Auditoria e Qualidade de Sistemas

Prof. Edgard Davidson C. Cardoso

Qualidade de Sistemas

JUNITPERF E DBUNIT

una

Automatização de Testes de Desempenho

- JUnitPerf é uma extensão do JUnit, um conjunto de decoradores de testes JUnit, que é utilizado para medir o desempenho e a escalabilidade dos testes referenciados
  - Clarkware Consulting - www.clarkware.com - desenvolvedora e mantenedora do JUnitPerf
  - JUnitPerf oferece classes que permitem construir objetos que recebem testes existentes do JUnit e acrescentam neles avaliação de desempenho
  - JUnitPerf não altera testes existentes. Pode-se ainda rodar os testes sem o JUnitPerf

Testes de Desempenho

- 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 (acesso concorrente e iterações)
  - 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

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 desempenho
  - 4. Escreva um TimedTest do JUnitPerf para cada teste novo e execute-o. O teste deve falhar. Se passar, não há problema de desempenho com o código
  - 5. Trabalhe no código até que os testes passem.

TimedTest

- 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.

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 e retorne no método suite() da classe
  - `TestSuite suite = new TestSuite();`
  - `suite.addTest(timedTest);`

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;
}
```

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;
}
```

Multiusuários

- 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;
   }
}
```

Tempo de resposta

- 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 aceitável 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
  TipoTrianguloTest("testObterTipoTriangulo");
  Test timedTest = new TimedTest(test, 5000);
  Test loadTest = new LoadTest(timedTest, 150);
  suite.addTest(loadTest);
  return suite;
}
```

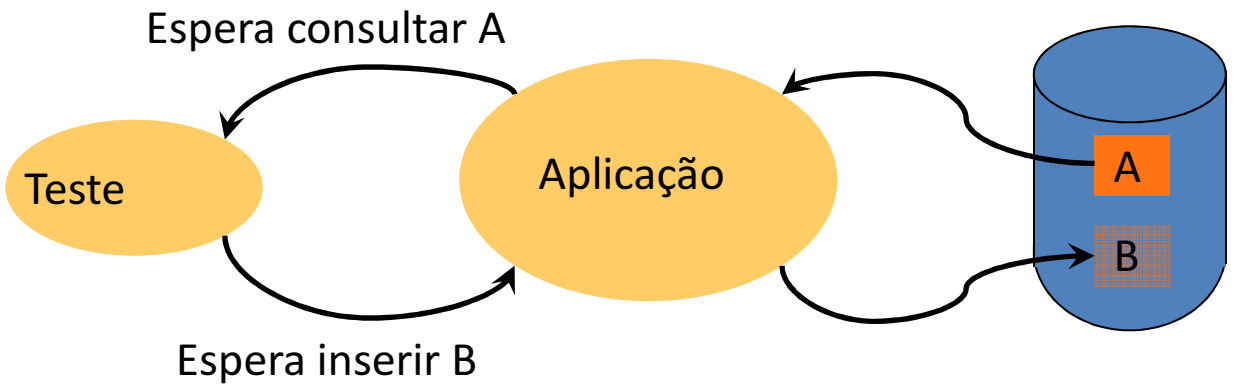
DBUNIT

Cenário

- Problema
  1. Colocar o banco de dados em um estado conhecido entre um teste e outro, ou seja, garantir que o conjunto de dados de teste estejam corretamente no BD.

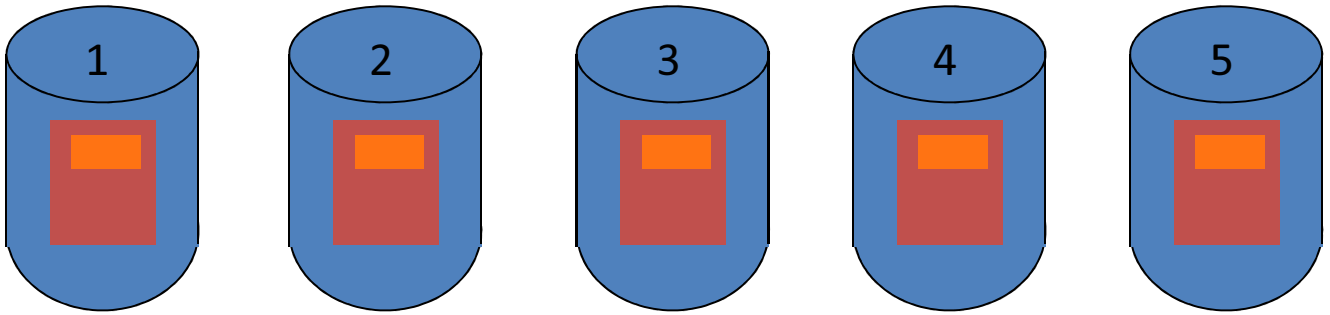
     Se vc estiver testando:

     - Inserções e Atualizações : Deve garantir que o registro não exista no BD.
     - Consultas e Remoções: Deve garantir que o registro exista no BD.



Cenário

- Problema
  2. Manter as várias instancias de bancos de dados do projeto sincronizadas em termos de estrutura e dados de testes.
     - Existem várias instâncias do BD no ambiente de construção do projeto para a realização de testes:
       1. Desenvolvimento: Cada desenvolvedor possui a sua instância.
       2. Homologação de desenvolvimento.
       3. Homologação da área de testes:
       4. Homologação do cliente.
       5. Produção.



Cenário

- Solução: DBUnit
  - Estende o JUnit
  - Classes Importantes:
    - IDataBaseConnection: Representa a conexão com o BD
    - IDataSet: Representa o conjunto de dados de teste que deve ser inserido no BD.
    - DataBaseOperation: Representa uma operação que pode ser realizada no BD antes e depois de cada caso de teste.

DBUnit

- Escrevendo Testes (1/6)

1. Criar um arquivo ou instância, contendo o conjunto de dados de teste (IDataSet).
   - Instância de BD:

```
IDatabaseConnection con = new DatabaseConnection(jdbcConnection);
IDataSet fullDataSet = con.createDataSet();
```

   - Arquivo XML

```
FlatXmlDataSet.write(fullDataSet,new FileOutputStream("full.xml"));
```

DBUnit

- Escrevendo Testes (2/6)

2. Incluir os seguintes imports na classe de teste:
   - `import org.dbunit.*;`

3. Estender a classe de caso de teste DataBaseTestCase (esta classe estende a classe do JUnit TestCase).
   - `public class SampleTest extends DatabaseTestCase`

DBUnit

- Escrevendo Testes (3/6)

4. Implementar os métodos getConnection() e getDataSet().
    - getConnection retorna a conexão com o banco de dados

```
protected IDatabaseConnection getConnection() throws Exception
```

    - getDataSet retorna o objeto que representa o conjunto de dados de teste

```
protected IDataSet getDataSet() throws Exception
```

DBUnit

- Escrevendo Testes (4/6)

5. Implementar (opcionalmente) os métodos getSetUpOperation() e getTearDownOperation() que são executados respectivamente antes e depois de cada caso de teste.
    - getSetUpOperation indica a ação a ser realizado no BD antes da execução de cada caso de teste.

```
protected DatabaseOperation getSetUpOperation() throws Exception
```

    - getTearDownOperation indica a ação a ser realizado no BD após a execução de cada caso de teste.

```
protected DatabaseOperation getTearDownOperation()throws Exception
```

DBUnit

- Escrevendo Testes (5/6)
  5. (cont.) Ações (DataBaseOperation) que podem ser realizadas, no BD, com o conjunto de dados de teste:

DataBaseOperation.UPDATE	Assume que a tab. existe no BD.
DataBaseOperation.INSERT	Assume que a tab. ñ existe no BD.
DataBaseOperation.DELETE	Deleta apenas os dados de teste.
DataBaseOperation.DELETE_ALL	Deleta todo o conteúdo das tab do BD.
DatabaseOperation.TRUNCATE	Trunca as tab. do BD e de testes.
DataBaseOperation.REFRESH	Atualiza dados existentes e insere dados não existentes no BD.
DataBaseOperation.CLEAN_INSERT	DELETE_ALL + INSERT (default do DBUnit executado antes de cada caso de teste).

DBUnit

- Escrevendo Testes (6/6)

6. Implementar (opcionalmente) os métodos setup() e tearDown(), como no JUnit.

   - `protected void setUp()`
   - `protected void tearDown()`

7. Defina um ou mais testes com métodos públicos **`testXXX()`** como no JUnit.

   - `public void testXXX()`

una

DBUnit

```
import org.dbunit.*;

public class SampleTest extends DatabaseTestCase {

public SampleTest(String name) { super(name); }

protected IDatabaseConnection getConnection() throws Exception {...}
protected IDataSet getDataSet() throws Exception {...}

protected DatabaseOperation getSetUpOperation() throws Exception {
    return DatabaseOperation.REFRESH; }
protected DatabaseOperation getTearDownOperation() throws Exception {
    return DatabaseOperation.NONE; }

protected void setUp() {…}
protected void tearDown(){…}

public void testTrianguloEq() {…}
```

DBUnit

- É possível utilizar o DBUnit com uma suite de testes JUnit já existente. Neste caso basta, por exemplo, incluir os comandos abaixo no método setup() da classe de testes do JUnit.

```
IDatabaseConnection con = new DatabaseConnection(jdbcConnection);
IDataSet dataSet = con.createDataSet();
DatabaseOperation.CLEAN_INSERT.execute(con, dataSet);
```

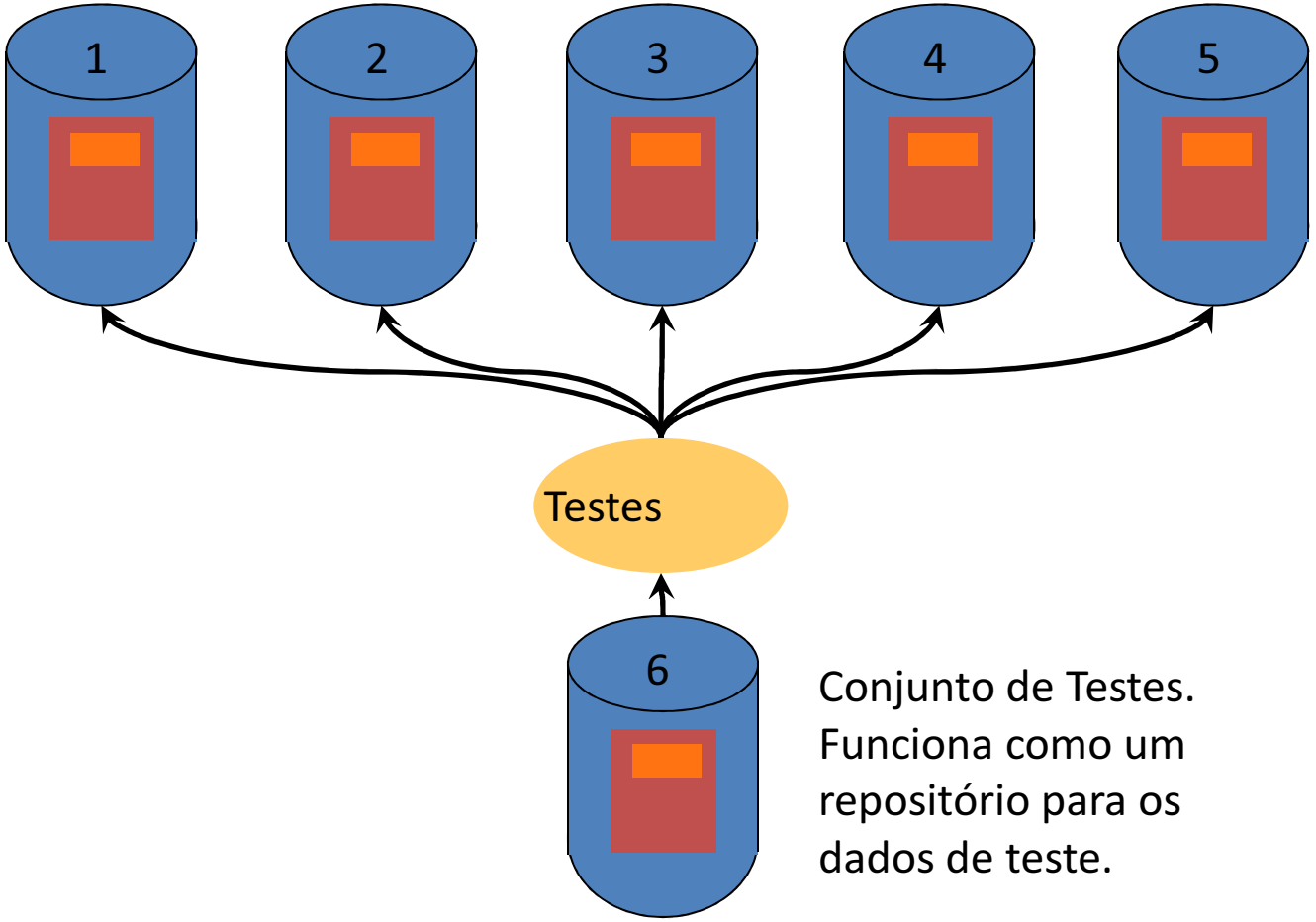
una

DBUnit

- Boas práticas
  - Utilize uma instância de desenvolvimento por desenvolvedor
  - Utilize múltiplos e pequenos conjuntos de dados de teste (IDataSet).
  - Realize, sempre que possível, o setup dos dados de teste (DataBaseOperation) por suite de testes, e não por caso de teste.
  - Reutilize a conexão do BD. Para tanto, sobrescreva o método `closeConnection()` na classe de testes, com uma implementação vazia.

una

Esquema de instâncias utilizando o DBUnit



Conjunto de Testes. Funciona como um repositório para os dados de teste.

Demais Ferramentas para Teste

- **Ferramentas de construção:** Ant, Maven 2
- **Ferramentas de controle de versão:** CVS, Subversion
- **Integração continua:** Continuum, CruiseControl, LuntBuild, Hudson, Openfire
- **Teste de unidade:** JUnit, TestNG, Cobertura
- **Teste de Integração, funcional e de performance:** StrutsTestCase, DbUnit, JUnitPerf, JMeter, SoapUI, the Sun JDK Tools, Eclipse, Selenium, FEST
- **Ferramentas de Métricas de Qualidade:** Checkstyle, PMD, FindBugs, Jupiter, Mylyn, QALab, StatSCM, StatSVN
- **Ferramenta de gerenciamento de erros:** Bugzilla, Tra
- **Ferramenta para documentação técnica:** SchemaSpy, Doxygen, UmlGraph)