Teste de Software

ACH 2006
Marcos L. Chaim

Conteúdo

- Introdução
- Objetivo e limitações do teste de software
- Taxonomia dos testes
- Técnicas de teste
- Elaboração de casos de teste funcionais
- Documentação de teste
- Execução automática de casos de teste usando Junit
- Considerações finais

Exercício inicial

- O programa tem como entrada três valores inteiros que representam o comprimento de cada lado de um triângulo. A saída do programa é uma mensagem indicando se o triângulo é escaleno, isósceles ou equilátero.
- Escreva um conjunto de casos de teste que você julgar adequado para testar este programa.

Verificando seus casos de teste...

1. Há um caso de teste para um triângulo escaleno válido?
   - Note que casos de teste como 1, 2, 3 e 2, 5, 10 não constituem lados de um triângulo válido.
2. Há casos de teste para triângulos isósceles e equiláteros válidos?
3. Há no mínimo três casos de teste para triângulos isósceles e escaleno em que a única diferença entre eles é a ordem dos dados de entrada?
4. Há um caso de teste no qual um dos lados é zero?

Verificando seus casos de teste...

5. Há um caso de teste no qual um dos lados é negativo?
6. Há um caso de teste que verifica se a soma de dois lados é igual ao comprimento do terceiro?
   - a + b = c;
7. Há no mínimo três casos de teste que verifiquem se a regra acima se aplica aos três lados, isto é:
   - a + b = c; a + c = b; b + c = a

Verificando seus casos de teste...

8. Há um caso de teste que verifica se a soma de dois lados é menor do que o comprimento do terceiro?
   - a + b > c;
9. Há no mínimo três casos de teste que verifiquem se a regra acima se aplica aos três lados, isto é:
   - a + b > c; a + c > b; b + c > a
10. Há um caso de teste em que o valor de todos lados é zero?

Verificando seus casos de teste...

11. Há no mínimo um caso de teste que utilize valores não-inteiros?
12. Há no mínimo um caso de teste que utilize valores incorretos (e.g., dois inteiros ao invés de três)?
13. Para cada caso de teste, está especificada a saída esperada para os dados de entrada?

Por que testar software ?

- Teste é uma atividade necessária em qualquer processo de engenharia.
  - Processos de engenharia visam gerar produtos com características ***funcionais*** e de desempenho ***esperadas***.
  - Software é um produto desse tipo.
- Visão tradicional do teste:
  - detectar a presença de defeitos que tenham passado desapercebidos durante o desenvolvimento.

Visão moderna do teste

- O desenvolvimento é baseado em teste (*test-based software development*):
  - Conceito elaborado nas metodologias ágeis: XP, SCRUM, Crystal etc.
- Uma representação de software somente está completa se ela possuir testes associados:
  - trecho de código → testes unitários;
  - documentos de especificação (casos de uso, estórias de usuário) → testes de aceitação.
- Testes automatizados.

Teste x Métodos Formais

- ***Métodos formais*** permitem ***provar*** matematicamente que o programa desenvolvido está de acordo com a sua especificação.
- Então ***porque testar***, se já é construído correto?
- **Questões relevantes:**
  - você viajaria em um avião no seu primeiro vôo, sabendo apenas que ele foi exaustivamente simulado em um túnel de vento?
  - em outras palavras: quem garante que as provas matemáticas estão corretas?

Objetivos do teste

- *“Program testing can best show the* presence *of errors but never their* absence” – Edsger Dijkstra
- Em outras palavras:
  - não é possível garantir que um programa está correto através de teste.
- Para que serve o teste então ?

Objetivos do teste

- Objetivo imediato:
  - determinar entradas – contra-exemplos – que fazem as saídas obtidas na execução dos testes diferirem das saídas esperadas.
  - ou seja, ***refutar*** a assertiva de que o produto está ***correto***.
- Objetivo principal:
  - aumentar a ***confiança*** de que o software está correto.

Objetivos do teste

- Qualquer teste serve para aumentar a confiança?
  - **Não!**
- Teste aumenta a confiança na correção do software desde que:
  - seja sistemático;
  - exercite aspectos importantes da especificação e da estrutura do software;
  - exercite aspectos comumente relacionados com a ocorrência de defeitos.

Limitações do teste

- ***Executar*** todas as entradas do software é, em geral, ***impossível***.
- Exemplo – compilador de programas escritos em uma linguagem XYZ:
  - o domínio de programas possíveis de serem escritos e compilados em XYZ é ***infinito***;
  - logo, ***compilar*** todos os possíveis programas escritos em XYZ é ***impossível***.

Limitações do teste

- Além disso, o *teste* não requer somente a *execução* dos casos de teste
- Os resultados da execução *precisam* ser *validados*, isto é, conferidos quanto à sua *adequação* à especificação do sistema
- E se a entrada de dados é *infinita*... o trabalho de validação dos testes também é *infinito*

Limitações do teste

- *Correção coincidente*:
  - associada àquelas situações em que o resultado do teste é correto apenas para um particular dado de teste.
- Teste não manifestou falha:
  - será que se tivesse sido utilizado outro dado menos infeliz o teste não iria manifestar uma falha e revelar a presença do defeito?

Limitações do teste

- Questões de indecidibilidade ligadas ao teste:
  - mostrar que dois programas são equivalentes, isto é, possuem a mesma funcionalidade.
    - Programas previamente corretos não podem ser utilizados como padrão de comparação.
  - mostrar que existe um conjunto de dados de entrada que executa uma seqüência particular de comandos (caminho) do programa em teste.
    - Testador deve verificar o caminho para determinar se há ou não um caso de teste que o execute.

Limitações do teste

- O testador neófito pode estar pensando:
  - o teste é uma tarefa recheada de armadilhas.
  - é possível realizar teste de qualidade com tantas limitações?
- Limitações existem, mas é possível viver com elas.
- A maneira de tratá-las vai depender de quanto o projeto de software está disposto a gastar em teste.

Mágica do Teste

"A seleção de casos de teste é uma tarefa importante que os testadores executam. A seleção inapropriada resulta em ***testar demais***, ***testar pouco***, ou ***testar coisas erradas***. A análise adequada de riscos e a redução do conjunto infinito de possibilidades para um ***conjunto gerenciável*** e ***efetivo*** de testes é onde está localizada a ***magia do teste***"

Taxonomia dos testes

- Os testes podem ser classificados tendo como base diferentes ***pontos de vista*** do sistema, a saber:
  - os requisitos do software;
  - a abrangência do software;
  - a fase de desenvolvimento do software.

Taxonomia dos testes

- Teste baseado nos requisitos:
  - Sistema/aceitação.
  - Stress/desempenho;
  - Segurança.

Taxonomia dos testes

- Baseado na estrutura do software:
  - Unitário;
  - Integração;

Taxonomia dos testes

- Baseado na fase de desenvolvimento:
  - teste da especificação;
  - teste do projeto;
  - teste da implementação:
    - código;
    - interface: gráfica; entre subsistemas.
    - teste da documentação;
  - teste de regressão (manutenção).

Técnicas de testes

- As técnicas de teste são ortogonais aos tipos de teste.
- Podem ser utilizadas nos diferentes tipos de testes.
- Técnicas de teste:
  - Ad hoc;
  - Sistemáticas: estrutural (caixa-branca); funcional (caixa-preta); baseado em defeitos.

Técnicas de teste

- *Ad hoc* ou intuitivo:
  - baseado na intuição do testador sobre o que deve ser testado.
  - exemplo:
    - casos de teste elaborados a partir de palpites da equipe de avaliadores.

Técnicas de teste

- Sistemático:
  - implica que os casos de teste foram derivados utilizando uma técnica de teste.
- Objetivo das técnicas de teste:
  - garantir que aspectos considerados importantes da especificação (funcionalidade e comportamento) ou da estrutura do software tenham sido exercitados pelo menos uma vez por algum caso de teste.

Técnicas sistemáticas

- Estruturais:
  - verificam aspectos estruturais de uma representação do software.
  - ex: todos os blocos seqüenciais de um programa, todos os estados de um diagrama de estado, etc.
- Funcionais:
  - verificar a funcionalidade e o comportamento.
  - ex: cenários de casos de uso; especificação do programa

Técnicas sistemáticas

- Baseadas em defeitos:
  - derivam casos de teste a partir de defeitos específicos (ou classes de defeitos) comuns em linguagens de programação.
  - o objetivo é mostrar a presença ou a ausência de tais defeitos no programa.

Teste estrutural

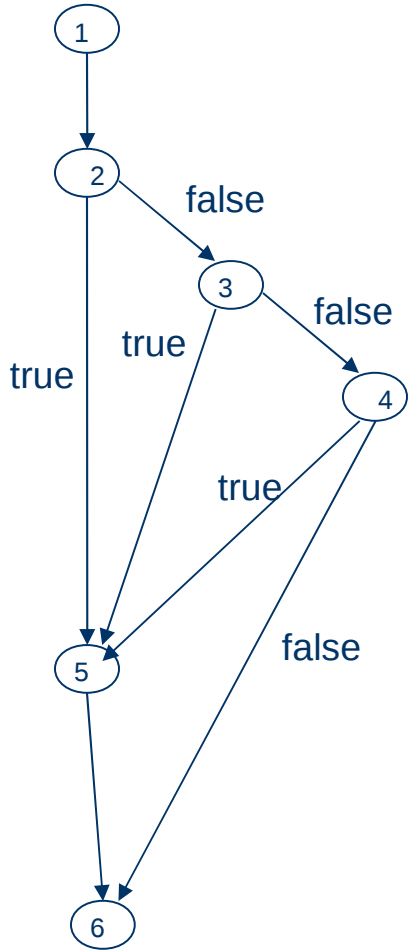
```
private boolean ehValido(int a, int b, int c)
{

boolean valido;
valido = true;

if ((a<=0) || (b<=0) ||(c<=0))
valido = false;

return valido;
}
```

1

2 3 4

5

6



Teste estrutural

- Critério de teste:
  - Determina *requisitos de teste* úteis para:
    - seleção de casos de teste que exercitem os requisitos do critério; ou
    - análise da adequação de um conjunto de casos de teste quanto aos requisitos do critério.

Teste estrutural

- Critérios de teste baseados em fluxo de controle:
  - **Critério Todos Nós:** exige que cada nó do grafo seja executado pelo menos uma vez;
  - **Critério Todos Ramos:** exige que cada ramo do grafo seja executado pelo menos uma vez;
  - **Critério Todos Caminhos:** exige que cada caminho, dado por uma seqüência finita de nós do grafo, seja executado pelo menos uma vez.

Teste estrutural

- Critérios de teste baseados em fluxo de dados:
  - estabelecem requisitos de teste que exigem a execução de caminhos entre a ***definição*** e o ***uso*** de uma variável;
  - por isso, os seus requisitos de teste são chamados de ***associações definição-uso***.
  - **Critério Todos Usos:** requer que todas as associações definição-uso dos tipos **(*i, j*, *x*)** e **(*i, (j,k), x*)** sejam exercitadas por casos de teste.

Teste estrutural

- Exercício:
  - Determine os requisitos de teste para os critérios ***todos os nós***, ***todos os ramos*** e ***todos os usos*** para o procedimento ***ehValido().***
  - Crie conjuntos de casos de teste que satisfaçam os requisitos de teste dos critérios ***todos os nós***, ***todos os ramos*** e ***todos os usos***.
  - Qual a dificuldade de utilização desses critérios?

Teste estrutural

- Exercício:
  - Determine os requisitos de teste para os critérios ***todos os nós***, ***todos os ramos*** e ***todos os usos*** para o procedimento ***ehValido().***
    - *Todos os nós:*
      - *Requisitos: executar pelo menos uma vez os nós 1; 2; 3; 4; 5 e 6.*
    - *Todos os ramos:*
      - *Requisitos: executar pelo menos uma vez os ramos (2,5); (2,3); (3,4); (3,5); (4,5) e (4,6).*
    - *Todos os usos:*
      - *Requisitos: executar pelo menos uma vez as adus (1,(2,5),a); (1,(2,3),a); (1,(3,4),b); (1,(3,5),b); (1,(4,5),c); (1,(4,6),c);. (1,5,valido); (1,6,valido) e (5,6,valido).*

Teste estrutural

- Exercício:
  - Crie conjuntos de casos de teste que satisfaçam os requisitos de teste dos critérios ***todos os nós***, ***todos os ramos*** e ***todos os usos***.
    - *Todos os nós:*
      - *Casos de teste: a=1, b=1, c=1; e a=0, b= 1, c=1.*
    - *Todos os ramos:*
      - *Casos de teste: a=1, b=1, c=1; a=1, b= 1, c=0; a=1, b= 0, c=1; e a=0, b= 1, c=1.*
    - *Todos os usos:*
      - *Casos de teste: a=1, b=1, c=1; a=1, b= 1, c=0; a=1, b= 0, c=1; e a=0, b= 1, c=1.*

Teste estrutural

- Exercício:
  - Qual a dificuldade de utilização desses critérios?
    - Impossível utilizá-los sem o apoio de uma ferramenta.
    - Alguns critérios (e.g., todos os usos) podem exigir muitos requisitos de teste, o que implica:
      - grande número de casos de teste a serem elaborados;
      - maior tempo de avaliação dos casos de testes da adequação ao critérios.

Técnicas baseadas em defeitos

- Baseadas na hipótese do ***programador competente***:
  - assume que os programadores experientes escrevem programas muito próximos do correto.
  - os programas incorretos contêm um desvio sintático que leva a resultados errados.
- Objetivo dos casos de teste é gerar saídas distintas para o programa em teste e para programas ***mutantes*** nos quais defeitos comuns foram incluídos.

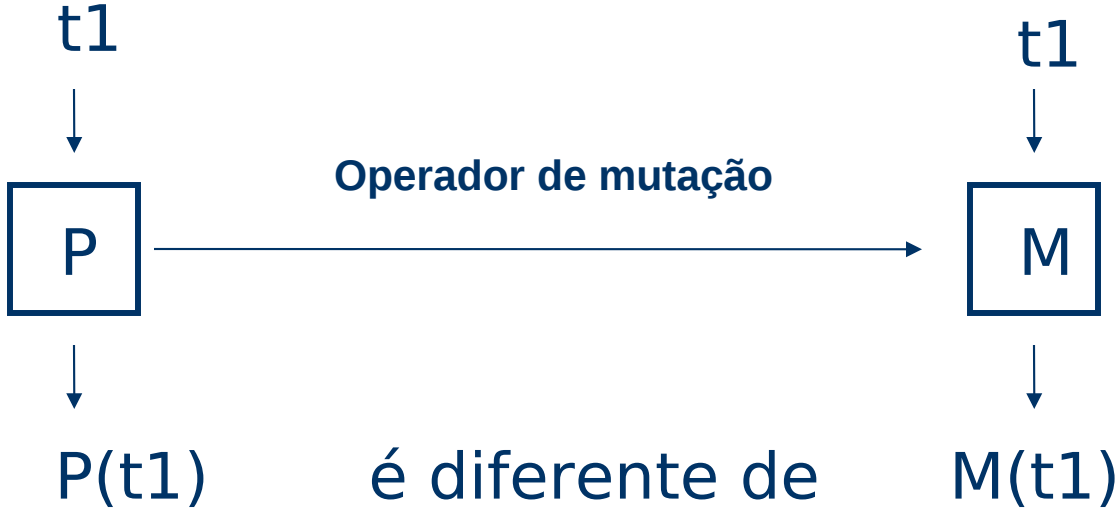
Técnicas baseadas em defeitos

- Análise de mutantes:

- Operador de mutação introduz um defeito comum:
  - elimina um linha;
  - altera um operador ou um operando.

Técnicas baseadas em defeitos

- Objetivo da análise de mutantes:



- Efeito de acomplamento:
  - ao derivar casos de teste que provocam saídas distintas para P e M, defeitos são detectados.

Técnicas baseadas em defeitos

- Estudos empíricos indicam que o teste baseado em defeitos é o mais eficaz na detecção de defeitos.
- Problemas:
  - para programas grandes, muitos mutantes são gerados → alto custo;
  - quando P(t1) = M(t1) é preciso determinar se P e M são equivalentes ou se P está errado. Não existe algoritmo que faça isto → **limitação do teste**!

Técnicas baseadas em defeitos

- Exercício:
  - Determine os mutantes do procedimento *ehValido()* após a aplicação do operador de mutação *eliminar um comando*.
  - Crie um conjunto de caso de casos de teste que faça distinção entre mutantes e procedimento.

Técnicas baseadas em defeitos

- Mutante 1:

```
private boolean ehValido(int a, int b, int c) {
boolean valido;
// valido = true;
if ((a<=0) || (b<=0) ||(c<=0))
    valido = false;
return valido;
}
```

Técnicas baseadas em defeitos

- Mutante 2:

```
private boolean ehValido(int a, int b, int c) {
boolean valido;
valido = true;
// if ((a<=0) || (b<=0) ||(c<=0))
    valido = false;
return valido;
}
```

Técnicas baseadas em defeitos

- Mutante 3:

```
private boolean ehValido(int a, int b, int c) {
boolean valido;
valido = true;
if ((a<=0) || (b<=0) ||(c<=0))
    /* valido = false */;
return valido;
}
```

Técnicas baseadas em defeitos

- Casos de teste:
  - a = 1, b = 1 e c = 1 (mutantes 1 e 3 mortos);
  - a = 0, b = 1 e c = 1 (mutante 2 morto).

Técnicas de teste funcionais

- Denominam-se também técnicas caixa-preta (black-box) ou comportamental.
- Os casos de teste são desenvolvidos a partir de documentos que especificam o comportamento pretendido do software.
- As técnicas mais conhecidas são:
  - particionamento de equivalência;
  - análise de valores limites.

Classes de equivalência

- Divide o domínio de entrada e saída do software em classes de equivalência para as quais devem ser gerados casos de teste.
- A intuição subjacente a essa heurística é que um único caso de teste é capaz de detectar a presença de defeitos relacionados com uma dada classe.

Classes de equivalência

- Dessa forma é possível diminuir o número de casos de teste necessários.
- Nem sempre essa intuição é verdadeira; porém, ela fornece um mecanismo sistemático de seleção de casos de teste.

Determinação das classes de equivalência

- Se uma entrada (saída) especifica uma faixa, então uma classe válida e duas inválidas devem ser selecionadas;
  - Condição: 0 < x < 10
    - Classe válida: 0 < x < 10;
    - Classe inválida: x <= 0;
    - Classe inválida: x >= 10

Determinação das classes de equivalência

- Se a entrada (saída) especifica um número de valores, então uma classe válida e duas inválidas devem ser identificadas.
  - Condição: imóvel pode possuir de um a seis proprietários
    - Classe válida:
      - de um a seis proprietários;
    - Classe inválida:
      - nenhum proprietário;
      - mais de 6 proprietários.

Determinação das classes de equivalência

- Se a entrada (saída) especifica um conjunto de valores, e suspeita-se que eles são tratados de maneira diferente, então deve ser identificada uma classe válida para cada valor e uma única classe inválida.
  - Condição: veículo deve ser: ônibus, caminhão, táxi, veículo de passeio ou motocicleta.
    - Classes válidas: ônibus, caminhão, táxi, veículo de passeio e motocicleta.
    - Classe inválida: trailer.

Determinação das classes de equivalência

- Se a entrada (saída) especifica uma determinada situação, devem ser identificadas uma classe válida e uma classe inválida.
  - Condição: primeiro caractere de um identificador deve ser uma letra.
    - Classe válida: primeiro caractere igual a letra.
    - Classe inválida: primeiro caractere diferente de letra.

Determinação das classes de equivalência

- Se uma entrada (saída) especifica uma condição booleana, então uma classe válida e uma inválida devem ser selecionadas.
  - Condição: os valores de entrada são inteiros positivos.
    - Classe válida: valor de entrada >= 0
    - Classe inválida: valor de entrada < 0

Identificando os casos de teste

- Passos:
  - ⇒ identifique as classes de equivalência;
  - ⇒ atribua um número único a cada classe de equivalência identificada;
  - ⇒ até que todas as classes de equivalência *válidas* tenham sido cobertas, escreva um novo caso de teste incluindo o maior número possível de classes *válidas* que ainda não foram cobertas;
  - ⇒ até que todas as classes de equivalência *inválidas* tenham sido cobertas, escreva um caso de teste para cada uma, e somente uma, classe *inválida* não coberta.

Exercício

- Utilizando o exemplo inicial (determinação do tipo do triângulo definido pelos seus lados), crie as classes de equivalência ***válidas*** e ***inválidas*** e os ***casos de teste***.

Classes de equivalência de entrada - Classes válidas e inválidas

Condições de Entrada		Classes válidas		Classes inválidas	
1		1.1		1.2	
2		2.1		2.2	
3		3.1		3.2	
4		4.1		4.2	

Classes de equivalência de entrada - Classe válidas e inválidas

Condição de Entrada		Classes válidas		Classes inválidas	
1	A > 0	1.1	A > 0	1.2	A <= 0
2	B > 0	2.1	B > 0	2.2	B <= 0
3	C > 0	3.1	C > 0	3.2	C <= 0
4	A = valor válido	4.1	A = valor válido	4.2	A = valor inválido
5	B = valor válido	5.1	B = valor válido	5.2	B = valor inválido
6	C = valor válido	6.1	C = valor válido	6.2	C = valor inválido
7	A + B > C	7.1	A + B > C	7.2	A + B <= C
8	B + C > A	8.1	B + C > A	8.2	B + C <= A
9	A + C > B	9.1	A + C > B	9.2	A + C <= B

Classes de equivalência de entrada – Casos de Teste

Caso de teste	Dados de entrada	Classes válidas satisfeitas	Classes inválidas satisfeitas
1			
2			
3			
4			

Classes de equivalência de entrada – Casos de Teste

Caso de teste	Dados de entrada	Classes válidas satisfeitas	Classes inválidas satisfeitas
1	2, 3, 4	1.1, 2.1, 3.1, 4.1, 5.1, 6.1, 7.1, 8.1, 9.1	--
2	-1, 3, 4	--	1.2, 7.2, 9.2
3	2, 0, 4	--	2.2, 7.2
4	4, 5, -2	--	3.2, 8.2, 9.2
5	‘a’, 5, 7	--	4.2
6	7, “b”, 8	--	5.2
7	5, 7, “c”	--	6.2
8	4, 7, 11	--	7.2
9	4, 1, 2	--	8.2
10	2, 5, 2	--	9.2

Exercício

- Crie casos de teste baseados nas classes de equivalência de saída.

Teste dos valores limites

- Essa heurística estabelece que os casos de teste devem ser derivados utilizando as classes de equivalência, selecionando-se elementos das classes que se localizam nas fronteiras, isto é, seus valores limites.

Teste dos valores limites

- O que são valores limites?
  - São as situações localizadas:
    - *exatamente* na fronteira das classes de equivalência;
    - *imediatamente antes ou imediatamente depois* das fronteiras entre classes de equivalência.
- Por que testar os valores limites?
  - Observou-se empiricamente que um grande número de defeitos eram detectados por valores localizados nas fronteiras das classes do domínio de entrada, e não pelos valores *centrais*.

Teste dos valores limites

- Diretrizes para a seleção de casos de teste baseado em valores limites (Myers, 1979):
  - se uma entrada (saída) especifica uma faixa limitada pelos valores *x* e *y*, então casos de teste devem ser projetados com os valores *x* e *y* e também com valores imediatamente superiores e inferiores a *x* e *y.*
  - Exemplo:
    - Se a faixa de entrada é –1.0 < x < 1.0, devem ser criados casos de teste com valores de x igual a *–1.0*, *1.0*, *1.1*, *-1.1*

Teste dos valores limites

- Se uma entrada (saída) especifica um número de valores, então casos de teste devem ser derivados para se exercitar o número máximo e o número mínimo de valores. Valores imediatamente acima e abaixo do número máximo e mínimo devem ser exercitados por casos de teste;
- Exemplo:
  - Um arquivo de entrada pode conter 1 a 25 registros, escreva casos de teste com 0, 1, 25 e 26 registros.

Teste dos valores limites

- As duas diretrizes anteriores devem ser igualmente utilizadas para testar as classes de equivalência do domínio de saída do programa.
- Exemplo:
  - Considere um programa que informa a faixa de desconto do imposto de renda. Existem três faixas:
    - 0,00 % – para valores até R$ 1.055,00;
    - 15,00 % -- para valores de R$ 1.055,01 até R$ 2.550,00; e
    - 27,50% -- para valores acima de R$ 2.550,00
  - Devem ser elaborados casos de teste que gerem taxas de 0,00%, 15%, 27,5% de desconto. É possível gerar um caso de teste que produza uma dedução negativa ou acima de 27,5%?

Exercício

- Utilizando o exemplo inicial (determinação do tipo do triângulo definido pelos seus lados), crie os testes de valores limites.

Casos de teste dos valores limites – Condições de entrada

Condição de entrada			Teste dos valores limites
1	condição	Valores limites	
2			
3			
4			

Casos de teste dos valores limites – Condições de saída

Condição de Entrada			Teste dos Valores Limites
1	A > 0	A = 0 A = 0.1 A = -0.1	0, 4, 5 0.1, 4, 5 -0.1, 4, 5
2	B > 0	B = 0 B = 0.1 B = -0.1	4, 0, 5 4, 0.1, 5 4, -0.1, 5
3	C > 0	C = 0 C = 0.1 C = -0.1	4, 5, 0 4, 5, 0.1 4, 5, -0.1
4	A = valor válido	--	--
5	B = valor válido	--	--
6	C = valor válido	--	--
7	A + B > C	A + B = C A + B = C + 0.1 A + B = C - 0.1	4, 5, 9 4, 5.1, 9 3.9, 5, 9
8	B + C > A	B + C = A B + C = A + 0.1 B + C = A - 0.1	9, 4, 5 9, 4.1, 5 9, 4, 4.9
9	A + C > B	A + C = B A + C = B + 0.1 A + C = B - 0.1	4, 9, 5 4.1, 9, 5 4, 9, 4.9

Casos de teste dos valores limites – Condições de saída

Condição de saída			Teste dos valores limites
1	condição	Valores limites	
2			
3			
4			

Casos de teste dos valores limites – Condições de saída

Condição de Saída			Teste dos Valores Limites
1	Escaleno	A != B e A != C e B != C	5, 7, 10
2	Isósceles	A = B e B != C A = C e B != C B = C e B != A	2, 2, 3 4, 5, 4 3, 4, 4
3	Equilátero	A = B = C	6, 6, 6

Documentação de teste

- Registra os casos de testes a serem executados e os resultados obtidos com a sua execução.
- Composta por:
  - **Plano de teste**: indica como proceder o teste do software e inclui o ambiente utilizado, as entradas e os resultados esperados
  - **Relatório de teste**: descreve o resultado do teste com informações sobre este resultado, mostrando as capacidades e as deficiências demonstradas pelo software

Plano de teste

- Deve ser possível ***repetir*** e ***entender*** o caso de teste
  - portanto, é importante caracterizar para cada teste:
    - seu objetivo;
    - sua configuração inicial (situação do sistema antes da execução do caso de teste) ; e
    - as funções a serem executadas, com os respectivos valores de entrada e as saídas esperadas.

Plano de teste - Exemplo

Teste (T3): Navegação, teclas de acesso rápido e alteração e exclusão de registros

- *estado inicial:* resultado do teste anterior (T2)
- *configurações: data* corrente e *usuário* user1

Função	Entrada	Saída Esperada
incluir animais	*Cod. Scl do animal:* 3964000673245 *reg*: nva0006 *nome:* brigite bardot *apelido:* bb *cat:* novilha	Msg.: Código do SCL com tamanho > 10
localizar	*Código SCL:* 3964006	Msg: código SCL de animal não cadastrado
localizar	*Código SCL:* 39640006	*Cod. Scl do animal:* 39640006 *reg*: nva0006 *nome:* brigite bardot *apelido:* bb *cat:* novilha
Alt+E (excluir)	*Alt+s (confirma a exclusão)*	Msg: registro excluído com sucesso
Alt+L (localizar)	*Nome:* vagoroso	*Cod. Scl do animal* 39650033 *reg:* rfo0033 *nome:* vagaroso *apelido:* vago *cat:* rufião
alterar	*Nome:* vigoroso *apelido*: vigor Alt+O (confirma a alteração)	Msg: dados alterados com sucesso

Plano de teste - Erros comuns

Teste (T3): Navegação, teclas de acesso rápido e alteração e exclusão de registros

- *estado inicial:* resultado do teste anterior (T2)
- *configurações: data* corrente e *usuário* user1

Função	Entrada	Saída Esperada
incluir animais	*Dados de um animal*	Msg de erro
localizar	*Código SCL:* 3964006	Msg de erro
localizar	*Código SCL:* 39640006	*Dados do animal exibidos corretamente*
Alt+E (excluir)	*Alt+s (confirma a exclusão)*	
Alt+L (localizar)	*Nome:* vagoroso	*Animal localizado*
alterar	*Nome:* vigoroso *apelido*: vigor Alt+O (confirma a alteração)	

Plano de teste - Erros comuns

Teste (T3): Navegação, teclas de acesso rápido e alteração e exclusão de registros

Função	Entrada	Saída Esperada
incluir animais	*Cod. Scl do animal:* 3964000673245 r*eg*: nva0006 *nome:* brigite bardot *apelido:* bb *cat:* novilha	Msg.: Código do SCL com tamanho > 10
localizar	*Código SCL:* 3964006	Msg: código SCL de animal não cadastrado
localizar	*Código SCL:* 39640006	*Cod. Scl do animal:* 39640006 r*eg*: nva0006 *nome:* brigite bardot *apelido:* bb *cat:* novilha
Alt+E (excluir)	*Alt+s (confirma a exclusão)*	Msg: registro excluído com sucesso
Alt+L (localizar)	*Nome:* vagoroso	*Cod. Scl do animal* 39650033 *reg:* rfo0033 *nome:* vagaroso *apelido:* vago *cat:* rufião
alterar	*Nome:* vigoroso *apelido*: vigor Alt+O (confirma a alteração)	Msg: dados alterados com sucesso

Exercício

- Utilizando o exemplo inicial (determinação do tipo do triângulo definido pelos seus lados) e os casos de teste já criados, elabore um plano de teste.

Exercício

Teste (T1): Teste de tipos de triângulo, de entradas inválidas e de triângulo inválido

- *estado inicial:* não há
- *configurações:* não há

Função	Entrada	Saída Esperada
Triang. escaleno	2, 3, 4	Triângulo escaleno
Triang. isósceles	3, 4, 4	Triângulo isósceles
Triang. equilátero	6, 6, 6	Triângulo equilátero
Lado negativo	-1, 3, 4	Lados do triângulo devem ter valor maior que 0
Lado caracter	5, ‘a’, 7	Lados do triângulo devem ser inteiros > 0
Triang. inválido	4, 7, 11	Valores informados não formam um triângulo

Relatório de teste

- Descreve os resultados observados com a execução dos casos de teste estabelecidos no plano de teste.
- É importante quando o relatório será entregue como resultado da atividade de teste.
- Na execução dos casos de teste:
  - se o resultado estiver de acordo com o esperado, utilizar um OK, indicando que nenhum defeito ou inconsistência foi encontrado.
  - caso contrário, deve ser utilizado um NOK, seguido de 1, 2 ou 3, indicando a severidade do defeito encontrado, bem como a descrição do que foi observado.

Relatório de teste

- Para descrever a severidade do defeito, pode ser utilizada a seguinte classificação:
  - 1 – baixa: é um problema, mas não causa uma saída imprópria; exemplo: erro na documentação ou emissão de mensagem pouco clara;
  - 2 – alta: produz uma saída incorreta;
  - 3 – crítica: interrompe a execução do software.

Relatório de teste - Exemplo

Teste (T3): Navegação, teclas de acesso rápido e alteração e exclusão de registros

- *estado inicial:* resultado teste anterior
- *configurações: data* corrente e *usuário* user1

Função	Entrada	Saída Esperada	Observado
incluir animais	*Cod. Scl do animal:* 3964000673245 *reg*: nva0006 *nome:* brigite bardot *apelido:* bb *cat:* novilha	Msg.: Código do SCL com tamanho > 10	NOk (2): Inclui o registro, mesmo com tamanho acima do limite permitido; Não emite mensagem de *feedback* da operação.
localizar	*Código SCL:* 3964006	Msg: código SCL de animal não cadastrado	Ok.
localizar	*Código SCL:* 39640006	*Cod. Scl do animal:* 39640006 *reg*: nva0006 *nome:* brigite bardot *apelido:* bb *cat:* novilha	Ok.
Alt+E (excluir)	*Alt+s (confirma a exclusão)*	Msg: registro excluído com sucesso	NOk (1): Não emite mensagem de *feedback* da operação.
Alt+L (localizar)	*Nome:* vagoroso	*Cod. Scl do animal* 39650033 *reg:* rfo0033 *nome:* vagaroso *apelido:* vago *cat:* rufião	Ok.
alterar	*Nome:* vigoroso *apelido*: vigor Alt+O (confirma a alteração)	Msg: dados alterados com sucesso	NOk (3): a operação alt+o na alteração de um registro leva a um erro que interrompe a execução do sistema. Após este erro, a base de dados ficou corrompida.

Exercício

- Utilizando o plano de teste elaborado e o programa existente no seu computador, execute seus casos de teste e gere o relatório correspondente.

Exercício

Teste (T1): Teste de tipos de triângulo, de entradas inválidas e de triângulc inválido

- *estado inicial:* não há
- *configurações:* não há

Função	Entrada	Saída Esperada	Observado
Triang. escaleno	2, 3, 4	Triângulo escaleno	Ok
Triang. isósceles	3, 4, 4	Triângulo isósceles	Ok
Triang. equilátero	6, 6, 6	Triângulo equilátero	Ok
Lado negativo	-1, 3, 4	Lados do triângulo devem ter valor maior que 0	Nok(1). Não informa o tipo do erro
Lado caracter	5, ‘a’, 7	Lados do triângulo devem ser inteiros > 0	Nok(1). Apesar de indicar erro na entrada, não informa o tipo do erro
Triang. inválido	4, 7, 11	Valores informados não formam um triângulo	Ok. Triângulo inválido

Execução automática de casos de teste com Junit

- *Rationale*:
  - Código pronto é aquele que possui testes automatizados associado.
- O que é o Junit?
  - *framework* para desenvolvimento de testes unitários automatizados → biblioteca java.
  - o conceito não é restrito a java: dunit, cunit etc. → xUnit!

Execução automática de casos de teste com Junit

- Objetivos:
  - gerar código que automatiza execução de casos de teste unitário;
  - limitar o código extra necessário para automatizar a execução dos casos de teste unitário;
  - limitar o julgamento humano (oráculo) na avaliação dos casos de teste → comparação automática das saídas obtidas com as saídas esperadas.
  - estimular o programador a escrever código de teste.

Execução automática de casos de teste com Junit

- Classe fundamental para criação de teste unitários:
  - ***TestCase*** → parte da biblioteca junit.jar (necessária no classpath).
- Métodos principais da classe ***TestCase***:
  - setUp();
  - tearDown().

Exemplo

- Considere a classe ***Money*** cujo objetivo é manipular quantias de dinheiro.
- Para criar os testes automatizados da classe ***Money*** é necessário antes criar a ***fixture –*** estrutura fixa para o teste automatizado.

Testes automatizados: *Fixture*

- Crie uma subclasse da classe ***TestCase***.
- Crie um construtor que aceita uma cadeia de caractere como parâmetro e passe-o para a ***superclasse***.
- Adicione variáveis tipo instância para serem utilizados nos testes.
- Reescreva o método *setUp().*
- Reescreva o método *tearDown()*.

importa biblioteca junit

```
import junit.framework.*;
public class MoneyTest extends TestCase {
private Money f12CHF;
private Money f14CHF;
private Money f28USD;
public MoneyTestCurso(String arg0) {
super(arg0);
}
public static void main(String[] args) {
}
protected void setUp() {
f12CHF= new Money(12, "CHF");
f14CHF= new Money(14, "CHF");
f28USD= new Money(28, "USD");
}
}
```

1 – cria subclasse de TestCase

3 - adiciona variáveis instância

2 – cria construtor

main

4 - setUp() – inicia variáveis

Exemplo

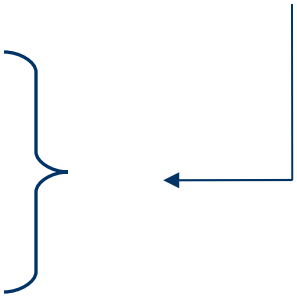
- Os testes automatizados da classe ***Money*** são métodos da classe ***MoneyTest***.
- Para comparar as saídas obtidas com as saídas esperadas, utiliza-se o objeto ***Assert***.

Testes automatizados: *test…()*

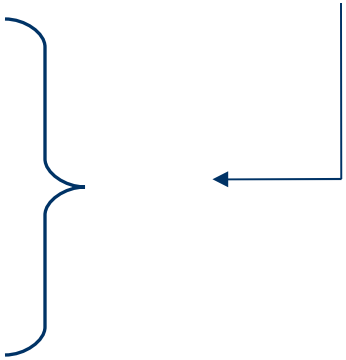
- Crie um método para cada teste automatizado.
- O nome do método deve ***sempre*** começar com ***test…()***.
- Incluir o método ***…TestRunner.run***() no método ***main()*** para invocar os métodos de teste.
- Utilize o objeto ***Assert*** para comparar saídas.

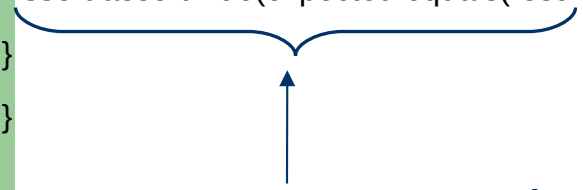
```
import junit.framework.*;
public class MoneyTest extends TestCase {
...;
public MoneyTestCurso(String arg0) {
super(arg0);
}
public static void main(String[] args) {
junit.textui.TestRunner.run(MoneyTest.class);
}
protected void setUp() {
...;
}
public void testSimpleAdd() {
Money expected= new Money(26, "CHF");
Money result= f12CHF.add(f14CHF);
Assert.assertTrue(expected.equals(result));
}
}
```

3 – …*TestRunner.run()*



1,2 – método *test…()*





4 – comparar saída com objeto Assert.

Teste automatizado – saídas

- Dois tipos de saídas:
  - Textual:
    - junit.textui.TestRunner.run()
  - Gráfica:
    - junit.swingui.TestRunner.run()
- ***Failures***:
  - Inconsistências detectadas via ***Assert***.
- ***Errors***:
  - Erros de execução:
    - *ArrayIndexOutOfBoundsException*

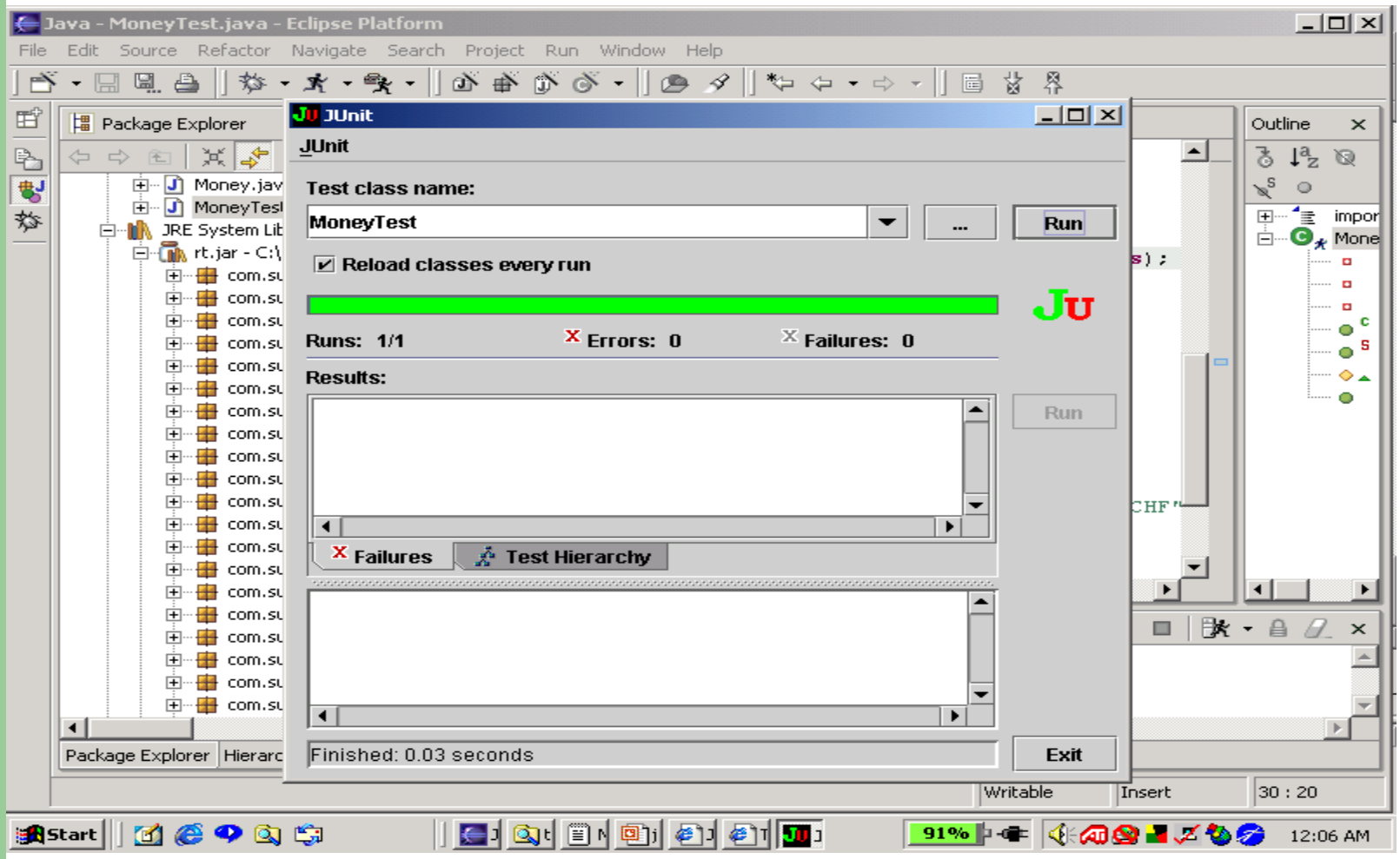
Teste automatizado – saída textual - ok

```
.
Time: 0.26

OK (1 test)
```

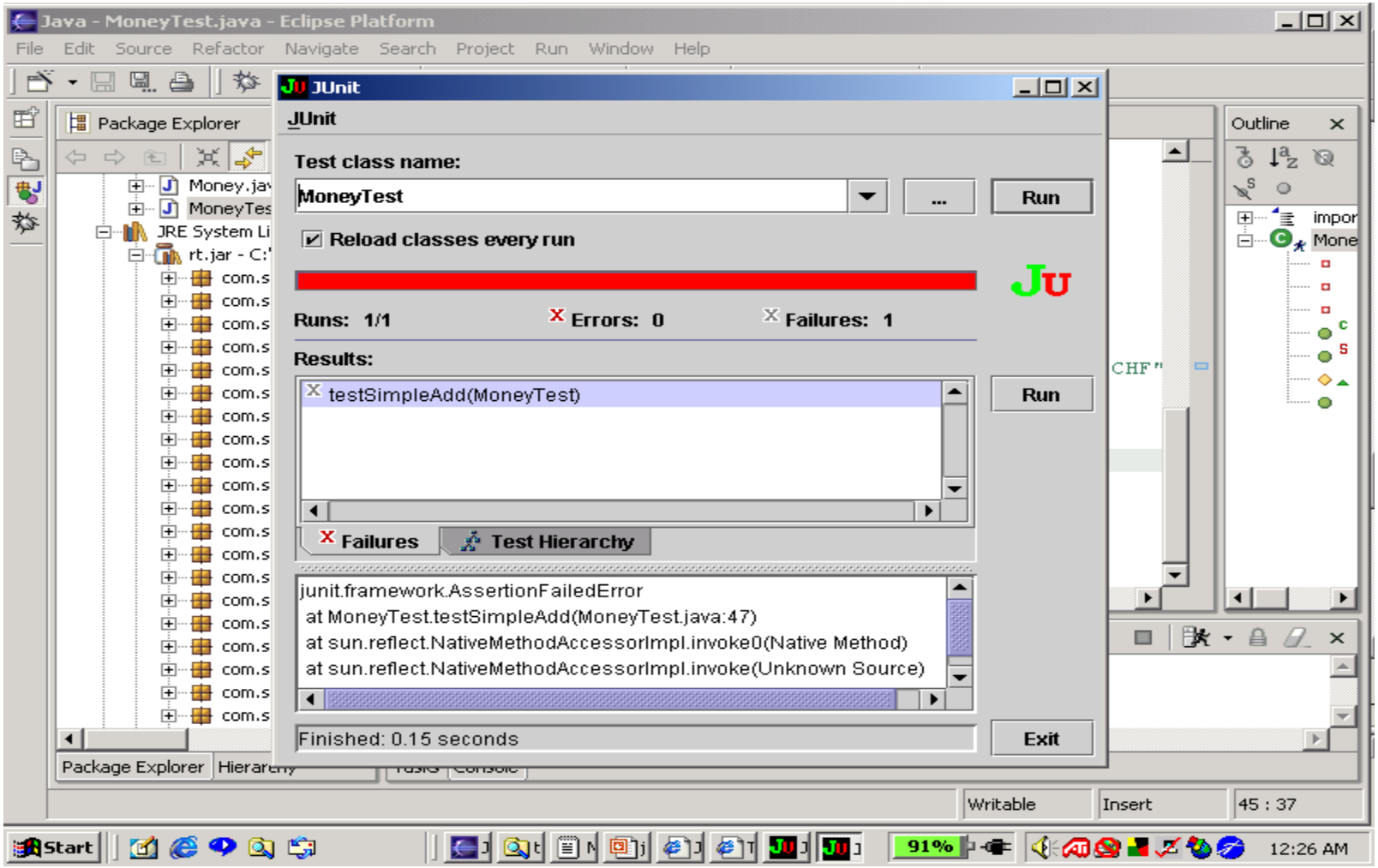
Teste automatizado – saída textual – nok

```
.F
Time: 0.02
There was 1 failure:
1) testSimpleAdd(MoneyTest)junit.framework.AssertionFailedError
at MoneyTest.testSimpleAdd(MoneyTest.java:47)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
at MoneyTest.main(MoneyTest.java:30)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

Teste automatizado – saída gráfica – ok



Teste automatizado – saída gráfica – nok



Exercício

- Foi desenvolvida uma classe ***triangulo*** que possui o método `String tipo(int a, int b, int c)` que retorna os seguintes valores:
  - STR_VALIDO = `"Válido";;`
  - STR_INVALIDO = `"Inválido";;`
  - STR_ESCALENO = `"Escaleno";;`
  - STR_ISOSCELES = `"Isósceles";;`
  - STR_EQUILATERO = `"Equilátero";`.
- Utilize o ambiente ***Eclipse*** para automatizar a execução dos casos de teste desenvolvidos para o programa do triângulo.

Considerações Finais

- Objetivos do teste:
  - refutar a assertiva de que o software está correto;
  - em falhando no primeiro objetivo, aumentar a confiança de que o software está correto.

Considerações Finais

- Limitações do teste:
  - teste de todo domínio de entrada é impossível.
  - indecibilidade:
    - determinar que um caminho é executável;
    - determinar que dois programas são equivalentes.
  - correção coincidente.
- Taxonomia dos testes.

Considerações Finais

- Técnicas de teste:
  - teste estrutural;
  - teste baseado em defeitos;
  - teste funcional.
- Documentação:
  - Plano e relatório de teste.
- Automação da execução dos testes:
  - Junit.