ORACLE

Atualizado para JAVA SE 8 (JDK 8)

Java
para iniciantes

6ª edição

Crie, compile e execute programas Java rapidamente

Herbert Schildt

bookman

Oracle Press™

O autor

O autor de best-sellers **Herbert Schildt** escreve incansavelmente sobre programação há quase três décadas e é uma das principais autoridades na linguagem Java. Seus livros de programação venderam milhões de cópias no mundo inteiro e foram traduzidos para diversos idiomas. É autor de vários livros sobre Java, incluindo *Java: The Complete Reference, Herb Schildt's Java Programming Cookbook* e *Swing: A Beginner's Guide*. Ele também escreveu sobre C, C++ e C#. Embora tenha interesse em todas as áreas da computação, seu foco principal são as linguagens de programação, incluindo compiladores, interpretadores e linguagens de controle robótico. Também tem grande interesse na padronização de linguagens. Schildt tem graduação e pós-graduação pela Universidade de Illinois. Seu site é www.HerbSchildt.com.

O editor técnico

Dr. Danny Coward trabalhou em todas as edições da plataforma Java. Ele conduziu a definição dos Java Servlets para a primeira versão da plataforma Java EE e para além dela, os serviços web para a plataforma Java ME, e a estratégia e planejamento de Java SE 7. Fundou a tecnologia JavaFX e, mais recentemente, projetou o maior acréscimo feito ao padrão Java EE 7, a API Java WebSocket. Da codificação em Java ao projeto de APIs com especialistas da indústria e ao trabalho por vários anos como executivo do Java Community Process, ele adquiriu uma perspectiva singularmente ampla de vários aspectos da tecnologia Java. Além disso, é autor de *JavaWebSocket Programming* e de um livro ainda a ser publicado sobre Java EE. Dr. Coward tem graduação, mestrado e doutorado em Matemática pela Universidade de Oxford.

S334j Schildt, Herbert.
Java para iniciantes : crie, compile e execute programas Java rapidamente [recurso eletrônico] / Herbert Schildt ; tradução: Aldir José Coelho Corrêa da Silva ; revisão técnica: Maria Lúcia Blanck Lisbôa. – 6. ed. – Porto Alegre : Bookman, 2015.

Editado como livro impresso em 2015.
ISBN 978-85-8260-337-6

1. Linguagem de programação - Java. I. Título.

CDU 004.438Java

Catalogação na publicação: Poliana Sanchez de Araujo – CRB 10/2094

Principais habilidades e conceitos

- Saber os fundamentos da classe
- Entender como os objetos são criados
- Entender como as variáveis de referência são atribuídas
- Criar métodos, retornar valores e usar parâmetros
- Usar a palavra-chave **return**
- Retornar um valor de um método
- Adicionar parâmetros a um método
- Utilizar construtores
- Criar construtores parametrizados
- Entender **new**
- Entender a coleta de lixo e os finalizadores
- Usar a palavra-chave **this**

Antes de poder se adiantar mais em seu estudo de Java, você precisa conhecer a classe. Classe é a essência de Java. Ela é a fundação na qual toda a linguagem Java se estrutura, porque define a natureza de um objeto. Como tal, ela forma a base da programação orientada a objetos em Java. Dentro de uma classe, são definidos dados e o código que age sobre eles. O código fica contido em métodos. Já que as classes, objetos e métodos são fundamentais para Java, eles serão introduzidos neste capítulo. Ter um entendimento básico desses recursos permitirá que você escreva programas mais sofisticados e compreenda melhor certos elementos-chave de Java descritos no próximo capítulo.

Fundamentos das classes

Já que toda a atividade dos programas Java ocorre dentro de uma classe, temos usado classes desde o início deste livro. É claro que só classes extremamente simples foram usadas e não nos beneficiamos da maioria de seus recursos. Como você verá, elas são significativamente mais poderosas do que as classes limitadas apresentadas até agora.

Comecemos examinando o básico. Uma classe é um modelo que define a forma de um objeto. Ela especifica tanto os dados quanto o código que operará sobre eles. Java usa uma especificação de classe para construir *objetos*. Os objetos são *instâncias* de uma classe. Logo, uma classe é basicamente um conjunto de planos

que especifica como construir um objeto. É importante deixar uma coisa bem clara: uma classe é uma abstração lógica. Só quando um objeto dessa classe é criado é que existe uma representação física dela na memória.

Outro ponto: lembre-se de que os métodos e variáveis que compõem uma classe são chamados de *membros* da classe. Os membros de dados também são chamados de *variáveis de instância*.

Forma geral de uma classe

Quando definimos uma classe, declaramos sua forma e natureza exatas. Fazemos isso especificando as variáveis de instância que ela contém e os métodos que operam sobre elas. Embora classes muito simples possam conter apenas métodos ou apenas variáveis de instância, a maioria das classes do mundo real contém ambos.

Uma classe é criada com o uso da palavra-chave **class**. Uma forma geral simplificada de uma definição **class** é mostrada aqui:

```
class nome da classe {
  // declara variáveis de instância
  tipo var1;
  tipo var2;
  // ...
  tipo varN;

  // declara métodos
  tipo método1(parâmetros) {
     // corpo do método
  }
  tipo método2(parâmetros) {
     // corpo do método
  }
  // ...
  tipo métodoN(parâmetros) {
     // corpo do método
  }
}
```

Embora não haja essa regra sintática, uma classe bem projetada deve definir apenas uma entidade lógica. Por exemplo, normalmente, uma classe que armazena nomes e números de telefone não armazena também informações sobre o mercado de ações, a média pluviométrica, os ciclos das manchas solares ou outros dados não relacionados. Ou seja, uma classe bem projetada deve agrupar informações logicamente conectadas. A inserção de informações não relacionadas na mesma classe desestruturará rapidamente seu código!

Até o momento, as classes que usamos tinham apenas um método: **main( )**. Você verá como criar outros em breve. No entanto, observe que a forma geral de uma classe não especifica um método **main( )**. O método **main( )** só é necessário quando a classe é o ponto de partida do programa. Alguns tipos de aplicativos Java, como os applets, também precisam de um método **main( )**.

Definindo uma classe

Para ilustrar as classes, desenvolveremos uma classe que encapsula informações sobre veículos, como carros, furgões e caminhões. Essa classe se chama **Vehicle** e conterá três informações sobre um veículo: o número de passageiros que ele pode levar, a capacidade de armazenamento de combustível e o consumo médio de combustível (em milhas por galão).

A primeira versão de **Vehicle** é mostrada a seguir. Ela define três variáveis de instância: **passengers**, **fuelcap** e **mpg**. Observe que **Vehicle** não contém método. Logo, atualmente é uma classe só de dados. (Seções subsequentes adicionarão métodos a ela.)

```
class Vehicle {
  int passengers; // número de passageiros
  int fuelcap;    // capacidade de armazenamento de combustível em galões
  int mpg;        // consumo de combustível em milhas por galão
}
```

Uma definição **class** cria um novo tipo de dado. Nesse caso, ele se chama **Vehicle**. Você usará esse nome para declarar objetos de tipo **Vehicle**. Lembre-se de que uma declaração **class** é só uma descrição de tipo; ela não cria um objeto real. Logo, o código anterior não faz nenhum objeto de tipo **Vehicle** passar a existir.

Para criar realmente um objeto **Vehicle**, você usará uma instrução como a mostrada abaixo:

```
Vehicle minivan = new Vehicle(); // cria um objeto Vehicle chamado minivan
```

Após essa instrução ser executada, **minivan** será uma instância de **Vehicle**. Portanto, terá realidade "física". Por enquanto, não se preocupe com os detalhes da instrução.

Sempre que você criar uma instância de uma classe, estará criando um objeto contendo sua própria cópia de cada variável de instância definida pela classe. Logo, todos os objetos **Vehicle** conterão suas próprias cópias das variáveis de instância **passengers**, **fuelcap** e **mpg**. Para acessar essas variáveis, você usará o operador ponto (.). O *operador ponto* vincula o nome de um objeto ao nome de um membro. A forma geral do operador ponto é mostrada aqui:

objeto.membro

Portanto, o objeto é especificado à esquerda e o membro é inserido à direita. Por exemplo, para atribuir o valor 16 à variável **fuelcap** de **minivan**, use a instrução a seguir:

```
minivan.fuelcap = 16;
```

Em geral, podemos usar o operador ponto para acessar tanto variáveis de instância quanto métodos.

Este é um programa completo que usa a classe **Vehicle**:

```
/* Um programa que usa a classe Vehicle.

   Chame este arquivo de VehicleDemo.java
*/
class Vehicle {
```

```
  int passengers; // número de passageiros
  int fuelcap;    // capacidade de armazenamento de combustível em galões
  int mpg;        // consumo de combustível em milhas por galão
}

// Essa classe declara um objeto de tipo Vehicle.
class VehicleDemo {
  public static void main(String args[]) {
    Vehicle minivan = new Vehicle();
    int range;

    // atribui valores a campos de minivan
    minivan.passengers = 7;
    minivan.fuelcap = 16; <-- Observe o uso do operador ponto
    minivan.mpg = 21;         para o acesso a um membro.

    // calcula a autonomia presumindo um tanque cheio de gasolina
    range = minivan.fuelcap * minivan.mpg;
    System.out.println("Minivan can carry " + minivan.passengers +
                       " with a range of " + range);
  }
}
```

Você deve chamar o arquivo que contém o programa de **VehicleDemo.java**, porque o método **main( )** está na classe chamada **VehicleDemo** e não na classe chamada **Vehicle**. Quando compilar esse programa, verá que dois arquivos **.class** foram criados, um para **Vehicle** e um para **VehicleDemo**. O compilador Java insere automaticamente cada classe em seu próprio arquivo **.class**. Não é necessário as classes **Vehicle** e **VehicleDemo** estarem no mesmo arquivo-fonte. Você pode inserir cada classe em seu próprio arquivo, chamados **Vehicle.java** e **VehicleDemo.java**, respectivamente.

Para executar o programa, você deve executar **VehicleDemo.java**. A saída a seguir é exibida:

```
Minivan can carry 7 with a range of 336
```

Antes de avançar, examinemos um princípio básico: cada objeto tem suas próprias cópias das variáveis de instância definidas por sua classe. Logo, o conteúdo das variáveis de um objeto pode diferir do conteúdo das variáveis de outro. Não há conexão entre os dois objetos exceto pelo fato de serem do mesmo tipo. Por exemplo, se você tiver dois objetos **Vehicle**, cada um terá sua própria cópia de **passengers**, **fuelcap** e **mpg**, e o conteúdo dessas variáveis será diferente entre os dois objetos. O programa abaixo demonstra esse fato. (Observe que a classe que tem **main( )** agora se chama **TwoVehicles**.)

```
// Este programa cria dois objetos Vehicle.

class Vehicle {
  int passengers; // número de passageiros
  int fuelcap;    // capacidade de armazenamento de combustível em galões
```

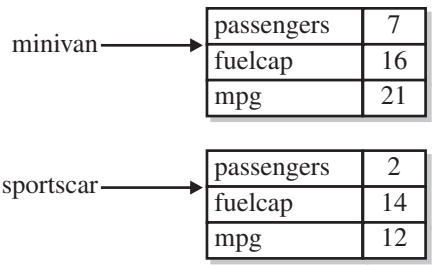
```
  int mpg;        // consumo de combustível em milhas por galão
}

// Essa classe declara um objeto de tipo Vehicle.
class TwoVehicles {
  public static void main(String args[]) {
    Vehicle minivan = new Vehicle();
    Vehicle sportscar = new Vehicle();

    int range1, range2;

    // atribui valores a campos de minivan
    minivan.passengers = 7;
    minivan.fuelcap = 16;
    minivan.mpg = 21;

    // atribui valores a campos de sportscar
    sportscar.passengers = 2;
    sportscar.fuelcap = 14;
    sportscar.mpg = 12;

    // calcula a autonomia presumindo um tanque cheio de gasolina
    range1 = minivan.fuelcap * minivan.mpg;
    range2 = sportscar.fuelcap * sportscar.mpg;

    System.out.println("Minivan can carry " + minivan.passengers +
                       " with a range of " + range1);

    System.out.println("Sportscar can carry " + sportscar.passengers +
                       " with a range of " + range2);
  }
}
```

Lembre-se de que **minivan** e **sportscar** referenciam objetos separados.

A saída produzida por esse programa é mostrada aqui:

```
Minivan can carry 7 with a range of 336
Sportscar can carry 2 with a range of 168
```

Como você pode ver, os dados de **minivan** são totalmente diferentes dos contidos em **sportscar**. A ilustração a seguir mostra essa situação.



Como os objetos são criados

Nos programas anteriores, a linha abaixo foi usada para declarar um objeto de tipo **Vehicle**:

```
Vehicle minivan = new Vehicle();
```

Essa declaração faz duas coisas. Em primeiro lugar, ela declara uma variável chamada **minivan** da classe **Vehicle**. Essa variável não define um objeto. Em vez disso, ela pode apenas *referenciar* um objeto. Em segundo lugar, a declaração cria uma cópia física do objeto e atribui à **minivan** uma referência a ele. Isso é feito com o uso do operador **new**.

O operador **new** aloca dinamicamente (isto é, aloca no tempo de execução) memória para um objeto e retorna uma referência a ele. Essa referência é, mais ou menos, o endereço do objeto na memória alocado por **new**. A referência é então armazenada em uma variável. Logo, em Java, todos os objetos de uma classe devem ser alocados dinamicamente.

As duas etapas da instrução anterior podem ser reescritas desta forma para mostrarmos cada etapa individualmente:

```
Vehicle minivan; // declara uma referência ao objeto
minivan = new Vehicle(); // aloca um objeto Vehicle
```

A primeira linha declara **minivan** como referência a um objeto de tipo **Vehicle**. Portanto, **minivan** é uma variável que pode referenciar um objeto, mas não é um objeto. Por enquanto, **minivan** não referencia um objeto. A próxima linha cria um novo objeto **Vehicle** e atribui à **minivan** uma referência a ele. Agora, **minivan** está vinculada a um objeto.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.