Programação orientada a objetos em Java

Profª. Heloisa Moura

Programação orientada a objetos em Java

O que vamos ver:

- Construtores;
- Palavra reservada this;
- Modificadores de acesso;
- Encapsulamento: métodos getters e setters;
- Sobrecarga de métodos e construtores (overload)

Programação orientada a objetos em Java
Construtores

Até agora...

```
Carro van = new Carro();
van.marca = “Fiat”;
van.modelo = “Ducato”;
 van.numeroPassageiros = 10;
 van.capCpmbustivel = 100;
 van.consumoCombustivel= 5;
```

Programação orientada a objetos em Java
Construtores

Quando usamos a palavra-reservada new , estamos construindo um objeto. Sempre quando o new é chamado, ele executa o construtor da classe.

O Construtor Default

Até agora, as nossas classes não tinham nenhum construtor. Então, como é que era possível dar new se todo new chama um construtor obrigatoriamente?

Quando você não declara nenhum construtor na sua classe, o Java cria um para você. Esse construtor é o construtor default. Ele não recebe nenhum parâmetro e o seu corpo é vazio.

Programação orientada a objetos em Java
Construtores

Isso é o mesmo que...

```java
package com.entra21.cursojavamanha.oop.exerciciospratica.metodos;

public class Carro {

	// TODO Auto-generated method stub
	String marca;
	String modelo;
	int numeroPassageiros;  // numero de passageiros
	double capCombustivel; // capacidade do tanque de combustível
	double consumoCombustivel; // consumo de combustível por km


	void exibirAutonomia(){ // METODO SEM RETORNO
	    System.out.println(" A autonomia do carro é: "  +capCombustivel * consumoCombustivel + " km " );
	    }

	double obterAutonomia(){ // MÉTODO COM RETORNO
	    System.out.println(" Método obterAutonomia do carro foi chamado "  );
	    return capCombustivel * consumoCombustivel;
	    }

	double calcularCombustivel(double km){
	    return km/consumoCombustivel ;
	}

}
```

Programação orientada a objetos em Java
Construtores

Isso...

```java
package com.entra21.cursojava.manha.oop.exerciciospratica.construtores;

public class Carro {

        // TODO Auto-generated method stub
        String marca;
        String modelo;
        int numeroPassageiros;  // numero de passageiros
        double capCombustivel; // capacidade do tanque de combustível
        double consumoCombustivel; // consumo de combustível por km

         Carro(){ //Boa pratica de programação. Sempre ter um construtor vazio
         }

        void exibirAutonomia(){ // METODO SEM RETORNO
            System.out.println(" A autonomia do carro é: "  +capCombustivel * consumoCombustivel + " km " );
            }

        double obterAutonomia(){ // MÉTODO COM RETORNO
            System.out.println(" Método obterAutonomia do carro foi chamado "  );
            return capCombustivel * consumoCombustivel;
            }

        double calcularCombustivel(double km){
            return km/consumoCombustivel ;
        }

}
```

Programação orientada a objetos em Java
Construtores

A partir do momento que você declara um construtor, o construtor default não é mais fornecido O construtor da classe é um bloco declarado com o mesmo nome que a classe.
Um construtor pode receber nenhum, um ou vários parâmetros, inicializando assim, algum tipo de informação.

```
class Carro{
        String marca;
        String modelo;
        int numPassageiros;  // numero de passageiros

        Carro() {
                System.out.println(“Construindo um Carro");
        }
}
```

Programação orientada a objetos em Java
Construtores

Então, quando fizermos:

```
Carro van = new Carro();
```

A mensagem “Construindo um Carro" aparecerá. É como uma rotina de inicialização que é chamada sempre que um novo objeto é criado. Um construtor pode parecer, mas não é um método. Não tem retorno e só é chamado durante a construção do objeto.

Programação orientada a objetos em Java
Construtores

```
class Carro{
        String marca;
        String modelo;
         int numPassageiros;  // numero de passageiros
        Carro() {
                numPassageiros = 4; //inicializa o atributo numPassageiros com 4
        }                                   quando for criado um novo objeto  da classe
                                            Carro
        Carro( String modelo_) {
                modelo = modelo_
        }
```

Programação orientada a objetos em Java
Construtores

A necessidade de um construtor

Tudo estava funcionando até agora. Para que utilizamos um construtor? A ideia é bem simples. Se todo Carro precisa de uma marca, numero de passageiros, como obrigar todos os objetos que forem criados a ter um valor desse tipo? É só criar um único construtor que receba essa String e um int. O construtor se resume a isso! Dar possibilidades ou obrigar o usuário de uma classe a passar parâmetros para o objeto durante o seu processo de criação.

Programação orientada a objetos em Java
Construtores

Por exemplo, não podemos abrir um arquivo para leitura sem dizer qual é o nome do arquivo que desejamos ler. Portanto, nada mais natural que passar uma String representando o nome de um arquivo na hora de criar um objeto do tipo de leitura de arquivo, e que isso seja obrigatório. Você pode ter mais de um construtor na sua classe, e, no momento do new , o construtor apropriado será escolhido. Existe um outro motivo, o outro lado dos construtores: facilidade. Às vezes, criamos um construtor que recebe diversos parâmetros para não obrigar o usuário de uma classe a chamar diversos métodos do tipo set.

Programação orientada a objetos em Java
Construtores

```
class Carro{
        String marca;
        String modelo;
        int numPassageiros;  // numero de passageiros

         Carro() {  Boa pratica de programação. Sempre ter um  construtor vazio
        }

        Carro(String marca_) {
                marca = marca_
        }
```

Programação orientada a objetos em Java
Modificadores de acesso

Exemplo na prática

Programação orientada a objetos em Java
Construtores

Palavra Reservada this

A instrução this é uma palavra-chave que sempre estará apontando para o objeto em memória. A mesma é uma forma de nos referirmos aos atributos e métodos do objeto em questão (que está na memória) sem chama-los pelo seu nome. Ou seja referencia atributos e métodos da própria classe.

A palavra this nos ajuda a resolver a seguinte situação:

Programação orientada a objetos em Java
Construtores

```
class Carro{
        String marca;
        String modelo;

         Carro() {
        }
         Carro( String marca_, String modelo_) {
                marca = marca_;
                modelo = modelo_;
         }
```

Os parâmetros marca_ e modelo_ apesar de poderem ser escritos com o _ e funcionar, não seria um código mais adequado.

Programação orientada a objetos em Java
Construtores

```
class Carro{
        String marca;
        String modelo;

         Carro() {
        }
         Carro( String marca, String modelo) {
                marca = marca;
                modelo = modelo;
         }
```

Poderíamos retirar o _ dos parâmetros. Também iria funcionar. Mas teríamos tanto parâmetro como atributos com o mesmo nome e isso nos deixa confuso, sem saber quem é o atributo e que é o parâmetro.
E COMO RESOLVER ISSO ???

Programação orientada a objetos em Java
Construtores

```
class Carro{
        String marca;
        String modelo;

         Carro() {
        }
         Carro( String marca, String modelo) {
                this.marca = marca;
                this.modelo = modelo;
         }
```

Utilizando a palavra this no atributo da classe. Ambos os atributos irão receber os respectivos parâmetros, que tem o mesmo nome, vindo do construtor. O código fica mais claro para entendimento e totalmente separado.

Programação orientada a objetos em Java
Construtores

É possível também utilizar o this dentro de métodos.

```
void exibirAutonomia(){ // METODO SEM RETORNO
        System.out.println(" A autonomia do carro é: "  +this.capCombustivel *
        this.consumoCombustivel + " km " );
}

double obterAutonomia(){ // MÉTODO COM RETORNO
        System.out.println(" Método obterAutonomia do carro foi chamado ");
        return this.capCombustivel * this.consumoCombustivel;
}
```

Programação orientada a objetos em Java
Construtores

Exemplo na pratica

Programação orientada a objetos em Java
Modificadores de acesso

Na programação Java quando definimos um atributo ou método, nós escolhemos como será a sua visibilidade (como faremos o encapsulamento), que são de quatro tipos: público, protegido, de pacote ou privado. Essa escolha é feita de acordo a necessidade, mas, para isso é necessário se conhecer qual a visibilidade de cada um dos tipos.

Programação orientada a objetos em Java
Modificadores de acesso

- Protected (protegido) Será visto em herança

  Diferente da pública, esta visibilidade só se aplica a atributos e métodos. Quando um atributo (ou operação) é protegido, no Java, ele é visível por objetos das subclasses (que estão dentro do pacote) ou mesmo por objetos de outras classes que se encontram dentro do mesmo pacote.

- De pacote (Default)

  Os métodos ou atributos marcados com este tipo de visibilidade são acessíveis por todos os objetos pertencentes a todas as classes que se encontram dentro do mesmo pacote. Quando queremos que nossos recursos sejam “de pacote”, não usamos nenhum tipo de modificador (public, protected ou private)

Programação orientada a objetos em Java
Modificadores de acesso

Private e public

- Private (privado)
  Este tipo de visibilidade é o mais restritivo possível, pois recursos privados só são acessíveis por objetos da mesma classe na qual eles foram definidos. É recomendável que todos os atributos de nossas classes sejam do tipo privado, pois não queremos que sejam modificados diretamente por outros objetos pertencentes a outras classes. Quando queremos que uma classe seja reutilizável, normalmente, definimos seus atributos como privado e suas operações como pública para ter acesso externo.

Programação orientada a objetos em Java
Modificadores de acesso

Private e public

- Public (público)
  Quando definimos um atributo (ou operação) como público, queremos dizer que ele é visível (acessível) por qualquer objeto de qualquer classe e em qualquer pacote, ou seja, é a visibilidade mais ampla que pode ser oferecida a um usuário da classe. Este tipo de visibilidade serve tanto para classe quanto para método e atributos.

Programação orientada a objetos em Java
Construtores

Exemplo na pratica

Programação orientada a objetos em Java
Encapsulamento

Na orientação a objetos encapsulamento significa ocultação de detalhes de implementação dos usuários de determinados objetos, em outras palavras, estamos protegendo nossos atributos contra uso indevido. Vamos pegar o seu carro como exemplo, quando você o liga, você nem sabe (e nem quer saber) o que acontece para que ele dê a partida. Na verdade, você quer é apenas a funcionalidade que esse pode lhe oferecer. Mas, como você não tem acesso às partes internas do motor, o objeto (no caso, carro) veio provido de uma interface para que o usuário (no caso, você) se comunicasse com essas partes oculta.

Programação orientada a objetos em Java
Encapsulamento

Na programação orientada a objetos acontece a mesma coisa, encapsulamos detalhes de implementação e oferecemos uma interface para a comunicação entre o objeto e os usuários deste.

E como fazemos isso?

Programação orientada a objetos em Java
Encapsulamento

Métodos getters e setters

Usamos métodos específicos para que seja possível acessar e modificar, os valores de atributos fora da classe (ou seja, indiretamente). Utilizando o modificador de acesso **private.** Tais atributos, passam a serem vistos diretamente, somente dentro da classe. Dessa forma métodos de fora dessa classe não conseguem acessar diretamente esses atributos. Evitando assim problemas de acesso indevido.

Chamamos a esses métodos getters e setters, respectivamente, e eles permitem que mantenhamos o conceito de encapsulamento aplicado aos membros da classe.

Programação orientada a objetos em Java
Encapsulamento

Métodos getters

O propósito de um método getter é obter um valor de um atributo declarado como private e permitir sua leitura de fora da classe.

Os métodos getters podem ser programados para exibir os dados em formatos apropriados.

Programação orientada a objetos em Java
Encapsulamento

Métodos setters

Um setter é um método que permite modificar o valor de um atributo da classe, geralmente um atributo que seja privado (que não seja acessível diretamente). Assim temos mais controle como os valores são atribuídos aos campos.
Os métodos setters podem ser programados para validar seus argumentos e rejeitar quaisquer tentativa de dados inválidos, como datas fora do intervalo, números negativos ou fora de escala, etc.

Programação orientada a objetos em Java
Encapsulamento

Exemplo na pratica

Programação orientada a objetos em Java

Sobrecarga de métodos e construtores (overload)

A sobrecarga (overloading) é usada para implementar métodos que realizam tarefas similares para parâmetros de tipos diferentes ou ainda para quantidade diferentes de parâmetros. Assim, podemos ter métodos com o mesmo nome na mesma classe, mas realizando suas tarefas de forma ligeiramente diferentes entre si.
Quando o método sobrecarregado é chamado, o compilador seleciona o método apropriado de acordo com o tipo, número e ordem dos parâmetros passados a ele. Também é possível fazer sobrecarga de construtores da mesma forma como em métodos.

E em tempo de execução como os métodos sobrecarregados são distintos entre si?

Programação orientada a objetos em Java

Sobrecarga de métodos e construtores (overload)

O compilador distingue entre os métodos sobrecarregados por sua **assinatura** – combinação do nome do método com o numero, tipo e ordem dos parâmetros. Não leva em consideração, porém, seu tipo e retorno.

Muito cuidado ao criar métodos sobrecarregados – nunca os crie usando listas de parâmetros idênticas!

Programação orientada a objetos em Java
Sobrecarga de métodos e construtores (overload)

Declarar método sobrecarregado

Para declarar métodos sobrecarregados basta criar dois ou mais métodos em uma classe que possuam o mesmo nome, mas numero, tipo e/ ou ordem de parâmetros distintos.

Exemplo na pratica

Exercicios