Princípios SOLID

Prof. Pedro Pongelupe

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Engenharia de Software

Sumário

SOLID Principles

Proposto por Robert Martin[1]:

- ***S**ingle responsibility*
- ***O**pen/Closed*
- ***L**iskov substitution*
- ***I**nterface segregation*
- ***D**ependency inversion*

[1] *Martin, Robert Cecil (2002). Agile software development: principles, patterns, and practices. Upper Saddle River, NJ: Pearson Education. ISBN 9780135974445.*

Single responsibility principle (SRP)

Single responsibility principle (SRP)

"Uma classe deve ter apenas um motivo para mudar, o que significa que ela deve ter uma única função."

- "Responsabilidade": motivo para mudança.
- Obtida com coesão e encapsulamento.

Roger Whitney, 2016, *Advanced Object-Oriented Design & Programming*

Tente descrever uma classe com 25 palavras ou menos, e não use "e" ou "ou". Se você não consegue fazer isso, pode ser que você tenha mais de uma classe.

Single responsibility principle (SRP)

Considere uma classe que realiza as seguintes operações:

1. Abrir uma conexão com o banco de dados.
2. Realizar consulta ao banco de dados.
3. Escrever esses dados em um arquivo.

Quais seriam seus "motivos para mudança":

1. E se for utilizado um novo banco de dados?
2. E se for utilizado um ORM para gerenciar as consultas?
3. E se a estrutura dos dados de escrita for alterada?

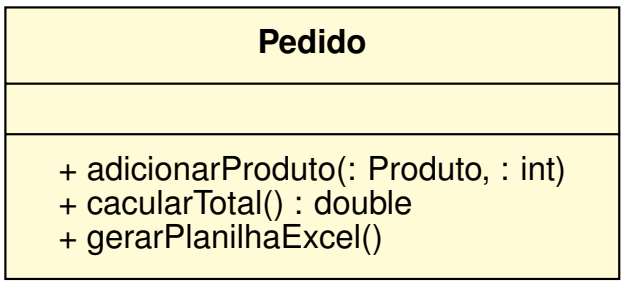
Cada mudança pode afetar as demais operações. O ideal seria ter três classes distintas, cada uma com sua responsabilidade.

Single responsibility principle (SRP)

Exemplo 1: Relatório de vendas (viola o SRP)

- Razões para mudança: estética x no conteúdo
- Proposta: separar conteúdo da apresentação.

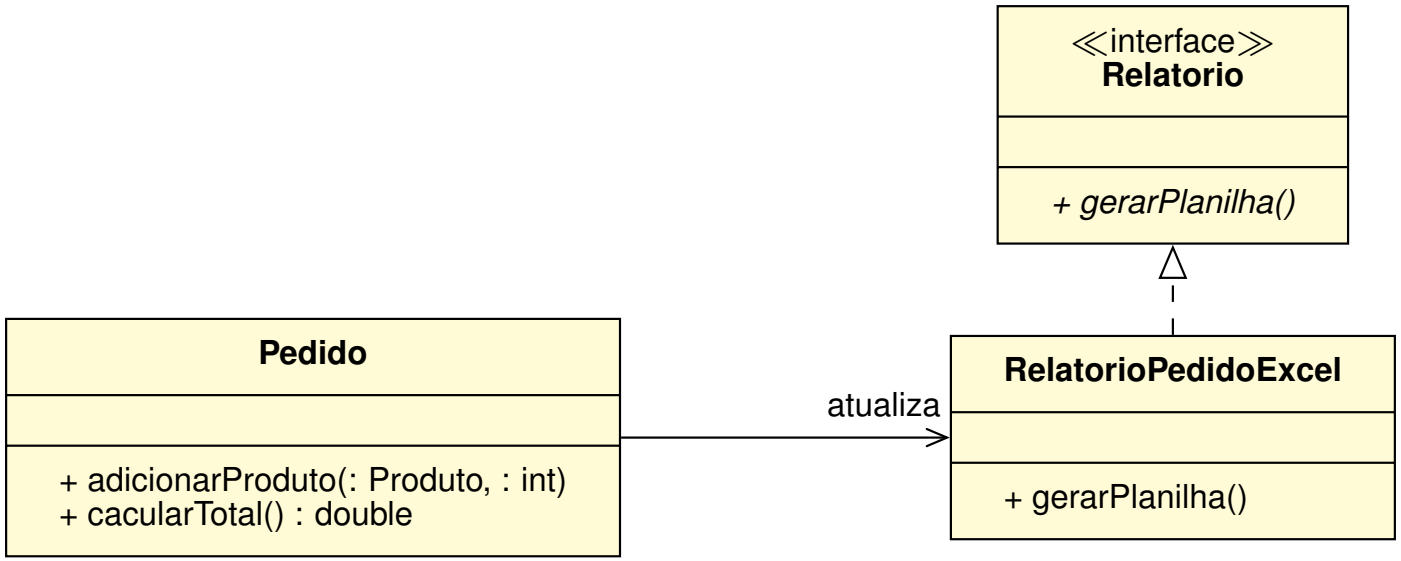
Exemplo 2: Gerar um pedido de compra de Produtos (viola o SRP).




```
public class Pedido
{
   public void adicionarProduto(Produto p, int q) { }
   public double calcularTotal() { }
   public void gerarPlanilhaExcel() { }
}
```

Single responsibility principle (SRP)

Exemplo 2 (cont.): Gerar um pedido de compra de Produtos.



Single responsibility principle (SRP)

Ideia geral do SRP

Uma classe deve ter um, e somente um, motivo para mudar. Sendo assim, uma determinada classe deve ter apenas uma responsabilidade dentro do sistema.

Problemas causados por violar o SRP

- Falta de coesão
- Alto acoplamento
- Dificuldade de testar
- Objeto deus

Open/closed principle (OCP)

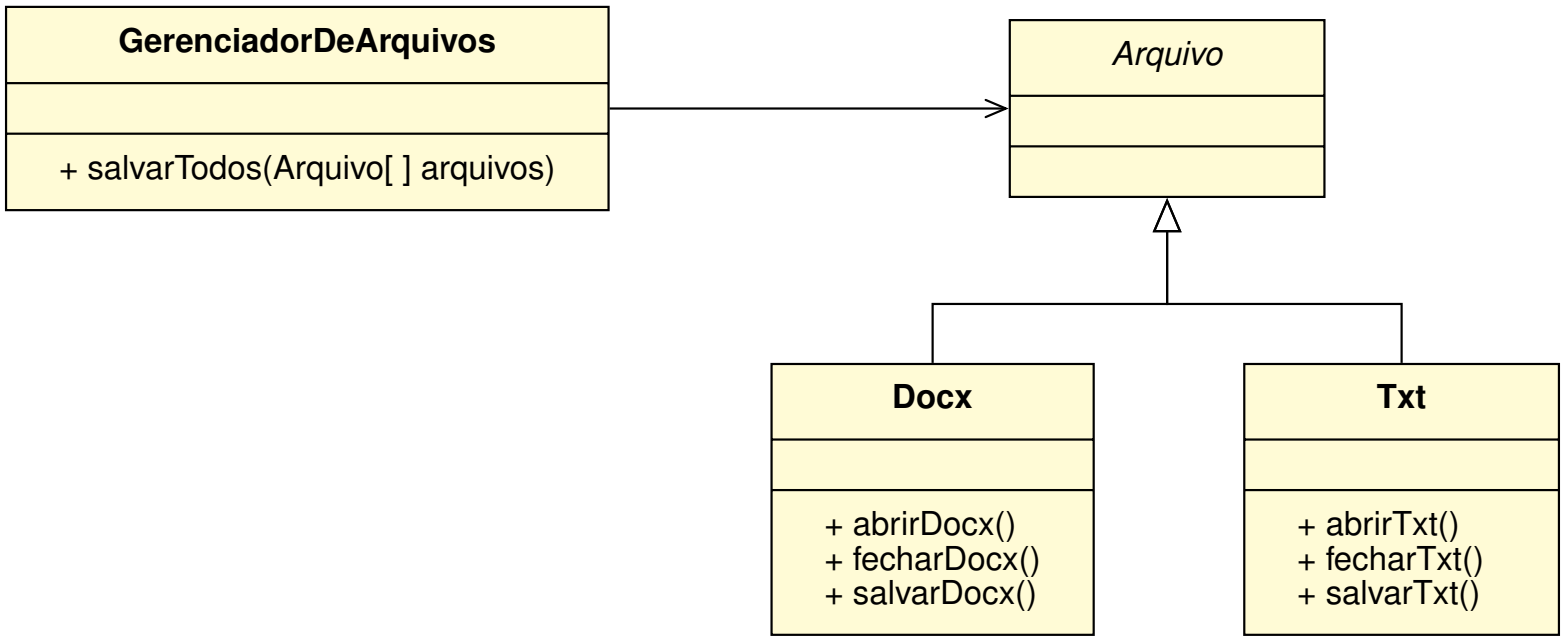
Open/closed principle (OCP)

"Objetos ou entidades devem ser abertos para extensão, mas fechados para modificação."

- Uma classe deve ser facilmente extensível, sem a necessidade de modificação.
- Mudanças apenas para correção de código.
- Extensão: classes abstratas e interfaces.

Open/closed principle (OCP)

Exemplo: Gerenciar arquivos de diversos formatos (viola o OCP).



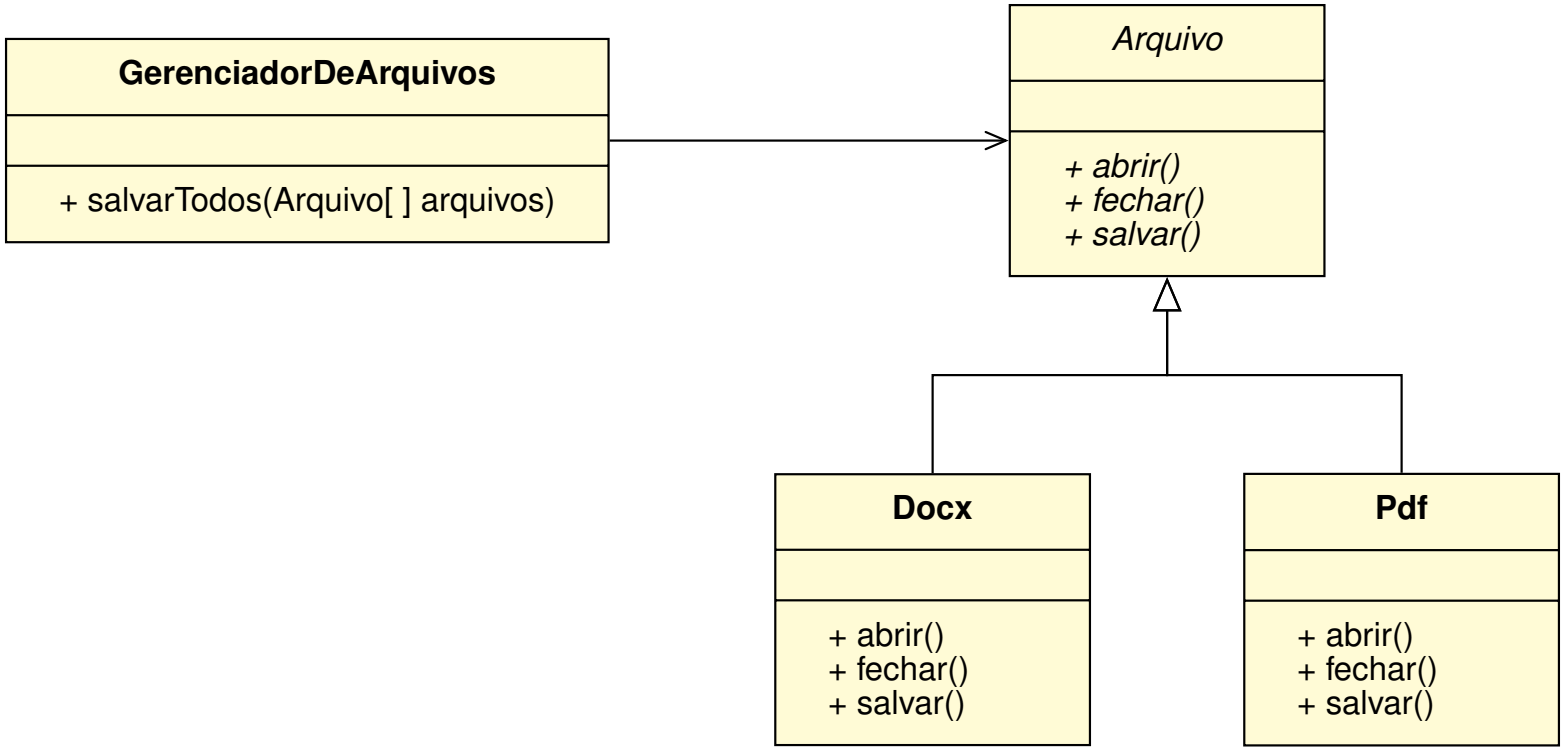
Open/closed principle (OCP)

```
public abstract class Arquivo {

}

public class Docx extends Arquivo {

    public void abrirDocx() { }
    public void fecharDocx() { }
    public void salvarDocx() { }
}

public class Txt extends Arquivo {

    public void abrirTxt() { }
    public void fecharTxt() { }
    public void salvarTxt() { }
}
```

```
public class GerenciadorDeArquivos {

    public salvartodos(Arquivo[] arquivos) {
        for (Arquivo arquivo : arquivos) {
            if (arquivo instanceof Docx)
                ((Docx) arquivo).salvarDocx();
            else if (arquivo instanceof Txt)
                ((Txt) arquivo).salvarTxt();
        }
    }
}
```

Open/closed principle (OCP)

Exemplo: Gerenciar arquivos de diversos formatos.



Open/closed principle (OCP)

```
public abstract class Arquivo {

    public abstract void abrir();
    public abstract void fechar();
    public abstract void salvar();
}


public class Docx extends Arquivo {

    public void abrir() { }
    public void fechar() { }
    public void salvar() { }
}
```

```
public class Pdf extends Arquivo {

    public void abrir() { }
    public void fechar() { }
    public void salvar() { }
}

public class GerenciadorDeArquivos {

    public salvartodos(Arquivo[] arquivos) {
        for (Arquivo arquivo : arquivos) {
            arquivo.salvar();
        }
    }
}
```

Open/closed principle (OCP)

Ideia geral do OCP

Se a classe Shulambs foi escrita pelo desenvolvedor A, e o desenvolvedor B precisa de alguma modificação nessa classe, ele deveria conseguir fazer isso facilmente estendendo a classe Shulambs, mas não modificando a mesma.

Problemas causados por violar o OCP

Ao adicionar um novo comportamento em uma classe podemos gerar um erro em uma funcionalidade já existente.

Liskov substitution principle (LSP)

Liskov substitution principle (LSP)

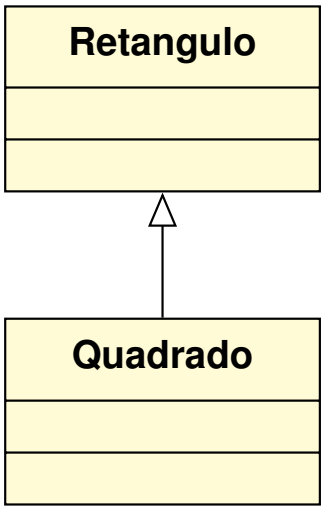
Se $q(x)$ é uma propriedade demonstrável dos objetos x de tipo T. Então $q(y)$ deve ser verdadeiro para objetos y de tipo S, onde S é um subtipo de T[a].

[a] *LISKOV, Barbara: Data abstraction and hierarchy, 1987*

- Objetos que fazem parte de um programa podem ser substituídos por instâncias de seus subtipos sem prejuízo para a correção do programa.

Liskov substitution principle (LSP)

- Violação comum do LSP: verificação de tipo em tempo de execução.
- Violação sutil: uso da herança desconsiderando o comportamento.
  - Ex: `Quadrado` *é um tipo de* `Retângulo`.



Liskov substitution principle (LSP)

```java
public class Retangulo {

  private double altura;
  private double largura;

  public double getArea() {
    return altura * largura;
  }

  public void setAltura(double altura) {
    if (altura > 0)
      this.altura = altura;
  }

  public void setLargura(double largura) {
    if (largura > 0)
      this.largura = largura;
  }
}
```

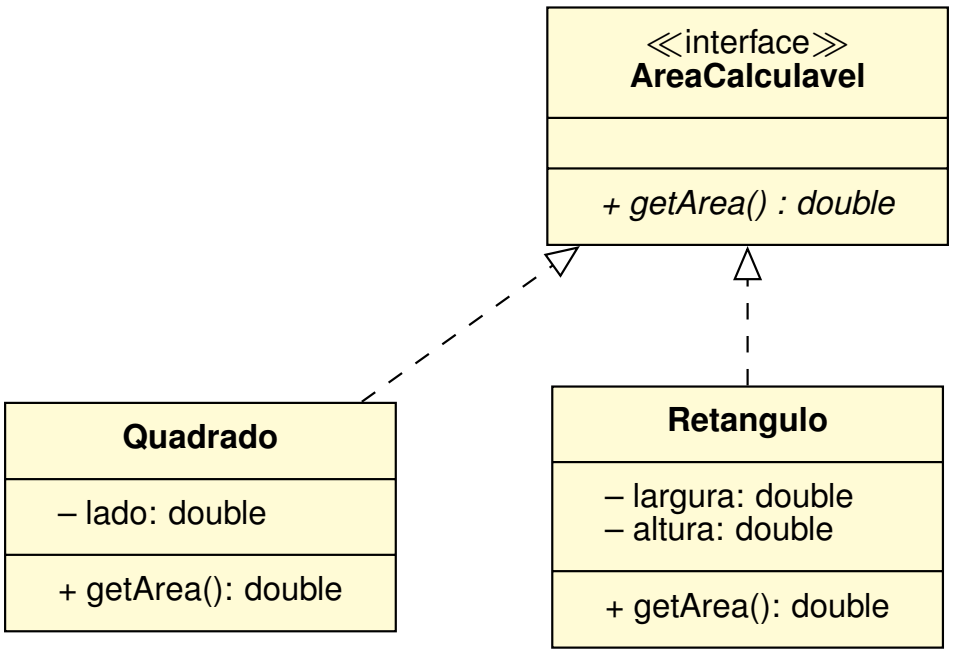
```java
public class Quadrado extends Retangulo {

  @Override
  public void setAltura(double altura) {
      super.setLargura(altura);
      super.setAltura(altura);
  }


  @Override
  public void setLargura(double largura) {
      super.setLargura(largura);
      super.setAltura(largura);
  }

}
```

Liskov substitution principle (LSP)

```
public class Aplicacao {
   public static void main(String[] args) {

      Retangulo ret = new Retangulo();
      ret.setaltura(3);
      ret.setLargura(2);

      if (ret.getArea() != 6) {
         System.out.println("Algo está errado!");
      }

      ret = new Quadrado();
      ret.setaltura(3);
      ret.setLargura(2);

      if (ret.getArea() != 6) {
         System.out.println("Algo está errado!");
      }
   }
}
```

Em resumo: Herança deve considerar a relação *"é-um"* para o comportamento do objeto.

Liskov substitution principle (LSP)



Liskov substitution principle (LSP)

Ideia geral do LSP

Uma classe derivada deve ser substituível por sua classe base.

Problemas causados por violar o LSP

- Retornos inesperados quando usamos polimorfismo.

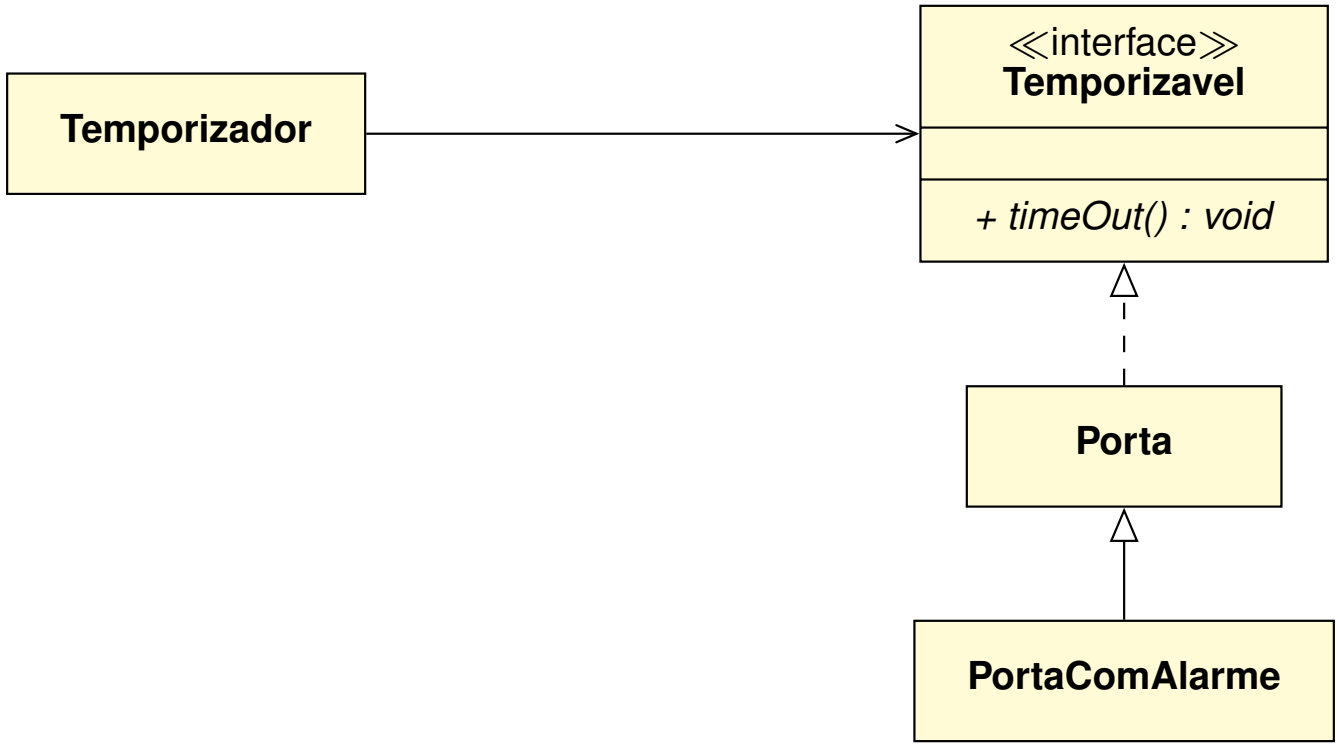
Interface segregation principle (ISP)

Interface segregation principle (ISP)

Clientes não devem ser forçados a implementar uma interface que não usam, ou seja, não devem ser forçados a depender de métodos que não usam.

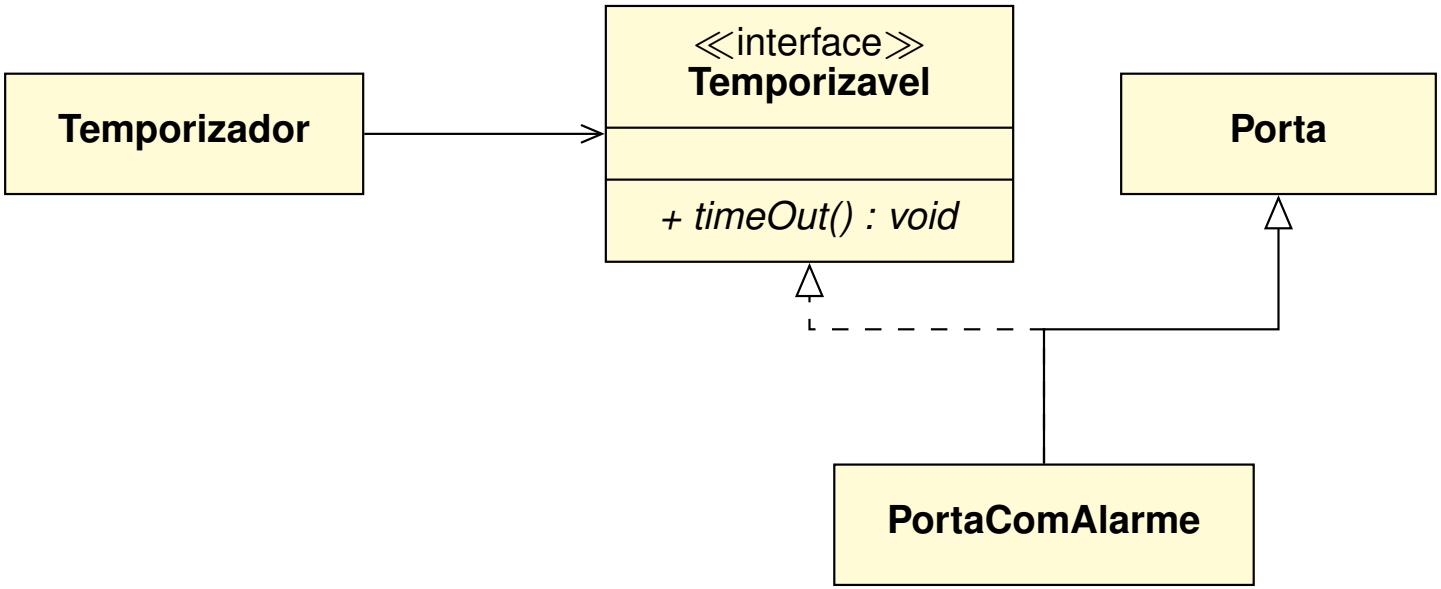
- O acúmulo de funções em uma interface pode torná-la não-coesa e forçar comportamentos não comuns a uma classe.
- Exemplo:
  - Uma `Porta` e um `Temporizador` (*timer*).
  - Uma porta de geladeira que deve disparar um alarme se ficar mais de 20s aberta.

Interface segregation principle (ISP)

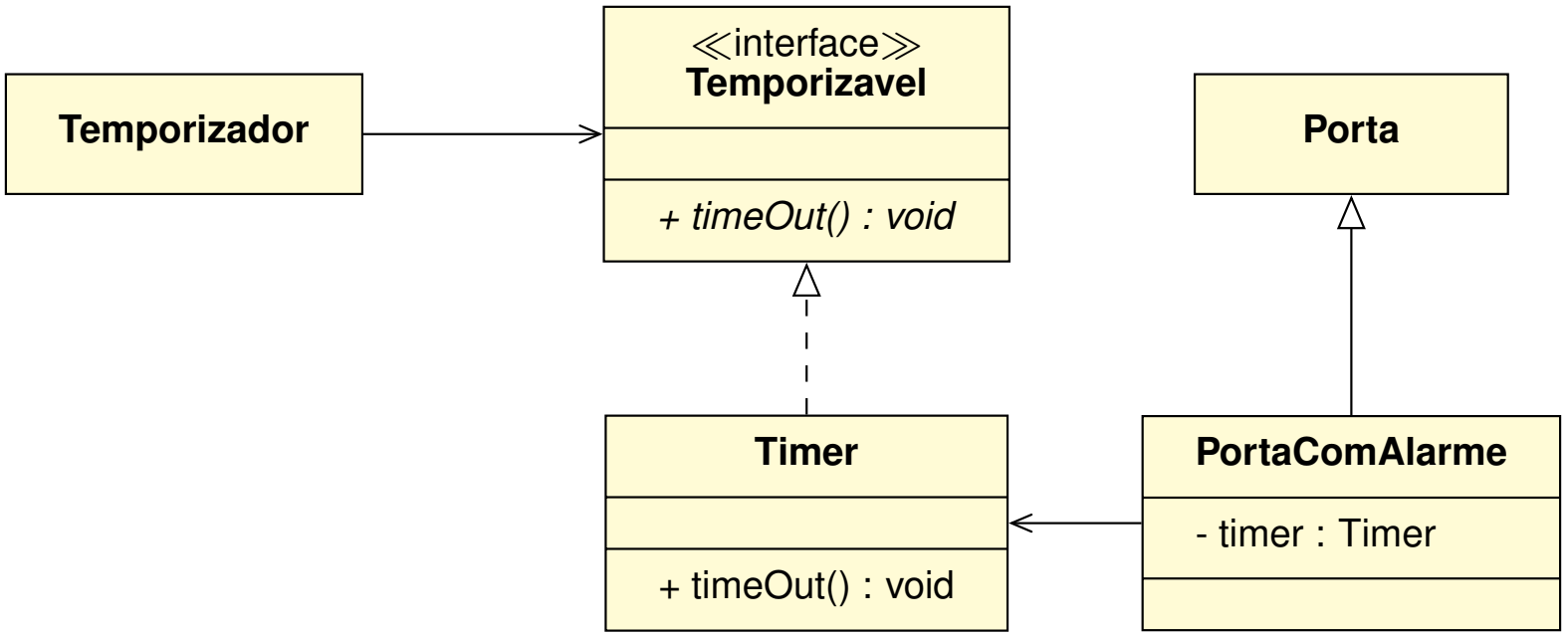


Problema: nem toda `Porta` é `Temporizável`.

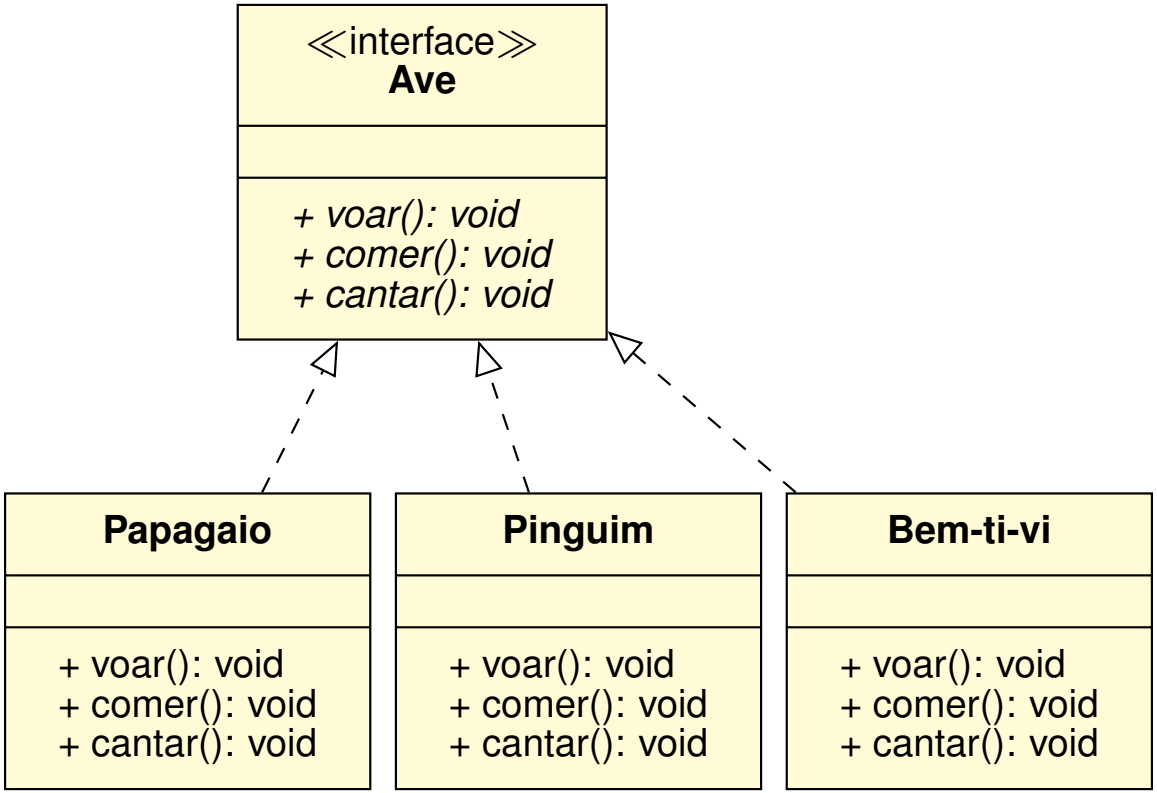
Interface segregation principle (ISP)



Interface segregation principle (ISP)

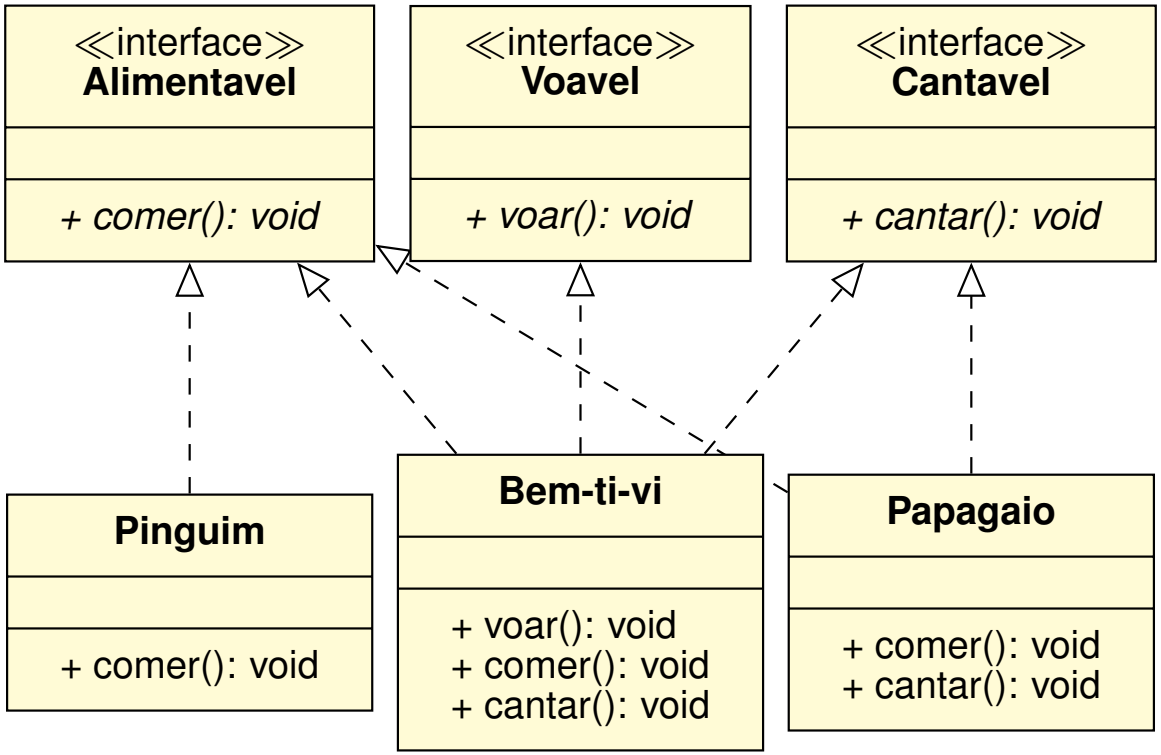


Interface segregation principle (ISP)



Problema: nem toda `Ave` é capaz de `voar` e `cantar`.

Interface segregation principle (ISP)



Interface segregation principle (ISP)

Ideia geral do ISP

Uma classe não deve ser forçada a implementar métodos que não irão utilizar.

Dependency inversion principle (DIP)

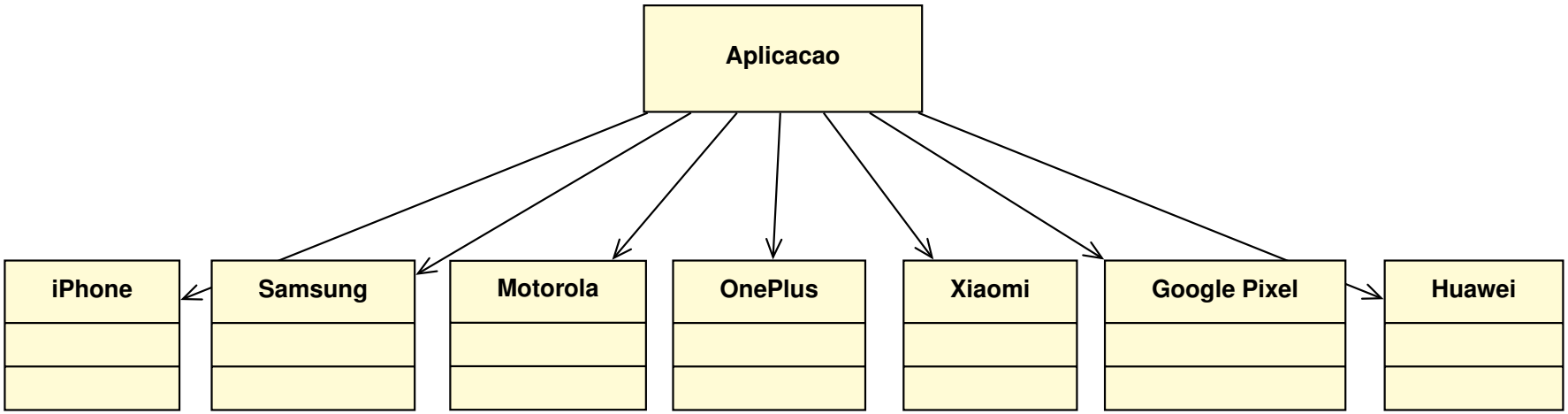
Dependency inversion principle (DIP)

"Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações; Abstrações não devem depender de detalhes. Detalhes devem depender de abstrações.".

- Nenhuma variável deve conter referência para uma classe concreta.
- Nenhuma classe deve derivar de uma classe concreta.
- Nenhum método deve sobrescrever métodos implementados em sua classe base.

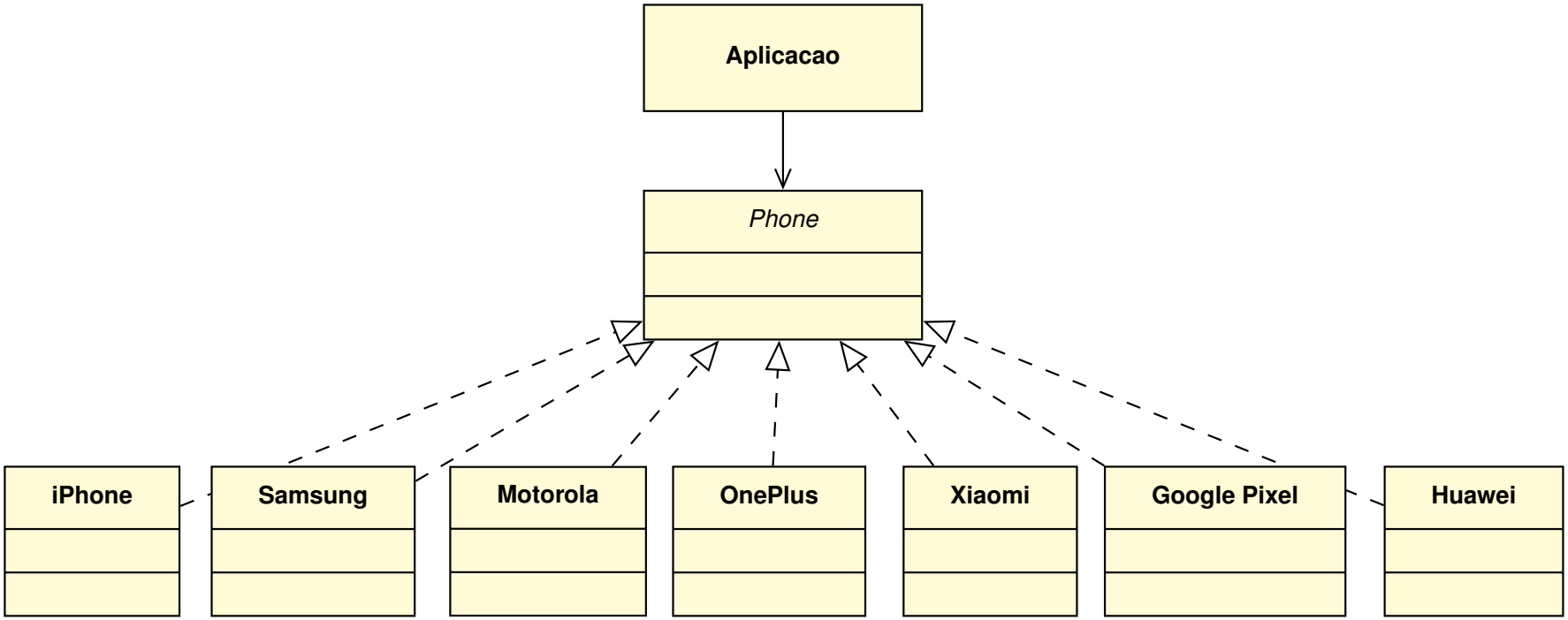
Dependency inversion principle (DIP)

Exemplo: Um sistema de uma loja que vende telefones (*viola o DIP*).



Dependency inversion principle (DIP)

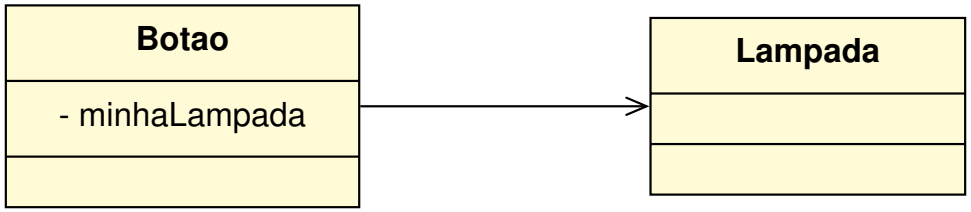
Exemplo: Um sistema de uma loja que vende telefones.



Dependency inversion principle (DIP)

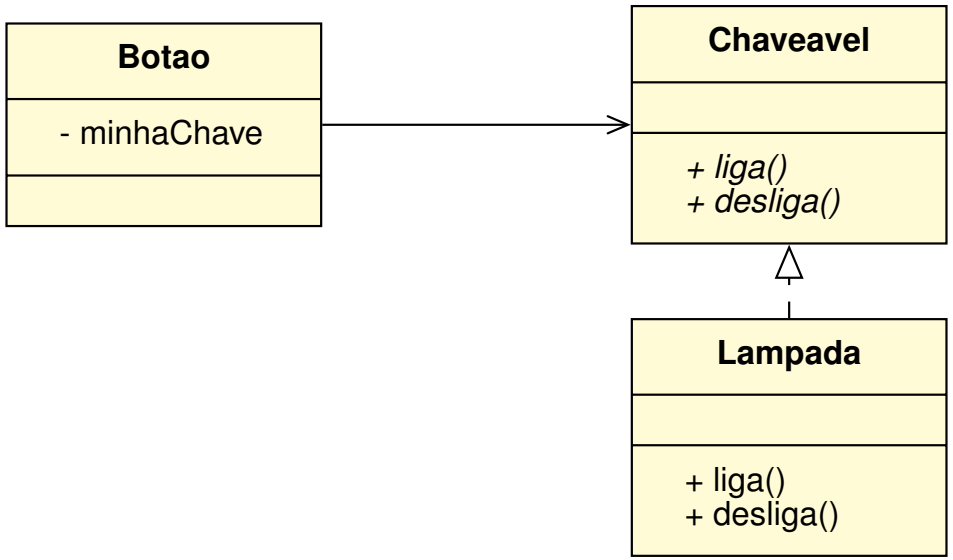
Exemplo: Um botão para controlar uma lâmpada (viola o DIP).

- E se a `Lâmpada` muda?
- E se queremos usar o botão para controlar um microondas?



Dependency inversion principle (DIP)

Exemplo: Um botão para controlar uma lâmpada.



Dependency inversion principle (DIP)

Ideia geral do DIP

Dependa de abstrações e não de implementações.

Problemas causados por violar o DIP

- Alto acoplamento

Obrigado!!

Muito obrigado pela atenção! Alguma dúvida? Bora praticar!!!

"*Paz, pão e terra.*"