# Capítulo 2

JAVA

Java pode ser encontrada em aproximadamente 3 bilhões de dispositivos. Similar a famosa linguagem C# da empresa Microsoft, Java é uma linguagem de programação orientada a objetos e ambas apresentam alto potencial e curva de aprendizado confortável para os iniciantes.

Este capítulo irá mostrar o básico sobre a linguagem de programação Java, que será uma porta de entrada para os iniciantes no mundo da programação.

## Aula 1- Conceitos básicos de Java

### 1.1 - Introdução

#### 1.1.1 - Por que Java?

Java é uma linguagem de programação orientada a objetos que já está muito difundida mundo afora. Não há um programador que não tenha ouvido falar dela e de seus benefícios. Há alguns que a odeiam, pois seus aplicativos acabam sendo um pouco mais pesados que o normal. Há outros que adoram, pois Java é fácil de aprender e utilizar, existindo milhares (talvez infinitas) possibilidades de aplicações a serem feitas em Java, uma vasta comunidade que está sempre a ajudar a todos que possuírem alguma dúvida sobre a linguagem, sem contar que é a linguagem base para POO. Outros temas como Análise e Projeto Orientado a Objetos tomam sempre como base a linguagem Java.



Figura 2. – Logo do Java

Ref.: http://eadplus.com.br/eadplus/wp-content/uploads/2015/08/wjava.jpg

Introduzida ao mundo em 1991, pela empresa Sun Microsystems, tinha como idealizadores Ed Frank, Mike Sheridan, Chris Warth, James Gosling e Patrick Naughton. Por ser uma linguagem de programação orientada a objetos, Java tem como vantagens a reusabilidade, foco na representação dos dados e criação de classes, objetos e instâncias que se comunicam através de funções.

### 1.2 – Começando no Java

Há quatro plataformas principais do Java: Java SE (Standard Edition), Java EE (Enterprise Edition), Java ME (Micro Edition) e JavaFX. Java SE é usada para aplicações desktop em geral e Java EE é uma extensão para aplicações web e servidores. JavaFX é uma plataforma para multimídia (também conhecido como Flash Players) com integração ao Java. Finalmente, Java ME é usada na criação de sistemas embarcados como celulares (mais antigos) e sistemas de controles industriais.

#### 1.2.1 - Nunca programou?

Se você é um absoluto novato no extraordinário mundo da programação então há três coisas que você precisa conhecer:

1. IDE – IntegratedDevelopmentEnvironment

Primeiro, antes de sair codificando em Java, você precisa de um ambiente de desenvolvimento integrado (IDE – Integrated Development Environment). As IDEs são ambientes preparados para o desenvolvimento de aplicações. Na prática, um simples editor de texto nos permite criar programas completos, mas as IDEs possuem ferramentas e facilidades que aumentam nossa produtividade (mais códigos por segundo) em mais de dez vezes.

Existem várias opções de IDEs gratuitas disponibilizadas para download. Separamos algumas delas:

* **Netbeans** (<http://netbeans.org>) - É a IDE mais utilizada mundialmente e a mais recomendada para iniciantes em Java. É potente, rápida e pode suportar todas as plataformas Java, desde a SE até a FX.
* **Eclipse:** (<http://www.eclipse.org>) - Uma das IDEs mais conhecidas é também, como o NetBeans, indicada para iniciantes.
* **Outras:** Existem outras IDEs boas como DrJava, BlueJ e IntelliJ. Fique a vontade para testá-las e escolher qual mais lhe agrada. Neste curso os exercícios serão codificados usando a IDE NetBeans.

1. JDK – Java Development Kit

A segunda coisa que você precisa saber é que uma aplicação Java só funciona acompanhada de um kit de desenvolvimento Java (JDK – Java Development Kit) que trabalha tanto como um compilador quanto um interpretador. Basicamente o compilador traduz o código escrito por nós em uma linguagem inteligível pela máquina (linguagem binária). Já o interpretator verifica semanticamente o texto codificado, verificando erros de digitação e de sintaxe mal formada.

DICA: Quando realizar o download do JDK em seu computador, certifique-se que está usando o JDK apropriado. Se no seu computador tem um sistema operacional de 32 bits, faça download do item que corresponde ao sufixo x86. Se é um sistema operacional de 64 bits, faça download do item com sufixo x64. É altamente recomendado que se faça download da última versão do JDK que está disponível neste web site.

1. Java não é JavaScript!

Antes de seguir em frente, saiba que JavaScript não faz parte do Java e nem se parece com Java, com exceção do nome.

Uma vez que o Java é utilizado para criação de aplicações stand-alone (aplicações que executam por si só) e applets (são aplicações menores, especificamente partes de outros softwares), e JavaScript é uma linguagem de script que é integrada a páginas da web para implementar funcionalidades que não podem ser alcançadas com simples HTML e CSS.

### 1.3 – Primeiros Passos: Como escrever um programa em Java

Depois de preparar seu ambiente de desenvolvimento, abra seu NetBeans e crie um novo projeto (Arquivo > Novo Projeto) e a seguinte tela irá aparecer:

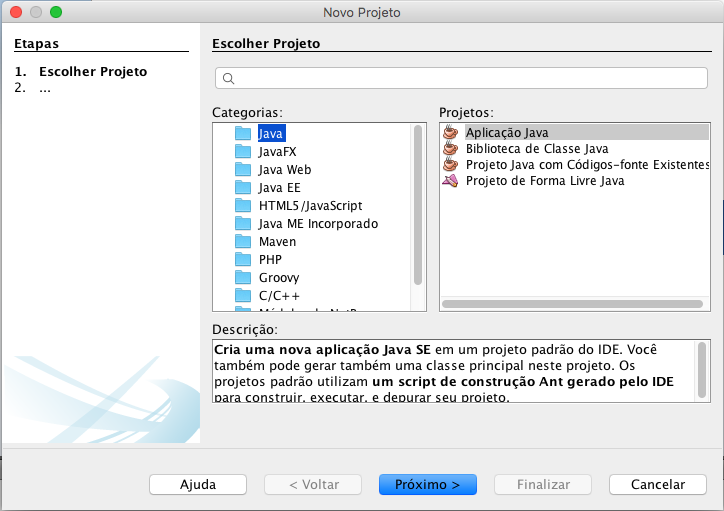


Figura 2. - Novo arquivo

Escolha **Java** em Categorias e na seção Projetos escolha **Aplicação Java.** Clique em **Próximo** e outra tela irá aparecer:

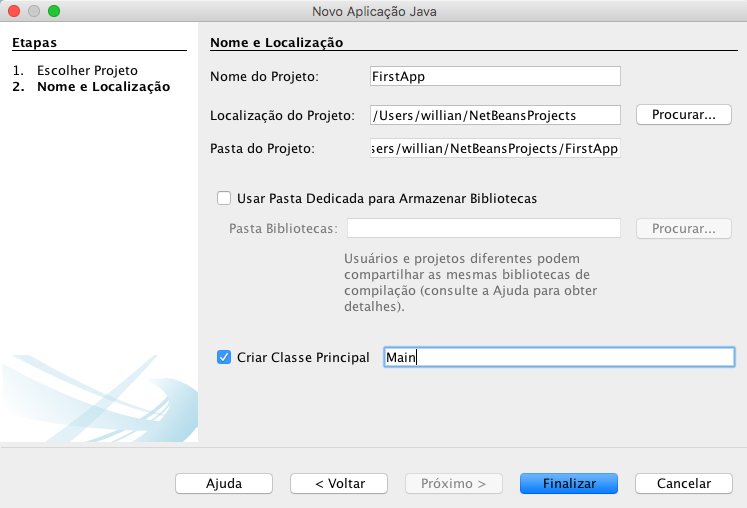


Figura 2. - Local do Arquivo

Nomeie o projeto para “FirstApp”, escolha o local em que ele será salvo e nomeie a classe principal para “Main”, depois Clique no botão **Finalizar**.

Indicamos que tenha o nome “Main” por se tratar da classe principal do programa em Java (*main* do inglês: principal).

O NetBeans ficará similar a esta tela:

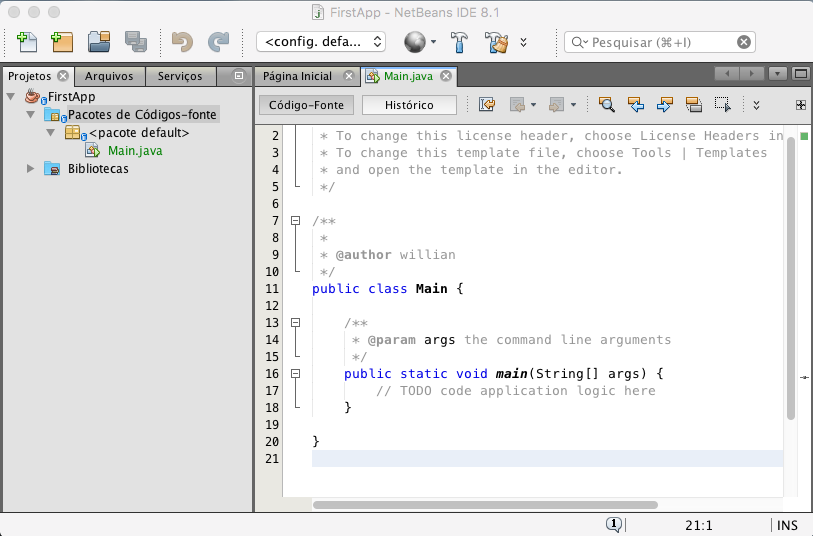


Figura 2. - Estado inicial de um projeto

Pronto. Temos nosso primeiro projeto. Por fim, execute o projeto pelo atalho F6 ou pelo botão /Users/willian/Desktop/cap02-img04.pnge veja o que acontece...

Nada! Por enquanto, se você configurou o ambiente e o projeto corretamente, a mensagem (de sucesso) “CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)” irá aparecer na parte inferior da tela. Mas vamos ao que interessa. Programar!

### 1.3 – Primeiro programa: Hello World

O primeiro programa que geralmente fazemos em qualquer nova linguagem de programação que estamos aprendendo é o Hello World. Segundo as lendas, se o Hello World não for o primeiro programa que tentamos executar ao aprender uma nova linguagem, nunca seremos bons nesta linguagem. Em Java não é diferente. Não é bom arriscar, não é? Queremos ser craques em Java.

Brincadeiras à parte, o Hello World é um programa simples que exibe uma mensagem na tela: “Hello World”. Na programação é bastante comum que o Hello World seja o primeiro programa feito quando estamos aprendendo uma nova linguagem de programação. Talvez seja uma tradição dos programadores. Mãos a obra! Localize o seguinte trecho de código no arquivo Main:

1. **public** **static** **void** main(String[] args) {
2. // TODO code application logic here
3. }

Listagem 2.1

E escreva dentro das chaves o seguinte comando:

System.out.println("Hello World");

Execute o código e veja que a mensagem “Hello World” apareceu no console (tela onde as saídas da aplicação Java irão aparecer). Pronto! Você criou seu primeiro programa Java! Acalme-se vamos explicar logo o que foi feito. Antes é importante tomar conhecimento de algumas definições.

### 1.4 – Classes, Objetos e Instâncias

A utilização de classes, objetos e instâncias é o que define uma linguagem de programação orientada a objetos. Existem seções que compõem um programa orientado a objeto, e a melhor maneira de entendê-las é compreender o relacionamento que existe entre elas.

* **Classe –** O nível mais alto neste grupo é a classe. Ela define as especificações e o comportamento de um objeto, por meio de atributos (propriedades) e métodos (ações).
* **Objeto –** Objetos aplicam as especificações definidas por uma classe, ou seja, existem a partir de um “molde” (classe). Eles são instâncias de uma classe.
* **Instância –** Por fim, as instâncias de uma classe compartilham o mesmo conjunto de propriedades, embora o conteúdo dessas propriedades seja diferente entre si.

Em programação, as classes são definições para os objetos, mas estas especificações não são carregadas na memória do computador. Ao invés disso, elas agem como se fossem meros “moldes, formas ou receitas de bolo”, prontos para serem usados caso um objeto seja instanciado sob esta classe. Então os objetos seriam o próprio *Bolo*. Eles são objetos quando não são diferenciados entre si, mas quando um bolo é diferente do outro (um é de cenoura e outro de chocolate, por exemplo) ele pode ser chamado de instância mas não deixa de ser um objeto.

Vamos a outro exemplo. Digamos que uma *Planta* é uma classe que representa os vegetais. Uma *Árvore* é um objeto da classe *Planta* e uma *Goiabeira*é uma instância do objeto *Árvore.* Ao mesmo tempo, *Goiabeira* é uma instância da classe *Planta,* que está no topo desta hierarquia.

### 1.5 – Programa Hello World (Explicação)

Definições dadas, vamos entender o funcionamento do nosso primeiro código.

1. **public** **class** Main {
3. /\*\*
4. \* @param args the command line arguments
5. \*/
6. **public** **static** **void** main(String[] args) {
7. // TODO code application logic here
8. }
10. }

Listagem 2.2

Na primeira linha o que vemos é a **definição (ou declaração) de uma classe**:

**public** **class** Main

A abertura de chave “{“ simboliza o começo de uma classe. Lembre-se sempre que cada abertura de chaves deve conter seu fechamento “}” para mostrar ao programa onde um bloco de código específico acaba. No programa Hello World, repare que a classe Main acabou depois que o resto das seções de código foram executadas. Demos o nome Main pois ela é a classe principal de um projeto Java. Ela é ponto de partida e de término da execução do programa.

Mas ainda não está claro. Ainda há mais informações nesta curta linha de código. Repare que a palavra public aparece nesta declaração de classe. Este é um exemplo claro de modificador de acesso, que será explicado com mais detalhes nos capítulos seguintes, mas em resumo eles implementam o encapsulamento ou controle para que outras classes possam acessar ou não certas informações desta classe. O modificador public significa que as informações contidas na classe podem ser acessadas livremente por qualquer outra classe. Se modificarmos o acesso da classe para private as informações da mesma poderiam ser acessadas apenas por ela mesma. Por enquanto podemos esquecer os modificadores de acesso. Os novos programas que criaremos ainda não precisarão deles e irão funcionar normalmente.

Nas próximas linhas encontramos os chamados **comentários:**

1. /\*\*
2. \* @param args the command line arguments
3. \*/

Listagem 2.3

Quando o código necessita de uma explicação mais detalhada para seu entendimento, ou quando queremos organizar o projeto com algumas notas pontuais em certos trechos de códigos temos a facilidade de utilizar os comentários. Há duas maneiras de utilizá-los:

* **Comentário de linha única –** Quando precisamos de apenas uma linha de comentário podemos utilizar barra dupla “//”. Exemplo:

//Comentário de linha única

* **Comentário de múltiplas linhas –** Quando é necessário uma verdadeira redação para explicação algo no código. Geralmente utilizado para documentação do projeto (Documentação de código não é o foco do projeto mas você pode pesquisar por ferramentas como Javadoc para melhor entendimento). Utilize “/\*” para iniciar um comentário de múltiplas linhas e “\*/” para finalizá-lo. Exemplo:

/\*

\* Muitas linhas de comentário.

\* Geralmente utilizada para documentação

\*/

Mais adiante podemos encontrar o ***método* main*,*** com a seguinte sintaxe:

**public** **static** **void** main(String[] args)

Aqui temos outras novas informações. Além do modificador de acesso public, ainda podemos identificar mais duas palavras chaves na declaração do método main*:* **static** e **void*.***

* **static:** esta palavra chave faz com que o bloco de código seguinte estará relacionado a classe e não a instância e objetos sob ela. Isto garante que o programa consiga executar livremente o método sem que haja a necessidade de se criar uma instância desta classe. Isto é crucial para o método main, já que ele é o ponto de entrada do programa.
* **void:** Este é o tipo de retorno do método. Neste caso ele não retorna nada (void). Isto permite que o programa informe o a Máquina Virtual Java (JVM – Java Virtual Machine) que o mesmo foi executado e finalizado corretamente.

Os parâmetros entre parênteses “String[] args” estão aí basicamente para suportar comandos de entrada via linha de comando. Finalmente, vamos ao último comando do programa Hello World:

System.out.println("Hello World");

Este é o método responsável pela exibição do texto “Hello World” na tela. A classe System, contida no JDK, possui um atributo out (responsável por manipular métodos de saída/exibição de dados), que por sua vez disponibiliza o método println (exibe na tela e quebra uma linha) que recebe o texto como parâmetro para exibí-lo.

### 1.6 – Resumo

Neste capítulo você aprendeu um pouco sobre o histórico do Java e viu que ele é uma das melhores opções para aprendizado de programação orientada a objetos. Aprendemos também sobre as diversas plataformas Java e sobre o JDK. Uma IDE é muito importante quando falamos de produtividade, então escolhemos a Netbeans. Finalmente criamos nosso primeiro programa em Java: o Hello World. Agora que somos programadores, vamos prosseguir no extraordinário mundo da Orientação a Objetos.

### 1.7 – Exercícios

|  |  |
| --- | --- |
| 1. | Quais são as plataformas principais do Java e para que servem? |
| 2. | O que são IDEs? Cite exemplos. |
| 3. | Qual é a importância do JDK para uma aplicação Java? |
| 4. | O que é um compilador e um interpretador? |
| 5. | Qual é diferença entre o Java e o JavaScript? |
| 6. | O que você entende por Classe, Objeto e Instância em POO? |
| 7. | Qual é a função da classe principal do Java? |
| 8. | Qual instrução utilizada para exibir um texto na tela? Use o exemplo Hello World. |
| 9. | Qual é o único método obrigatório do Java? |
| 10. | Cite com exemplos a diferença entre Classes, Objetos e Instâncias. |

### 1.8 – TDP

Vimos neste capítulo que o método main possui alguns argumentos que não foram explicados:

1. **public** **static** **void** main(String[] args) {
2. System.out.println("Hello World");
3. }

Listagem 2.4

#### 1.8 - Linha de comando

O parâmetro args é utilizado para quando executamos uma aplicação Java via linha de comando. Este parâmetro recebe uma lista de argumentos do tipo String. Estes argumentos são passados pela linha de comando ou internamente pela IDE utilizada.

#### Tarefa

Construa um programa no bloco de notas de seu computador (não utilize IDE) que possua os seguintes requisitos:

1. Primeiramente crie o programa Hello World e execute-o via linha de comando.
2. Atualize o código para exibir “Hello **seu** **nome**“, sendo que “seu nome” será um argumento enviado via linha de comando.

Agora que conseguimos executar um programa Java usando linhas de comando, vimos que, sem sombra de dúvidas, o uso de uma IDE facilita e aumenta a produtividade da programação.

Lembra de Nelson? Ele gosta de Java e provavelmente vai pedir que o projeto dos filmes seja codificado nesta linguagem.

#### Tarefa

Faça uma pesquisa sobre as melhores IDEs para Java. Escolha uma delas, faça download e instale-a no seu computador.

## Aula 2 - Programação em Java

Criamos nosso primeiro programa, mas ainda há muito o que saber. Esta aula apresentará a sintaxe fundamental da linguagem Java. A seguir temos uma listagem de itens que servirão de referência para as aulas seguintes.

### 2.1 – Palavras reservadas

Em Java, assim como em outras linguagens de programação, há algumas palavras especialmente designadas e pré-estabelecidas para uso de certos métodos e estruturas já utilizados, além algumas que veremos no decorrer do curso. Tais palavras não devem ser utilizadas para nomear suas construções Java, como as variáveis, que aprofundaremos mais a frente.

O Java possui uma lista pequena de palavras reservadas, descritas a seguir:

|  |  |  |
| --- | --- | --- |
| abstract | assert | boolean |
| break | byte | case |
| catch | char | class |
| const | continue | default |
| do | double | else |
| enum | extends | final |
| finally | float | para |
| goto | if | implements |
| import | instanceof | int |
| interface | long | native |
| new | package | private |
| protected | public | return |
| short | static | strict |
| super | switch | synchronized |
| this | throw | throws |
| transient | try | void |
| volatile | while |  |

Obs: As palavras true, false e null não são palavras reservadas. Essas palavras são conhecidas como “literais”, ou seja, são valores literais. Um número inteiro por exemplo é um literal. São como valores propriamente ditos.

Você irá perceber utilizando o NetBeans, ou qualquer outra IDE, que uma coloração pode ser atribuída a estas palavras reservadas.

### 2.2 – Comentários no código

É uma boa prática entre a comunidade programadora que o programa seja bem compreendido evisualmente despoluído. Para isso, e para que você mesmo compreenda seu próprio programa,é que se utiliza, entre outras ferramentas, o Comentário no código.

Geralmente o Comentário é utilizado para especificar o autor do código, sua utilidade e a utilidade de cada bloco nele contido, facilitando assim a interpretação do próximo programador que irá utilizar o seu código.

Para inserir um comentário, basta digitar duas barras “//” para que tudo que for escrito na linha, após as barras, se torne um comentário. Isto faz com que somente a linha onde não foi digitada a barra dupla seja um código.

1. //Comentario de uma linha
2. codigo;
4. codigo; //Comentario

Listagem 2.5

Para inserir um comentário de múltiplas linhas inicie-o com “/\*” e termine-o com “\*/”. Lembre-se sempre de terminar o seu comentário desta forma para que não haja erro de compilação.

1. /\*
2. HelloWorld
3. Este programa imprime as palavras "Hello World" na tela do usuário.
4. **@author** Aluno
5. \*/
6. codigo;

Listagem 2.6

### 2.3 – Nomeando Classes

Na linguagem Java é possível definir os nomes de suas classes. É uma boa prática que os nomes de suas classes não se iniciem com números ou caracteres especiais e se iniciem com letra maiúscula. Caso o nome da sua classe seja um nome composto, iniciar os nomes com letra maiúscula e sem separação por espaço ou caractere especial. Assim sua classe se chamaria, por exemplo, Carros ou ModelosCarros, esta convenção é conhecida como *camel case*.

Às vezes será necessário utilizar o mesmo nome de classe para expressar dois conceitos diferentes, isto pode gerar um conflito chamado *colisão de nomes*, que pode ser evitado empacotando suas classes.

Um pacote é um mecanismo de *namespace* e atua como um organizador, que torna os nomes neste pacote exclusivos dentro dele, podendo ser utilizados novamente em outros pacotes, mas não no mesmo pacote. E também, serve para você organizar seus aplicativos mais complexos em unidades de funcionalidade discretas.

### 2.4 – Definição de pacote

Para definir um pacote, utiliza-se o seguinte esquema:

1. **package** orgType.orgName.appName.compName;

Listagem 2.7

Onde:

orgType é o tipo de organização;

orgName é o nome do domínio da organização;

appName é o nome do aplicativo abreviado;

compName é o nome do componente.

Não é obrigatório seguir este modelo, é apenas uma convenção de pacote, nem tão pouco especificar um pacote, basta somente criar nomes exclusivos para cada classe num pacote padrão. Mas é uma boa escolha seguir estas convenções descritas aqui.

### 2.5 – Instruções de importação

As importações (import) se assemelham à definição de classe. Muitas das classes não estão localizadas por padrão no compilador e devem ser importadas para serem utilizadas. Uma *instrução de importação* informa o compilador Java onde estão localizadas certas classes as quais você fará uso.

Uma instrução de importação aparece da seguinte forma:

**import** ClasseASerImportada;

Desta maneira, se importa uma classe específica. Uma maneira de importar todas as classes localizadas numa mesma pasta é:

**import** java.awt.\*;

Ou seja, utilizando o asterisco (\*), você importará, neste caso, todos os subpacotes pertencentes ao pacote AWT, mas tal método pode tornar seu código menos legível por outros programadores, então procure sempre importar somente as classes específicas necessárias para seu desenvolvimento.

### 2.6 – Variáveis

Dentro de uma classe, os valores de suas variáveis distinguem e diferenciam cada instância, além de definirem o estado desta. As variáveis possuem um accessSpecifier (ou modificador de acesso), um dataType (tipo) e um variableName (nome da variável) e opcionalmente um initialValue (valor inicial).

Os accessesSpecifier’s podem ser do tipo public, protected ou private. Estes tipos definem a visibilidade de um atributo em seu pacote ou globalmente, na aplicação. O modificador public libera o acesso do atributo para a aplicação inteira, enquanto o protected permite que somente classes do mesmo pacote acessem tal atributo. Por fim o modificador private deixa o acesso do atributo restrito à classe a qual ele pertence.

Um datatype é o tipo da variável, e ela pode ser do tipo primitivo (ex: int) ou outro tipo de classe que veremos posteriormente.

O variableName, é o nome que você pode atribuir à variável e por convenção, usa-se o padrão *camel case*, mas neste caso se inicia com letra minúscula.

O initialValue serve para que você inicialize uma variável de instância ao declará-la e assim o compilador não gerará uma variável padrão, mas este assunto nos aprofundaremos mais adiante. Um exemplo de definição de classe para Carros pode ser:

1. **public** **class** Car {
3. **private** String model;
4. **private** int year;
5. **private** String color = “red”;
7. }

Listagem 2.8

Note que a última variável tem accessSpecifier private do tipo String, com o nome “color” e foi inicializada com o valor “red”.

### 2.7 – Métodos

Os métodos de uma classe definem seu comportamento. Há duas categorias de métodos: os *construtores* e os vários *outros tipos* de métodos no qual os construtores são usados apenas para criar uma instância de uma classe e os outros tipos podem ser usados para qualquer outro comportamento do aplicativo.

A combinação de elementos estruturais que definem um método é chamada de *assinatura de métodos,* que inclui elementos como: accessSpecifier, returnType, methodName, argumentList.

#### 2.7.1 - Métodos construtores

Estes métodos, utilizados para instanciar uma classe são declarados da seguinte forma:

1. accessSpecifier ClassName([argumentList]) {
2. constructorStatement(s)
3. }

Listagem 2.9

O accssesSpecifier dos construtores são semelhantes aos das variáveis, pois são ligados a atributos de privacidade dos métodos. O nome do construtor deve corresponder ao nome da classe. Se tivermos a classe Car, então nossoo método construtor também de chamará Car.

Os construtores são métodos opcionais, se estes não forem fornecidos, o compilador fará um construtor padrão, sem argumentos. Caso forneça um construtor sem argumentos (*no-arg*) o compilador não fornecerá um padrão à você.

Os argumentList são listas de argumentos (argumentType) ou parâmetros usados dentro dos construtores, delimitadas por parênteses e são semelhantes aos argumentType das variáveis. Os argumentTypes são do tipo primitivo ou outro tipo de classe assim como nas variáveis, e em uma argumentList, os argumentTypes listados são separados com vírgula e não podem ter o mesmo nome entre si.

O exemplo carro ficaria da seguinte maneira:

1. **public** **class** Car {
3. **private** String model;
4. **private** **int** year;
5. **private** String color = “red”;
7. **public** Car(String model, **int** year, String color) {
8. **this**.model = model;
9. **this**.year = year;
10. **this**.color = color;
12. }
13. }

Listagem 2.10

Neste caso podemos atribuir todos os valores de Car no momento de sua construção:

Car car = **new** Car("Up", 2014, "white" );

#### 2.7.2 - Definição de classe com um construtor

Um construtor no-arg e um construtor com uma argumentList podem ser exemplificados na Listagem :

1. **public** **class** Car {
3. **private** String model;
4. **private** **int** year;
5. **private** String color = “red”;
7. **public** Car(String model, **int** year, String color) {
8. **this**.model = model;
9. **this**.year = year;
10. **this**.color = color;
12. }
13. }

Listagem 2.11

Observe a palavra-chave “this” que é abreviação de “*thisobject*”, se referindo a um objeto deste construtor, ou seja, quando há duas variáveis com o mesmo nome, você deve usar a palavra this para se referir a uma delas. No caso, year é uma variável de classe e um parâmetro de construtor, então this.year, como está dentro do bloco está se referindo ao parâmetro year do construtor e atribuindo um valor a ele (que é um numero inteiro também chamado de “year”). Para Car ficar mais interessante, vamos utilizar mais tipos de métodos.

#### 2.7.3 - Outros métodos

Assim como o método construtor, muitos outros tipos de métodos rodam funções específicas. E diferente dos métodos construtores, os outros métodos possuem algum tipo de retorno, que veremos no decorrer do curso.

#### 2.7.4 - Métodos estáticos e de instância

Métodos de *instância* dependem do estado de uma instância de objeto específica para seus comportamentos. *Métodos estáticos* são aqueles cujos comportamentos não dependem de qualquer estado do objeto, por isso são chamados às vezes de métodos de classe eseu comportamento ocorre no nível de classe.

Por exemplo, utiliza-se a classe JDK Logger para informações de saída para o console. Para criar uma instância da classe Logger, você não instancia uma classe Logger e sim, você chama um método estático chamado getLogger(). Para chamar um método estático, usa-se o nome da classe que contém o método estático, desta maneira:

Logger l = Logger.getLogger("NewLogger");

Aqui o nome da classe é Logger e getLogger(...)é o nome do método. Assim, parachamar um método estático, é necessárioapenas o nome da classe.

### 2.8 – Tipo de dados primitivos

Você já viu brevemente os accessSpecifier e variableName que as variáveis podem ter, agora veremos alguns tipos de dataTypes que podemos atribuir às variáveis. Entre os tipos de variáveis, temos as que armazenam os valores de tipos primitivos de dado, não pode ser um objeto, apenas um valor puro. Na Tabela a seguir são mostrados os tipos de dados e seus parâmetros.

##### Tabela 1. Tipos primitivos da linguagem Java

| **Tipo** | **Tamanho** | **Valor padrão** | **Faixa de valores** |
| --- | --- | --- | --- |
| boolean | n/a | false | true ou false |
| byte | 8 bits | 0 | -128 a 127 |
| char | 16 bits | (não designado) | Faixa dos hexadecimais |
| short | 16 bits | 0 | -32768 a 32767 |
| int | 32 bits | 0 | -2147483648 a 2147483647 |
| long | 64 bits | 0 | -9223372036854775808 a 9223372036854775807 |
| float | 32 bits | 0,0 | 1,17549435.10-38 a 3,4028235.1038 |
| double | 64 bits | 0,0 | 4,9.10-24 a 1,7976931348623157.10308 |

A escolha da sua variável vai depender da precisão de seus dados. Quanto maior a faixa de valores, maior será o consumo de memória para a compilação.

### 2.9 – Strings

#### 2.9.1 - Strings

Variáveis do tipo *strings* são objetos de primeira classe do tipo String, que comparado à linguagem C, é semelhante ao tipo de dado primitivo char, mas este pode manter somente um caractere Unicode, e no String pode-se armazenar palavras ou frases como valor.

Há vários modos para instanciar um objeto String e atribuir seu valor. Dois deles são:

1. String greeting = "hello";
2. //ou
3. greeting = **new** String("hello");

Listagem 2.12

Como Strings são objetos de primeira classe na linguagem Java, você pode usar a palavra reservada new para instanciá-los.

#### - Concatenando strings

O sinal de soma (+) é uma abreviação para concatenar String, ou seja, combinar Strings, ou String com outros tipos de variáveis ou objetos.

info("Name: " + p.getName());

#### 2.9.3 - Encadeando chamadas de métodos

O encadeamento é uma prática usada com objetos imutáveis como String, na qual uma modificação sempre retorna a modificação, mas não muda o original, e então se trabalha com a modificação. Podemos associar como uma memória, que armazena o valor original e passa a usar o valor modificado para objetos imutáveis. As variáveis primitivas quando mudadas, perdem seu valor original.

### 2.10 – Operadores

##### Operadores aritméticos da linguagem Java

Utilizados para operações aritméticas e outras expressões estes podem ser: Unários, onde apenas um operador é necessário para realizar a operaçãoou Binários, onde dois operadores são necessários para realizar a operação. Estes operadores estão listados na seguinte Tabela:

##### Tabela 2. Operadores aritméticos da linguagem Java

| **Operador** | **Uso** | **Descrição** |
| --- | --- | --- |
| + | x + y | Inclui x e y |
| - | x - y | Subtrai y de x |
| - | -x | Negativa de x |
| \* | x \* y | Multiplica x e y |
| / | x / y | Divide x por y |
| % | x % y | Retorna o restante da divisão de x por y (o operador de módulo) |
| ++ | x++ | Incrementa a por 1; calcula o valor de x antes de incrementar |
| ++ | ++x | Incrementa x por 1; calcula o valor de x depois de incrementar |
| -- | x-- | Decrementa x por 1; calcula o valor de x antes do decremento de |
| -- | --x | Decrementa x por 1; calcula o valor de x após decremento de |
| += | x += y | Abreviação de x = x + y |
| -= | x -= y | Abreviação de x = x – y |
| \*= | x \*= y | Abreviação de x = x \* y |
| %= | x %= y | Abreviação de x = x % y |

#### Operadores adicionais

Além destes, outros símbolos são utilizados como operadores em Java:

* Ponto (.), que qualifica nomes de pacotes e instancia métodos.
* Parênteses (()), que delimitam uma lista de parâmetros separados por vírgula em um método.
* **new**, que (quando seguido por um nome de construtor) instancia um objeto.

Outros operadores, como os condicionais, serão vistos a seguir.

### 2.11 – Operadores condicionais e instruções de controle

Aqui aprenderemos como informar aos seus programas como eles devem atuar em condições diferentes utilizando operadores de condição e instrução como if, else, else if e também >, == e &&.

#### 2.11.1 - Operadores relacionais e condicionais

Estes operadores servem para tomadas de decisões que frequentemente se iniciam como uma expressão booleana (avaliada como true ou false). Essas expressões usam operadores relacionais e condicionais que são descritos na Tabela abaixo.

##### Tabela 3. Operadores relacionais e condicionais

| **Operador** | **Uso** | **Retorna true se...** |
| --- | --- | --- |
| > | x > y | x for maior que y |
| >= | x >= y | x é maior que ou igual a y |
| < | x < y | x é menor que y |
| <= | x <= y | x é menor que ou igual a y |
| == | x == y | x é igual a y |
| != | x != y | x não é igual a y |
| && | x && y | x e y são true, condicionalmente avalia y (se x for false, y não será avaliado) |
| || | x || y | x ou y é true; condicionalmente avalia y (se x for true, y não será avaliado) |
| ! | !x | x é false |
| & | x & y | x e y são true; sempre avalia y |
| | | x | y | x ou y é true; sempre avalia y |
| ^ | x ^ y | x e y são diferentes |

#### 2.11.2 - A instrução if

Vamos usar alguns dos operadores aprendidos:

1. **int** number = 8;
3. **if**((number %2 == 0) && (number < 10)){
4. System.out.print(number + " e par e menor que 8");
5. }

Listagem 2.13

Aqui conhecemos uma instrução nova: acondicional if (se). Esta instrução avalia as expressões booleanas entre parênteses e as executa se estas forem verdadeiras. Aqui, a variável number recebe o valor 8, então podemos interpretar o código da seguinte forma: Se o resto da divisão de 8 por dois for 0 (ou seja, se 8 é par) e 8 é menor que 10, então imprima na tela “8 é par e menor que 10). Como não há mais nenhuma instrução, a execução para por aqui.

### 2.12 – Escopo da variável

Cada variável de um aplicativo está em um *escopo*, ou um *namespace*. Se esta variável for requisitada e ela estiver fora do escopo, ocorrerá um erro de compilação. Veja um exemplo de variáveis fora do escopo:

1. **public** **class** SomeClass {
2. **private** String someClassVariable;
3. **public** **void** someMethod(String someParameter) {
4. String someLocalVariable = "Hello";
6. **if** (**true**) {
7. String someOtherLocalVariable = "Howdy";
8. }
9. someClassVariable = someParameter; // permitido
10. someLocalVariable = someClassVariable; // permitido
11. someOtherLocalVariable = someLocalVariable;// Variavel fora de escopo
12. }
13. **public** **void** someOtherMethod() {
14. someLocalVariable = "Hello there";// Variável for a de escopo
15. }
16. }

Listagem 2.14

Dentro de SomeClass, someClassVariable é acessível por todos os métodos não estáticos. Dentro de someMethod, someParameter é visível, mas fora desse método não é, e o mesmo ocorre para someLocalVariable. Dentro do bloco if, someOtherLocalVariable  é declarado e, fora desse bloco if , ele está fora do escopo. Por essa razão, dizemos que Java tem escopo de bloco, pois blocos (delimitados por { chaves }) definem os limites do escopo.

#### 2.12.1 - A instrução else

Muitas vezes, em programação, desejamos efetuar uma operação apenas se outra for falsa (ou falhar) ao ser avaliada em uma expressão condicional. O else é utilizado para que, caso a expressão em if for falsa, ele execute a expressão dentro de seu bloco. Traduzindo, else significa “senão”.

1. **int** year;
3. **if** (year <= 1990){
4. System.out.print("Carro clássico");
5. }**else** {
6. System.out.print("Carro contemporâneo");
7. }

Listagem 2.15

Ou seja, o else é parecido com o if mas ele somente irá executar se a condição if for falsa, ou seja, **se** (year for maior ou igual a 2015), {imprimir na tela “ Carro clássico”, **senão**, {imprimir na tela “Carro contemporâneo”}.

Também é possível usar else para uma verificação if adicional usando “else if”.

1. **if** (conditional) {
2. } **else** **if** (conditional2) {
3. } **else** **if** (conditional3) {
4. } **else** {
5. }

Listagem 2.16

Se a condicional for false, a condicional2 será avaliada, se essa for false, a condicional3 será avaliada e se esta também for falsa, o else será aplicado.

### 2.13 – Loops

Para que nosso código execute a mesma ação repetidas vezes, até um limite de vezes estabelecido, poderemos usar duas construções de loops: o for e o while.

#### 2.13.1 - O que é um loop?

É uma construção de programação que executa repetidamente enquanto uma condição não é atendida, como por exemplo, solicitar que um programa leia os registros de um arquivo até que chegue em um registro específico do arquivo, processando cada um deles.

#### Exemplo de um loop *for*

Um loop for é construído da seguinte forma:

1. **for**(**int** a=0, a<11, a++){
2. System.out.print(a + " ,");
3. }

Listagem 2.17

As condições para que haja a repetição do bloco for são escritas entre parênteses, logo à frente dessa palavra reservada. Tais condições são: a variável a que é inicializada com0, se a condição a<11 for true, então o bloco será executado, imprimindo um número na tela e incrementando a com 1.A variável a agora possui o valor anterior acrescentado de 1 (0+1), e passa a ter o valor 1, assim os loops são realizados até que a condição a<11 seja verdadeira, ou seja, quando a variável a receber o valor 10, o bloco será executado pela última vez e o loop terminará.

#### 2.13.2 - Loops while

Um loop while é construído da seguinte forma:

1. **while** (condição) {
2. // comandos
3. }

Listagem 2.18

Como é de se notar, o while é bem parecido com o for, mas ele continua executando o teste condicional enquanto ele for true, e por causa disso ele pode nunca ser executado se a condicional for false logo na primeira vez da execução.

Em do/while, os comandos são realizados antes da verificação condicional. Sendo assim, os comandos são verificados ao menos uma vez sempre:

1. **do**{
2. //comandos
3. } **while** (condicao);

Listagem 2.19

#### 2.13.3 - Ramificação de loop

Par resgatar um loop antes que a expressão condicional seja avaliada como false, na linguagem Java temos a instrução **break**.

1. **int** a = 0;
2. **while** (a < 3) {
3. **if** (a == 1) {
4. **break**;
5. }
6. a++;
7. }

Listagem 2.20

Aqui a instrução break encerra o loop quando a é igual a 1, ou seja, antes da condição ser avaliada como false (quando a >= 3), a instrução break encerrará o loop.

### 2.14 - Resumo

Nesta aula você aprendeu sobre itens que servirão de referência para uma Programação Java, que também serão utilizados em aulas seguintes. Aprendemos que na linguagem Java existe um conjunto de palavras reservadas, e que cada uma terá um propósito específico na linguagem. Comentários no código é tido como uma boa prática que os programadores devem adotar, para de melhorar o entendimento do programa. Você também pôde entender o que as variáveis representam dentro de uma classe e alguns tipos de dados primitivos que podemos atribuir à elas. Um dos tipos de variáveis importantes dentro da linguagem Java são as Strings. Outros itens abordados foram os Operadores, Escopo da Variável e Loops.

### 2.15 – Exercícios

|  |  |
| --- | --- |
| 1. | O que são palavras reservadas? Cite exemplos. |
| 2. | Para que servem os comentários? Quais são os tipos de comentários? Cite exemplos. |
| 3. | Como as classes devem ser declaradas/nomeadas? |
| 4. | O que são pacotes? |
| 5. | Para que serve uma instrução de importação? Mostre um exemplo de importação de uma classe específica e outro para importa de um pacote inteiro. |
| 6. | O que são as variáveis e constantes? Quando utilizar uma constante no lugar de uma variável? |
| 7. | Quando utilizar a palavra "this"? |
| 8. | O que são Strings? Como concatenar Strings? |
| 9. | Cite os tipos de operadores. |
| 10. | Quais são os modificadores de acesso? Cite as diferenças |
| 11. | Explique o funcionamento da instrução "if". Quando utilizar a instrução "else"? |
| 12. | O que são Loops? Quais tipos de loops existentes? O que é ramificação de loop? |
|  |  |

### 2.16 - TDP

#### 2.16.1 - A reunião

O dia da reunião chegou e adivinha? Você se destacou! Você havia sido o único a preparar o ambiente e a realizar uma pesquisa de mercado (sim a pesquisa sobre filmes que você fez, é chamada de pesquisa de mercado). Nelson gostou muito da pesquisa de mercado, pois é uma etapa do processo de desenvolvimento de software designada aos gerentes.

O resultado desta reunião foi que você agora é o substituto de Nelson. Ele vai tirar um período de licença médica e você deverá tocar o barco daqui para frente.

#### 2.16.2 - Estudo de caso

Nelson falou com o cliente e coletou uma série de informações para você. Leia o caso:

“O cliente é chamado **Júlio**. Júlio é um cinéfilo, crítico e colunista de um jornal famoso. Ele é simplesmente obcecado por filmes e qualquer outra arte audiovisual.

Júlio possui uma agenda, onde anota a data de lançamento de praticamente todos os filmes. Ele sempre monitora esta agenda, pois quando tais datas chegam, ele vai ao cinema, assiste o filme e publica uma crítica sobre ele em uma coluna do jornal. Faz isso religiosamente e praticamente todas as semanas ele assiste alguns filmes.

No início do ano Júlio saiu de férias e foi fazer um passeio em seu barco à vela, e como era de se esperar, não desgrudou de sua agenda.

Por um pequeno descuido, ele a deixou cair no mar. Ela afundou como uma âncora.

Júlio no desespero, abortou suas férias e tentou reescrever sua agenda, que tinha registros lançamentos de filmes até o ano 2020. Perdeu dois dias árduos nesta empreitada.

Agora, para não ter férias perdidas, resolveu contratar nossa empresa para criar um aplicativo desktop que cadastre estes filmes e os mantenham seguros”

A próxima etapa é a que chamamos de Storyboard.

#### 2.16 - Storyboard

O storyboard é uma sequencia de imagens que narra uma história de forma simples e objetiva. Em Análise de Projetos, ela é uma forma de simular o uso de um produto antes mesmo dele começar a ser desenvolvido.

Esta simulação revela se o produto realmente é viável ou não.

Dado um caso, ou problema, o storyboard deve narrar o mesmo caso, introduzindo o produto como uma solução.

#### Tarefa

Crie um storyboard que narre o acidente de Júlio mas que, com a introdução do aplicativo, prive-o da necessidade de reescrever sua agenda.

## Aula 3 –Java em Prática

Agora que sabemos o básico de Java, iremos simular o funcionamento e as características de um carro, desde sua cor e tamanho até o cálculo de consumo de combustível por litro. Esta atividade não possui nenhum atrativo visual, então você não verá um carro vermelho andando na tela, pois iremos utilizar apenas os recursos do console Java (aquele que mostrava o “Hello World” ou um monte de erro, lembra-se?). Então aperte o cinto e deixe a imaginação fluir, afinal POO é uma abstração do mundo real.

### 3.1 – Análise de Requisitos do Projeto

O desenvolvimento de um projeto de software bem estruturado começa com a fase de Análise de Requisitos. Nela devemos coletar qualquer informação que possa ser útil, inclusive os *requisitos.*

Resumidamente existem dois tipos de requisitos de software: os requisitos funcionais e os não funcionais. Os requisitos não funcionais são definidos nesta etapa, pois devem estar implícitos em qualquer projeto. Exemplos de requisitos não funcionais:

* Performance
* Usabilidade
* Manutenibilidade
* Segurança
* Etc.

Os requisitos funcionais caracterizam as funcionalidades que o software terá e geralmente são definidas pelo cliente. Exemplos de requisitos funcionais (um sistema de controle de biblioteca):

* Como usuário eu gostaria de pegar livros emprestados
* Como usuário eu gostaria de ver a data de devolução do livro
* Como usuário eu gostaria de pesquisar por livros do acervo
* Como administrador eu gostaria de aplicar multa de atraso de devolução dos livros.

Pois bem, vamos entender melhor na prática.

#### 3.1.1 - Definindo os requisitos

Imagine que estamos numa sala de reunião com o cliente. Ele irá dizer o que ele pretende com o projeto, então é a nossa hora de ouvir:

Cliente – “O intuito deste software é que ele me auxilie qual é o carro que possui consumo de combustível, para que eu possa compra-lo. Gostaria de testar diversos modelos de carro, inserindo suas características e simulando viagens para ver seu consumo de combustível e durabilidade. Mais precisamente eu gostaria que o sistema recebesse a cor, modelo, marca, ano de fabricação de um conjunto de carros. Como ações eu gostaria que os carros andassem por um tempo por mim em uma velocidade fixa de 30km/h e ao final de cada viagem o sistema perguntasse quanto foi o gasto de combustível. Por fim como resultado final eu gostaria de ver o cálculo de km/L de cada carro e comparar qual foi o mais econômico”.

Ok. Vamos interpretar. O cliente quer cadastrar um conjunto de carros e ver qual é o mais econômico entre eles. Para isso o sistema deverá permitir que o usuário cadastre alguns carros, informando sua cor, modelo, marca e ano de fabricação. Após ter cadastrado os carros, devemos colocá-los para andar numa velocidade fixa de 30km/h, mas antes o usuário informará o tempo que cada um andará. No final de cada viagem o usuário deverá informar o consumo de combustível do carro nesta viagem.

A ideia é bastante simples, pois se pensarmos logicamente, o carro mais econômico é aquele no qual o consumo de combustível foi menor em um tempo de viagem maior. Poderíamos fazer este cálculo numa planilha de Excel, mas o cliente quer um programa Java. Já que somos programadores Java, não vamos discutir.

Finalmente vamos listar os requisitos:

1. Cadastro de carros:
   1. Especificar COR, MARCA, MODELO, ANO DE FABRICAÇÃO e VELOCIDADE.
2. Viagem:
   1. Especificar o tempo em que o carro irá andar.
   2. Fazer o carro andar.
   3. Pedir o consumo de combustível após a viagem.
3. Analisar viagem:
   1. Fazer o cálculo de quilômetros por litro.
   2. Exibir o carro mais econômico dentre os cadastrados.

Finalizamos a fase de Análise de Requisitos.

DICA: “Em Análise de Projeto Orientado a Objetos a fase de análise de requisitos é bem mais ampla e complexa. Isto que fizemos foi uma mini-análise. Se você quiser saber mais sobre este assunto pesquise por Análise de Projeto Orientado a Objetos.”

### 3.2 – Mão na massa

Após termos coletados os requisitos, abra seu NetBeans e crie um novo projeto com (com – retirar) um nome de sua escolha. Neste curso o nome deste projeto será *ProjetoCarro.* Se você quiser obter exatamente os mesmos resultados apresentados nas imagens a seguir, escolha o mesmo nome. Escolha um local para salvá-lo e renomeie a classe principal para Main (também é opcional), como já visto em aulas anteriores.

#### 3.2.1 - Criando uma nova classe

Agora vamos criar a classe *Carro*. Siga os seguintes passos:

Clique em **Arquivo > Novo Arquivo** e a seguinte tela aparecerá:

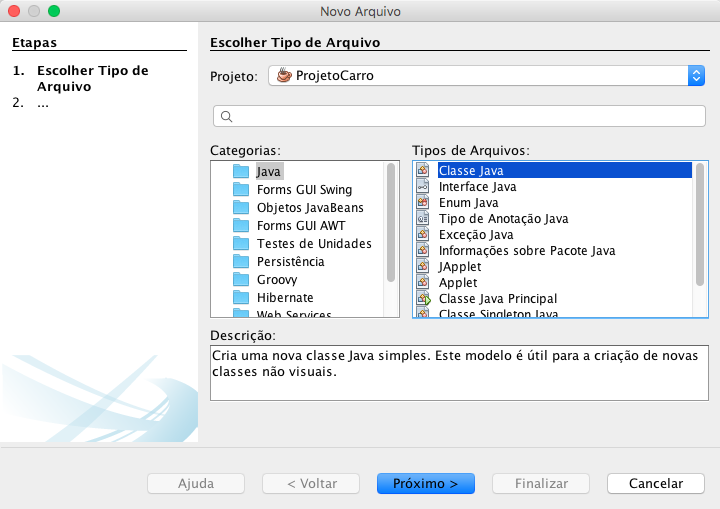


Figura 2. - Novo Arquivo

Escolha a opção **Java** em Categorias e em seguida **Classe Java** em Tipos de Arquivos. Clique em **Próximo**.

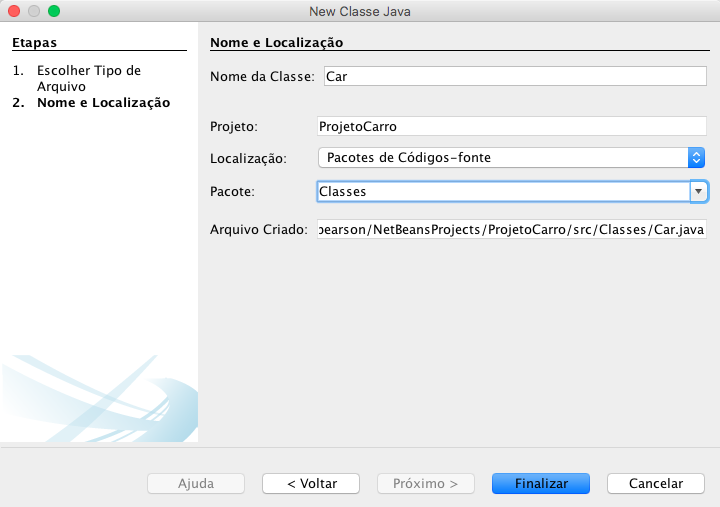


Figura 2. – Nome e localização do arquivo

Uma boa prática de programaçãoé utilizar a nomenclatura de classes, métodos e variáveis em inglês, então no campo **Nome da Classe** insira o nome *Car.* Localize o campo **Pacote** e crie o pacote *Classes,* pois é altamente recomendado que não criemos arquivos.java (a extensão do arquivo onde criaremos nossa classe) no *pacote default.* O resultado final será uma tela similar a esta:

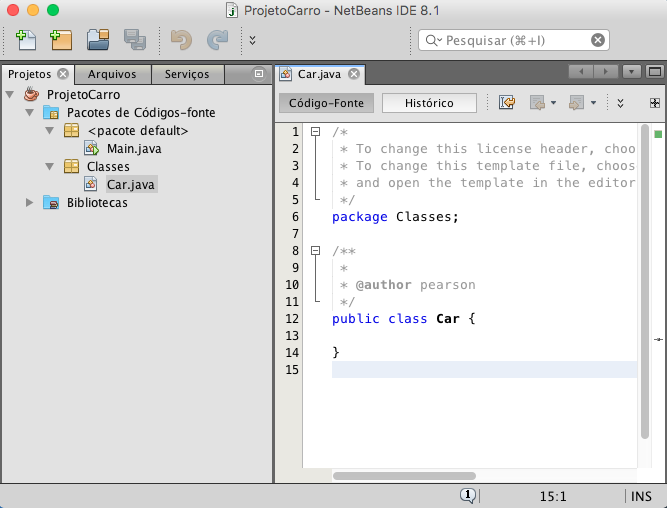


Figura 2. - Estado inicial do projeto

Repare que apareceu a linha:

**package** Classes;

Esta linha é usada para definição de namespaces, ou seja, organizar um conjunto de itens que tem relação ao nome deste pacote. Neste caso, todas as classes serão inseridas no pacote Classes.

#### 3.2.2 - Criando os atributos

Uma das características de uma classe é a definição de atributos. Os atributos são propriedades ou características que um objeto deve ter. A classe não define estas características, ela apenas dita quais são os tipos de características que um objeto/instância pode ter.

No ProjetoCarro podemos identificar algumas características que um carro deve ter:

* Cor
* Marca
* Modelo
* Ano
* Velocidade

Estas são as características que conseguimos identificar na análise de requisitos. O próximo passo será inserir estes atributos na classe Car.

Os atributos deverão possuir os seguintes tipos:

* Cor - String
* Marca - String
* Modelo - String
* Ano - Integer
* Velocidade - Float

Agora feche o livro, tente declarar estes atributos e volte quando for finalizado.

Conseguiram? Para que possamos adicionar atributos à classe *Car*, basta especificar o tipo e o identificador (nome único a ser escolhido) do atributo, lembrando que o ponto-e-virgula (;) é obrigatório no final de cada comando. Um atributo também pode ser chamado de variável. Então vamos lá.

Dentro do bloco que corresponde à classe *Car*, insira os seguintes atributos:

1. // Atributos
2. String color;               // Cor
3. String model;               // Marca
4. String brand;               // Modelo
5. Integer manufacturingYear;  // Ano de fabricação
6. Float speed;                // Velocidade

Listagem 2.21

O arquivo completo ficará assim:

1. **package** Classes;
3. /\*\*
4. \* Classe Carro
5. \* @author pearson
6. \*/
7. **public** **class** Car {
9. // Atributos
10. String color;               // Cor
11. String model;               // Marca
12. String brand;               // Modelo
13. Integer manufacturingYear;  // Ano de fabricação
14. Float speed;                // Velocidade
16. }

Listagem 2.22

Os comentários descritos são opcionais. Foram colocados apenas para maior entendimento do código.

#### 3.2.3 - Testando os atributos da classe Car

Se executarmos o programa, nada irá acontecer. Isto porque não há nada no método *main* da classe *Main.* Como discutido anteriormente, o método *main* é o ponto de partida de um programa em Java.

Localize o método *main* e instancie um objeto Car em uma variável do tipo Car. O código será o seguinte:

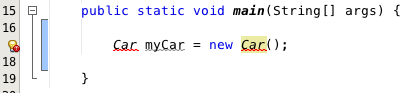


Figura 2. - Instanciando um objeto Car

Parece que nosso código está com erro. Antes de mais nada, precisamos importar a classe Car para a classe Main. Isto porque ambas estão em pacotes diferentes, então a classe Main não consegue identificar de onde vem a classe Car. Use o seguinte comando antes da declaração da classe main:

**import** Classes.Car;

O arquivo completo ficará assim:

1. **import** Classes.Car;
2. /\*\*
3. \*
4. \* @author pearson
5. \*/
6. **public** **class** Main {
8. /\*\*
9. \* @param args the command line arguments
10. \*/
11. **public** **static** **void** main(String[] args) {
13. Car myCar = **new** Car();
15. }
17. }

Listagem 2.23

Vamos examinar a instanciação do objeto carro na variável myCar:

Car myCar = **new** Car();

Primeiro estamos declarando a variável myCar com tipo Car. Depois podemos observar a presença da palavra reservada ***new.*** Esta palavra aloca um espaço de memória do computador para uma instância da classe Car. Para acessar este espaço de memória e modificá-lo, basta utilizar a variável myCar. O símbolo “=” atribui à variável myCar uma instância/objeto da classe Car.

Agora vamos dar as características ao objeto myCar. Escreva o seguinte código abaixo da declaração de myCar:

myCar.color = "Branco";

Repare que um erro aconteceu. Uma das facilidades de se usar o NetBeans é que ele aponta os erros de programação. Neste caso ele diz: “color is not public in Car; cannot be accessed from outside package”, que significa que o atributo *color* não pode ser acessado por uma classe fora do pacote Classes, isto porque não deixamos nenhum atributo com o modificador de acesso *public.* Por padrão o modificador de acesso dos atributos é o ***protected****,* que permite que apenas classes do mesmo pacote ou classes que herdam (veremos herança mais para frente) a classe onde o atributo em questão está declarado acessem esta propriedade. Como Car está em um pacote diferente da classe Main, deixe todos os atributos da classe Car com o modificador de acesso *public* (por enquanto). A classe Car ficará assim:

1. **package** Classes;
3. /\*\*
4. \* Classe Carro
5. \* @author pearson
6. \*/
7. **public** **class** Car {
9. // Atributos
10. **public** String color;               // Cor
11. **public** String model;               // Marca
12. **public** String brand;               // Modelo
13. **public** Integer manufacturingYear;  // Ano de fabricação
14. **public** Float speed;                // Velocidade
16. }

Listagem 2.24

Veja que não há mais erros na classe Main, então preencha todos os atributos que faltam. Veja o exemplo:

1. **public** **static** **void** main(String[] args) {
3. // Instanciando um novo carro
4. Car myCar = **new** Car();
6. // Dando características ao myCar
7. myCar.color = "Branco";
8. myCar.model = "Gol";
9. myCar.brand = "Chevrolet";
10. myCar.manufacturingYear = 2014;
11. myCar.speed = 30.0f;
13. }

Listagem 2.25

Estamos quase lá. Agora vamos mostraro carro no console usando uma técnica sofisticada. Entre na classe Car e abaixo da declaração dos atributos nós vamos sobrescrever o método *toString()*. Toda classe Java herda da classe *Object,* e a classe *Object* possui um método chamado *toString()*que é responsável por converter um objeto para um texto que o represente. Pode parecer complicado agora este conceito de herança, mas iremos explicar a fundo no capítulo seguinte. Estamos utilizando o método *toString()*apenas para facilitar a saída de dados e não precisar escrever tantos *System.ou.println().*

O intuito do toString() é retornar uma String mais amigável. A String retornada deverá ser a seguinte: “Gol Branco da marca Chevrolet, ano 2014” (ignore a velocidade). A classe Car ficará assim:

1. **public** **class** Car {
3. // Atributos
4. **public** String color;               // Cor
5. **public** String model;               // Marca
6. **public** String brand;               // Modelo
7. **public** Integer manufacturingYear;  // Ano de fabricação
8. **public** Float speed;                // Velocidade
10. @Override
11. **public** String toString() {
12. **return** model + " " + color + " da marca " + brand + ", ano " + manufacturingYear;
13. }
15. }

Listagem 2.26

Agora basta exibir o conteúdo do objeto *myCar* adicionado o seguinte comando no final do método *main:*

System.out.println(myCar);

Execute o programa e veja o resultado. Se uma String com as características do carro apareceu no console, deu tudo certo. Vamos para a próxima etapa.

### 3.3 – Resumo

Nesta aula, implementamos um projeto prático onde pudemos simular o funcionamento e as características de um carro. Você pôde ver a importância da fase de Análise de Requisitos no início do desenvolvimento de um projeto de software. É a partir daí que inicia a fase de desenvolvimento. Aprendemos como os atributos devem ser aplicados a uma classe e também foi abordado o método toString().

### 3.4 – Exercícios

|  |  |
| --- | --- |
| 1. | O que você entende por requisitos funcionais e não funcionais? Cite exemplos. |
| 2. | O que são atributos? Cite mais dois sinônimos de atributos. |
| 3. | O que é necessário para adicionar um atributo a uma classe? |
| 4. | Para que serve o método toString(), de onde ele vem? Como é a sua implementação? |
| 5. | Como acessamos e atribuímos valor a um atributo de um objeto. |

### 3.5 – TDP

#### Fase de análise de requisitos

Depois de escutarmos o caso do cliente, é preciso formalizar os requisitos. Os requisitos são funcionalidades que o software deverá ter.

Após diversas reuniões com o cliente, o gerente de comercial formalizou os seguintes requisitos que entrarão no contrato:

#### Projeto Filmes

O projeto filmes se trata de uma plataforma de cadastro e consulta de filmes e seriados. Detalhes como sinopse, data de lançamento e elenco estarão contidos no sistema. Os requisitos são os seguintes:

##### Requisitos não funcionais

1. O sistema deverá possuir uma interface gráfica
2. O sistema deverá armazenar os dados em arquivo
3. O sistema deverá funcionar totalmente offline
4. O sistema deverá fazer tratamento de erros, mostrando mensagens de alerta caso o usuário insira algum dado inválido.

##### Requisitos funcionais

1. O sistema deverá cadastrar filmes
   1. Um filme terá como conteudo:
      1. Nome
      2. Nome original
      3. Data de lançamento
      4. Gênero
      5. Duração
      6. Sinopse
      7. Produtoras
      8. Países de origem
      9. Orçamento
      10. Receita
2. O sistema deverá cadastrar Gêneros
   1. O gênero deverá ser utilizado no cadastro e edição de filmes
   2. Um gênero terá como conteúdo
      1. Nome
      2. Identificador
3. O sistema deverá cadastrar os Créditos do Filme
   1. O crédito de um filme terá como conteúdo:
      1. Equipe
         1. Uma equipe é formada pelas pessoas que estiveram nos bastidores do filme.
         2. Um integrante de equipe terá como conteúdo:
            1. Um nome
            2. Um cargo ou posição
      2. Elenco
         1. Um elenco é formado por atores do filme.
         2. Um ator terá como conteúdo:
            1. Um nome
            2. Nome do personagem
4. Os filmes deverão ser editáveis
5. Os filmes deverão ser deletáveis

Este é o documento inicial de requisitos. Você deve sentir-se a vontade em adicionar mais requisitos ao projeto. A próxima fase é localizar e definir as entidades que possivelmente virarão classes.

#### Tarefa

Localize e liste todas as possíveis classes do projeto.

## Aula 4 - Métodos e Encapsulamento

Nossa classe Car já possui atributos que irão dar as características de seus objetos, mas ainda precisamos adicionar certas ações aos nossos carros. Reveja a aula anterior e refresque os requisitos do projeto na sua cabeça.

No ProjetoCarro podemos identificar algumas ações que um carro deve realizar:

* Andar
* Parar
* Receber informação sobre o consumo quando o carro termina a viagem
* Calcular a quantidade de quilômetros por litro que o carro faz.

### 4.1 – Métodos

Vamos criar o método Andar. Para seguir a convenção, nomearemos o método em inglês. Na análise de requisitos definimos que o usuário deverá informar o tempo em que o carro deverá andar. Em Java o método ficaria da seguinte maneira:

1. **public** **void** drive(Integer timeInMinutes){
2. System.out.println("O "+ model + " " + color + " está andando.");
3. }

Listagem 2.27

O método drive realiza a ação Andar, nesta etapa deveremos usar a imaginação, pois não veremos um carro andar na tela, apenas será exibido um texto informando que o carro está andando. Como parâmetro, drive recebe o tempo em minutos na variável timeInMinutes do tipo Integer. Devemos utilizá-la futuramente, mas quando o método drive terminar, a variável timeInMinutes será perdida, pois ela só existirá no bloco referente ao método drive. Chamamos isto de escopo local.

Um escopo local só existe dentro de um determinado bloco de código. Uma variável criada dentro de um método ou um parâmetro recebido pelo método serão descartados após a execução do mesmo. Então como iremos manter salvo o valor da variável timeInMinutes? Criando mais um atributo na classe Car para armazenar este valor.

Crie um atributo chamado timeInMinutes de tipo Integer com modificador de acesso *private* (escolhemos *private* pois nenhuma outra classe precisará acessar este atributo)e no final do método drive insira a seguinte linha de código:

**this**.timeInMinutes = timeInMinutes;

A classe Car ficará da seguinte maneira:

1. **package** Classes;
3. /\*\*
4. \* Classe Carro
5. \* @author pearson
6. \*/
7. **public** **class** Car {
9. // Atributos
10. **public** String color;               // Cor
11. **public** String model;               // Marca
12. **public** String brand;               // Modelo
13. **public** Integer manufacturingYear;  // Ano de fabricação
14. **public** Float speed;                // Velocidade
15. **private** Integer timeInMinutes;
17. // Métodos
18. @Override
19. **public** String toString() {
20. **return** model + " " + color + " da marca " + brand + ", ano " + manufacturingYear;
21. }
23. /\*\*
24. \* Método Andar
25. \* @param timeInMinutes - Inteiro que representa os minutos que o carro andará
26. \*/
27. **public** **void** drive(Integer timeInMinutes){
28. System.out.println("O "+ model + " " + color + " está andando.");
29. **this**.timeInMinutes = timeInMinutes;
30. }
32. }

Listagem 2.28

No método drive, podemos ver a seguinte linha de comando:

**this**.timeInMinutes = timeInMinutes;

O que está acontecendo é uma atribuição de valor na variável timeInMinutes que está vindo como parâmetro do método no atributo da classe timeInMinutes. O uso do *this* é para diferenciar qual variável estamos acessando, sendo que this explicita o atributo da classe. Esta linha de código pode ser lida da seguinte forma: “A variável timeInMinutes DESTA classe recebe o valor de timeInMinutos que veio via parâmetro”.

Entre no método main e insira a seguinte linha de código para que possamos testar:

myCar.drive(25); // Estamos informando que o carro irá andar por 25 minutos

A mensagem “O Gol Branco está andando.”. Agora vamos ao próximo passo.

### 4.2 –Encapsulamento

Antes de implementar os próximos métodos é necessário encapsular os atributos e criar um construtor para a classe Car.

* Encapsulamento: Tem a utilidade de isolar partes do programa para que o acesso a estas partes fique, de forma controlada, limitada. A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações. No nosso projeto já temos dois atributos que são de uso exclusivo da classe Car. Iremos encapsular todos os atributos como boa prática.
* Construtores: Criar construtores é uma maneira mais fácil de dar as características ao objeto. Um construtor permite que seja passado via parâmetro todas as informações básicas que o objeto deve conter, no momento em que ele é instanciado.

Vamos primeiro criar um construtor da nossa classe. O intuito é que não precisemos escrever as seguintes linhas para todo novo carro que for instanciado:

1. myCar.color = "Branco";
2. myCar.model = "Gol";
3. myCar.brand = "Chevrolet";
4. myCar.manufacturingYear = 2014;
5. myCar.speed = 30.0f;

Listagem 2.29

Entre na classe Car e clique com o botão direito do mouse em uma linha após a declaração dos atributos. Confira na seguinte imagem:

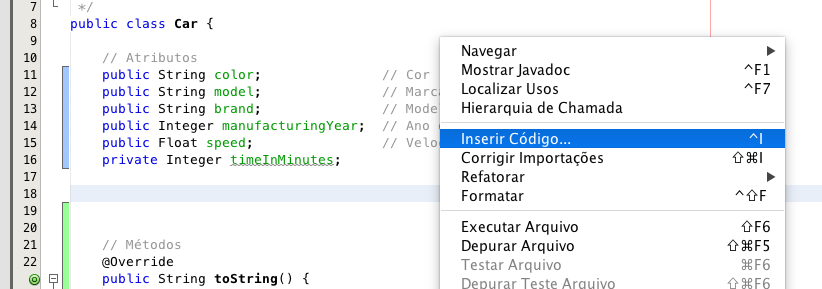


Figura 2. - Atalho inserir código

Clique em **Inserir Código...**e logo em seguida clique em **Construtor...**

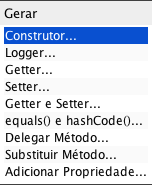


Figura 2. - Atalho inserir construtor

A seguinte tela aparecerá:

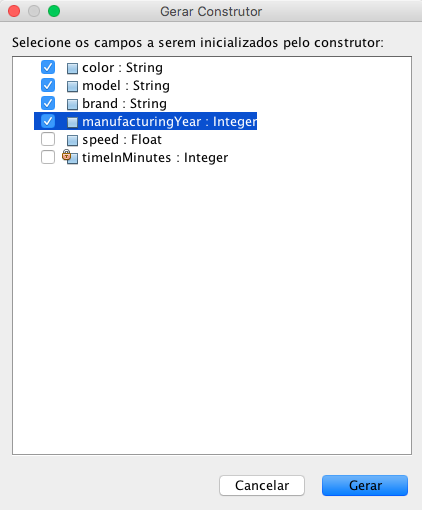


Figura 2.11 - Tela Gerar Construtor

Selecione todos os atributos, menos speed e timeInMinutes, pois estes não serão manipulados fora da classe. Clique em gerar. O novo código ficará como mostrado abaixo:

1. // Atributos
2. **private** String color;               // Cor
3. **private** String model;               // Marca
4. **private** String brand;               // Modelo
5. **private** Integer manufacturingYear;  // Ano de fabricaÃ§Ã£o
6. **private** Float speed;                // Velocidade
7. **private** Integer timeInMinutes;
9. **public** Car(String color, String model, String brand, Integer manufacturingYear) {
10. **this**.color = color;
11. **this**.model = model;
12. **this**.brand = brand;
13. **this**.manufacturingYear = manufacturingYear;
14. }

Listagem 2.30

Repare que o método main apontará erros. Apague tudo que está dentro do método main e substitua pelo seguinte código:

1. **public** **static** **void** main(String[] args) {
3. // Instanciando um novo carro
4. Car myCar = **new** Car("Branco", "Gol", "Chevrolet", 2014);
6. System.out.println(myCar);
8. myCar.drive(25);
10. }

Listagem 2.31

Execute e veja que obtivemos o mesmo resultado, com menos linhas de código no método main.

Um construtor permite que passemos informações por parâmetro no momento da criação de uma nova instância. Qualquer tratamento especial que o objeto tiver que se submeter antes de ser instanciado pode ser feito no método construtor. Uma classe pode ter vários métodos construtores, devido ao que chamamos de Polimorfismo (veremos nos capítulos seguintes).

### 4.3 – Obtendo entrada do usuário

Voltando aos requisitos da Aula 1.3. Quando o carro parar deveremos informar o consumo de combustível naquela viagem. Vamos criar o método parar (stop).

O método stop irá pedir para o usuário informar o consumo de combustível da viagem. Crie mais um atributo privado chamado consumption do tipo Float. Nele será armazenado a relação km/l.

**private** Float consumption;

Por fim, insira o método stop, faça o cálculo de consumo de combustível do carro e exiba-o:

Ao utilizar o Scanner não se esqueça de importar a classe com o comando:

importjava.util.Scanner;

1. **public** **void** stop() {
2. // Informa que o carro parou
3. System.out.println("O "+ model + " " + color + " parou.");
5. // Instancia um Scanner que receberá a entrada
6. Scanner scanner = **new** Scanner(System.in);
7. System.out.print("Digite o quanto foi gasto (em litros): ");
9. // Pergunta quantos litros foram gastos
10. Float litersSpent = scanner.nextFloat();
12. // Cálculo de consumo
13. speed = 30.0f; // Assumindo que a velocidade é de 30km/h
14. Float hoursDriving = (**float**)timeInMinutes/60.0f; // Converte minutos para horas
15. Float distanceDrove = speed \* hoursDriving; // Calcula a distância que o carro andou
16. consumption = distanceDrove/litersSpent; // Calcula a relação Km/L
18. // Mostra o consumo na tela
19. System.out.println("O "+ model + " " + color + " faz "+ consumption +" Km/L");
21. }

Listagem 2.32

A explicação do código está nos comentários.

Adicione a chamada do método stop na main e execute o código.

Vamos também alterar o método drive para que o usuário possa especificar o tempo que o carro deve andar.

O novo método drive será:

1. **public** **void** drive(){
2. System.out.println("O "+ model + " " + color + " está andando.");
4. // Instancia um Scanner
5. Scanner scanner = **new** Scanner(System.in);
6. System.out.print("Digite o tempo que o carro andará: ");
8. // Pergunta quanto tempo o carro deverá andar no teste
9. timeInMinutes = scanner.nextInt();
11. }

Listagem 2.33

A classe Main ficará da seguinte maneira:

1. **import** Classes.Car;
2. /\*\*
3. \*
4. \* @author pearson
5. \*/
6. **public** **class** Main {
8. /\*\*
9. \* @param args the command line arguments
10. \*/
11. **public** **static** **void** main(String[] args) {
13. // Instanciando um novo carro
14. Car myCar = **new** Car("Branco", "Gol", "Chevrolet", 2014);
16. System.out.println(myCar);
18. myCar.drive();
19. myCar.stop();
21. }
23. }

Listagem 2.33

E a classe Car ficará assim:

1. **package** Classes;
3. **import** java.util.Scanner;
4. /\*\*
5. \* Classe Carro
6. \* @author pearson
7. \*/
8. **public** **class** Car {
10. // Atributos
11. **private** String color;               // Cor
12. **private** String model;               // Marca
13. **private** String brand;               // Modelo
14. **private** Integer manufacturingYear;  // Ano de fabricação
15. **private** Float speed;                // Velocidade
16. **private** Integer timeInMinutes;      // Tempo em minutos
17. **private** Float consumption;          // Quilômetros por litro
19. **public** Car(String color, String model, String brand, Integer manufacturingYear) {
20. **this**.color = color;
21. **this**.model = model;
22. **this**.brand = brand;
23. **this**.manufacturingYear = manufacturingYear;
24. }
26. // Métodos
27. @Override
28. **public** String toString() {
29. **return** model + " " + color + " da marca " + brand + ", ano " + manufacturingYear;
30. }
32. **public** **void** drive(){
33. System.out.println("O "+ model + " " + color + " está andando.");
35. // Instancia um Scanner
36. Scanner scanner = **new** Scanner(System.in);
37. System.out.print("Digite o tempo que o carro andará: ");
39. // Pergunta quanto tempo o carro deverá andar no teste
40. timeInMinutes = scanner.nextInt();
42. }
44. **public** **void** stop() {
45. // Informa que o carro parou
46. System.out.println("O "+ model + " " + color + " parou.");
48. // Instancia um Scanner que receberá a entrada
49. Scanner scanner = **new** Scanner(System.in);
50. System.out.print("Digite o quanto foi gasto (em litros): ");
52. // Pergunta quantos litros foram gastos
53. Float litersSpent = scanner.nextFloat();
55. // Cálculo de consumo
56. speed = 30.0f; // Assumindo que a velocidade é de 30km/h
57. Float hoursDriving = (**float**)timeInMinutes/60.0f; // Converte minutos para horas
58. Float distanceDrove = speed \* hoursDriving; // Calcula a distância que o carro andou
59. consumption = distanceDrove/litersSpent; // Calcula a relação Km/L
61. // Mostra o consumo na tela
62. System.out.println("O "+ model + " " + color + " faz "+ consumption +" Km/L");
64. }
66. }

Listagem 2.34

Para finalizar este projeto do carro faça um programa que utilize a classe Car. O programa deve receber via entrada de usuário todas as características de um carro e armazenar em um objeto carro. Armazene quantos carros forem necessários, deixando no poder do usuário a hora de parar de cadastrar carros. Ao terminar o cadastro faça com que todos os carros ande (o usuário deverá fornecer o tempo de cada carro) e logo em seguida faça com que todos os carros pare (o usuário deverá fornecer o consumo em litros de cada carro). Após este processamento exiba o carro considerado mais econômico dentre os cadastrados pelo usuário.

### 4.4 – Resumo

Nesta aula adicionamos ações à classe, baseados nos conceitos de Métodos. Aprendemos a encapsular atributos e a criar um construtor para a Classe. Também vimos como instanciar um novo objeto e como o programa recebe a entrada do usuário.

### 4.5 – Exercícios

|  |  |
| --- | --- |
| 1. | O que são métodos? |
| 2. | Quais são as categorias de métodos? |
| 3. | Quais são os elementos de uma assinatura de método. |
| 4. | O que é encapsulamento de atributos? Para que serve? |
| 5. | Qual o intuito da criação de construtores para uma classe? |
| 6. | Como instanciar um novo objeto? Utilize o exemplo do Carro. |
| 7. | Qual é a instrução utilizada para receber a entrada do usuário? Mostre um exemplo. |
| 8. | Para finalizar o projeto faça um programa que utilize a classe Car. O programa deve receber via entrada de usuário todas as características de um carro e armazenar em um objeto carro. Armazene quantos carros forem necessários, deixando no poder do usuário a hora de parar de cadastrar carros. Ao terminar o cadastro faça com que todos os carros ande (o usuário deverá fornecer o tempo de cada carro) e logo em seguida faça com que todos os carros pare (o usuário deverá fornecer o consumo em litros de cada carro). Após este processamento exiba o carro considerado mais econômico dentre os cadastrados pelo usuário. |

### 4.6 – TDP

#### 4.6.1 - Identificando métodos e atributos

Você está se saindo muito bem. O seu time de desenvolvimento possui um arquiteto de software. O arquiteto é responsável por modelar e organizar tecnicamente o projeto, além de tentar converter a linguagem dos requisitos do sistema para uma linguagem mais técnica.

João, seu arquiteto está com dificuldades em identificar os atributos e métodos das classes do projeto.

#### Tarefa

Ajude João. Identifique todos os atributos e métodos das classes. Não é necessário implementar, apenas liste.