRepository.Salvar(usuario);
```

```
    //Mão na massa
```

```
    var email = CriarEmailDeAlteracaoDeSenhaPara(usuario.Email);
```

```
    _emailService.SendEmail(email);
```

```
}
```

1 reference | 0 changes | 0 authors, 0 changes

```
private MailMessage CriarEmailDeAlteracaoDeSenhaPara(string email)
```

```
{
```

```
    return new MailMessage(from: "mailFrom@sistema.com", to: email, subject: "Senha alterada", body: "Sua senha foi alterada");
```

```
}
```

```
}
```

[TestMethod]

✓ | 0 references | 0 changes | 0 authors, 0 changes

```
public void AlterarSenhaDoUsuario()
```

```
{
```

```
    string login = "vitor";
```

```
    string senha = "password";
```

```
    string novaSenha = "password@123";
```

```
    mockUsuarioRepository.Setup(x => x.ObterUsuario(login, senha)).Returns(mockUsuario.Object).Verifiable();
```

```
    mockUsuario.Setup(x => x.SetSenha(novaSenha)).Verifiable();
```

```
    mockUsuarioRepository.Setup(x => x.Salvar(mockUsuario.Object)).Verifiable();
```

```
    var servicoDeseguranca = new ServicoDeSeguranca(mockUsuarioRepository.Object);
```

```
    servicoDeseguranca.AlterarSenha(login, senha, novaSenha);
```

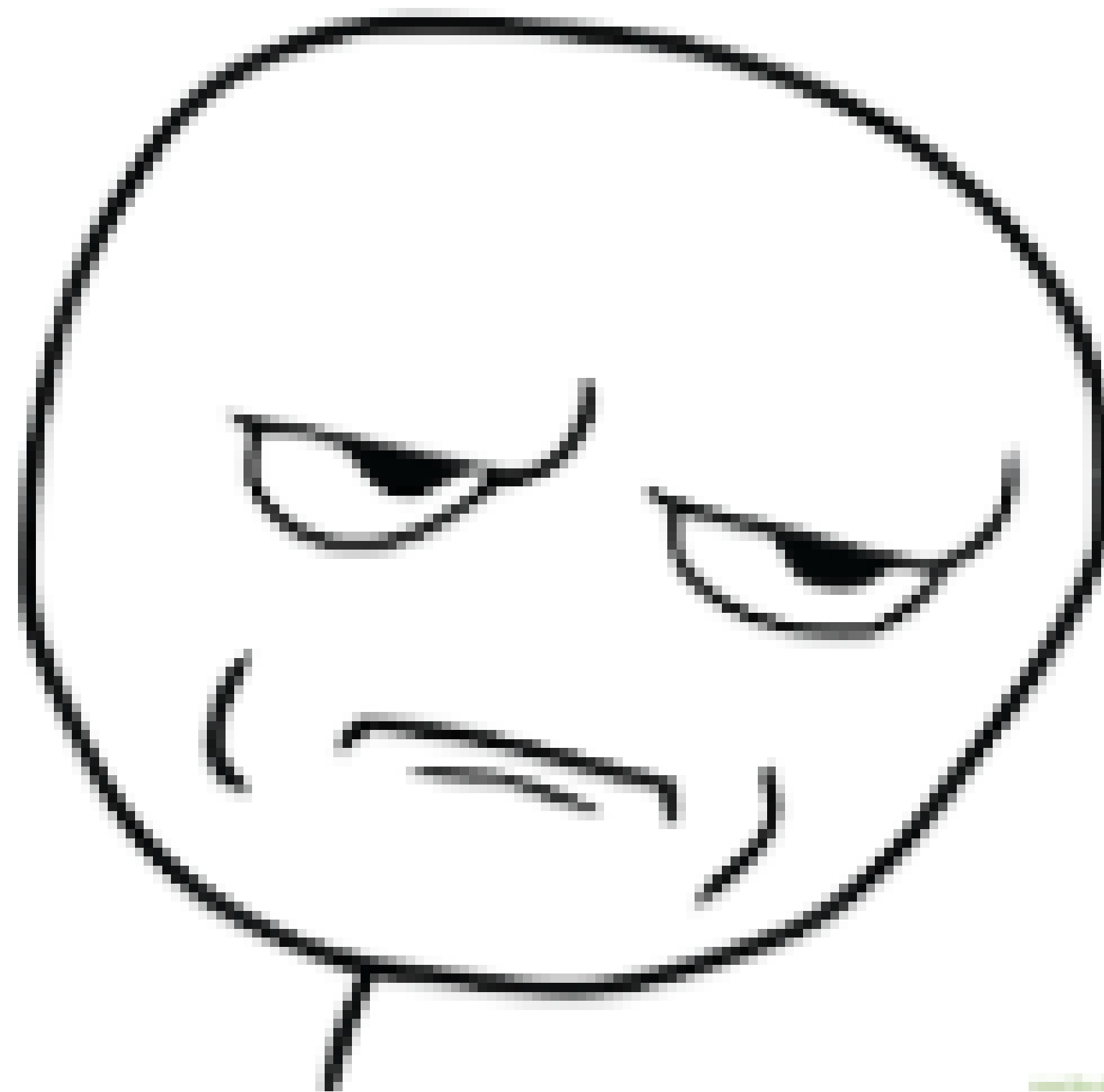
```
    mockUsuarioRepository.Verify(x => x.ObterUsuario(login, senha), Times.Once);
```

```
    mockUsuario.Verify(x => x.SetSenha(novaSenha), Times.Once);
```

```
    mockUsuarioRepository.Verify(x => x.Salvar(mockUsuario.Object), Times.Once);
```

```
}
```


Um dia como qualquer outro...



[TestMethod]

✔ | 0 references | 0 changes | 0 authors, 0 changes

```
public void AlterarSenhaDoUsuario()
```

```
{
```

```
    string login = "vitor";
```

```
    string senha = "password";
```

```
    string novaSenha = "password@123";
```

```
    mockUsuarioRepository.Setup(x => x.ObterUsuario(login, senha)).Returns(mockUsuario.Object).Verifiable();
```

```
    mockUsuario.Setup(x => x.SetSenha(novaSenha)).Verifiable();
```

```
    mockUsuarioRepository.Setup(x => x.Salvar(mockUsuario.Object)).Verifiable();
```

```
    mockEmailService.Setup(x => x.SendEmail(It.IsAny<MailMessage>())).Verifiable();
```

```
    var servicoDeseguranca = new ServicoDeSeguranca(mockUsuarioRepository.Object, mockEmailService.Object);
```

```
    servicoDeseguranca.AlterarSenha(login, senha, novaSenha);
```

```
    mockUsuarioRepository.Verify(x => x.ObterUsuario(login, senha), Times.Once);
```

```
    mockUsuario.Verify(x => x.SetSenha(novaSenha), Times.Once);
```

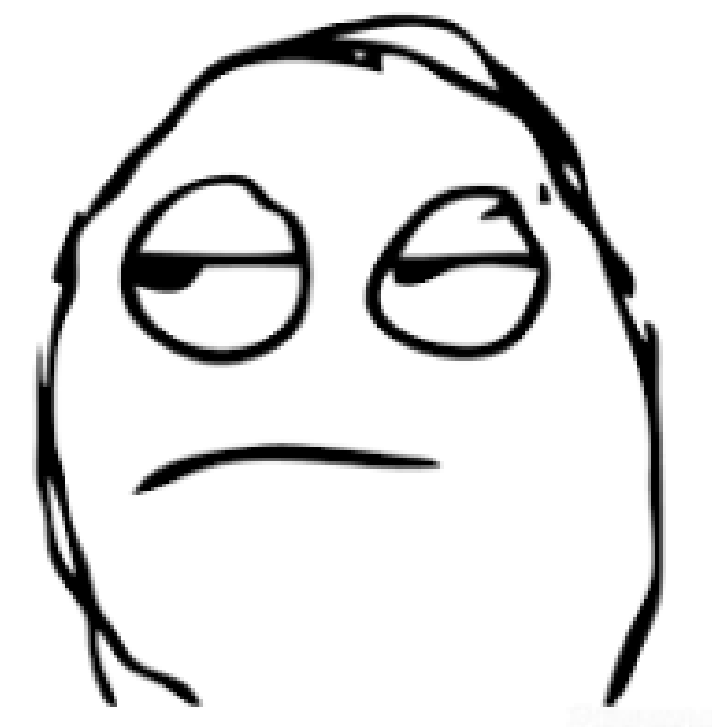
```
    mockUsuarioRepository.Verify(x => x.Salvar(mockUsuario.Object), Times.Once);
```

```
    mockEmailService.Verify(x => x.SendEmail(It.IsAny<MailMessage>()), Times.Once);
```

```
}
```

Um dia como qualquer outro...

- Boss: Então cara, dá pra fazer um esquema pra gente enviar ou não um e-mail de confirmação? Coloca um if lá só pra disparar ou não esse e-mail. (isso não acontece)
- Você: hmm... Ok... (não dou 3 dias pra pedir pra adicionar um log de alteração, meu código estava bonitinho, já, já vira um monstrengo.)



Um dia como qualquer outro...

- Você: (vou fazer logo uma alteração aqui)

```
[TestMethod]
```

```
0 references | 0 changes | 0 authors, 0 changes
```

```
public void AlterarSenhaDoUsuarioEEnviarEmailDeConfirmacao()  
{
```

```
servicoDeseguranca.AlterarSenhaEEnviarEmail(login, senha, novaSenha);
```

Refatorando para um padrão

Decorator Pattern

- permite adicionar um comportamento a um objeto já existente em tempo de execução, ou seja, agrega dinamicamente responsabilidades adicionais a um objeto

Aplicabilidade (by the book):

- Acrescentar ou remover responsabilidades a objetos individuais dinamicamente.
- pode usar um ou mais decoradores para englobar um objeto
- Prover alternativa flexível ao uso de subclasses para se estender a funcionalidade de uma classe

Refatorando para um padrão

5 references | 0 changes | 0 authors, 0 changes

```
public interface IServicoDeSeguranca
```

```
{
```

6 references | 2/2 passing | 0 changes | 0 authors, 0 changes

```
void AlterarSenha(string login, string senha, string novaSenha);
```

```
}
```


2 references | 0 changes | 0 authors, 0 changes

```
public class ServicoDeSeguranca : IServicoDeSeguranca
```

```
{
```

```
    readonly IUserarioRepository _usuarioRepository;
```

1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes

```
public ServicoDeSeguranca(IUserarioRepository usuarioRepository)
```

```
{
```

```
    _usuarioRepository = usuarioRepository;
```

```
}
```

6 references | 2/2 passing | 0 changes | 0 authors, 0 changes

```
public void AlterarSenha(string login, string senha, string novaSenha)
```

```
{
```

```
    var usuario = _usuarioRepository.ObterUsuario(login, senha);
```

```
    usuario.SetSenha(novaSenha);
```

```
    _usuarioRepository.Salvar(usuario);
```

```
}
```

```
}
```

[TestMethod]

✓ | 0 references | 0 changes | 0 authors, 0 changes

```
public void AlterarSenhaDoUsuario()
```

```
{
```

```
    string login = "vitor";
```

```
    string senha = "password";
```

```
    string novaSenha = "password@123";
```

```
    mockUsuarioRepository.Setup(x => x.ObterUsuario(login, senha)).Returns(mockUsuario.Object).Verifiable();
```

```
    mockUsuario.Setup(x => x.SetSenha(novaSenha)).Verifiable();
```

```
    mockUsuarioRepository.Setup(x => x.Salvar(mockUsuario.Object)).Verifiable();
```

```
    var servicoDeseguranca = new ServicoDeSeguranca(mockUsuarioRepository.Object);
```

```
    servicoDeseguranca.AlterarSenha(login, senha, novaSenha);
```

```
    mockUsuarioRepository.Verify(x => x.ObterUsuario(login, senha), Times.Once);
```

```
    mockUsuario.Verify(x => x.SetSenha(novaSenha), Times.Once);
```

```
    mockUsuarioRepository.Verify(x => x.Salvar(mockUsuario.Object), Times.Once);
```

```
}
```

2 references | 0 changes | 0 authors, 0 changes

```
public class ServicoDeSegurancaAlterarSenhaEEnviaEmail : IServicoDeSeguranca
{
    readonly IServicoDeSeguranca _servicoDeSeguranca;
    readonly IUsuarioRepository _usuarioRepository;
    readonly IEmailService _emailService;
```

1 reference | 0 changes | 0 authors, 0 changes

```
public ServicoDeSegurancaAlterarSenhaEEnviaEmail(
    IServicoDeSeguranca servicoDeSeguranca,
    IEmailService emailService,
    IUsuarioRepository usuarioRepository
) ...
```

5 references | 1/1 passing | 0 changes | 0 authors, 0 changes

```
public void AlterarSenha(string login, string senha, string novaSenha)
{
    _servicoDeSeguranca.AlterarSenha(login, senha, novaSenha);

    EnviarEmail(login); //compondo a operação com novas funcionalidades
}
```

1 reference | 0 changes | 0 authors, 0 changes

```
private void EnviarEmail(string login)
{
    var user = _usuarioRepository.ObterUsuario(login);

    var email = CriarEmailDeAlteracaoDeSenhaPara(user.Email);

    _emailService.SendEmail(email);
}
```

1 reference | 0 changes | 0 authors, 0 changes

```
private MailMessage CriarEmailDeAlteracaoDeSenhaPara(string email) ...
```

```
}
```


[TestMethod]

✔ | 0 references | 0 changes | 0 authors, 0 changes

```
public void AlterarSenhaDoUsuarioEnviandoEmailDeConfirmacao()
{
    string login = "vitor";
    string senha = "password";
    string novaSenha = "password@123";


    mockUsuarioRepository.Setup(x => x.ObterUsuario(login)).Returns(mockUsuario.Object).Verifiable();
    mockEmailService.Setup(x => x.SendEmail(It.IsAny<MailMessage>())).Verifiable();

    var mockServicoDeSeguranca = new Mock<IServicoDeSeguranca>();

    var servicoDeseguranca = new ServicoDeSegurancaAlterarSenhaEEnviaEmail(
        mockServicoDeSeguranca.Object,
        mockEmailService.Object,
        mockUsuarioRepository.Object);

    servicoDeseguranca.AlterarSenha(login, senha, novaSenha);

    mockServicoDeSeguranca.Verify(x => x.AlterarSenha(login, senha, novaSenha), Times.Once);
    mockEmailService.Verify(x => x.SendEmail(It.IsAny<MailMessage>()), Times.Once);
}
```

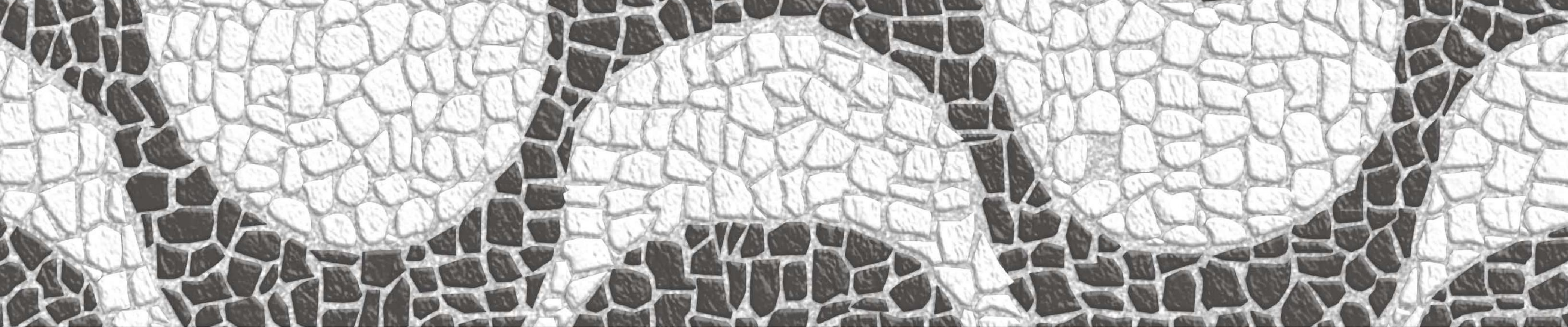

The image features a repeating mosaic pattern of irregular, light-colored stone tiles with dark grey grout. This pattern covers the top and bottom portions of the slide, framing a central white rectangular area.

Perguntas!?



Obrigado!





<Coders> in

</Rio>

