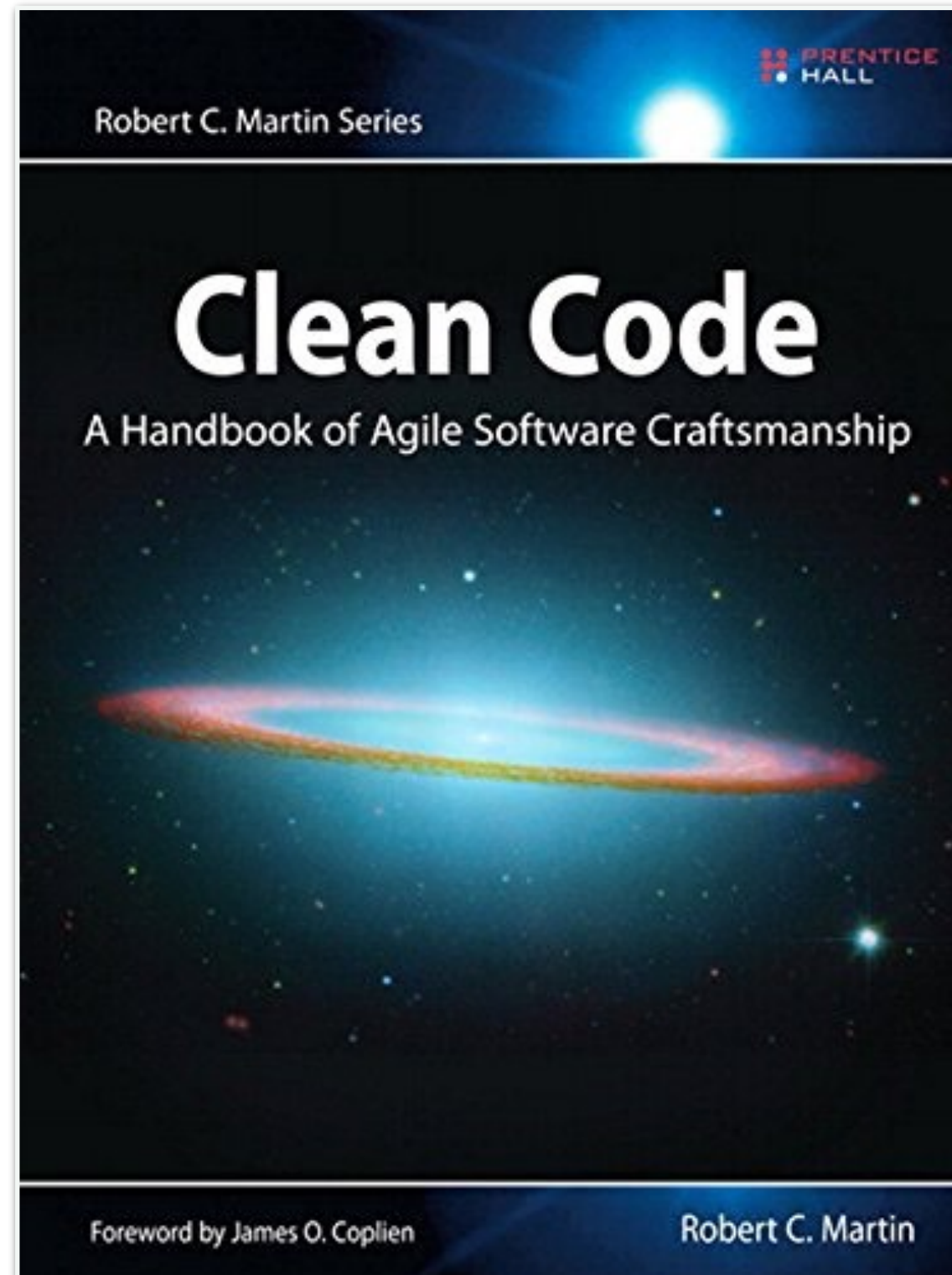


# Engenharia de Software II

## Código limpo - parte 2: funções

Prof. André Hora  
DCC/UFMG  
2019.1



# Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- Funções (cap 3)
- Código externo (cap 8)
- Testes de unidade (cap 9)
- Classes (cap 10)

# Agenda

- Legibilidade (cap 2)
- Comentários em código (cap 4)
- Formatação (cap 5)
- **Funções (cap 3)**
- Código externo (cap 8)
- Testes de unidade (cap 9)
- Classes (cap 10)

# Funções

# Tente compreender este código em 3 minutos

## Quais os problemas?

continua...

```
public static String testableHtml(
    PageData pageData,
    boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup =
                PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage
                );
            if (suiteSetup != null) {
                WikiPagePath pagePath =
                    suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
        WikiPage setup =
            PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath =
                wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                .append(setupPathName)
                .append("\n");
        }
    }
}
```

```
buffer.append(pageData.getContent());
if (pageData.hasAttribute("Test")) {
    WikiPage teardown =
        PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
    if (teardown != null) {
        WikiPagePath teardownPath =
            wikiPage.getPageCrawler().getFullPath(teardown);
        String teardownPathName = PathParser.render(teardownPath);
        buffer.append("\n")
            .append("!include -teardown .")
            .append(teardownPathName)
            .append("\n");
    }
    if (includeSuiteSetup) {
        WikiPage suiteTeardown =
            PageCrawlerImpl.getInheritedPage(
                SuiteResponder.SUITE_TEARDOWN_NAME,
                wikiPage
            );
        if (suiteTeardown != null) {
            WikiPagePath pagePath =
                suiteTeardown.getPageCrawler().getFullPath(suiteTeardown);
            String pagePathName = PathParser.render(pagePath);
            buffer.append("!include -teardown .")
                .append(pagePathName)
                .append("\n");
        }
    }
}
pageData.setContent(buffer.toString());
return pageData.getHtml();
}
```

# Entendeu?

- Provavelmente você não entendeu muito
- Existe muita coisa acontecendo na função, em vários níveis de abstração (ex: append, crawler, parser, buffer, etc)
- Existe muito código duplicado
- Condicionais aninhados

# Após uma simples refatoração

```
public static String renderPageWithSetupsAndTeardowns(
    PageData pageData, boolean isSuite
) throws Exception {
    boolean isTestPage = pageData.hasAttribute("Test");
    if (isTestPage) {
        WikiPage testPage = pageData.getWikiPage();
        StringBuffer newPageContent = new StringBuffer();
        includeSetupPages(testPage, newPageContent, isSuite);
        newPageContent.append(pageData.getContent());
        includeTeardownPages(testPage, newPageContent, isSuite);
        pageData.setContent(newPageContent.toString());
    }

    return pageData.getHtml();
}
```

**Tente compreender  
o código em 3 minutos**



# Entendeu?

- Provavelmente não todos os detalhes, mas:
  - Provavelmente você entendeu que essa função realiza a **inclusão de setups e teardowns** em uma página de testes e então **renderiza em HTML**
  - Relacionada a teste web
- Mais fácil derivar essa informação da segunda função que da primeira

# Funções

- O que torna a segunda função fácil de ler e entender?
- Como podemos fazer uma função comunicar sua intenção?
- Quais atributos uma função deve ter?

# Função deve ser pequena

- Primeira regra: funções devem ser pequenas
- Baseado na experiência do autor (50 anos de desenvolvimento!)
- Idealmente, ifs, elses, whiles devem ter **uma linha**

```
public static String renderPageWithSetupsAndTeardowns(  
    PageData pageData, boolean isSuite  
) throws Exception {  
    boolean isTestPage = pageData.hasAttribute("Test");  
    if (isTestPage) {  
        WikiPage testPage = pageData.getWikiPage();  
        StringBuffer newPageContent = new StringBuffer();  
        includeSetupPages(testPage, newPageContent, isSuite);  
        newPageContent.append(pageData.getContent());  
        includeTeardownPages(testPage, newPageContent, isSuite);  
        pageData.setContent(newPageContent.toString());  
    }  
  
    return pageData.getHtml();  
}
```

# Função deve ser pequena

- Primeira regra: funções devem ser pequenas
- Baseado na experiência do autor (50 anos de desenvolvimento!)
- Idealmente, ifs, elses, whiles devem ter **uma linha**

```
public static String renderPageWithSetupsAndTeardowns(  
    PageData pageData, boolean isSuite  
) throws Exception {  
    boolean isTestPage = pageData.hasAttribute("Test");  
    if (isTestPage) {  
        WikiPage testPage = pageData.getWikiPage();  
        StringBuffer newPageContent = new StringBuffer();  
        includeSetupPages(testPage, newPageContent, isSuite);  
        newPageContent.append(pageData.getContent());  
        includeTeardownPages(testPage, newPageContent, isSuite);  
        pageData.setContent(newPageContent.toString());  
    }  
  
    return pageData.getHtml();  
}
```



```
public static String renderPageWithSetupsAndTeardowns(  
    PageData pageData, boolean isSuite  
) throws Exception {  
    boolean isTestPage = pageData.hasAttribute("Test");  
    if (isTestPage(pageData))  
        includeSetupAndTeardownPages(pageData, isSuite);  
    return pageData.getHtml();  
}
```

# Função deve realizar apenas uma tarefa

- A primeira função realiza muitas tarefas
  - Cria buffers, busca páginas, busca heranças, faz parsers, acrescenta strings, gera HTML, etc
  - Possui vários níveis de abstração:
    - Alto nível: `getHtml()`
    - Nível intermediário: `PathParser.render(pagePath)`
    - Baixo nível: `append("\n")`
- A última função faz apenas uma coisa: *adiciona setups e teardowns em páginas de testes*

```

public static String testableHtml(
    PageData pageData,
    boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup =
                PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage
                );
            if (suiteSetup != null) {
                WikiPagePath pagePath =
                    suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
        WikiPage setup =
            PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath =
                wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                .append(setupPathName)
                .append("\n");
        }
    }
    buffer.append(pageData.getContent());
    if (pageData.hasAttribute("Test")) {
        WikiPage teardown =
            PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
        if (teardown != null) {
            WikiPagePath tearDownPath =
                wikiPage.getPageCrawler().getFullPath(teardown);
            String tearDownPathName = PathParser.render(tearDownPath);
            buffer.append("\n")
                .append("!include -teardown .")
                .append(tearDownPathName)
                .append("\n");
        }
        if (includeSuiteSetup) {
            WikiPage suiteTeardown =
                PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_TEARDOWN_NAME,
                    wikiPage
                );
            if (suiteTeardown != null) {
                WikiPagePath pagePath =
                    suiteTeardown.getPageCrawler().getFullPath(suiteTeardown);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -teardown .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
    }
    pageData.setContent(buffer.toString());
    return pageData.getHtml();
}

```

VS.

```

public static String renderPageWithSetupsAndTeardowns(
    PageData pageData, boolean isSuite
) throws Exception {
    boolean isTestPage = pageData.hasAttribute("Test");
    if (isTestPage(pageData))
        includeSetupAndTeardownPages(pageData, isSuite);
    return pageData.getHtml();
}

```

# Switches

- Difícil fazer um switch que realiza uma tarefa
- Por natureza, eles realizam N tarefas
- Podemos colocar switches em funções de baixo nível e garantir que ele nunca será repetido

# Switches

Qual o problema desta função?

```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```



# Switches

Qual o problema desta função?

```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```

- Grande e vai crescer se novos empregados forem adicionados
- Realiza mais de uma tarefa
- Viola o *Open Closed Principle* (entidades devem ser abertas para extensão, mas fechadas para modificação)
- **Principal:** outras funções terão a mesma estrutura (duplicação)
  - ex: isPayday(), deliverPay(), etc

# Principal Problema: Outras funções terão a mesma estrutura (duplicação)

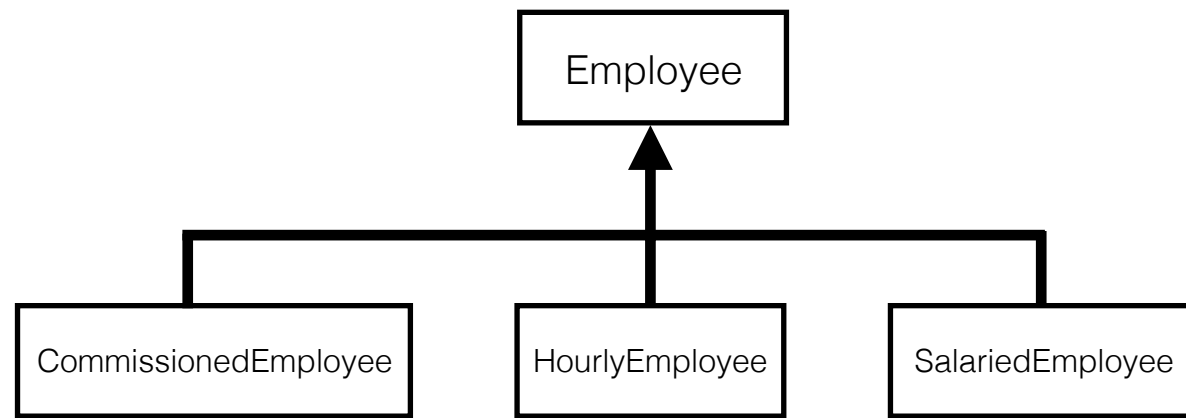
```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```

```
public boolean isPayday(Employee e, Date date)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return isCommissionedPayday(e, date);
        case HOURLY:
            return isHourlyPayday(e, date);
        case SALARIED:
            return isSalariedPayday(e, date);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```

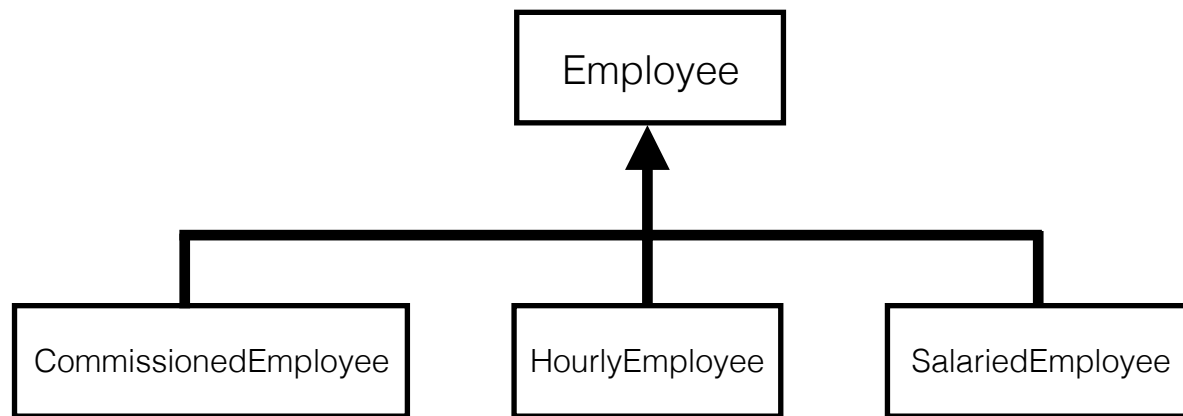
```
public boolean deliverPay(Employee e, Money pay)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return deliverCommissionedPay(e, pay);
        case HOURLY:
            return deliverHourlyPay(e, pay);
        case SALARIED:
            return deliverSalariedPay(e, pay);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```

# Switches

- Solução: utilizar padrão de projeto Abstract Factory
- Fábrica utiliza o switch para criar instâncias apropriadas de empregados
- Switch aparece apenas uma vez no código
- As várias funções que usam o switch (calculatePay(), isPayday(), deliverPay(), etc) serão despachadas polimorficamente



```
public abstract class Employee {
    public abstract boolean isPayday();
    public abstract Money calculatePay();
    public abstract void deliverPay(Money pay);
}
-----
public interface EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}
-----
public class EmployeeFactoryImpl implements
    EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
        switch (r.type) {
            case COMMISSIONED:
                return new CommissionedEmployee(r);
            case HOURLY:
                return new HourlyEmployee(r);
            case SALARIED:
                return new SalariedEmployee(r);
            default:
                throw new InvalidEmployeeType(r.type);
        }
    }
}
```



```

public abstract class Employee {
    public abstract boolean isPayday();
    public abstract Money calculatePay();
    public abstract void deliverPay(Money pay);
}

-----

public interface EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}

-----

public class EmployeeFactoryImpl implements
    EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
        switch (r.type) {
            case COMMISSIONED:
                return new CommissionedEmployee(r);
            case HOURLY:
                return new HourlyEmployee(r);
            case SALARIED:
                return new SalariedEmployee(r);
            default:
                throw new InvalidEmployeeType(r.type);
        }
    }
}

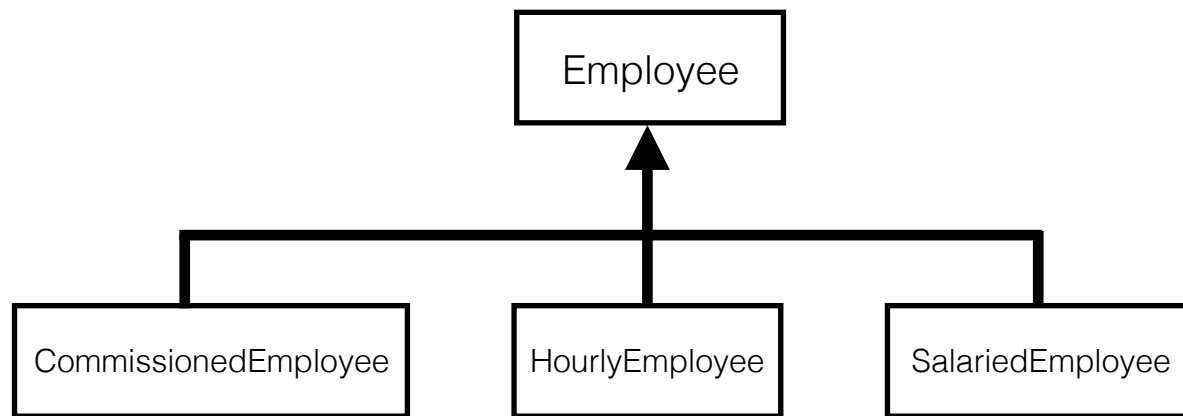
```

```

public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}

```





```

public abstract class Employee {
    public abstract boolean isPayday();
    public abstract Money calculatePay();
    public abstract void deliverPay(Money pay);
}

-----

public interface EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}

-----

public class EmployeeFactoryImpl implements
    EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
        switch (r.type) {
            case COMMISSIONED:
                return new CommissionedEmployee(r);
            case HOURLY:
                return new HourlyEmployee(r);
            case SALARIED:
                return new SalariedEmployee(r);
            default:
                throw new InvalidEmployeeType(r.type);
        }
    }
}
  
```

```

public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
  
```

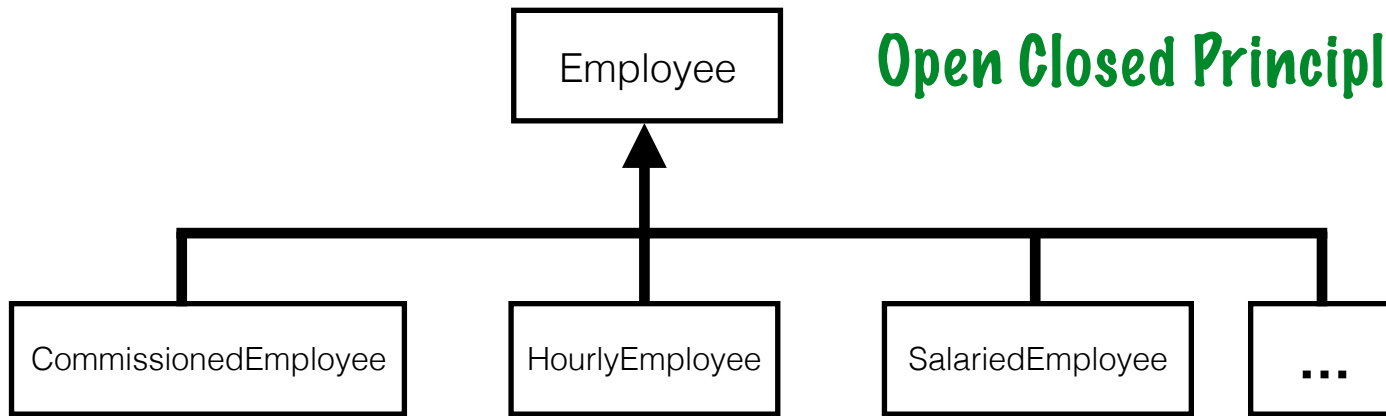


```

public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    e.calculatePay();
}
  
```



## Open Closed Principle



```
public abstract class Employee {
    public abstract boolean isPayday();
    public abstract Money calculatePay();
    public abstract void deliverPay(Money pay);
}

-----

public interface EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}

-----

public class EmployeeFactoryImpl implements
    EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
        switch (r.type) {
            case COMMISSIONED:
                return new CommissionedEmployee(r);
            case HOURLY:
                return new HourlyEmployee(r);
            case SALARIED:
                return new SalariedEmployee(r);
            default:
                throw new InvalidEmployeeType(r.type);
        }
    }
}
```

```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```



```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
    e.calculatePay();
}
```



# Switches

- Regra geral: switches são tolerados se aparecem apenas uma vez, são utilizados para criar objetos e estão escondido em uma herança
- Claro, essa regra é muito rígida; cada caso é único



# Exercício

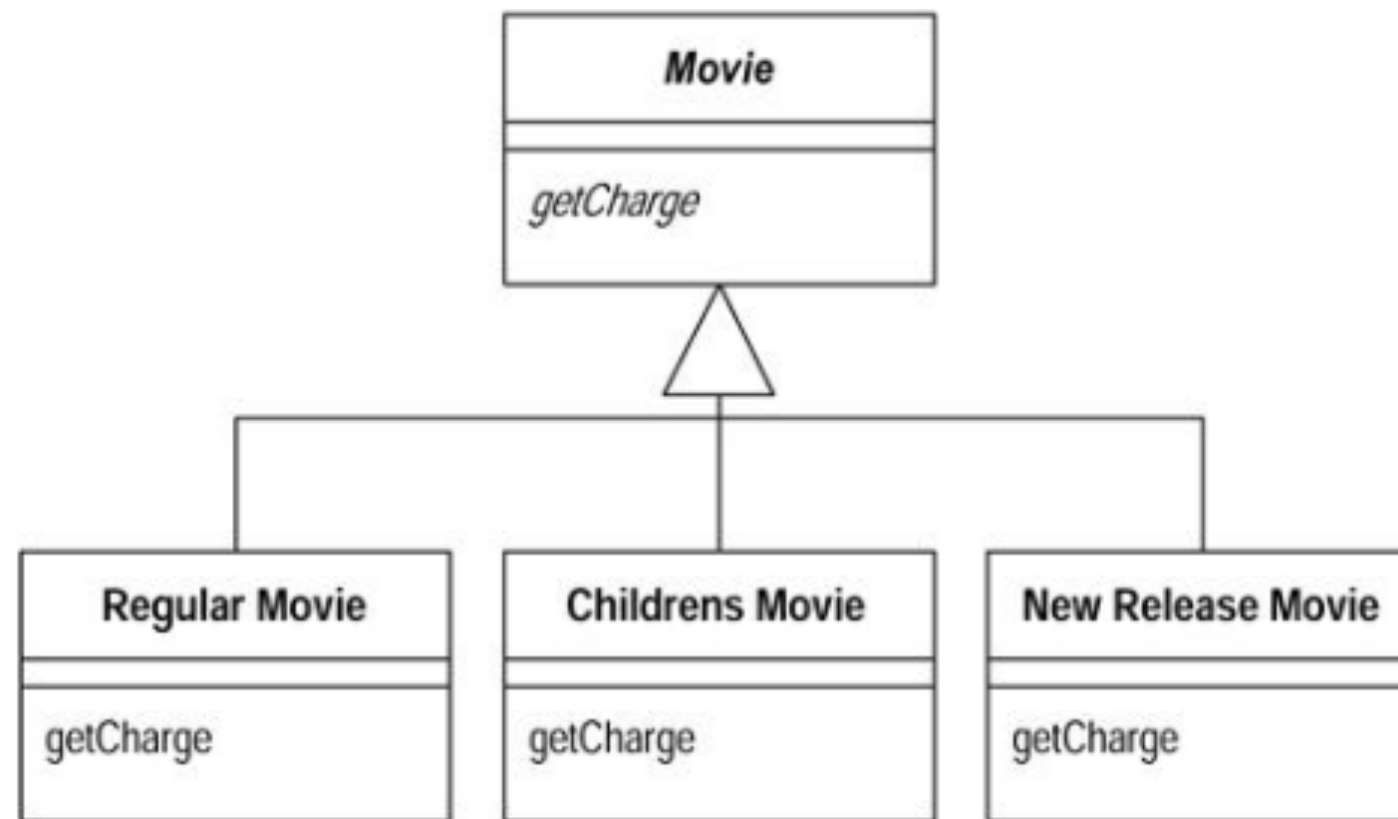
**Elabore uma solução para remover o switch abaixo**

1. Apresente o diagrama de classes
2. Apresente o código das classes criadas

```
public double getCharge() {  
    double amount = 0;  
    switch (getMovie().getPriceCode()) {  
        case Movie.REGULAR:  
            amount += 2;  
            if (getDaysRented() > 2)  
                amount += (getDaysRented() - 2) * 1.5;  
            break;  
        case Movie.NEW_RELEASE:  
            amount += getDaysRented() * 3;  
            break;  
        case Movie.CHILDREN:  
            amount += 1.5;  
            if (getDaysRented() > 3)  
                amount += (getDaysRented() - 3) * 1.5;  
            break;  
    }  
    return amount;  
}
```

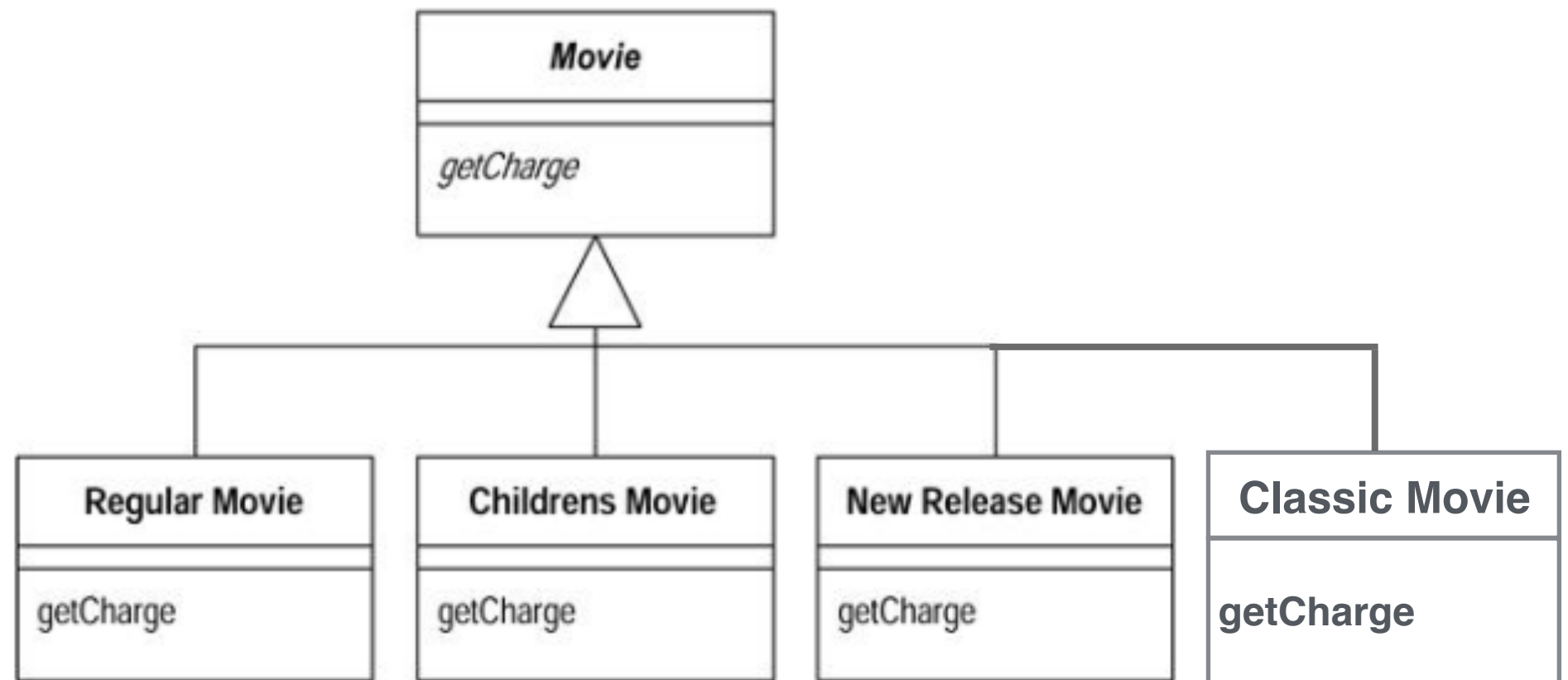
**class Movie**

# Exercício



- Isso nos permite substituir o switch por polimorfismo
- Cada tipo de filme terá suas regras de negócio
- Novos tipos poderão ser adicionados

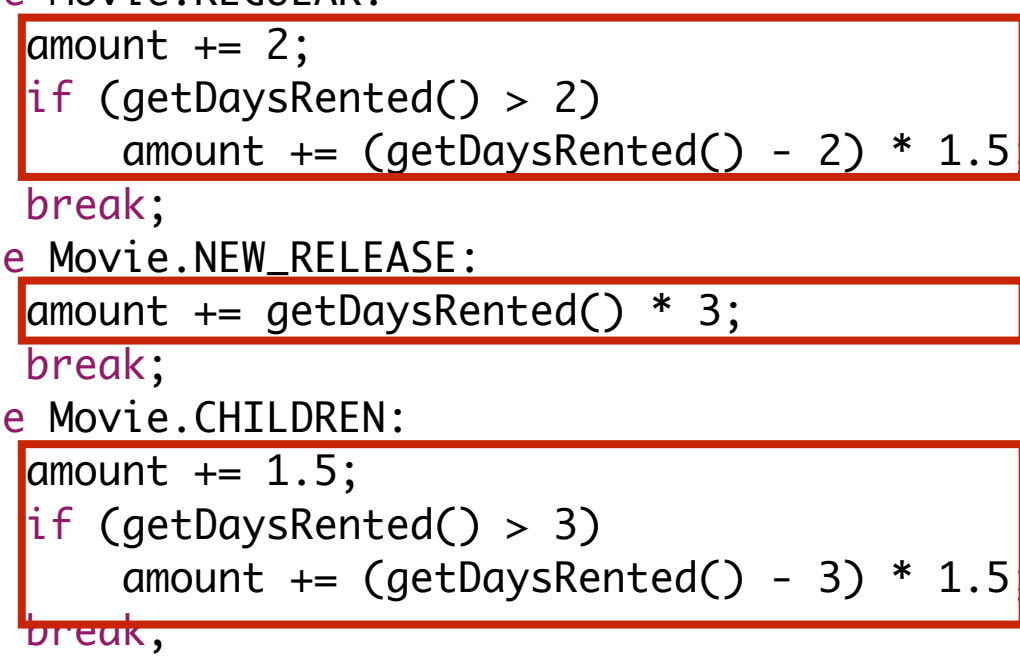
# Exercício



- Isso nos permite substituir o switch por polimorfismo
- Cada tipo de filme terá suas regras de negócio
- Novos tipos poderão ser adicionados

# Exercício

```
public double getCharge() {  
    double amount = 0;  
    switch (getMovie().getPriceCode()) {  
        case Movie.REGULAR:  
            amount += 2;  
            if (getDaysRented() > 2)  
                amount += (getDaysRented() - 2) * 1.5;  
            break;  
        case Movie.NEW_RELEASE:  
            amount += getDaysRented() * 3;  
            break;  
        case Movie.CHILDREN:  
            amount += 1.5;  
            if (getDaysRented() > 3)  
                amount += (getDaysRented() - 3) * 1.5;  
            break;  
    }  
    return amount;  
}
```



## RegularMovie

```
public double getCharge(int daysRented) {  
    double amount = 2;  
    if (daysRented > 2)  
        amount += (daysRented - 2) * 1.5;  
    return amount;  
}
```

## NewReleaseMovie

```
public double getCharge(int daysRented) {  
    return daysRented * 3;  
}
```

## ChildrensMovie

```
public double getCharge(int daysRented) {  
    double amount = 1.5;  
    if (daysRented > 3)  
        amount += (daysRented - 3) * 1.5;  
    return amount;  
}
```

# Utilizar nomes descritivos

- Nomes descritivos são melhores que nomes curtos e enigmáticos
- Nomes descritivos são melhores que comentários descritivos
- Nomear entidades não é uma tarefa fácil (apesar de parecer fácil)
  - IDEs modernas tornam fácil mudar nomes
- Seja consistente nos nomes (ex: utilizar mesmo prefixo ou sufixo)

# Argumentos de Funções

- Número ideal: zero
- Em seguida: um, dois
- Deve ser evitado: três ou mais
- Razões para utilizar poucos argumentos:
  - Torna a função mais complexa
  - Dificulta a criação de testes (todas as combinações devem ser testadas)

# Um argumento

- Realizar uma pergunta sobre o argumento
  - `boolean fileExists("MyFile.txt")`
- Operar no argumento, transformar e retornar ele
  - `InputStream fileOpen("MyFile.txt")`
- Alterar o estado do sistema
  - `void passwordAttemptFailedNtimes(int attempts)`
- **Evitar**: `void transform(StringBuffer out)`
- **Melhor**: `StringBuffer transform(StringBuffer in)`

# Um argumento

- Realizar uma pergunta sobre o argumento
  - `boolean fileExists("MyFile.txt")`
- Operar no argumento, transformar e retornar ele
  - `InputStream fileOpen("MyFile.txt")`
- Alterar o estado do sistema
  - `void passwordAttemptFailedNtimes(int attempts)`
- **Evitar**: `void transform(StringBuffer out)` **confuso**
- **Melhor**: `StringBuffer transform(StringBuffer in)`  
**transformação fica explícito**



# Efeitos Colaterais

- Evitar funções que realizam tarefas “escondidas”, gerando um efeito colateral
- Exemplos:
  - Mudanças inesperadas em variáveis, parâmetros ou atributos
  - Realização de tarefas distintas do seu objetivo

# Efeitos Colaterais

- Função *checkPassword* valida *userName* e *password*, retornando true ou false
- Mas também possui um efeito colateral. Qual?

```
public class UserValidator {
    private Cryptographer cryptographer;

    public boolean checkPassword(String userName, String password) {
        User user = UserGateway.findByName(userName);
        if (user != User.NULL) {
            String codedPhrase = user.
                getPhraseEncodedByPassword();
            String phrase = cryptographer.decrypt(codedPhrase, password);
            if ("Valid Password".equals(phrase)) {
                Session.initialize();
                return true;
            }
        }
        return false;
    }
}
```

# Efeitos Colaterais

- Função *checkPassword* valida *userName* e *password*, retornando true ou false
- Mas também possui um efeito colateral. Qual?

```
public class UserValidator {  
    private Cryptographer cryptographer;  
  
    public boolean checkPassword(String userName, String password) {  
        User user = UserGateway.findByName(userName);  
        if (user != User.NULL) {  
            String codedPhrase = user.  
                getPhraseEncodedByPassword();  
            String phrase = cryptographer.decrypt(codedPhrase, password);  
            if ("Valid Password".equals(phrase)) {  
                Session.initialize();  
                return true;  
            }  
        }  
        return false;  
    }  
}
```

# Efeitos Colaterais

- Função *checkPassword* valida *userName* e *password*, retornando true ou false
- Mas também possui um efeito colateral. Qual?

```
public class UserValidator {  
    private Cryptographer cryptographer  
  
    public boolean checkPassword(String  
        User user = UserGateway.findByName(userName);  
        if (user != User.NULL) {  
            String codedPhrase = user.  
                getPhraseEncodedByPassword();  
            String phrase = cryptographer.  
                if ("Valid Password".equals(phrase)  
                Session.initialize();  
            return true;  
        }  
    }  
}
```

**Pelo seu nome, a função não deveria iniciar uma sessão**

**Logo, essa função só deve ser chamada em certos momentos**

**Melhor: `checkPasswordAndInitializeSession()`**

# Don't repeat yourself

- Não repita a si mesmo
- Duplicação é a raiz de diversos problemas
- Vários princípios e práticas foram criados para evitar ou eliminar duplicação de código
- Funções, métodos, OO, programação estruturada, programação orientada a aspectos, em parte, são estratégias para eliminar duplicação

```

public static String testableHtml(
    PageData pageData,
    boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup =
                PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage
                );
            if (suiteSetup != null) {
                WikiPagePath pagePath =
                    suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
        WikiPage setup =
            PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath =
                wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                .append(setupPathName)
                .append("\n");
        }
    }
}

```

# Onde está a duplicação?

continua...

```

buffer.append(pageData.getContent());
if (pageData.hasAttribute("Test")) {
    WikiPage teardown =
        PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
    if (teardown != null) {
        WikiPagePath tearDownPath =
            wikiPage.getPageCrawler().getFullPath(teardown);
        String tearDownPathName = PathParser.render(tearDownPath);
        buffer.append("\n")
            .append("!include -teardown .")
            .append(tearDownPathName)
            .append("\n");
    }
    if (includeSuiteSetup) {
        WikiPage suiteTeardown =
            PageCrawlerImpl.getInheritedPage(
                SuiteResponder.SUITE_TEARDOWN_NAME,
                wikiPage
            );
        if (suiteTeardown != null) {
            WikiPagePath pagePath =
                suiteTeardown.getPageCrawler().getFullPath(suiteTeardown);
            String pagePathName = PathParser.render(pagePath);
            buffer.append("!include -teardown .")
                .append(pagePathName)
                .append("\n");
        }
    }
}
pageData.setContent(buffer.toString());
return pageData.getHtml();
}

```

# Onde está a duplicação?

continua...

```
public static String testableHtml(
    PageData pageData,
    boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup =
                PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage
                );
            if (suiteSetup != null) {
                WikiPagePath pagePath =
                    suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
        WikiPage setup =
            PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath =
                wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                .append(setupPathName)
                .append("\n");
        }
    }
}
```

```
buffer.append(pageData.getContent());
if (pageData.hasAttribute("Test")) {
    WikiPage teardown =
        PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
    if (teardown != null) {
        WikiPagePath teardownPath =
            wikiPage.getPageCrawler().getFullPath(teardown);
        String teardownPathName = PathParser.render(teardownPath);
        buffer.append("\n")
            .append("!include -teardown .")
            .append(teardownPathName)
            .append("\n");
    }
    if (includeSuiteSetup) {
        WikiPage suiteTeardown =
            PageCrawlerImpl.getInheritedPage(
                SuiteResponder.SUITE_TEARDOWN_NAME,
                wikiPage
            );
        if (suiteTeardown != null) {
            WikiPagePath pagePath =
                suiteTeardown.getPageCrawler().getFullPath(suiteTeardown);
            String pagePathName = PathParser.render(pagePath);
            buffer.append("!include -teardown .")
                .append(pagePathName)
                .append("\n");
        }
    }
}
pageData.setContent(buffer.toString());
return pageData.getHtml();
}
```



# Onde está a duplicação?

continua...

```
public static String testableHtml(
    PageData pageData,
    boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup =
                PageCrawlerImpl.getInheritedPage(
                    SuiteResponder.SUITE_SETUP_NAME, wikiPage
                );
            if (suiteSetup != null) {
                WikiPagePath pagePath =
                    suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
    }
    WikiPage setup =
        PageCrawlerImpl.getInheritedPage(
            SuiteResponder.SETUP_NAME, wikiPage
        );
    if (setup != null) {
        WikiPagePath
            wikiPagePath =
                setup.getPageCrawler().getFullPath(setup);
        String setupPathName = PathParser.render(wikiPagePath);
        buffer.append("!include -setup .")
            .append(setupPathName)
            .append("\n");
    }
}
```

```
buffer.append(pageData.getContent());
if (pageData.hasAttribute("Test")) {
    WikiPage teardown =
        PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
    if (teardown != null) {
        WikiPagePath teardownPath =
            wikiPage.getPageCrawler().getFullPath(teardown);
        String teardownPathName = PathParser.render(teardownPath);
        buffer.append("\n")
            .append("!include -teardown .")
            .append(teardownPathName)
            .append("\n");
    }
}
```

- Não é fácil de ver
- 4 locais para mudar
- 4 oportunidades para introduzir erros

```
WikiPagePath pagePath =
    suiteTeardown.getPageCrawler().getFullPath (suiteTeardown);
String pagePathName = PathParser.render(pagePath);
buffer.append("!include -teardown .")
    .append(pagePathName)
    .append("\n");
}
```

```
pageData.setContent(buffer.toString());
return pageData.getHtml();
}
```



# Mensagem Final

- Funções devem ser refinadas ao longo do tempo
- Primeira versão nunca é perfeita
  - Geralmente, longa e complicada
  - Talvez com loops aninhados e longas listas de argumentos
  - Possivelmente com nomes ruins e código duplicado
- **Importante:** sempre escrever testes
  - Desse modo, funções podem ser refinadas, divididas, renomeadas, removidas as duplicações, com a garantia dos testes