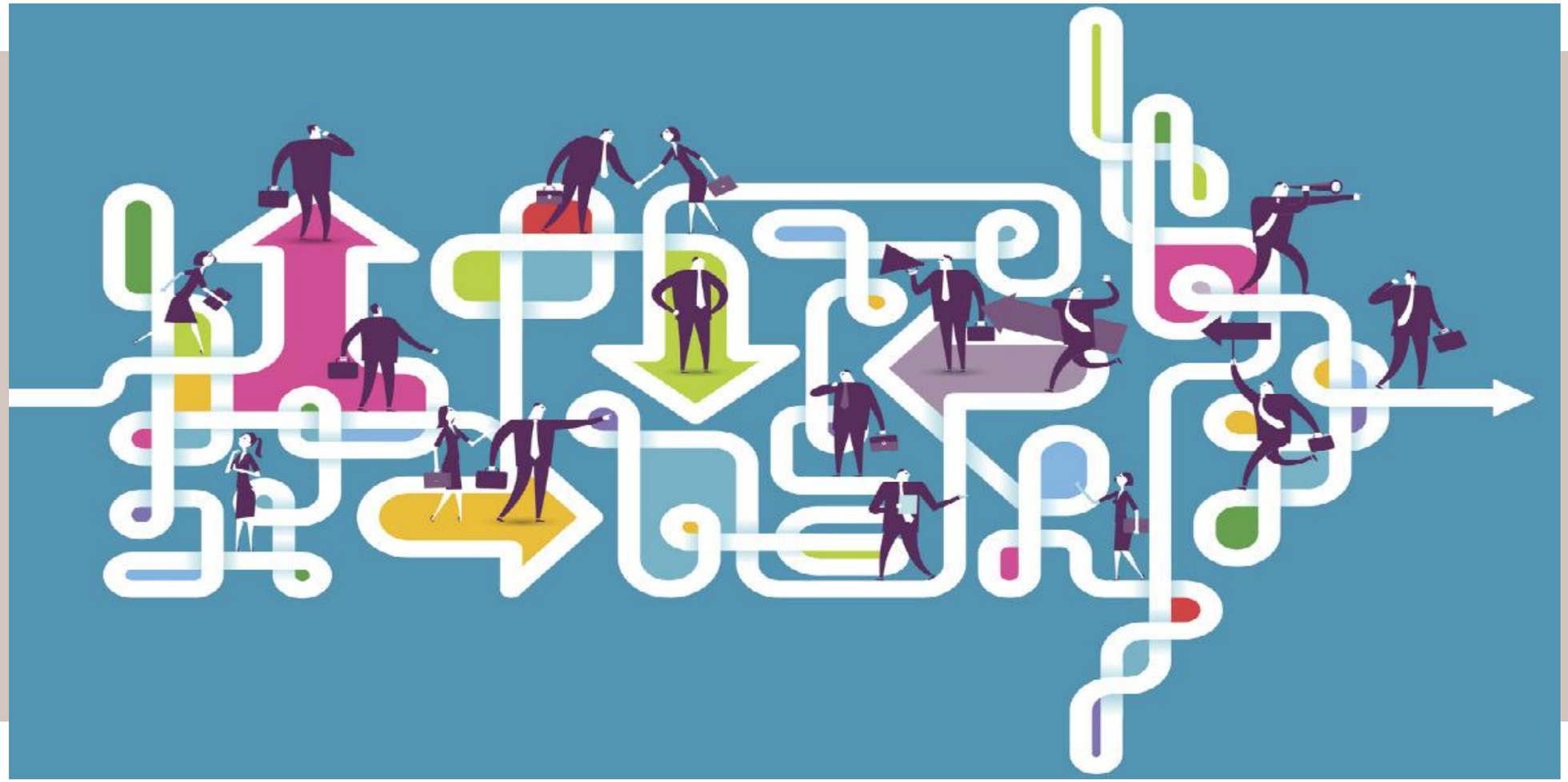
Princípios e boas práticas



Matheus Neder – Arquiteto de Soluções

Princípios e boas práticas

```
string Weekday1(int day)
{
    switch (day)
    {
        case 1:
            return "Monday";
        case 2:
            return "Tuesday";
        case 3:
            return "Wednesday";
        case 4:
            return "Thursday";
        case 5:
            return "Friday";
        case 6:
            return "Saturday";
        case 7:
            return "Sunday";
        default:
            throw new InvalidOperationException("day must be in range 1 to 7");
    }
}
```

```
string Weekday2(int day)
{
    if ((day < 1) || (day > 7)) throw new InvalidOperationException("day must be in range 1 to 7");
    string[] days = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
    return days[day - 1];
}
```

Princípios e boas práticas

```
string Weekday1(int day)
{
    switch (day)
    {
        case 1:
            return "Monday";
        case 2:
            return "Tuesday";
        case 3:
            return "Wednesday";
        case 4:
            return "Thursday";
        case 5:
            return "Friday";
        case 6:
            return "Saturday";
        case 7:
            return "Sunday";
        default:
            throw new InvalidOperationException("day must be in range 1 to 7");
    }
}
```

KISS

Keep It Simple Stupid

```
string Weekday2(int day)
{
    if ((day < 1) || (day > 7)) throw new InvalidOperationException("day must be in range 1 to 7");
    string[] days = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
    return days[day - 1];
}
```

Princípios e boas práticas


```
/// <summary>
/// Inclui um novo Cliente
/// </summary>
/// <param name="addClientRequest">Cliente</param>
/// <returns>Client</returns>
[HttpPost]
[ProducesResponseType(typeof(ClientCoreException), 400)]
[ProducesResponseType(typeof(AddClientPost), 200)]
0 references | Luciano Lima, 9 days ago | 1 author, 1 change | 3 work items | 0 requests | 0 exceptions
public async Task<IActionResult> AddClientAsync([FromBody] AddClientPost addClientRequest)
{
    try
    {
        var client = Mapper.Map<Client>(addClientRequest);

        await clientService.AddClientAsync(client);

        return Ok(client);
    }
    catch (CoreException e)
    {
        return BadRequest(e.Message);
    }
}
```

Princípios e boas práticas


```
/// <summary>
/// Inclui um novo Cliente
/// </summary>
/// <param name="addClientRequest">Cliente</param>
/// <returns>Client</returns>
[HttpPost]
[ProducesResponseType(typeof(ClientCoreException), 400)]
[ProducesResponseType(typeof(AddClientPost), 200)]
0 references | Luciano Lima, 9 days ago | 1 author, 1 change | 3 work items | 0 requests | 0 exceptions
public async Task<IActionResult> AddClientAsync([FromBody] AddClientPost addClientRequest)
{


    var client = Mapper.Map<Client>(addClientRequest);

    await clientService.AddClientAsync(client);

    return Ok(client);


}
```

Princípios e boas práticas

S Single Responsibility Principle (SRP)

O Open/Closed Principle (OCP)

L Liskov Substitution Principle (LSP)

I Interface Segregation Principle (ISP)

D Dependency Inversion Principle (DIP)

Robert C. Martin (Uncle Bob) – Design Principles and Design Patterns – 2000

Single Responsibility Principle (SRP)

Single Responsibility Principle (SRP)

- Uma classe deve ter uma única responsabilidade;
- Uma classe deve ter apenas uma razão para mudança;
- A mudança em uma classe deve afetar apenas um ator (Clean Architecture)

Single Responsibility Principle (SRP)

- Qual propósito da sua classe?
- Será que é realmente isso que essa classe deve ter?
- Será que é realmente isso que a classe deve fazer?

Single Responsibility Principle (SRP)

```
1  public class DebitoContaCorrente
2  {
3      public void ValidarSaldo(int valor) { }
4
5      public void DebitarConta(int valor) { }
6
7      public void EmitirComprovante() { }
8  }
```

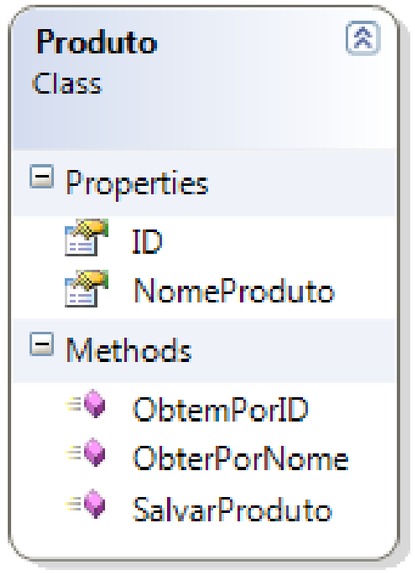
Eduardo Pires - http://www.eduardopires.net.br/2013/05/single-responsibility-principle-srp/

Single Responsibility Principle (SRP)

```
1  public class DebitoContaCorrente
2  {
3      public void DebitarConta(int valor) { }
4  }
5
6  public class SaldoContaCorrente
7  {
8      public void ValidarSaldo(int valor) { }
9  }
10
11 public class ComprovanteContaCorrente
12 {
13     public void EmitirComprovante() { }
14 }
```

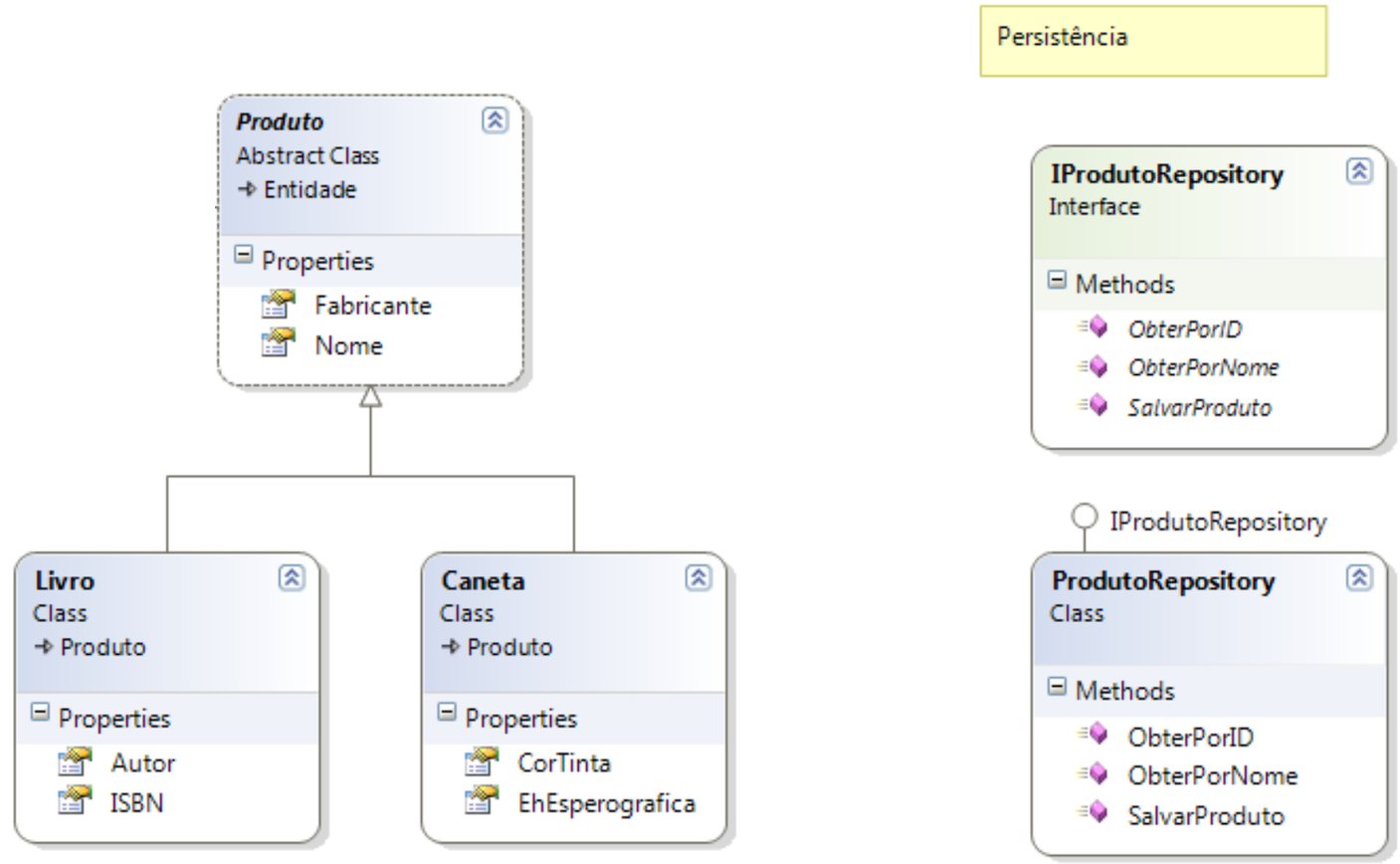
Eduardo Pires - http://www.eduardopires.net.br/2013/05/single-responsibility-principle-srp/

Single Responsibility Principle (SRP)



https://www.devmedia.com.br/arquitetura-o-principio-da-responsabilidade-unica/18700

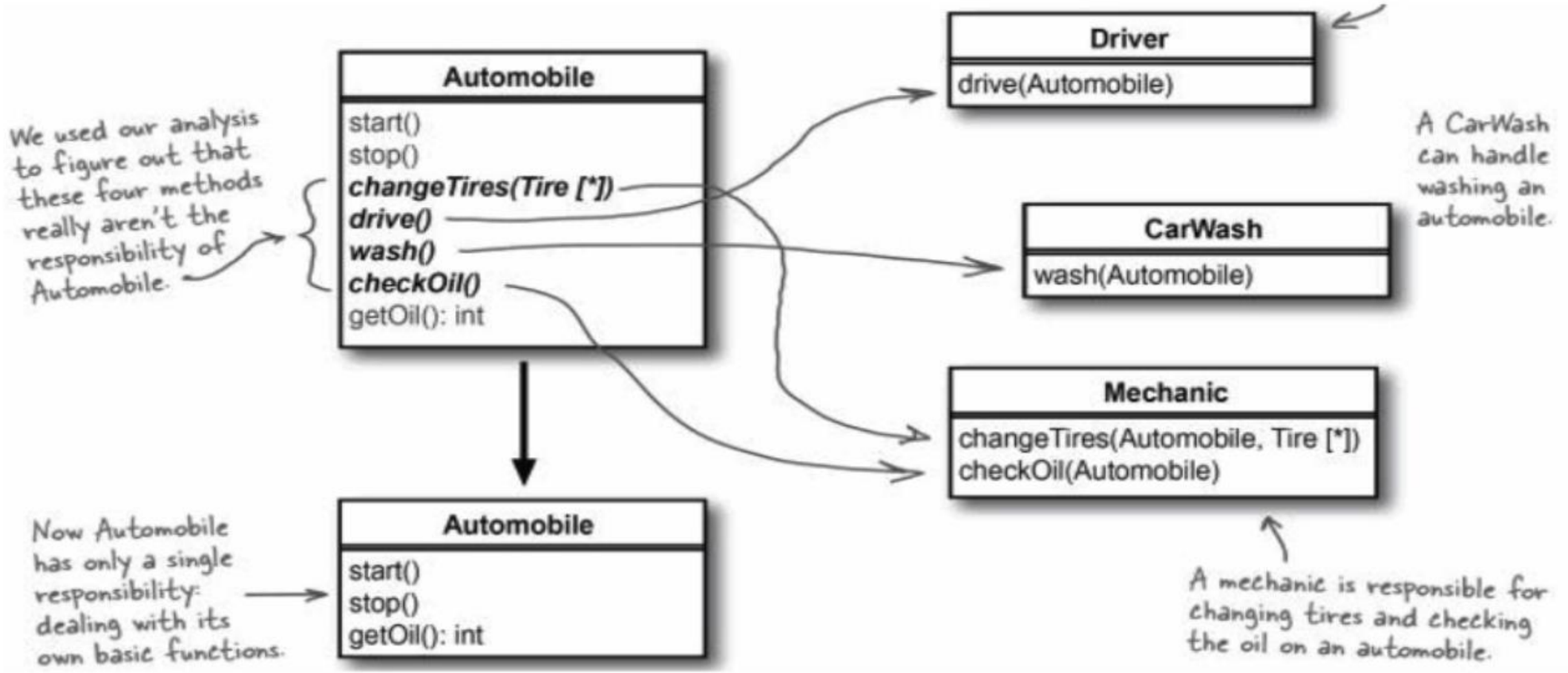
Single Responsibility Principle (SRP)



https://www.devmedia.com.br/arquitetura-o-principio-da-responsabilidade-unica/18700

Single Responsibility Principle (SRP)

- **Going from multiple responsibilities to single responsibility -**



https://www.slideshare.net/dhaval201279/oo-design-principles-heuristics

Open/Closed Principle (OCP)

- Fechado para modificações porém aberto para extensibilidade;
- Você deve ser capaz de estender o comportamento das classes sem precisar modificá-las.

Open/Closed Principle (OCP)

```
1  public enum TipoDebito { ContaCorrente, Poupanca }
2
3  public class Debito
4  {
5      public void Debitar(int valor, TipoDebito tipo)
6      {
7          if (tipo == TipoDebito.Poupanca)
8          {
9              // Debita Poupanca
10         }
11         if (tipo == TipoDebito.ContaCorrente)
12         {
13             // Debita ContaCorrente
14         }
15     }
16 }
```

Eduardo Pires - http://www.eduardopires.net.br/2013/05/open-closed-principle-ocp/

Open/Closed Principle (OCP)

```
1  public abstract class Debito
2  {
3      public abstract void Debitar(int valor);
4  }
5
6  public class DebitoContaCorrente : Debito
7  {
8      public override void Debitar(int valor)
9      {
10         // Debita Conta Corrente
11     }
12 }
13
14 public class DebitoContaPoupanca : Debito
15 {
16     public override void Debitar(int valor)
17     {
18         // Debita Conta Poupança
19     }
20 }
21
22 public class DebitoContaInvestimento : Debito
23 {
24     public override void Debitar(int valor)
25     {
26         // Debita Conta Investimento
27     }
28 }
```

Eduardo Pires - http://www.eduardopires.net.br/2013/05/open-closed-principle-ocp/

Liskov Substitution Principle (LSP)

Se *q(x)* é uma propriedade demonstrável dos objetos *x* de tipo *T*.

Então *q(y)* deve ser verdadeiro para objetos *y* de tipo *S* onde *S* é um subtipo de *T*.

Barbara Liskov, 1993

Liskov Substitution Principle (LSP)

Se *q(x)* é uma propriedade demonstrável dos objetos *x* de tipo *T*.

Então *q(y)* deve ser verdadeiro para objetos *y* de tipo *S* onde *S* é um subtipo de *T*.

Barbara Liskov, 1993

Classes derivadas devem ser substituíveis por suas classes bases e garantir o comportamento esperado na classe base.

Uncle Bob, 2000

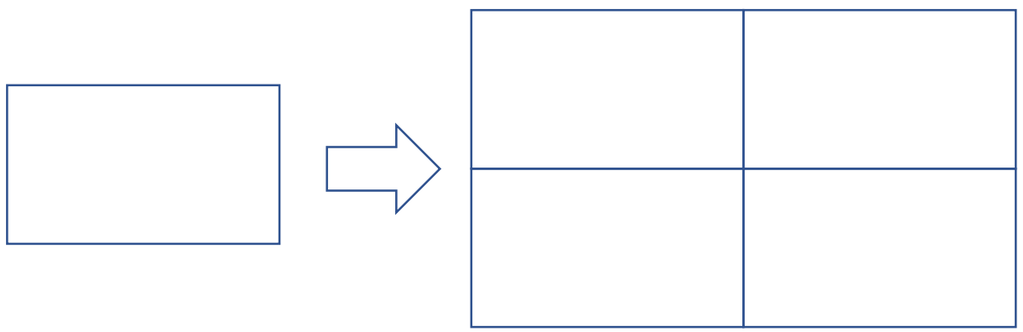
Liskov Substitution Principle (LSP)

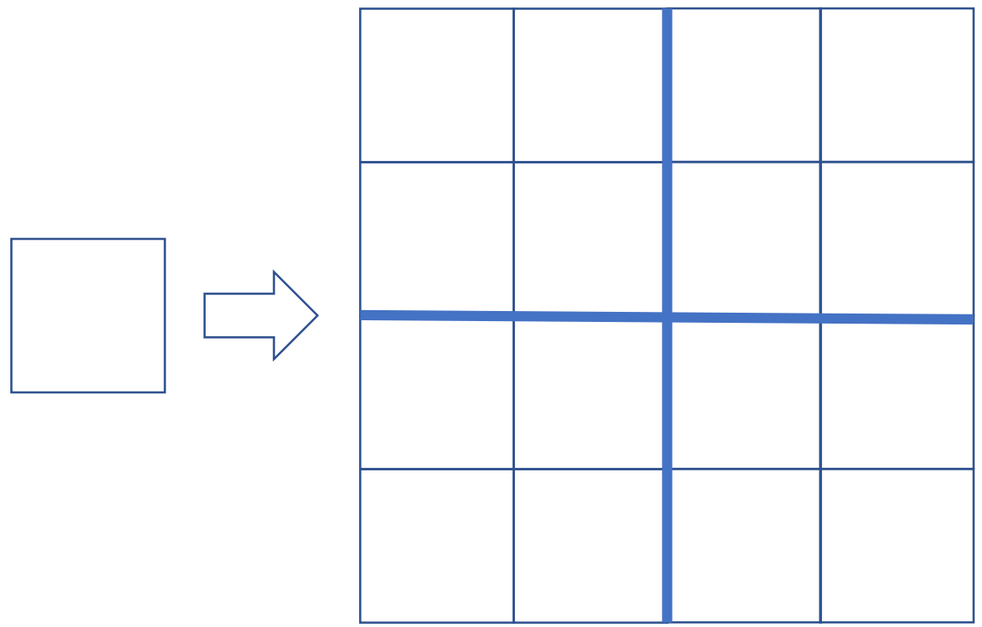
```
2 references | 0 changes | 0 authors, 0 changes
class Retangulo
{
    6 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public virtual double Altura { get; set; }
    6 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public virtual double Comprimento { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public double Area { get { return Altura * Comprimento; } }
}

1 reference | 0 changes | 0 authors, 0 changes
class Quadrado : Retangulo
{
    6 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public override double Altura { set { base.Altura = base.Comprimento = value; } }
    6 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public override double Comprimento { set { base.Altura = base.Comprimento = value; } }
}
```

Liskov Substitution Principle (LSP)

```
0 references | 0 changes | 0 authors, 0 changes
class Operacoes
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Crescer(Retangulo retangulo)
    {
        retangulo.Altura *= 2;
        retangulo.Comprimento *= 2;
    }
}
```





Interface Segregation Principle (ISP)

Construa interfaces com granularidade fina que sejam específicas para o cliente.

Uncle Bob, 2000

Interface Segregation Principle (ISP)

```
0 references | 0 changes | 0 authors, 0 changes
public interface IBaseRepository<TEntity> where TEntity : class
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    void Gravar(TEntity obj);
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    void Atualizar(TEntity obj);
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    IEnumerable<TEntity> ObterTodos();
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    TEntity ObterPorId(int id);
}
```

Interface Segregation Principle (ISP)

```
0 references | 0 changes | 0 authors, 0 changes
public class ClienteRepository : IBaseRepository<Cliente>
{
    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Gravar(Cliente obj)
    {
        //ação de gravar
    }

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Atualizar(Cliente obj)
    {
        //ação de atualizar
    }

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public IEnumerable<Cliente> ObterTodos()
    {
        //ação de listar todos
        return Enumerable.Empty<Cliente>();
    }

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Cliente ObterPorId(int id)
    {
        //ação de listar por id
        return default(Cliente);
    }
}
```

Interface Segregation Principle (ISP)

```
0 references | 0 changes | 0 authors, 0 changes
public class CidadeRepository : IBaseRepository<Cidade>
{
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Gravar(Cidade obj)
    {
        throw new NotImplementedException();
    }

    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Atualizar(Cidade obj)
    {
        throw new NotImplementedException();
    }

    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public IEnumerable<Cidade> ObterTodos()
    {
        //ação para obter todas as cidades
        return Enumerable.Empty<Cidade>();
    }

    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Cidade ObterPorId(int id)
    {
        //ação para obter uma cidade por id
        return default(Cidade);
    }
}
```

Interface Segregation Principle (ISP)

```
0 references | 0 changes | 0 authors, 0 changes
public interface IBaseEscritaRepository<TEntity> where TEntity : class
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    void Gravar(TEntity obj);
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    void Atualizar(TEntity obj);
}
```

```
0 references | 0 changes | 0 authors, 0 changes
public interface IBaseLeituraRepository<TEntity> where TEntity : class
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    IEnumerable<TEntity> ObterTodos();
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    TEntity ObterPorId(int id);
}
```

Interface Segregation Principle (ISP)

```
0 references | 0 changes | 0 authors, 0 changes
public class ClienteRepository : IBaseEscritaRepository<Cliente>, IBaseLeituraRepository<Cliente>
{
    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Gravar(Cliente obj)
    {
        //ação de gravar
    }

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void Atualizar(Cliente obj)
    {
        //ação de atualizar
    }

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public IEnumerable<Cliente> ObterTodos()
    {
        //ação de listar todos
        return Enumerable.Empty<Cliente>();
    }

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Cliente ObterPorId(int id)
    {
        //ação de listar por id
        return default(Cliente);
    }
}
```

Interface Segregation Principle (ISP)

```
0 references | 0 changes | 0 authors, 0 changes
public class CidadeRepository : IBaseLeituraRepository<Cidade>
{
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public IEnumerable<Cidade> ObterTodos()
    {
        //ação para obter todas as cidades
        return Enumerable.Empty<Cidade>();
    }

    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Cidade ObterPorId(int id)
    {
        //ação para obter uma cidade por id
        return default(Cidade);
    }
}
```

Dependency Inversion Principle (DIP)

Dependa de abstrações (interfaces) ao invés de classes concretas.

Uncle Bob, 2000

Interfaces

```
2 references
class RepositorioPessoas
{
    0 references
    public Pessoa[] ObterPessoas()
    {
        var query = @"
        select top 100 nome
            , salario
            , data_de_nacimento
            , tipo
        from pessoas";

        return Db.Query<Pessoa>(query);
    }

    1 reference
    public void Atualizar(Pessoa pessoa)
    {
        var query = "update pessoas set ...";

        Db.Query(query);
    }
}
```

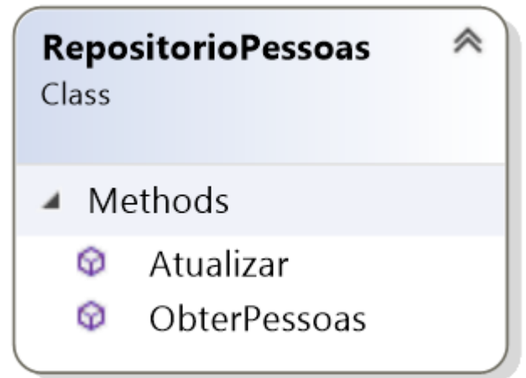
Dependency Inversion Principle (DIP)

```
1 reference
class Exemplo02
{
    private RepositorioPessoas repositorioDePessoas;

    0 references
    public Exemplo02(RepositorioPessoas repositorioDePessoas)
    {
        this.repositorioDePessoas = repositorioDePessoas;
    }

    0 references
    public void CorrigirDataDeNascimento(Pessoa pessoa, DateTime novaDataDeNascimento)
    {
        pessoa.DataDeNascimento = novaDataDeNascimento;

        repositorioDePessoas.Atualizar(pessoa);
    }
}
```



Dependency Inversion Principle (DIP)

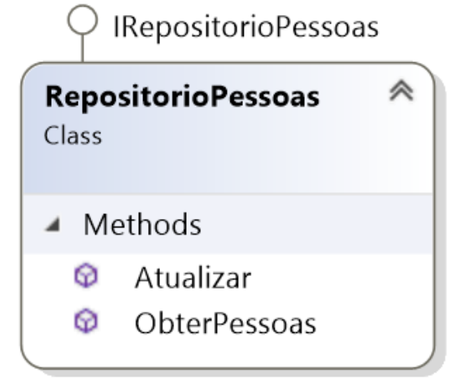
```
1 reference
interface IRepositorioPessoas
{
    2 references
    void Atualizar(Pessoa pessoa);
    1 reference
    Pessoa[] ObterPessoas();
}
```

```
2 references
class RepositorioPessoas : IRepositorioPessoas
{
    1 reference
    public Pessoa[] ObterPessoas()
    {
        var query = @"
        select top 100 nome
            , salario
            , data_de_nacimento
            , tipo
        from pessoas";

        return Db.Query<Pessoa>(query);
    }

    2 references
    public void Atualizar(Pessoa pessoa)
    {
        var query = "update pessoas set ...";

        Db.Query(query);
    }
}
```



Dependency Inversion Principle (DIP)

```
1 reference
class Exemplo02
{
    private IRepositorioPessoas repositorioDePessoas;

    0 references
    public Exemplo02(IRepositorioPessoas repositorioDePessoas)
    {
        this.repositorioDePessoas = repositorioDePessoas;
    }

    0 references
    public void CorrigirDataDeNascimento(Pessoa pessoa, DateTime novaDataDeNascimento)
    {
        pessoa.DataDeNascimento = novaDataDeNascimento;

        repositorioDePessoas.Atualizar(pessoa);
    }
}
```

https://goo.gl/forms/xqTzlynPITmdr9CF3