



Análise de mutantes Testes de mutação

Criado: ago/2011

Últ. alteração: mar/2013



Tópicos

- Principais técnicas de testes baseados em falhas
- Testes de mutação
- Análise de mutantes vivos
- Escore de mutação
- Operadores de mutação
- Exemplo de ferramenta



Testes baseados em falhas

- Modelo de base:
 - enganos típicos cometidos durante o processo de desenvolvimento
- Exemplos de técnicas
 - Estáticas
 - Semeadura de erros (*Error Seeding*) (Budd, 1981)
 - Análise de Mutantes (Teste de Unidade) (DeMillo et al., 1978)
 - Mutação de Interface (Teste de Integração) (Delamaro et al., 2001)
 - Mutação de modelos de estado (Testes baseados em modelos) (Fabbri et al. 1994)
 - Dinâmicas
 - Injeção de falhas (Arlat et al., 1989)



Hipótese do Programador Competente

- Um dos princípios básicos da técnica: um programador competente escreve programas corretos ou próximo do correto.
 - Assumindo a validade desta hipótese: *bugs* são introduzidos no programa por meio de **pequenos desvios sintáticos** que fazem com que a execução do produto leve a um comportamento incorreto.
 - Portanto, cada mutante tem uma única diferença sintática em relação ao programa original P .
 - O testador deve construir um conjunto de teste que mostre que tais modificações criaram produtos que não são corretos (Agrawal et al., 1989).



Efeito de acoplamento

- Segundo princípio da técnica (DeMillo et al., 1978):
 - Falhas (*bugs*) complexas são uma composição de *bugs* simples.
 - Conjuntos de teste que revelam falhas simples são também capazes de revelar falhas complexas (Budd, 1980).



Passos (1)

- Dados um produto que se deseja testar P e um conjunto de teste T . Os passos de aplicação do Teste de Mutação são:

1. Execução do produto original

- P é executado com T
- Se ocorrer um defeito (*failure*), o teste termina.
- Se nenhum defeito for identificado, P ainda pode conter *bugs* que T não foi capaz de revelar.

2. Geração dos mutantes

- P é submetido a um conjunto operadores de mutação que transformam P em M_1, M_2, \dots, M_n , denominados **mutantes** de P .

Operadores de mutação são regras que representam enganos freqüentes ou desvios sintáticos relacionados com uma determinada linguagem de programação.



Passos (2)

3. Execução dos mutantes

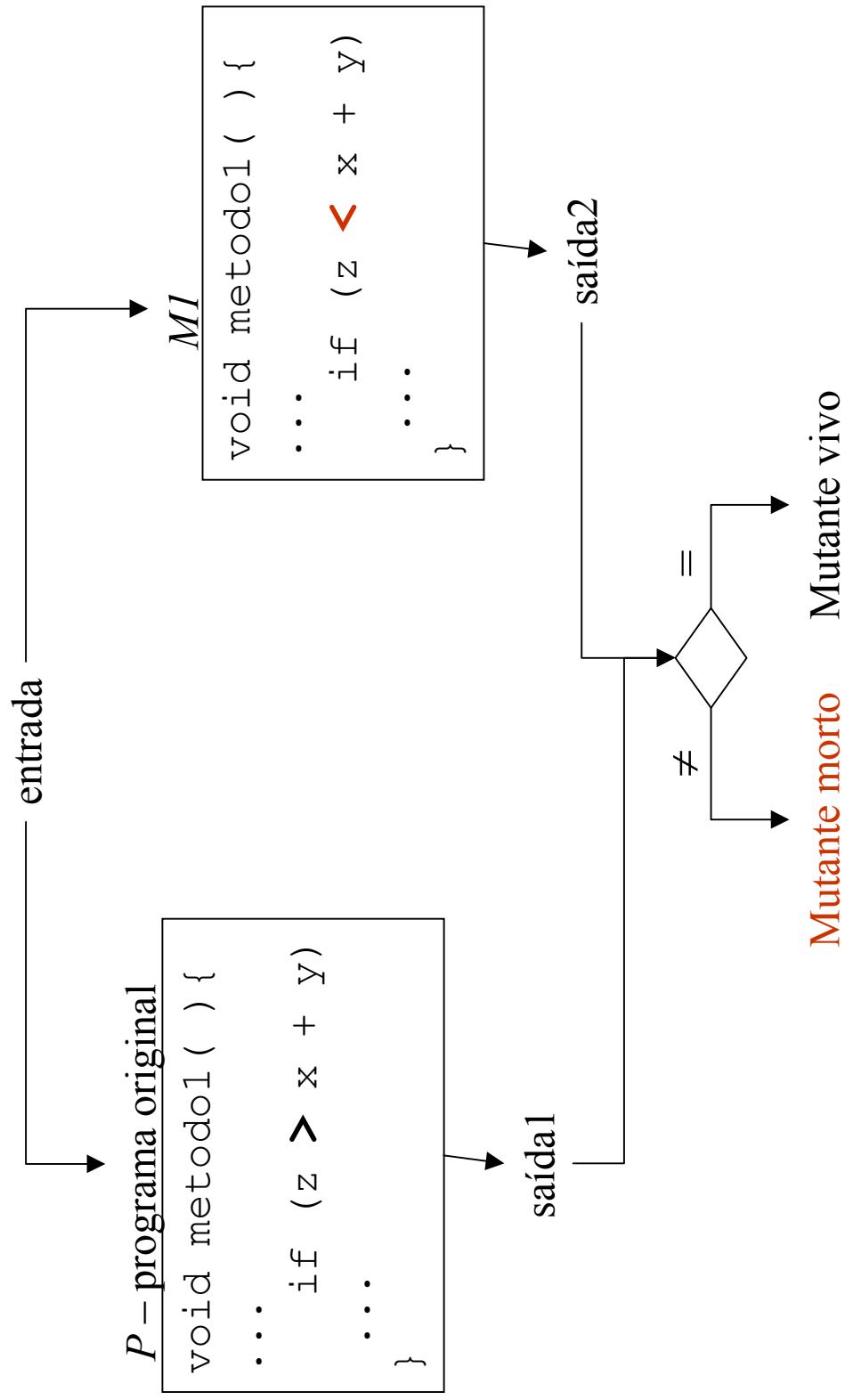
- Os mutantes são executados com o mesmo conjunto de teste T
- **Mutantes mortos**
 - resultados diferentes de P
 - **Mutantes vivos**
 - resultados idênticos ao de P
- ☞ O ideal é ter apenas mutantes mortos!

4. Análise dos mutantes vivos

- Identificar possível equivalência em relação a P
- Exportar uma fraqueza do conjunto de teste T



Esquema geral da técnica





Escore de Mutação

- Medida objetiva para avaliar a adequação de T em relação ao Teste de Mutação

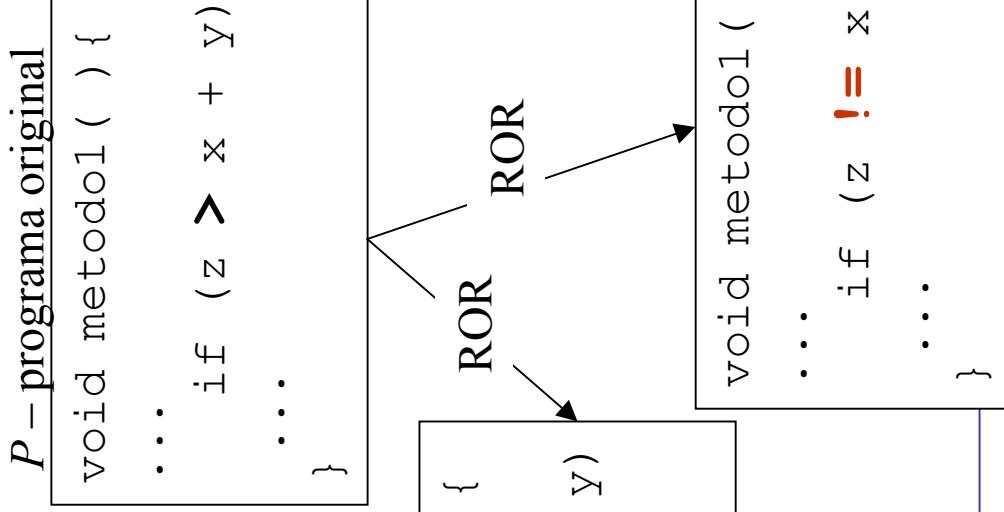
$$ms(P, T) = \frac{DM(P, T)}{M(P) - EM(P)}$$

- Em que:
 - $DM(P, T)$: número de mutantes mortos por T ;
 - $M(P)$: total de mutantes gerados;
 - $EM(P)$: número de mutantes equivalentes a P .

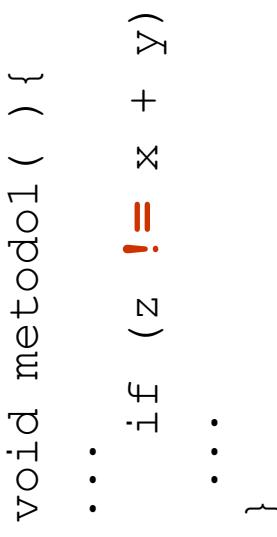


Criação de mutantes

- Uso de **operadores** de mutação para aplicar as pequenas modificações sintáticas aos programas



- A cada aplicação de um operador de mutação, um novo mutante *M* de *P* é criado



- Exemplo:
 - ROR (*Relational Operator Replacement*): substitui um operador relacional por outro



Exemplo de operadores – ferramenta µJava

Operador	Descrição
AOR	Substituição de operador aritmético por outro operador aritmético: unário, binário e de atalho (++, --)
AOI	Inserção de operador aritmético unário / de atalho
AOD	Supressão de operador aritmético unário / de atalho
ROR	Substituição de operador relacional por outro operador relacional
COR	Substituição de operador condicional por outro: &&, , &, , ^
COI	Inserção de operador condicional unário (!)
COD	Supressão de operador condicional unário (!)
SOR	Substituição de operador de deslocamento por outro
LOR	Substituição de operador lógico (bitwise) por outro: &, , ^
LOI	Inserção de operador lógico unário: ~
LOD	Supressão de operador lógico unário: ~
ASR	Substituição de operador de atribuição (de atalho) por outro: +==, -=, ...



Operadores para programas OO

- Existem várias propostas de operadores para criar falhas em aspectos específicos de OO, tais como:
 - Ocultação de informação (controle de acesso):
 - Operador para modificar o controle de acesso a atributos e métodos
 - Herança
 - Operadores para modificar uso de herança, tais como: suprimir “overriding” de métodos, suprimir “super”, ...
 - ...



Exemplo de operadores OO - μJava

Operador	Descrição
IOD	Suprime método sobrescrito
IOR	Troca nome de método sobrescrito
ISI	Insere <i>super</i>
JTI	Insere this
OMD	Suprime método sobrecarregado
EAM	Troca método de acesso



Análise de mutantes vivos

- Mutante equivalente
 - Um mutante M é dito **equivalente** a P se para qualquer dado de entrada $d \in D$, o comportamento de M é igual ao de P : $M(d) = P(d)$
 - Mutante revelador de falha (*fault-revealing*)
 - Um mutante é dito ser **revelador de falha** se para algum caso de teste t , tal que $P(t) = M(t)$, conclui-se que $P(t)$ não está de acordo com a especificação, ou seja, a presença de uma falha em P foi revelada
- ⌚ Se $t \notin T \rightarrow$ o que dizer de T ?



Limitações

- Principal problema é o grande número de mutantes gerados.
 - Mutantes precisam ser compilados e executados;
 - Mutantes vivos precisam ser analisados devido a possível equivalência.
- Requer bom conhecimento da implementação do produto para facilitar a análise de mutantes vivos:
 - Determinar se um mutante é equivalente ao programa original: problema indecidível



Vantagens

- É fácil de ser entendido para qualquer produto “executável”, seja especificação ou implementação.
- É um dos critérios de teste mais eficaz em para revelar falhas.
- É fácil automatizar a geração dos mutantes



Ferramentas

- Mothra (1980): mutação para programas em Fortran77
- Outras ferramentas:
fonte: http://en.wikipedia.org/wiki/Mutation_testing

- Jumble: Bytecode based mutation testing tool for Java
- MuJava: Bytecode based mutation system for Java programs
- PIT: Bytecode based mutation testing tool for Java
- Jester: Source based mutation testing tool for Java
- Heckle: Mutation testing tool for Ruby
- Nester: Mutation testing tool for C#
- Mutagenesis: Mutation testing tool for PHP
- Proteum: Mutation testing tool for C
- ...

- Usage :
- [1] Select files to test
 - [2] Select mutation operators to apply
 - [3] Push "RUN" button
 - [4] Wait with endurance. ^.^;

File		Java		Mutation Operator		
		Method-level		Class-level		
		Operator	Operator	Operator		
<input type="checkbox"/>	List.java	<input checked="" type="checkbox"/>	AORB	<input checked="" type="checkbox"/>	IHI	
<input type="checkbox"/>	Stack.java	<input checked="" type="checkbox"/>	AORS	<input checked="" type="checkbox"/>	IHD	
<input type="checkbox"/>	orgapachehelbcellConstants.java	<input checked="" type="checkbox"/>	AOIU	<input checked="" type="checkbox"/>	IOD	
<input type="checkbox"/>	orgapachehelbcellExceptionConstants.java	<input checked="" type="checkbox"/>	AOIS	<input checked="" type="checkbox"/>	IOP	
<input type="checkbox"/>	orgapachehelbcellRepository.java	<input checked="" type="checkbox"/>	AODU	<input checked="" type="checkbox"/>	IOR	
<input type="checkbox"/>	orgapachehelbcellAccessFlags.java	<input checked="" type="checkbox"/>	AODS	<input checked="" type="checkbox"/>	ISI	
<input type="checkbox"/>	orgapachehelbcellAttribute.java	<input checked="" type="checkbox"/>	ROR	<input checked="" type="checkbox"/>	ISD	
<input type="checkbox"/>	orgapachehelbcellReaderAttributeReader.java	<input checked="" type="checkbox"/>	COR	<input checked="" type="checkbox"/>	IPC	
<input type="checkbox"/>	orgapachehelbcellFormatException.java	<input checked="" type="checkbox"/>	COD	<input checked="" type="checkbox"/>	PNC	
<input type="checkbox"/>	orgapachehelbcellFormatException.java	<input checked="" type="checkbox"/>	COI	<input checked="" type="checkbox"/>	PMD	
<input type="checkbox"/>	orgapachehelbcellParser.java	<input checked="" type="checkbox"/>	SOR	<input checked="" type="checkbox"/>	PPD	
<input type="checkbox"/>	orgapachehelbcellCode.java	<input checked="" type="checkbox"/>	LOR	<input checked="" type="checkbox"/>	PCI	
<input type="checkbox"/>	orgapachehelbcellCodeException.java	<input checked="" type="checkbox"/>	LOI	<input checked="" type="checkbox"/>	PCC	
<input type="checkbox"/>	orgapachehelbcellConstant.java	<input checked="" type="checkbox"/>	LOD	<input checked="" type="checkbox"/>	PCD	
<input type="checkbox"/>	orgapachehelbcellConstantClass.java	<input checked="" type="checkbox"/>	ASRS	<input checked="" type="checkbox"/>	PRV	
<input type="checkbox"/>	orgapachehelbcellConstantCP.java			<input checked="" type="checkbox"/>	OMR	
<input type="checkbox"/>	orgapachehelbcellConstantDouble.java			<input checked="" type="checkbox"/>	OMD	
<input type="checkbox"/>	orgapachehelbcellConstantFieldref.java			<input checked="" type="checkbox"/>	OAN	
<input type="checkbox"/>	orgapachehelbcellConstantFloat.java			<input checked="" type="checkbox"/>	JTI	
<input checked="" type="checkbox"/>	orgapachehelbcellConstantInteger.java			<input checked="" type="checkbox"/>	JTD	
<input type="checkbox"/>	orgapachehelbcellConstantInterfaceMethodref.java			<input checked="" type="checkbox"/>	JSI	
<input type="checkbox"/>	orgapachehelbcellConstantLong.java			<input checked="" type="checkbox"/>	JSD	
<input type="checkbox"/>	orgapachehelbcellConstantMethodref.java			<input checked="" type="checkbox"/>	JID	
<input type="checkbox"/>	orgapachehelbcellConstantNameAndType.java			<input checked="" type="checkbox"/>	JDC	
<input type="checkbox"/>	orgapachehelbcellConstantObject.java			<input checked="" type="checkbox"/>	EOA	
<input type="checkbox"/>	orgapachehelbcellConstantPool.java			<input checked="" type="checkbox"/>	EOC	
<input type="checkbox"/>	orgapachehelbcellConstantString.java			<input checked="" type="checkbox"/>	EAM	
<input type="checkbox"/>	orgapachehelbcellConstantUtf8.java			<input checked="" type="checkbox"/>	EMM	
<input type="checkbox"/>	orgapachehelbcellConstantValue.java					
<input type="checkbox"/>	orgapachehelbcellDeprecated.java					
<input type="checkbox"/>	orgapachehelbcellDescendingVisitor.java					
		<input type="checkbox"/> None	<input checked="" type="checkbox"/> All			
						<input type="button" value="Generate"/>

Visualização dos mutantes

Mutants Generator **Traditional Mutants Viewer** **Class Mutants Viewer**

Select a Class : **Stack**

* Summary *	
Op	#
IHD	2
IHD_1	1
IHD_2	1
IOD	2
IOD_1	1
IOD_2	2
IOP	0
IOR	1
ISI	0
ISD	0
IPC	0
PNC	0
PMD	1
PPD	0
PCI	0
PCC	0
PCD	0
PRV	0
OMR	2
OMD	0
OAN	0
JTI	0
JTD	0
JSI	0
JSD	0
JID	1
JDC	0
EOA	0
EOC	0
EAM	0
EMM	0

Original

```

1 (line 11) Stack s1;=> List s1;
2
3     Stack s1;
4
5     12
6     13
7     14
8     15
9     16
10    17
11    18
12    19
13    20
14
15
16
17
18
19
20
21
22
23
24

```

Mutant

```

1 // This is mutant program.
2 // Author :ysma
3
4 public class Stack extends List
5 {
6
7     int top;
8
9     static int LENGTH= 5;
10
11    List s1;
12
13    List l1;

```

Total: 10

Resultados dos testes de mutação

TestCase Runner

Execute only class mutants Execute only traditional mutants Execute all mutants

Class : Stack TestCase: StackTest Time-Out: 3 seconds

Traditional Mutants Result

Op	#	Live	Killed
AORB	4	IHD_2	IHD_1
AORS	11	IHD_1	IHD_1
AOIU	8	IOD_2	IHD_2
AOIS	100	IOP_0	IOP_1
AODU	1	IOR_1	IHD_1
AODS	0	ISI_0	IHD_1
ROR	20	ISD_0	IHD_1
COR	4	IPC_0	IHD_1
COD	0	PNC_0	IHD_1
COI	0	PMD_1	IHD_1
SOR	0	PPD_0	IHD_1
LOR	0	PCI_0	IHD_1
LOI	29	PCC_0	IHD_1
LOD	0	PCD_0	IHD_1
ASRS	0	PRV_0	IHD_1
OMR	2	OMR_2	IHD_1
OMD	0	OMD_0	IHD_1
OAN	0	OAN_0	IHD_1
JTI	0	JTI_0	IHD_1
JTD	0	JTD_0	IHD_1
JSI	0	JSI_0	IHD_1
JSD	0	JSD_0	IHD_1
JID	1	JID_1	IHD_1
JDC	0	JDC_0	IHD_1
EOA	0	EOA_0	IHD_1
EOC	0	EOC_0	IHD_1
EAM	0	EAM_0	IHD_1
EMM	0	EMM_0	IHD_1

Total : 177 Total : 10

Class Mutants Result

Op	#	Live	Killed
AOIS_21	136	IHD_1	IHD_1
AOIS_22	41	IHD_1	IHD_1
AOIS_10	177	IHD_1	IHD_1
AOIS_100	23.0%	IHD_1	IHD_1
AOIS_24	AOIS_26	IHD_1	IHD_1
AOIS_11	AOIS_12	IHD_1	IHD_1
AOIS_12	AOIS_13	IHD_1	IHD_1
AOIS_13	AOIS_14	IHD_1	IHD_1
AOIS_15	AOIS_16	IHD_1	IHD_1
AOIS_16	AOIS_17	IHD_1	IHD_1
AOIS_17	AOIS_18	IHD_1	IHD_1
AOIS_18	AOIS_19	IHD_1	IHD_1
AOIS_19	AOIS_20	IHD_1	IHD_1
AOIS_20	AOIS_21	IHD_1	IHD_1
AOIS_21	AOIS_22	IHD_1	IHD_1
AOIS_22	AOIS_23	IHD_1	IHD_1
AOIS_23	AOIS_24	IHD_1	IHD_1
AOIS_24	AOIS_25	IHD_1	IHD_1
AOIS_25	AOIS_26	IHD_1	IHD_1
AOIS_26	AOIS_27	IHD_1	IHD_1
AOIS_27	AOIS_28	IHD_1	IHD_1
AOIS_28	AOIS_29	IHD_1	IHD_1
AOIS_29	AOIS_30	IHD_1	IHD_1
AOIS_30	AOIS_31	IHD_1	IHD_1
AOIS_31	AOIS_32	IHD_1	IHD_1
AOIS_32	AOIS_33	IHD_1	IHD_1
AOIS_33	AOIS_34	IHD_1	IHD_1
AOIS_34	AOIS_35	IHD_1	IHD_1
AOIS_35	AOIS_36	IHD_1	IHD_1
AOIS_36	AOIS_37	IHD_1	IHD_1
AOIS_37	AOIS_38	IHD_1	IHD_1
AOIS_38	AOIS_39	IHD_1	IHD_1
AOIS_39	AOIS_40	IHD_1	IHD_1
AOIS_40	AOIS_41	IHD_1	IHD_1
AOIS_41	AOIS_42	IHD_1	IHD_1
AOIS_42	AOIS_43	IHD_1	IHD_1
AOIS_43	AOIS_44	IHD_1	IHD_1
AOIS_44	AOIS_45	IHD_1	IHD_1
AOIS_45	AOIS_46	IHD_1	IHD_1
AOIS_46	AOIS_47	IHD_1	IHD_1
AOIS_47	AOIS_48	IHD_1	IHD_1
AOIS_48	AOIS_49	IHD_1	IHD_1
AOIS_49	AOIS_50	IHD_1	IHD_1
AOIS_50	AOIS_51	IHD_1	IHD_1
AOIS_51	AOIS_52	IHD_1	IHD_1
AOIS_52	AOIS_53	IHD_1	IHD_1
AOIS_53	AOIS_54	IHD_1	IHD_1
AOIS_54	AOIS_55	IHD_1	IHD_1
AOIS_55	AOIS_56	IHD_1	IHD_1
AOIS_56	AOIS_57	IHD_1	IHD_1
AOIS_57	AOIS_58	IHD_1	IHD_1
AOIS_58	AOIS_59	IHD_1	IHD_1
AOIS_59	AOIS_60	IHD_1	IHD_1
AOIS_60	AOIS_61	IHD_1	IHD_1
AOIS_61	AOIS_62	IHD_1	IHD_1
AOIS_62	AOIS_63	IHD_1	IHD_1
AOIS_63	AOIS_64	IHD_1	IHD_1
AOIS_64	AOIS_65	IHD_1	IHD_1
AOIS_65	AOIS_66	IHD_1	IHD_1
AOIS_66	AOIS_67	IHD_1	IHD_1
AOIS_67	AOIS_68	IHD_1	IHD_1
AOIS_68	AOIS_69	IHD_1	IHD_1
AOIS_69	AOIS_70	IHD_1	IHD_1
AOIS_70	AOIS_71	IHD_1	IHD_1
AOIS_71	AOIS_72	IHD_1	IHD_1
AOIS_72	AOIS_73	IHD_1	IHD_1
AOIS_73	AOIS_74	IHD_1	IHD_1
AOIS_74	AOIS_75	IHD_1	IHD_1
AOIS_75	AOIS_76	IHD_1	IHD_1
AOIS_76	AOIS_77	IHD_1	IHD_1
AOIS_77	AOIS_78	IHD_1	IHD_1
AOIS_78	AOIS_79	IHD_1	IHD_1
AOIS_79	AOIS_80	IHD_1	IHD_1
AOIS_80	AOIS_81	IHD_1	IHD_1
AOIS_81	AOIS_82	IHD_1	IHD_1
AOIS_82	AOIS_83	IHD_1	IHD_1
AOIS_83	AOIS_84	IHD_1	IHD_1
AOIS_84	AOIS_85	IHD_1	IHD_1
AOIS_85	AOIS_86	IHD_1	IHD_1
AOIS_86	AOIS_87	IHD_1	IHD_1
AOIS_87	AOIS_88	IHD_1	IHD_1
AOIS_88	AOIS_89	IHD_1	IHD_1
AOIS_89	AOIS_90	IHD_1	IHD_1
AOIS_90	AOIS_91	IHD_1	IHD_1
AOIS_91	AOIS_92	IHD_1	IHD_1
AOIS_92	AOIS_93	IHD_1	IHD_1
AOIS_93	AOIS_94	IHD_1	IHD_1
AOIS_94	AOIS_95	IHD_1	IHD_1
AOIS_95	AOIS_96	IHD_1	IHD_1
AOIS_96	AOIS_97	IHD_1	IHD_1
AOIS_97	AOIS_98	IHD_1	IHD_1
AOIS_98	AOIS_99	IHD_1	IHD_1
AOIS_99	AOIS_100	IHD_1	IHD_1

Total : 177 Total : 10

MuJava
Obtido em
ago/11 em:
<http://cs.gmu.edu/~offutt/mujava/>



Exercício

```
// Soma elementos nas posições pares de um vetor de inteiros
// Obtém menor elemento em posição ímpar no vetor
1. public class Exemplo {
2.     public static void main(String args[]) {
3.         int vet[] = {2, 5, 1, 8, 4, 9, 3, 7, 6, 8};
4.         int i;
5.         int somapar = vet[0];
6.         int menorimpar = vet[1];
7.         for (i=1; i<vet.length; i++) {
8.             if (i%2==0){
9.                 somapar += vet[i];
10.            } else{
11.                if (vet[i] < menorimpar) {
12.                    menorimpar = vet[i];
13.                }
14.            }
15.        }
16.        System.out.println("Soma dos pares = "+somapar);
17.        System.out.println("Menor nr. ímpar = "+menorimpar);
18.    }
19. }
```



Exercício (cont.)

- Para o código dado:
 - Qual a resposta esperada para o vetor fornecido como entrada de teste?
 - Suponha que foi criado um mutante substituindo, na linha 7, $i=1$ por $i=0$. A entrada de teste fornecida mataria esse mutante? Por que?
 - Crie uma entrada que mate esse mutante.
 - Crie uma entrada que não mate esse mutante.



Sumário principais pontos aprendidos