Formabase
Aprenda passo a passo

Java
Funcionalidades Eclipse

Metodologia

- Interprete o documento calmamente e com atenção.
- Acompanhe a execução do exercício no seu computador.
- Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

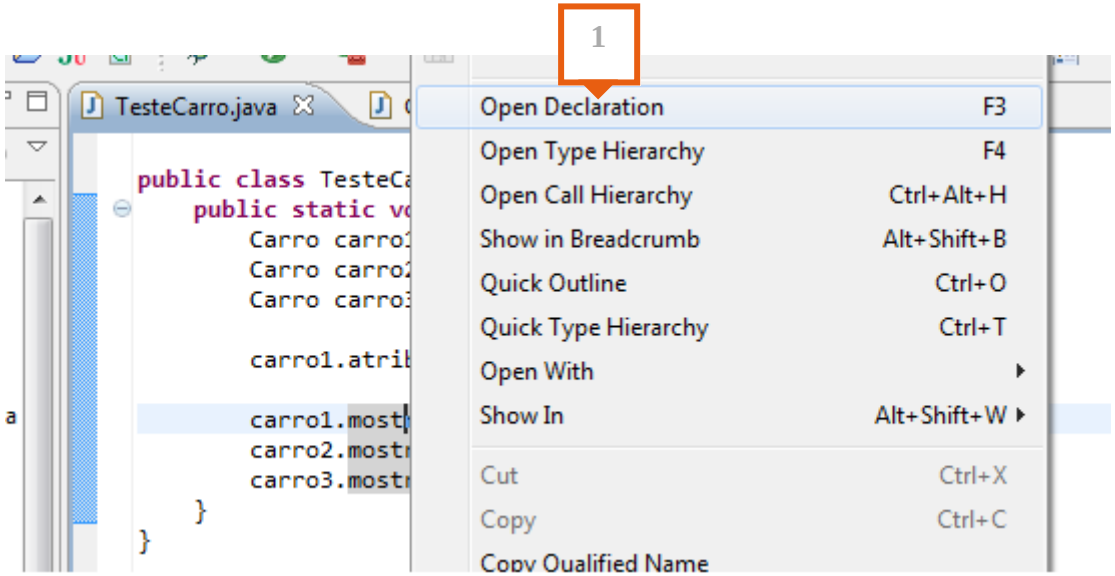
Conteúdo programático

1. Open Declaration

Esta funcionalidade do Eclipse permite aceder a métodos e atributos de forma imediata. Através de um pedaço de código com uma chamada a uma função, ou o uso de um atributo, é possível navegar diretamente até ao sítio onde este está definido, de forma a percebermos o que está a ser feito.

- **Abra** a classe `TesteCarro.java`

- Agora, sobre o texto "mostrar" da linha `carro1.mostrar();` **clique** no **botão direito** do rato e **selecione** a opção **Open Declaration** 1



- Repare como o Eclipse abriu automaticamente a classe `Carro` com o cursor no início do método `mostrar`.

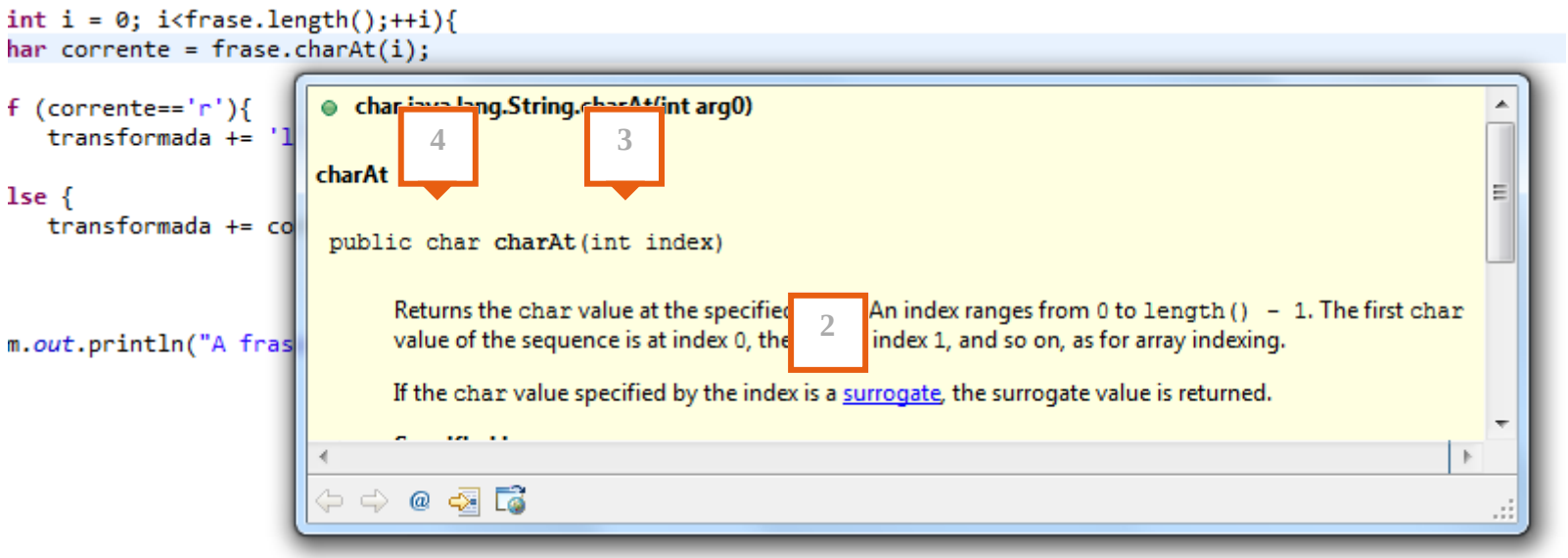
Esta funcionalidade tanto pode ser usada em métodos como atributos e variáveis.

2. Help

Em vez da navegação para o código de um método, é possível ter acesso a uma pequena descrição do que este faz.

- **Abra** a classe `ManipulacaoStrings.java`

- **Coloque** o **cursor** no método `charAt` e **pressione** a tecla `F2`

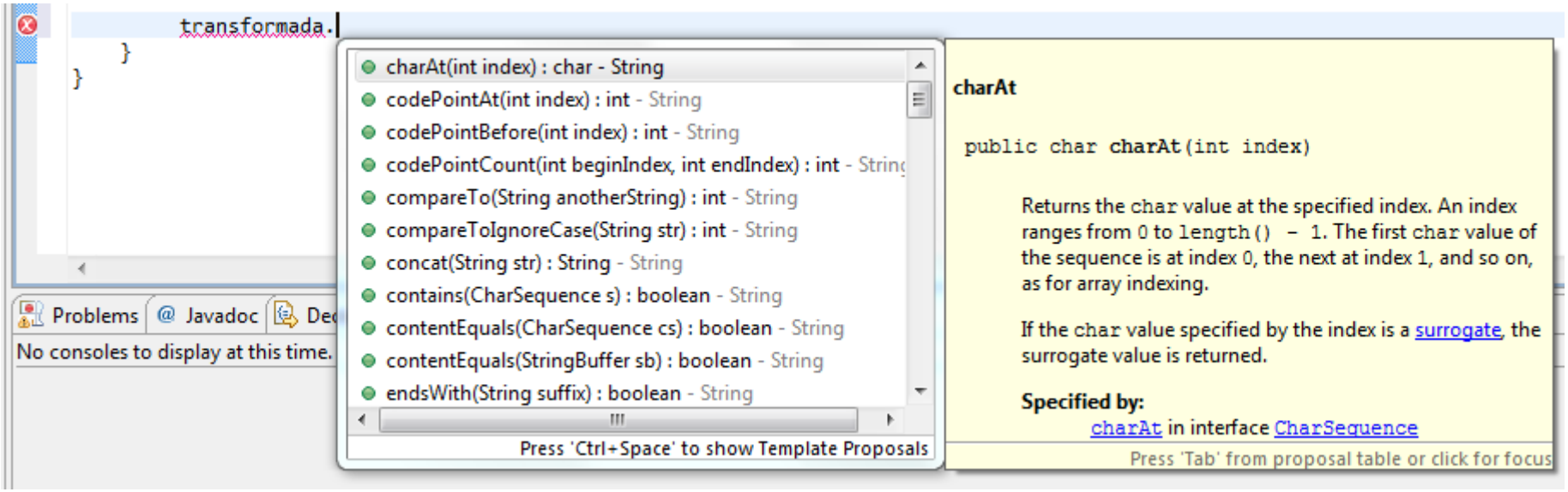


Neste momento está a ver a documentação existente para esse método. Esta janela tem a **descrição do objetivo do método** `2` , assim como dos **parâmetros que este recebe** `3` e do **tipo de retorno** `4` .

3. Autocomplete

A janela de preenchimento automático de código aparece assim que escrevemos o caratere "." a seguir a um objeto. Possui todos os métodos disponíveis para esse objeto bem como a descrição dos mesmos.

- No **fim** do método `main` da classe `ManipulacaoStrings` **escreva** `transformada.`



- Uma vez que a variável `transformada` é do tipo **String**, são apresentados todos os métodos associados.

Pode navegar neste menu e ver o objetivo de cada um deles.

- **Apague** a linha que acabou de inserir

- **Volte** à classe `TesteCarro` e **teste** esta funcionalidade com o objeto `carro1`

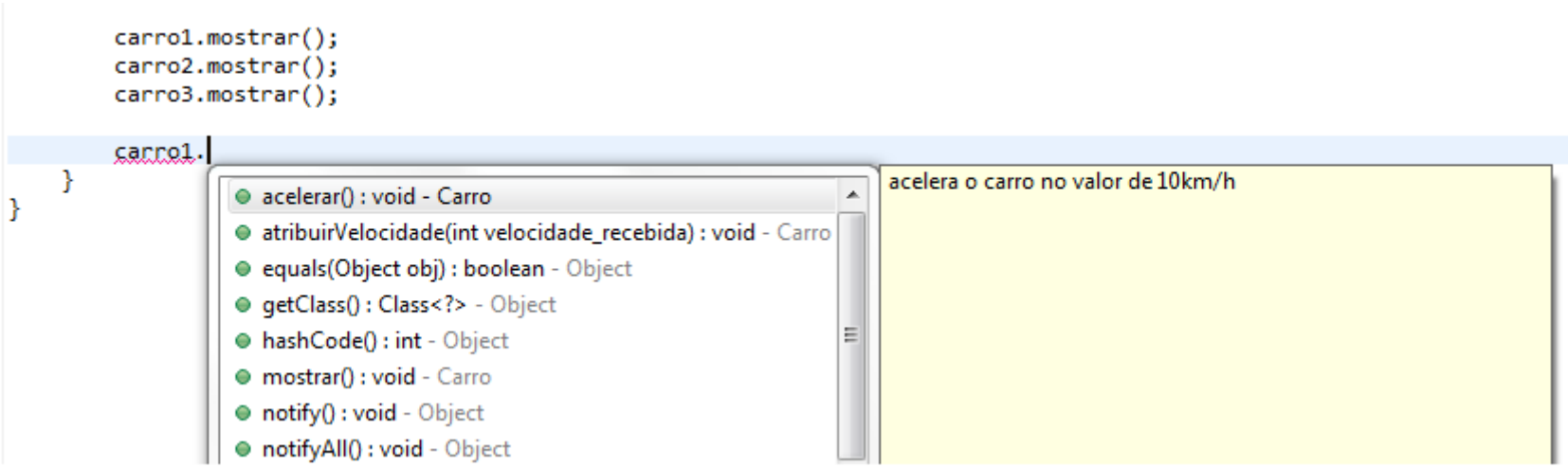
Não tem qualquer descritivo correto? Isto porque é preciso dar informação ao Java para que ele a inclua nesta funcionalidade de ajuda.

- Na classe `Carro` **adicione** as instruções destacadas:

```
/**
* acelera o carro no valor de 10km/h
*/
public void acelerar(){
    velocidade += 10;
}
```

- Agora, na classe `TesteCarro` **escreva** no final do método `main` a instrução `carro1.`

  - Repare como a informação adicional que inseriu na classe `Carro` é agora visível nesta janela de ajuda.



Com esta funcionalidade é possível visualizar a documentação de todos os nossos métodos. Isto é bastante útil pois não nos obriga a abrir o ficheiro da classe para saber o que o método faz. A documentação contempla todo o método, o que significa que é possível introduzir descritivos para os parâmetros e retornos dos métodos. Esta documentação tem um pequeno automatismo para nos ajudar a criá-la.

- **Coloque-se** na classe `Carro` e, na linha acima do método `travar`, **escreva** `/**` e, de seguida, **pressione** a tecla `ENTER`

  - Repare como o Eclipse automaticamente lhe construiu o resto do bloco de documentação.

- **Introduza** um descritivo para este método

- Agora **aplique** o mesmo para o **construtor**

  - Como vimos atrás, o bloco de documentação é criado de forma automática, mas neste caso fica com um aspeto um pouco diferente:

```java
/**
 *
 * @param matricula_carro
 * @param marca_carro
 * @param velocidade_carro
 */
public Carro(String matricula_carro , String marca_carro, int velocidade_carro){
    marca = marca_carro;
    matricula = matricula_carro;
    velocidade = velocidade_carro/CONVERSAO_METROS_SEGUNDO;
}
```

O Eclipse deteta a existência de parâmetros e coloca-os na descrição do método. Na documentação Java cada parâmetro é identificado por `@param` e o seu descritivo é incluído à frente.

Vamos então terminar a documentação deste método.

- **Reescreva** este bloco de comentários para o seguinte:

```java
/**
* Construtor de carro que recebe uma matricula, uma marca e um carro
* e devolve um objeto do tipo carro com essas caracteristas
* @param matricula_carro matricula do carro
* @param marca_carro marca do carro
* @param velocidade_carro velocidade corrente do carro
 */
```

Para a descrição do tipo de retorno usa-se `@return` seguido do descritivo do valor de retorno do método. Neste caso concreto, esse bloco não foi adicionado automaticamente pelo Eclipse pois os métodos que estamos a observar não possuem tipo de retorno. Para um método com parâmetros e tipo de retorno a documentação seria então algo semelhante ao seguinte:

```java
/**
* Descritivo do método
```

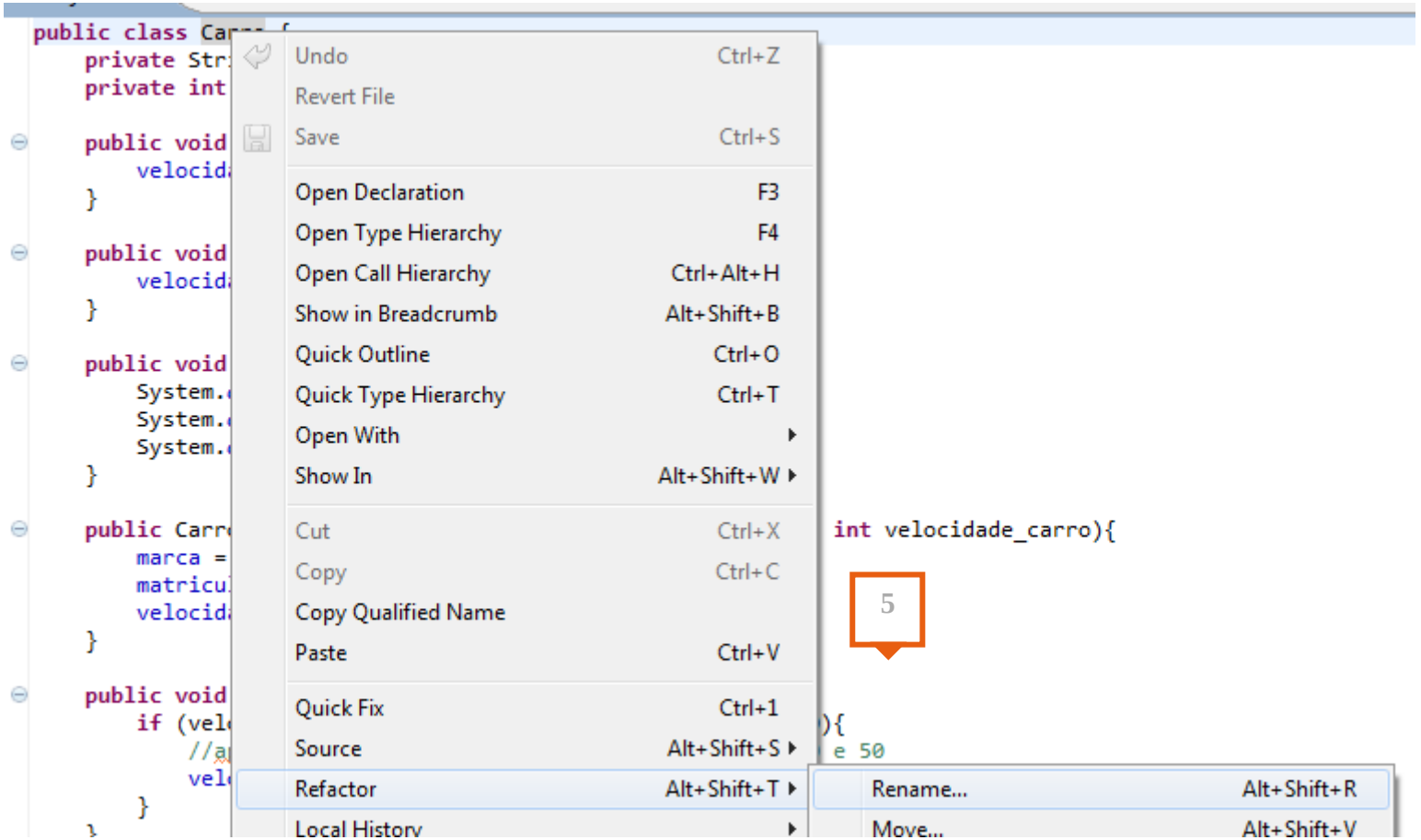
```
* @param parametro1 descritivo do parametro1
* @param parametro2 descritivo do parametro2
* @return descritivo do valor de retorno
*/
public int metodo (int parametro1, String parametro2){
```

4. Refatorização

O conceito de **refatorização** compreende a automatização de alterações no código. Um dos casos mais comuns de uso é a alteração do nome de uma classe, método ou atributo.

- **Abra** a classe Carro

- Com o cursor posicionado no **nome** da classe, **clique** no **botão direito** do rato, **escolha** a opção **Refactor** e, de seguida, **clique** em **Rename** 5



- **Altere** o nome `Carro` para `CarroExperiencia` e **pressione** a tecla `ENTER`

  - Confirme que todos os locais onde se encontrava a palavra `Carro` foram substituídos por `CarroExperiencia`.

O código foi também alterado noutras classes que usem o objeto `Carro`.

- **Confirme** que na classe `TesteCarro` o nome dos objetos foi **alterado**

> ⚠ Sempre que precisar de modificar nomes de variáveis, métodos ou nomes de classe deve utilizar esta funcionalidade. Apenas assim tem a garantia que todo o código feito até então continua correto!

- **Volte** a utilizar esta funcionalidade para **restituir** o nome original da classe

5. Step Into

No módulo de **Debug** foram abordados os botões de controlo de fluxo do programa como **Resume**, **Terminate** e **Step Over**. No entanto, ainda existe um outro botão útil: **Step Into** 6 .

Quando se executa uma linha de código que chama um determinado método, o **Step Into** continua a execução passo a passo dentro desse método, enquanto que o **Step Over** não. O **Step Over** corre o código método internamente e apenas vemos a execução linha a linha após a chamada do método.

- **Abra** a classe `TesteCarro`

- **Adicione** um **Breakpoint** na primeira linha do método `main`

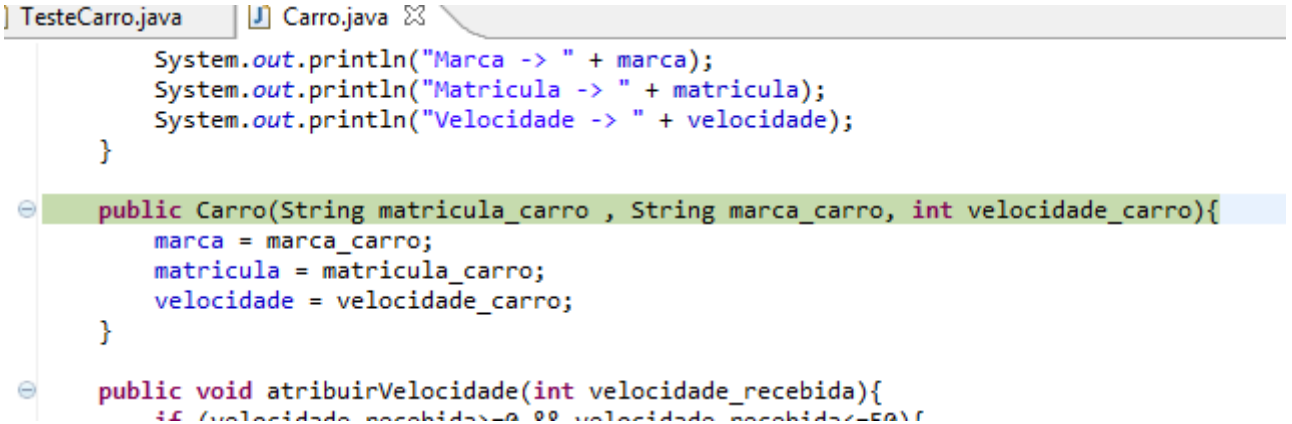
- Agora **execute** o código em modo **Debug**

- **Execute** na primeira linha um **Step Over** 7

  - Apenas vê a execução após ter sido chamado o construtor para o objeto `carro1`.

- Para a segunda linha **aplique** um **Step Into** 6

  - Repare como a execução passou para dentro do construtor de "Carro" com os parâmetros que estavam indicados no `main`.



Depois de executar o método até ao fim, o Eclipse vai retomar a execução na classe `TesteCarro`.

- **Execute** passo a passo até chegar ao **fim** do método

- **Confirme** que o código voltou à execução na classe `TesteCarro`

Com esta funcionalidade conseguimos controlar com precisão que código pretendemos ver a executar quando se trata de chamadas a métodos.

6. Package

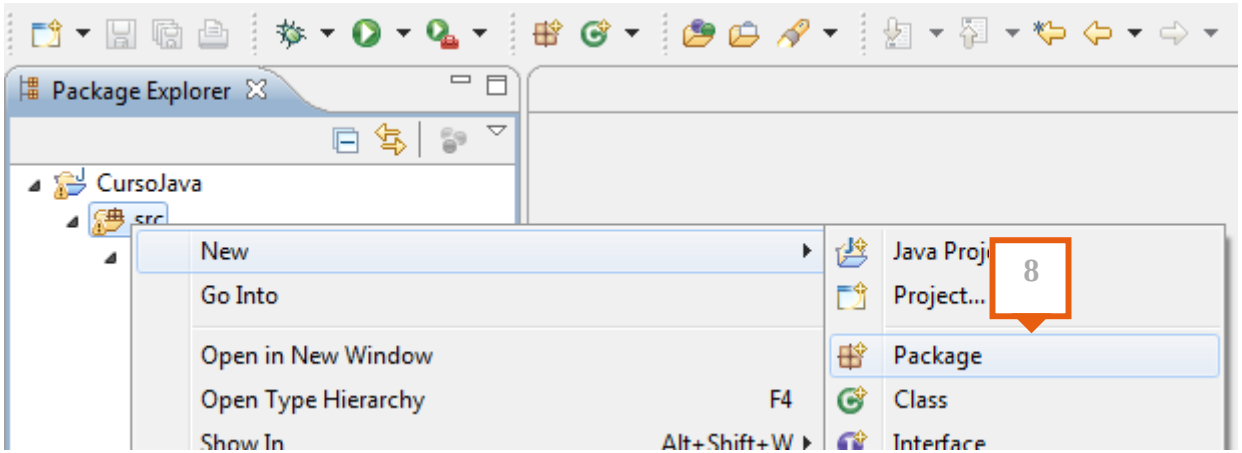
Tal como num sistema operativo existe o conceito de pasta com o objetivo de agrupar ficheiros que pertençam a uma determinada categoria, também em Java existe o conceito de **package**. Um **package** não é mais do que uma forma de agrupar classes Java segundo um determinado critério. Isto é particularmente importante em projetos de alguma dimensão, pois acrescenta alguma organização nas classes.

Analise os ficheiros (classes java) que já existem na sua pasta de aluno. São vários, correto? Agora imagine que estavam organizados por "pastas". Não seria muito mais percetível?

Vamos fazer um exemplo prático:

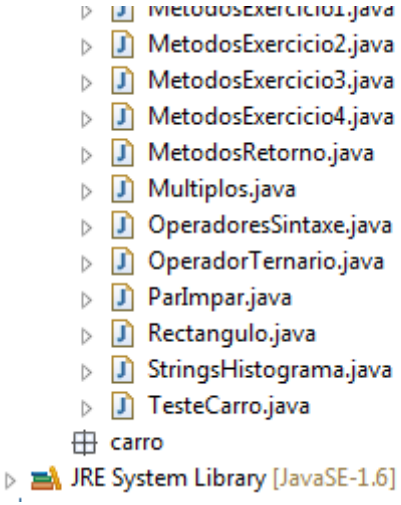
- A partir da janela **Package Explorer**, clique na pasta `CursoJava` com o **botão direito** do rato

- Agora **selecione** a opção New > **Package** 8



- Agora, na janela que aparece, **defina** como nome do package **carro**, e **clique** em **Finish**

  - Repare como é mostrado um novo bloco no **Package Explorer**.

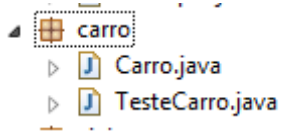


Uma vez que este package não contém ainda nenhuma classe, o Eclipse identifica-o a branco.

- **Clique** na classe `Carro` no **Package Explorer** e **arraste-a** para cima do **package** `carro`

- **Faça** o **mesmo** para a classe `TesteCarro`

  - Confirme que ficou com as duas classes dentro do **package** carro:



- **Organize** por **packages** o resto das classes que foram criadas desde o início do curso

É possível utilizar classes que pertençam a outro package. Para isto basta importar a classe que está dentro do package. A sintaxe para isto é:

```
import package.classe;
```

Para o exemplo do carro, a utilização desta classe noutros packages seria feita com a seguinte instrução:

```
import carro.Carro;
```

Note também como o Eclipse adicionou automaticamente uma linha no topo de cada classe identificando o package a que ela pertence.

⚠ Uma classe fora do package default não pode importar uma classe do package default.

Para utilizar todas as classes de um package basta usar o caratere `*`:

```
import carro.*;
```

Se reparar na linha de código que é necessária incluir para fazer leituras no teclado vai notar alguma familiaridade:

```
import java.util.Scanner;
```

De facto, até a este ponto já tínhamos vindo a utilizar packages incluídos no Java. No caso acima referido estamos a usar a classe `Scanner` que se encontra no package `util`. Existem muitos outros packages com várias classes para vários domínios, pelo que é provável que para qualquer programa que pretendamos desenvolver, já existam classes que nos dêem algum suporte.

Pode sempre consultar a documentação online do Java através do respetivo site, onde constam todos os packages disponibilizados e as suas descrições.