

10

J820

Testes de performance com JUnitPerf

Testes automáticos de performance

- Ferramentas como JProbe, Optimizelt e outros profilers oferecem diferentes tipos de estatísticas sobre gargalos no código
 - Desvantagem: são aplicações interativas e requerem avaliação dos resultados
- Para testes automáticos de performance pode-se usar o **JUnitPerf**, que é um **decorador** sobre testes de JUnit
 - JUnitPerf oferece classes que permitem construir objetos que recebem testes existentes do JUnit e acrescentam neles avaliação de performance
 - JUnitPerf não altera testes existentes. Pode-se ainda rodar os testes sem o JUnitPerf

Testes de performance

- **JUnitPerf** (www.clarkware.com)
 - Coleção de decoradores para medir performance e escalabilidade em testes JUnit **existentes**
- **TimedTest**
 - Executa um teste e mede o **tempo** transcorrido
 - Define um tempo máximo para a execução. Teste falha se execução durar mais que o tempo estabelecido
- **LoadTest**
 - Executa um teste com uma **carga** simulada
 - Utiliza timers para distribuir as cargas usando distribuições randômicas
 - Combinado com **TimerTest** para medir tempo com carga
- **ThreadedTest**
 - Executa o teste em um thread separado

Como usar JUnitPerf

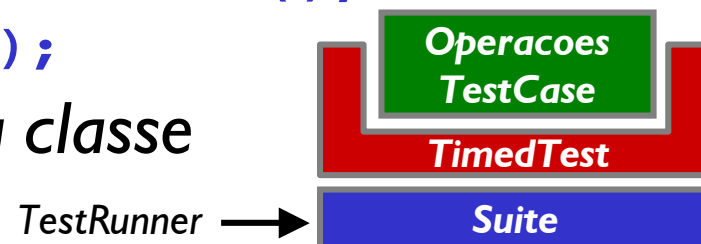
- *Primeiro é preciso estimar os valores ideais para execução dos testes*
 - *1. Escreva testes JUnit para o seu código*
 - *2. Execute um profiler para descobrir os gargalos. Utilize os dados obtidos como parâmetros para estabelecer os valores máximos aceitáveis para cada método*
 - *3. Escreva testes do JUnit (se não existirem) para os trechos críticos quanto à performance*
 - *4. Escreva um TimedTest do JUnitPerf para cada teste novo e execute-o. O teste deve falhar. Se passar, não há problema de performance com o código*
 - *5. Trabalhe no código até que os testes passem.*

Como funciona?

- Exemplo de execução de um **TimedTest**
 1. Recuperar instante de tempo (antes da execução do JUnit test case)
 2. Chamar `super.run(TestResult)` para executar o teste JUnit original
 3. Recuperar o tempo transcorrido após a execução do teste JUnit. Se o tempo for maior que o permitido então uma exceção `AssertionFailedError` é provocada (o que faz o teste falhar)
- **TimedTest** estende `junit.framework.Test`
 - Um **TimedTest**, ao ser criado recebe como argumento um teste JUnit e o tempo máximo que deve durar.

Como usar um TimedTest

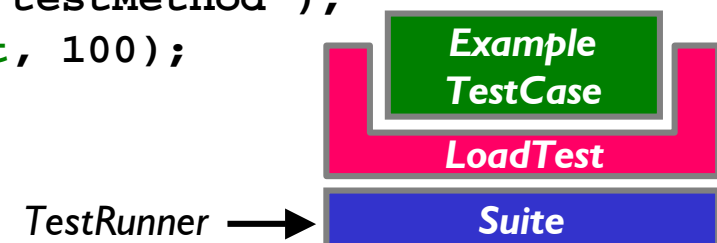
- Importe `com.clarkware.junitperf.TimedTest` e escreva um método `public static Test suite()`
- Em `suite()`, crie uma instância de um teste existente. A instância deve conter apenas um teste. Não use
`Test testCase = OperacoesTest.suite();`
- Use
`Test testCase = new OperacoesTest("testSoma");`
- Depois passe a instância do teste para um `TimedTest`
`Test timedTest = new TimedTest(testCase, 2000);`
- Coloque tudo em um `TestSuite`
`TestSuite suite = new TestSuite();`
`suite.addTest(timedTest);`
e retorne no método `suite()` da classe



LoadTest

- Permite simular carga, por exemplo, vários usuários acessando a aplicação ao mesmo tempo
 - *Essencial para descobrir problemas que podem surgir em ambientes multiusuário (por exemplo: problemas de concorrência e integridade de dados usados por vários usuários)*
- Para simular os cenários de teste há dois timers
 - *ConstantTimer: espera tempo fixo entre requisições (default: zero)*
 - *RandomTimer: espera um tempo aleatório entre requisições*
- Exemplo: LoadTest simples (executa uma vez por usuário) com 100 usuários simultâneos

```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    Test test = new ExampleTestCase("testMethod");  
    Test loadTest = new LoadTest(test, 100);  
    suite.addTest(loadTest);  
    return suite;  
}
```



Outros exemplos de LoadTest

- *LoadTest com 100 usuários, cada um executando o teste 10 vezes*

```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    Test test = new ExampleTestCase("testMethod");  
    Test loadTest = new LoadTest(test, 100, 10);  
    suite.addTest(loadTest);  
    return suite;  
}
```

- *Usando intervalos aleatórios (com variação entre 500 e 1000 ms) entre requisições*

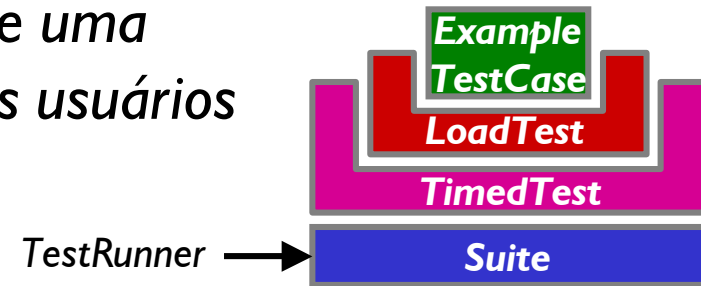
```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    Timer timer = new RandomTimer(1000, 500);  
    Test test = new ExampleTestCase("testMethod");  
    Test loadTest = new LoadTest(test, 100, 10, timer);  
    suite.addTest(loadTest);  
    return suite;  
}
```

Tempo total de uma operação multiusuário

- Pode-se especificar o tempo máximo de uma operação quando executada por muitos usuários

```
import com.clarkware.junitperf.*;
import junit.framework.*;
```

```
public class ExampleLoadTest extends TestCase {
    public ExampleLoadTest(String name) { super(name); }
    public static Test suite() {
        TestSuite suite = new TestSuite();
        Timer timer = new ConstantTimer(1000);
        int maxUsr = 10;
        int iters = 10;
        long maxElapsedTime = 20000;
        Test test = new ExampleTestCase("testOneSecondResp");
        Test loadTest = new LoadTest(test, maxUsr, iters, timer);
        Test timedTest = new TimedTest(loadTest, maxElapsedTime);
        suite.addTest(timedTest);
        return suite;
    }
}
```



TestCase existente

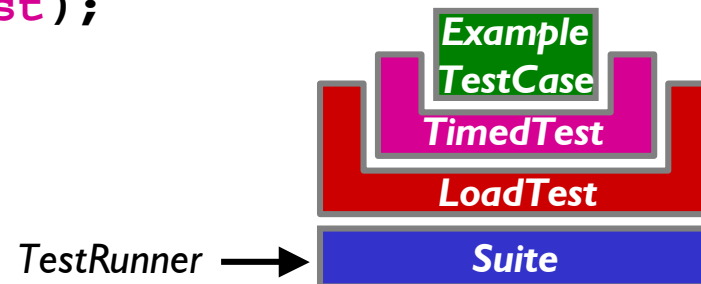
Decorador de carga

Decorador de tempo

Tempo de resposta para cada cliente


- Exemplo: garantir que **cada** usuário tenha um tempo de resposta de 3 segundos quando houver 100 usuários simultâneos
 - **Não é a mesma coisa** que testar se a operação inteira executou em tempo aceitavel para todos os usuários (exemplo anterior)
 - O tempo de resposta de **um** usuário pode ser aceitável, ou do grupo como um todo mas o de alguns usuários pode não ser (alguns podem ter resposta muito rápida e compensar lentidão de outros)

```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    Test test = new ExampleTestCase("testStress");  
    Test timedTest = new TimedTest(test, 3000);  
    Test loadTest = new LoadTest(timedTest, 100);  
    suite.addTest(timedTest);  
    return suite;  
}
```



- *1. Utilize os arquivos fornecidos*
 - *Rode os testes JUnit da aplicação fornecida. Garanta que todos executam com sucesso*
 - a) Escreva testes JUnitPerf para que **testConcatena()** não demore mais que 0.001 segundo para ser executado: troque as concatenações de String por StringBuffer para fazê-lo passar.*
 - b) Acrescente uma carga de 50 usuários e execute **testConcatena()**. Use synchronized no código para evitar a corrupção dos dados.*
 - c) Crie um novo teste que garanta que os 50 usuários levem no máximo 0.050 segundos para completar a operação.*

- [1] *Documentação JUnitPerf*. junitperf.sourceforge.net
- [2] Hightower/Lesiecki. *Java Tools for eXtreme Programming*. Wiley, 2002
- [3] Eric Burke & Brian Coyner. *Java eXtreme Programming Cookbook*. O'Reilly, 2003



Curso J820
Produtividade e Qualidade em Java:
Ferramentas e Metodologias
Revisão 1.1

© 2002, 2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br