**Orientação a objetos no Java**

Programas grandes são difíceis de manter – por isso, é um bom hábito separá-los em unidades mais ou menos isoladas. **Em Java, isso é feito utilizando objetos, que são compostos por atributos e métodos definidos a partir de classes**, que por sua vez são organizadas em pacotes. Esses conceitos são tão centrais em Java que não se pode programar na linguagem sem utilizá-los.

Todo programa em Java usa classes e objetos, e compreender esses conceitos é fundamental para compreender a própria linguagem. Na prática, sistemas de software reais são grandes e precisam ser fatorados em partes relativamente independentes para serem viáveis. Como em Java isso é feito com classes e objetos, compreendê-los é imprescindível para escrever qualquer programa significativo.

Relacionado: [**Curso Básico de Java e Orientação a Objetos**](https://www.devmedia.com.br/curso/basico-de-java-e-orientacao-a-objetos/360)

Antigamente, os programas de computador eram escritos como uma série de instruções, como receitas culinárias: dados os “ingredientes” (arquivos no disco, interfaces de rede, memória, ciclos do processador etc.), uma série de passos era seguida. A semelhança era tão notável que várias linguagens de programação exigiam que os “ingredientes” do programa (variáveis, arquivos etc.) fossem reservados antes de se escrever o “modo de preparo”.

Esta abordagem era ineficiente para grandes programas, com muitas de funcionalidades. Para tentar solucionar isso, uma maneira diferente de programar foi proposta: os softwares não seriam mais vistos apenas como instruções alterando recursos, mas sim como vários pedaços, pequenos *softwares* trabalhando em conjunto para alcançar o objetivo. Os programas não seriam mais pensados como receitas, mas sim como carros e empresas: várias peças, relativamente independentes e fáceis de trocar, exerceriam tarefas complicadas isoladamente, sendo coordenadas de modo que o resultado final fosse uma máquina fácil de dirigir, ou uma organização eficiente. Assim como o operário que põe o motor na carroceria não precisa compreender seu funcionamento interno, e a secretária não precisa saber dos detalhes do setor de *marketing*, o programador poderia se concentrar em uma parte do programa, sabendo que as outras partes poderiam ser escritas depois, ser feitas por outros programadores ou até mesmo compradas. O “programa como uma receita” ainda estaria nessas partes (assim como o motor do carro segue uma série de passos, e o funcionário uma série de procedimentos), mas essas “receitas” seriam menores, mais fáceis de entender, alterar e substituir, podendo haver em uma só parte vários “modos de preparo”.

O primeiro método é chamado [**programação estruturada**](https://www.devmedia.com.br/introducao-a-programacao-estruturada/24951) e ainda hoje é usado. Entretanto, o segundo modo se tornou popular e preferido de vários programadores e organizações. É chamado **programação orientada a objetos e é desse modo que se programa em Java**.

**Objetos e classes**

Em Java os programas são escritos em pequenos pedaços separados, chamados de *objetos*. Objetos são pequenos programas que guardam dentro de si os dados – em suma, as *variáveis* – que precisam para executar suas tarefas. Os objetos também trazem em si, como sub-rotinas, as instruções para processar esses dados. As variáveis que um objeto guarda são chamadas de *atributos*, e as suas sub-rotinas são chamadas de [***métodos***](https://www.devmedia.com.br/trabalhando-com-metodos-em-java/25917). Guarde bem esses nomes, pois você os verá ainda muitas vezes.

Saiba mais: [**Métodos, atributos e classes no Java**](https://www.devmedia.com.br/metodos-atributos-e-classes-no-java/25404)

Em Java, objetos são criados a partir de modelos que os descrevem. Esses modelos são chamados de *classes*. É dentro dessas classes que definimos que atributos os objetos conterão e que métodos os objetos fornecerão.

**Java e Eclipse**

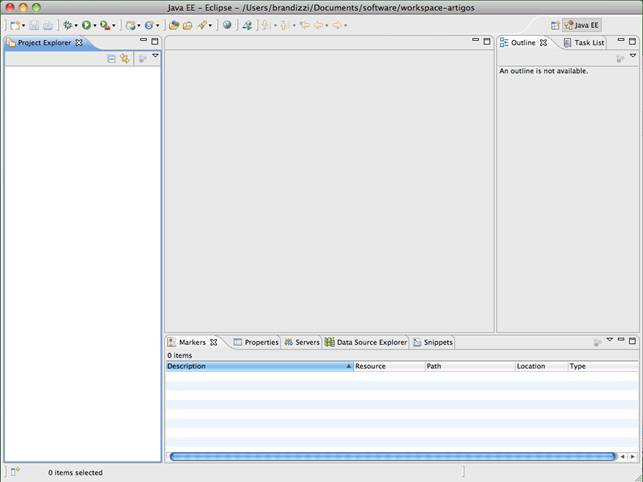
Agora que já sabemos um pouco sobre como se programa em Java, vejamos o que é Java.

**Java é uma linguagem de programação orientada a objetos estática e fortemente tipada**. Programas escritos em Java são *compilados*, como em C e Pascal, mas o resultado da compilação são arquivos que não podem ser executados diretamente pelo computador ou sistema operacional, de modo que precisam também ser *interpretados*, como em BASIC e [**Python**](https://www.devmedia.com.br/guia/python/37024). Java é compilada porque isso permite encontrar erros antes de executar o programa e facilita executar os programas mais eficientemente, e é interpretada porque isso permite que o mesmo arquivo compilado possa ser executado em qualquer computador ou sistema operacional que tenha um interpretador Java – o que, atualmente, equivale a quase *qualquer* computador, de smartphones a mainframes.

Relacionado: [**Guia Completo de Java**](https://www.devmedia.com.br/guia/linguagem-java/38169)

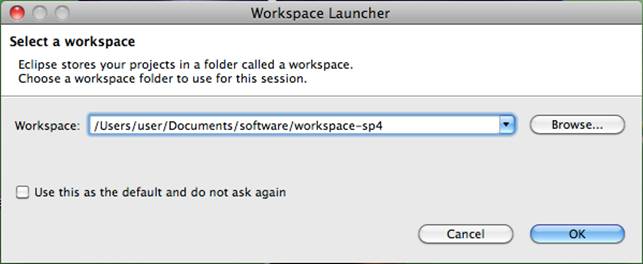
O interpretador Java, além de interpretar as instruções compiladas, também é responsável por retirar da memória objetos que não são mais utilizados. O algoritmo que executa essa tarefa de “limpeza” é chamado de *coletor de lixo*, e é um dos grandes atrativos do Java: em linguagens mais tradicionais, como C e C++, o programador é responsável por limpar a memória na mão, o que é uma tarefa tediosa e até perigosa.

Apesar de tudo, Java é uma linguagem relativamente complicada e verborrágica. Exige-se que se digite muito, facilitando que se erre. É preciso compilar os programas para depois rodá-los. As classes que criarão os objetos dos programas devem ficar em hierarquias de diretórios precisamente definidas, tanto na hora da compilação quanto durante a execução. Cuidar de todos esses detalhes é cansativo e facilita o erro. Por isso, para programar *a sério* em Java, o uso de um ambiente integrado de desenvolvimento – ou como são mais conhecidos, *IDEs* – é praticamente indispensável.

**Figura 1**. IDE Eclipse.

Neste artigo, trabalharemos com a IDE *Eclipse*. O Eclipse é um *software* popular, poderoso e gratuito. Assumimos que o Eclipse, assim como o ambiente Java, já estão devidamente instalados. A **Figura 1**apresenta um screenshot do Eclipse.

Relacionado: [**Conhecendo o Eclipse**](https://www.devmedia.com.br/conhecendo-o-eclipse-uma-apresentacao-detalhada-da-ide/25589)

**Figura 2.** Primeira tela do Eclipse, para escolha do workspace.

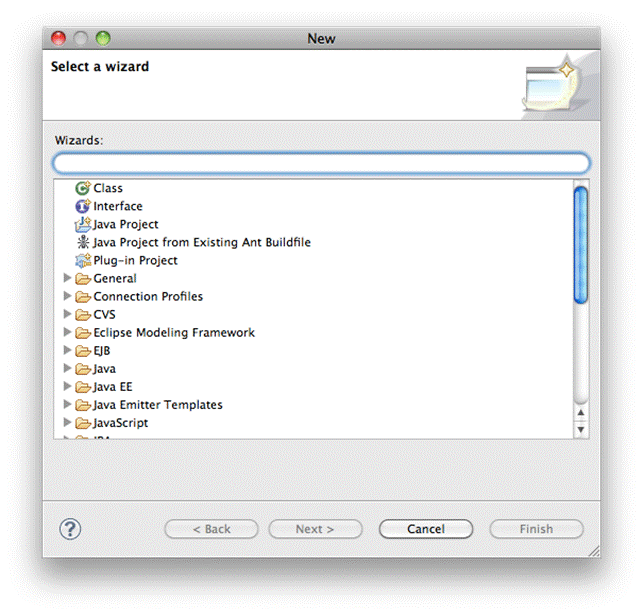
Para chegar a essa tela da **Figura 1**, abra o Eclipse. Primeiro, ele perguntará que workspace você quererá usar, como pode ser visto na **Figura 2**. Workspace é o diretório onde se cria e trabalha em projetos. Se você não tem nenhuma razão para mudar o padrão, apenas clique em Ok. Feito isso, aparecerá a tela da **Figura 3**. Esta tela aparece todas as vezes que se abre um workspace novo; para ir ao ambiente de trabalho, clique no ícone do canto superior direito, com o rótulo [**Workbench**](https://www.devmedia.com.br/mysql-workbench-modelagem-de-dados-passo-a-passo/30021).

**Figura 3.** Tela inicial de um *workspace* novo do Eclipse.

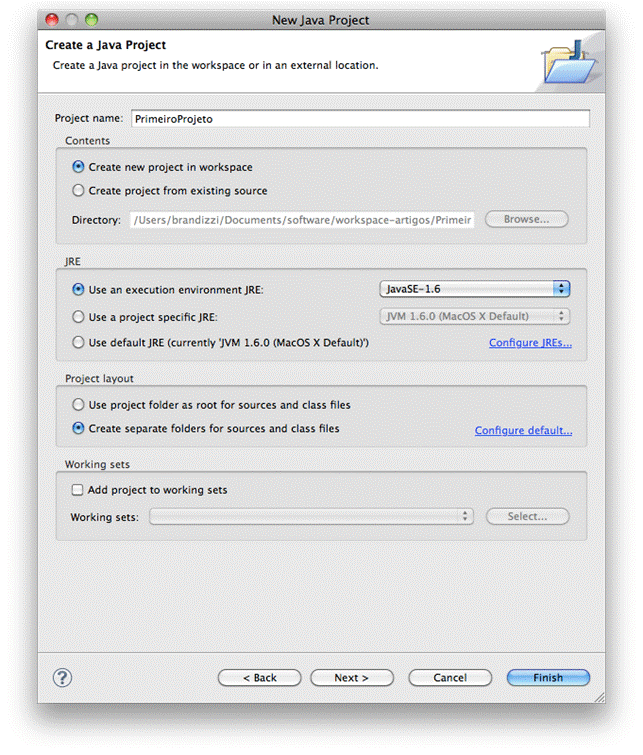
É possível usar vários workspaces com uma mesma [**instalação do Eclipse**](https://www.devmedia.com.br/instalando-o-eclipse-java-development-kit-parte-1/11512). Muitos programadores costumam manter vários workspaces, com projetos relacionados em cada um. Por exemplo, se o desenvolvedor trabalha como freelancer para várias empresas, pode ter um projeto para cada uma; se o desenvolvedor também é estudante, pode manter um projeto para o trabalho e outro para o curso; e assim por diante.

**Classes e pacotes no Eclipse**

Para criar um programa em Java usando o Eclipse, precisamos primeiro criar um *projeto* do Eclipse. Um projeto é uma unidade de trabalho genérica, mas podemos assumir nesse tutorial que um projeto é um programa. Para criá-lo quando o Eclipse já estiver apresentando a tela da **Figura 1**, pressione Control + N. Feito isso você verá a janela da **Figura 4**.

**Figura 4.** Caixa de diálogo para criação de novos elementos.

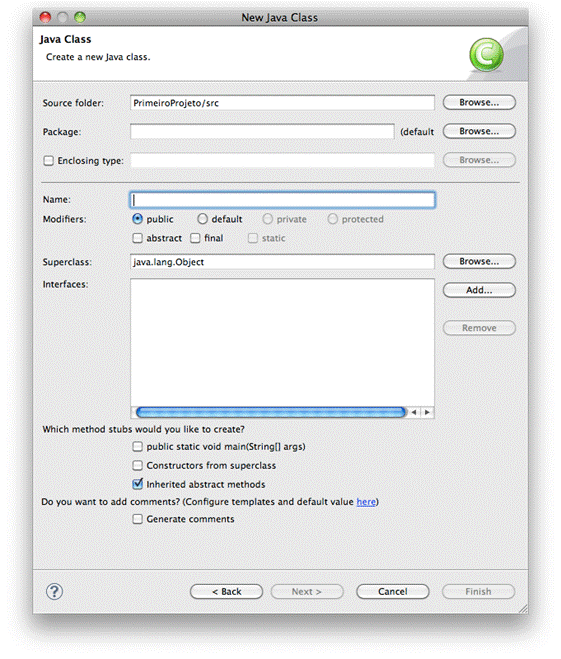
Nela, selecione a opção Java Project e clique em Next. Na tela que pode ser vista na **Figura 5**, podemos configurar vários detalhes do projeto. Por ora, vamos apenas dar um nome para o projeto. Poderíamos clicar em Next e mudar outras configurações, mas não precisamos disso agora, então clique apenas em Finish. O projeto aparecerá no compartimento do lado esquerdo, na aba Project Explorer.

**Figura 5.** Criando um novo projeto.

Criado o projeto, podemos criar as classes. Pressione novamente Control + N, escolha a opção Class e clique em Next. A janela da **Figura 6** aparecerá com várias opções para definição da classe. No campo rotulado Name, digite PrimeiraClasse. Um alerta aparecerá dizendo que é recomendável declarar um pacote (package) também; para declará-lo, digite “artigoinicial.primeironome” no campo rotulado *Package*. Depois clique em Finish.

Pacotes são agrupamentos que podem conter classes e outros pacotes. Cada um é um diretório, e os arquivos de código-fonte das classes – assim como os arquivos compilados – que pertencem a um pacote devem vir dentro do diretório correspondente. Em artigoinicial.primeironome, por exemplo, há dois pacotes: um chamado artigoinicial e outro chamado primeironome, que está dentro do primeiro. Os pacotes são importantes para organizar o código, pois programas podem ter literalmente milhares de classes.

Pacotes também evitam conflitos. É possível usar classes produzidas por outras pessoas e organizações, e alguns nomes são bem comuns, como *File*, *User*, *List* etc. Em Java, toda classe é compilada para um arquivo que tem o mesmo nome – e não seria possível ter dois arquivos com o mesmo nome no mesmo diretório. Porém, se ponho minha classe File no pacote br.com.minhaempresa.nomedoprojeto e você puser a sua em br.org.suaong.projeto, é possível usar ambas as classes em um mesmo programa.

**Figura 6.** Caixa de diálogo para criação de nova classe.

A classe aparecerá no painel do lado direito, como visto na **Figura 7**, e um editor abrirá no centro da tela. Nele, estará o seguinte código:

public class PrimeiraClasse {}

Esse código define uma classe vazia. Na primeira linha, o pacote da classe é declarado. A palavra public significa que a classe é *pública*, quaisquer outras classes podem usar serviços que ela proveja. Já a palavra class define que o que será declarado agora é uma classe; o nome da classe, PrimeiraClasse, deve vir logo após a palavra class. As chaves que se seguem ({ }) definem o *corpo da classe*: é entre elas que os atributos e métodos devem ser descritos. Por fim, é visível na **Figura 7** que o nome do arquivo em que o código-fonte está é PrimeiraClasse.java. Em Java, é obrigatório que o arquivo tenha o nome da classe principal que ele contém seguido da extensão *.java*. Felizmente, todos esses detalhes já foram resolvidos pelo Eclipse.

**Figura 7.** Nova classe vista no painel.

**Método main()**

Todo programa em Java precisa de uma classe que crie os primeiros objetos e lhes passe as tarefas iniciais. Quem faz isso é um método especial, chamado main(). A forma desse método é a seguinte:

public static void main(String[] args) {

}

A palavra public, aqui, significa que qualquer outra classe pode utilizar esse método. main é o nome do método, e antes do nome deve vir o tipo do valor que ele retornará; como o método main não retorna nada, diz-se que ele retorna void. Já a palavra static tem um sentido bem especial: significa que o método pode ser invocado mesmo sem ser criado nenhum objeto da classe. Na maioria das vezes, os métodos de uma classe só podem ser chamados por seus objetos, mas um método static escapa dessa restrição.

Note que o programa é iniciado por um método chamado main(), não por uma função, como em C ou C++. Em Java, não existem funções “livres”, apenas métodos.

Após o nome do método deve vir a lista de parâmetros. main() recebe apenas um parâmetro: um vetor (também chamado array) de objetos da classe String, que representa sequências de caracteres. A declaração do parâmetro é o seu tipo seguido do seu nome (no caso, foi chamado de args, mas poderia ser qualquer identificador não reservado). Note que há um par de colchetes ([]) logo à frente da palavra String; os colchetes significam que o parâmetro args não é uma String, mas sim um array de String. Por ora, esse parâmetro pode ser ignorado.

Em C e C++, declara-se que uma variável é um vetor colocando o par de colchetes após o nome da variável, como em char array[]. É possível fazer isso também em Java, mas geralmente é recomendado colocar o par de colchetes após o tipo, pois isso deixaria claro que o tipo da variável não é String, mas sim um vetor.

O parâmetro que o método main() recebe é um array com todos os argumentos que foram passados, pela linha de comando, para o programa Java, de modo semelhante aos parâmetros argc e argv da função main() de C.

**Olá, mundo!**

Agora que já foi visto como criar uma classe e declarar o método main(), pode-se adicionar o método à classe já criada. Vejamos, por exemplo, como criar o tão famoso “Hello, World”. Para isto, basta acrescentar, dentro do método main() a seguinte linha:

System.out.println("Olá, mundo!");

A classe resultante pode ser vista na **Listagem 1**. Para executá-la, basta clicar no botão Run do Eclipse ( Botão Run do Eclipse). Aparecerá uma aba na parte inferior da janela, chamada Console, e nela haverá uma parte em branco em que estará escrito “Olá, Mundo!”, como pode ser visto na **Figura 8**.

Enfim, acabamos de criar nosso primeiro programa em Java. No começo do artigo foi dito que os programas em Java são compostos por vários objetos; até agora, porém, pode-se ter a impressão de que não só nenhum objeto foi utilizado como apenas uma classe foi usada. Seria um engano: no nosso programa várias classes e objetos foram utilizados.

**Listagem 1**. Exemplo de classe simples com o método main().

public class PrimeiraClasse {

public static void main(String[] args) {

System.out.println("Olá, mundo!");

}

}

**Figura 8.** O que o programa imprime no terminal sai na aba Console.

Para começar, o texto “Olá, Mundo!” foi representado por uma instância de String. (“*Instância* de X” é sinônimo de “objeto da classe X”.) Este objeto String foi passado como parâmetro para o método println() de um objeto apontado pela variável out. A variável out aponta para um objeto da classe PrintStream, que é capaz de imprimir o que for necessário no terminal. A variável out é um atributo estático (isto é, não está dentro de um objeto comum, mas está dentro de uma classe) da classe System, que contém várias ferramentas úteis para o programador.

A expressão "Olá, Mundo!" não representa na verdade um objeto da classe String, mas um objeto vetor de caracteres. Entretanto, Java converte automaticamente vetores de caracteres para objetos da classe String.

O programa da **Listagem 1**, simples como é, usou pelo menos dois objetos e quatro classes. Além destes, é possível que outros objetos e classes tenham sido usados pelos objetos explicitamente mostrados no código. É aí que está a mágica da programação orientada a objetos: todos os detalhes dessa execução foram escondidos, cada objeto cuida dos seus problemas. Com cada objeto resolvendo um “pedaço” da tarefa maior, nem nos damos conta de quantos objetos o programa usa!

**Saiba mais sobre Java ;)**

* [**Guias de Java 66 Guias**](https://www.devmedia.com.br/guias/java):  
  Encontre aqui os Guias de estudo que vão ajudar você a aprofundar seu conhecimento na linguagem Java. Desde o básico ao mais avançado. Escolha o seu!
* [**Como se tornar um Programador Java**](https://www.devmedia.com.br/guia/carreira-programador-java/37809):  
  Aprender Java não é uma tarefa simples, mas seguindo a ordem proposta nesse Guia, você evitará muitas confusões e perdas de tempo no seu aprendizado. Vem aprender java de verdade, vem!
* [**Todos os Cursos de Java**](https://www.devmedia.com.br/cursos/java):  
  Torne-se um programador Java completo. Aqui você encontra cursos sobre as mais ferramentas e frameworks do universo Java. Aprenda a desenvolver sites e web services com JSF, Jersey, Hibernate e mais.

O programa da **Listagem 1** utiliza várias classes e objetos, mas o programador só criou uma classe. Programas mais sofisticados, mesmo que só um pouco, precisam de mais classes.

Considere um programa que leia os nomes de pessoas e imprima o nome completo e as iniciais. Este seria um programa simples que poderia ser escrito com apenas uma classe, mas provavelmente ficaria mais elegante com mais de uma. Por exemplo, uma classe que guardasse os vários nomes da pessoa poderia ser criada. Esta classe precisaria ter atributos para guardar os pedaços do nome:

public class Pessoa {

private String primeiroNome;

private String ultimoNome;

private String nomesDoMeio;

}

No código acima, uma classe chamada Pessoa é definida com três atributos para apontar para os três trechos do nome. Essa classe é um modelo, uma “fôrma” para criar objetos, e cada objeto vai ter seus próprios atributos. Vamos examinar uma das declarações: private String primeiroNome;. A palavra-chave private serve para declarar o atributo como privado: apenas métodos definidos na própria classe Pessoa podem alterar ou ler esses atributos. String, como já vimos antes, é o nome de uma classe que representa uma sequência de caracteres; nessa declaração, significa que o atributo apontará para um objeto criado a partir da classe String – ou, como se costuma dizer, é do tipo String. Por fim, aparece o nome do atributo, primeiroNome. O ponto-e-vírgula que se segue é exigido, em Java, após qualquer declaração de variável (o que inclui atributos) e depois de qualquer comando.

Se você quiser testar esses códigos você mesmo, sugerimos que crie um novo projeto e nele crie as novas classes, como foi explicado anteriormente. Isto é, inclusive, um bom exercício! Experimente!

Confira: [**Como dar os primeiros passos como programador Java**](https://www.devmedia.com.br/java-por-onde-comecar/38224)

Um objeto desta classe não deve apenas “armazenar” dados, mas também processá-los: além dos “ingredientes”, tem de ter também as “receitas”. No caso, o processamento deve ser feito por métodos. Um método necessário é aquele que retorne o nome inteiro a partir dos nomes armazenados no objeto. Esse método poderia ser assim:

public String getNomeCompleto() {

String nomeCompleto = primeiroNome + " " + nomesDoMeio + " " + ultimoNome;

return nomeCompleto;

}

A primeira linha, public String getNomeCompleto(), especifica o método. Primeiro, declara-se, através da palavra-chave public, que o método é público – isto é, qualquer método, de qualquer classe, pode invocá-lo. O método retorna objetos do tipo String e se chama getNomeCompleto. O par de parênteses vazio significa que ele não recebe parâmetro algum.

O conteúdo do método vem entre um par de chaves. Dentro do método, declara-se uma nova variável, do tipo String, chamada nomeCompleto. Ao contrário de variáveis como primeiroNome, nomeCompleto não é um atributo, mas sim uma *variável local*. primeiroNome existirá desde a criação do objeto até sua retirada da memória, mas nomeCompleto só existirá enquanto o método getNomeCompleto() estiver sendo executado, e para cada chamada do método uma nova variável será criada.

A variável nomeCompleto recebe o resultado da concatenação de String. O sinal de atribuição é =, e a concatenação de String é feita através do operador +. Ao final, um ponto-e-vírgula sinaliza o fim deste comando. Abaixo, temos o comando return. Quando ele é invocado, o método termina e o valor que está à sua frente (no caso, o valor referenciado pela variável nnomeCompleto) é retornado.

Como todo programa em Java precisa de uma classe com o método main(), criamos uma nova classe, denominada ProgramaNome, que tenha esse método:

public class ProgramaNome {

public static void main(String[] args) {

}

}

Até agora, não vimos como criar objetos. Para isso, Java provê uma ferramenta especial chamada *construtor*, que são definidos de forma semelhante a métodos (embora não sejam métodos). Nós não definimos nenhum construtor em Pessoa, mas isso não é necessário para criar objetos: toda classe sem construtor definido tem, por padrão, um construtor simples. Os construtores têm o mesmo nome de suas classes e são invocados pelo comando new. A linha abaixo chama o construtor padrão de Pessoa; esse construtor cria um novo objeto, e o objeto é armazenado na variável pessoa:

Pessoa pessoa = new Pessoa();

Em Java, não é possível dar o mesmo nome a uma classe e a um objeto, mas podemos criar uma classe chamada Pessoa e uma variável chamada pessoa porque Java distingue entre maiúsculas e minúsculas. Assim, Pessoa, pessoa, PESSOA e pEsSoA são nomes diferentes.

Para criar uma pessoa que se chamasse, por exemplo, João Paulo Silva, poderíamos tentar criar a seguinte classe:

public class ProgramaNome {

public static void main(String[] args) {

Pessoa pessoa = new Pessoa();

pessoa.primeiroNome = “João”;

pessoa.nomeDoMeio = “Paulo”;

pessoa.ultimoNome = “Silva”;

System.out.println(pessoa.getNomeCompleto());

}

}

No entanto, isto não é possível porque os atributos são privados. Apenas os métodos da classe Pessoa podem acessá-los. Isso pode ser solucionado de várias maneiras, e uma das mais elegantes é criando o nosso próprio construtor, como abaixo:

public Pessoa(String primeiro, String meio, String ultimo) {

primeiroNome = primeiro;

ultimoNome = ultimo;

nomesDoMeio = meio;

}

A declaração do construtor é sempre o nome da classe seguido pela lista de parâmetros. A palavra public indica que o construtor é público, de modo que pode ser invocado por qualquer classe. Um ponto importante sobre construtores é que eles não criam nem retornam objetos; quem faz isso é a palavra reservada new. O construtor apenas executa algum procedimento sobre o objeto criado pelo comando new. Este construtor, no caso, recebe os nomes como parâmetros e os atribui aos atributos.

Agora, podemos trocar as quatro primeiras linhas do método main() inválido por apenas a seguinte:

Pessoa pessoa = new Pessoa("João", "Paulo", "Silva");

A classe Pessoa completa pode ser vista na **Listagem 2**. A classe ProgramaNome, por sua vez, encontra-se na **Listagem 3**. Ao executar o programa, obtém-se no console a seguinte saída:

João Paulo Silva

Mostramos como fazer um programa que imprima o nome completo da pessoa. Agora, propomos um exercício: altere esse programa para que imprima também as iniciais. Crie um método chamado getIniciais() na classe Pessoa. Uma dica: estude o método charAt() da classe String. Boa sorte!

**Listagem 2**.Classe Pessoa, que gera o nome completo a partir das partes.

public class Pessoa {

private String primeiroNome;

private String ultimoNome;

private String nomesDoMeio;

public Pessoa(String primeiro, String meio, String ultimo) {

primeiroNome = primeiro;

ultimoNome = ultimo;

nomesDoMeio = meio;

}

public String getNomeCompleto() {

String nomeCompleto = primeiroNome + " " + nomesDoMeio + " " + ultimoNome;

return nomeCompleto;

}

}

**Listagem 3**. Classe ProgramaNome, que utiliza a classe Pessoa para gerar um nome completo a partir das partes.

public class ProgramaNome {

public static void main(String[] args) {

Pessoa pessoa = new Pessoa("João", "Paulo", "Silva");

System.out.println(pessoa.getNomeCompleto());

}

}

**Herança**

Em Java, os tipos dos objetos são classes. Entretanto, ao contrário dos tipos primitivos, que não são classes (veja o**quadro** “Tipos primitivos”), um objeto pode ter vários tipos – desde que esses tipos estejam hierarquicamente relacionados. Esses relacionamentos são como taxonomias biológicas. Por exemplo, todo mamífero é um animal, e todo cachorro é mamífero. Há uma hierarquia entre esses conceitos, que vai do mais geral (animal) ao mais específico (cachorro).

Por exemplo, poderíamos ter uma classe Animal que representasse animais em geral e as atividades que todos eles executam. Uma dessas atividades é comer, e nossa classe poderia ser como a abaixo. (Se for testá-la, crie um novo projeto e chame-o de “Animais”.)

public class Animal {

public void comer(String alimento) {

System.out.println("Eu estou comendo "+alimento);

}

}

Uma classe de mamíferos *estenderia* a classe de animais, porque mamíferos podem fazer coisas que outros animais não podem – por exemplo, gerar leite. Uma classe Mamifero estenderia a classe Animal usando a palavra reservada extends em sua declaração:

public class Mamifero extends Animal {

}

Um novo método, lactar(), também poderia ser adicionado:

public String lactar() {

return "leite";

}

O resultado pode ser visto na **Listagem 4**.

Como mamíferos são animais, também precisam do método comer(), e aí entra uma das vantagens da herança: como a classe Mamifero estende a classe Animal, ela *herda* todos os métodos de Animal, então não é preciso reescrevê-los. O programa da **Listagem 5** representa bem isso: note como o método comer() é invocado, mesmo sem ser declarado em Mamifero. Esse programa imprimirá:

Eu estou comendo plantas

Eu estou comendo queijo

O produto da lactação é leite

**Listagem 4**. Classe Mamifero.

public class Mamifero extends Animal {

public String lactar() {

return "leite";

}

}

**Listagem 5**. Superclasse e subclasse em uso.

public class AnimaisAgindo {

public static void main(String[] args) {

Animal animal = new Animal();

Mamifero mamifero = new Mamifero();

animal.comer("plantas");

mamifero.comer("queijo");

String produto = mamifero.lactar();

System.out.println("O produto da lactação é " + produto);

}

}

**Tipos primitivos**

Quase todos os tipos de Java são classes, e quase todos os valores são objetos, incluindo strings e vetores. Entretanto, alguns tipos em Java não são classes, e seus valores não são objetos Esses tipos são chamados *tipos primitivos* e representam, basicamente, valores numéricos. A **Tabela Q1** os lista.

**Tabela Q1.** Tipos primitivos em Java.

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| int | Representa números inteiros, de -2147483648 a 2147483647 |
| byte | Representa números inteiros, de -128 a 127 |
| short | Representa números inteiros, de -32768 a 32767 |
| long | Representa números inteiros, de -9223372036854775808 a 9223372036854775807 |
| char | Valores numéricos que vão de 0 a 0xFFFF. Entretanto, não são usados em operações matemáticas; representam caracteres, de acordo com o encoding UTF-16. |
| float | Números com frações usando representação de ponto flutuante. Vai de valores tão pequenos quanto 1,4 ∙ 10-45 até 3,4 ∙ 1038. Representa uma grande faixa de números, mas perde em eficiência comparado com tipos inteiros. |
| double | Números com frações usando representação de ponto flutuante. Abrange uma faixa ainda maior que a do tipo float (de 4,9 ∙ 10-324 até 1,7 ∙ 10308), com precisão ainda maior, mas são ainda mais lentos e ocupam mais memória. |