



www.eteg.com.br

R. Sergipe, 472 – Pilotis – Funcionários

CEP: 30.130-170 Belo Horizonte MG

Telefone: (31) 3889-0990

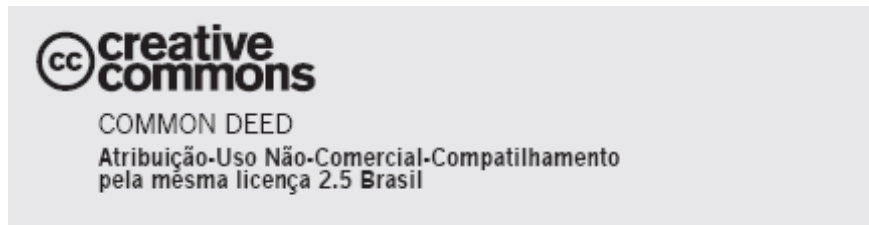
Introdução ao Apache Maven

Curso Introdução
ao Apache Maven

Copyright 2008 by Eteg Tecnologia da Informação Ltda

Autor: Walter dos Santos Filho

Editor: Walter dos Santos Filho



Este trabalho está licenciado sob uma Licença Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil. Para ver uma cópia desta licença, visite <http://creativecommons.org/licenses/by-nc-nd/2.5/br/> ou envie uma carta para Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Mesmo tendo sido tomadas todas as precauções na confecção deste material, a Eteg Internet não se responsabiliza por perdas ou danos decorrentes das informações contidas neste documento, seja por omissão ou erros. Sugestões e comentários podem ser enviados para: treinamento@eteg.com.br

Eteg, eBabel, Capri, Integrato, Sinon, Safira e Calipso e o logo Eteg são marcas da Eteg Tecnologia da Informação Ltda. Sun, Java, JDK são marcas registradas da Sun Microsystems, Inc. Microsoft, Microsoft Windows, .Net, C# e Visual Basic são marcas registradas da Microsoft Corporation. Websphere, Rational Unified Process e Rational Rose são marcas registradas da IBM. Oracle 9i, Oracle9iAs são marcas da Oracle Corporation. JBoss é marca registrada da JBoss Inc.

Todos os demais produtos descritos neste material são marcas registradas de seus respectivos proprietários.

Caso você não tenha obtido este documento diretamente da Eteg, verifique em nosso site se você está com a versão mais recente.



www.eteg.com.br

Índice Geral

Introdução ao Apache Maven.....	1
Capítulo 1 – Introdução.....	5
Processo de Construção de um Software.....	6
Visão Conceitual.....	7
Visão Física.....	9
Objetivos e Funcionalidades do Maven.....	11
Ant versus Maven.....	13
Modelo de Objetos de Projetos (POM).....	14
Repositório Maven.....	17
Repositórios e Coordenadas.....	19
Introdução aos Plug-ins.....	22
Revisão e Laboratório.....	24
Capítulo 2: Instalação, Utilização e Principais Recursos.....	25
Instalando o Maven 2.....	26
Criando o Primeiro Projeto com Archetype.....	27
O Plug-in Archetype.....	30
Alterando o POM.....	31
Compilando e Testando o Projeto.....	33
Empacotando e Disponibilizando Localmente.....	37
Gerando Documentação Javadoc.....	39
Configurando o Maven para uso com o Eclipse.....	41
Definindo Dependências para o Projeto.....	43
Revisão e Laboratório.....	46
Capítulo 3: Trabalhando com vários Projetos.....	47
Motivação.....	48
Definindo o Projeto Principal (Master).....	49
Criando uma Nova Biblioteca.....	52
Criando o Projeto para o Site.....	53
Herança entre POMs.....	56
Empacotando o Projeto.....	58
Revisão e Laboratório.....	59
Capítulo 4: Configurações de Plug-ins e Relatórios.....	60
Introdução.....	61
Configurando a Fase de Compilação.....	63
Configurando o Empacotamento da Aplicação.....	65
Propriedades.....	67
Plug-in Report e Website do Projeto	69
Utilizando o Plug-in Report	72
Utilizando outros Plug-ins para Relatórios.....	75
Relatórios JavaDoc e JXR.....	77
Disponibilizando o Website	79
Revisão e Laboratório.....	80
Bibliografia.....	81

Capítulo 1 – Introdução

Processo de Construção de um Software

Processo de Construção de Software

- Cada dia, novos requisitos tornam a construção do software mais complexa
- Manter um processo que possibilite gerar versões de forma consistente e rapidamente é um desafio
- O Maven é um software para construção de software integrado com a internet
 - Código livre
 - Permite gerenciar projetos complexos (multiprojetos)

A construção de um software atualmente não é mais simplesmente construir um projeto simples, monolítico e individual. Cada dia novos requisitos são necessários para se criar as chamadas “soluções corporativas”: segurança, robustez, acessibilidade, integração, etc.

Manter o controle do projeto, de forma que seja possível criar uma versão do software que seja consistente e padronizada é um desafio e por isto, várias ferramentas foram criadas: make, Ant, build, entre outras. O Apache Maven, com seu sistema de construção de software, é a evolução deste tipo de ferramenta através da integração com a internet.

Sobre o Maven

O Maven é um projeto de código livre, mantido pela Apache Software Foundation, criado originalmente para gerenciar o complexo processo de criação do projeto Jakarta Turbine. Desde seu início, o Maven tem sido utilizado por projetos de código livre e também proprietário. Com a versão 2, o Maven passou de uma ferramenta de construção para uma ferramenta complexa de gestão de construção de software, com várias funcionalidades e aplicável a maioria dos cenários de desenvolvimento de software.

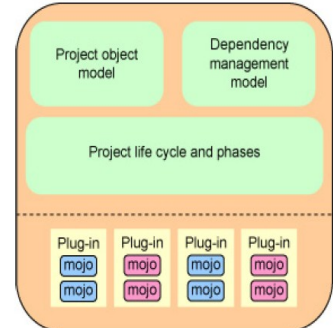
Importante: Neste curso, sempre que nos referirmos ao Maven, estaremos tratando da versão 2, também chamada de Maven 2.

Visão Conceitual

Visão Conceitual

- Nesta visão, os principais componentes do Maven são:
 - Project Object Model (POM): Configura o funcionamento do Maven.
 - Dependency Management Model: Gerencia dependências entre artefatos
 - Ciclo de vida e fases: Controlam o processo de construção
 - Plug-ins: São extensões do Maven

Figure 1. Maven 2 object and operation models
Maven 2



Para compreender o processo de construção de software com o Maven, é importante entender um conjunto de modelos conceituais que explicam como as coisas funcionam.

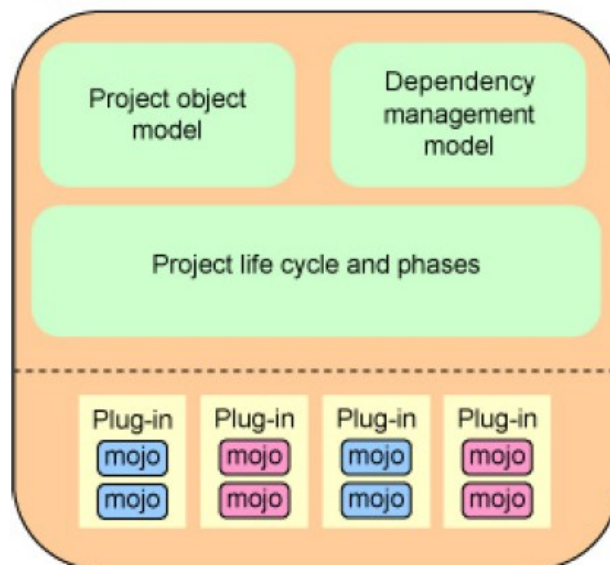


Figura 1: Modelo de objetos e de operação do Maven

Os componentes principais da figura anterior são:

- Project object model (POM): O POM é o componente principal para o Maven. Parte deste modelo faz parte do mecanismo interno do Maven. Outra parte é informada por um arquivo pom.xml editado por você.
- Dependency management model: A gestão de dependência é uma parte do processo de construção de software que muitas vezes é ignorada em projetos, mas que é fundamental. O Maven é reconhecidamente uma das melhores

ferramentas para a construção de projetos complexos, especialmente pela sua robustez na gestão de dependências.

- Ciclo de vida de construção e fases: Associado ao POM, existe a noção de ciclo de vida de construção e fases. É através deste modelo que o Maven cria uma “ponte” entre os modelos conceituais e os modelos físicos. Quando se usam plug-ins, é por meio deste componente que tais plug-ins são gerenciados e passam a seguir um ciclo de vida bem definido.
- Plug-ins: estendem as funcionalidades do Maven.

Visão Física

Visão Física

- POM, como já foi dito, pode ser alterado através do arquivo pom.xml
- Artefatos: cada unidade do Maven que forma um componente
- Repositórios: onde o Maven procura por artefatos
 - Pode ser local ou remoto
- Mecanismo de resolução de dependências
 - Permite identificar dependências entre artefatos, inclusive dependências transitivas
- Plug-ins: Estão associados às fases do ciclo de vida

A figura a seguir mostra os componentes físicos do Maven com os quais você pode interagir:

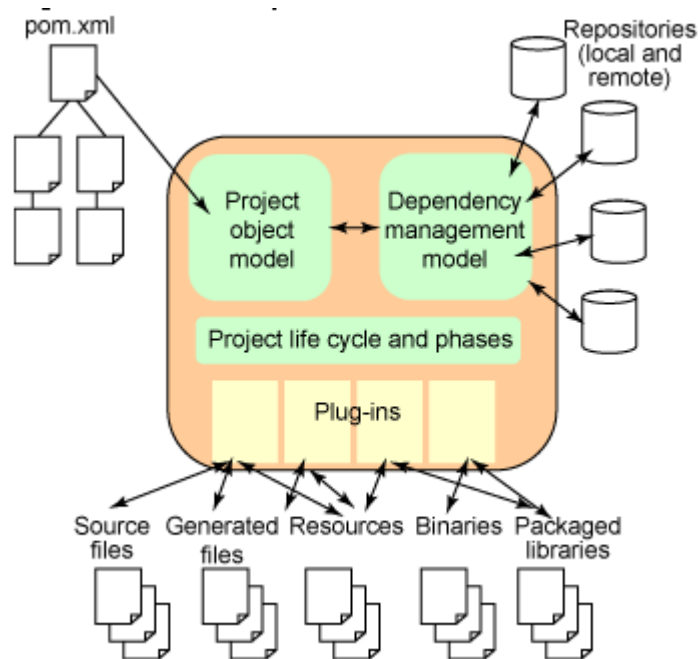


Figura 2: Arquivos, diretórios e repositórios utilizados pelo Maven

Na figura anterior, o **POM (Project Object Model)**, como já foi dito, é formado por dados internos ao Maven e por um ou mais arquivos **pom.xml**. Os arquivos pom.xml formam uma espécie de hierarquia, com cada um herdando os atributos do seu pai. O próprio Maven provê um POM que está no topo da hierarquia e portanto define os valores *default* para todos os projetos.

As dependências são especificadas como parte do arquivo pom.xml. O Maven identifica as dependências do projeto de acordo com seu modelo de gerenciamento de dependências. O Maven procura por componentes dependentes (chamados **artefatos** na terminologia do Maven) no **repositório local** e em **repositórios globais**. Artefatos encontrados em repositórios remotos são baixados para o repositório local para um acesso posterior mais eficiente. O **mecanismo de resolução de dependências** do Maven pode identificar **dependências transitivas** (A depende de B que depende de C, logo, A depende de C).

Plug-ins são configurados e descritos no arquivo pom.xml. O plug-in propriamente dito é gerenciado como um artefato pelo sistema de gerenciamento de dependências e baixados quando eles são necessários para realizar alguma atividade de construção.

Cada plug-in pode ser associado a várias fases do ciclo de vida. O Maven mantém uma máquina de estados que transita pelo ciclo de vida e dispara os plug-ins quando necessário.

Objetivos e Funcionalidades do Maven

Objetivos e Funcionalidades do Maven

- Algumas dúvidas comuns no desenvolvimento de software:
 - Qual deve ser a estrutura de diretórios para organizar o projeto?
 - Como devem ser estruturados os diretórios de forma a organizar o código-fonte, código de teste, bibliotecas, configuração, documentação e relatórios do projeto?
 - De onde as bibliotecas Java (JARS) deverão ser baixados?
- O Maven...
 - Define como o projeto é tipicamente organizado e construído.
 - Utiliza convenção sobre configuração
 - É flexível e extensível

No desenvolvimento de projetos de software, o desenvolvedor encontrará algumas perguntas como:

- Qual deve ser a estrutura de diretórios para organizar o projeto?
- Como devem ser estruturados os diretórios de forma a organizar o código-fonte, código de teste, bibliotecas, configuração, documentação e relatórios do projeto?
- De onde as bibliotecas Java (JARS) deverão ser baixados?
- Quais versões das bibliotecas (JARS) deverão ser utilizadas?
- Qual a melhor forma de resolver conflitos entre bibliotecas?
- Como manter o projeto com as versões mais recentes das bibliotecas?
- Como as bibliotecas e recursos de compilação, execução e teste devem ser separados?
- Existe alguma forma de executar todos os testes durante o processo de construção e obter um percentual de cobertura dos testes?
- É possível executar testes para mensurar a qualidade do código ou medir desempenho durante o processo de construção?
- É possível executar testes de integração durante o processo de construção?
- Como separar os scripts para diferentes atividades de construção?

O Maven busca resolver estas questões fornecendo um modelo de construção que pode ser reutilizado por todos os projetos de software. O Maven abstrai a estrutura do projeto e seu conteúdo, seguindo os princípios da "convenção sobre a configuração", "execução declarativa do processo de ciclo de vida do desenvolvimento", "reúso da lógica de construção entre projetos" e "organização lógica das dependências dos projetos".

Em resumo, o Maven:

- Define como o projeto é tipicamente construído.
- Utiliza convenções para facilitar a configuração do projeto e assim, sua construção.

Introdução ao Apache Maven

- Ajuda os usuários a compreender e organizar melhor a complexa estrutura dos projetos e suas variações.
- Prescreve e força a utilização de um sistema de gerenciamento de dependências comprovadamente eficaz, permitindo que times de projeto em locais diferentes compartilhem bibliotecas.
- É flexível para usuários avançados, permitindo que definições globais sejam redefinidas e adaptadas de forma declarativa (alterando-se a configuração, alterando metadados ou através da criação de plug-ins).
- É extensível.
- Está em constante evolução, incorporando novas práticas e funcionalidades identificadas como comuns em comunidades de usuários.

Ant versus Maven

Ant versus Maven

- O Ant é outra ferramenta para configuração comumente utilizada

Critério	Ant	Maven
Instalação	Muito fácil	Muito fácil
Tempo para iniciar um novo projeto	5 minutes	15 minutos
Tempo para adicionar uma nova funcionalidade	10 minutos para adicionar um novo alvo	2 minutos para adicionar uma nova meta (goal).
Tempo de aprendizado para um novo desenvolvedor	30 minutos. Muito fácil para se entender e suporte por parte de ferramentas que é muito bom.	2 horas. Pode ser confuso no início.
Layout padrão?	Não (pode ser bom, você pode alterá-lo como quiser)	Sim (pode ser bom, pois todos os seus projetos seguem o mesmo layout)
Suporte a múltiplos projetos?	Sim, mas você tem que criar toda a lógica para tratar múltiplos projetos.	Sim, nativo com o Maven Reactor.
Geração de documentação	Nativamente, não, mas existem plug-ins.	Sim
Integração com IDEs?	Sim e muito boa	Sim, mas básica
Gerência de dependências?	Sim, mas requer plug-in (Ivy)	Sim, nativo

O Apache Ant é outra ferramenta comumente utilizada para a construção de projetos. Mas qual a diferença entre o Ant e o Maven?

Primeiramente, é importante esclarecer que cada ferramenta dá ênfase a um aspecto do problema de construção de aplicação. O Ant é uma ferramenta multiplataforma para construir projetos escritos em Java. Maven, por sua vez, descreve o projeto em um nível mais alto toma "emprestadas" várias tarefas do Ant. A tabela a seguir mostra algumas diferenças entre as ferramentas em mais alto nível:

Critério	Ant	Maven
Instalação	Muito fácil	Muito fácil
Tempo para iniciar um novo projeto	5 minutes	15 minutos
Tempo para adicionar uma nova funcionalidade	10 minutos para adicionar um novo alvo	2 minutos para adicionar uma nova meta (goal).
Tempo de aprendizado para um novo desenvolvedor	30 minutos. Muito fácil para se entender e suporte por parte de ferramentas que é muito bom.	2 horas. Pode ser confuso no início.
Layout padrão?	Não (pode ser bom, você pode alterá-lo como quiser)	Sim (pode ser bom, pois todos os seus projetos seguem o mesmo layout)
Suporte a múltiplos projetos?	Sim, mas você tem que criar toda a lógica para tratar múltiplos projetos.	Sim, nativo com o Maven Reactor.
Geração de documentação	Nativamente, não, mas existem plug-ins.	Sim
Integração com IDEs?	Sim e muito boa	Sim, mas básica
Gerência de dependências?	Sim, mas requer plug-in (Ivy)	Sim, nativo

Modelo de Objetos de Projetos (POM)

Modelo de Objetos de Projeto (POM)

- Define as informações necessárias para que o Maven possa executar um conjunto de metas e o software seja construído.
- O POM é formado pelo POM interno do Maven e pelo conteúdo do arquivo pom.xml.
- O pom.xml é formado por vários elementos que:
 - Permitem configurar informações sobre o projeto (equipe, controle de versão, licença, etc).
 - Permitem configurar as dependências
 - Permitem configurar os plug-ins do Maven
 - Definir e configurar relatórios a serem gerados

O Modelo de Objetos de Projeto (POM, em inglês) define as informações necessárias para que o Maven possa executar um conjunto de metas e que a construção do software possa ser realizada. A representação do POM é feita através de um arquivo XML chamado pom.xml. No conceito do Maven, o projeto é um conceito que vai além de um conjunto de arquivos. O projeto contém arquivos de configuração, uma lista de desenvolvedores que atuam em papéis, um sistema de controle de ocorrências (issues), informações sobre a empresa e licenças, a URL onde o projeto reside, as dependências e todo o código que dá vida ao projeto. De fato, para o Maven, um projeto não precisa ter nada mais do que um simples arquivo pom.xml.

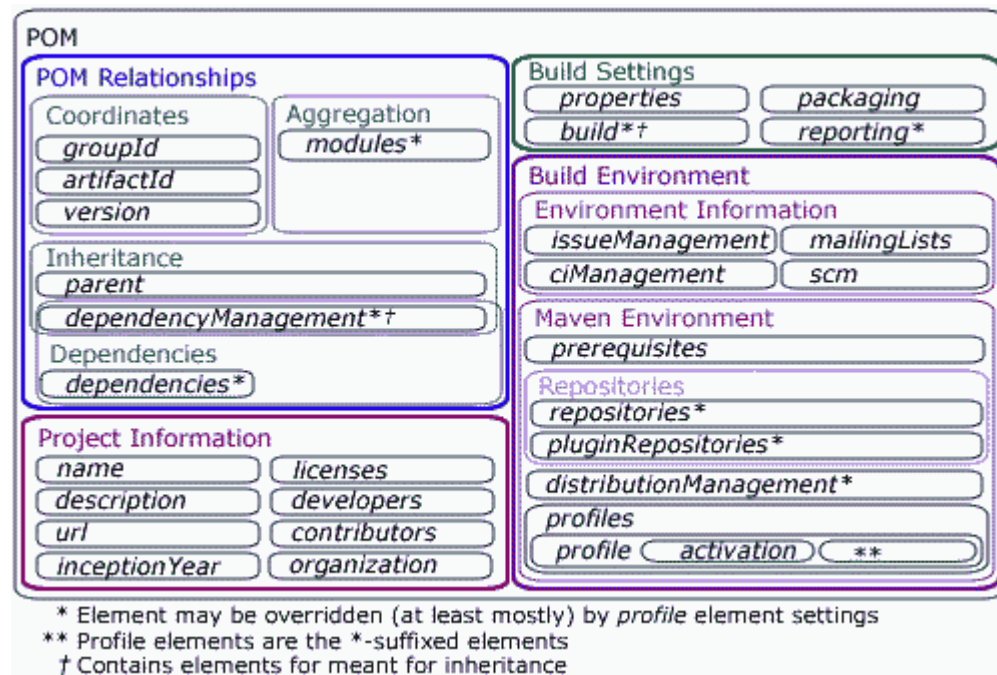


Figura 3: Estrutura de elementos componentes do POM

A estrutura do pom.xml

A estrutura do POM é grande e complexa, como pode ser visto na figura anterior. Ao longo do curso, veremos várias destas estruturas.

Um esqueleto do arquivo pom.xml pode ser visto a seguir:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <!-- POM Relationships -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagement>
  <dependencies>...</dependencies>
  <modules>...</modules>

  <!-- Project Information -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <developers>...</developers>
  <contributors>...</contributors>
  <organization>...</organization>

  <!-- Build Settings -->
  <packaging>...</packaging>
  <properties>...</properties>
  <build>...</build>
  <reporting>...</reporting>

  <!-- Build Environment -->
  <!-- Environment Information -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <!-- Maven Environment -->
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
```

```
<distributionManagement>...</distributionManagement>  
<profiles>...</profiles>  
</project>
```


Repositório Maven

Repositórios Maven

- Podem ser locais ou remotos
- O repositório local sempre é pesquisado primeiro a procura de artefatos.
 - Geralmente, se encontra em C:\Documents and Settings\USUARIO\.m2 ou ~/.m2.
- Os repositórios remotos são pesquisados quando não se encontra um artefato no repositório local.
 - Se o artefato não for encontrado em lugar algum, um erro é gerado.
 -

Como foi dito anteriormente, o Maven utiliza dois tipos de repositórios para armazenar os artefatos: o local e o remoto.

Repositório local

O repositório local armazena os artefatos localmente no seu computador. É criado um diretório .m2 no seu diretório de usuário (no Unix, ~/.m2 e no Windows, C:\Documents and Settings\USUARIO\.m2). Os artefatos locais são acessados rapidamente e são consultados sempre que o Maven precisa resolver uma dependência ou executar um plug-in. Quando um artefato necessário não está presente no repositório local, é feita uma consulta a um ou mais repositórios remotos.

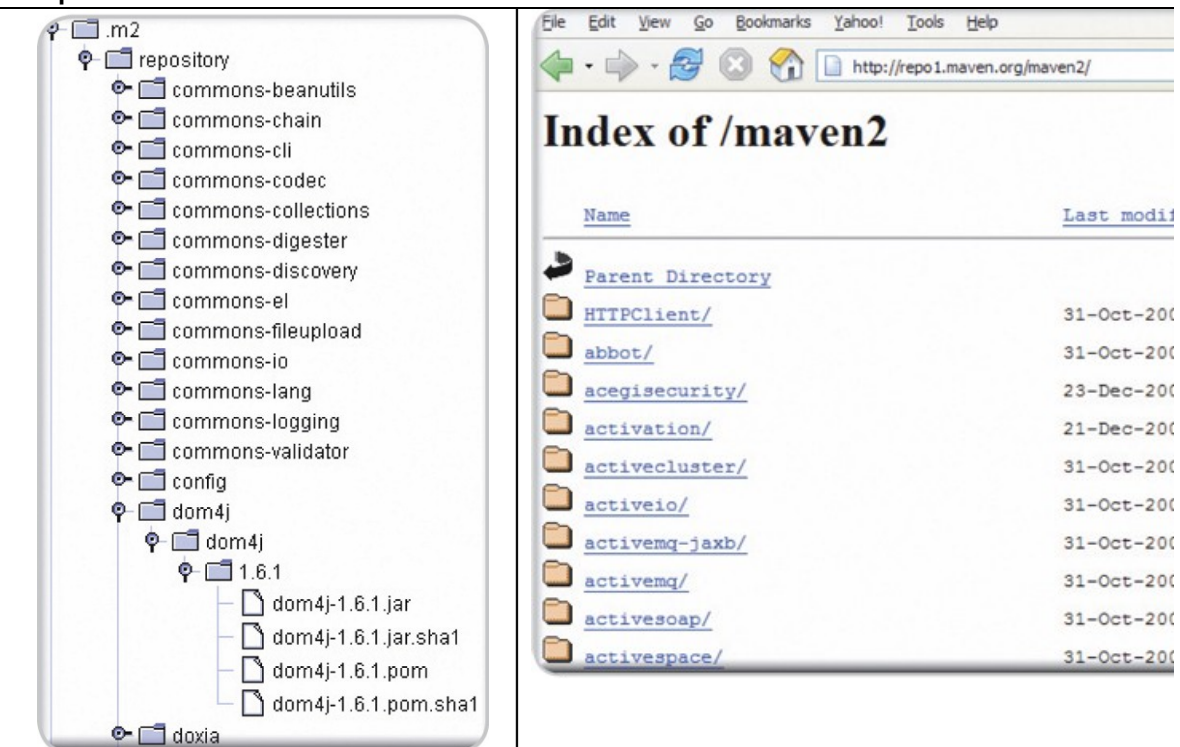


Figura 4: Exemplo de repositório local (esq.) e remoto (dir.)

Repositório remoto

O repositório remoto pode estar em um dos vários repositórios Maven espalhados pela internet ou mesmo na sua intranet. O principal repositório remoto atualmente é o site <http://repo1.maven.org/maven2/>. Vários projetos de código livre publicam suas bibliotecas nestes repositórios remotos. Assim, se seu projeto utilizará algum framework de código livre comumente utilizado (como Hibernate ou Spring), com muita chance as últimas versões das bibliotecas estarão disponíveis para download por parte do Maven.

Importante: A lista de repositórios remotos é configurada no arquivo `$MAVEN/conf/settings.xml` (global) ou `$USERHOME/.m2/settings.xml` (usuário), onde `$USERHOME` é o diretório pessoal do usuário.

Localizando artefatos

Baseando-se na informação de dependência descrita no POM, o Maven tentará resolver as dependências na seguinte ordem:

1. A dependência é procurada no repositório local.
2. A dependência é procurada em todos os repositórios remotos configurados.
3. Falhando 1 e 2, um erro é reportado.

O elemento `<repositories>` no arquivo `settings.xml` pode ser utilizado para configurar repositórios remotos.

Um ponto positivo do Maven é que com a utilização de repositórios locais, é garantido que apenas uma cópia do artefato é mantida, independentemente se um ou vários projetos fazem uso dele. Isto evita que ocorram conflitos especialmente quando se utilizam múltiplos projetos.

Repositórios e Coordenadas

Repositórios e Coordenadas

- Artefatos geralmente são empacotados na forma de bibliotecas Java (JAR).
- Para indexar um repositório, o Maven utiliza o conceito de coordenadas:
 - Identificam de forma única um artefato.
 - São formadas por três partes: Group Id, Artefact Id e versão.

```
<dependencies>
<dependency>
<groupId>br.com.eteg</groupId>
<artifactId>CalculadoraCientifica</artifactId>
<version>1.0-SNAPSHOT</version>
</dependency>
</dependencies>
```

Um artefato é geralmente empacotado e distribuído na forma de um arquivo JAR contendo uma biblioteca binária ou um executável. Na prática, um artefato também pode ser um arquivo WAR ou EAR, arquivos típicos da plataforma Java para empacotamento.

Para indexar a coleção de artefatos armazenados nos repositórios, o Maven utiliza a estrutura de diretórios do sistema de arquivos do sistema operacional. A indexação do repositório se baseia no fato de que um artefato pode ser identificado de forma única por meio de suas coordenadas.

Coordenadas

Uma coordenada para o Maven é um conjunto de valores que identificam de forma única um artefato. Uma coordenada é composta de três partes de informação, separadas por hífen:

- Group ID: A entidade ou organização responsável pela produção do artefato. Por exemplo, br.com.eteg pode ser um group id.
- Artifact ID: Nome do artefato. Por exemplo, para um projeto com uma classe chamada CalculadoraCientifica pode usar também este nome.
- Versão: Um número de versão para o artefato. O formato suportado segue a forma mmm.nnn.bbb-qqqqqq-dd, onde mmm é versão maior, nnn a menor versão e bbb é o nível da correção de bugs. Opcionalmente, qqqqq (qualificador) ou dd (número da construção - build number) também podem ser adicionados ao número da versão.

As coordenadas do Maven são usadas tanto na configuração do repositório quanto no POM. Por exemplo, para especificar uma dependência a um projeto chamado CalculadoraCientifica, na versão 1.0-SNAPSHOT, um arquivo pom.xml inclui uma declaração como esta:

```
...
<dependencies>
```

```
<dependency>
<groupId>br.com.eteg</groupId>
<artifactId>CalculadoraCientifica</artifactId>
<version>1.0-SNAPSHOT</version>
</dependency>
</dependencies>

...
```

Importante: o qualificador SNAPSHOT é especial e indica para o Maven que o projeto está em desenvolvimento e portanto é para recuperar a versão mais recente disponível.

Para especificar a dependência de um projeto conhecido como o Junit, versão 3.8.2, podemos usar a seguinte coordenada:

```
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.2</version>
</dependency>
</dependencies>
```

Como o repositório do Maven nada mais é do que uma organização em árvore, você pode ver os artefatos armazenados em disco. A figura a seguir mostra o artefato Junit JUnit 3.8.2:

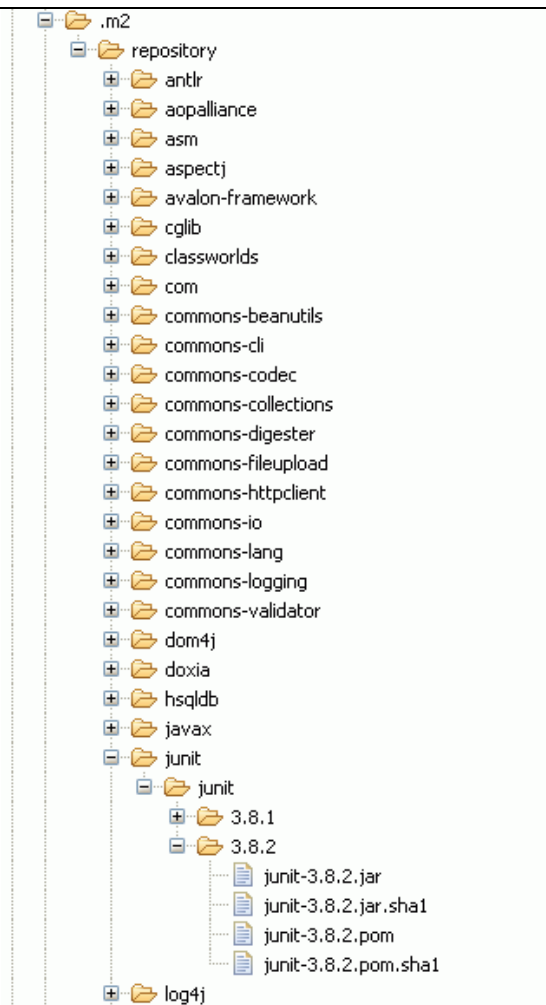


Figura 5: Organização do repositório segundo coordenadas

Introdução aos Plug-ins

Introdução aos Plug-ins

- A maior parte das tarefas executadas pelo Maven são feitas por meio de plug-ins.
- Um plug-in é formado por um ou mais mojos, que são as menores unidades de execução do Maven.
- Alguns plug-ins estão associados a fases do ciclo de vida e são executados automaticamente.
 - Por exemplo, o plugin maven-compile é executado para a fase compile
- Uma meta (goal) são tarefas no Maven. Ao executar um mojo, estamos executando uma meta.
 - Exemplo: mvn jar:jar

A maior parte das tarefas realizadas pelo Maven são feitas por meio de plug-ins. O Maven atua como um maestro na coordenação das ações dos plug-ins.

Instalação de plug-ins

Maven traz consigo vários plug-ins pré-configurados, que baixam quaisquer dependências pela internet, sem necessidade de configurações. A maior parte dos projetos de software usará esse conjunto pré-configurado e raramente necessitará de plug-ins adicionais.

Dica: Consulte o site <http://maven.apache.org/plugins/> para obter uma lista de plug-ins disponíveis.

Mojos em plug-ins

Plug-ins são módulos de software que se enquadram no modelo de extensão do Maven. Os plug-ins podem ser criados em Java, Ant ou Beanshell. Cada tarefa dentro do plug-in é chamada mojo. Em muitos casos, um plug-in pode ser visto como um conjunto de mojos.

Um mojo é executado pelo mecanismo do Maven durante o ciclo de vida da construção do projeto. Cada mojo está associado a uma ou mais fases.

Para listar os mojos de um plug-in, basta digitar, mvn -P plugin, como por exemplo:

```
mvn -P jar
```

Fases do ciclo de vida (tópico opcional)

O ciclo de vida da construção do projeto está dividido em fases, conforme a figura a seguir:

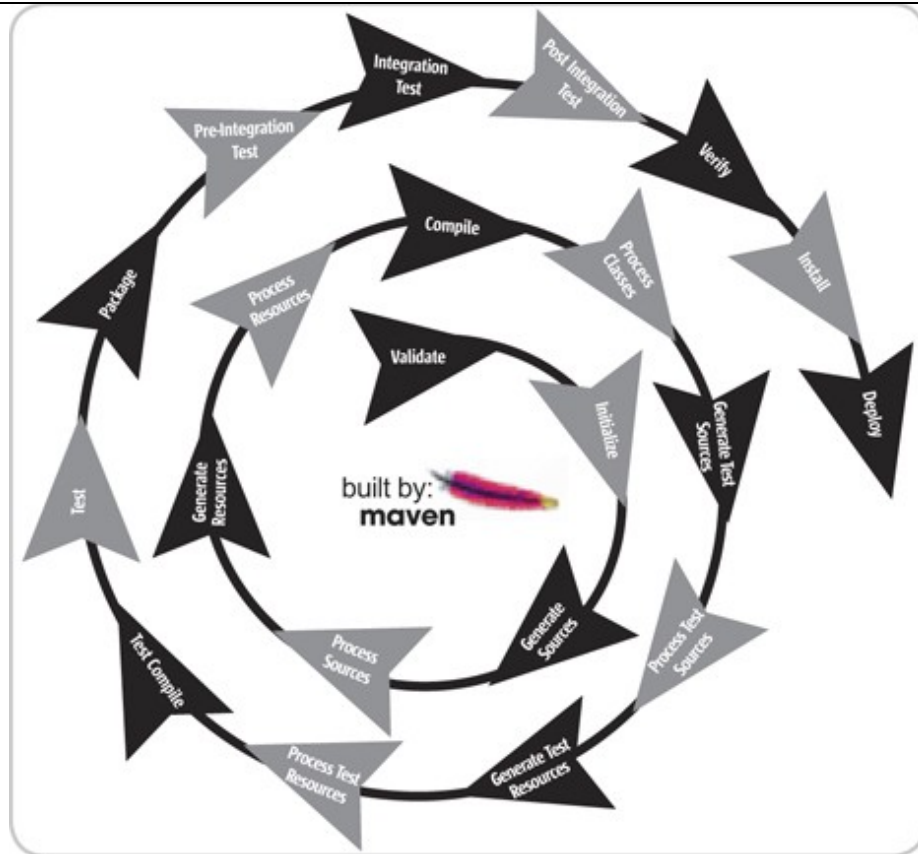


Figura 6: Fases do ciclo de vida do Maven

Para que uma fase seja executada, todas as outras anteriores devem também ser executadas. Por exemplo, para executar a fase *compile*, as fases *validate*, *initialize*, *generate sources*, *process sources*, *generate resources* e *process resources* necessariamente serão executadas.

Para disparar uma fase do ciclo de vida de construção, basta executar o comando `mvn nome_da_fase`. Por exemplo:

```
mvn compile
```

Dica: A diferença de um comando Maven que está associado a uma fase e a um plug-in é que no caso de fases, o comando não tem dois-pontos (:) e corresponde exatamente a uma fase do ciclo.

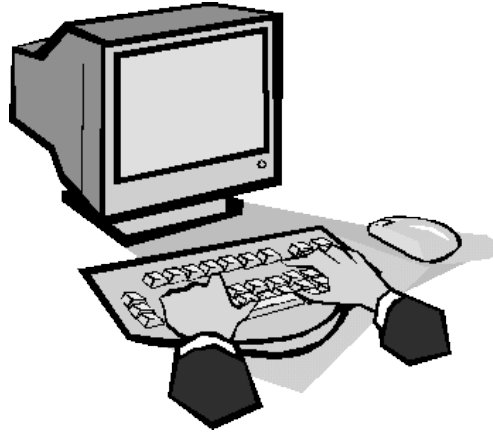
Goals

Goals (ou metas em português) goal são tarefas na terminologia do Maven. Geralmente, as metas são especificadas como argumentos para o Maven na linha de comando, mas algumas metas podem ser chamadas por outras metas. Metas são disponibilizadas por plug-ins instalados ou na forma de *script* no arquivo `maven.xml`. Para visualizar todas as metas disponíveis, utilize o comando

```
maven -g
```

Revisão e Laboratório

Revisão e Laboratório



- 1- Cite algumas das funcionalidades do Maven que podem ser utilizadas para o desenvolvimento de softwares.
- 2- O que é o Project Object Model (POM) do Maven?
- 3- Qual é a finalidade dos plug-ins do Maven?
- 4- Quais são os tipos de repositórios de artefatos do Maven? Qual destes tipos sempre é verificado primeiro a procura de artefatos? Por quê?
- 5- O que é uma coordenada para o Maven? Como ela é representada no sistema de arquivos?

Capítulo 2: Instalação, Utilização e Principais Recursos

Instalando o Maven 2

Instalando o Maven 2

- Pré-requisitos:
 - JDK 1.4 ou posterior.
 - Memória e disco: não há requisito mínimo; para repositório local, aproximadamente 100Mb de espaço livre.
 - Sistema operacional: os principais suportados pelo Java.
- Instalação
 - Baixe os arquivos do site <http://maven.apache.org>
 - Descompacte-os
 - Adicione o diretório bin do Maven ao PATH do sistema.

Pré-requisitos

JDK	1.4 ou posterior (para a execução do Maven, mas você pode compilar código para as versões 1.3 e anteriores)
Memória	Não há um mínimo, depende do processo de construção. Geralmente, 64Mb
Disco	Não há mínimo. Aproximadamente 100MB para repositório local típico.
Sistema operacional	O Maven foi testado no Windows XP, Fedora Core e Mac OS X. Teoricamente, funciona em qualquer sistema onde uma máquina virtual compatível funcione.

Instalação

O primeiro passo é fazer o download no site <http://maven.apache.org>.

Em seguida, descompactar os arquivos em algum diretório a sua escolha, que iremos referenciar neste curso como \$MAVEN.

Adicionar o diretório \$MAVEN/bin à variável PATH.

Para verificar a instalação, execute no prompt/shell de comando o seguinte comando:
`mvn -help`

Criando o Primeiro Projeto com Archetype

Criando o Primeiro Projeto com Archetype

- O primeiro projeto é uma biblioteca para calculadora financeira:
 - Funções `rate()`, `nper()`, `pmt()` e `pv()`
- Para criar o projeto:
- As bibliotecas necessárias para o plug-in que não estiverem no repositório local serão baixadas automaticamente
- Para criar um projeto para o IDE Eclipse:

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=CalculadoraFinanceira
```

```
mvn eclipse:eclipse
```

Nesta e nas próximas seções, vamos criar um pequeno projeto para ilustrar o funcionamento do Maven. O projeto é uma calculadora financeira, que possui as seguintes operações: `rate()`, `nper()`, `pmt()` e `pv()`.

Para criar um projeto de partida, execute o comando:

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=CalculadoraFinanceira
```

Nota: Sempre que um plug-in necessitar de bibliotecas que não estão no repositório local, o Maven tentará baixá-las dos repositórios remotos, por isto, certifique-se de ter uma conexão com a internet.

Este comando (que deve estar na mesma linha) cria o arquétipo (archetype) do projeto, gerando um arquivo `pom.xml` como este:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.eteg</groupId>
  <artifactId>CalculadoraFinanceira</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>CalculadoraFinanceira</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
```

```
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

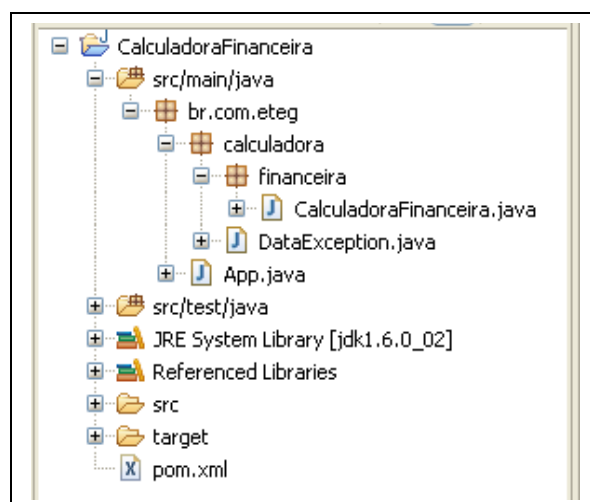
Não é necessário informar a versão para o plugin Archetype, pois o default é 1.0-SNAPSHOT. Em seguida, crie o projeto para o Eclipse:

```
mvn eclipse:eclipse
```

O código (com partes suprimidas) para a classe `CalculadoraFinanceira` é:

```
public class CalculadoraFinanceira {
    public static double fv(double rate, int nPer, double pmt, double
pv,
        int due) throws DataException {...}
    public static double pmt(double rate, int nper, double pv, double fv,
        int due) throws DataException {...}
    public static int nPer(double rate, double pmt, double pv, double fv,
        int due) throws DataException {...}
    public static double pv(double rate, int nPer, double pmt, double fv,
        int due) throws DataException {...}
    public static double rate(int nPer, double pmt, double pv, double fv,
        int due, double guess) throws DataException {...}
}
```

Após adicionar o código-fonte e outra classe (`DataException`) para tratamento de exceção, o projeto ficou conforme a figura a seguir:



A classe `App` foi alterada para o seguinte código:

```
package br.com.eteg;
import java.text.DecimalFormat;
```

```
import br.com.eteg.calculadora.financeira.CalculadoraFinanceira;
import br.com.eteg.calculadora.DataException;
public class App {
    public static void main(String[] args) {
        try {
            System.out.println(
                new DecimalFormat("R$ ###,##0.00").format(
                    CalculadoraFinanceira.pmt(0.1, 10, -10000, 0, 0));
        } catch (DataException e) {
            e.printStackTrace();
        }
    }
}
```

O resultado impresso ao executar é:

R\$ 1.627,45.

O Plug-in Archetype

O Plug-in Archetype

- Permite criar a primeira versão do POM para o projeto
- Existem vários modelos de projetos disponíveis

- Para visualizar a lista, use o comando:

```
mvn archetype:generate
```

- Exemplo da criação de uma aplicação web:

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=exemploWeb \
-DarchetypeArtifactId=maven-archetype-webapp
```

O plug-in archetype, como vimos, serve para criar a primeira versão do POM para o projeto. Cada projeto que é usado como modelo é chamado de *archetype*, daí o nome do plug-in. Existem vários modelos de projeto que podem ser utilizados. Uma forma de ver todos os que estão disponíveis é executar a meta *generate*:

```
mvn archetype:generate
```

Será exibida uma lista de todos os modelos disponíveis e será solicitado que você informe o número do modelo a ser utilizado.

Uma forma mais direta de se utilizar o plug-in é especificar qual *archetype* deve ser usado. Por exemplo:

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=exemploWeb -DarchetypeArtifactId=maven-archetype-webapp
```

No exemplo anterior, será criado um projeto para uma aplicação web em Java.

Alterando o POM

Alterando o POM

- Alterar a forma de empacotamento (JAR, EAR, WAR, POM)
- Alterar o nome e a URL do projeto
- Adicionar dependências
- Configurar plugins

```
<packaging>jar</packaging>
```

```
<name>CalculadoraFinanceira</name>
<url>http://maven.apache.org</url>
```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

O Maven é configurado para construir seu projeto através do arquivo pom.xml. Conforme vimos anteriormente, o plug-in Archetype permite gerar um arquivo pom.xml padrão, onde informamos as coordenadas. Vamos analisar outras seções do arquivo pom.xml:

```
<packaging>jar</packaging>
```

Este elemento permite configurar como será empacotado o projeto. No nosso caso, por ser uma biblioteca, vamos utilizar o formato JAR.

```
<name>CalculadoraFinanceira</name>
<url>http://maven.apache.org</url>
```

Estes elementos permitem configurar o nome e a URL para o projeto.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

O elemento dependencies permite configurar dependências para o projeto e especificar qual é o escopo. No caso, há uma dependência da biblioteca JUnit, para testes unitários. Note que o escopo é apenas test, logo, a biblioteca não é necessária em outras etapas do ciclo de vida de construção. Veremos mais sobre dependências nos próximos capítulos.

Muitas outras configurações podem ser feitas no pom.xml. Por exemplo, podemos configurar plug-ins que são executados em algum momento do ciclo de vida:

```
<build>
<plugins>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.5</source>
<target>1.5</target>
</configuration>
</plugin>
</plugins>
</build>
```

No exemplo, estamos indicando ao plugin compiler que ele deverá compilar as classes de forma que elas sejam compatíveis com a versão 1.5 do Java (o default é 1.4, que não suporta funcionalidades mais recentes, como generics).

Compilando e Testando o Projeto

Compilando e Testando o Projeto

- Para compilar o projeto, execute a fase compile:
- Antes de testar o projeto, é necessário criar testes unitários.

```
mvn compile
```

- Para testar, execute a fase test:

```
mvn test
```

Para compilar um projeto, execute a fase compile do Maven:

```
C:\curso\CalculadoraFinanceira>mvn compile
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building CalculadoraFinanceira
[INFO]    task-segment: [compile]
[INFO] -----
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Compiling 3 source files to
C:\curso\CalculadoraFinanceira\target\classes
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 3 seconds
[INFO] Finished at: Thu Oct 23 22:22:10 BRT 2008
[INFO] Final Memory: 3M/10M
[INFO] -----
```

Antes de testar, é necessário criar um arquivo de teste. No diretório src/test/java do projeto gerado, foi adicionado um pacote chamado br.com.eteg.calculadora.financeira e nele incluída a classe CalculadoraFinanceiraTest:

```
package br.com.eteg.calculadora.financeira;
```

```
import junit.framework.TestCase;

public class CalculadoraFinanceiraTest extends TestCase {

    public void testPmt() throws DataException {
        double result =
            CalculadoraFinanceira.pmt(0.05, 36, -10000, 0, 0);
        assertEquals(604.34, result, 0.01);
    }

    public void testNPer() throws DataException {
        int result =
            CalculadoraFinanceira.nPer(0.05, 1000, -10000, 0, 1);
        assertEquals(14, result);
    }

    public void testRate() throws DataException {
        double result =
            CalculadoraFinanceira.rate(60, -1000, 0, 100000, 0, 0.01);
        assertEquals(1.61, result * 100, 0.01);
    }

    public void testPv() {
        try {
            CalculadoraFinanceira.pv(10, -60, 0, 0, 1);
            fail("Excecao nao foi levantada");
        } catch (DataException e) {
            e.printStackTrace();
        }
    }

    public void testFv() {
        fail("Not yet implemented");
    }

}
```

Não é objetivo do curso explicar testes unitários, mas entenda que 4 dos 5 testes irão passar. O último (testeFv()) falha porque não foi implementado. Para executar os testes do projeto, basta executar o comando:

```
mvn test
```

O resultado é o seguinte:

```
C:\curso\CalculadoraFinanceira>mvn test
[INFO] Scanning for projects...
```

```
[INFO] -----
[INFO] Building CalculadoraFinanceira
[INFO]    task-segment: [test]
[INFO] -----
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory:
C:\eclipse\workspace\CalculadoraFinanceira\target\surefire-reports

-----

T E S T S

-----

Running br.com.eteg.calculadora.financeira.CalculadoraFinanceiraTest
Tests run: 5, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.11 sec
<<< FAILURE!

Running br.com.eteg.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015
sec

Results :

Failed tests:
    testFv(br.com.eteg.calculadora.financeira.CalculadoraFinanceiraTest)

Tests run: 6, Failures: 1, Errors: 0, Skipped: 0

[INFO] -----
[ERROR] BUILD FAILURE
[INFO] -----
[INFO] There are test failures.

Please refer to C:\curso\CalculadoraFinanceira\target\surefire-reports
for the individual test results.

[INFO] -----
[INFO] For more information, run Maven with the -e switch
```

Introdução ao Apache Maven

```
[INFO] -----  
[INFO] Total time: 3 seconds  
[INFO] Finished at: Thu Oct 23 23:06:32 BRT 2008  
[INFO] Final Memory: 4M/8M  
[INFO] -----
```

Os resultados dos testes estão disponíveis no diretório surefire-reports.

Empacotando e Disponibilizando Localmente

Empacotando e Disponibilizando Localmente

- Para empacotar o projeto (no exemplo, uma biblioteca JAR):

```
mvn package
```

```
mvn jar:jar
```

- Para instalar a biblioteca no repositório local, execute a fase install:

```
mvn install
```

Para empacotar o projeto, isto é, gerar, neste caso, uma biblioteca Java no formato JAR, basta executar o plug-in jar ou então a fase package:

```
C:\curso\CalculadoraFinanceira>mvn jar:jar
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'jar'.
[INFO] -----
[INFO] Building CalculadoraFinanceira
[INFO]    task-segment: [jar:jar]
[INFO] -----
[INFO] [jar:jar]
[INFO] Building jar: C:\
CalculadoraFinanceira\target\CalculadoraFinanceira-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 3 seconds
[INFO] Finished at: Thu Oct 23 23:10:53 BRT 2008
[INFO] Final Memory: 4M/8M
[INFO] -----
```

Note que o arquivo gerado tem o formato do nome conforme as coordenadas do Maven e inclui também a versão.

Para instalar a biblioteca no repositório local do Maven, basta executar:

```
mvn install
```

Note que se ocorrer algum erro nos testes, o Maven reportará o erro. Isto porque o comando `mvn install` também executa os testes e o empacotamento. No caso do projeto exemplo, é necessário corrigir o teste que causa o problema antes de instalar a biblioteca no repositório local.

Utilizando a fase package

Ao invés de executar o comando `mvn jar:jar` (que executa uma meta), também podemos escolher empacotar a aplicação executando uma fase do ciclo de vida específica para este fim: `package`. Esta fase, além de executar a meta `jar:jar`, também executa a fase de compilação, testes e a meta `jar:jar`:

`mvn package`

Gerando Documentação Javadoc

Gerando Documentação Javadoc

- A documentação no formato Javadoc apresenta os detalhes das classes, pacotes, métodos e variáveis que compõem o projeto Java
- Para gerar a documentação:

```
mvn javadoc:javadoc
```

```
mvn javadoc:jar
```

Constructor Detail
CalculadoraFinanceira <pre>public CalculadoraFinanceira()</pre>
Method Detail
fv <pre>public static double fv(double rate, int nPer, double pmt, double pv, int due) throws DataException</pre> <p>Parameters: rate - is the interest rate per period. nPer - is the total number of payment periods in an annuity. pmt - is the payment made each period; it cannot change over the life</p>

Uma das grandes novidades que Java ajudou a popularizar foi a documentação da API das classes através da ferramenta Javadoc. É possível utilizar a ferramenta Javadoc através do próprio Maven. Os arquivos são gerados em \$PROJECT/target/site/apidocs, onde \$PROJECT é o diretório do projeto. Para isto, utilize o comando:

```
mvn javadoc:javadoc
```

Se você quiser, também pode gerar a documentação e imediatamente empacotá-la em um arquivo JAR:

```
mvn javadoc:jar
```

Dica: Nos próximos capítulos veremos como integrar a geração do Javadoc junto com a documentação do site.

Constructor Detail
CalculadoraFinanceira <pre>public CalculadoraFinanceira()</pre>
Method Detail
fv <pre>public static double fv(double rate, int nPer, double pmt, double pv, int due) throws DataException</pre> <p>Parameters: rate - is the interest rate per period. nPer - is the total number of payment periods in an annuity. pmt - is the payment made each period; it cannot change over the life</p>

Figura 7: Documentação Javadoc

Configurando o Maven para uso com o Eclipse

Configurando o Maven para uso com o Eclipse

- O Maven possui algumas metas para integração com o Eclipse:

`eclipse:configure-workspace` `eclipse:eclipse` `eclipse:clean`

- O Eclipse possui um bom plug-in para integração com o Maven, o m2Eclipse (gratuito)
- Entretanto, você pode escolher utilizar os recursos que o Eclipse já provê (através da opção “External Tools”)

Já vimos que o Maven possui um plug-in para gerar o projeto para o Eclipse. Este plug-in tem as seguintes metas (mojos):

<code>eclipse:configure-workspace</code>	Usado para adicionar a variável M2_REPO ao Eclipse. Isto pode ser feito no próprio Eclipse, através do menu Window>Preferentes, opção Java>Build Path>Classpath Variables
<code>eclipse:eclipse</code>	Cria os arquivos de configuração do Eclipse
<code>eclipse:clean</code>	Exclui os arquivos usados pelo Eclipse

Mas, e o contrário? Como utilizar o Maven dentro do Eclipse? Para isto, podemos usar um plug-in para o IDE. Um dos mais famosos é o m2eclipse, disponível em <http://m2eclipse.codehaus.org>.

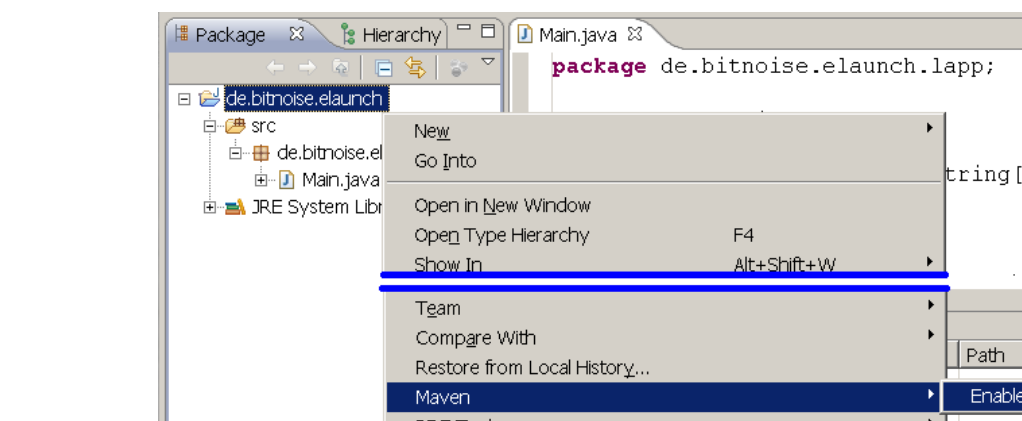


Figura 8: m2Eclipse integrado ao Eclipse

Mas para utilizar o Maven com o Eclipse, não é necessário um plug-in. Muitos preferem configurá-lo como uma ferramenta externa, pois um novo plug-in pode deixar o Eclipse mais sobrecarregado e lento. Consulte a documentação do Eclipse para saber como configurar uma ferramenta externa. A página <http://blog.danielfmartins.com/2007/05/16/debugando-uma-aplicacao-maven-com->

Introdução ao Apache Maven

[o-eclipse/](#) contém um detalhamento sobre o procedimento, inclusive com opção de depuração.

Definindo Dependências para o Projeto

Definindo Dependências para o Projeto

- Altere o arquivo pom.xml
 - Inclua um ou mais elementos dependency
- É possível eliminar a transitividade
- Novos repositórios remotos podem ser incluídos na lista de pesquisa

```
<project>
  <!-- partes omitidas --> ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.12</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
  <!-- partes omitidas --> ...
</project>
```

Projetos Java geralmente fazem uso de muitas bibliotecas existentes, afinal, não há necessidade de ficar sempre inventando a roda. Bibliotecas utilitárias, como a do log4j, do Apache Commons e de frameworks como Spring e Hibernate são publicadas constantemente nos principais repositórios do Maven. Para o Maven, qualquer dependência que esteja registrada para um projeto é tida como um artefato.

Alterando o arquivo pom.xml para incluir dependências

A seção *dependencies* do arquivo pom.xml lista todas as dependências externas que o projeto necessita para que ele seja construído. Para cada dependência, é necessário definir as informações sobre *groupId*, *artifactId*, *version* e *scope*, ou seja, é necessário informar as coordenadas e o momento em que a dependência é utilizada. O valor para o elemento *scope* poderá ser *runtime*, *test*, *provided* e *compile*. Se *scope* não for informado, significa que são todos eles. *Provided* é usado para bibliotecas que estão disponíveis em servidores de aplicação, como por exemplo, as bibliotecas para a API Servlet. Exemplo:

```
<project>
  <!-- partes omitidas -->
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

```
<groupId>log4j</groupId>
<artifactId>log4j</ artifactId >
<version>1.2.12</version >
<scope>compile</scope >
</ dependency>
</dependencies>
<!-- partes omitidas -->
...
</project>
```

No exemplo anterior, foram definidas dependências para duas bibliotecas: log4j e JUnit.

Dica: Uma pesquisa na internet pode retornar um ou mais arquivos pom.xml que contêm exatamente a configuração para a dependência que você precisa.

Eliminando a transitividade

O Maven tem como um dos seus principais pontos fortes a gerência de dependências. As principais características do gerenciador de dependências do Maven incluem a dependência transitiva. Para a dependência transitiva, é possível interromper a transitividade. Isto é útil quando várias bibliotecas dependem de versões específicas de outra biblioteca e você só quer que a mais recente seja utilizada. Para eliminar uma biblioteca com a transitividade habilitada, podemos utilizar o elemento *exclusions* no elemento *dependency*:

```
<dependency>
  <groupId>org.acegisecurity</groupId>
  <artifactId>acegi-security</artifactId>
  <version>1.0.6</version>
  <exclusions>
    <exclusion>
      <groupId>org.springframework</groupId>
      <artifactId>spring-support</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

No exemplo anterior, foi definida uma dependência ao artefato *acegi-security*, mas foi indicado que não é para transitar a dependência para o artefato *spring-support*.

Incluindo repositórios

Alguns projetos podem publicar seus artefatos em repositórios remotos diferentes daqueles pesquisados pelo Maven por padrão. Também é possível que você queira criar um repositório remoto Maven na sua intranet. Nestes casos, é possível especificar repositórios adicionais para o Maven através do arquivo pom.xml:

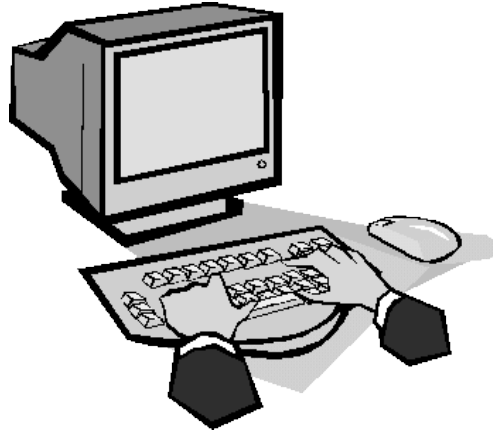
```
<project>
  <!-- partes omitidas -->
```

```
<repositories>
  <repository>
    <id>maven</id>
    <name>maven</name>
    <url>http://repo1.maven.org/maven2/</url>
  </repository>
  <repository>
    <id>intranet</id>
    <name>Intranet Eteg</name>
    <url>http://intranet.eteg.net/java/mvn</url>
  </repository>
</repositories>
<!-- partes omitidas -->
</project>
```

É necessário especificar o identificador e a URL do repositório e opcionalmente o seu nome. Os repositórios são pesquisados na ordem em que aparecem no pom.xml.

Revisão e Laboratório

Revisão e Laboratório



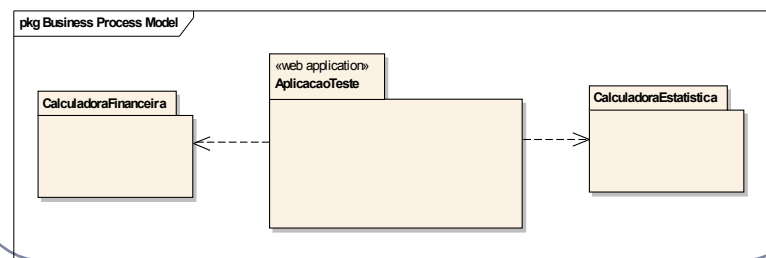
- 1- Qual comando é utilizado para acionar o Maven?
- 2- Utilizando plug-in archetype, escreva a linha de comando para se criar um arcabouço de projeto Maven cujo grupo seja br.com.minhaempresa e o identificador do artefato seja curso.
- 3- Qual comando é utilizado para compilar um projeto gerenciado pelo Maven? No caso, você está utilizando uma fase do ciclo de vida de construção ou um plug-in? Como você sabe?
- 4- Por que ao executar a fase test, o Maven sempre executa a fase compile antes?
- 5- Qual comando é utilizado para disponibilizar um artefato no repositório local?
- 6- Como podem ser definidas dependências em projetos gerenciados pelo Maven?

Capítulo 3: Trabalhando com vários Projetos

Motivação

Motivação

- Em projetos maiores, é necessário fazer uma segmentação em módulos.
- O Maven facilita a gestão de construção de projetos compostos por vários módulos.
- Para o exemplo, usaremos os seguintes módulos:



Até agora, trabalhamos com um projeto só e foi relativamente fácil utilizar o Maven. Entretanto, em muitos casos, será necessário trabalhar com um conjunto de diferentes projetos, a fim de se manter uma boa modularização. Por exemplo, podemos ter bibliotecas separadas das aplicações web ou mesmo duas ou mais aplicações web. Organizar tudo isto, onde temos várias dependências intra-projetos e com artefatos externos, pode ser muito complicado. Felizmente, o Maven foi projetado para simplificar o trabalho.

Havíamos criado um projeto para a *CalculadoraFinanceira*, mas novos requisitos exigem agora que uma *calculadora estatística* seja criada. Esta calculadora irá avaliar a média, mediana e desvio padrão de uma amostra de dados. Finalmente, também se exigiu que fosse disponibilizada uma aplicação web que permita entrar com os dados para as funções e obter o resultado do cálculo. Desta forma, podemos ver os projetos e as inter-dependências como no diagrama a seguir:

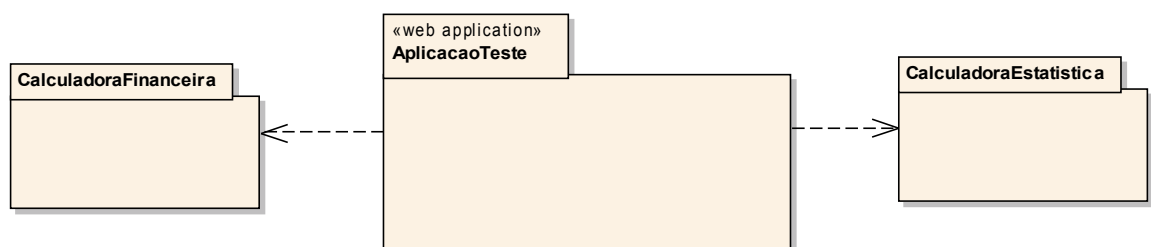


Figura 9: Inter-dependências entre os projetos para o curso

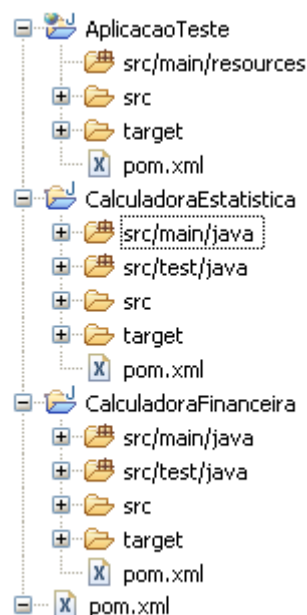
Neste capítulo vamos utilizar o Maven para construir todos esses projetos.

Definindo o Projeto Principal (Master)

Definindo o Projeto Principal (Master)

- O projeto principal é necessário para que o Maven possa gerenciar a construção multi-projetos.
 - É formado simplesmente por um arquivo pom.xml
 - É necessário especificar quais módulos compõem o projeto
 - As dependências podem ser especificadas no projeto principal e “herdadas” pelos módulos

Quando trabalhamos com vários projetos ou módulos de código que formam um projeto de desenvolvimento de software, precisamos criar um projeto principal para que o Maven possa construir toda a aplicação. O projeto principal é formado simplesmente pelo arquivo pom.xml. Geralmente, este arquivo estará no diretório que contém os subdiretórios para os módulos, como ilustrado na figura a seguir:



```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
<groupId>br.com.eteg</groupId>

<artifactId>ProjetoCalculadora</artifactId>

<packaging>pom</packaging>

<version>1.0-SNAPSHOT</version>

<name>Projeto Calculadora para Curso</name>

<description>
    Projeto usado para testes no curso de Maven da Eteg.
</description>

<url>http://maven.apache.org</url>

<modules>
    <module>CalculadoraFinanceira</module>
    <module>CalculadoraEstatistica</module>
    <module>AplicacaoTeste</module>
</modules>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>br.com.eteg</groupId>
            <artifactId>CalculadoraFinanceira</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>br.com.eteg</groupId>
            <artifactId>CalculadoraEstatistica</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>
```

Primeiramente, é necessário definir o identificador do artefato por meio do elemento *artifactID* (no caso, *ProjetoCalculadora*). Para informar que se trata de múltiplos projetos o elemento *packaging* deve ser igual a *pom*.

Organização de dependências

O marcador *modules* especifica que o projeto é formado por 3 módulos: *CalculadoraFinanceira*, *CalculadoraEstatistica* e *AplicacaoTeste*. Os submódulos do projeto podem herdar propriedades do seu arquivo *pom.xml*. Isto significa que nenhum dos três submódulos precisa declarar a dependência ao artefato *junit* (ou qualquer outro que porventura estivesse no projeto principal). Outras configurações, como por exemplo, a de *plug-ins* (vista nos próximos capítulos) também podem ser herdadas.

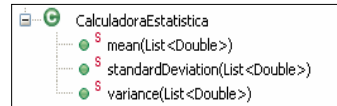
O marcador *dependencyManagement* somente é utilizado para o caso de submódulos. Os submódulos podem especificar uma dependência de qualquer um dos artefatos listados sem necessitar especificar um número de versão. Isto é útil para diminuir o impacto nos projetos caso alguma versão de um artefato seja alterado.

Importante: A propriedade `${project.version}` é avaliada pelo Maven durante a execução.

Criando uma Nova Biblioteca

Criando uma nova Biblioteca

- Para ilustrar, será criada uma nova biblioteca chamada CalculadoraEstatistica.



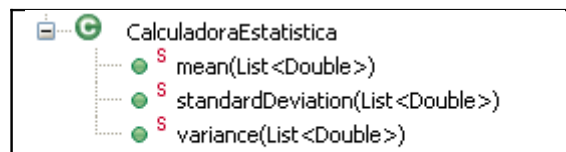
- Para criar o projeto, os mesmos passos utilizados anteriormente serão utilizados.

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=CalculadoraEstatistica
```

```
mvn eclipse:eclipse
```

A biblioteca CalculadoraEstatistica

Esta biblioteca tem a mesma estrutura de projetos da CalculadoraFinanceira. A diferença básica está nos nomes dos pacotes e nos métodos disponíveis nas classes:



Também foi criado um teste unitário para a classe CalculadoraEstatistica.

Criando o projeto

O procedimento para criar o projeto é idêntico ao visto na seção "Criando o Primeiro Projeto com Archetype":

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=CalculadoraEstatistica
```

Se você estiver utilizando o IDE Eclipse, então gere o projeto com o comando:

```
mvn eclipse:eclipse
```

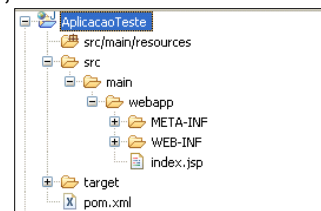
Criando o Projeto para o Site

Criando o Projeto para o Site

- Para o site, podemos escolher um *archetype* mais específico:

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=AplicacaoTeste \
-DarchetypeArtifactId=maven-archetype-webapp
```

- Para que o Maven gere um projeto compatível com o WTP do Eclipse, é necessário alterar o pom.xml gerado.



Para o site, podemos escolher um *archetype* mais específico.

```
mvn archetype:create -DgroupId=br.com.eteg \
-DartifactId=AplicacaoTeste \
-DarchetypeArtifactId=maven-archetype-webapp
```

Desta vez, para integrar o projeto ao Eclipse, precisamos antes configurar a integração com o Web Toolkit Project (WTP) do Eclipse, assim poderemos explorar todas as funcionalidades disponíveis no IDE. O arquivo pom.xml gerado é mostrado a seguir, já com as alterações para suportar múltiplos projetos e com as dependências. A parte em **negrito** é necessária para a integração com o WTP:

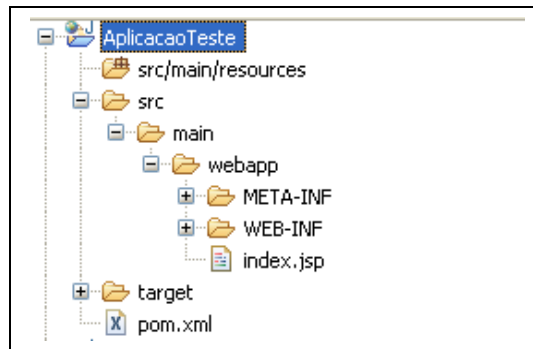
```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <parent>
    <groupId>br.com.eteg</groupId>
    <artifactId>ProjetoCalculadora</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.eteg</groupId>
  <artifactId>AplicacaoTeste</artifactId>
  <packaging>war</packaging>
  <name>AplicacaoTeste Maven Webapp</name>
```

```
<dependencies>
  <dependency>
    <groupId>br.com.eteg</groupId>
    <artifactId>CalculadoraFinanceira</artifactId>
  </dependency>
  <dependency>
    <groupId>br.com.eteg</groupId>
    <artifactId>CalculadoraEstatistica</artifactId>
  </dependency>
  <dependency>
    <groupId>commons-validator</groupId>
    <artifactId>commons-validator</artifactId>
    <version>1.3.1</version>
  </dependency>
</dependencies>
<build>
  <finalName>AplicacaoTeste</finalName>
  <!-- add Eclipse WTP support -->
  <pluginManagement>
    <plugins>
      <plugin>
        <artifactId>maven-eclipse-plugin</artifactId>
        <configuration>
          <wtpversion>2.0</wtpversion>
          <wtpapplicationxml>true</wtpapplicationxml>
          <wtpmanifest>true</wtpmanifest>
          <downloadSources>true</downloadSources>
          <downloadJavadocs>true</downloadJavadocs>
          <projectNameTemplate>
            [artifactId]-[version]
          </projectNameTemplate>
          <manifest>
            ${basedir}/src/main/resources/META-INF/MANIFEST.MF
          </manifest>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>
```

Novamente, crie o projeto Eclipse com o comando:

```
mvn eclipse:eclipse
```

O projeto ficará com esta organização:



O site contará com somente uma página JSP (index.jsp) e fará uso da biblioteca Apache Commons Validator para converter e validar a entrada.

Herança entre POMs

Herança entre POMs

- Os módulos “herdam” as definições do projeto principal
 - Desta forma, vários elementos, entre eles, version e dependências poderão ser omitidos nos módulos.
 - No pom.xml de um módulo, é necessário informar a coordenada do projeto principal.

Descendo um nível, até o diretório da CalculadoraFinanceira, o novo arquivo pom.xml modificado para o módulo fica assim:

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <parent>
    <groupId>br.com.eteg</groupId>
    <artifactId>ProjetoCalculadora</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.eteg</groupId>
  <artifactId>CalculadoraFinanceira</artifactId>
  <packaging>jar</packaging>
  <name>CalculadoraFinanceira</name>
</project>
```

Nota: O projeto CalculadoraEstatistica tem seu POM com a mesma estrutura do projeto CalculadoraFinanceira.

O elemento *parent* especifica a relação do módulo com o POM do projeto principal. Como podemos ver, ao herdar de outro POM, o arquivo ficou bem mais simples. Pudemos eliminar vários elementos, entre eles o elemento *version* e o elemento *dependencies*.

Dica: Você pode especificar o caminho do arquivo pom.xml caso o layout do projeto não seja idêntico ao padrão seguido pelo Maven e apresentado anteriormente. Para isto, utilize o elemento *relativePath* dentro do elemento *parent*. Por exemplo:

```
<relativePath>../parent/pom.xml</relativePath>
```

Empacotando o Projeto

Empacotando o Projeto

- Se as bibliotecas não tiverem sido instaladas, ao tentar empacotar a aplicação, será gerado um erro. Instale as bibliotecas com o comando: `mvn install`
- Para empacotar a aplicação, no diretório onde está o pom.xml do projeto principal, digite: `mvn package`
- É possível executar a aplicação com o plug-in para o Jetty:

```
mvn org.mortbay.jetty:maven-jetty-plugin:run
```

Finalmente, para empacotar o projeto como um todo, podemos ir até o diretório onde está o arquivo pom.xml principal e executar o comando:

```
mvn package
```

Para cada submódulo que compõe o projeto, o Maven irá passar pelo ciclo de vida de construção e ao final, teremos uma aplicação pronta para ser instalada.

Importante: Se tentarmos construir a aplicação web ela irá falhar até que as bibliotecas CalculadoraFinanceira e CalculadoraEstatistica tenham sido instaladas no repositório local. Em cada projeto, execute o comando `mvn install`.

No diretório *target* do projeto AplicacaoTeste você poderá encontrar um arquivo WAR pronto para ser instalado em um servidor.

Executando a Aplicação

Para executar uma aplicação web, é necessário ter um servidor compatível com a especificação Servlet 2.4/JSP 2.0. Felizmente, existe um servidor leve, gratuito e que é integrado ao Maven através de um plug-in. Este servidor é o Jetty.

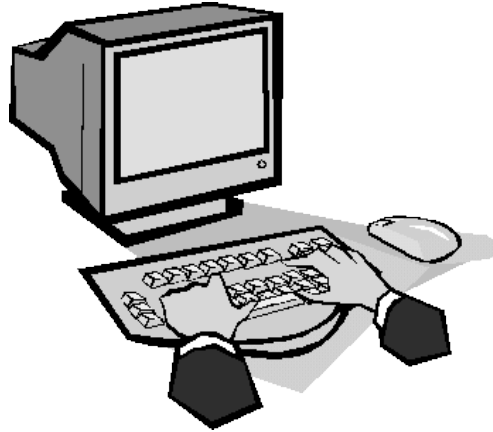
Para executar a aplicação, vá até o diretório AplicacaoTeste e execute o comando:

```
mvn org.mortbay.jetty:maven-jetty-plugin:run
```

Em seguida, basta navegar para <http://localhost:8080/AplicacaoTeste/>.

Revisão e Laboratório

Revisão e Laboratório



- 1- Qual deve ser o valor do elemento *packaging* para o projeto principal quando trabalhamos com múltiplos projetos no Maven?
- 2- Se todos os submódulos de um sistema necessitam de um mesmo artefato, como podemos declarar a dependência a ele?
- 3- Como podemos criar rapidamente um projeto web com o Maven?

Capítulo 4: Configurações de Plug-ins e Relatórios

Introdução

Introdução

- O Maven enfatiza a convenção sobre a configuração
 - Mas nem sempre é possível usar as opções default
- Veremos como configurar alguns plug-ins associados a importantes fases do ciclo de vida
 - Toda a configuração será feita no arquivo pom.xml
 - Veremos como utilizar propriedades
- Também veremos como gerar relatórios e o site para o projeto com o Maven

Neste capítulo veremos como alterar os padrões de configuração para alguns plug-ins dos ciclos de vida de construção do projeto. Toda a alteração será feita através do arquivo pom.xml do projeto principal (apesar de que também podemos fazer nos arquivos pom.xml dos submódulos).

Basicamente, as configurações para os plug-ins são informadas por meio da adição de um ou mais elementos *plugin* com sub-elementos *configuration* no arquivo pom.xml:

```
<project>
  <!--Partes omitidas -->
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.0.2</version>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
          <optimize>true</optimize>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <!--Partes omitidas -->
</project>
```

Introdução ao Apache Maven

Também configuraremos a geração de relatórios do Maven e veremos como criar um site para o projeto.

Configurando a Fase de Compilação

Configurando a Fase de Compilação

- O compilador Java pode ser configurado por meio de vários parâmetros
- Podemos determinar a compatibilidade de código-fonte e do código compilado
 - Por exemplo, *generics* requer código-fonte e compilado compatíveis com a versão 1.5
- Podemos configurar opções que afetam o desempenho
 - Opções debug e optimize

A compilação é uma das mais importantes fases do ciclo de construção. Para que esta fase possa ser utilizada, muitas vezes precisamos configurar alguns parâmetros que serão informados ao compilador.

O compilador Java pode utilizar opções para a versão de compatibilidade tanto para o código-fonte quanto para o código compilado. Isto significa que para o código-fonte, algumas funcionalidades não estarão disponíveis para versões mais antigas. Por exemplo, *generics* foi uma funcionalidade introduzida na versão 5.0 do Java e não está disponível nas versões anteriores. Para o código compilado, é importante certificar que você está utilizando a mesma versão da máquina virtual que executará a aplicação, pois do contrário, aparecerá uma exceção como esta:

```
java.lang.UnsupportedClassVersionError: Bad version number in .class file
```

Para confirmar o plug-in *compile*, altere o POM, informando os parâmetros:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.0.2</version>
  <configuration>
    <source>1.5</source>
    <target>1.5</target>
    <optimize>true</optimize>
    <debug>false</debug>
  </configuration>
</plugin>
```

Introdução ao Apache Maven

A opção *optimize* instrui o compilador a otimizar o código, enquanto a opção *debug* instrui a adicionar informações para depuração. Quando você for gerar uma versão para produção, é uma boa escolha otimizar e remover as informações de depuração.

Importante: Por default o compilador será executado com *source* igual a 1.3 e *target* igual a 1.1.

Configurando o Empacotamento da Aplicação

Configurando o Empacotamento da Aplicação

- Podemos empacotar a aplicação de várias formas em Java:
 - Bibliotecas e aplicações são empacotadas em arquivos com extensão JAR
 - Aplicações web são empacotadas em arquivos com extensão WAR
 - Aplicações que empacotam bibliotecas, várias aplicações web e componentes EJB utilizam arquivos com extensão EAR
 - Todos os arquivos de empacotamento são compatíveis com o formato ZIP

Neste curso vimos o empacotamento de dois tipos de artefatos: bibliotecas JAR e arquivos web WAR. Algumas configurações úteis podem ser feitas para os plug-ins de empacotamento:

Bibliotecas JAR

Existem várias opções que podem ser utilizadas com esta forma de empacotamento.

Criando um JAR executável:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  ...
  <configuration>
    <archive>
      <manifest>
        <addClasspath>true</addClasspath>
        <mainClass>br.com.eteg.Main</mainClass>
      </manifest>
    </archive>
  </configuration>
  ...
</plugin>
```

Você informa a classe que é a principal para o programa (possui um método main()). Com isto, você pode executar o programa digitando:

```
java -jar arquivo.jar
```

Ou então, em um ambiente gráfico (Windows, Gnome, KDE), clicar duas vezes sobre o arquivo.

Aplicações web

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.0.2</version>
  <configuration>
    <warSourceExcludes>**/CVS</warSourceExcludes>
    <warSourceExcludes>**/.svn</warSourceExcludes>
    <archiveClasses>true</archiveClasses>
  </configuration>
</plugin>
```

Por default, todo o código-fonte da aplicação web é copiado. Para excluir alguma parte, você pode utilizar o elemento `warSourceExcludes`. No exemplo, os dois asteriscos (**) significam "recursivamente". Para `**/CVS`, significa que todos os diretórios CVS serão eliminados do empacotamento.

A configuração `archiveClasses` indica que as classes Java para a aplicação (*.class) serão antes empacotadas em uma biblioteca JAR. O valor default é false.

Propriedades

Propriedades

- São valores escritos em notação especial que podem ser utilizados para substituir parâmetros de entrada para a configuração
 - São alteradas em um único lugar, mas podem ser utilizadas em vários
 - Facilitam a manutenção em caso de manutenção
 - Um bom exemplo é definir dependências que seguem um mesmo número de versão em suas coordenadas

Propriedades no Maven são valores escritos em uma notação especial que podem ser utilizados para substituir parâmetros de entrada para a configuração. As propriedades atuam como uma variável que pode ser utilizada em vários lugares, mas é definida e alterada em um único ponto. Claramente, a principal vantagem de se utilizar propriedades é facilitar a manutenção quando ocorrem mudanças.

Um bom exemplo são versões de artefatos. Para ilustrar, vamos considerar o Spring Framework, um famoso framework para desenvolvimento em Java. O Spring Framework é bem modular e possui vários artefatos, como podemos ver no fragmento de um arquivo pom.xml típico:

```
<!-- spring -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring-release-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring-release-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring-release-version}</version>
</dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring-release-version}</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring-release-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring-release-version}</version>
    <scope>runtime</scope>
</dependency>
```

Como podemos ver, ao invés de escrever o valor para a versão do Spring em cada artefato, foi utilizada uma propriedade chamada `spring-release-version`. A sintaxe para propriedades é iniciá-las com `$` e delimitar o nome entre chaves.

Para definir uma propriedade no Maven, basta incluí-la debaixo do elemento *properties*:

```
<project>
    <!-- Partes omitidas -->
    <properties>
        <spring-release-version>2.5.1</spring-release-version>
    </properties>
    <!-- Partes omitidas -->
</project>
```

Plug-in Report e Website do Projeto

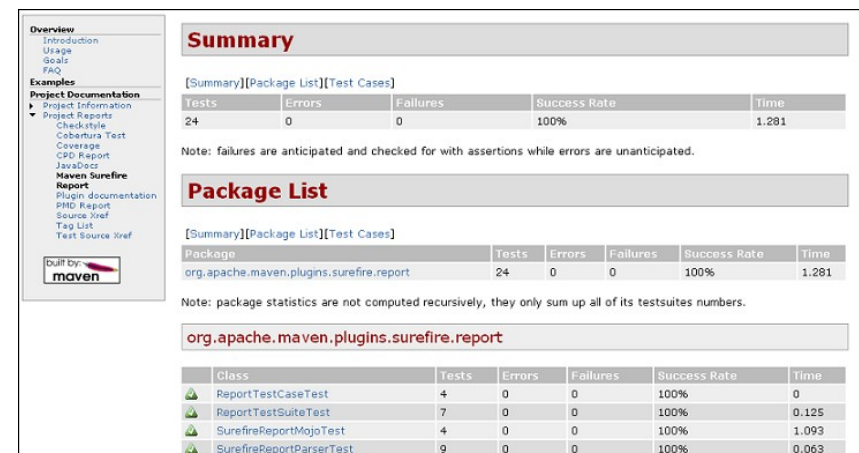
Plug-in Report e Website do Projeto

- A criação da infra-estrutura para gerar relatórios e um site de projeto pode demandar grande esforço
 - O Maven já traz consigo vários recursos para simplificar esta tarefa
 - Para criar um site rapidamente, utilize a fase site:


```
mvn site
```
 - Mas gerar os relatórios e o site sem um propósito claro não faz sentido algum
- Para incluir relatórios ao projeto, edite o arquivo pom.xml
 - Adicione o elemento reporting
 - Configure as opções de geração de relatórios
- Os relatórios e o site podem ser gerados em português (ou outro idioma)
 - Informe a opção locales para o o plug-in

A criação da infra-estrutura para gerar relatórios e o site para um projeto pode demandar grande esforço que quase sempre é repetitivo. Por isto o Maven é configurado para executar os relatórios do projeto por padrão. Não é necessário alterar nada para já se ter uma versão dos relatórios. Os relatórios receberão um conjunto de estilos (CSS) e farão parte do site da documentação do projeto. Basta executar:

```
mvn site
```



The screenshot shows the Maven Surefire Report website. On the left is a navigation menu with links like Overview, Introduction, Usage, Build, FAQ, Examples, Project Documentation, Project Information, Project Reports, Checkstyle, Cobertura Test, Coverage, CPD Report, JavaDocs, Maven Surefire Report, Plugin documentation, PMD Report, Source View, Tag List, and Test Source View. The main content area has a 'Summary' section with a table of tests, errors, failures, success rate, and time. Below that is a 'Package List' section with a table of packages and their test results. At the bottom is a table of classes and their test results.

Summary					
[Summary][Package List][Test Cases]					
Tests	Errors	Failures	Success Rate	Time	
24	0	0	100%	1.281	

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List					
[Summary][Package List][Test Cases]					
Package	Tests	Errors	Failures	Success Rate	Time
org.apache.maven.plugins.surefire.report	24	0	0	100%	1.281

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

org.apache.maven.plugins.surefire.report					
Class	Tests	Errors	Failures	Success Rate	Time
ReportTestCaseTest	4	0	0	100%	0
ReportTestSuiteTest	7	0	0	100%	0.125
SurefireReportMojoTest	4	0	0	100%	1.093
SurefireReportParserTest	9	0	0	100%	0.063

Figura 10: Exemplo de site gerado pelo Maven

Entretanto, fique atento aos comentários feitos por Tim O'Brien em seu blog. Ele critica o simples fato de se usar o Maven para gerar relatórios que não realmente utilizados. O blog está disponível em:

http://www.oreillyn.net.com/onjava/blog/2006/03/maven_project_info_reports_con.html

Nota: O plug-in Report tem várias opções de configuração. Muitas delas não serão abordadas no curso por serem avançadas. Para a lista completa de metas do plug-in, consulte:

Adicionando Relatórios ao Projeto

Para adicionar um relatório ao projeto, é necessário adicionar o plug-in ao elemento *reporting* do POM. Os relatórios que serão gerados devem ser informados debaixo do elemento *reports*:

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <reportSets>
          <reportSet>
            <reports>
              <report>dependencies</report>
              <report>project-team</report>
              <report>mailing-list</report>
              <report>cim</report>
              <report>issue-tracking</report>
              <report>license</report>
              <report>scm</report>
            </reports>
          </reportSet>
        </reportSets>
        <version>2.0.8</version>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Para visualizar a documentação, você poderá utilizar a meta:

```
mvn site:run
```

Nota: Para multiprojetos, os links somente ficarão corretos quando você executar a fase *site-deploy* (vista ainda neste capítulo).

Configurando o suporte ao idioma português do Brasil

Para configurar o suporte ao nosso idioma, informe o elemento *locales*:

```
<project>
  ...
```

```
...
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <configuration>
        <locales>pt_BR</locales>
      </configuration>
    </plugin>
    ...
  </plugins>
</reporting>
...
</project>
```

Utilizando o Plug-in Report

Utilizando o Plug-in Report

- Integração com controle de versão
- Listas de discussão para o projeto
- Licença para o projeto
- Equipe do projeto

Controle de Versão

O plug-in Report tem uma meta chamada `project-info-reports:scm` que gera o relatório com informações sobre o controle de versão do projeto. Para configurá-lo, é necessário alterar o POM do projeto:

```
<project>
...
<scm>
  <connection>
    scm:svn:http://eteg.net/svn/curso/trunk/site
  </connection>
  <developerConnection>
    scm:svn:https://eteg.net/svn/curso/trunk/site
  </developerConnection>
  <url>http://eteg.net/viewvc/curso/trunk/site</url>
</scm>
...
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <configuration>
        <webAccessUrl>
          http://eteg.net/viewvc/curso/trunk
```



```

        </webAccessUrl>
        <anonymousConnection>
            scm:svn:http://eteg.net/svn/curso/trunk
        </anonymousConnection>
        <developerConnection>
            scm:svn:https://eteg.net/svn/curso/trunk
        </developerConnection>
    </configuration>
</plugin>
...
</plugins>
<reporting>
...
</project>

```

Consulte o administrador da rede ou o gerente de configuração de seu projeto para saber quais são os valores válidos para as configurações acima.

Listas de Discussão

Permite configurar o relatório sobre as listas de discussão para o projeto. Exemplo:

```

<project>
...
    <mailingLists>
        <mailingList>
            <name>Lista de usuários</name>
            <subscribe>subscribe-curso@eteg.net</subscribe>
            <unsubscribe>unsubscribe-curso@eteg.net</unsubscribe>
        </mailingList>
    </mailingLists>
...
</project>

```

Licença para o projeto

Permite configurar a licença a ser usada. Se for especificada uma URL, ela será automaticamente incluída ao conteúdo:

```

<project>
...

    <licenses>
        <license>
            <name>GNU Lesser General Public License</name>

```

```
<url>http://www.gnu.org/copyleft/lesser.html</url>
<comments>
  Mais detalhes em http://hibernate.org/356.html.
</comments>
<distribution>repo</distribution>
</license>
</licenses>
...
</project>
```

Equipe do Projeto

Para detalhar a equipe do projeto, existem duas opções: detalhá-los como desenvolvedores que fazem parte da equipe principal ou como colaboradores. Por exemplo:

```
<project>
...
  <developers>
    <developer>
      <id>bart</id>
      <name>Bart Simpson</name>
      <email>bart at simpsons.com</email>
      <organization>20th Century Fox</organization>
      <roles>
        <role>Desenvolvedor</role>
      </roles>
    </developer>
  </developers>
  <contributors>
    <contributor>
      <name>Fred Flintstone</name>
      <email>fred at stoneage.org</email>
      <organization>Bedrock</organization>
      <roles>
        <role>Escritor técnico</role>
      </roles>
    </contributor>
  </contributors>
  ...
</project>
```

Utilizando outros Plug-ins para Relatórios

Utilizando Outros Plug-ins para Relatórios

- Resultado de testes unitários
- Análise de código
 - Utiliza a ferramenta de análise de código PMD
 - Pode encontrar desde problemas simples, como variáveis que são declaradas e não utilizadas até problemas mais complexos como código duplicado

Resultados de testes unitários

Para gerar o relatório com o resultado da execução dos testes unitários, configure o plug-in Surefire:

```
<project>
...
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
...
</project>
```

Análise de código

Uma excelente ferramenta de análise de código é o PMD (<http://pmd.sf.net>). Para gerar relatórios com esta ferramenta através do Maven, configure o plug-in:

```
<project>
...
<reporting>
  <plugins>
```

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <version>2.3</version>
  <configuration>
    <linkXref>true</linkXref>
    <targetJdk>1.5</targetJdk>
  </configuration>
</plugin>
</plugins>
</reporting>
...
</project>
```

Relatórios JavaDoc e JXR

Relatórios Javadoc e JXR

- O plug-in Javadoc permite gerar a documentação a partir do código-fonte Java do projeto.
- O plug-in JXR permite criar uma referência cruzada entre as classes Java, transformando o código-fonte em um conjunto de páginas HTML que podem ser navegadas
 - Por exemplo, são criados links entre as classes

O plug-in Javadoc permite gerar a documentação do projeto neste formato (apenas classes Java são suportadas). Para configurá-lo, adicione o relatório ao POM:

```
<project>
...
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
    </plugin>
  </plugins>
...
</reporting>
...
</project>
```

O plug-in JXR

Este plug-in produz a referência cruzada do código-fonte do projeto. Por meio do relatório gerado fica fácil navegar pelo código. Para utilizá-lo, configure o POM como neste exemplo:

```
<project>
...
<build>
...

```

```
</build>
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jxr-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
...
</project>
```

Nota: Vários outros relatórios podem ser gerados e também é possível adicionar conteúdo ao site, mas estes tópicos fogem do escopo deste curso. Para maiores informações, consulte o site do Maven.

Disponibilizando o Website

Disponibilizando o Website

- Para implantar o site gerado pelo Maven, primeiramente você deve ter um destino configurado no arquivo pom.xml:

```
<project>
...
<distributionManagement>
  <site>
    <id>website</id>
    <url>scp://empresa.com.br/site/docs/project/</url>
  </site>
</distributionManagement> ...
</project>
```

- Em seguida, execute a fase site-deploy
- Dependendo da configuração, será solicitado o login e a senha de conexão

Para implantar o site gerado pelo Maven, primeiramente você deve ter um local configurado no arquivo pom.xml:

```
<project>
...
<distributionManagement>
  <site>
    <id>website</id>
    <url>scp://www.empresa.com.br/site/docs/project/</url>
  </site>
</distributionManagement>
...
</project>
```

O elemento `<id>` identifica o repositório configurado no arquivo settings.xml usando o elemento `<servers>`. A informação sobre a autenticação deve estar configurada.

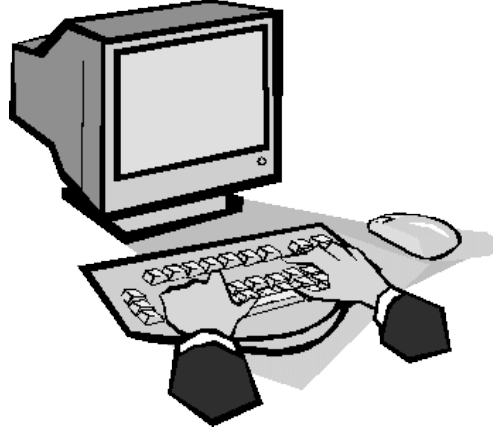
O elemento `<url>` configura o servidor onde será feita a disponibilização do site. Atualmente, somente cópia por SCP, RSync e cópia local são suportadas.

Para disponibilizar o site, utilize a fase site-deploy do ciclo de vida:

```
mvn site-deploy
```

Revisão e Laboratório

Revisão e Laboratório



- 1- Suponha que você queira utilizar no código a palavra-chave *assert* de Java, mas ficou sabendo que ela só está disponível nas versões iguais ou posteriores à 1.4. Como você pode configurar o Maven para compilar o código-fonte?
- 2- Cite uma vantagem do uso de propriedades.
- 3- Qual plug-in deve ser configurado para que a documentação gerada pelo Maven fique em português?
- 4- Qual plug-in permite gerar uma referência cruzada do código Java?

Bibliografia

<http://www.ibm.com/developerworks/java/library/j-maven/>

<https://www6.software.ibm.com/developerworks/education/j-mavenv2/section2.html>

<http://java.sys-con.com/node/393300>

<http://www.kleineikenscheidt.de/stefan/archives/2006/01/comparison-table-ant-vs-maven.html>

<http://www.javaworld.com/javaworld/jw-05-2006/jw-0529-maven.html>

<http://www.javaworld.com/javaworld/jw-12-2005/jw-1205-maven.html>

<http://maven.apache.org>