Classes, objetos, métodos

Classes

- 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, ter um entendimento básico, permitirá que você escreva programas e compreenda melhor certos elementos-chave de Java.

Fundamentos das classes

- Já que todas as atividades dos programas Java ocorre dentro de uma classe, utilizamos as classes simples desde o começo, porém elas são significativamente mais poderosas do que as utilizadas até agora.
- Começamos 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.

Fundamentos das classes

- Logo, uma classe é basicamente um conjunto de planos que especifica como construir um objeto.
- É importante deixar claro que uma classe é uma abstração.
- Só quando um objeto dessa classe é criado é que existe uma representação física dele na memória.

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.
- Uma classe é criada com o uso da palavra-chave class.
- Uma forma geral simplificada de uma definição class:
- Vejamos o exemplo da pg 105.

Forma Geral de uma Classe

- Até o momento, as classes que usamos tinha apenas um método: main().
- 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 as 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).
- Exercício da pg 106.

Como os objetos são criados

- A linha abaixo é 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 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.

Como os objetos são criados

- O operador new aloca dinamicamente 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.

Como os objetos são criados

- 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 do 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.

Variáveis de referência e a atribuição

- Em uma operação de atribuição, variáveis de referência, agem diferentemente das de um tipo primitivo, como int.
- Quando a atribuição se dá entre variáveis de tipo primitivo, a situação é simples. A variável da esquerda recebe uma cópia do valor da direita. Quando se dá entre variáveis de referência de objetos, é um pouco mais complicado, porque estamos alterando o objeto para o qual a variável de referência aponta. O efeito dessa diferença pode causar alguns resultados inesperados.
- Exemplo da pg 109.

Métodos

- Os métodos são sub-rotinas que tratam os dados definidos pela classe e, em muitos casos, dão acesso a esses dados.
- Um método contém uma ou mais instruções. **Em um código Java bem escrito, cada método executa apenas uma tarefa**. Cada método tem um nome e é esse nome que é usado para chamá-lo. Em geral, podemos dar a um método o nome que quisermos. No entanto, lembre-se de que main() está reservado para o método que começa a execução do programa. Além disso, não use palavras-chave Java para nomear métodos.

Métodos

- Normalmente os métodos de uma classe tratam e dão acesso aos dados da classe.
- Com isso em mente, lembre-se de que o método main() dos exemplos anteriores calculava a autonomia de um veículo multiplicando seu consumo pela capacidade de armazenamento de combustível.
- Embora tecnicamente correta, essa não é a melhor maneira de realizar o cálculo.
- O cálculo de autonomia de um veículo é algo que é realizado mais adequadamente pela própria classe Vehicle.

Métodos

- Ao adicionar à classe Vehicle um método que calcule a autonomia, você estará melhorando sua estrutura orientada a objetos.
- Para adicionar um método a Vehicle, especificando-o dentro da declaração da classe.
- Por exemplo o exemplo da pg 111 mostra um método criado na classe **Vehicle** chamado **range()** que exibe a autonomia do veículo.

Retornando um método

- Em geral, há duas condições que fazem um método retornar.
- A primeira, como o método **range()** do exemplo anterior mostra, é quando a chave de fechamento do método é alcançada.
- A segunda é quando uma instrução **return** é executada.
- Há duas formas de **return** – uma para uso em métodos **void**(métodos que não retornam valor) e outra para o retorno de valores.

Retornando um método

- A primeira forma conheceremos mais detalhadamente agora.
- Iremos agora entender como retornar valores.
- No exemplo da pg 113, note que você pode causar o encerramento imediato de um método **void** usando somente **return**; - Quando esta instrução é executada, o controle do programa volta para o chamador, saltando qualquer código restante no método.

Retornando um valor

- Embora não sejam raros métodos com tipo de retorno **void,** a maioria dos métodos retorna um valor. Na verdade, a possibilidade de retornar uma valor é um dos recursos mais úteis dos métodos.
- Os valores de retorno são usados para vários fins em programação. Em alguns casos, o valor de retorno contém o resultado de um cálculo. Em outros, pode simplesmente indicar sucesso ou falha. Em outros ainda, pode conter um código de status.
- Qualquer que seja a finalidade, o uso de valores de retorno é parte integrante da programação Java.

Retornando um valor

- Os métodos retornam um valor de rotina chamadora usando essa forma de **return: - return** *valor*;
- Aqui *valor* é o valor retornado. Essa forma de **return** só pode ser usada com métodos que tenham sido de retorno diferente de **void**. Além disso, um método não **void** deve retornar um valor usando essa versão do **return**.
- O exercicio da pg 114 modifica **range()** para retornar a autonomia em vez de exibi-la.

Usando parâmetros

- Podemos passar um ou mais valores para um método quando ele é chamado.
- Um valor passado para um método se chama argumento. Dentro do método, a variável que recebe o argumento se chama parâmetro.
- Os parâmetros são declarados dentro dos parênteses que vêm após o nome do método. A sintaxe de declaração de parâmetros é a mesma usada para variáveis.
- Um parâmetro faz parte do escopo de seu método e, exceto pela tarefa especial de receber um argumento, ele age como qualquer variável local.

Usando parâmetros

- Um método pode ter mais de um parâmetro.
- Simplesmente declare cada parâmetro, separado um do outro com uma vírgula.
- Quando são usados vários parâmetros, cada parâmetro especifica seu próprio tipo, que pode diferir dos outros.

Método Parametrizado

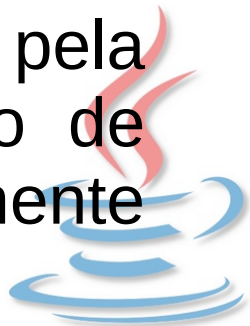
- Você pode usar um método parametrizado para adicionar um novo recurso à classe Vehicle: a possibilidade de calcular a quantidade de combustível necessária para cobrir uma determinada distância. Esse novo método chama fuelneeded().
- Ele recebe o número de milhas que você quer percorrer e retorna quantos galões de gasolina são necessários.
- O exercício da pg. 117 mostra como funciona.
- O exercício da pg. 119 é um resumo de tudo o que vimos até o momento.

Construtores

- Nos exemplos anteriores, as variáveis de instância de cada objeto Vehicle tiveram de ser configuradas manualmente com o uso de uma sequencia de instruções.
- Uma abordagem como essa nunca seria usada em um código Java escrito profissionalmente.
- Além de ser propensa a erros(você pode esquecer de configurar um dos campos), há uma maneira melhor de executar essa tarefa: o construtor.

Construtores

- Um construtor inicializa um objeto quando esse é criado.
- Ele tem o mesmo nome de sua classe e é sintaticamente semelhante a um método.
- No entanto, os construtores não têm um tipo de retorno explícito.
- Normalmente, usamos um construtor para fornecer valores iniciais para as variáveis de instâncias definidas pela classe ou para executar algum outro procedimento de inicialização necessário à criação de um objeto totalmente formado.



Construtores

- Todas as classes têm construtores, mesmo quando não definimos um, porque Java fornece automaticamente um construtor padrão que inicializa todas as variáveis membros com seus valores padrão, que são zero, null e false, para tipos numéricos, tipos de referência e booleans, respectivamente.
- No entanto, quando definimos nosso próprio construtor, o construtor padrão não é mais usado.

Construtores Parametrizados

- Embora o construtor sem parâmetros seja adequado em algumas situações, quase sempre você precisará de um construtor que aceite um ou mais parâmetros.
- Os parâmetros são adicionados a um construtor da mesma forma que são adicionados a um método: apenas declare-os dentro de parênteses após o nome do construtor.
- O exercício da pg 126 mostra como adicionar o construtor na classe **Vehicle**.

O operador new revisitado

- Examinaremos com detalhes o operador new. No contexto de uma atribuição, o operador new tem esta forma geral:

var-classe = new nome-classe(lista-arg);

- Aqui, var-classe é uma variável do tipo de classe que está sendo criada.
- Nome-classe é o nome da classe que está sendo instanciada.

O operador new revisitado

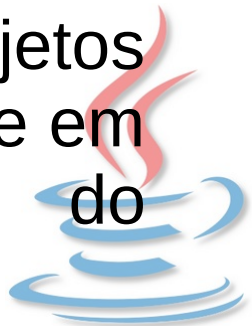
- O nome da classe seguido por uma lista de argumentos entre parênteses(que pode estar vazia) especifica o construtor da classe. Se uma classe não definir seu próprio construtor, new usará o construtor padrão fornecido por Java.
- Logo, new pode ser usado para criar um objeto de qualquer tipo de classe. O operador new retorna uma referência ao objeto recém-criado, que é atribuído a var-classe.
- Já que a memória é finita, é possível que new não consiga alocar memória para um objeto por não existir memória suficiente. Se isso ocorrer, haverá uma exceção de tempo de execução.

Coleta de Lixo e Finalizadores

- Vimos que a alocação de objetos se dá dinamicamente a partir de uma porção de memória livre com o uso do operador new.
- Como explicado, a memória não é infinita e o espaço livre pode se extinguir. Portanto, é possível que new falhe por não haver memória suficiente para a criação do objeto desejado.
- Logo, um componente-chave de qualquer esquema de alocação dinâmica é a recuperação de memória livre de objetos não usados, com a disponibilização dessa memória para realocações subsequentes.

Coleta de Lixo e Finalizadores

- Em muitas linguagens de programação, a liberação de memória já alocada é realizada manualmente.
- Por exemplo, em C ++, usamos o operador delete para liberar memória que foi alocada.
- No entanto, Java usa uma abordagem diferente, mas livre de problemas: a coleta de lixo.
- O sistema de coleta de lixo de Java reclama objetos automaticamente – ocorrendo de maneira transparente em segundo plano, sem nenhuma intervenção do programador.



Coleta de Lixo e Finalizadores

- Funciona assim: quando não existe nenhuma referência a um objeto, ele não é mais considerado necessário e a memória ocupada é liberada.
- Essa memória reciclada pode então ser usada para uma alocação subsequente.
- A coleta de lixo só ocorre esporadicamente durante a execução do programa.
- Ela não ocorrerá só porque existem um ou mais objetos que não são mais usados.

Coleta de Lixo e Finalizadores

- A título de eficiência, geralmente o coletor de lixo só é executado quando duas condições são atendidas: há objetos a serem reciclados e há a necessidade de reciclá-los.
- Lembre-se, a coleta de lixo é demorada, logo, o sistema de tempo de execução.
- Java só a executa quando apropriado.
- Portanto, não temos como saber exatamente quando ela ocorrerá.

A palavra-chave this

- Quando um método é chamado, ele recebe automaticamente um argumento implícito, que é uma referência ao objeto chamador(isto é, o objeto em que o método é chamado).
- Essa referência se chama **this**.
- Exercício pg 132 serve para entender **this**, primeiro considere um programa que cria uma classe chamada **Pwr** para calcular o resultado de um número elevado e alguma potência inteira.