# *Orientação a Objetos e Java*

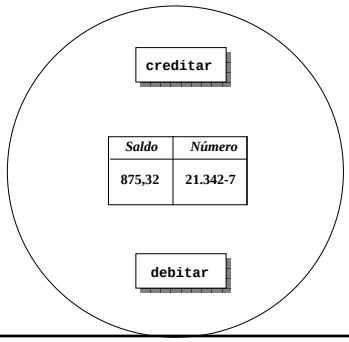
Sérgio Soares
sergio@dsc.upe.br

# *Objetos, classes, métodos e atributos*

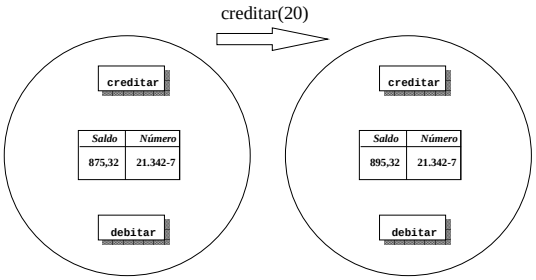
## *Programação Orientada a Objetos*

- Foco nos dados (objetos) do sistema, não nas funções
- Estruturação do programa é baseada nos dados, não nas funções
- As funções mudam mais do que os dados

## *Objeto Conta Bancária*




## *Estados do Objeto Conta*



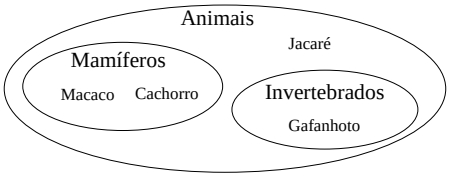

## *Objetos*

- *Objetos*
  - comportamento + características
  - métodos + atributos
  - estado encapsulado

## *Classes*

- *Classes*

  agrupamento de objetos do mesmo tipo




## *Definindo Classes em Java*

```
public class NomeDaClasse {
    CorpoDaClasse
}
```

O corpo de uma classe pode conter
- atributos
- métodos
- construtores (inicializadores)
- outras classes...

## *Estrutura mínima de um programa em Java*

```
public class <nome> {
    public static void main (String[] args) {
        <declarações>
        <comandos>
    }
}
```

**Onde, `main`: método por onde se inicia a execução**
**`public`: parâmetro de acesso**
**`static`: indica que `main` se aplica à classe**
**`void`: indica que `main` não retorna um valor**

## *Exemplo*

```
public class LeImprime {
    /** Lê e imprime um string */
    public static void main(String[] args) {
        String nome;
        nome = Util.readStr();
        System.out.println(nome);
    }
}
```

## *Definindo Atributos em Java*

```
public class Livro {
    private int anoDePublicacao;
    private int numeroDePaginas;
    private String titulo;
    ...
}
```

- cada atributo tem um tipo específico que caracteriza as propriedades dos objetos da classe
- **`int`** e **`String`** denotam os tipos cujos elementos são inteiros e strings

## *Tipos em Java*

- Primitivos
  - **`char`**
  - **`int`**
  - **`boolean`**
  - **`double`**
  - **...**
- Referência
  - **classes (`String`, `Object`, `Livro`, `Conta`, etc.)**
  - **interfaces**
  - **arrays**

Os elementos de um tipo primitivo são valores, enquanto os elementos de um tipo referência são (referências para) objetos!

## *Strings (`String`)*

- Não é um tipo primitivo e sim uma classe
- Literais: **`“” “a” “DSC \n UPE \n”`**
- Operadores: **`+`** (concatenação)

ex.: **`“maio ” + “ de ” + 99`** = **`“maio de 99”`**

Note a conversão de inteiro para string

Há uma conversão implícita para todos os tipos primitivos

## *Mais operadores sobre strings*

- Comparação (igualdade) de dois strings `a` e `b`

```
String a ...
String b ...
a.equals(b) ou b.equals(a)
```

- Tamanho de um string **`a`**

```
a.length()
```

## *Information Hiding*

```
public class Livro {
    private int anoDePublicacao;
    ...
}
```

*A palavra reservada **`private`** indica que os atributos só podem ser acessados (isto é, lidos ou modificados) pelas operações da classe correspondente*

## *Information Hiding e Java*

- Java não obriga o uso de **`private`**, mas vários autores consideram isto uma pré-condição para programação orientada a objetos
- O bug do ano 2000 e **`private`**...
- Grande impacto em extensibilidade
- Usem **`private`**!

## *Definindo Atributos em Java*

```
public class Pessoa {
    private int anoDeNascimento;
    private String nome, sobrenome;
    private boolean casado = false;
  ...
}
```

- vários atributos de um mesmo tipo podem ser declarados conjuntamente
- podemos especificar que um atributo deve ser inicializado com um valor específico

## *Definindo Métodos em Java*

```
public class Conta {
    private String numero;
    private double saldo;

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

Um método é uma operação que realiza ações e modifica os valores dos atributos do objeto responsável pela sua execução

## *Definindo Métodos em Java*

```
public class Conta {
    ...
    public void debitar(double valor) {
       saldo = saldo - valor;
    }
}
```

parâmetros do método

corpo do método

tipo de retorno

Por quê o método `debitar` não tem como Parâmetro o número da conta?

## *Definindo Métodos em Java*

- O tipo do valor a ser retornado pelo método
- Nome do método
- Lista, possivelmente vazia, indicando o tipo e o nome dos argumentos a serem recebidos pelo método

Usa-se **`void`** para indicar que o método não retorna nenhum valor, apenas altera os valores dos atributos de um objeto

## *Definindo Métodos em Java*

```
public class Conta {
    private String numero;
    private double saldo;

    public String getNumero() {
       return numero;
    }
    public double getSaldo() {
       return saldo;
    }
    ...
}
```

Os métodos que retornam valores como resultado usam o comando **`return`**

## *O Corpo do Método*

- Comandos que determinam as ações do método
- Estes comandos podem
  - realizar simples atualizações dos atributos de um objeto
  - retornar valores
  - executar ações mais complexas como se comunicar com outros objetos

## *Comunicação entre objetos*

- Os objetos se comunicam para realizar tarefas
- A comunicação é feita através da troca de mensagens ou chamada de métodos
- Cada mensagem é uma requisição para que um objeto execute uma operação específica

```
conta.creditar(45.30)
```

variável contendo referência para objeto

nome do método a ser executado

## *Imprimindo na tela*

```
public class Conta {
  private String   numero;
  private double   saldo;

  public void imprimirSaldo() {
    System.out.println("Conta: " +
       numero + " Saldo: R$" + saldo);
  }
  ...
}
```

concatenação de **`String`** e conversão de tipos

A tela do computador é representada em Java por um objeto especial, armazenado na variável **`System.out`**

## *Imprimindo na tela*

O código de impressão na tela faz parte da GUI do sistema

e não deve ser misturado ao

código inerente ao negócio, como acontece no exemplo anterior

## *Exercício*

- Implemente o método **`transferir`** da classe **`Conta`**, para realizar a transferência de uma conta para outra

Dica: a palavra reservada **`this`** denota uma referência para o objeto que está executando o método no qual ela se encontra

## *Exercício*

- Utilizando apenas os conceitos ilustrados até aqui, defina parcialmente em Java as classes que fazem parte dos sistemas sendo desenvolvidos