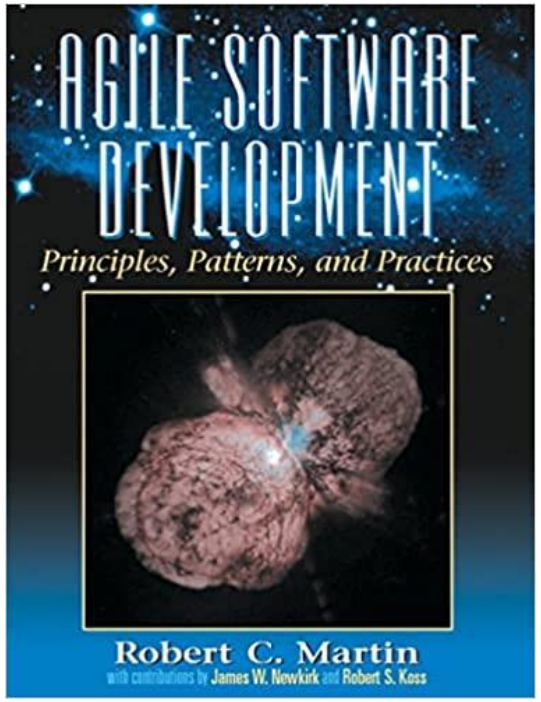
# ARQUITETURA DE SOFTWARE

## Design Orientado a Objetos (OOD)

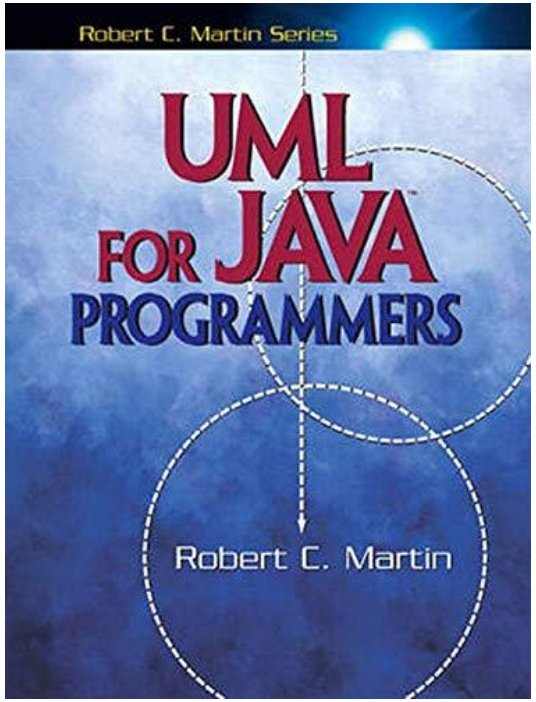
Geucimar Briatore
geucimar@up.edu.br

Atualizado em 10/2022

# Literatura pós Manifesto Ágil




2002 (Robert C. Martin)




2003 (Robert C. Martin)

butunclebob.com

# The Principles of OOD

# O “cheiro” do design ruim (Design smells)

- **Repetição de código**: Repetições características de copiar e colar (ctrl-c + ctrl-v);
- **Complexidade desnecessária**: Estruturas de código inteligentes que não são necessárias no momento, mas que podem ser úteis algum dia;
- **Rigidez**: O sistema é difícil de mudar porque cada vez que você muda uma coisa, tem que mudar outra coisa em uma sucessão interminável de mudanças;
- **Fragilidade**: Uma mudança em uma parte do sistema faz com que ele quebre em outras partes não relacionadas;
- **Imobilidade**: Não permite componentizar para reutilizar partes do sistema;
- **Viscosidade**: É difícil refatorar o código, tudo leva uma eternidade. O programador prefere fazer hacks/POGs (Programação Orientada a Gambiarras).

# Princípios fundamentais (SOLID)

**S**ingle Responsiblity Principle (Princípio da responsabilidade única):
**Uma classe deve ter um, e somente um, motivo para ser alterada**.

**O**pen-Closed Principle (Princípio aberto-fechado):
**Objetos devem estar abertos para extensão, mas fechados para modificação**.

**L**iskov Substitution Principle (Princípio da substituição de Liskov):
**Uma classe derivada deve ser substituível por sua classe base**.

**I**nterface Segregation Principle (Princípio da segregação de interface):
**Os clientes não devem depender de métodos que não usam**.

**D**ependency Inversion Principle (Princípio da inversão da dependência):
**Dependa de interfaces ou classes abstratas ao invés de classes concretas**.

# Revisão: interfaces, classes abstratas e concretas

- A **interface** é a especificação mais abstrata possível de uma classe. Ela não permite realizar nenhuma implementação nos métodos;
- A **classe abstrata** serve para especificar algumas partes e implementar outras para a classe concreta. Uma classe abstrata postergará parte de sua implementação, ou toda, para métodos definidos nas subclasses; portanto assim como a interface, uma classe abstrata não pode ser instanciada. Os métodos que uma classe abstrata declara, mas não implementa, são chamados de métodos abstratos;
- Os objetos são criados através da instanciação de uma **classe concreta** que implementa atributos e métodos. O processo de instanciar uma classe aloca espaço na memória para este objeto; as classes que não são abstratas são chamadas de classes concretas.

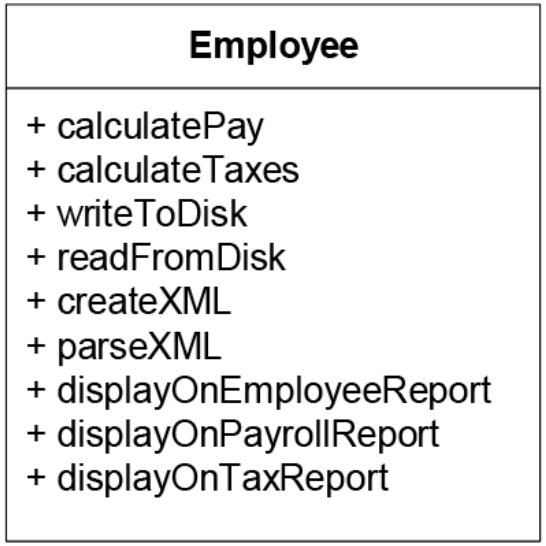
# 1. Single Responsibility Principle (SRP)

**Uma classe deve ter um, e somente um, motivo para ser alterada.**

# Efeitos da violação da responsabilidade única

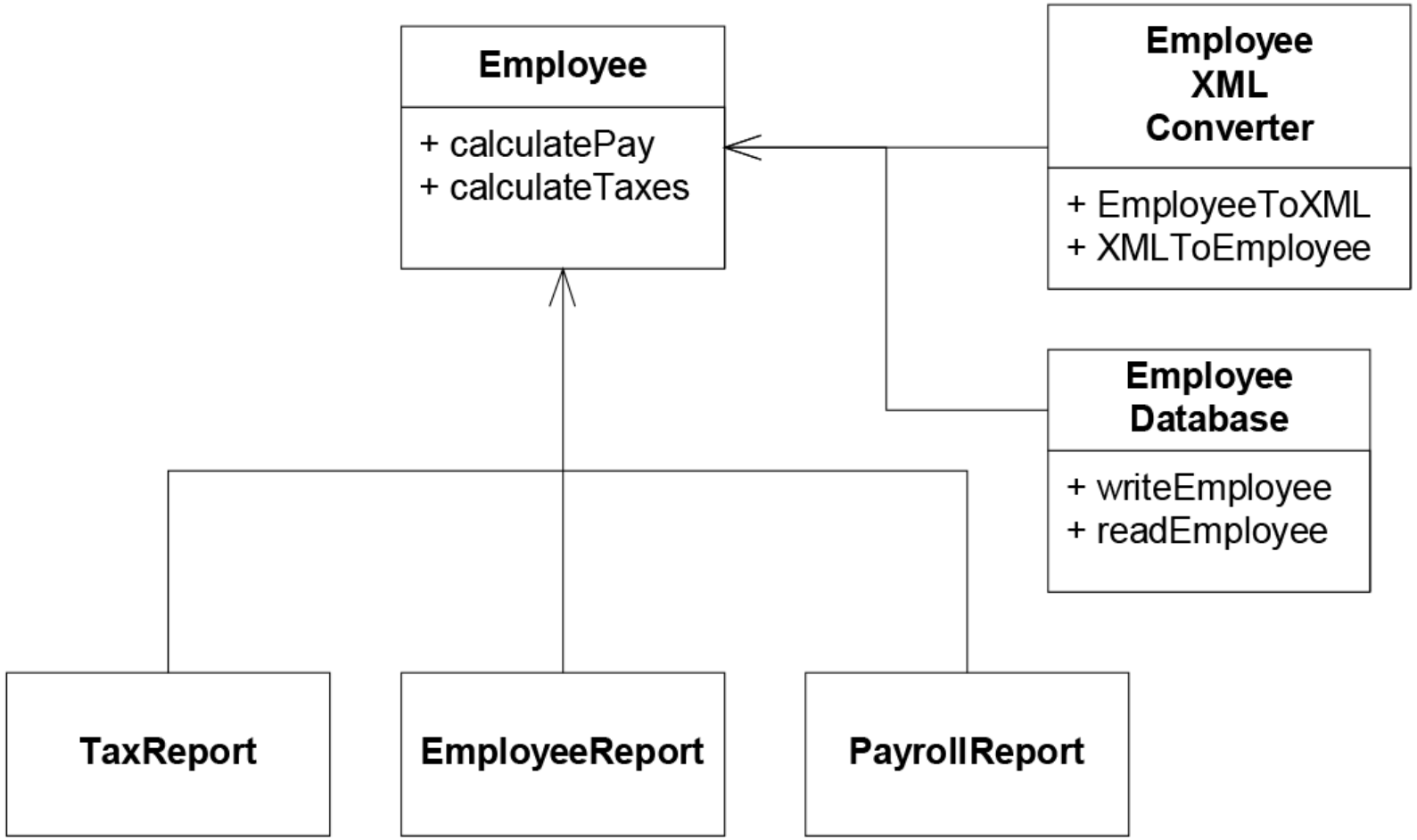
- Falta de coesão: a classe não deve assumir responsabilidades que não e sua;
- Alto acoplamento: responsabilidades geram um maior nível de dependências, deixando o sistema engessado e frágil para alterações;
- Dificuldades na implementação de testes automatizados;
- Dificuldades para reutilização de código.

# Classe com muitas responsabilidades




```
public class Employee {
  public double calculatePay();
  public double calculateTaxes();
  public void writeToDisk();
  public void readFromDisk();
  public String createXML();
  public void parseXML(String xml);
  public void displayOnEmployeeReport(
                          PrintStream stream);
  public void displayOnPayrollReport(
                          PrintStream stream);
  public void displayOnTaxReport(
                          PrintStream stream);
}
```

# Separação de interesses



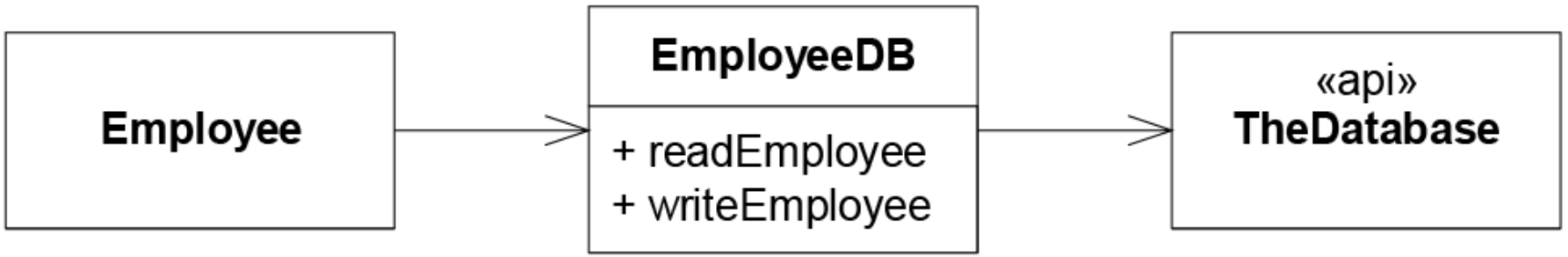
## 2. Open-Closed Principle (OCP)

**Objetos ou entidades devem estar abertos para extensão, mas fechados para modificação.**

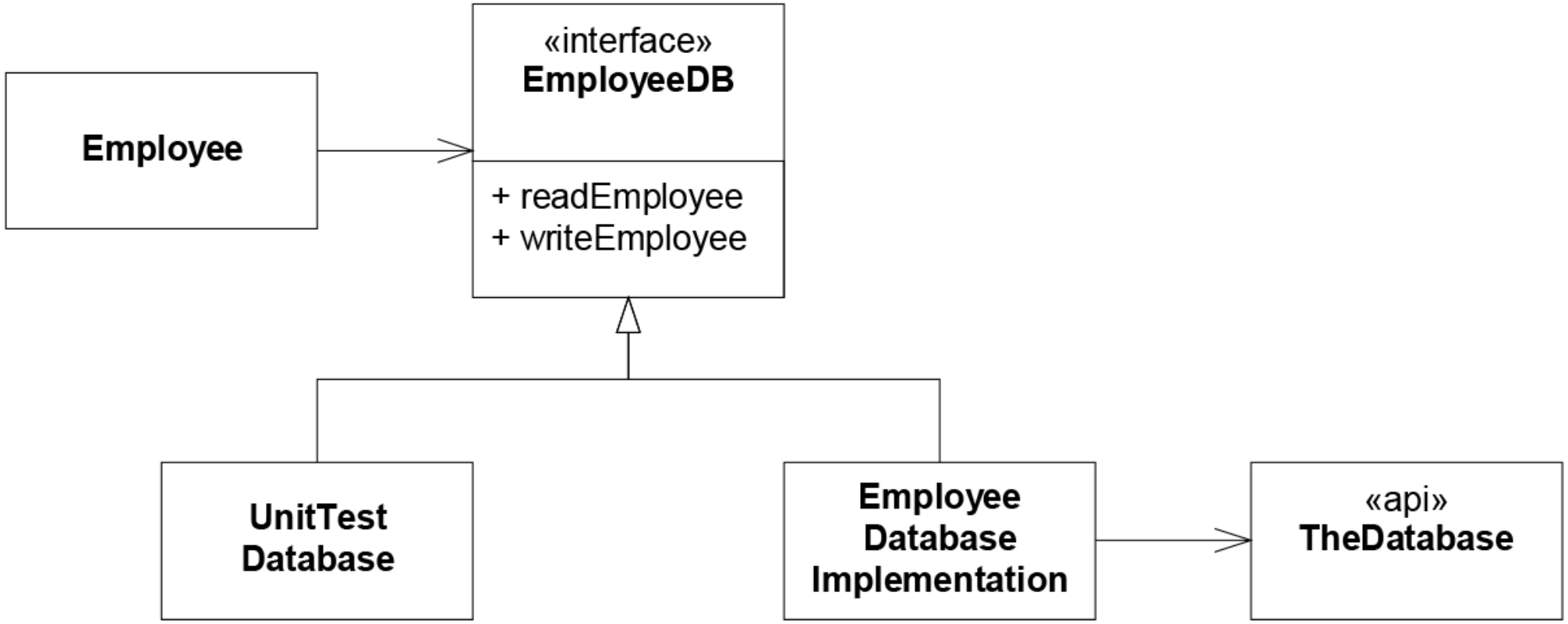
# Efeitos da violação da OCP

- Dificuldade para adicionar novas funcionalidades;
- Não é possível incluir novos recursos sem alterar o código original.

# Violação da OCP



# Refatoração de acordo com a OCP



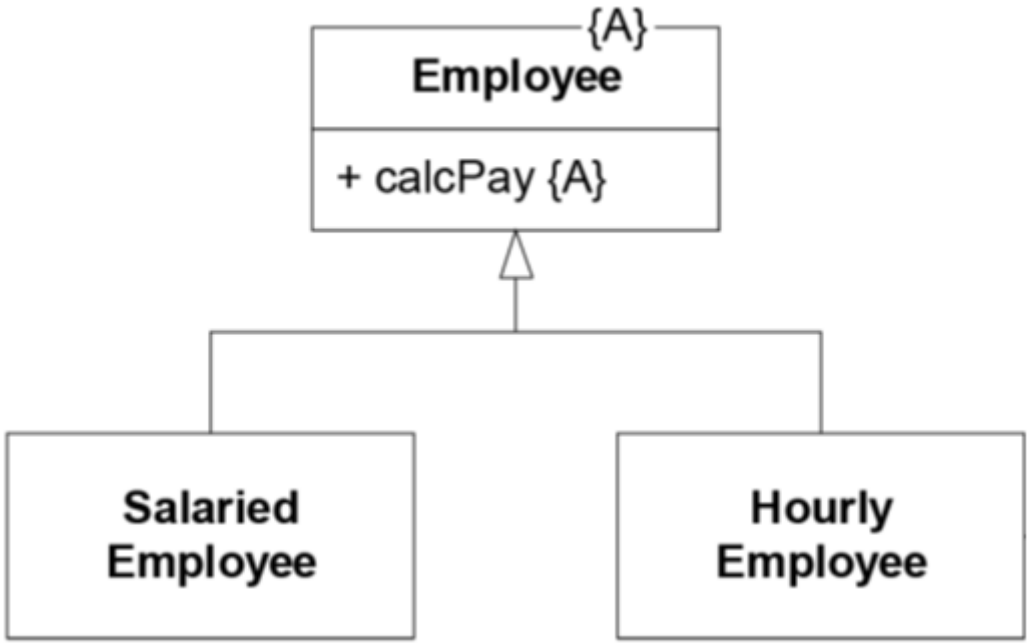
## 3. Liskov Substitution Principle (LSP)

**Uma classe derivada deve ser substituível por sua classe base.**

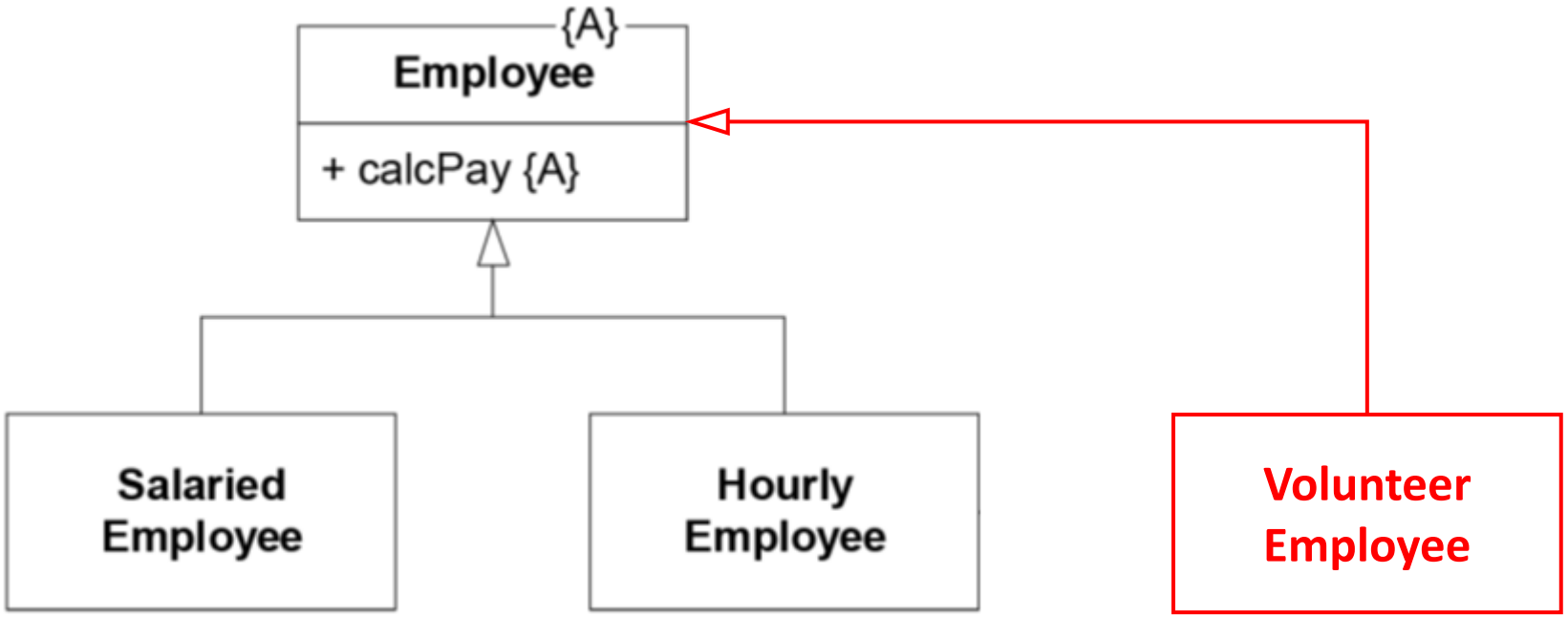
# Efeitos da violação da LSP

- Implementar métodos que não fazem nada;
- Retornar valores inconformes com a classe base;
- Lançar exceções inesperadas.

# Exemplo LSP



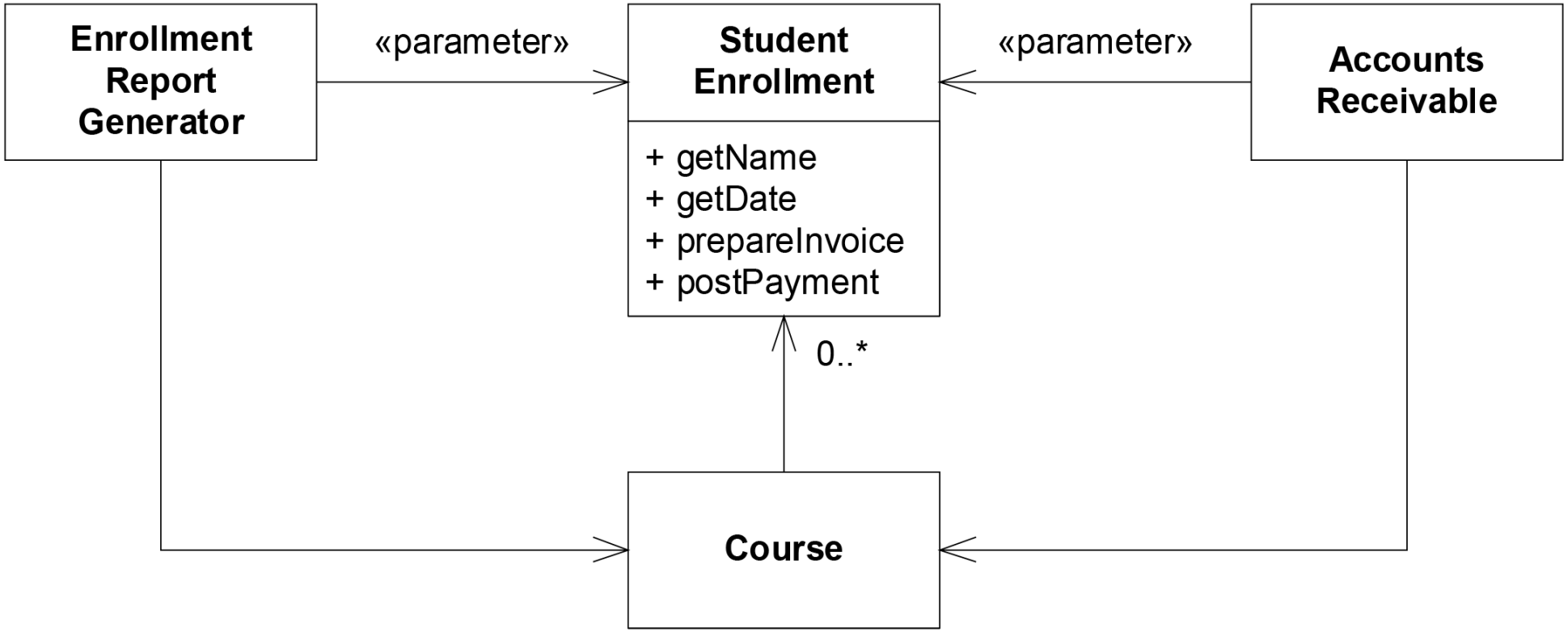
# Violação da LSP



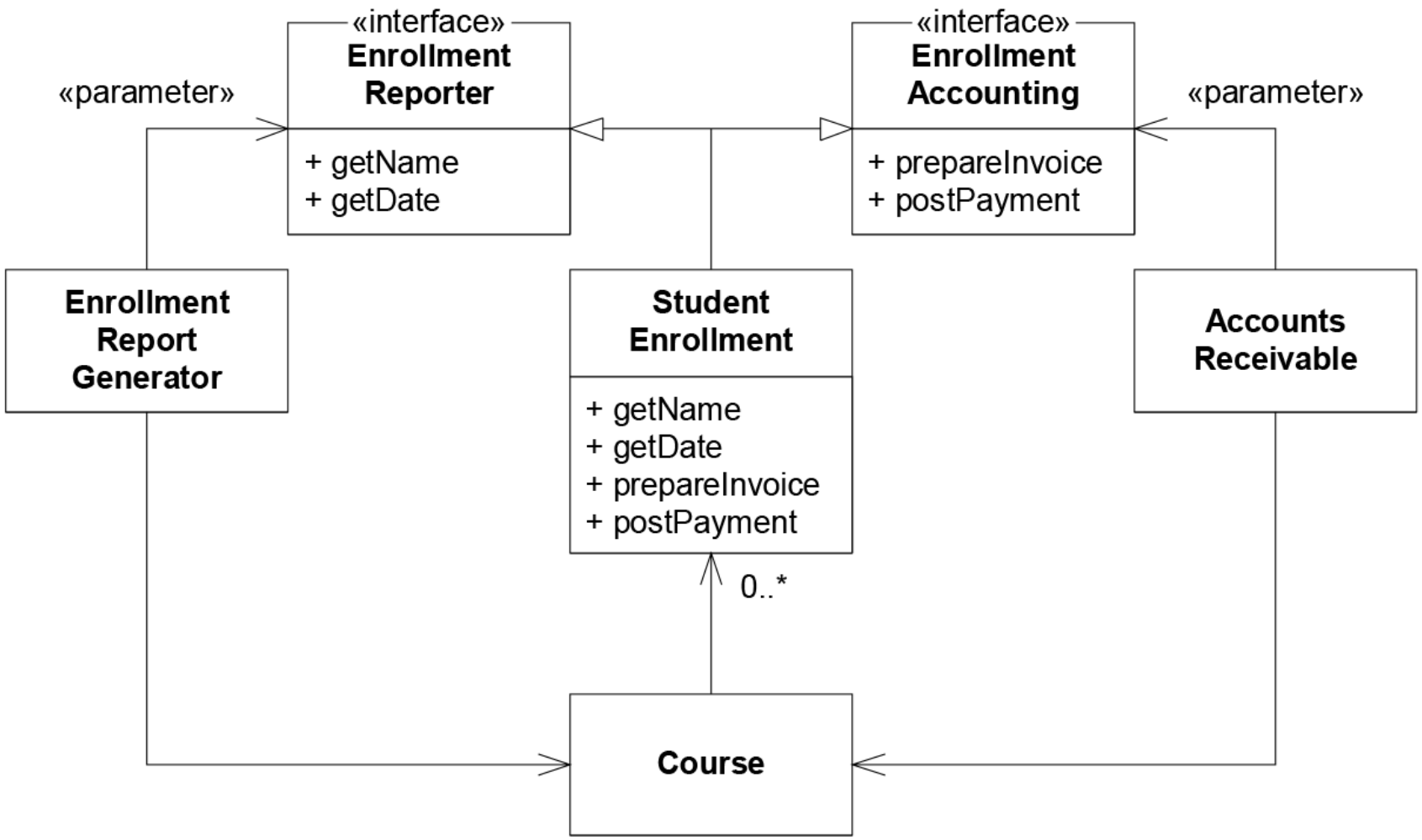
# 4. Interface Segregation Principle (ISP)

**Os clientes não devem depender de métodos que não usam.**

# Exemplo de um sistema não segregado (isolado)



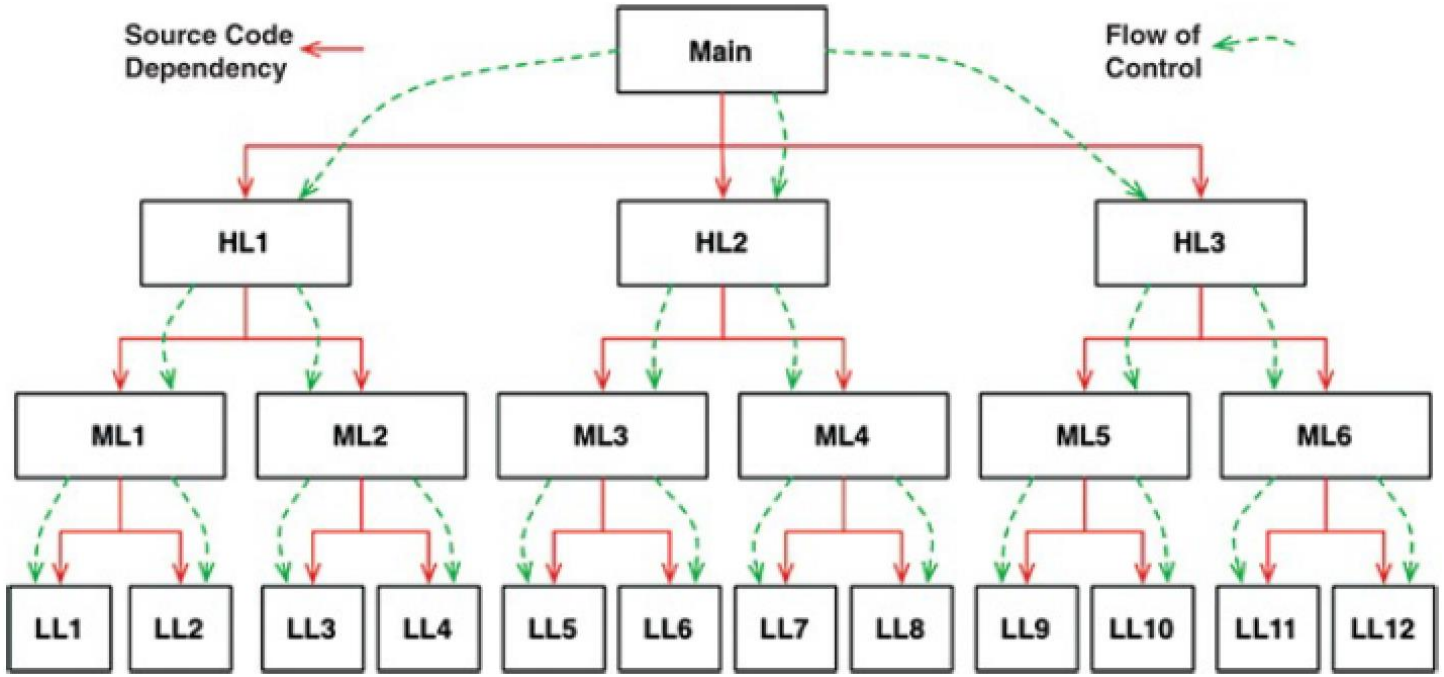
# Segregação de dependências com interfaces



## 5. Dependency Inversion Principle (DIP)

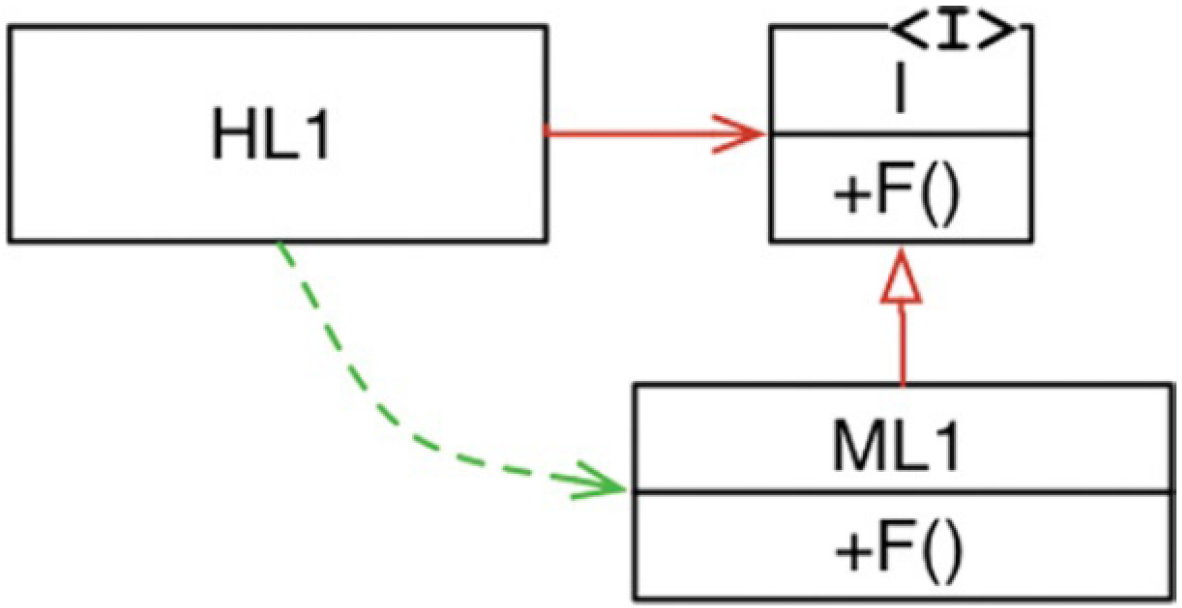
**Dependa de interfaces ou classes abstratas ao invés de classes concretas.**

# Fluxo comum de controle e dependências




Na **árvore de típica de dependências**, as funções principais são chamadas de alto nível (HL), que chamam as de nível médio (ML), que chamam as de baixo nível (LL). Neste caso, as chamadas das dependências seguem o fluxo de controle.

# Inversão de dependências (DIP)




Observe que a dependência de código-fonte (o relacionamento de herança) entre ML1 e a interface I aponta na direção oposta em relação ao fluxo de controle. Isso é chamado de **inversão de dependência** e suas implicações na arquitetura de software são profundas.

# Injeção de dependências (DI)