Código Limpo e Métricas de Código-Fonte

Paulo Meirelles

paulormm@ime.usp.br

Com que frequência as pessoas lavam os pratos na cozinha de um restaurante?

A próxima refeição levará mais tempo e será mais cara porque estamos limpando a cozinha

A funcionalidade que você está solicitando é cara (porque o código não está limpo e precisaremos alterá-lo)

Seu código-fonte é assim?

É melhor limpar aos poucos?

Ou deixar acumular sujeira e bagunça?

Algumas sujeiras são difíceis de limpar com o passar do tempo

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloImasters extends MIDlet implements Comman
  private fi
  private fina
  public Hello
    textBox = new
    textBox.addCommand(
    textBox.setCommandL
  }
  public void startApp() {
    Displayable telaAtual = Di
    if(telaAtual == null) {
      Display.getDisplay(this).setCu
    }
  }
  public void pauseApp() { }
  public void destroyApp(boolean b) { }
  void sair() {
    destroyApp(false);
    notifyDestroyed();
  }
  public void commandAction(Command c, Displayable d) {
    if (c == sairCommand) {
      sair();
    }
  }
}
```

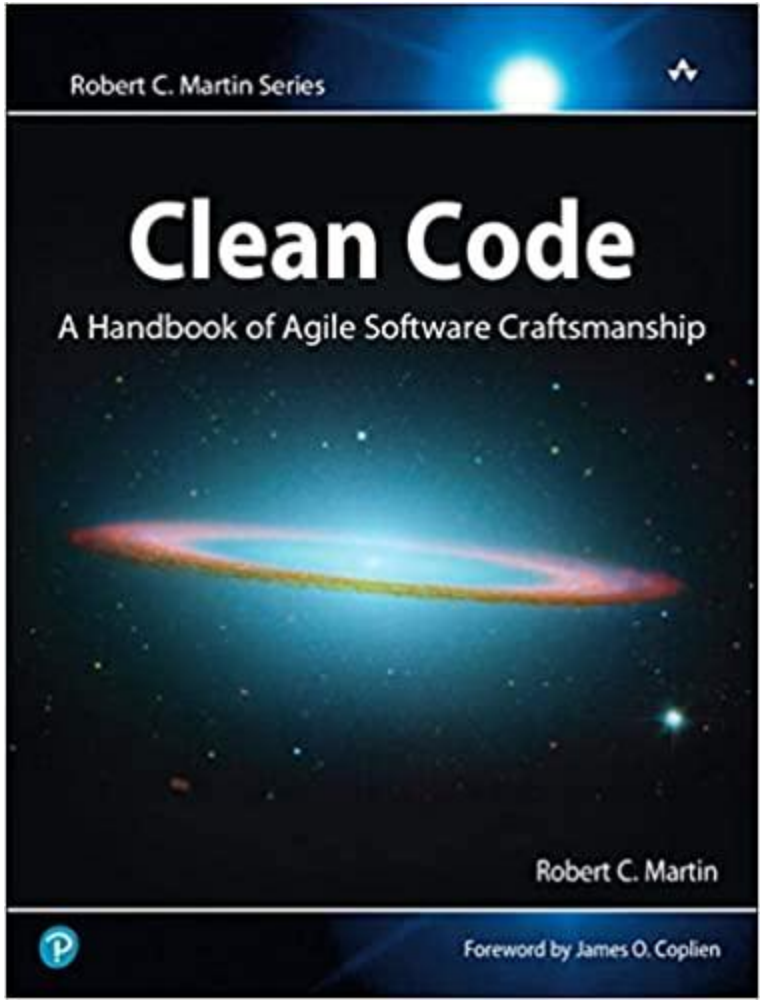
Como limpar o código?

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloImasters extends MIDlet implements Comman
  private fi
  private fina

  public Hello
    textBox = new
    textBox.addCommand(
    textBox.setCommandL
  }

  public void startApp() {
    Displayable telaAtual = Di
    if(telaAtual == null) {
      Display.getDisplay(this).setCu
    }
  }

  public void pauseApp() { }

  public void destroyApp(boolean b) { }

  void sair() {
    destroyApp(false);
    notifyDestroyed();
  }

  public void commandAction(Command c, Displayable d) {
    if (c == sairCommand) {
      sair();
    }
  }
}
```

Mas, o que é um **Código Limpo**?

Agenda

- *Código Limpo*
  - *Nomes expressivos*
  - *Funções limpas*
- *Coesão*
- *Acoplamento*
- *Métricas de Código-fonte*



Robert Martin, entrevistou grandes ***especialistas em desenvolvimento de software****, questionando-os quanto a uma definição para código limpo. Cada um dos entrevistados elaborou respostas diferentes, destacando características subjetivas, como elegância, facilidade de alteração e simplicidade, e outras puramente técnicas, incluindo a falta de duplicações, presença de testes de unidade e de aceitação e a minimização do número de entidades.*

O que é código limpo?

Bjarne Stroustrup
Inventor do C++

*“Gosto que meu código seja **elegante** e **eficiente**. A lógica deve ser **direta** para dificultar a ocultação de bugs, as **dependências mínimas** para facilitar a manutenção, **tratamento de erros** completo de acordo com uma estratégia articulada e **desempenho próximo do ideal** para não tentar as pessoas a bagunçar o código com otimizações sem princípios. O código limpo **faz uma coisa bem**.”*

O que é código limpo?

Grady Booch
Autor do Livro “Object Oriented Analysis and Design with Applications”

*“Código limpo é **simples** e **direto**. O código limpo é **lido como uma prosa bem escrita**. O código limpo nunca obscurece a intenção do designer, mas está cheio de abstrações nítidas [claramente definidas] e linhas **diretas** de controle”.*

O que é código limpo?

Dave Thomas
Fundador da OTI,
estrategística do projeto Eclipse

*"O código limpo pode ser **lido e aprimorado** por um desenvolvedor que não seja seu autor original. Possui **testes unitários e de aceitação**. Tem **nomes significativos**. Ele fornece **uma maneira** em vez de muitas maneiras de fazer uma coisa. Possui **dependências mínimas**, que são explicitamente definidas e fornece u**ma API clara e mínima**. O código deve ser **literal**, pois, dependendo do idioma, nem todas as informações necessárias podem ser expressas claramente apenas no código."*

O que é código limpo?

Michael Feathers
Author of Working Effectively
With Legacy Code

“Eu poderia listar todas as qualidades que noto no código limpo, mas há uma qualidade abrangente que leva a todas elas. Código limpo sempre parece que foi ***escrito por alguém que se importa****. Não há* ***nada óbvio que você possa fazer para torná-lo melhor****. Todas essas coisas foram pensadas pelo autor do código e, se você tentar imaginar melhorias, será levado de volta para onde está, apreciando o código que alguém deixou para você -* ***código deixado por alguém que se preocupa profundamente com o arte.****”*

O que é código limpo?

Ron Jeffries
Autor do “Extreme Programming Installed”

“Em ordem de prioridade, código limpo e simples:

Executa todos os testes

Não contém duplicação

Expressa todas as ideias de design *que estão no sistema*

Minimiza o número de entidades*, como classes, métodos, funções e similares.*

O que é código limpo?

Ward Cunningham
Inventor do Wiki,
pioneiro nos padrões de design e
Extreme Programming

“Você sabe que está trabalhando em um código limpo quando ***cada rotina que você lê acaba sendo exatamente o que você esperava.*** *Você pode chamá-lo de código bonito quando os códigos também* ***fazem parecer que a linguagem foi feita para o problema****.”*

Simples

Eficiente

Direto

Expressivo

Não contém duplicação

Acontece que é o que você esperava

Executa todos os testes

Cheio de significado

Literal

Boa leitura

Belo: quando a linguagem foi feita para o problema

Mínimo

Código Limpo: Nomes expressivos

- Um programa é basicamente composto de palavras reservadas e nomes
- Escolher bons nomes toma tempo; no entanto, isso acaba economizando mais tempo ainda
- Os nomes devem ser expressivos e eliminar dúvidas

Código Limpo: Nomes expressivos

```
public List<int[]> getThem() {

    List<int[]> list1 = new ArrayList<int[]>();

    for (int[] x : theList)
        if(x[0] == 4)
                list1.add(x);
    return list1;
}
```

- O que esse método seleciona?
- Quais tipos de coisa estão em theList?
- Qual a relevância da posição zero?
- O que significa “4”?

Código Limpo: Nomes expressivos

```
public List<int[]> getFlaggedCells() {
   List<int[]> flaggedCells = new ArrayList<int[]>();
   for (int[] cell : gameBoard)
       if(cell[STATUS_VALUE] == FLAGGED)
              flaggedCells.add(cell);
   return flaggedCells;
}
```

- Seleciona as células marcadas
- theList é um tabuleiro com células!
- Posição zero é o status
- “4” significa “marcado”

Nomes expressivos

```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if(cell.isFlagged())
                flaggedCells.add(cell);
    return flaggedCells;
}
```

Código Limpo: Nomes expressivos

- **Seja claro e consistente**
  - Use nomes legíveis
  - É melhor que os nomes sejam relativamente curtos, desde que sejam claros
  - Economizar alguns toques no teclado não é uma boa razão para promover a obscuridade
  - Use nomes que possam ser usados em buscas
  - Use os padrões da linguagem (camelCase etc.)

Código Limpo: Nomes expressivos

- **Comunique bem o que você quer dizer**
  - Use nomes do domínio da solução
    - Padrões de projeto, algoritmos, termos matemáticos etc.
  - Use nomes do domínio do problema
    - Atores envolvidos, tipos de produtos etc.
  - Não confunda o leitor
    - Uma palavra para cada conceito (get, fetch, retrieve)
    - Evite piadinhas ou metáforas de significado obscuro

Código Limpo: Funções (métodos)

- O que a função faz **deve ser óbvio**
  - Permite entender o problema

- A implementação da função **deve ser simples**
  - Permite entender a solução

Código Limpo: Funções (métodos)

- Uma função ↔ uma abstração
  - Uma função faz uma e apenas uma coisa (corretamente!)

- Evitar efeitos colaterais
  - OU modifica um objeto OU devolve alguma coisa

Código Limpo: Funções (métodos)

- DRY: Don’t Repeat Yourself
  - Se um trecho de código aparece em mais de um lugar, há uma abstração implícita
    - E, portanto, as funções que usam esse trecho fazem mais de uma coisa!

Código Limpo: Funções (métodos)

- Uma função que faz apenas uma coisa não pode ser dividida em seções
  - Se não é possível extrair uma sub-função de dentro dela cujo nome não seria o mesmo que o nome que ela já tem, ela provavelmente faz só uma coisa

Código Limpo: Funções (métodos)

```
public void pay() {
    for (Employee e : employees) {
        if (e.isPayday()) {
            Money pay = e.calculatePay();
            e.deliverPay(pay);
        }
    }
}
```

- **Faz mais que uma coisa!**
- Itera por todos os empregados
- Checa para ver quais empregados precisam ser pagos
- Paga os empregados

```
public void pay() {
    for (Employee e : employees)
        payIfNecessary(e);
}


private void payIfNecessary(Employee e) {
    if (e.isPayday())
        calculateAndDeliverPay();
}


private void
calculateAndDeliverPay(Employee e) {
    Money pay = e.calculatePay();
    e.deliverPay(pay);
}
```

Será um exagero?

Cada função é mais simples, mas zilhões de funções também são um aumento na complexidade

Código Limpo: Funções (métodos)

- A implementação da função deve ser simples
- Funções “pequenas”
- Um único nível de abstração
  - Quando detalhes se misturam a conceitos mais abstratos, mais e mais detalhes tendem a se insinuar
    - É o primeiro passo rumo às funções gigantes

Código Limpo: Funções (métodos)

- Evitar estruturas aninhadas
- Blocos de if's/while's/else's devem ser diretos no que fazem (provavelmente, apenas chamar uma função)
- Condicionais provavelmente devem ir para uma função separada

Código Limpo: Funções (métodos)

```
def primosAte(N) :
  criaInteirosDesmarcadosAte(N)
  marcaMultiplos( )
  colocaNaoMarcadosNoResultado( )

  retorna resultado
```

Crivo de Erastótenes (primos de 1 a N)

- Criar conjunto de elementos que represente os inteiros de 1 a N
- Marcar todos números múltiplos de outros
- Coletar todos aqueles que não estão marcados (os primos)

Código Limpo: Funções (métodos)

- O número de argumentos de uma função deve ser pequeno
  - Argumentos dificultam os testes
  - um grande número de argumentos sugere que a função faz mais que uma coisa
  - um grande número de argumentos sugere que a função é usada de maneiras muito díspares

Código Limpo: Funções (métodos)

- O número de argumentos de uma função deve ser pequeno
  - Argumentos que na verdade são “flags” praticamente garantem que a função faz mais que uma coisa
    - rotaciona (int angulo, bool sentidoHorario)
    - rotacionaHorario (int angulo), rotacionaAntiHorario (int angulo)

O que é código limpo?

Um código limpo está inserido em um estilo de programação que busca a proximidade a três valores: **expressividade, simplicidade e flexibilidade**

O que é código limpo?

As **versões iniciais** de um método, classe e outras estruturas **nunca são exatamente uma boa solução**

É necessário tempo e preocupação com cada elemento desde o nome escolhido para uma variável até uma hierarquia de classes

O que queremos em Projeto Orientado a Objetos?

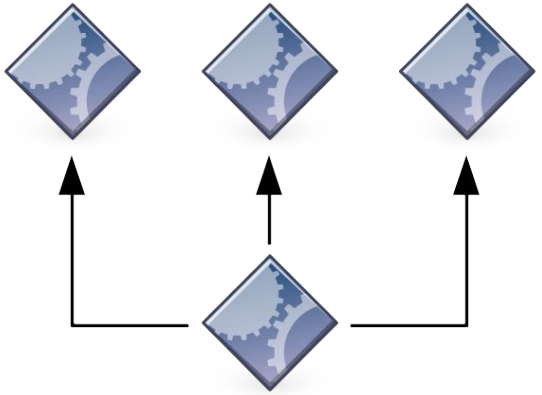
- Limitar a quantidade de informação que o leitor lida ao ler métodos: **classes menores possível**
  - Facilita a leitura e entendimento
  - Auxilia a criar unidades
  - Classes coesas e evitar duplicações

O que queremos em Projeto Orientado a Objetos?

- Sistemas OO devem ser compostos de muitas classes pequenas
  - *é mais fácil encontrar um objeto em muitas gavetas pequenas do que em poucas gavetas grandes e cheias*

O que queremos em Projeto Orientado a Objetos?

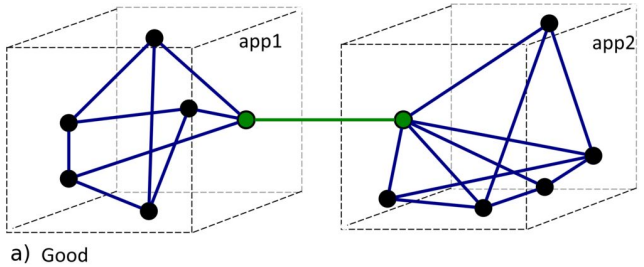
Queremos maximizar a **coesão** das classes e minimizar o **acoplamento** entre essas classes



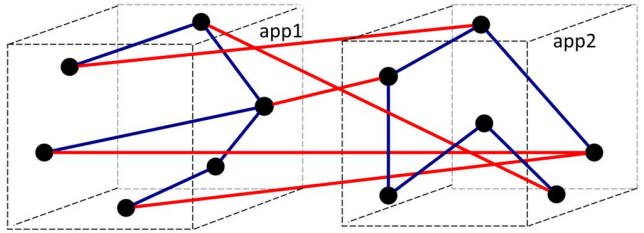
Coesão

Coesão

Toda classe deve implementar uma única funcionalidade ou serviço. Todos os métodos e membros de uma classe devem atuar para implementar essa funcionalidade



a) Good



b) Bad

Coesão

- Facilita a implementação, entendimento e manutenção
- Facilita a alocação de um único responsável por manter uma classe
- Torna a classe testável

```
classe Pilha:
  int maxPosicoes
  int topo
  vetor[maxPosicoes] elementos

  def vazia?():
    return topo == 0

  def cheia?():
    return topo == maxPosicoes

  def insere(Elemento elemento):
    levanta_excecao("Pilha cheia") if cheia?
    elementos[topo] = elemento
    topo += 1

  def remove_topo():
    levanta_excecao("Pilha vazia") if vazia?
    topo -= 1
    elementos[topo]
```

```
classe Pilha:
  int maxPosicoes
  int topo
  vetor[maxPosicoes] elementos

  def vazia?():
    return topo == 0

  def cheia?():
    return topo == maxPosicoes

  def insere(Elemento elemento):
    levanta_excecao("Pilha cheia") if cheia?
    elementos[topo] = elemento
    topo += 1

  def remove_topo():
    levanta_excecao("Pilha vazia") if vazia?
    topo -= 1
    elementos[topo]
```

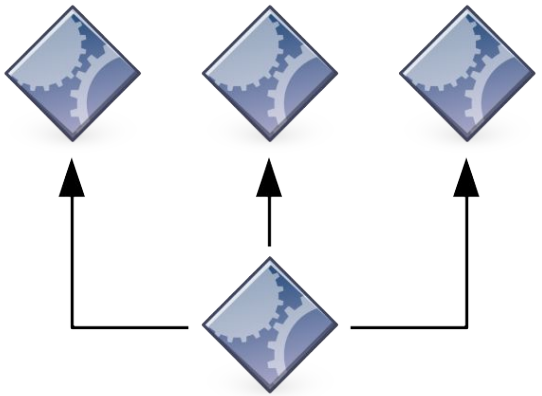
Os métodos da classe Pilha estão **intimamente relacionados** com suas variáveis de instância

```
classe Pilha:
  int maxPosicoes
  int topo
  vetor[maxPosicoes] elementos

  def vazia?():
    return topo == 0

  def cheia?():
    return topo == maxPosicoes

  def insere(Elemento elemento):
    levanta_excecao("Pilha cheia") if cheia?
    elementos[topo] = elemento
    topo += 1

  def remove_topo():
    levanta_excecao("Pilha vazia") if vazia?
    topo -= 1
    elementos[topo]
```

A classe Pilha desempenha funções que tratam apenas do funcionamento de uma pilha

Acoplamento

- É a força de conexão entre duas classes

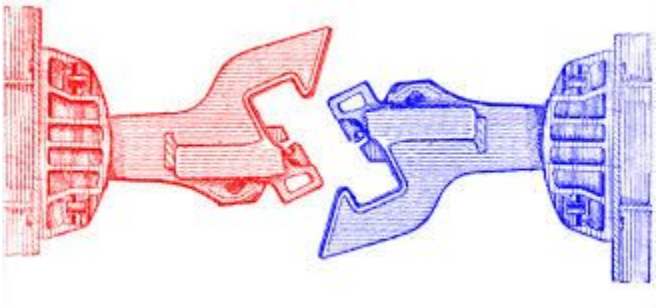


Acoplamento

- Classes não são ilhas, assim dependem de outras classes
- Chamam métodos de outra classes
- Herdam outras classes

Acoplamento

- O problema é a forma como se dá esse acoplamento



Acoplamento

- Dependências podem se expressar de diversas formas
  - uso de um método de outra classe
  - manipulação e modificação de dados de outra classe

Quando o acoplamento é bom?

- Classe A usa apenas métodos públicos da Classe B
- A interface provida por B é estável

CarrinhoCompra só usa métodos públicos de Map

```
public class CarrinhoCompra {

    private  Map<String, Integer> puzzles;

    public CarrinhoCompra() {
        this.puzzles = new HashMap<String, Integer>();
    }

    public void addCarrinho(String nome, int numPecas) {
        this.puzzles.put(nome, numPecas);
    }

}
```

A interface de Map é estável

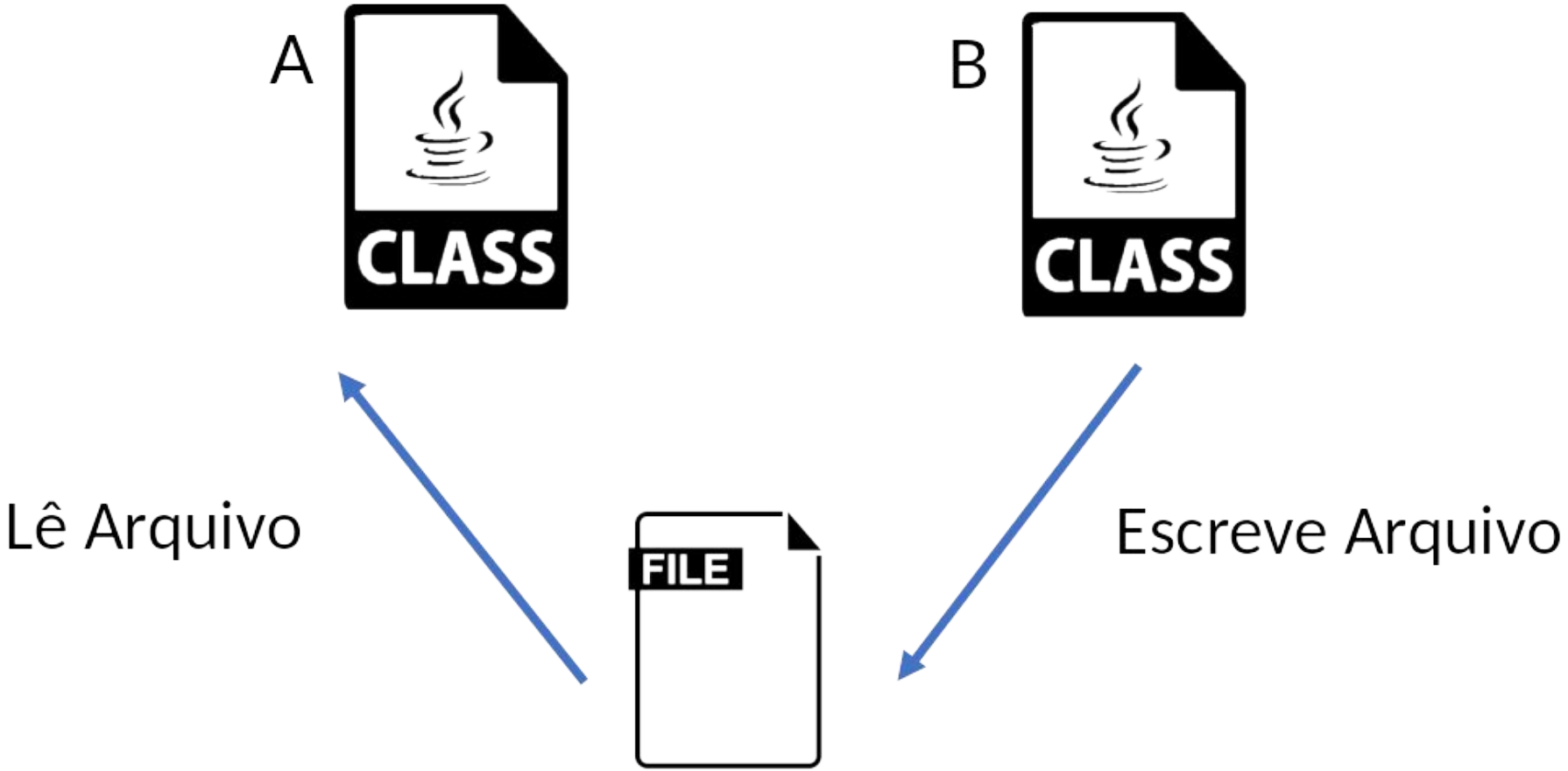
```
public class CarrinhoCompra {

    private  Map<String, Integer> puzzles;

    public CarrinhoCompra() {
        this.puzzles = new HashMap<String, Integer>();
    }

    public void addCarrinho(String nome, int numPecas) {
        this.puzzles.put(nome, numPecas);
    }

}
```

Quando o acoplamento é ruim?

- Classe A realiza acesso direto a um arquivo ou banco de dados da Classe B
  - B não sabe que A é um cliente

- Classe A e B compartilham uma estrutura de dados global

- Interface de B não é estável
  - dependência entre as classes não é mediada por uma interface estável


A
CLASS
B
CLASS
Lê Arquivo
FILE
Escreve Arquivo

```
public class ClasseA {

    private void buscaInimigo() {
        String nomeInimigo;
        //Le o arquivo inimigos.json
        //Salva no nomeInimigo
    }
}

public class ClasseB {

    private void escreveArquivo() {
        //Escreve um arquivo inimigos.json
    }
}
```

As classes não se conhecem

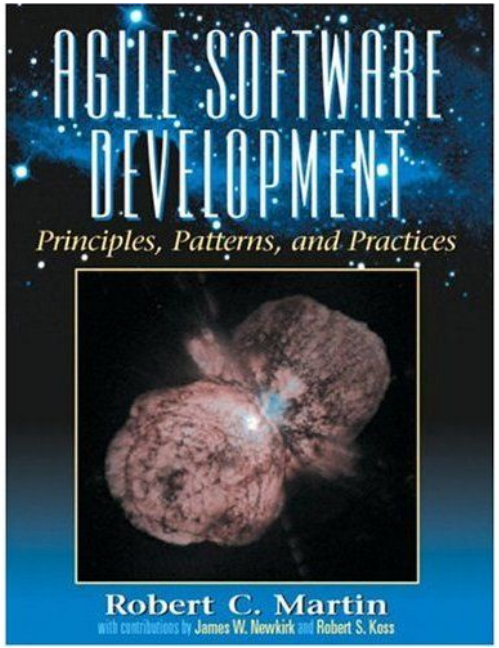
```java
public class ClasseA {

    private void buscaInimigo(ClasseB b) {
        String nomeInimigo;
        nomeInimigo = b.getInimigo();
        //Le o arquivo inimigos.json
        //Salva no nomeInimigo
    }
}
public class ClasseB {

    String nomeInimigo;
    public String getInimigo() {
        return nomeInimigo;
    }
    private void escreveArquivo() {
        //Escreve um arquivo inimigos.json
    }
}
```

Agora a dependência está explícita

Princípios SOLID

- Recomendações Concretas para atender as propriedades como:
  - coesão e acoplamento

- Pensando sempre na manutenção e evolução do software



Princípios SOLID

SRP – Single Responsibility Principle

OCP – Open-Closed Principle

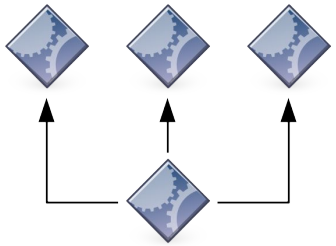
LSP – Liskov Substitution Principle

ISP – The Interface Segregation Principle

DIP – Dependency Inversion Principle

...

Antes de aprendermos princípios avançados de POO, é bom entendermos métricas de código-fonte.



Código Limpo e Métricas de Código-Fonte

Paulo Meirelles

paulormm@ime.usp.br