

AspectJML Online

Trabalho de graduação - Ciência da Computação
Universidade Federal de Pernambuco

2017.1

Aluno: Igor Leão
Orientador: Henrique Rebêlo

Agenda

- Introdução
- Caracterização do problema
- Objetivos
- Metodologia
- Fundamentação teórica
- Abordagem proposta
- Demonstração da solução

Introdução

- Surgimento da programação orientada a objetos
 - Preocupação a respeito da confiabilidade
 - Reuso
 - Software de larga escala
- *Design by Contract*
 - *Mayer*
 - *Ideia de contratos*
 - *Restrições de comportamento*
 - *Checados em tempo de execução*

Introdução

- *Design by Contract*
 - *PyContract*
 - *JML*
- *Contratos como um interesse transversal*
 - *AspectJ*
 - *Quebra do raciocínio modular*
 - *Documentação*
- *AspectJML*
 - *O melhor dos dois mundos*

Caracterização do problema

- AspectJML
 - Ambiente de desenvolvimento incipiente
 - Difícil configuração de um ambiente de desenvolvimento
 - Dependências
 - Plugins
 - Atualização de software
- Relato de caso
 - Cadeira de aspectos em 2017.1, Henrique Rebêlo, CIn UFPE
 - Problemas na atualização da JVM, Ana Cavalcanti, University of York

Caracterização do problema

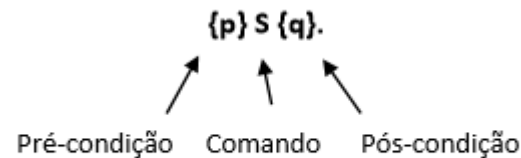
- Ambiente de difícil configuração
 - Dificulta a disseminação da linguagem
 - Dificulta seu ensino

Objetivos

- Revisão da literatura
 - Entender o surgimento do *Design by Contract*
 - *Investigação sobre o estado do paradigma*
- Abordagem para resolução do problema
- Demonstração da solução

Fundamentação teórica

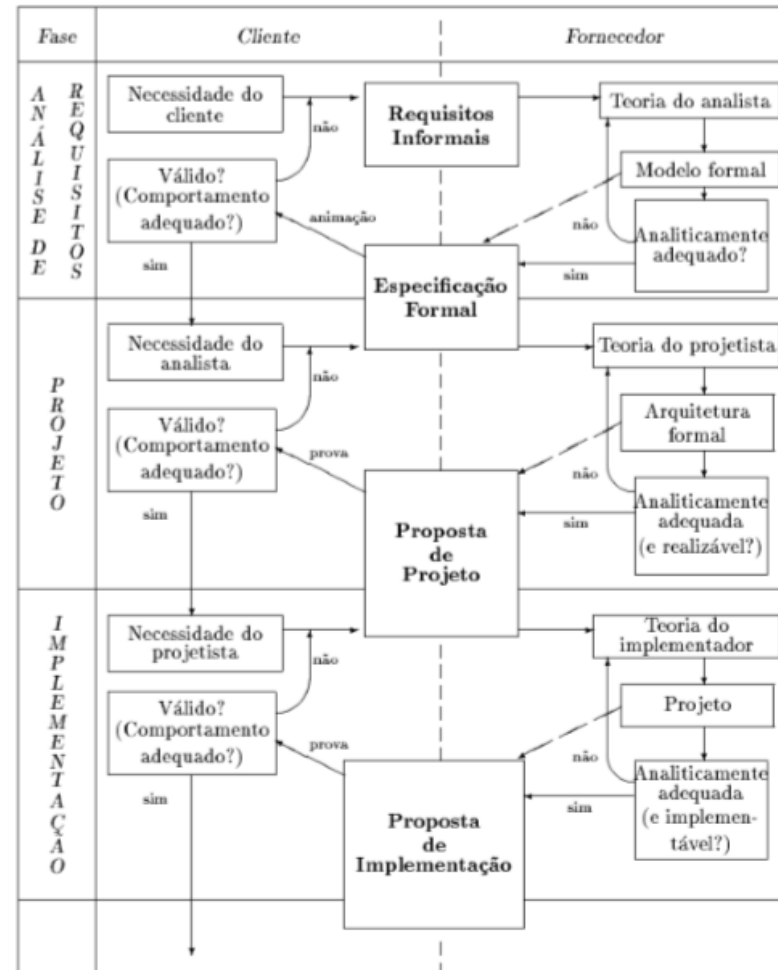
- An axiomatic Basis for Computer Programming (Hoare, 1969)
 - Raciocínio com em axiomas
- Tripla de Hoare
 - como a execução do código afeta estado de execução do programa



Equação 1: Demonstração da tripla de Hoare

Fundamentação teórica

- Semântica Axiomática
 - Em busca de um modelo
 - O formal e o informal
 - Modelo cascata
 - Modelo contratual



Fundamentação teórica

- *Design by Contract*
 - *Contratos em tempo de execução*
 - *PyContract, JML entre outros*
 - *Pré-condições, declarações de invariante e pós-condições*
 - *JML*
 - *Especificação no universo Java*
 - *requires , ensures \result, entre outras*
 - *Aspectos*

Abordagem proposta

- IDE para desenvolvimento de programas da linguagem
 - *Code/ compiling as a servisse*
 - *Sem preocupação com dependências ou plugins*
 - *Sem preocupação com o ambiente de desenvolvimento*

Abordagem proposta

- A *IDE* deve estar presente através da *World Wide Web*, sendo disponível para estudantes, entusiastas e profissionais que desejam editar, compilar e executar programas *AspectJML*.
- A *IDE* deve mostrar o resultado de compilação do programa de entrada que foi codificado usando a própria *IDE*.
- A *IDE* deve mostrar o resultado de execução do programa de entrada que foi codificado usando a própria *IDE*.
- A *IDE* deve possibilitar o *download* do código que foi editado na própria *IDE*.
- A *IDE* deve possibilitar o *upload* do código que foi baixado por meio da própria *IDE*.
- A *IDE* deve possibilitar manter uma visão da estrutura do projeto através de uma *TreeView*.
- A *IDE* deve possibilitar uma visão da estrutura do projeto através do uso de abas.

Abordagem proposta

- Seleção entre diferentes temas
- Seleção entre diferentes tamanhos de fonte
- Possibilidade de esconder ou exibir a *TreeView*.
- Possibilidade de esconder ou exibir o resultado da compilação e execução.
- Possibilidade de escolher exibir apenas o resultado da compilação ou apenas o resultado da execução.
- Possibilidade de baixar tanto o último código no servidor quando o código recém editado pelo usuário que ainda não foi alvo de compilação.
- Possibilidade de carregar projetos exemplos para finalidade de ensino.
- Possibilidade de renomear seus arquivos, que são representados tanto por meio de abas quando em uma estrutura de árvore.
- Possibilidade de nomear e renomear a unidade semântica de todos os arquivos na forma de um projeto.
- Possibilidade de deletar arquivos e pacotes.
- Possibilidade facilitar a manipulação de código através de atalhos.

Abordagem proposta

- Tecnologias

- *Front-end*

- *AngularJS*
 - *Code Mirror*
 - *Jquery*

- Back-end

- Ruby
 - Rails
 - Docker
 - Isolamento entre os processos
 - Isolamento entre o sistema de arquivos

Conclusões

- Tornou-se possível
 - Desenvolvimento de aplicações *AspectJML* através de uma interface *WEB*
 - Abstração do ambiente de desenvolvimento tal qual o sistema operacional e a versão do compilador
 - Criação de uma *IDE* capaz de fornecer recursos previamente não disponíveis, como realce de sintaxe e auto completar, específicos do *AspectJML*
- *Limitações*
 - *Polimento*
 - *Funcionalidades*
 - *Run, Projetos exemplos*
 - *Bugs*

Demonstração da solução

```
1 package learning.aspectjml;
2
3 import org.aspectj.lang.annotation.Aspect;
4 import org.aspectj.lang.annotation.Before;
5 import org.aspectj.lang.JoinPoint;
6
7 public class Main {
8     public static void main(String[] args) {
9         C c = new C();
10        c.m(10);
11        c.n(10);
12    }
13 }
14
15 class C {
16     public void m(int x){
17         System.out.println("C.m(int)");
18     }
19
20     public void n(int x){
21         this.adv();
22     }
23
24     @org.aspectjml.lang.annotation.Before("execution(void m(int))")
25     public void adv(){
26         System.out.println("C.adv()");
27     }
28
29     @org.aspectjml.lang.annotation.Before("call(void adv())")
30     public void adv2(JoinPoint thisJP){
31         System.out.println("C.adv2()");
32         System.out.println("-- "+thisJP.toString());
33         System.out.println(""+thisJP.getSourceLocation());
34     }
35 }
36
37 @Aspect
38 class A2 extends A{
39     @Before("execution(* *.main(..)")
40     public void adv(){
41         System.out.println("A2.adv()");
```

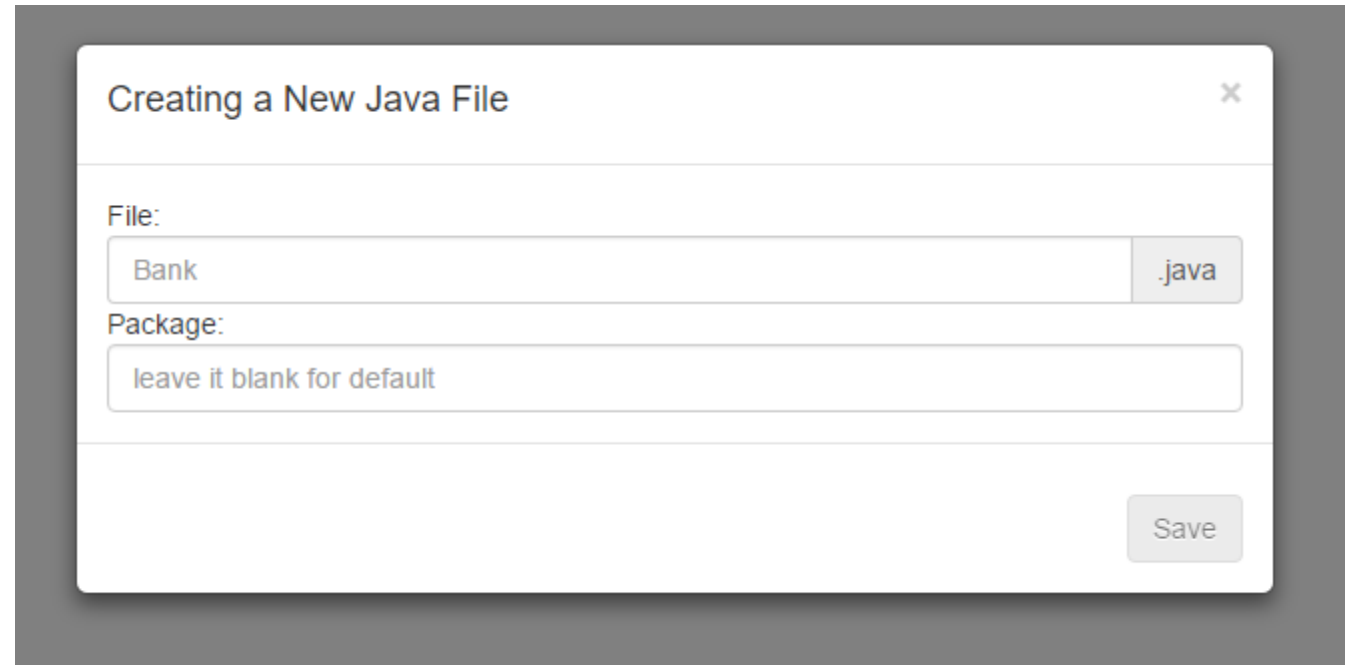
More Info

Compile and Execute Mirror Current Tab Max. Output Clear Output Eclipse 9pt

(compiler output will display here)

(program output will display here)

Demonstração da solução



The image shows a dialog box titled "Creating a New Java File" with a close button (X) in the top right corner. The dialog contains two input fields: "File:" and "Package:". The "File:" field has the text "Bank" and a ".java" extension button to its right. The "Package:" field has the text "leave it blank for default". A "Save" button is located at the bottom right of the dialog.

Creating a New Java File ×

File:
Bank .java

Package:
leave it blank for default

Save

Demonstração da solução

```
1 package learning.aspectjml;
2
3 import org.aspectj.lang.annotation.Aspect;
4 import org.aspectj.lang.annotation.Before;
5 import org.aspectj.lang.JoinPoint;
6
7 public class Main {
8     public static void main(String[] args) {
9         C c = new C();
10        c.m(10);
11        c.n(10);
12    }
13 }
14
15 class C {
16     public void m(int x){
17         System.out.println("C.m(int)");
18     }
19
20     public void n(int x){
21         this.adv();
22     }
23
24     @org.aspectjml.lang.annotation.Before("execution(void m(int))")
25     public void adv(){
26         System.out.println("C.adv()");
27     }
28
29     @org.aspectjml.lang.annotation.Before("call(void adv())")
30     public void adv2(JoinPoint thisJP){
31         System.out.println("C.adv2()");
32         System.out.println("-- "+thisJP.toString());
33         System.out.println(""+thisJP.getSourceLocation());
34     }
35 }
36
37 @Aspect
38 class A2 extends A{
39     @Before("execution(* *.main(..)")
```

```
1 package learning.aspectjml;
2
3 import org.aspectj.lang.annotation.Aspect;
4 import org.aspectj.lang.annotation.Before;
5 import org.aspectj.lang.JoinPoint;
6
7 public class Main {
8     public static void main(String[] args) {
9         C c = new C();
10        c.m(10);
11        c.n(10);
12    }
13 }
14
15 class C {
16     public void m(int x){
17         System.out.println("C.m(int)");
18     }
19
20     public void n(int x){
21         this.adv();
22     }
23
24     @org.aspectjml.lang.annotation.Before("execution(void m(int))")
25     public void adv(){
26         System.out.println("C.adv()");
27     }
28
29     @org.aspectjml.lang.annotation.Before("call(void adv())")
30     public void adv2(JoinPoint thisJP){
31         System.out.println("C.adv2()");
32         System.out.println("-- "+thisJP.toString());
33         System.out.println(""+thisJP.getSourceLocation());
34     }
35 }
36
37 @Aspect
38 class A2 extends A{
39     @Before("execution(* *.main(..)")
```

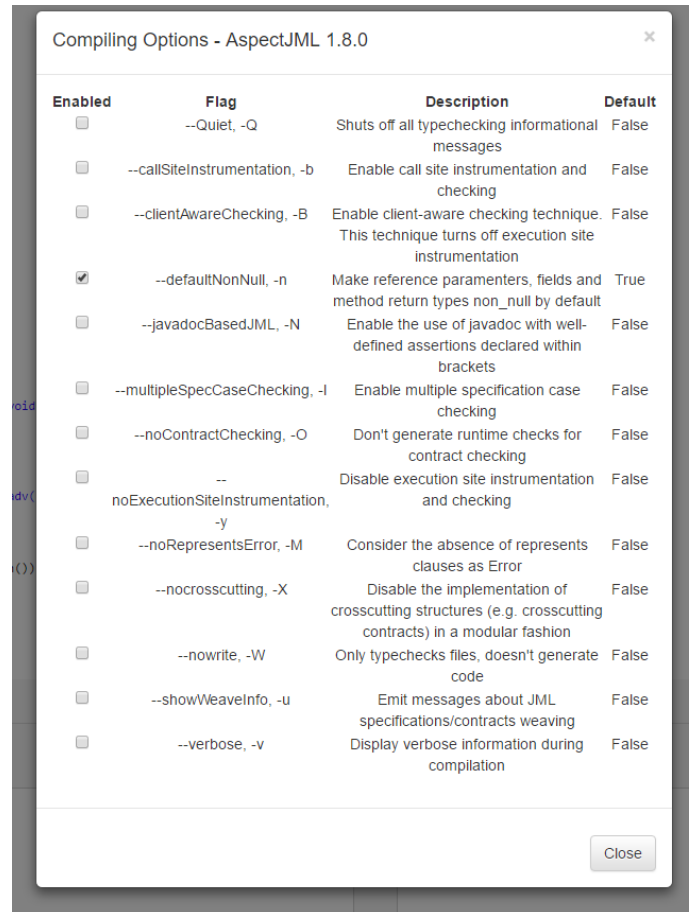
Main.java x A3.aj x

Un-Mirror Tab Max. Output Clear Output

Eclipse 9pt

(compiler output will display here)

Demonstração da solução



Enabled	Flag	Description	Default
<input type="checkbox"/>	--Quiet, -Q	Shuts off all typechecking informational messages	False
<input type="checkbox"/>	--callSiteInstrumentation, -b	Enable call site instrumentation and checking	False
<input type="checkbox"/>	--clientAwareChecking, -B	Enable client-aware checking technique. This technique turns off execution site instrumentation	False
<input checked="" type="checkbox"/>	--defaultNonNull, -n	Make reference parameters, fields and method return types non_null by default	True
<input type="checkbox"/>	--javadocBasedJML, -N	Enable the use of javadoc with well-defined assertions declared within brackets	False
<input type="checkbox"/>	--multipleSpecCaseChecking, -I	Enable multiple specification case checking	False
<input type="checkbox"/>	--noContractChecking, -O	Don't generate runtime checks for contract checking	False
<input type="checkbox"/>	--noExecutionSiteInstrumentation, -y	Disable execution site instrumentation and checking	False
<input type="checkbox"/>	--noRepresentsError, -M	Consider the absence of represents clauses as Error	False
<input type="checkbox"/>	--nocrosscutting, -X	Disable the implementation of crosscutting structures (e.g. crosscutting contracts) in a modular fashion	False
<input type="checkbox"/>	--nowrite, -W	Only typechecks files, doesn't generate code	False
<input type="checkbox"/>	--showWeaveInfo, -u	Emit messages about JML specifications/contracts weaving	False
<input type="checkbox"/>	--verbose, -v	Display verbose information during compilation	False

Close