


Laboratório de Engenharia de Software



## Aula 07


# Revisões, Inspeções e Padrões de Programação 1

*“Quantas regras!”* ALUNO ESPANTADO

*“A forma mais rápida de desenvolver um novo hábito é não ter outra alternativa”* [Hadden 1999]

Alessandro Garcia  
LES/DI/PUC-Rio  
Agosto 2010


## Especificação




Laboratório de Engenharia de Software

- Objetivo dessa aula
  - Entender como certas faltas podem ser detectadas com análise estática
  - Estudar catálogos de padrões de programação e a importância deles
- Referência básica:
  - Capítulos 4, 12 e apêndices do livro

Set 2009Alessandro Garcia © LES/DI/PUC-Rio2 / 35

|                                       |  |   |
|---------------------------------------|--|---|
| Laboratório de Engenharia de Software | <b>Objetivos</b>   |  |
|                                       | <ul style="list-style-type: none"> <li>• Descrever o relacionamento entre análise estática e teste</li> <li>• Entender diferentes categorias de análise estática:             <ul style="list-style-type: none"> <li>– Revisão</li> <li>– Inspeção</li> <li>– Medição (já parcialmente discutido)</li> </ul> </li> <li>• Exemplos...             <ul style="list-style-type: none"> <li>– executar uma revisão (de trechos) de código</li> </ul> </li> <li>• Compreender a importância de aderir a padrões de programação estabelecidos             <ul style="list-style-type: none"> <li>– Hoje: padrões de <i>faltas</i></li> </ul> </li> </ul> |   |
|                                       | Set 2009   | Alessandro Garcia © LES/DI/PUC-Rio 3 / 35   |

|                                       |   |   |
|---------------------------------------|---|---|
| Laboratório de Engenharia de Software | <b>Erros de Compilação</b>  |  |
|                                       | <ul style="list-style-type: none"> <li>• Os erros de compilação ocorrem quando o programa que escrevemos não obedece às regras da linguagem</li> <li>• Este tipo de erros são normalmente fáceis de detectar</li> <li>• Os mais frequentes são:             <ul style="list-style-type: none"> <li>– Esquecer um ponto e vírgula</li> <li>– Esquecer de fechar uma chaves</li> <li>– Esquecer de fechar as aspas num <i>printf</i> ou <i>scanf</i></li> <li>– Esquecer de fechar um comentário</li> <li>– Esquecer a declaração de variáveis</li> <li>– Escrever mal uma palavra (ex: <i>studio.h</i> em vez de <i>stdio.h</i>)</li> </ul> </li> <li>• Estamos interessados aqui em detectar <b><i>faltas básicas</i></b> que não são capturadas pelo compilador             <ul style="list-style-type: none"> <li>– ... até mesmo antes das atividades de teste!</li> </ul> </li> </ul> |   |
|                                       | Set 2009  | Alessandro Garcia © LES/DI/PUC-Rio 4 / 35   |

## "Crenças" populares em...

- ... desenvolvimento de software
  - programadores são *inteligentes e espertos*
  - tais programadores **não** cometem erros *primários* ou *estúpidos*
  - nós temos técnicas (parcialmente automatizáveis) suficientes para detecção preliminar de bugs
    - programação em pares (*pair programming*)
    - teste de módulos e integração
  - defeitos remanescentes em software são sutis (i.e. não triviais) e requerem técnicas de detecção extremamente sofisticadas



## Você introduziria este tipo de erro?

```
if ( i == null) {  
    ... // algum código  
    try {  
        i.close();  
        ...  
    } // algum código  
}
```



## Você introduziria este tipo de erro?



```
if ( i == null) {  
    ... // algum código  
    try {  
        i.close();  
        ...  
    } // algum código  
}
```

- Ooopss...
- Este código estava na implementação de Eclipse 3.0.0 M8
  - nós sempre podemos nos surpreender pelos defeitos grotescos que inserimos em nosso próprio código
  - **somente foi descoberto na versão 3!**

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

7 / 35

## Por que erros ocorrem?




- Ninguém é perfeito
- Tipos comuns de erros:
  - características de linguagem e APIs mal compreendidas
  - erros tipográficos
    - exemplo: *parênteses* que foram esquecidos ou uso errado de *operadores booleanos*
  - invariantes de módulos ou métodos mal compreendidas
- Pesquisadores recentemente fizeram uma busca em código de produtos comerciais existentes bem conhecidos e...
  - encontraram “zilhões” de faltas classificados como primárias

Set 2009

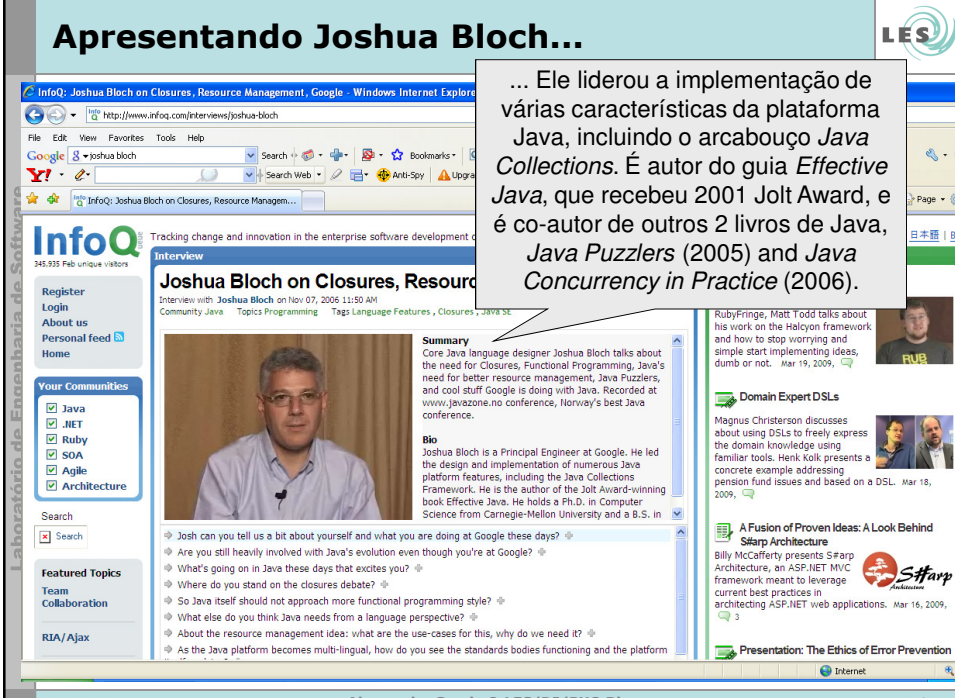
Alessandro Garcia © LES/DI/PUC-Rio

8 / 35

## Apresentando Joshua Bloch...




... Ele liderou a implementação de várias características da plataforma Java, incluindo o arcabouço *Java Collections*. É autor do guia *Effective Java*, que recebeu 2001 Jolt Award, e é co-autor de outros 2 livros de Java, *Java Puzzlers* (2005) and *Java Concurrency in Practice* (2006).



Set 2009      Alessandro Garcia © LES/DI/PUC-Rio      9 / 35

## Análise do código de JDK



- Foram encontrados 4 laços (*loops*) infinitos
- Incluindo um deles escrito por Joshua Bloch

```

public String foundType() {
    return this.foundType();
}
    
```

Set 2009      Alessandro Garcia © LES/DI/PUC-Rio      10 / 35

## Você também erra!



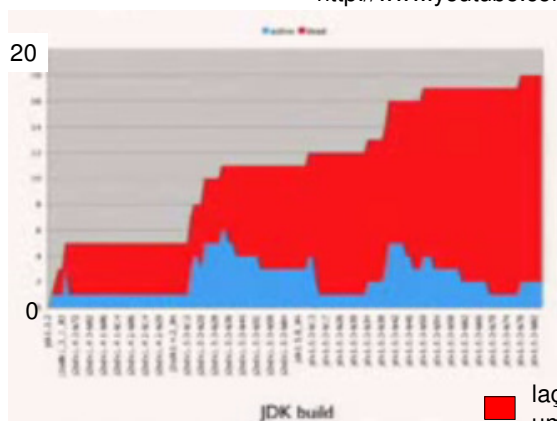
- Se Joshua Bloch comete erros deste tipo, **você também** pode cometer!
  - A única desvantagem de vocês é que...
    - erros infantis = nota menor ☹
    - ... alguns de vocês ainda não são ricos como ele ☺
- Ele está na Google, mas não foi este “bug” que o tirou da Sun! ☺
- Pessoas inteligentes cometem erros primários
- Admita isso e use abordagens sistemáticas para encontrar tais erros

## Laços Recursivos Infinitos



- Histórico do Projeto Sun JDK

<http://www.youtube.com/watch?v=GgK20Yv9QRk>



**Não capturados  
por casos de teste!**

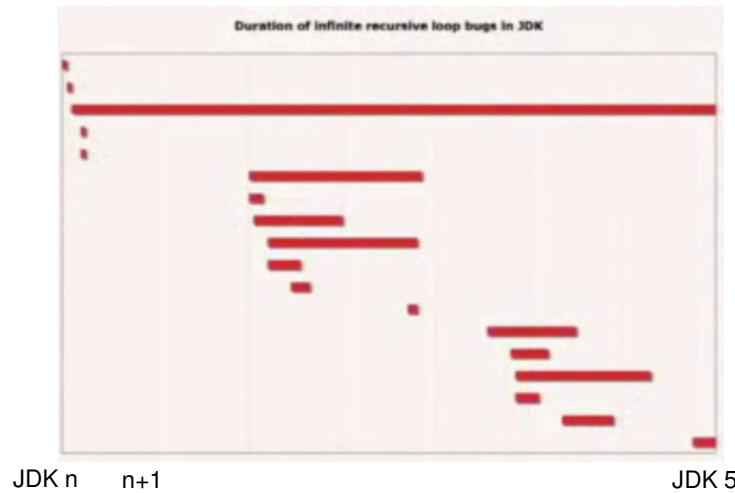
**Capturados  
posteriormente  
por ferramentas  
de análise estática!**

- laços removidos em um construto
- laços ainda presentes em um construto

## Duração de laços recursivos infinitos em JDK



- “Nascimento” e “morte” das instâncias de laços infinitos



Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

13 / 35

## Faltas recorrentes...



- ... são mais fáceis de serem detectadas com análise estática!
  - ... ou mesmo evitadas inicialmente através da identificação de padrões de programação


Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

14 / 35

Laboratório de Engenharia de Software

## O que é análise estática?



“Família de técnicas de análise de programa onde o programa não é executado (ao contrário de análise dinâmica)”


Foldoc 2005

Quais as diferenças entre teste e análise estática?

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
15 / 35

Laboratório de Engenharia de Software

## O que é análise estática?



- Análise estática é uma atividade para garantia de qualidade de software baseada na investigação do código fonte
  - Tipicamente baseado em catálogos de regras e recomendações
    - Universal ou por organização
- A motivação para usar análise estática são várias:
  - Encontrar defeitos (faltas) que irão causar erros e mesmo falhas [corretude]
  - Garantir que bons princípios de programação modular serão aderidos [manutenibilidade, reuso e (indiretamente) corretude]
  - Garantir que mal uso de recursos não seja incorporado no programa [eficiência/performance]
  - Garantir que o código fonte é bem estruturado, apresentado e documentado [manutenibilidade, reuso e (indiretamente) corretude]

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
16 / 35



## Por que se preocupar com análise estática?



- **Motivação inicial: "Ninguém é Perfeito!"**
  - ... e algumas das falhas são recorrentes
- Processos efetivos de teste identificam, em geral, 65% das faltas
  - o melhor processo de teste encontra 85% das faltas (defeitos) em um sistema crítico [Capers Jones]
- Na prática, um processo de teste típico irá expor 30-40% dos defeitos [Standish Group]
- Com análise estática você pode descobrir até 65% dos defeitos [Capers Jones]
- Teste e análise estática podem encontrar categorias complementares de faltas [Capers Jones]
  - devido a experiências diferentes de testadores e analisadores do código

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

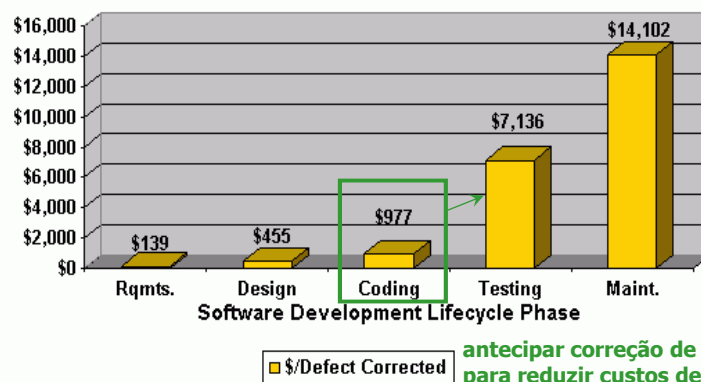
17 / 35

## Reduzindo custos com análise estática...



### Costs of Correcting Defects

Source: B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*



Set 2009


Alessandro Garcia © LES/DI/PUC-Rio

18 / 35

Laboratório de Engenharia de Software

## Por que se preocupar com análise estática?

- Teste é uma atividade essencial no processo de desenvolvimento de software, mas apresenta as seguintes limitações:
  - Teste é muito caro
  - *Teste aponta a presença de faltas, mas **ainda é necessário aplicar análise estática** e depuração para encontrá-las*
- Quando aplicar análise estática?
  - ... antes de teste
  - ... mais faltas são encontradas antes de teste, menos depuração será necessário mais tarde
  - ... mas teste ainda é necessário para certificarmos que os módulos se comportam de acordo com a especificação
    - Além disso, inspeção e revisão de código é sujeito a imperfeições pois são realizadas por seres humanos




Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
19 / 35

Laboratório de Engenharia de Software

## Tipos de Análise Estática


- **Revisão**: leitura crítica da documentação anotando os problemas encontrados
- **Inspeção**: revisão realizada segundo um procedimento bem definido e documentado, com papéis dos envolvidos muito bem delineados
- **Medição**: obtenção de medições ou avaliações segundo uma ou mais métricas bem definidas
- Em todos os casos:
  - os *artefatos sob análise* podem ser: código fonte, especificação, projetos, manual, casos de teste, etc...
  - objetivo é identificar faltas e não observâncias de padrões, inadequação dos próprios padrões, representações incompreensíveis, eficácia das métricas, etc...



Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
20 / 35

Laboratório de Engenharia de Software

## Tipos de Revisão




- Revisão pelo próprio autor
  - *problemas?*

Set 2009Alessandro Garcia © LES/DI/PUC-Rio21 / 35

Laboratório de Engenharia de Software

## Tipos de Revisão




- Revisão pelo próprio autor
- Revisão por colegas (*peer review*)
- Revisões progressivas (*round-robin review*)
  - objetivo: explorar as melhores habilidades de cada revisor
  - geralmente, é considerada uma forma de inspeção

Set 2009Alessandro Garcia © LES/DI/PUC-Rio22 / 35

Laboratório de Engenharia de Software

## Padrões de programação




- Cada padrão estabelece:
  - “**regras**” que são de aplicação obrigatória
    - “exceções” que estabelecem alternativas para regras
    - exemplo: *padrões de falta*
  - “**recomendações**” que devem ser seguidas na medida do possível
    - exemplo: *padrões de documentação*
- Os padrões podem ser adaptados a cada organização
- Catálogo de padrões tem dois tipos de usuários:
  - ajudam programadores a desenvolver artefatos de qualidade
  - ajudam inspetores de código – aplicando análise estática – a detectarem faltas em programas

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
23 / 35

Laboratório de Engenharia de Software

## Catálogos de Padrões de Programação



- Para evitar ou encontrar faltas: *padrões de faltas*
- Padrões de *documentação ou estilo*
  - facilita comunicação, compreensão e manutenção de um programa
  - evita introdução de faltas mais tarde através das modificações

**Observação:** maioria dos padrões de programação em [Staa, 00] (livro texto) são padrões de documentação/estilo;

Referência adicional para **padrões de falta**: **C Traps and Pitfalls**  
Andrew Koenig

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
24 / 35

## Exercício Simples de Revisão de Código



- Determinar se os programas estão se comportando corretamente (para certos valores de entrada), de acordo com a intenção do programador
- Caso não estiver, especifique um padrão de programação a ser seguido para evitar/identificar tal falta no futuro

## Expressões de Fluxo Controle



### Módulo **Contador**

(código parcial)

```
...
int contador = 0;

int incrementa( int val ) {
    if (val = 1 )
        contador++;

    printf ("\n Resultado: %d", contador);
    printf ("\n ---- \n");
}
...
```

0 = falso  
1 = verdadeiro

A intenção do programador é que a função *incrementa* vá incrementar o valor de contador somente se seu parâmetro, val, é igual a 1 ("verdadeiro").

- Qual é valor de **contador** ao chamar a função **incrementa** com *falso* (=0)?

## Expressões de Fluxo Controle



### Módulo Contador

(código parcial)

```
...
int contador = 0;

int incrementa( int val ) {
    if (val = 1 )
        contador++;

    printf ("\n Resultado: %d", contador);
    printf ("\n ---- \n");
}
...
```

A intenção do programador era que a função *incrementa* irá incrementar o valor de contador somente se seu parâmetro, *val*, é igual a 1 ("verdadeiro").

Entretanto, o operador de atribuição '=' é usado ao invés do operador de igualdade '=='; então, **contador** será incrementado independentemente do valor de **val**.

- Qual é valor de **contador** ao chamar a função **incrementa** com *falso* (=0)? **1**

- Regra: use o operador de igualdade == para *if*, *for* e laços *while*

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

27 / 35

## Por que é um...



... erro tão comum mesmo com programadores experientes?



Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

28 / 35

## Por que este é um erro comum mesmo...



- ... com programadores experientes?
  - `= e ==` são **visualmente similares**:

```
while (c == ' ' || c == '\t' || c == '\n')
    c = getc (f);
```
  - linguagens derivadas de Algol, tais como Pascal e Ada, usam
    - `:=` para atribuição; e
    - `=` para comparação (ao contrário de C, C++ e Java)
  - alguns compiladores C fornecem uma mensagem de alerta (warning) para condições da `e1 = e2`
    - Para evitar tais alertas, quando o resultado de uma atribuição deve ser comparado com zero, torne a comparação explícita:
    - Ao invés de ...

```
if (x = y)
    foo();
```

...escreva

```
if ((x = y) != 0)
    foo();
```

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

29 / 35

## Expressões Ambíguas



### Módulo Divisor

(código parcial)

```
...
[assuma que variáveis estão
definidas corretamente]

int div_ponteiro( int x ) {

    y = x/*p; /* p aponta para o divisor */;

    printf ("\n Resultado: %d", y);
    printf ("\n ---- \n");
}
...
```

A intenção do programador é que a função `div_ponteiro` vá dividir o valor de `x` pelo valor apontado por `p`, e atribuir o resultado para `y`.

- Assumindo que `x = 8` e valor apontado por `p` é `2`, qual será valor de **y**?

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

30 / 35

## Expressões Ambíguas



### Módulo Divisor

(código parcial)

```
...
[assume que variáveis estão
definidas corretamente]

int div_ponteiro( int x ) {

    y = x/*p; /* p aponta para o divisor */;

    printf ("\n Resultado: %d", y);
    printf ("\n ---- \n");
}
...
```

A intenção do programador é que a função *div\_ponteiro* vá dividir o valor de x pelo valor apontado por p, e atribuir o resultado para y.

Entretanto, o compilador entende '/\*' como um único elemento e, portanto, trata o resto da expressão como comentário.

- Assumindo que x = 8 e valor apontado por p é 2, qual será valor de **y**? **8**

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

31 / 35

## Expressões Ambíguas



- /\* começa comentário
  - o compilador tratará o texto do programa como comentário até que \*/ apareça
- Aplicando um dos seguintes padrões (regras) de programação evita ou resolve o problema:  
 $y = x / *p$  /\* p aponta para o divisor \*/;  
ou  
 $y = x/( *p)$  /\* p aponta para o divisor \*/;

- Regra: use espaços entre operandos e operadores, e/ou use parênteses para eliminar ambigüidades

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

32 / 35



## Cláusulas if...



**falta ou defeito**

```
public int map( int i ) {  
    if( i > 0 )  
    {  
        if( i > 10 )  
            return 10;  
        else  
            return -1;  
        return 0;  
    }  
}
```

**associação desejada**

**associação real**

Erros: momento em que tais retornos são gerados

**FALHA:** pode ser observada pelo usuário ou administrador... ou pelo próprio módulo cliente...

A **intenção** do programador é a seguinte:

|                       |             |
|-----------------------|-------------|
| $i > 10$              | retornar 10 |
| $i \text{ in } 1..10$ | retornar 0  |
| $i \leq 0$            | retornar -1 |

O comportamento real é:

|                       |             |
|-----------------------|-------------|
| $i > 10$              | retornar 10 |
| $i \text{ in } 1..10$ | retornar -1 |
| $i \leq 0$            | retornar 0  |

O compilador, na prática, associa a cláusula *else* com o *if* interno

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

33 / 35

## Cláusulas if...



- A ambigüidade pode ser sempre removida usando-se chaves
- **Regra:** evite confundir associação de blocos "else" através do uso de chaves

```
public int map( int i ) {  
    if( i > 0 ) {  
        if( i > 10 )  
            return 10;  
        }  
    else  
        return -1;  
    return 0;  
}
```

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

34 / 35

Laboratório de Engenharia de Software

## Qual o problema com o trecho de código?

LES

```

...
[assuma que variáveis estão
definidas corretamente]

switch (color) {
  case 1: printf ("\n vermelho");
  case 2: printf ("\n amarelo");
  case 3: printf ("\n azul");
}
...

```

A intenção do programador é que cada valor de **color** acarrete na impressão da cor correspondente.

- O que será impresso se valor de color = 2?

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
35 / 35

Laboratório de Engenharia de Software

## Qual o problema com o trecho de código?

LES

```

...
[assuma que variáveis estão
definidas corretamente]

switch (color) {
  case 1: printf ("\n vermelho");
           break;
  case 2: printf ("\n amarelo");
           break;
  case 3: printf ("\n azul");
           break;
}
...

```

A intenção do programador é que cada valor de **color** acarrete na impressão da cor correspondente.

Entretanto, o programa irá imprimir todos os valores, começando pela cor correta.

- O que será impresso se valor de color = 2?

**amarelo**  
**azul**

- Regra: use "breaks" sempre que o valor não deve ser mais checado

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
36 / 35

## Switch: C x Pascal



- Programadores habituados com Pascal tipicamente enfrentam tal problema

### C

```
switch (color) {  
    case 1: printf ("red");  
             break;  
    case 2: printf ("yellow");  
             break;  
    case 3: printf ("blue");  
             break;  
}
```

### Pascal

```
case color of  
    1: write ('red');  
    2: write ('yellow');  
    3: write ('blue')  
end
```

- Omissão de "breaks" em programas C também se tornam comum na medida que switch se torna mais complexo

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

37 / 35

## Desafio....



- ... Na definição de padrões de falta:
  - erros típicos dos programadores variam de linguagem para linguagem...
- Por exemplo, exemplos de padrões de faltas típicas em Java

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

38 / 35

Laboratório de Engenharia de Software

## Expressões Ambíguas

- O que este programa vai imprimir?

```

public class Printer {
    public static void main( String[] args ) {
        System.out.println( "2 + 9 = " + 2 + 9 );
    }
}

```

O código deveria imprimir "2 + 9 = 11"....

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
39 / 35

Laboratório de Engenharia de Software

## Expressões Ambíguas

- O que este programa vai imprimir?

```

public class Printer {
    public static void main( String[] args ) {
        System.out.println( "2 + 9 = " + 2 + 9 );
    }
}

```

- Regra: use parênteses para resolver ambiguidades**

O código deveria imprimir "11"....

Entretanto, **é impresso "29"**.

**"2 + 9 = " é uma string e os operadores '+' são tratados como concatenadores de string. 2 e 9 são assim interpretados como strings separados.**

Se o código 2 + 9 é delimitado com parênteses, a operação será tratada como operação de adição de inteiros, antes da concatenação com "2 + 9 = ".

Set 2009
Alessandro Garcia © LES/DI/PUC-Rio
40 / 35

| Regras de Inspeção                    |                           | LES  |
|---------------------------------------|---------------------------|--|
| Laboratório de Engenharia de Software | <b>Faltas de dados</b>    | <p>Todas as constantes foram nomeadas (apropriadamente)?</p> <p>Todas variáveis foram inicializadas antes dos respectivos valores serem utilizados?</p> <p>Existe possibilidade de "buffer overflow"?</p> <p>Caso strings de caracteres sejam usados, um delimitador está sendo utilizado?</p> <p>O limite superior de arrays deve ser igual ao tamanho do array ou tamanho - 1?</p> |
|                                       | <b>Faltas de controle</b> | <p>Para cada expressão condicional, a condição está correta?</p> <p>Todos os laços estão terminando de forma apropriada?</p> <p>As expressões compostas fazem uso correto de parênteses?</p> <p>Em blocos "case", todos os possíveis casos estão sendo explicitamente tratados?</p> <p>Se "break" é necessário no fim de cada bloco "case", tal "break" foi realmente incluído?</p>  |
| Set 2009                              |                           | Alessandro Garcia © LES/DI/PUC-Rio 41 / 35   |

| Regras de Inspeção                    |                                  | LES  |
|---------------------------------------|----------------------------------|--|
| Laboratório de Engenharia de Software | <b>Faltas de Entrada e Saída</b> | <p>Algum valor de entrada não esperado pode causar problemas?</p> <p>Todas as variáveis de entrada em parâmetros são usadas certamente?</p> <p>Todas as variáveis de saída são atribuídas um valor localmente antes de serem retornadas?</p> |
|                                       | <b>Faltas de Exceções</b>        | <p>Todas as condições excepcionais (p.e. fim de arquivo) são consideradas?</p>   |
| Set 2009                              |                                  | Alessandro Garcia © LES/DI/PUC-Rio 42 / 35   |

## Regras de Inspeção



### Faltas de Interface

Todas chamadas de funções ou métodos têm parâmetros na ordem correta?  
Se módulos compartilham espaços de memória, todos possuem o mesmo modelo da estrutura da memória compartilhada?

### Faltas de Armazenamento

Se uma estrutura de dados ligada é modificada, todos os ponteiros foram corretamente atualizados?  
Se armazenamento dinâmico é usado, o espaço necessário foi alocado corretamente?  
O espaço alocado é explicitamente desalocado na medida que não é mais necessário?

## Exemplos de Falhas Introduzidas...



- ... Por programadores experientes e “espertos”
- ... Presentes em sistemas de software reais, amplamente utilizados

“Using Static Analysis for Software Defect Detection”

<http://www.youtube.com/watch?v=GgK20Yv9QRk>

**Google TechTalks**

July 6, 2006

William Pugh

Uso e desenvolvimento da ferramenta: FindBugs

Google engEDU  
Tech Talks Channel

## Exemplos de Ferramenta: FindBugs



- Ferramenta para detecção de faltas em programas Java
  - “open source” project
  - analisa classfiles (e não código fonte)
  - usado no desenvolvimento de software na “Google”
    - ... também Oracle, Bank of America, etc...
  - gera arquivos XML com relatórios das faltas encontrados
  - total de downloads em SourceForge.net: +225.340 (2006)

## Alguns outros poucos exemplos de...




- analisadores estáticos:

Aivosto Project Analyzer, ASSENT, ccount, Cleanscape, LintPlus, ClearMaker, CMT++, CodeCompanion, CodeSurfer, Dependency, Walker floppy/fflow, ftnchek, Hindsight/SQA, Krakatau, LDRA, Testbed, (static analysis) McCabe QA, METRIC Metrics Tools, ParaSoft, CodeWizard, Jtest PC-lint/FlexeLint, PC-Metric, PolySpace Verifier, Plum Hall SQS, QA J QA C, QA C++, QA Fortran, QStudio for Java Pro Safer, C Toolset, sclc, SofAudit, SSW Code, Auditor, STATIC

Laboratório de Engenharia de Software

## Dúvidas do Trabalho





- Hoje: enviar emails com dúvidas para mim e Prof. Francisco
  - [afgarcia@inf.puc-rio.br](mailto:afgarcia@inf.puc-rio.br) e [fdmneto@gmail.com](mailto:fdmneto@gmail.com)
- Terça e quarta-feira:
  - Horário de monitoria e qualquer outro horário
  - Emails ou marcar horário presencial com Prof. Francisco

Set 2009

Alessandro Garcia © LES/DI/PUC-Rio

47 / 35





Laboratório de Engenharia de Software

## Aula 07

### Revisões, Inspeções e Padrões de Programação 1

*“Quantas regras!”* ALUNO ESPANTADO

*“A forma mais rápida de desenvolver um novo hábito é não ter outra alternativa”* [Hadden 1999]

Alessandro Garcia  
LES/DI/PUC-Rio  
Março 2009

