

# OSGi Modularity - Tutorial (traduzido)

segunda-feira, 25 maio 2015, 11:09 AM

## OSGi Modularidade - Tutorial

**Lars Vogel**

Versão 5.0

Copyright © 2008, 2009, 2010, 2012, 2013, 2014, 2015 vogella GmbH

2015/03/18

### OSGi com o Eclipse Equinox

Este tutorial apresenta um panorama da OSGi e sua camada de modularidade. Para este tutorial Eclipse 4.4 (Luna) é usado.

## Índice

### 1. Introdução na modularidade software com OSGi

- 1.1. Qual é a modularidade do software?
- 1.2. O que é OSGi?
- 1.3. Implementações OSGi
- 1.4. Plug-in ou feixes como componente de software
- 1.5. Convenção de nomenclatura: simples plug-in

### 2. metadados OSGi

- 2.1. O arquivo de manifesto (MANIFEST.MF)
- 2.2. Bundle-SymbolicName e Versão
- 2.3. O controle de versão semântica com OSGi

### 3. Definir as dependências do plug-in

- 3.1. Especificar dependências de plug-in através do arquivo de manifesto
- 3.2. Ciclo de vida de plug-ins em OSGi
- 3.3. Importações dinâmicas de pacotes

### 4. Definir a API de um plug-in

- 4.1. Especificando a API de um plug-in
- 4.2. API provisória
- 4.3. API provisória com exceções através x amigos-

### 5. Encontre o plug-in para uma determinada classe

### 6. Usando o console OSGi

- 6.1. O console OSGi
- 6.2. Bundles necessários
- 6.3. Telnet
- 6.4. O acesso ao console do Eclipse OSGi

### 7. Faça o download do Eclipse SDK

### Plug-in modelo de dados: 8. Exercício

- 8.1. Alvo do exercício

- 8.2. Crie o plug-in para o modelo de dados
- 8.3. Criar a classe base
- 8.4. Gerar construtores
- 8.5. Gerar métodos get e set
- 8.6. Ajuste o getter gerado e métodos setter
- 8.7. Gerar toString (), hashCode () e equals () métodos
- 8.8. Escreva um método copy ()
- 8.9. Criar a interface para o serviço TODO
- 8,10. Definir a API do plug-in modelo

## 9. Exercício: pacote de serviço

- 9.1. Alvo do exercício
- 9.2. Criar um plug-in provedor de modelo de dados (serviço de plug-in)
- 9.3. Definir as dependências do serviço de plug-in
- 9.4. Fornecer uma implementação da interface IToDoService
- 9.5. Criar uma fábrica
- 9.6. Exportar o pacote no serviço plug-in

## 10. Tutorial: Usando o Activator e exportação de seu pacote

- 10.1. Criar um novo Bundle
- 10.2. Codificação
- 10.3. Corrida
- 10.4. Exporte seu pacote

## 11. Executando um servidor OSGi autônomo

### 12. Sobre este site

- 12.1. Doações para apoiar cursos livres
- 12.2. Perguntas e discussão
- 12.3. Licença para este tutorial e seu código

### 13. Links e Literatura

- 13.1. Código Fonte
- 13.2. Recursos OSGi
- 13.3. Recursos vogella

# 1. Introdução na modularidade software com OSGi

## 1.1. Qual é a modularidade do software?

Um aplicativo é composto por diferentes partes, estas são normalmente chamados *componentes de software* ou *módulos de software* .

Esses componentes interagem uns com os outros através de uma interface de programação de aplicativos (API). O API é definida como um conjunto de classes e métodos que podem ser utilizados a partir de outros componentes. Um componente também tem um conjunto de classes e métodos que são considerados como internos ao componente de software.

Se um componente utiliza uma API a partir de outro componente, que tem uma dependência para outra componente, ou seja, ele requer o outro componente existe e funciona correctamente.

Um componente que é utilizado por outros componentes deve tentar manter a sua estabilidade API para evitar que uma alteração afectar outros componentes. Mas deve ser livre para mudar a sua implementação interna.

Java, na sua versão actual (Java 8), não fornece nenhuma forma estruturada para descrever dependências de componentes de software. Java suporta apenas o uso de modificadores de acesso, mas cada classe pública pode ser chamado de outro componente de software. O que se deseja é uma maneira de definir explicitamente a API de um componente de software. A especificação OSGi preenche esta lacuna.

## 1.2. O que é OSGi?

OSGi é um conjunto de especificações que, na sua especificação de núcleo, define um modelo de componentes e serviços para Java. Uma vantagem prática de OSGi é que cada componente de software pode definir sua API através de um conjunto de pacotes Java exportados e que cada componente pode especificar suas dependências necessárias.

Os componentes e serviços podem ser instalados de forma dinâmica, ativado, desativado, atualizado e desmontados.

## 1.3. Implementações OSGi

A especificação OSGi tem várias implementações, por exemplo Eclipse Equinox, Knopflerfish OSGi ou Apache Felix.

Eclipse Equinox é a implementação de referência da especificação OSGi base. É também o ambiente de tempo de execução em que se baseiam as aplicações Eclipse.

## 1.4. Plug-in ou feixes como componente de software

A especificação OSGi define um pacote como a menor unidade de modularização, ou seja, em OSGi um componente de software é um pacote. O modelo de programação Eclipse normalmente os chama de *plug-in*, mas estes termos são intercambiáveis. Um plug-in é sempre válido um pacote válido e um pacote válido é sempre um plug-in válido. Neste livro, o uso de *plug-in* é o preferido, para ser consistente com a terminologia do Eclipse plug-in de desenvolvimento.

Um plug-in é uma unidade auto-suficiente coesa, que define explicitamente suas dependências a outros componentes e serviços. Ele também define a sua API através de pacotes Java.

## 1.5. Convenção de nomenclatura: simples plug-in

Um plug-in pode ser gerado pelo Eclipse através do *Arquivo → Novo → Outros ... → Plug-In Desenvolvimento → Plug-In Projeto* entrada do menu. O assistente correspondente permite especificar várias opções. Este livro chama plug-ins gerados com as seguintes opções de um *simples plug-in* ou *pacote simples*.

- Sem Activator
- Não há contribuições para a interface do usuário
- Não é uma rica aplicação cliente 3.x
- Criação sem um modelo

# 2. metadados OSGi

## 2.1. O arquivo de manifesto (MANIFEST.MF)

Tecnicamente plug-ins OSGi são *.jar* arquivos com informações meta adicional. Esta meta-informação é armazenada no */MANIFEST.MF META-INF* arquivo. Este arquivo é chamado o *manifesto* de arquivo e faz parte da especificação Java padrão e OSGi acrescenta metadados adicionais para ele. De acordo com a especificação Java, qualquer Java runtime deve ignorar metadados desconhecido. Portanto, os plug-ins podem ser usados sem restrições em outros ambientes Java.

A listagem a seguir é um exemplo de um arquivo de manifesto.

```
Manifest-Version: 1.0
Bundle-manifestVersion: 2
Bundle-Name: Popup Plug-in
Bundle-SymbolicName: com.example.myosgi; Singleton: = true
Bundle-Version: 1.0 . 0
Bundle-Activator: com.example.myosgi.Activator
Exigir-Bundle: org.eclipse.ui,
    org.eclipse.core.runtime
```

Bundle-ActivationPolicy: preguiçoso  
Bundle-RequiredExecutionEnvironment: JavaSE- 1.6

A tabela a seguir dá uma explicação dos identificadores no arquivo de manifesto. Para obter informações adicionais sobre o identificador único see [Seção 2.2, "Bundle-SymbolicName e Version"](#) e para obter informações sobre o esquema de versão, que é normalmente usado em OSGi ver [Seção 2.3, "Semântica Versioning com OSGi"](#).

**Tabela 1. identificadores OSGi no arquivo de manifesto**

Identificador	Descrição
Bundle-Name	Breve descrição do plug-in.
Bundle-SymbolicName	O identificador único do plug-in. Se este plug-in está usando a funcionalidade de ponto de extensão Eclipse, deve ser marcado como Singleton. Você pode fazer isso adicionando a seguinte declaração após o identificador Bundle-SymbolicName:  <code>; Singleton: = true</code>
Bundle-Version	Define a versão plug-in e deve ser incrementado se uma nova versão do plug-in é publicado.
Bundle-Activator	Define uma classe ativador opcional que implementa o <code>BundleActivator</code> interface. Uma instância dessa classe é criada quando o plug-in é ativado. Sua <code>start ()</code> e <code>stop ()</code> métodos são chamados sempre que o plug-in é iniciado ou parado. Um ativador OSGi pode ser utilizado para configurar o plug-in durante a inicialização. A execução de um ativador aumenta o tempo de inicialização do aplicativo, portanto, esta funcionalidade deve ser usada com cuidado.
Bundle-RequiredExecutionEnvironment (BREE)	Especifique qual versão Java é necessário para executar o plug-in. Se essa exigência não for cumprida, o tempo de execução OSGi não carregar o plug-in.
Bundle-ActivationPolicy	Definir isso como <i>preguiçoso</i> vai contar o tempo de execução OSGi que este plug-in só deve ser ativado caso um de seus componentes, ou seja, classes e interfaces são usadas por outros plug-ins. Se não for definida, o tempo de execução Equinox não ativar o plug-in, ou seja, serviços prestados por este plug-in não estão disponíveis.
Bundle-ClassPath	O Bundle-ClassPath especifica onde para carregar as classes do pacote. O padrão é ". que permite que as classes de ser carregado a partir da raiz do feixe. Você também pode adicionar arquivos JAR a ele, estes são chamados de <i>arquivos JAR aninhados</i> .

## 2.2. Bundle-SymbolicName e Versão

Cada plug-in tem um nome simbólico que é definida através do `Bundle-SymbolicName` propriedade. O nome começa por convenção com o nome de domínio reverso do autor plug-in, por exemplo, se você possui o "example.com" domínio, em seguida, o nome simbólico começaria com "com.example".

Cada plug-in também tem um número de versão no `Bundle-Version` propriedade.

O `Bundle-Version` eo `Bundle-SymbolicName` identifica exclusivamente um plug-in.

## 2.3. O controle de versão semântica com OSGi

OSGi recomenda usar um `<principal>`. `<minor>`. `<remendo>` esquema para o número da versão que é definida através do `Bundle-Version` identificador de campo. Se você mudar de plug-in de código que você aumentar a versão de acordo com o seguinte conjunto de regras.

- `<Remendo>` aumenta se todas as alterações são compatíveis com versões anteriores.

- <Menor> aumenta se API pública mudou, mas todas as mudanças são compatíveis com versões anteriores.
- <Principal> é aumentada se as mudanças não são compatíveis com versões anteriores.

Para obter mais informações sobre este regime versão ver o [Eclipse Versão Numeração Wiki](#) .

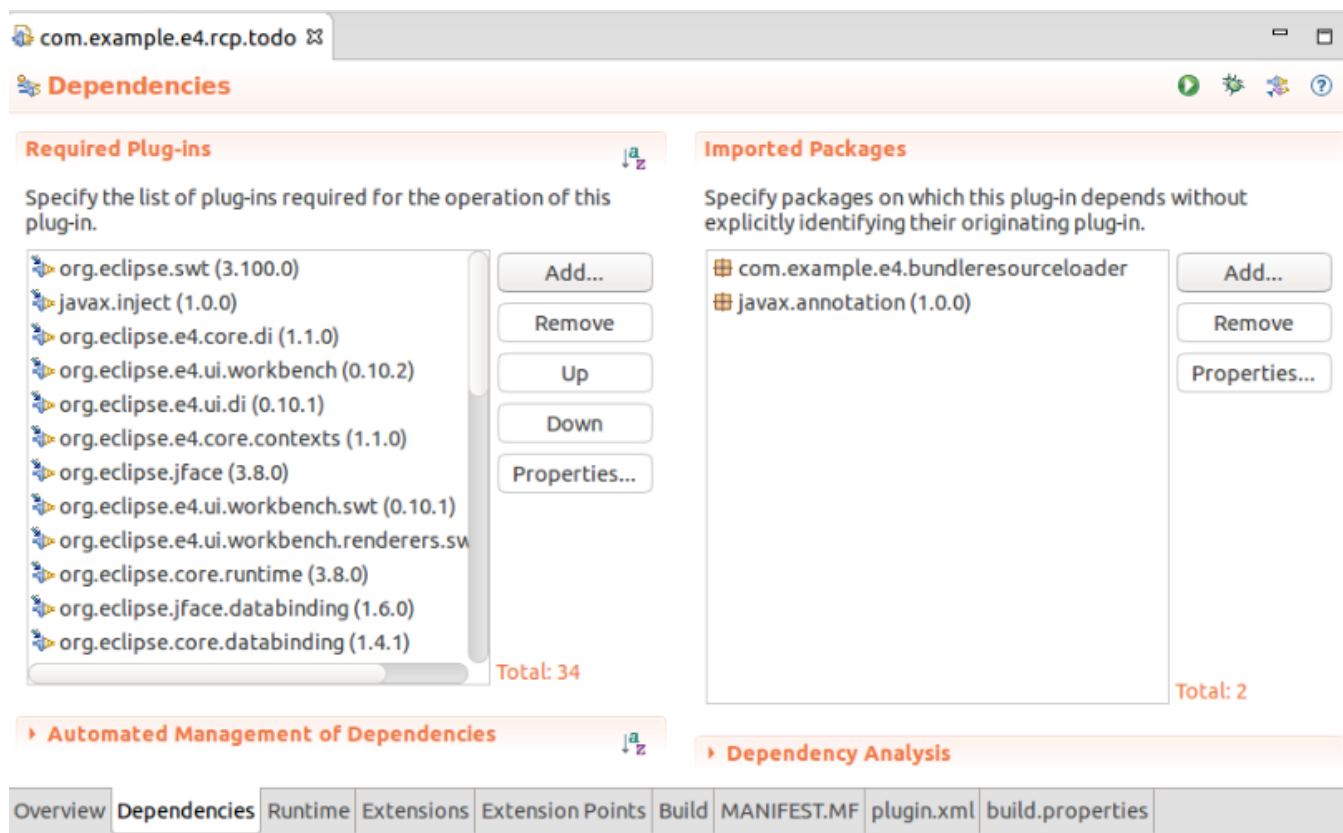
## 3. Definir as dependências do plug-in

### 3.1. Especificar dependências de plug-in através do arquivo de manifesto

Um plug-in pode definir dependências a outros componentes de software através do seu arquivo de manifesto. OSGi impede o acesso a aulas sem uma dependência definido e joga uma `ClassNotFoundException` . A única exceção são pacotes do Java Runtime Environment (com base na definição Bundle-RequiredExecutionEnvironment do plug-in); estes pacotes estão sempre disponíveis para um plug-in sem uma dependência explicitamente definido.

Se você adicionar uma dependência para o seu arquivo de manifesto, o Eclipse IDE adiciona automaticamente o correspondente JAR arquivo para o seu classpath do projeto.

Você pode definir as dependências de forma como dependências de plug-in ou dependências de pacotes. Se você definir uma dependência plug-in seu plug-in pode acessar todos os pacotes exportados deste plug-in. Se você especificar uma dependência de pacotes você pode acessar esse pacote. Usando dependências do pacote permite que você troque o plug-in que proporciona este pacote em um ponto posterior no tempo. Se você precisar deste flexibilidade preferem o uso de dependências de pacotes.



Um plug-in pode definir que depende de uma determinada versão (ou uma faixa) de outro pacote, por exemplo, plug-in A pode definir que depende de plug-in C na versão 2.0, enquanto plug-in B define que depende a versão 1.0 do plug-in C.

O tempo de execução OSGi garante que todas as dependências estão presentes antes de começar um plug-in. OSGi lê o arquivo de manifesto de um plug-in durante sua instalação. Ele garante que todos os plug-ins dependentes também são resolvidos e, se necessário, ativa-los antes que o plug-in é iniciado.

### 3.2. Ciclo de vida de plug-ins em OSGi

Com a instalação de um plug-in no tempo de execução OSGi o plug-in é mantido em um cache de pacote local. O tempo de execução OSGi, em seguida, tenta resolver suas dependências.

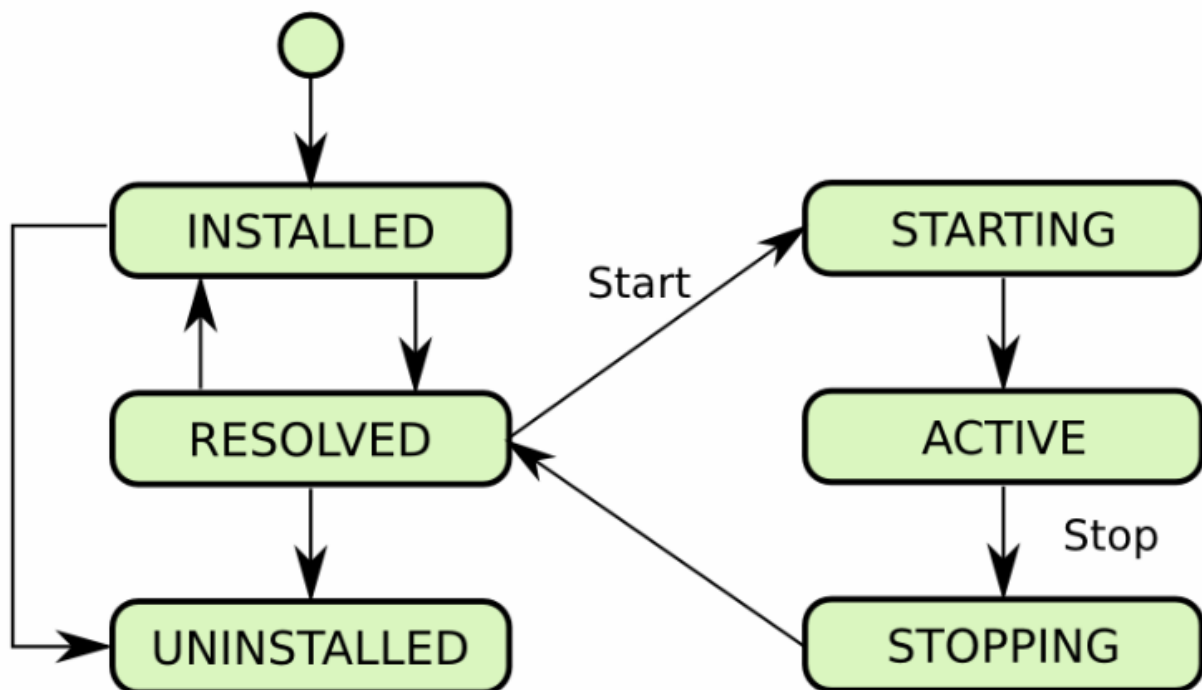
Se todas as dependências necessárias sejam resolvidos, o plug-in é no *RESOLVIDO* estado caso contrário ele permanece no *INSTALADO* status.

No caso de existir vários plug-ins que pode satisfazer a dependência, é usado o plug-in com a mais alta versão em vigor.

Se as versões são os mesmos, é usado o plug-in com o menor identificador exclusivo (ID). Cada plug-in recebe esse ID atribuída pelo quadro durante a instalação.

Quando o plug-in é iniciado, seu estado é *COMEÇAR* . Depois de um início bem sucedido, torna-se *ACTIVE* .

Este ciclo de vida é representado no gráfico a seguir.



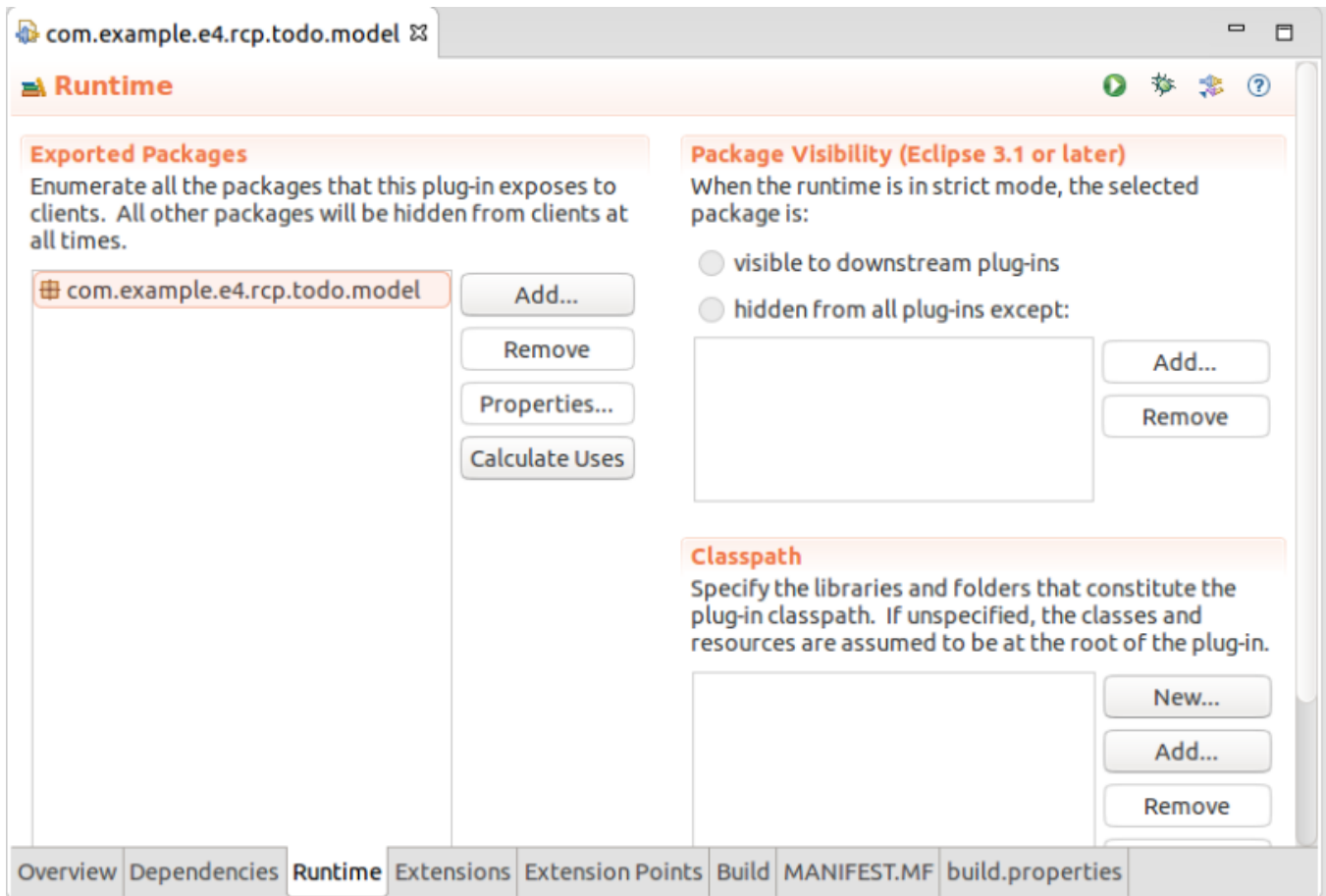
### 3.3. Importações dinâmicas de pacotes

Por razões de legado OSGi suporta uma importação dinâmica de pacotes. Veja [OSGi Wiki para importações dinâmicas](#) para detalhes. Você não deve usar esse recurso, é um sintoma de um projeto não-modular.

## 4. Definir a API de um plug-in

### 4.1. Especificando a API de um plug-in

No *MANIFEST.MF* arquivo de um plug-in também define a sua API através do Export-Package Identifier. Todos os pacotes que não são explicitamente exportados não são visíveis para outros plug-ins.



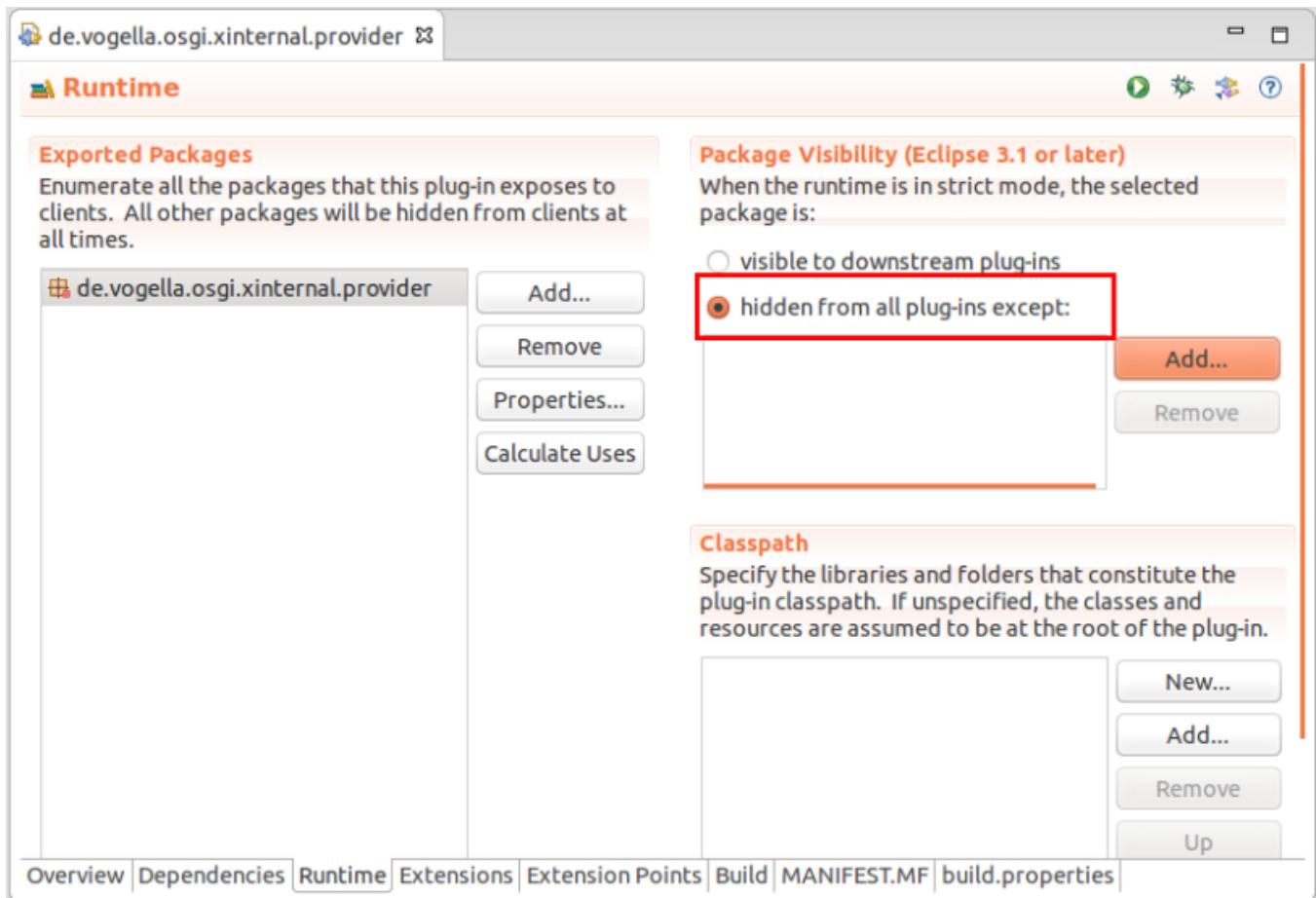
Todas estas restrições são aplicadas através de um OSGi específico `carregador de classe`. Cada plug-in tem seu próprio carregador de classe. Acesso a classes restrito não é possível.

Infelizmente OSGi não pode impedi-lo de usar Java reflexão para acessar essas classes. Isto é porque OSGi é baseado no tempo de execução Java que ainda não suporta uma camada de modularidade.

## 4.2. API provisória

Via o `-x interno` bandeira o tempo de execução OSGi pode marcar um pacote exportado como provisória. Isso permite que outros plug-ins para consumir as classes correspondentes, mas indica que essas classes não são considerados API oficial.

A figura abaixo mostra como definir um pacote como `x-interno` no editor de manifesto.



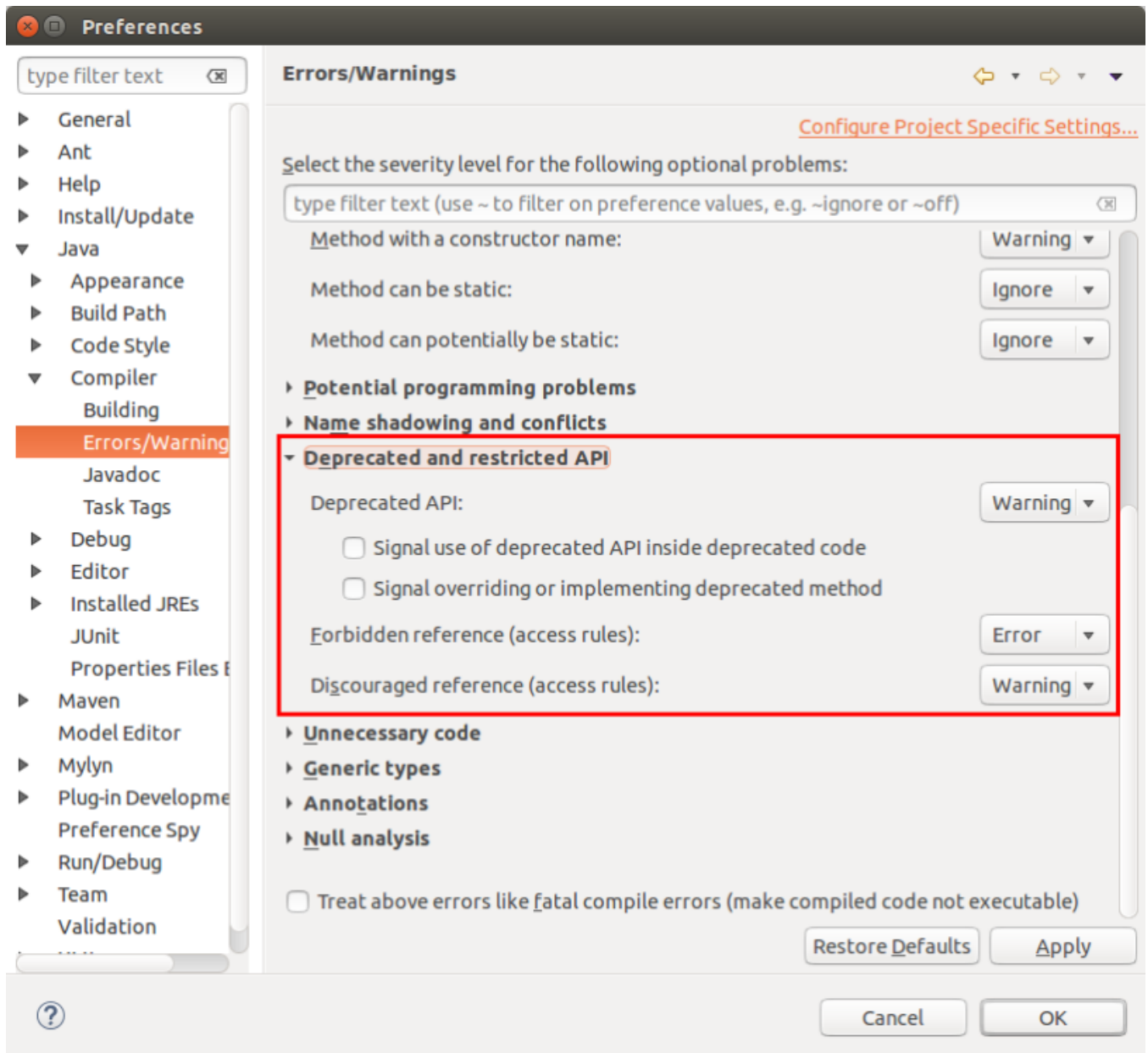
É assim que o arquivo de manifesto correspondente parece.

```
Manifest-Version: 1.0
Bundle-manifestVersion: 2
Bundle-Name: Provider
Bundle-SymbolicName: de.vogella.osgi.xinternal.provider
Bundle-Version: 1.0 .0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE- 1,6
Export-Package: de.vogella.osgi.xinternal.provider; x-interno: = true
```

Você pode configurar como o editor Eclipse Java mostra o uso da API provisório. Tal acesso pode ser configurado para ser exibido como, erro, aviso ou se tal acesso deve ser resultará em nenhuma mensagem adicional.

O padrão é exibir uma mensagem de aviso. Você pode ajustar isso nas preferências do Eclipse através dos *Janela* → *Preferências* → *Java* → *Compiler* → *Erros / Avisos* configuração de preferência.





#### 4.3. API provisória com exceções através x amigos-

Você pode definir que um conjunto de plug-ins pode acessar API provisória sem uma mensagem de aviso ou erro. Isto pode ser feito através do `x-amigos` directiva. Esta bandeira é adicionado se você adicionar um plug-in para o *Pacote Visibilidade* seção sobre o *Runtime* guia do editor de manifesto.

```
Manifest-Version: 1.0
Bundle-manifestVersion: 2
Bundle-Name: Provider
Bundle-SymbolicName: de.vogella.osgi.xinternal.provider
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: de.vogella.osgi.xinternal.provider; x-amigos: = "another.bundle"
```

O `-x amigos-` configuração tem o mesmo efeito que `-x interno` todos os plug-ins mencionados nos x amigos de definição de pode acessar o pacote sem receber um erro ou mensagem de aviso, mas.

## 5. Encontre o plug-in para uma determinada classe

Você frequentemente tem que encontrar o plug-in para uma determinada classe. O Eclipse IDE torna fácil encontrar o plug-in para uma classe. Após ativar o *Incluir todos os plug-ins de alvo em Java Pesquisa* ajuste nas preferências do IDE do Eclipse você pode usar o *tipo aberto* de diálogo ( **Ctrl + SHIFT + T** ) para encontrar o plug-in para uma classe. O *JAR* arquivo é mostrado neste diálogo eo prefixo do arquivo JAR é tipicamente o plug-in que contém essa classe.

## 6. Usando o console OSGi

### 6.1. O console OSGi

O console OSGi é como um shell de linha de comando. Neste console, você pode digitar um comando para executar uma ação OSGi. Isto pode ser útil para analisar problemas na camada OSGi de sua aplicação.

Use, por exemplo, o comando `ss` para obter uma visão geral de todos os pacotes, o seu estatuto e junte-id. A tabela a seguir é uma referência dos comandos mais importantes OSGi.

Tabela 2. comandos OSGi

Comando	Descrição
Socorro	Lista os comandos disponíveis.
ss	Lista os bundles instalados e seus status.
ss <i>vogella</i>	Listas de pacotes e seu status que têm <i>vogella</i> dentro de seu nome.
iniciar <bundle-id>	Inicia o pacote com o <bundle-id> ID.
parar <bundle-id>	Pára o pacote com o <bundle-id> ID.
diag <bundle-id>	Diagnostica um pacote particular. Ele lista todas as dependências em falta.
instalar <i>URL</i>	Instala um pacote a partir de uma URL.
desinstalar <bundle-id>	Desinstala o pacote com o <bundle-id> ID.
pacote <pacote-id>	Mostra informações sobre o pacote com o <bundle-id> ID, incluindo os serviços registrados e usados.
cabeçalhos	Mostra as informações MANIFEST.MF para um pacote.

<code>&lt;bundle-id&gt;</code>	
serviços <i>filtro</i>	Mostra todos os serviços disponíveis e seus consumidores. Filter é um filtro LDAP opcional, por exemplo, para ver todos os serviços que fornecem uma implementação ManagedService usar os "serviços (objectclass = * ManagedService)" comando.

## 6.2. Bundles necessários

Você tem que adicionar os seguintes pacotes para a configuração do tempo de execução para usar o console OSGi.

- org.eclipse.equinox.console
- org.apache.felix.gogo.command
- org.apache.felix.gogo.runtime
- org.apache.felix.gogo.shell

## 6.3. Telnet

Se você especificar o `-console` parâmetro na configuração do seu lançamento Eclipse permitirá que você interaja com o console OSGi. Uma configuração de lançamento OSGi criado com o IDE do Eclipse contém esse parâmetro por padrão. Via o seguinte parâmetro você pode abrir uma porta para o qual você pode se conectar através do protocolo telnet.

```
-console 5555
```

Se você abrir uma sessão telnet para o console OSGi, você pode usar a conclusão de tabulação e um histórico dos comandos semelhantes ao *Bash* shell no Linux.

## 6.4. O acesso ao console do Eclipse OSGi

Você também pode acessar o console OSGi de seu IDE Eclipse em execução. No *Console* Ver você encontrar uma entrada de menu com a dica de ferramenta *Abrir Console* . Se você selecionar *Anfitrião OSGi Console* , você terá acesso à sua instância OSGi em execução.

Por favor, note que interfira com o seu executando o Eclipse IDE através do console OSGi, pode colocar o Eclipse IDE em um estado ruim.

## 7. Faça o download do Eclipse SDK

Se você planeja construir funcionalidades adicionais sobre a plataforma Eclipse, você deve baixar a versão mais recente do Eclipse. Lançamentos oficiais são estáveis e uma boa base para a construção de sua funcionalidade adicional em cima dela.

O Eclipse IDE é fornecida em diferentes sabores. Embora você possa instalar as ferramentas necessárias em qualquer pacote Eclipse, é normalmente mais fácil de baixar a distribuição Eclipse padrão que contém todas as ferramentas necessárias para o plug-in de desenvolvimento. Outros pacotes acrescenta mais ferramentas que não são necessários para o Eclipse plug-in de desenvolvimento.


Navegue até o [site de download do Eclipse](#) e fazer o download do *Eclipse padrão* pacote.


Home Downloads Users Members Committers Resources Projects About Us Google™ Cu

# Eclipse Downloads

**Packages** Developer Builds

Eclipse Kepler (4.3.1) SR1 Packages for Linux


**Eclipse Standard 4.3.1**, 197 MB  
Downloaded 1,216,368 Times [Other Downloads](#)


 [Linux 32 Bit](#)  
[Linux 64 Bit](#)

The Eclipse Platform, and all the tools needed to develop and debug it: Java and Plug-in Development Tooling, Git and CVS...

## Package Solutions

[Filter Packages](#)

**Eclipse IDE for Java EE Developers**, 245 MB  
Downloaded 588,972 Times

 [Linux 32 Bit](#)  
[Linux 64 Bit](#)

Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...

## Plug-in modelo de dados: 8. Exercício

### 8.1. Alvo do exercício

Neste exercício, você cria um plug-in para a definição do seu modelo de dados. Você também tornam este modelo de dados disponíveis para outros plug-ins.

### 8.2. Crie o plug-in para o modelo de dados

Criar um plug-in do projeto simples (veja [Seção 1.5, "Convenção de nomenclatura: simples plug-in"](#)) chamado *com.example.e4.rcp.todo.model*.

A figura abaixo mostra a segunda página do assistente de projeto de plug-in e as configurações correspondentes. Pressione o *Concluir* botão nesta página para evitar o uso de modelos.

**New Plug-In Project**

**Content**  
Enter the data required to generate the plug-in.

**Properties**

ID:

Version:

Name:

Vendor:

Execution Environment:

**Options**

☐ Generate an activator, a Java class that controls the plug-in's life cycle  
Activator:

☐ This plug-in will make contributions to the UI

☐ Enable API analysis

**Rich Client Application**  
Would you like to create a 3.x rich client application? ☐ Yes ☒ No

### 8.3. Criar a classe base

Crie o `com.example.e4.rcp.todo.model` pacote ea classe seguinte modelo.

```
pacote com.example.e4.rcp.todo.model;

import java.util.Date;

público classe Todo {

    privado final de longo id;
    privada resumo String = "" ;
    privada description String = "" ;
    privada boolean done = false;
    privada Data dueDate;

}
```

**Nota:** Você vê um erro para o seu campo id final. Este erro é resolvido na [Seção 8.4, "construtores Gerar"](#) seção.

## 8.4. Gerar construtores

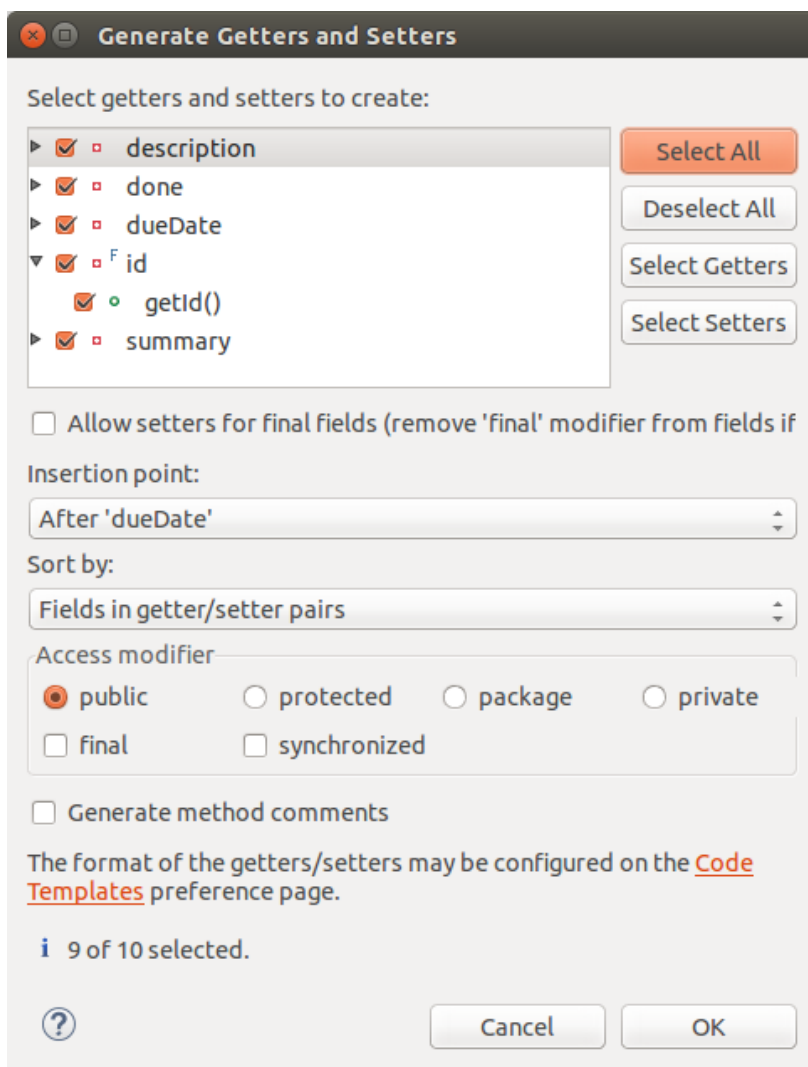
Selecione *Fonte* → *Gerar Construtor usando Campos ...* para gerar um construtor usando todos os campos. Usar a mesma abordagem para criar um outro construtor usando apenas o *ID* de campo.

**Aviso:** Certifique-se de ter criado os dois construtores, porque eles são necessários os seguintes exercícios.

## 8.5. Gerar métodos get e set

Use a *Fonte* → *Gerar Getter e Setter ...* menu para criar getters e setters para os seus campos.

**Nota:** O *ID* é definitiva e, portanto, Eclipse cria apenas um getter. Este correta e desejado.



## 8.6. Ajuste o getter gerado e métodos setter

Ajuste o getter gerado e setter para o `dueDate ()` campo para fazer cópias defensivas. A `Data de` classe não é imutável e queremos evitar que uma instância do `Todo` pode ser alterado a partir do exterior sem o setter correspondente.

```

pública Data getDueDate () {
    return new Date (dueDate.getTime ());
}

público vazio setDueDate (Data dueDate) {
    este .dueDate = new Date (dueDate.getTime ());
}

```

A classe resultante deve se parecer com a seguinte listagem.

```

pacote com.example.e4.rcp.todo.model;

import java.util.Date;

público classe Todo {

    privado final de longo id;
    privada resumo String = "" ;
    privada description String = "" ;
    privada boolean done = false;
    privada Data dueDate;

    pública Todo ( long id) {
        este .id = id;
    }

    pública Todo ( long id, resumo String, descrição String, boolean feito,
        Data dueDate) {
        este .id = id;
        este .summary = resumo;
        este .Descrição = descrição;
        este .done = feito;
        este .dueDate = dueDate;
    }

    pública de longo getId () {
        return id;
    }

    pública de Cordas getSummary () {
        return resumo;
    }

    público vazio setSummary (resumo String) {
        este .summary = resumo;
    }
}

```

```

}

pública de Cordas getDescription () {
    return descrição;
}

público vazio setDescription (descrição String) {
    este .Descrição = descrição;
}

público boolean isDone () {
    return feito;
}

público vazio setDone ( boolean feito) {
    este .done = feito;
}

pública Data getDueDate () {
    return new Date (dueDate.getTime ());
}

público vazio setDueDate (Data dueDate) {
    este .dueDate = new Date (dueDate.getTime ());
}

}

```

**Nota:** Porque é que o *ID* de campo marcado como final? Vamos usar este campo para gerar os `iguais` e `hashCode ()` métodos, portanto, ele não deve ser mutável. Alterar um campo que é usado nos `equals` e `hashCode ()` métodos podem criar erros que são difíceis de identificar, ou seja, um objeto está contido em um `HashMap` mas não foi encontrado.

## 8.7. Gerar `toString ()`, `hashCode ()` e `equals ()` métodos

Usar o Eclipse para gerar um `toString ()` método para o `Todo` classe com base no *id* e *resumo* campo. Isto pode ser feito através do menu Eclipse *Fonte* → *Gerar toString () ...* .

Também usar o Eclipse para gerar um `hashCode ()` e `equals ()` método baseado na *id* campo. Isto pode ser feito através do menu Eclipse *Fonte* → *Gerar hashCode () e equals () ...* .

## 8.8. Escreva um método `copy ()`

Adicione a seguinte `copy ()` método para a classe.

```

público cópia Todo () {
    retornar nova Todo ( este .id, este .summary,
        este .Descrição, este .done,
        este .dueDate);
}

```



## 8.9. Criar a interface para o serviço TODO

Crie o seguinte `ITodoService` interface.

```
pacote com.example.e4.rcp.todo.model;

import java.util.List;

público interface de ITodoService {

    Todo getTodo ( long id);

    boolean saveTodo (todo todo);

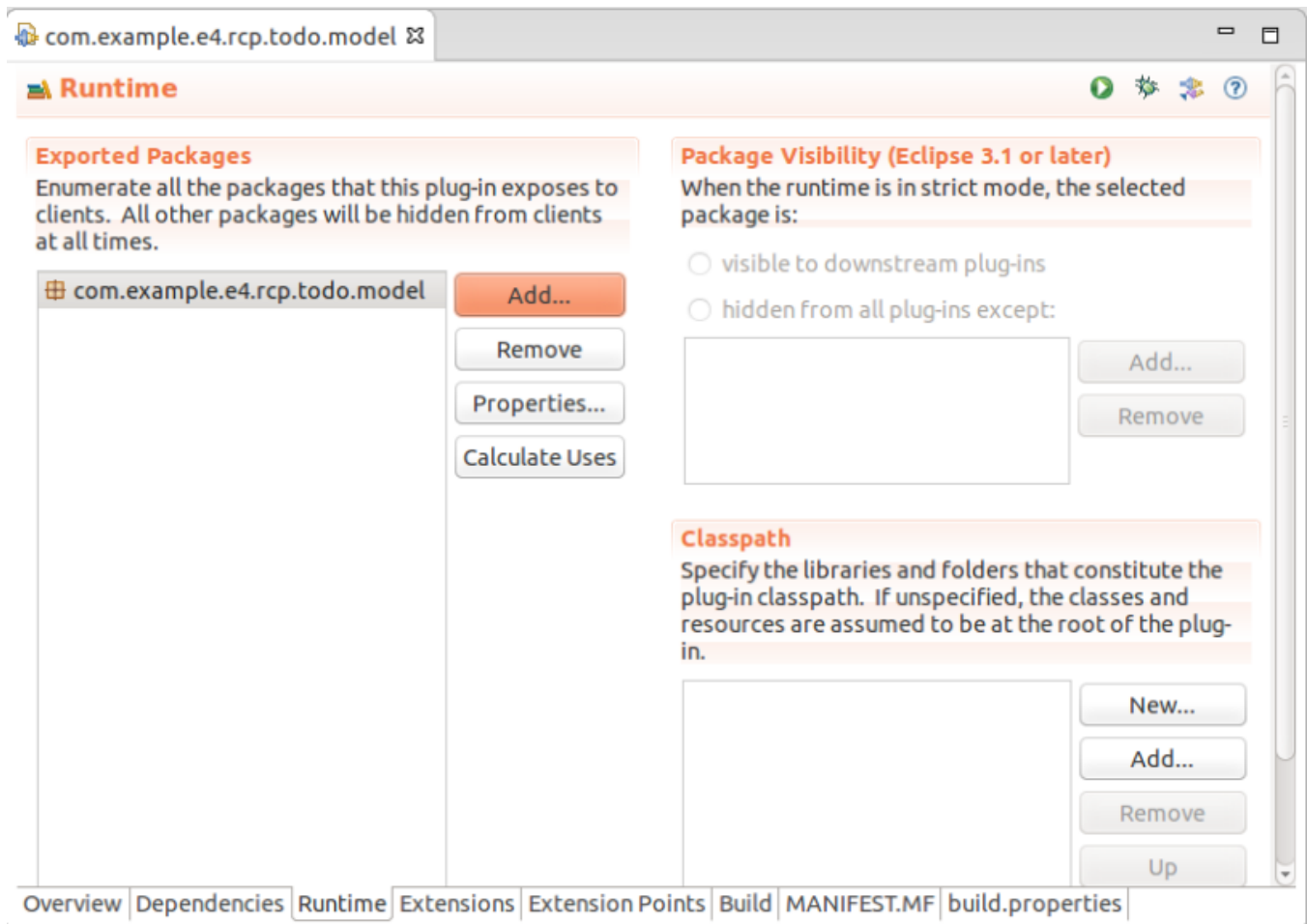
    boolean deleteTodo ( long id);

    List <Todo> getTodos ();
}
```

## 8.10. Definir a API do plug-in modelo

Definir que o `com.example.e4.rcp.todo.model` pacote é lançado como API. Isso exige que você exporte este pacote.

Para isso abra o `MANIFEST.MF` arquivo e selecione o *Runtime* guia. Adicionar `com.example.e4.rcp.todo.model` para os pacotes exportados.



## 9. Exercício: pacote de serviço

### 9.1. Alvo do exercício

Neste exercício, você cria um plug-in para uma implementação de serviço que permite o acesso aos dados.

Esta implementação do serviço utiliza o armazenamento de dados transitórios, ou seja, os dados não é mantida entre aplicativo for reiniciado. Para manter os dados você pode estender essa classe para armazenar os dados, por exemplo, em um banco de dados ou sistema de arquivos. Como este armazenamento não é especial para aplicações Eclipse RCP, não é abrangido por este livro.

### 9.2. Criar um plug-in provedor de modelo de dados (serviço de plug-in)

Criar um novo plug-in simples (veja [Seção 1.5, "Convenção de nomenclatura: simples plug-in"](#)) do projeto chamado `com.example.e4.rcp.todo.services`. Este plug-in é chamado *serviço TODO* plug-in na seguinte descrição.

**Ponta:** O sistema operacional MacOS pisa pastas que terminam com `.serviço` especial, por isso usamos os `.Os serviços` finais.

### 9.3. Definir as dependências do serviço de plug-in

Adicione o `com.example.e4.rcp.todo.model` plug-in como dependência para o seu serviço de plug-in. Para conseguir isso, abra o `MANIFEST.MF` arquivo e selecione o *Dependências* guia e adicione o `com.example.e4.rcp.todo.model` pacote para os *pacotes importados*.

### 9.4. Fornecer uma implementação da interface `ITodoService`

Crie o `com.example.e4.rcp.todo.services.internal` pacote em seu serviço plug-in e criar a seguinte classe.

```
empacotar com.example.e4.rcp.todo.services.internal;
```

```
importação java.util.ArrayList;
```

```
import java.util.Date;
```

```
import java.util.List;
```

```
importação com.example.e4.rcp.todo.model.ITodoService;
```

```
importação com.example.e4.rcp.todo.model.Todo;
```

```
/** * Impl exemplo de serviço, os dados não é mantido * Entre aplicativos é reiniciado */
```

```
público classe MyTodoServiceImpl implementa ITodoService {
```

```
    privado estático int atual = 1 ;
```

```
    privada List <Todo> todos;
```

```
público MyTodoServiceImpl () {
```

```
    todos = createInitialModel ();
```

```
}
```

```
// Sempre retornar uma nova cópia dos dados
```

```
Override
```

```
público List <> Todo getTodos () {
```

```
    List <Todo> list = new ArrayList <Todo> ();
```

```
    para (Todo todo: todos) {
```

```
        list.add (todo.copy ());
```

```
    }
```

```
    voltar lista;
```

```
}
```

```
// Criar um novo ou atualizar um objeto existente Todo
```

```
Override
```

```
público sincronizado boolean saveTodo (Todo newTodo) {
```

```
    Todo updateTodo = findById (newTodo.getId ());
```

```
    se (updateTodo == null) {
```

```
        updateTodo = new Todo (atual ++);
```

```
        todos.add (updateTodo);
```

```
    }
```

```
    updateTodo.setSummary (newTodo.getSummary ());
```

```
    updateTodo.setDescription (newTodo.getDescription ());
```

```
    updateTodo.setDone (newTodo.isDone ());
```

```
    updateTodo.setDueDate (newTodo.getDueDate ());
```

```
    retornar true;
```

```
}
```

```
Override
```

```
pública Todo getTodo ( long id) {
```

```
    Todo Todo findById = (id);
```

```

    se (TODO! = null) {
        return todo.copy ();
    }
    retornar null;
}

Override
público boolean deleteTodo ( long id) {
    Todo deleteTodo = findById (id);

    se (deleteTodo! = null) {
        todos.remove (deleteTodo);
        retornar true;
    }
    retornar false;
}

// Dados de exemplo
privado List <Todo> createInitialModel () {
    List <Todo> list = new ArrayList <Todo> ();
    list.add (createTodo ( "modelo de aplicação" , "flexível e extensível" ));
    list.add (createTodo ( "DI" , "Inject como modo de programação" ));
    list.add (createTodo ( "OSGi" , "Serviços" ));
    list.add (createTodo ( "SWT" , "Widgets" ));
    list.add (createTodo ( "JFace" , "Especialmente espectadores!" ));
    list.add (createTodo ( "CSS Styling" , "Estilo sua aplicação" ));
    list.add (createTodo ( "Eclipse serviços" , "Seleção, modelo, Part" ));
    list.add (createTodo ( "Renderer" , "Different kit de ferramentas UI" ));
    list.add (createTodo ( "camada de compatibilidade" , "Run Eclipse 3.x" ));
    retorno lista;
}

privado Todo createTodo (resumo String, descrição String) {
    retornar nova Todo (++ atual, resumo, descrição, falso, novo Date ());
}

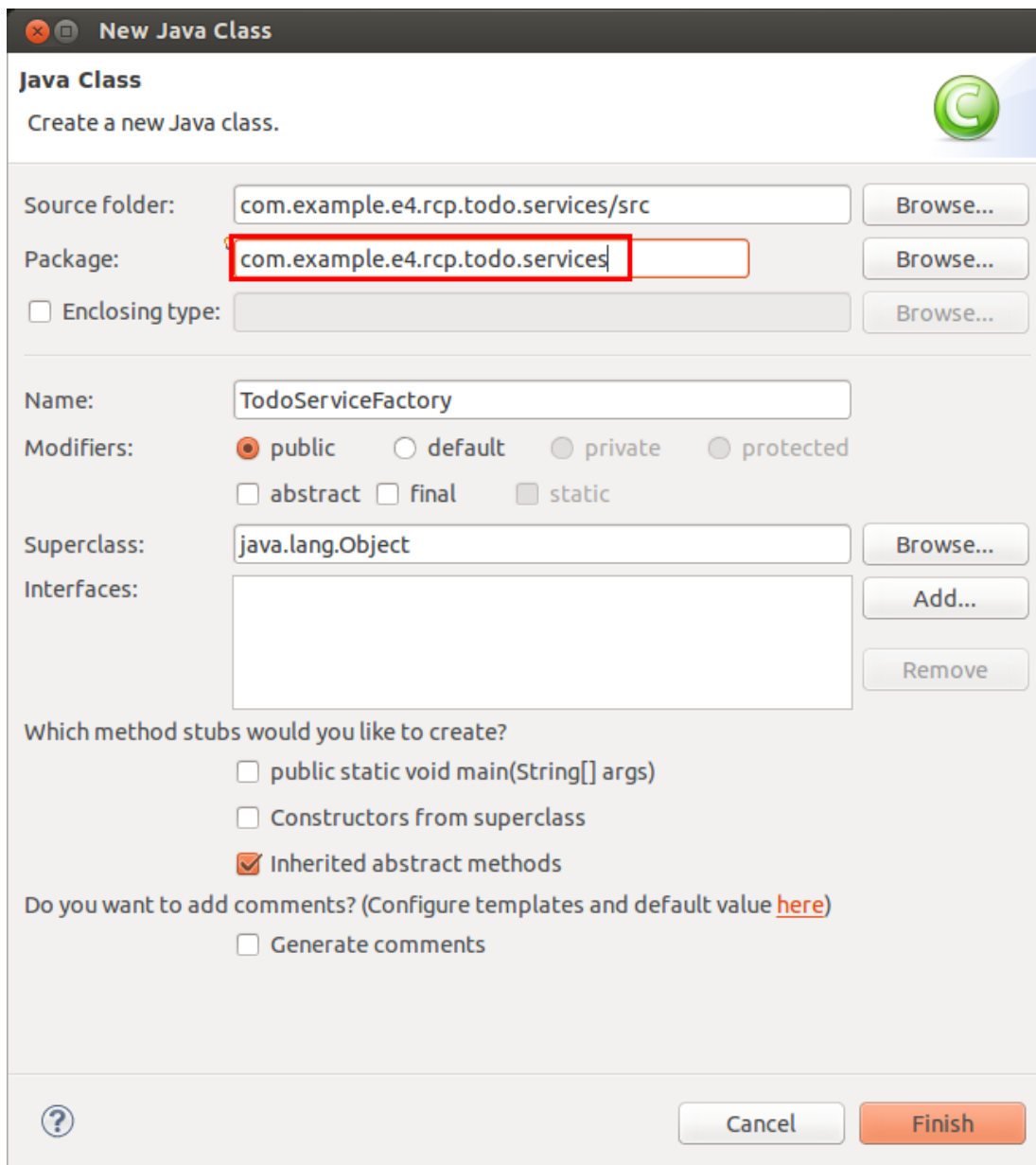
privado Todo findById ( long id) {
    para (Todo todo: todos) {
        se (id == todo.getId ()) {
            return TODO;
        }
    }
    retornar null;
}
}

```

## 9.5. Criar uma fábrica

Criar uma nova classe chamada *TodoServiceFactory* no `com.example.e4.rcp.todo.services` pacote. Para isso, você pode precisar a seguinte dica.

**Ponta:** Na sua configuração padrão, o Eclipse IDE esconde pacotes pais se eles não contêm quaisquer classes. Durante a especificação de sua classe, você pode definir o pacote correto. Isto está representado na figura seguinte.



The screenshot shows the 'New Java Class' dialog box in Eclipse. The 'Package' field is highlighted with a red rectangle and contains the text 'com.example.e4.rcp.todo.services'. The 'Name' field contains 'TodoServiceFactory'. The 'Superclass' field contains 'java.lang.Object'. The 'Which method stubs would you like to create?' section has 'Inherited abstract methods' checked. The 'Do you want to add comments?' section has 'Generate comments' unchecked. The 'Finish' button is highlighted in orange.

A classe fornece acesso ao seu `ITodoService` execução através de um método estático. Pode ser considerado como uma *fábrica* para a `ITodoService` interface. Uma fábrica esconde a criação do exemplo concreto de uma determinada interface.

```
pacote com.example.e4.rcp.todo.services;  
  
import com.example.e4.rcp.todo.model.ITodoService;  
import com.example.e4.rcp.todo.services.internal.MyTodoServiceImpl;
```

```
/ ** Fábrica fornece acesso ao prestador do serviço TODO ** /
```

```

público classe TodoServiceFactory {

    privados estáticos IToDoService todoService = novas MyToDoServiceImpl ();

    público estático IToDoService getInstance () {
        return todoService;
    }

}

```

## 9.6. Exportar o pacote no serviço plug-in

Exportar o `com.example.e4.rcp.todo.services` pacote através do `MANIFEST.MF` arquivo no *Runtime* guia, de modo que ele está disponível para outros plug-ins.

**Nota:** Por favor note que o conjunto de ferramentas Eclipse não suporta a exportação de embalagens vazias. Você tem que criar pelo menos uma classe no pacote antes de você é capaz de exportá-lo.

## 10. Tutorial: Usando o Activator e exportação de seu pacote

Neste exercício, você cria um outro pacote que usa um `Activator`. Você também executá-lo dentro do Eclipse. Ao final deste capítulo, você também vai exportar o seu pacote para utilizá-lo mais tarde em um servidor OSGi autônomo.

### 10.1. Criar um novo Bundle

Criar um novo projeto simples plug-in "com.vogella.osgi.firstbundle.internal" via *Arquivo* → *Novo* → *Outros ...* → *Plug-in Development* → *Plug-in do projeto*.

### 10.2. Codificação

Crie a seguinte classe de discussão.

```

empacotar com.vogella.osgi.firstbundle.internal;

público classe MyThread estende Tópico {
    privado volátil booleano ativo = true;

    público vazio run () {
        enquanto (ativo) {
            System.out.println ( "console OSGi Olá" );
            tentar {
                Thread.sleep ( 5000 );
            } preendedor (exceção e) {
                System.out.println ( "Thread interrompida" + e.getMessage ());
            }
        }
    }

    público vazio stopThread () {
        ativo = false;
    }
}

```

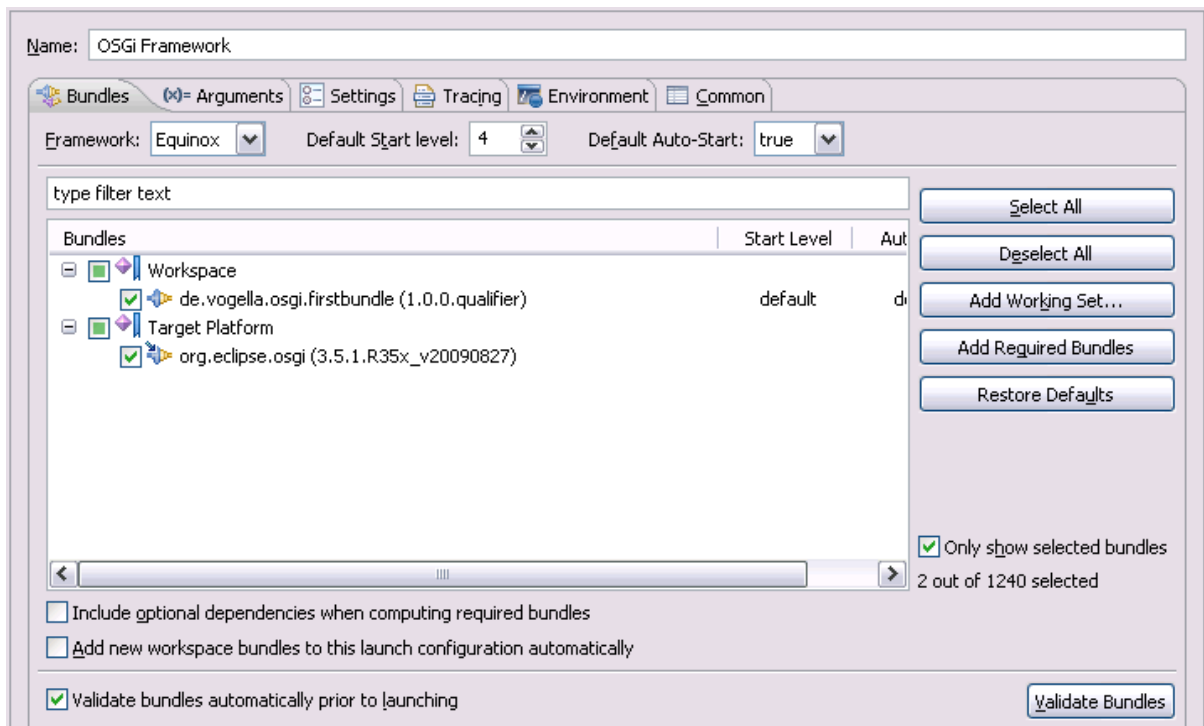
```
}  
}
```

Altere a classe `Activator.java` para o seguinte.

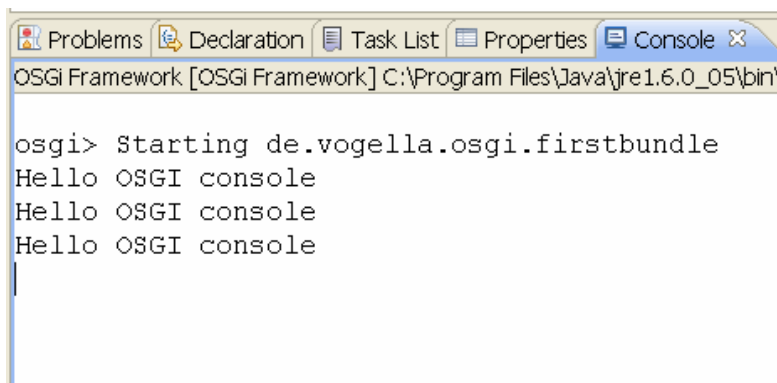
```
pacote com.vogella.osgi.firstbundle;  
  
importação org.osgi.framework.BundleActivator;  
importação org.osgi.framework.BundleContext;  
  
importação de.vogella.osgi.firstbundle.internal.MyThread;  
  
público classe Activator implementa BundleActivator {  
    privado MyThread myThread;  
  
    público vaziao partida (contexto BundleContext) lança Exceção {  
        System.out.println ( "Iniciando com.vogella.osgi.firstbundle" );  
        myThread = new MyThread ();  
        myThread.start ();  
    }  
  
    público vaziao stop (contexto BundleContext) lança Exceção {  
        System.out.println ( "Parando com.vogella.osgi.firstbundle" );  
        myThread.stopThread ();  
        myThread.join ();  
    }  
  
}
```

### 10.3. Corrida

Escolha o seu *MANIFEST.MF* arquivo, clique com o botão direito e selecione *Executar Como* → *Configuração de execução* . Criar uma configuração de lançamento OSGi Framework. desmarcar todos os pacotes, exceto o seu `de.vogella.osgi.firstbundle`. Depois pressione o *Add pacotes requeridos* . Isso adiciona o `org.eclipse.osgi` pacote à sua configuração de execução.



Execute esta configuração. Cada cinco segundo uma nova mensagem é gravada no console.

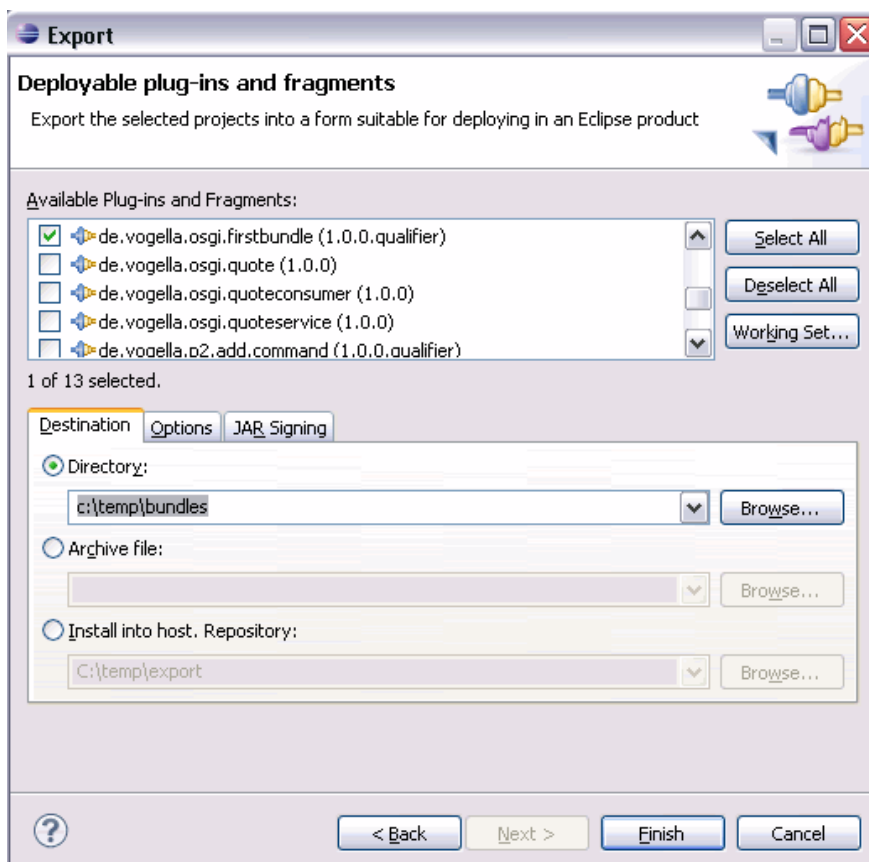
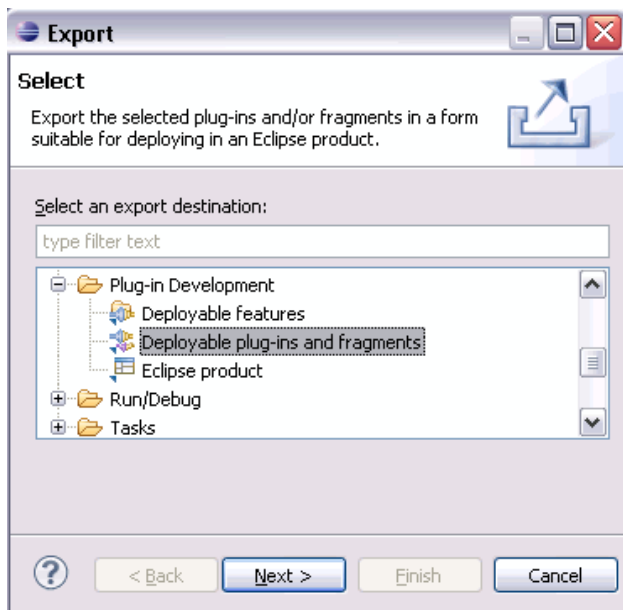


**Ponta:** No caso você está se perguntando em qual pasta OSGi começa você encontrá-lo em `{workspace_loc} /. Metadata / plugins / org.eclipse.pde.core / <runconfig>`. Esta lista pasta os bundles instalados por meio do arquivo "dev.properties" e defina o espaço de trabalho Eclipse como referência no arquivo config.ini através do `osgi.bundles = referência \ arquivo \ comunicado`. Desta forma, você pode atualizar seus pacotes no ambiente OSGi executando diretamente sem qualquer implantação.

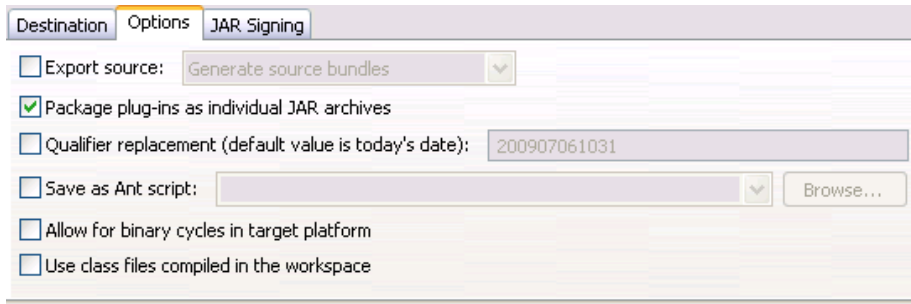
#### 10.4. Exporte seu pacote

Exporte seu pacote. Isso permitirá que você instalá-lo em um tempo de execução OSGi. Selecione seu pacote e escolha Arquivo -> Exportar -> Plug-in Development -> "plug-ins e fragmento Implementáveis".





Unflag a opção para exportar a fonte.



## 11. Executando um servidor OSGi autônomo

Este capítulo irá mostrar como executar Equinox OSGi como um stand-alone tempo de execução.

Em seu diretório de instalação do Eclipse identificar o arquivo `org.eclipse.osgi *.jar`. Este arquivo deve estar na pasta "plug-in". Copie esse arquivo jar para um novo local, por exemplo, `c: \ temp \ osgi-servidor`. Renomeie o arquivo para `"org.eclipse.osgi.jar"`.

Comece o seu servidor OSGi através do seguinte comando.

```
java -jar org.eclipse.osgi.jar -console
```

Você pode usar "instalar URL" para instalar um pacote a partir de uma determinada URL. Por exemplo, para instalar o seu pacote de "c: \ temp \ bundles" uso:

```
arquivo de instalação: c: \ temp \ feixes \ plugins \ de.vogella.osgi.firstbundle_1,0 .0  
.jar
```

**Ponta:** Você provavelmente precisará corrigir o caminho eo nome do pacote no seu sistema.

Você pode começar em seguida, o pacote com início eo id.


```
C:\temp\osgi-server>java -jar org.eclipse.osgi_3.4.2.R34x_v20080826-1230.jar -console  
osgi> install file:c:\temp\bundles\plugins\de.vogella.osgi.firstbundle_1.0.0.jar  
Bundle id is 1  
osgi> start 1  
Starting de.vogella.osgi.firstbundle  
osgi> Hello OSGI console
```

**Ponta:** Você pode remover todos os pacotes instalados com o parâmetro -clean.

## 12. Sobre este site

### 12.1. Doações para apoiar cursos livres

Por favor, considere uma contribuição , se este artigo lhe ajudou. Ela vai ajudar a manter nosso conteúdo e nossas atividades de

Open Source. 

### 12.2. Perguntas e discussão

Escrever e atualizar estes tutoriais é um monte de trabalho. Se este serviço à comunidade livre foi útil, você pode apoiar a causa, dando uma dica, bem como relatar erros de digitação e erros factuais.

Se você encontrar erros neste tutorial, por favor avise-me (veja o [topo da página](#) ). Por favor, note que, devido ao alto volume de feedback que eu receber, eu não posso responder perguntas a sua implementação. Certifique-se de ter lido o [vogella FAQ](#) como eu não responder a perguntas já respondidas lá.

### 12.3. Licença para este tutorial e seu código

Este tutorial é Open Content sob a [CC BY-NC-SA 3.0 DE](#) licença. O código-fonte neste tutorial é distribuído sob a [licença Eclipse Public](#) . Veja a [Licença vogella](#) página para obter detalhes sobre os termos de reutilização.

## 13. Links e Literatura

### 13.1. Código Fonte

[Fonte Código de exemplos](#)

### 13.2. Recursos OSGi

[Homepage OSGi](#)

[Homepage Equinox](#)

[Equinox guia de iniciação rápida](#)

[Serviços OSGi Blueprint](#)

### 13.3. Recursos vogella

#### TREINAMENTO

A empresa vogella fornece abrangentes **serviços de formação e de educação** de especialistas nas áreas de Eclipse RCP, Android, Git, Java, Gradle e Primavera. Nós oferecemos o treinamento público e inhouse. Independentemente do curso que você decida tomar, você está garantido para experimentar o que muitos antes de você se refere como **"a melhor classe de TI que já participei"** .

#### SERVIÇO E SUPORTE

A empresa vogella oferece **consultoria de especialistas** , serviços de apoio ao desenvolvimento e coaching. Nossos clientes variam de empresas Fortune 100 para os desenvolvedores individuais.