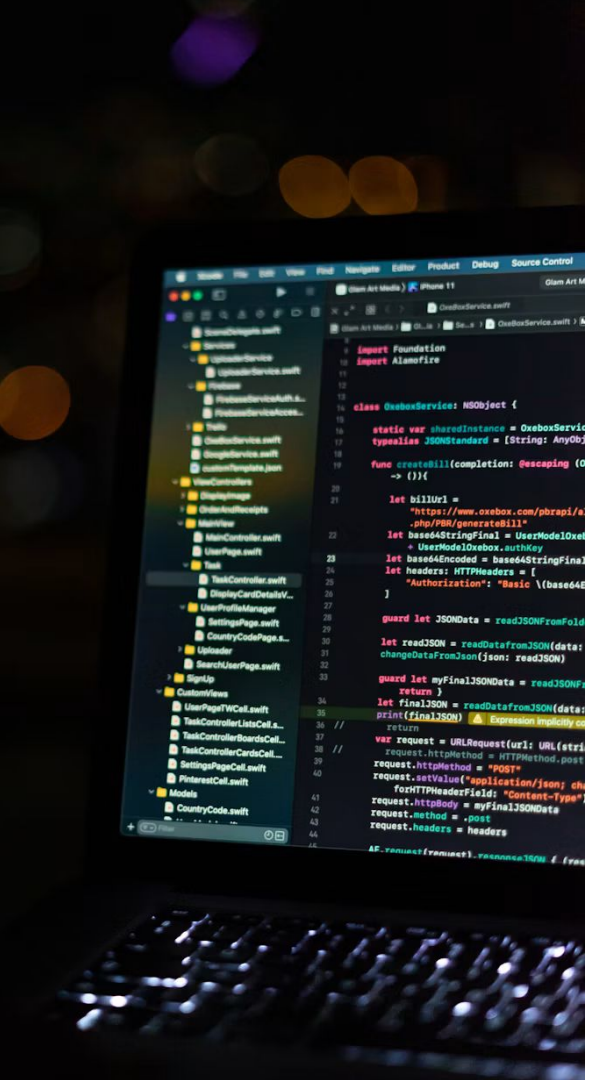


5954018 - Programação Orientada a Objetos

Aula 5 - Classes e Objetos

Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Objetivos de Aprendizagem | Autoavaliação de progresso

Utilize esta lista como um guia para verificar se você consolidou os conceitos essenciais desta aula.

01

Entender o conceito de classe e objeto.

02

Compreender como declarar classes em Java.

03

Compreender o conceito de construtor em Java e o seu funcionamento.

04

Entender como instanciar e manipular objetos em Java.

Créditos: **slides baseados no material de aula original do Prof. Clever Farias.**

Conceito de Classe

CONCEITO

O que é uma Classe?

Uma classe funciona como um **modelo, molde ou blueprint** a partir do qual objetos individuais são criados.

Ela centraliza a definição da estrutura que os futuros objetos compartilharão:

- **Dados (Atributos):** O que os objetos da classe sabem ou armazenam.
- **Comportamentos (Métodos):** O que os objetos da classe podem fazer.

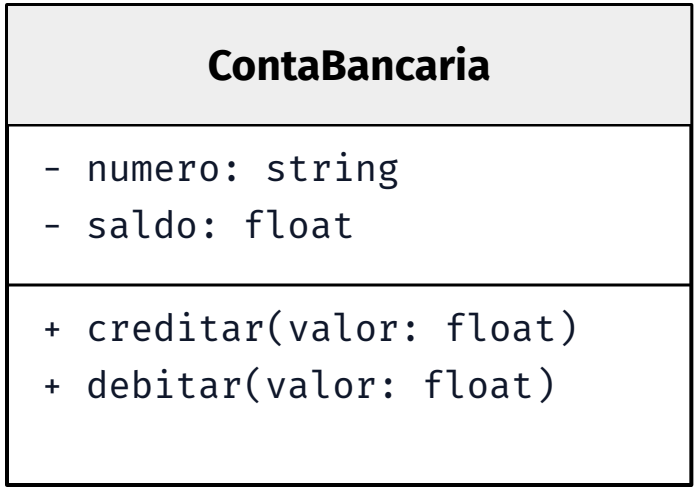


Diagrama de Classe UML para Conta Corrente

Declaração de Classe em Java

Para Classes: Criar é Declarar

Em Java, uma classe é declarada utilizando a palavra-chave `class`.

Regras importantes de padronização:

- **PascalCase:** O nome da classe deve sempre iniciar com letra maiúscula (ex: `ContaBancaria`).
- **Nome do Arquivo:** O arquivo de código fonte deve possuir o mesmo nome da classe pública (ex: `ContaBancaria.java`).
- Geralmente, uma classe por arquivo.

```java
/**
 * Representa uma conta
 * bancária simples.
 */
public class ContaBancaria {
    String numero;
    float saldo;
}
```

Exemplo Prático de Sintaxe em Java

Atributos e Variáveis

Há diferentes tipos de variáveis em Java:

- **Atributos de Classe,** também chamados de campos.
- **Variáveis em um método ou bloco de código**, também chamadas de variáveis locais.
- **Variáveis na declaração de métodos**, também chamadas de parâmetros

Importante: Mesmo quando não especificado, todas as variáveis em Java são inicializadas com um valor padrão.

```java
/**
 * Representa uma conta
 * bancária simples.
 */
public class ContaBancaria {
    String numero;
    float saldo;
}
```

Exemplo Prático de Sintaxe em Java

Métodos

CONCEITO

O que são Métodos?

Métodos representam ações que podem ser executadas por um objeto. Eles podem receber parâmetros e também alterar o estado do objeto.

SINTAXE

Parâmetros e Estrutura

- A lista de **parâmetros** aparece entre parênteses, separada por vírgulas.
- Cada parâmetro é definido pelo seu **tipo** e **identificador**.
- Os parênteses são obrigatórios mesmo que não haja parâmetros.
- O corpo do método deve ser delimitado por chaves `{ }`.

```java
/**
 * Representa uma conta
 * bancária simples.
 */
public class ContaBancaria {
    String numero;
    float saldo;

    /**
     * Adiciona valor ao saldo.
     */
    public void creditar(float valor) {
        saldo = saldo + valor;
    }
}
```

Exercício Rápido

Acrescente o método **debitar** à declaração da classe **ContaBancaria**. O método deve **validar** se há dinheiro suficiente.

```
/**
 * Representa uma conta
 * bancária simples.
 */
public class ContaBancaria {
    String numero;
    float saldo;

    /**
     * Adiciona valor ao saldo.
     */
    public void creditar(float valor) {
        saldo = saldo + valor;
    }
}
```

Construtor: Método Especial

CONCEITO

O que são Construtores?

Construtores inicializam objetos no momento da criação. São responsáveis pela inicialização das **variáveis de estado** (atributos) de cada objeto.

Construtor é executado **antes** que qualquer outro código em outros métodos seja executado.

CARACTERÍSTICAS

- Eles têm o **mesmo nome da classe**.
- Eles **não possuem tipo de retorno** (nem mesmo `void`).
- São chamados exclusivamente através do operador `new`.

```java
/**
 * Representa uma conta
 * bancária simples.
 */
public class ContaBancaria {
    String numero;
    float saldo;

    /**
     * Inicializa uma conta bancária.
     */
    public ContaBancaria(String numeroInicial) {
        numero = numeroInicial;
        saldo = 0.0f;
    }

    public void creditar(float valor) {
        saldo = saldo + valor;
    }
}
```

O Conceito de Objeto

CONCEITO

O que é um Objeto?

Um **objeto** é uma instância de uma classe.

A classe funciona como o **modelo** (molde), definindo os atributos e comportamentos.

O objeto é uma **conta concreta** criada a partir desse modelo, possuindo sua própria identidade e valores de estado.

Objetos e Instanciação

CONCEITO

Para Objetos: Criar é Instanciar

Instanciar significa **criar** um objeto na memória a partir de uma classe.

Como funciona:

- Feita através da palavra-chave `new`.
- Aplica-se a um dos construtores definidos para a classe: `new Construtor()`

Referências de Objetos:

- A instância criada é associada a uma **variável de referência**.
- Uma referência armazena a **localização do objeto na memória**.
- É utilizada posteriormente para acessar e manipular o objeto.

```java
/**
 * Programa principal: BancoMain.java
 */
public class BancoMain {
    /**
     * Executa o programa.
     */
    public static void main(String[] args) {
        ContaBancaria conta =
          new ContaBancaria("001");
        conta.saldo = 100.0f;
        conta.creditar(50.0f);
    }
}
```

Objetos e Instanciação (cont.)

Instâncias Únicas

Cada instância de uma classe é única na memória. Dois objetos criados com os mesmos parâmetros **não são o mesmo objeto**, pois suas referências apontam para endereços distintos.

Referências Independentes

Variáveis de referência operam de forma independente. Se uma delas for alterada para apontar para outro local, **a outra permanece intacta** apontando para a instância original.

```java
/**
 * Programa principal: BancoMain.java
 */
public class BancoMain {
    /**
     * Executa o programa.
     */
    public static void main(String[] args) {
        ContaBancaria adaConta =
                new ContaBancaria("001");
        ContaBancaria bobConta =
                new ContaBancaria("001");

        System.out.println(adaConta == bobConta);
    }
}
```

Exercício Rápido: Instanciação

PRÁTICA

Escreva o código em Java para criar e configurar duas instâncias independentes.

Requisitos de cada conta:

- **Número diferente:** garanta identificadores únicos na memória.
- **Saldo diferente:** atribua valores iniciais distintos para cada conta.
- **Operação de crédito:** realize um depósito chamando o método adequado.

Dica: Use `System.out.println(conta.saldo);` para imprimir os saldos das contas.

Objetos e Instanciação (cont.)

Valor null

Objetos precisam ser instanciados **antes** de serem manipulados.

A referência de um objeto não inicializado aponta para o “vazio”.

A palavra chave `null` pode ser utilizada para garantir uma “inicialização” para a referência do objeto. Em alguns casos, o compilador exige que esta atribuição seja feita.

```
/**
 * Programa principal: BancoMain.java
 */
public class BancoMain {
    /**
     * Executa o programa.
     */
    public static void main(String[] args) {
        ContaBancaria conta = null;
        System.out.println(conta.saldo);
    }
}
```

Tudo é Objeto em Java?

Objetos & Classes

A estrutura padrão da linguagem Java

- A maior parte dos elementos do programa é organizada como **objetos**.
- Criados na memória a partir de suas respectivas **classes**.
- Manipulados de forma segura através de **referências**.

Exceção: Primitivos

Valores guardados diretamente na memória

- Tipos básicos como `int`, `float`, `double` e `boolean`.
- **Não são objetos** e não são instanciados a partir de classes.
- Armazenam seus valores de forma direta e eficiente.

Modificadores de Acesso

Encapsulamento

Uma das principais vantagens da orientação a objetos.

Encapsulamento **protege o estado interno** do objeto. Os **detalhes internos** e os campos de uma classe devem ser protegidos do acesso externo direto.

Encapsular campos e definir métodos específicos para a manipulação controlada desses atributos dentro da própria classe.

Há **quatro tipos de modificadores de acesso** aplicáveis a campos e métodos:

`public`

Garante que o campo ou método declarado pode ser acessado e executado a partir de **qualquer classe**.

`private`

Garante que o campo ou método declarado só pode ser acessado a partir da **classe onde foi declarado**.

`protected`

Similar ao `private`, exceto quando ocorre a **especialização de classes**.

(tema futuro de aula)

Default

(Ausência de modificador)

Similar ao `public`, mas somente no contexto de um **mesmo pacote**.

(tema futuro de aula)

Encapsulamento

Atributos Protegidos

Devem ser protegidos contra acesso externo direto. Por isso, declaramos como `private`.

Acesso Controlado

Métodos públicos garantem uma interface segura e controlada para interagir com o objeto.

```
public class ContaBancaria {
    private float saldo;

    public void creditar(float valor) {
        if (valor > 0.0f) {
            saldo = saldo + valor;
        }
    }
}
```

Encapsulamento (cont.)

Métodos de Consulta (Getters)

Permitem a leitura segura do estado interno (atributos) sem permitir sua alteração direta.

Métodos de Alteração (Setters)

Permitem a alteração segura do estado interno (atributos).

```java
public class ContaBancaria {
    private String numero;

    /** Retorna o número da conta. */
    public String getNumero() {
        return numero;
    }

    /** Altera o número da conta. */
    public void setNumero(String numero) {
        if (numero != null && !numero.isEmpty()) {
            this.numero = numero;
        }
    }
}
```

Exercício Rápido

01. Encapsular

Torne os atributos de `ContaBancaria` privados.

02. Tentar Acesso

Depois, tente acessar `conta.saldo` no método `main`.

03. Analisar Erro

Observe atentamente o erro produzido pelo compilador.

Encapsulamento (cont.)

Regras de Negócio

Controle de Estado

Métodos podem proteger o estado do objeto.

Exemplo: Validação de Créditos e Débitos

- Créditos negativos não devem alterar o saldo.
- Débitos maiores que o saldo não devem ser permitidos.

Exercício Rápido

Implemente as validações ao lado e verifique os seguintes casos:

- creditar -50.0f;
- debitar 30.0f;
- debitar 500.0f

```java
/**
* Adiciona valor positivo ao saldo.
*/
public void creditar(float valor) {
    if (valor > 0.0f) {
        saldo = saldo + valor;
    }
}

/**
* Retira valor se houver saldo.
*/
public void debitar(float valor) {
    if (valor > 0.0f && valor <= saldo) {
        saldo = saldo - valor;
    }
}
```

Exercício: Crie a classe que define um Pokémon (simplificado)

Pikachu #0025

Electric

Atributos:
nome, tipo, pontos de vida (HP), ataque (ATK) e defesa (DEF).

dittobase.com

Sobreposição de Métodos

CONCEITO

Capacidade de definir múltiplos métodos com o **mesmo nome** na mesma classe, desde que possuam assinaturas distintas.

REGRAS DA ASSINATURA

- **O que define:** Composta pelo nome do método e a lista de tipos de seus parâmetros.
- **Diferenciação:** O compilador determina qual método chamar com base nos argumentos passados.
- **Retorno:** O tipo de retorno **não** faz parte da assinatura do método.

```java
/**
 * Representa uma conta
 * bancária simples.
 */
public class ContaBancaria {
    String numero;
    float saldo;

    public void creditar(float valor) {
        saldo = saldo + valor;
    }

    public void creditar(int valorInteiro) {
        saldo = saldo + valorInteiro;
    }

    public void creditar() {
        saldo += 100;
    }

}
```

Sobreposição de Construtor

SOBREPOSIÇÃO

Vários Construtores?

Uma classe pode ter vários construtores, desde que a assinatura destes sejam distintas.

Toda classe deve ter **pelo menos** um construtor.

O compilador Java automaticamente define um construtor *default*:

- Sem argumentos (e com corpo vazio)
- Para qualquer classe que não tenha um construtor *default*.

```
public class ContaBancaria {
    String numero;
    float saldo;

    public ContaBancaria(String numeroInicial) {
        numero = numeroInicial;
        saldo = 0.0f;
    }

    public ContaBancaria(
          String numeroInicial,
          float saldoInicial) {
        this.numero = numeroInicial;
        this.saldo = saldoInicial;
    }
}
```

Manipulação de Objetos

CONCEITO

Auto-referência do Objeto

Cada objeto pode acessar sua própria referência usando a palavra-chave `this`.

UTILIDADE

- **Diferenciação:** Ajuda a diferenciar atributo e parâmetro com o mesmo nome (evita sombreamento).
- **Clareza:** Deixa explícito que estamos acessando o estado interno do objeto atual.
- **Escopo:** Representa o próprio objeto que está executando o método ou construtor.

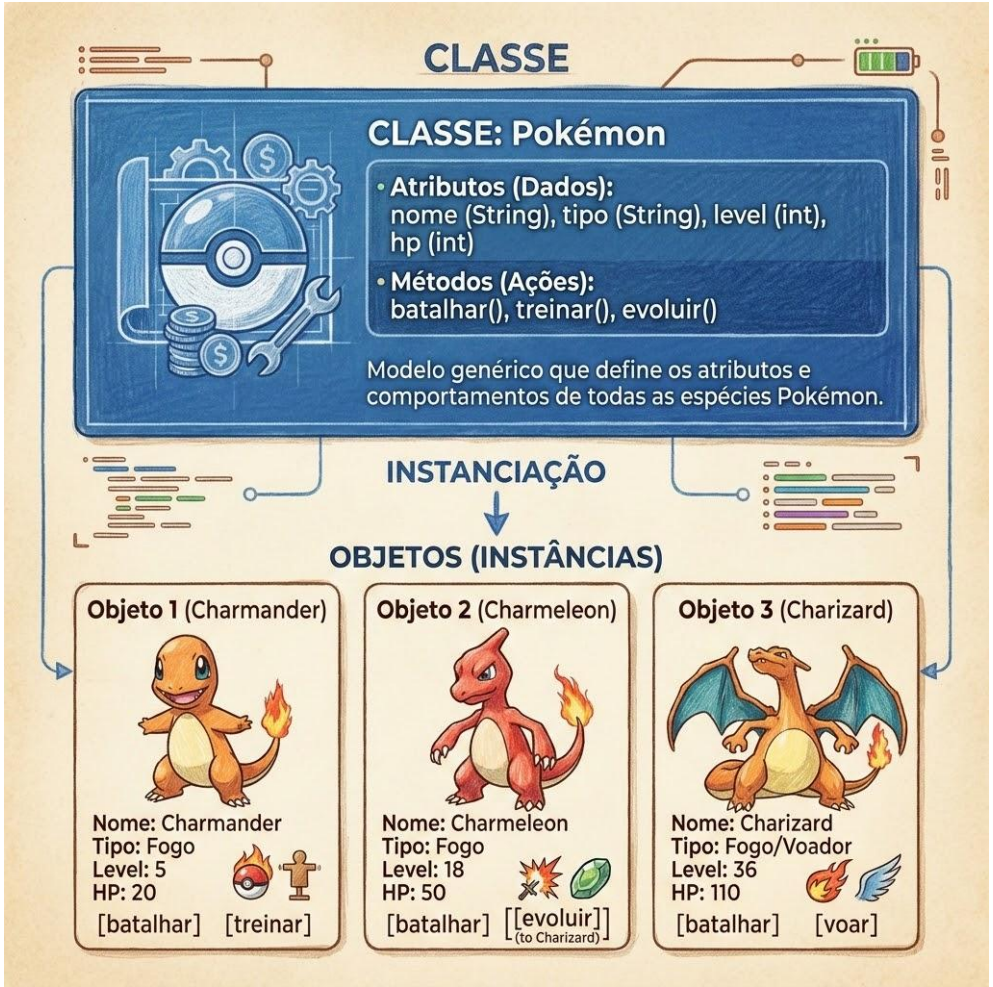
```
public class ContaBancaria {
    private String numero;
    private float saldo;

    /** Inicializa uma conta bancária. */
    public ContaBancaria(String numero,
      float saldo) {
        this.numero = numero;
        this.saldo = saldo;
    }
}
```

Exercício

Modifique a classe Pokemon para incluir:

- Um novo atributo: **nível**
- Um método chamado **exibirDados()** que mostra na tela: nome, tipo e nível
- Um método chamado **receberDano(int dano)** que reduz os pontos de vida do Pokémon.
- Um novo construtor que recebe, além do nome e do tipo, o nível do Pokémon.

Use a classe que você declarou para criar ao menos **3 Pokémon** de sua preferência. Dica: Consulte informações sobre Pokémon em https://www.dittobase.com/pokedex



Conclusão

Resumo do Aprendizado

Nesta aula, criamos a base para programar com classes e objetos em Java.

Classes e Estruturas
Classes definem atributos e métodos.

Instanciação
Objetos são criados com `new`.

Inicialização
Construtores inicializam objetos.

Encapsulamento
Modificadores controlam acesso.

CONTEÚDO EXTRA

Material Adicional

Acesse materiais complementares, guias de estudo e referências adicionais no ambiente virtual para aprofundar os tópicos desta aula.

Praticando Java: Orientação a Objetos com classes, atributos e métodos

Classes e objetos Java

Rebuilding Pokémon with Object Oriented Programming

Dúvidas e Discussão