PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli

ORIENTAÇÃO A OBJETOS

Uma retomada conceitual

Aula 03-B

PARADIGMA DE PROGRAMAÇÃO

- Baseada na composição e interação entre diversas unidades de softwares (os objetos);
- Programas compostos por objetos com propriedades e operações;
- A estrutura dos dados é definida com as funções (métodos) que poderão ser executados;
- Usa-se, normalmente, a UML (Unified Modeling Language) para modelar soluções desse paradigma;
- A execução do código não é mais sequencial em 1 único arquivo.

PARADIGMA DE PROGRAMAÇÃO

Princípios:

- Qualquer coisa é um objeto;
- Objetos realizam tarefas por meio de requisição de serviços a outros objetos;
- Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares;
- A classe é um repositório para comportamento associado ao objeto;
- Classes são organizadas em hierarquias.

PARADIGMA DE PROGRAMAÇÃO

- O paradigma de orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados objetos. Cada objeto é responsável por realizar tarefas específicas. É pela interação entre os objetos que uma tarefa computacional é realizada.

- Um sistema de software orientado a objetos consiste em objetos **em colaboração** com o objetivo de realizar as funcionalidades desse sistema. Cada objeto é responsável por tarefas específicas. É graças à cooperação entre objetos que a computação do sistema se desenvolve.

CLASSE

- É o modelo geral. Uma classe é um tipo definido pelo usuário (e não um tipo primitivo).
- Classe é a representação genérica do que um objeto pode ter e do que o mesmo objeto pode fazer.
- Possui especificações (características e comportamentos) que a identifiquem.
- É um molde que será usado para construir objetos e representar elementos da vida real.
- Agrupamento de elementos que possuam as mesmas características e comportamentos

Características = Atributos

Comportamentos = Métodos

ATRIBUTOS

- São as características definidas em uma classe;
- "Variáveis" que todos os objetos de uma classe irão possuir. Podem possuir valores diferentes, mas pertencem a um mesmo molde (a uma mesma classe)
- Deve ser "global" : um atributo serve para TODOS os objetos de uma classe.
- Armazenam os dados do objeto.

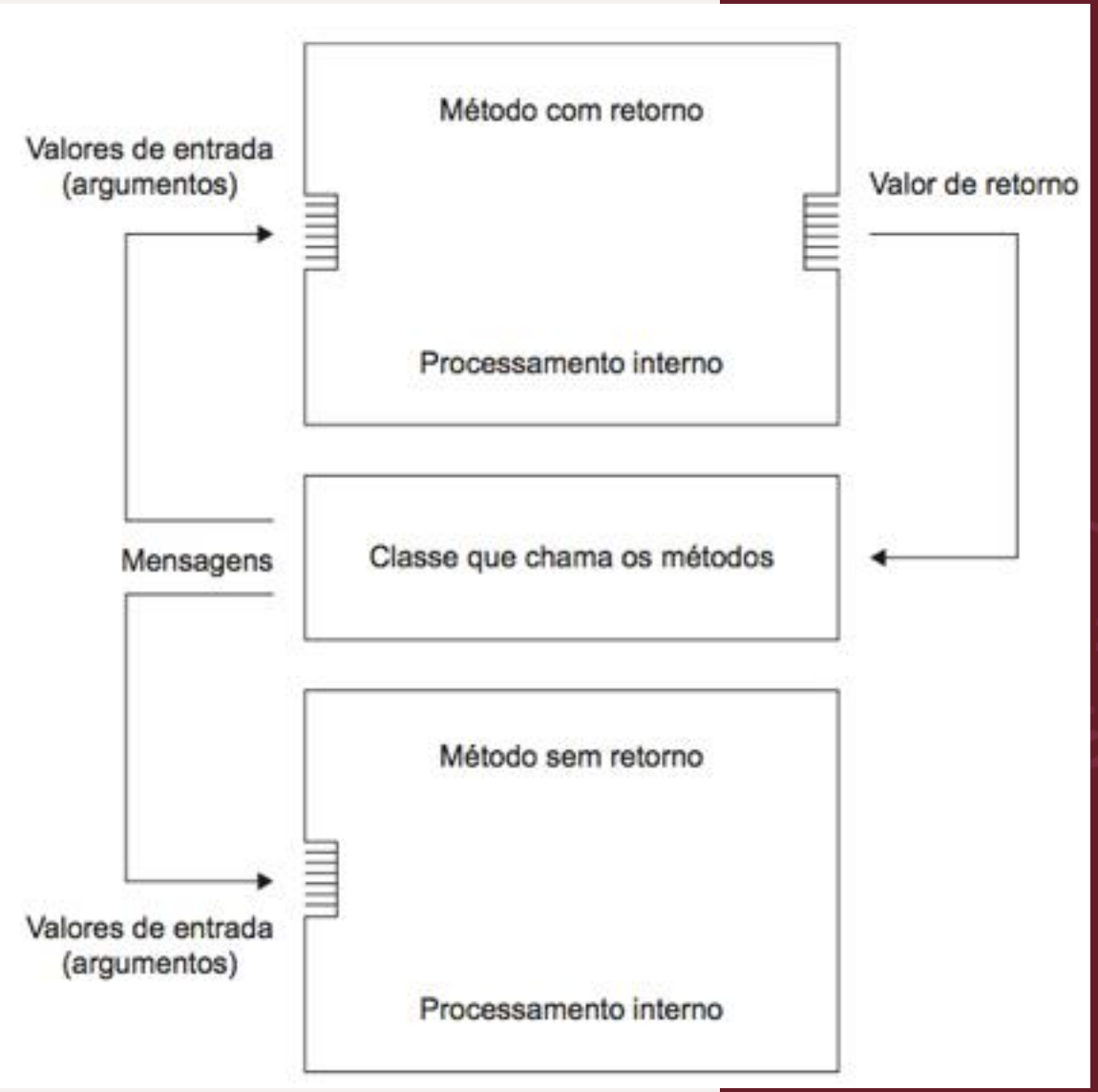
```
public class Pessoa {

    String nome;

    int idade;

    double altura;

    double peso;

}
```

MÉTODOS

- É através dos métodos que são definidas as operações que podem ser executadas com ou sobre um objeto. Popularmente, diz-se que os métodos definem o comportamento da classe.

- Um método é dividido em 3 partes:
  - Retorno do método
  - Nome do método
  - Parâmetros do método

- Podem exibir parâmetros na sua declaração, os quais serão fundamentais na sua implementação.



MODIFICADORES DE ACESSO

- define a visibilidade do método. Trata-se de uma forma de especificar se o método é visível apenas à própria classe em que está declarado ou pode ser visualizado (e utilizado) por classes externas.:
- **public:** o método é visível por qualquer classe. É o qualificador mais aberto no sentido de que qualquer classe pode usar esse método.
- **private:** o método é visível apenas pela própria classe. É o qualificador mais restritivo.
- **protected:** o método é visível pela própria classe, por suas subclasses e pelas classes do mesmo pacote.

EXEMPLO GERAL

```java
package cap06;
public class MetodosSemRetorno {
    public static void main(String args[]) {
        imprimir();
        imprimirTexto("Ola");
        mostrarQuadrado(10);
        somar(10, 20);
        mostrarMaior(10, 20, 30);
        mostrarSexoPorExtenso('F');
    }
    public static void imprimir() {
        System.out.println("Aprendendo a Linguagem Java");
    }
    public static void imprimirTexto(String texto) {
        System.out.println(texto);
    }
    public static void somar(int a, int b) {
        System.out.println(a + b);
    }
    public static void mostrarQuadrado(int valor) {
        System.out.println(Math.pow(valor, 2));
    }
    public static void mostrarMaior(int a, int b, int c) {
        System.out.println(Math.max(c, Math.max(a, b)));
    }
    public static void mostrarSexoPorExtenso(char sexo) {
        if (sexo == 'F') {
            System.out.println("Feminino");
        } else if (sexo == 'M') {
            System.out.println("Masculino");
        } else {
            System.out.println("Sexo desconhecido");
        }
    }
}
```

MÉTODO CONSTRUTOR

- É o primeiro método a ser executado quando um objeto de uma classe é instanciada;
- SEMPRE recebe o mesmo nome da classe;
- ACEITA sobrecarga (mais de um método construtor, com parâmetros diferentes).
- NÃO possui retorno (não devolve valores).

SOBRECARGA DE MÉTODOS

- É a possibilidade de existir mais de um método com o MESMO nome identificador.
- Item que deve ser diferencial entre esses métodos de mesmo nome:
  - Parâmetros (quantidade ou tipo de variáveis);

```
1  package cap06;
2  public class AreaComSobrecarga {
3      public static void main(String args[]) {
4          System.out.println("Área de um quadrado..." + calcularArea(3));
5          System.out.println("Área de um retangulo.." + calcularArea(3, 2));
6          System.out.println("Área de um cubo......." + calcularArea(3, 2, 5));
7      }
8      public static double calcularArea(int x) {
9          return (x * x);
10     }
11     public static double calcularArea(int x, int y) {
12         return (x * y);
13     }
14     public static double calcularArea(int x, int y, int z) {
15         return (x * y * z);
16     }
17 }
```

OBJETOS

- Representam os casos particulares e específicos de uma classe. São denominados instâncias de uma classe e são independentes entre si.
- Cada um possui a ”sua vida”, suas características próprias. Objetos semelhantes compartilham o mesmo "molde" (a mesma classe).
- Possui como tipo o NOME de uma classe.

OPERADOR NEW

- Classe pode ter tamanhos variáveis. Muitos atributos e muitos métodos. Objeto de uma classe possui TUDO o que a classe implementa.
- Reserva o espaço necessário na área de memória para um objeto
- O tamanho da área na memória é em função do tipo do objeto (da classe que objeto representa).
- OBS: objetos são do tipo referência (como são as String - não são do tipo primitivo).
- Referenciam endereços de memória (e não valor específico).

EM RESUMO...

- **Classe:** reúne as características comuns de todos os objetos – atributos e métodos;
- **Atributos:** são as características que a classe implementa; as propriedades dos objetos;
- **Métodos:** ações dos objetos (cálculos e funções);
- **Objetos:** são as coisas que queremos identificar / manipular na programação; representam as "variáveis" do tipo de uma classe específica.

NA PRÁTICA

- Diretórios da Arquitetura Restful
  - Entities
  - Dtos
  - Controllers
  - Services
  - Repositories
  - Utils

NA PRÁTICA

- Classe
- Atributos
  - Modificadores de acesso
- Métodos
  - Método Construtor
  - Getters / Setters
  - Sobrecarga de método
- Objetos

NA PRÁTICA - ROTEIRO

- Classe Perfil
- Inserir atributos
- Gerar métodos GETTERS / SETTERS

- Classe Perfil Service
- Criar método para cadastrar um novo perfil
- Criar método para listar todos os perfis

NA PRÁTICA - ROTEIRO

- Classe Equipe
- Inserir atributos
- Gerar métodos GETTERS / SETTERS

- Classe Equipe Service
- Criar método para cadastrar uma nova equipe
- Criar método para listar todas as equipes

NA PRÁTICA - ROTEIRO

- Classe Categoria
- Inserir atributos
- Gerar métodos GETTERS / SETTERS

- Classe CategoriaService
- Criar método para cadastrar uma nova categoria
- Criar método para listar todas as categorias

NA PRÁTICA - ROTEIRO

- Classe Usuário
- Inserir atributos
- Gerar métodos GETTERS / SETTERS

- Classe UsuarioService
- Criar método para cadastrar um novo usuário
- Criar método para listar todos os usuários

NA PRÁTICA - ROTEIRO

- Contexto AutenticacaoService
- Criar classes e métodos para autenticação de usuários

RECORDS / Registros

- São estruturas imutáveis;
- Uma vez declarado e preenchido valores, não podem mais ser alterados;
- DTOs: Data Transfer Objects
- São as estruturas que serão recebidas e devolvidas na comunicação com a API
- Devem implementar `Serializable`
  - Objeto é serializado / desserializado ao ser transmitido via rede de computadores

ANOTAÇÕES

- `@Data`: liga o lombok na classe e dispensa getters/setters
- `@Entity`: define uma entidade / tabela do banco de dados
- `@Table (name="", schema = "")` : mapeamento de qual tabela a classe se refere
- `@Serial`: mapeia um serialUID único para serialização de objetos

ANOTAÇÕES

- `@Id`: define o campo que será a chave primária
- `@SequenceGenerator(`

  `name = ”SEQ",`

  `sequenceName = ”schema.nome_seq",`

  `allocationSize = 1 )`

  : mapeia qual sequence será usada para popular o ID
- `@GeneratedValue (strategy = GenerationType.SEQUENCE, generator = "SEQ")` : define a forma de obtenção / geração da sequence

ANOTAÇÕES

- `@Column(name = ".....", length = 30, nullable = false)`: define a coluna com o tamanho e NOT NULL
- `@Column(name = ".....", columnDefinition = "TEXT")` : define a coluna para o tipo TEXT (área de texto)
- `@Temporal(TemporalType.DATE)` : deve vir definido junto com a coluna do tipo data
- `@JsonFormat(pattern = "dd/MM/yyyy")` : formata a exibição da coluna no JSON
- `@Enumerated(EnumType.STRING)` : para campos vindos de um ENUM
- `@Transient` : para atributos que não são colunas da entidade

ANOTAÇÕES

- `@ManyToOne(fetch = FetchType.LAZY)` : muitos para um
  - FetchType.LAZY : não busca no ato da consulta / apenas quando necessário
  - FetchType.EAGER: busca assim que o objeto é retornado do banco de dados
- `@OneToOne`: um para um
- `@OneToMany`: um para muitos
  - O atributo deve ser do tipo Array, Lista, Collection, etc.
- `@JoinColumn(name = ”id_nessa_tabela", referencedColumnName = "id_tabela_origem")` : relacionamento entre as tabelas

ANOTAÇÕES

- `@OneToMany`
- `@JoinTable(name = ”tabela_ligacao", schema = ”nome_schema", joinColumns = @JoinColumn(name = "id_tabela_atual_na_de_ligacao"), inverseJoinColumns = @JoinColumn(name = "id_outra_tabela_na_ligacao"))`

- `@JsonManagedReference`

ANOTAÇÕES - EXEMPLO

- `@OneToMany`
- `@JoinTable(name = ”usuario_equipe", schema = ”public",`

`joinColumns = @JoinColumn(name = "id_usuario"),`

`inverseJoinColumns = @JoinColumn(name = "id_equipe"))`

- `@JsonManagedReference`
- `private List<Equipe> equipes;`

ANOTAÇÕES

- `@Service`: define que uma classe será um serviço
- `@Autowired`: injeção de dependências controladas pelo Spring (evita a necessidade de ficar abrindo instâncias de objetos de classes chave)
- `@RestController` : define um controlador
- `@RequestMapping("/caminho")` : define a rota do controlador
- `@GetMapping("/listar/{cpf}")` : define um endpoint de consulta. {cpf} é o parâmetro necessário
- `@PathVariable` : lê o atributo informado na rota
- `@RequestPart` : informa a necessidade de um arquivo / classe de complementação enviado para a rota (endpoint)
- `@RequestBody`: informa a necessidade de receber dados (como um JSON)

ANOTAÇÕES

- `@PostMapping("/add/{cpf}")`: endpoint para gravação de dados
- `@DeleteMapping("/remove/{cpf}/{id}")` : endpoint para remoção de dados
- `@ResponseBody ResponseEntity` : retorno de um dto para o front-end

- `@Column (name="", insertable=false)`
  `insertable` = indica que uma coluna não deve ser incluída nas instruções SQL INSERT. Isso significa que o valor dessa coluna não será fornecido quando uma nova entidade é inserida no banco de dados.
  `updatable` = indica que uma coluna não deve ser atualizada nas instruções SQL UPDATE. Isso significa que o valor dessa coluna não será fornecido quando uma entidade é atualizada no banco de dados.

BANCO DE DADOS - JPA

- A comunicação / manipulação com banco de dados é feita via Hibernate JPA;
- O framework se encarrega de executar ações básicas e fundamentais na manipulação de dados
- Uma interface deve ser declarada + `extends` de `JpaRepository`

- Exemplo:

```
public interface AlunoRepository extends JpaRepository<Aluno, Long> {

}
```

PROGRAMAÇÃO AVANÇADA ORIENTADA A OBJETOS

Prof. Me. Renan Caldeira Menechelli