

Computação Orientada a Objetos

Organização de classes em pacotes

Slides baseados em:

Deitel, H.M.; Deitel P.J. **Java: Como Programar**, Pearson
Prentice Hall, 6a Edição, 2005. **Seções 8.16 e 8.17**

Tutorial Java da Sun, disponível em:

<http://java.sun.com/docs/books/tutorial/java/package/packages.html>

Profa. Karina Valdivia Delgado
EACH-USP

Pacotes (packages)

- Ponto forte do Java:
 - contém muitas classes predefinidas
- Essas classes são agrupadas em **pacotes**.
Por exemplo:
 - `java.lang`: classes fundamentais;
 - `java.io`: classes de E/S;
- O conjunto de pacotes de Java é chamado de **API** do Java (Java Application Programming Interface).

Pacotes (packages)

- Um **pacote** contém um conjunto de classes relacionadas
- O programador também pode agrupar suas próprias classes em pacotes.
- Para que?
 - facilitar a localização de tipos
 - facilitar a reutilização: programas podem importar classes de outros pacotes
 - evitar conflitos de nomes
 - fazer controle de acesso.

Exemplo

- Considere um grupo de classes para representar uma coleção de objetos gráficos (Shape, Circle, Rectangle e Triangle)

Exemplo

```
//no arquivo Shape.java  
public abstract class Shape {  
    . . .  
}
```

```
// no arquivo Circle.java  
public class Circle extends Shape {  
    . . .  
}
```

```
// no arquivo Rectangle.java  
public class Rectangle extends Shape {  
    . . .  
}
```

```
// no arquivo Triangle.java  
public class Triangle extends Shape {  
    . . .  
}
```

Criar um pacote **graphics** e colocar essas classes como parte do pacote

Por que agrupar classes em pacotes ?

- Para que você e outros programadores possam determinar facilmente que estes **tipos são relacionados**
- Para que os nomes de seus tipos **não entrem em conflito com nomes** de tipos de outros pacotes
 - cada pacote cria seu próprio espaço de nomes
- Para que os tipos dentro de seu pacote possam **acessar** uns aos outros de forma irrestrita, porém restringindo o acesso a tipos de outros pacotes.

Criando um pacote

- Para criar um pacote, coloque tipos (classes, interfaces, etc) dentro dele;
- A primeira linha de cada arquivo-fonte deve conter o comando *package* seguido do nome do pacote.
- Havendo múltiplas classes em um mesmo arquivo, somente *uma pode ser public*, e deve ter o mesmo nome do arquivo-fonte.
- Somente os membros públicos de um pacote são visíveis ao meio externo.

Exemplo

```
// no arquivo Circle.java  
  
package graphics;  
  
public class Circle extends Shape {  
    . . .  
}
```

- A classe *circle* é um membro público do pacote *graphics*.

Lembrando visibilidade

- `public` – o item em questão é visível a outras classes
- `private` – visível apenas aos componentes da classe atual
- `protected` – visível somente a classe atual e seus descendentes (pelo conceito de herança)

Acesso de pacote

- Se nenhum modificador de acesso (public, protected ou private) for usado para um método ou variável, eles terão **acesso de pacote**.
- Se temos múltiplas classes no mesmo pacote, essas classes poderão acessar diretamente os membros de acesso de pacote de outras classes por meio de referências a objetos
- O acesso de pacote é raramente utilizado

Exemplo

```
//classes no mesmo pacote
```

```
public class DataTest
{
    public static void main( String args[] )
    {
        Data data = new Data();
        data.number=77;
        data.string= "Oi";

    } // fim de main
} // fim da classe DataTest
```

```
public class Data
{
    int number;
    String string;
    ...

```

```
} // fim da classe Data
```

Nomeando um pacote

- Se nenhum nome de pacote for utilizado, seus tipos serão membros de um **pacote default**, que é um pacote sem nome

Esta prática só faz sentido em aplicações:

- muito pequenas
- de caráter temporário

Nomeando um pacote

- Com programadores Java do mundo todo escrevendo classes, interfaces etc, é provável que um mesmo nome seja dado a classes diferentes...
- Exemplo: a classe `Rectangle` já existe no pacote `java.awt`;
- No entanto, o compilador permite esta duplicidade.
- Por quê ?

Espaço de nomes

- As duas classes `Rectangle` do exemplo estão em pacotes distintos, e o nome completo de uma classe inclui o nome de seu pacote:
 - `graphics.Rectangle`
 - `java.awt.Rectangle`
- Mas e se dois programadores usarem o mesmo nome para seus pacotes ?

Convenção para nomes de pacotes

- Companhias usam seus nomes de domínio da Internet em ordem reversa para nomear seus pacotes, por ex:
 - `br.com.companhia.pacote`
- Conflitos de nomes só precisam ser resolvidos por convenção dentro da própria companhia, por ex:
 - `br.com.companhia.região.pacote`

Usando membros de um pacote

- Apenas tipos public de um pacote são visíveis fora do pacote no qual foram definidos
- O acesso pode ser feito de 3 formas:
 1. Fazendo referência ao nome completo do tipo
 2. Importando o tipo de seu pacote
 3. Importando o pacote inteiro.

1. Usando o nome completo

- Se o tipo está no mesmo pacote, basta usar seu nome simples. Ex: Rectangle
- Se o tipo pertence a outro pacote, podemos usar seu nome completo. Ex: graphics.Rectangle
- O nome completo pode ser usado normalmente em qualquer referência ao tipo em questão.Ex:

```
graphics.Rectangle rectangle = new graphics.Rectangle();
```

1. Usando o nome completo

e se precisarmos referenciar este mesmo tipo muitas vezes ?

- Se o tipo está no mesmo pacote, basta usar seu nome simples. Ex: Rectangle
- Se o tipo pertence a outro pacote, podemos usar seu nome completo. Ex: graphics.Rectangle
- O nome completo pode ser usado normalmente em qualquer referência ao tipo em questão.Ex:

```
graphics.Rectangle rectangle = new graphics.Rectangle();
```

2-Importando um tipo de um pacote

- Para importar um tipo específico de um pacote usamos `import` logo depois da definição do pacote. Ex:

```
import graphics.Rectangle;
```

- `Rectangle` pode então ser referenciado normalmente pelo seu nome simples. Ex:
`Rectangle rectangle = new Rectangle();`

2-Importando um tipo de um pacote

e se precisarmos de muitos tipos de um mesmo pacote?

- Para importar um tipo específico de um pacote usamos `import` logo depois da definição do pacote. Ex:

```
import graphics.Rectangle;
```

- `Rectangle` pode então ser referenciado normalmente pelo seu nome simples. Ex:

```
Rectangle rectangle = new Rectangle();
```

3-Importando um pacote inteiro

- Usa-se o comando `import.*` .Ex:
import graphics.;*
- Assim qualquer tipo do pacote `graphics` pode ser referenciado pelo seu nome simples
- Mas o caractere `*` não pode ser usado para representar substrings (curingas)!
import graphics.A;* // não funciona !!!

3-Importando um pacote inteiro

- Usa-se o comando `import t.*`. Ex:

```
import graphics.*;
```

- Assim qualquer tipo de objeto pode ser referenciado pelo nome do pacote

- Mas o caractere `*` não pode ser usado para representar substrings

```
import graphics.A*; // não funciona !!!
```

- Não implica em perda de performance em tempo de execução
- Pode trazer problemas com classes com o mesmo nome

Observação

- Pacotes não são hierárquicos !
- Importando `java.util.*` não significa que podemos referenciar a classe `Pattern` como `regex.Pattern`.
- É preciso referenciá-la como:
 - `java.util.regex.Pattern` ou
 - (se importamos `java.util.regex.*`) simplesmente como `Pattern`.

Resolvendo ambiguidades

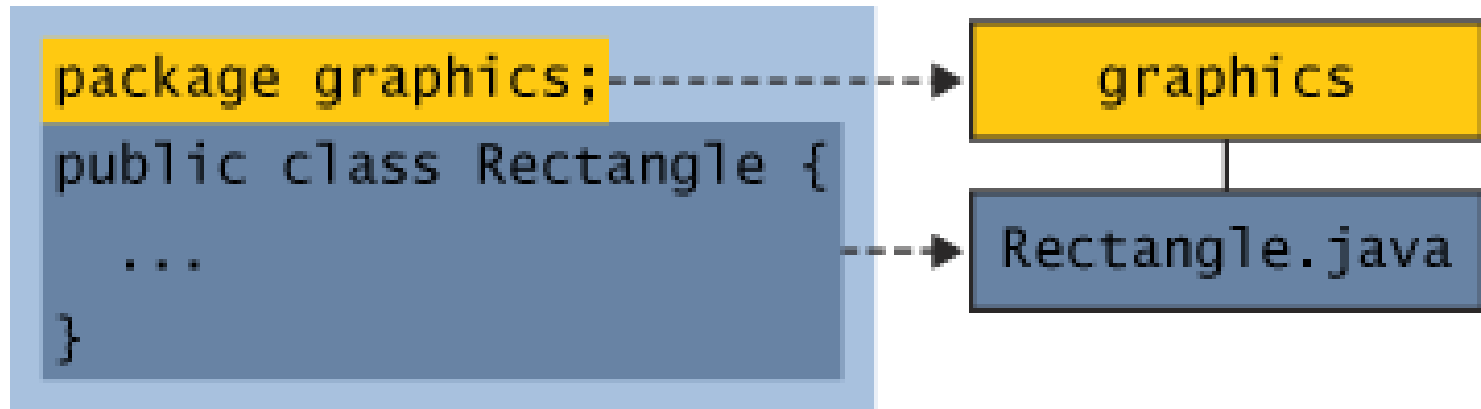
- E se importamos dois pacotes que possuem classes com o mesmo nome?
- Cada classe precisa ser referenciada pelo seu nome completo, por ex:

```
import graphics;  
import java.awt;  
  
Rectangle rectangle;           // ambíguo  
graphics.Rectangle rectangle;  // correto
```


Gerenciamento de arquivos fonte e de classes

- O código-fonte de uma classe é armazenado em um arquivo texto cujo nome é o próprio nome do tipo, e cuja extensão é `.java`
- Este arquivo é colocado em um diretório de nome igual ao do pacote ao qual ele pertence

Gerenciamento de arquivos fonte e de classes (cont.)



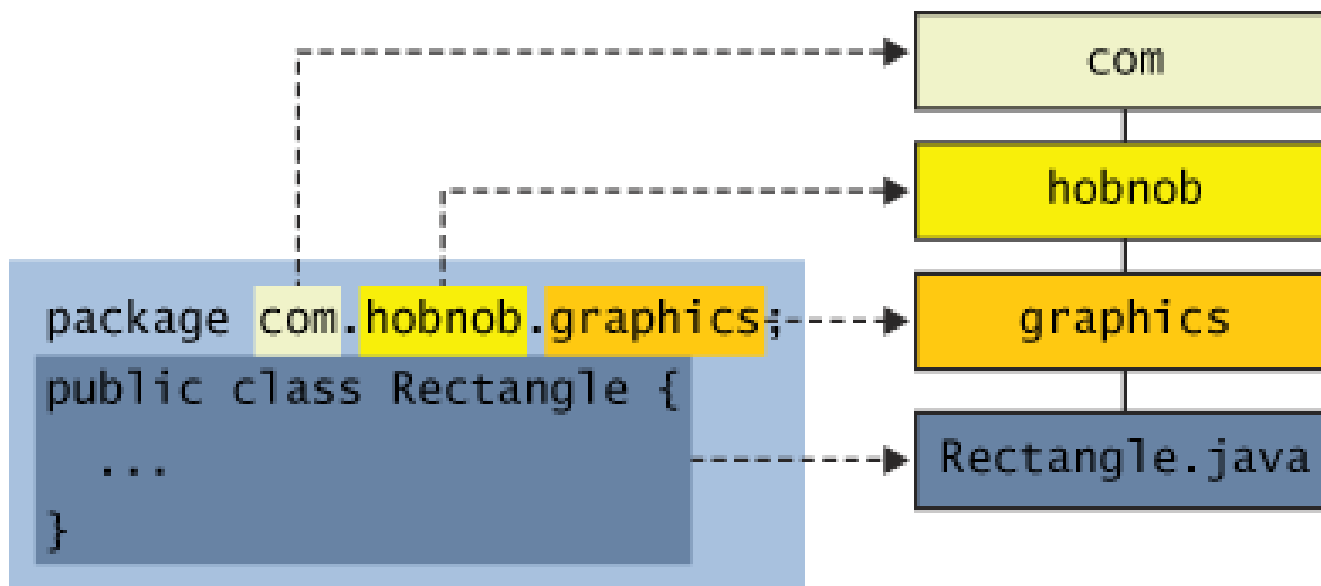
- Em um sistema de arquivos a estrutura de nomes reflete a estrutura de diretórios. Ex:

Nome da classe: `graphics.Rectangle`

Caminho ao arquivo: `graphics\Rectangle.java`

Gerenciamento de arquivos fonte e de classes (cont.)

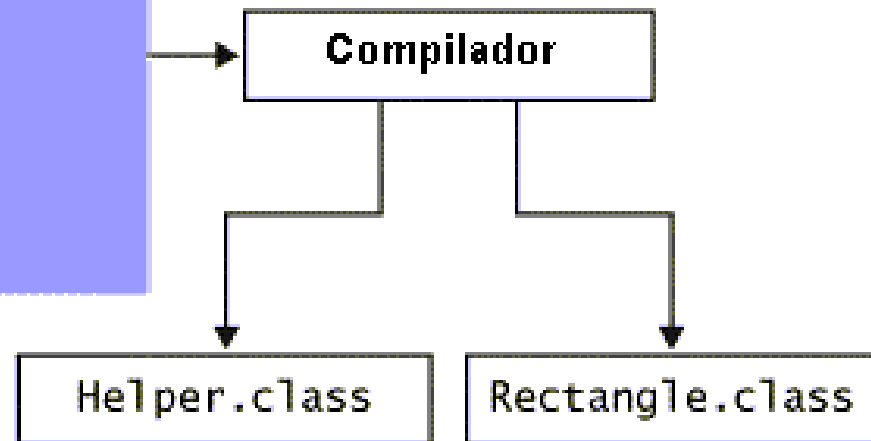
- Se hobnob.com criasse o pacote `graphics` do exemplo, a estrutura de diretórios seria a seguinte:



Arquivos de saída (.class)

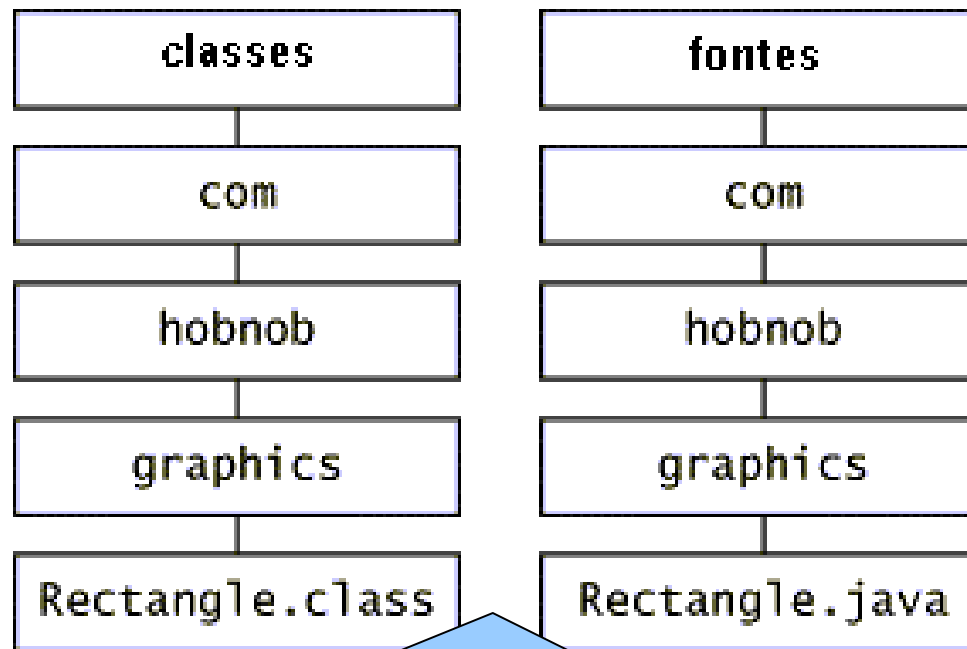
- Ao ser compilado, um arquivo-fonte (.java) gera um arquivo de saída de mesmo nome para cada classe, interface etc, porém com a extensão .class.

```
package com.hobnob.graphics;  
public class Rectangle {  
    ...  
}  
public class Helper {  
    ...  
}
```



Gerenciando `.java` e `.class`

- Os arquivos `.class` também devem ficar em uma estrutura de diretórios refletindo seus nomes, mas não necessariamente junto dos arquivos-fonte.



Esta separação permite a disponibilização de programas aos usuários sem revelar o código-fonte.

Resumo

- Para criar um pacote, coloque as classes e interfaces dentro dele e use a declaração `package` na primeira linha do código-fonte;
- Ao usar uma classe de um pacote externo, há três opções:
 - Referenciá-la pelo nome completo;
 - Importá-la e usar o nome simples;
 - Importar o pacote inteiro.

Exercício

- Suponha que você criou as classes abaixo no pacote default, e agora decidiu que elas devem ser organizadas em pacotes como segue:

Pacote

- `mygame.server`
- `mygame.shared`
- `mygame.client`

Classe

`Server`
`Utilities`
`Client`

- (1) que linha de código adicionar a cada classe ?
- (2) quais diretórios criar e quais arquivos colocar em cada um ?

Resposta

- ✓ Em `Server.java`, adicionamos: `package mygame.server;`
- ✓ Em `Utilities.java`, adicionamos: `package mygame.shared;`
- ✓ Em `Client.java`, adicionamos: `package mygame.client;`

2)

Dentro do diretório `mygame` criamos três subdiretórios: `server`, `shared` e `client`.

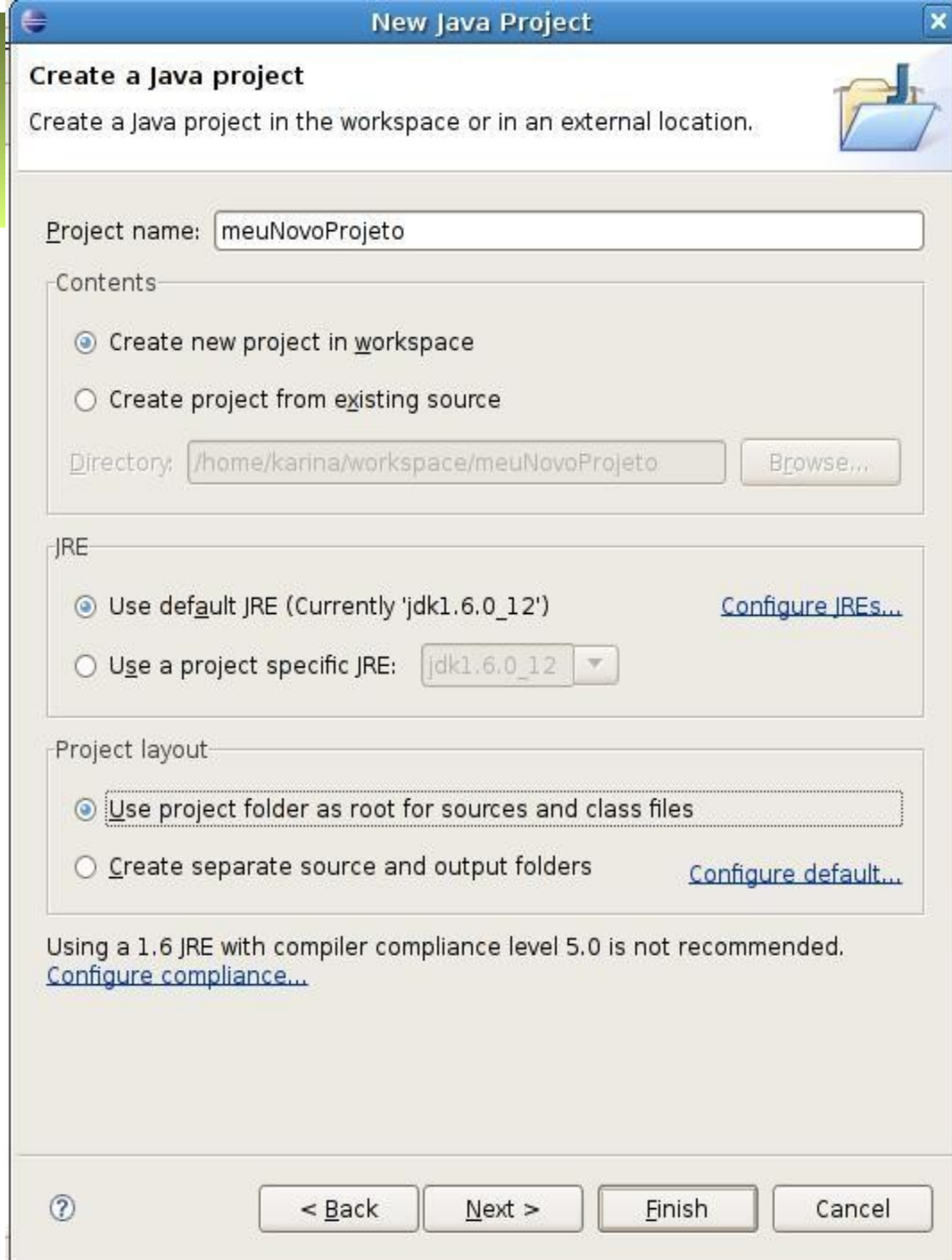
- ✓ Em `mygame/server/` colocamos: `Server.java`
- ✓ Em `mygame/shared/` colocamos: `Utilities.java`
- ✓ Em `mygame/client/` colocamos: `Client.java`

Pacotes no Eclipse

- O ambiente Eclipse apresenta um conjunto de facilidades para gerenciamento de pacotes de forma automática
- Ao criar uma nova classe é possível especificar o pacote a qual ela pertence
- O Eclipse se encarrega da criação de diretórios

Criando um novo projeto

- 1) Menu File >> New >> Java Project
- 2) Dê um nome ao projeto



The screenshot shows the 'New Java Project' dialog box. The title bar reads 'New Java Project'. The main heading is 'Create a Java project', followed by the instruction 'Create a Java project in the workspace or in an external location.' and a folder icon. The 'Project name' field contains 'meuNovoProjeto'. Under the 'Contents' section, the radio button 'Create new project in workspace' is selected. The 'Directory' field shows '/home/karina/workspace/meuNovoProjeto' with a 'Browse...' button. The 'JRE' section has 'Use default JRE (Currently 'jdk1.6.0_12')' selected, with a 'Configure JREs...' link. The 'Project layout' section has 'Use project folder as root for sources and class files' selected, with a 'Configure default...' link. A warning message at the bottom states: 'Using a 1.6 JRE with compiler compliance level 5.0 is not recommended. [Configure compliance...](#)'. The bottom navigation bar includes a help icon, '< Back', 'Next >', 'Finish', and 'Cancel' buttons.

Create a Java project
Create a Java project in the workspace or in an external location.

Project name:

Contents

☒ Create new project in workspace

☐ Create project from existing source

Directory:

JRE

☒ Use default JRE (Currently 'jdk1.6.0_12') [Configure JREs...](#)

☐ Use a project specific JRE: ▼

Project layout

☒ Use project folder as root for sources and class files

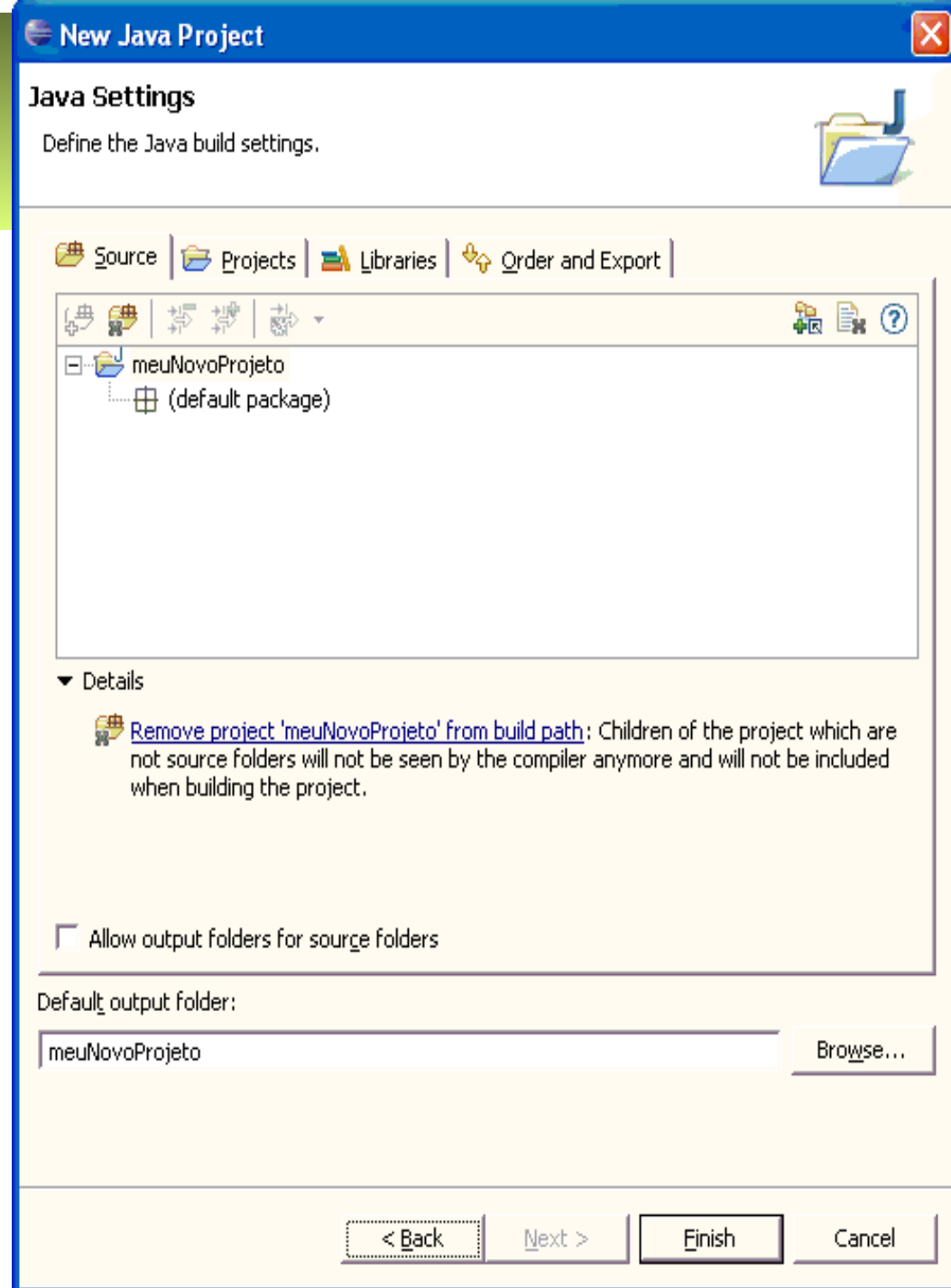
☐ Create separate source and output folders [Configure default...](#)

Using a 1.6 JRE with compiler compliance level 5.0 is not recommended.
[Configure compliance...](#)

? < Back Next > Finish Cancel

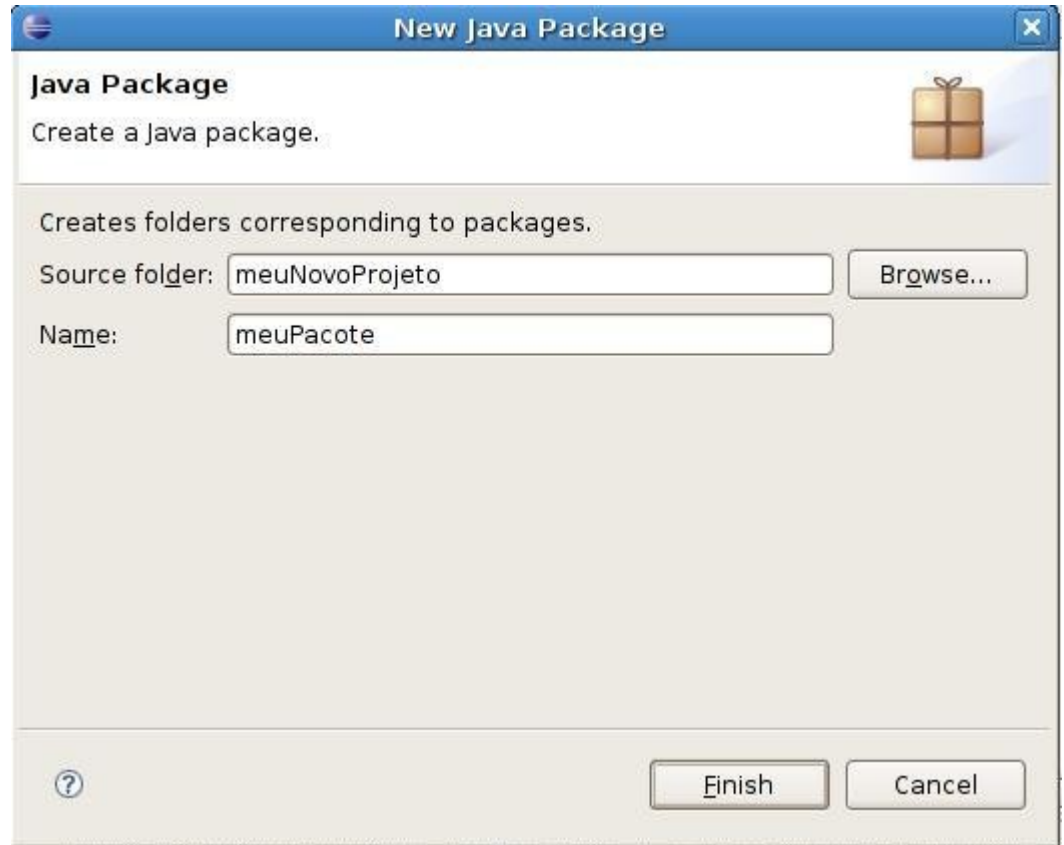
Criando um novo projeto

- 1) Menu File >> New >> Java Project
- 2) Dê um nome ao projeto



Criando um novo pacote

- 1) Menu File >> New >> Package
- 2) Dê um nome ao pacote



Criando uma nova classe

- 1) Menu File >> New >> Class
- 2) Dê um nome à classe
- 3) Se usar pacote diferente de Default, dê nome ao pacote no campo Package

Certifique-se de especificar se deseja criar um método main;

New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments as configured in the [properties](#) of the current project?
☐ Generate comments

Criando uma nova classe

A screenshot of the Eclipse IDE interface. The top window is titled 'MinhaClasse.java'. The code editor shows a Java package declaration 'package meuPacote;' followed by a class declaration 'public class MinhaClasse {'. Inside the class, there is a Javadoc comment '/** * @param args */' and a 'main' method signature 'public static void main(String[] args) {' with a green comment '// TODO Auto-generated method stub'. The code is color-coded: keywords in red, package/class names in black, comments in green, and parameters in blue. The IDE has a standard toolbar and a scroll bar on the right.

```
package meuPacote;

public class MinhaClasse {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Eclipse cria o “esqueleto” da classe conforme estipulado, já dentro de seu pacote (se for o caso).