Especialização em Engenharia de Software

Programação Orientada a Objetos

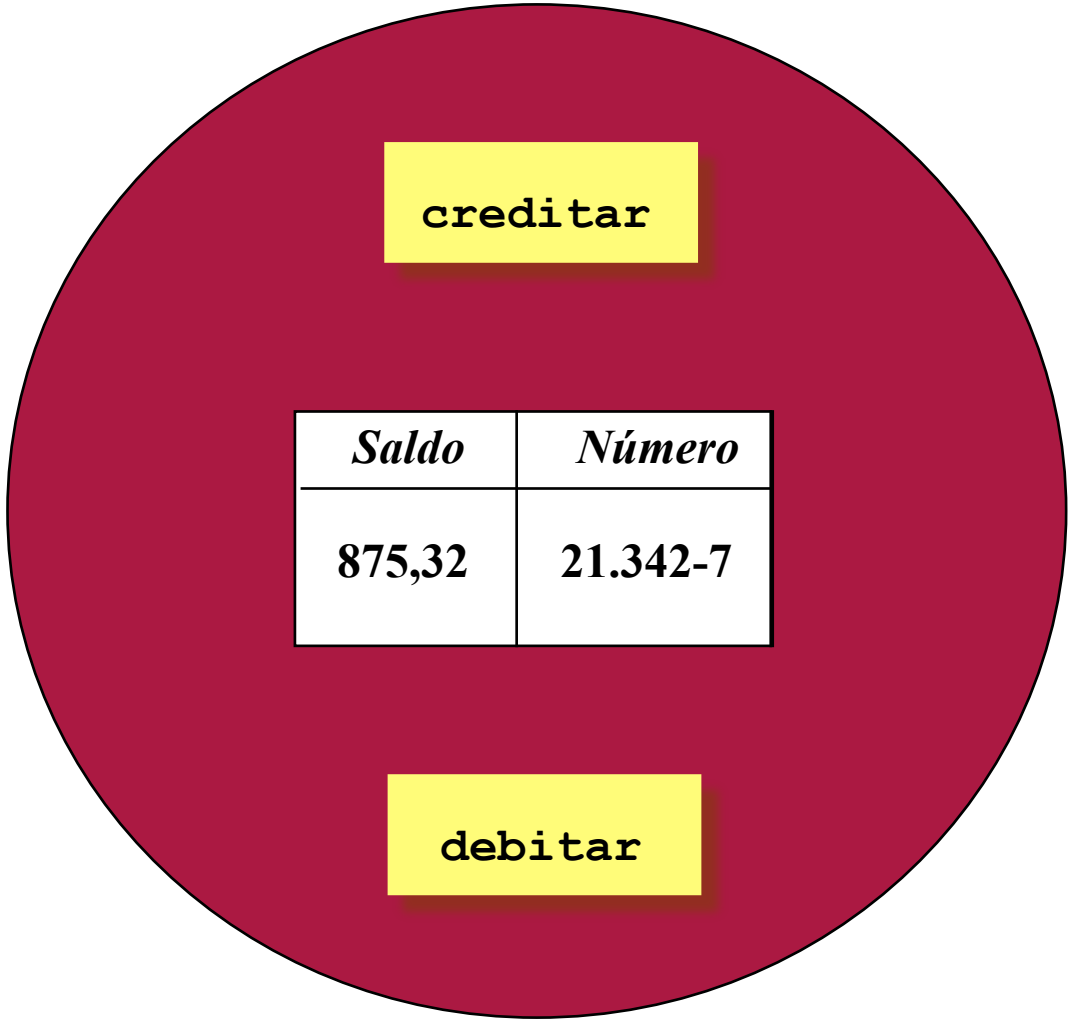
Conceitos gerais e array

Sérgio Soares
scbs@cin.ufpe.br

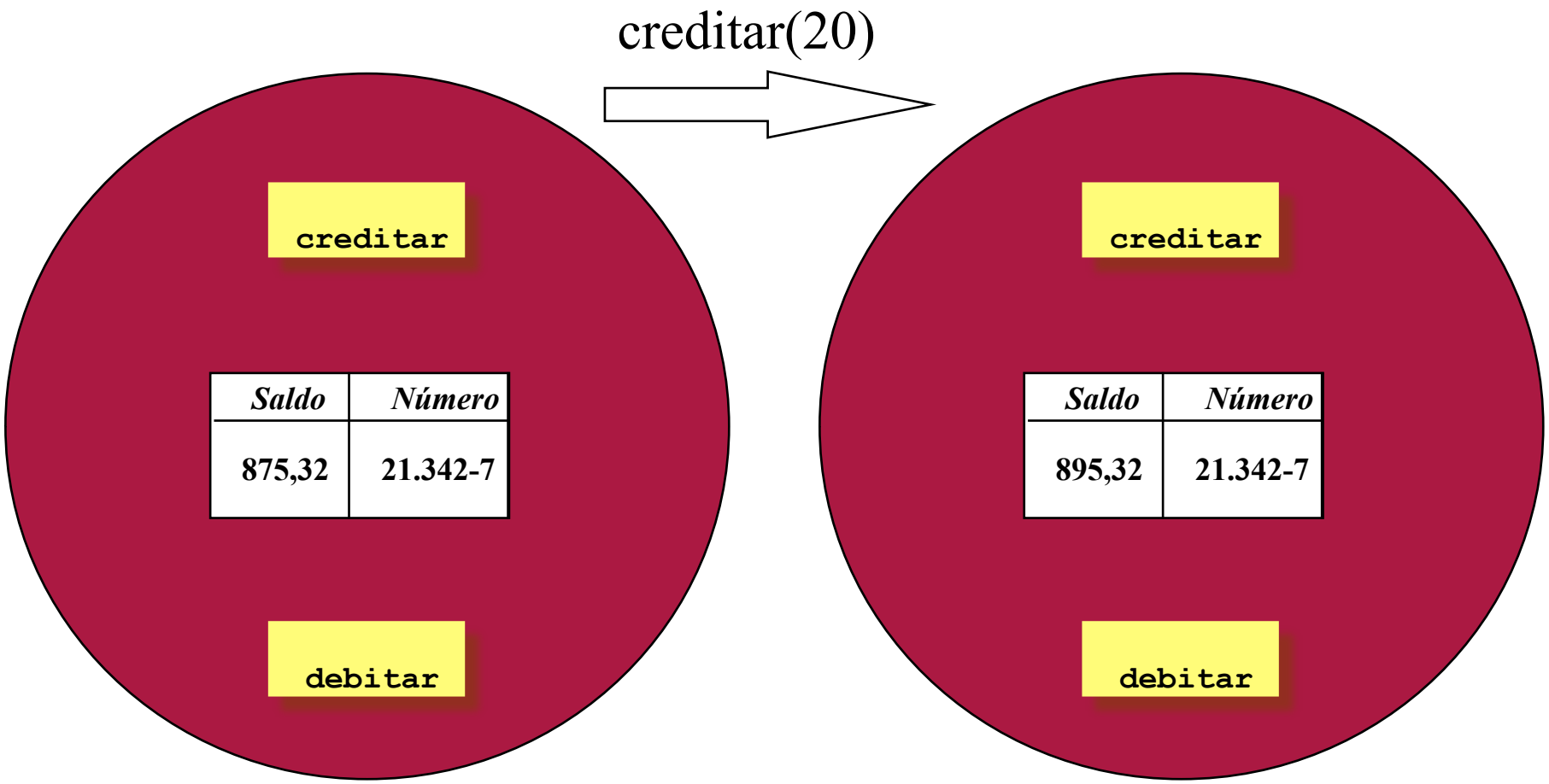
Programação Orientada a Objetos

- 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

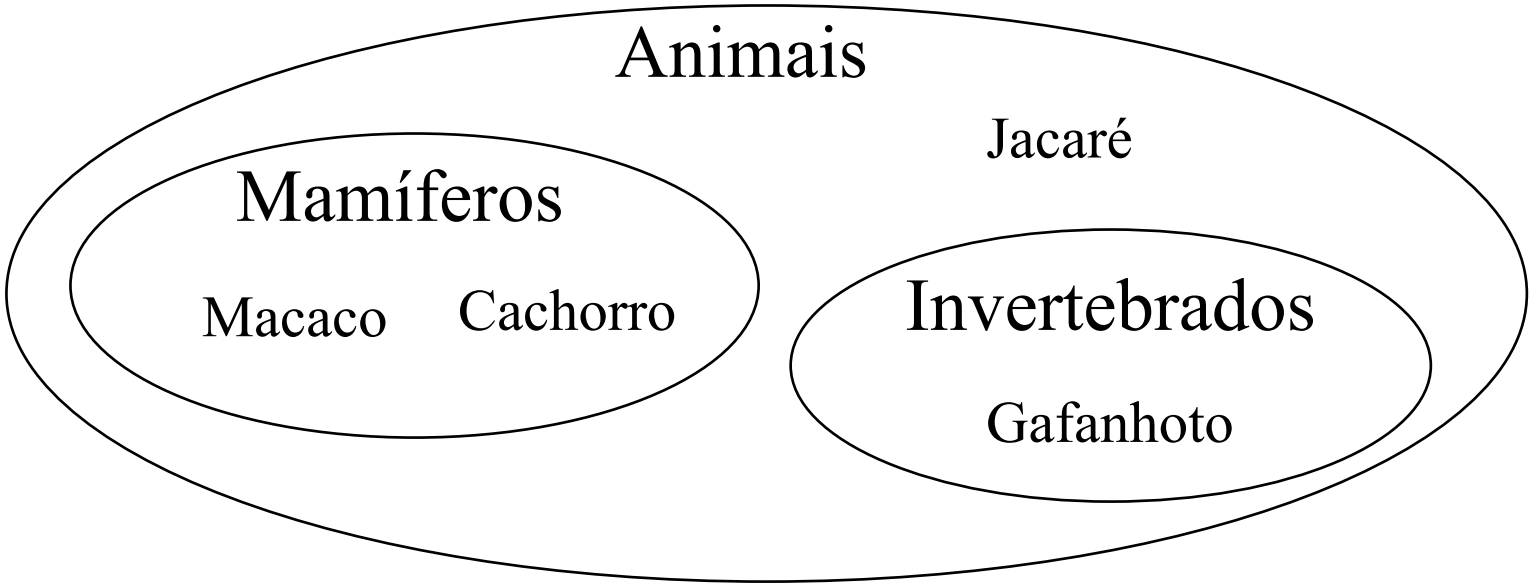
  métodos + atributos

  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

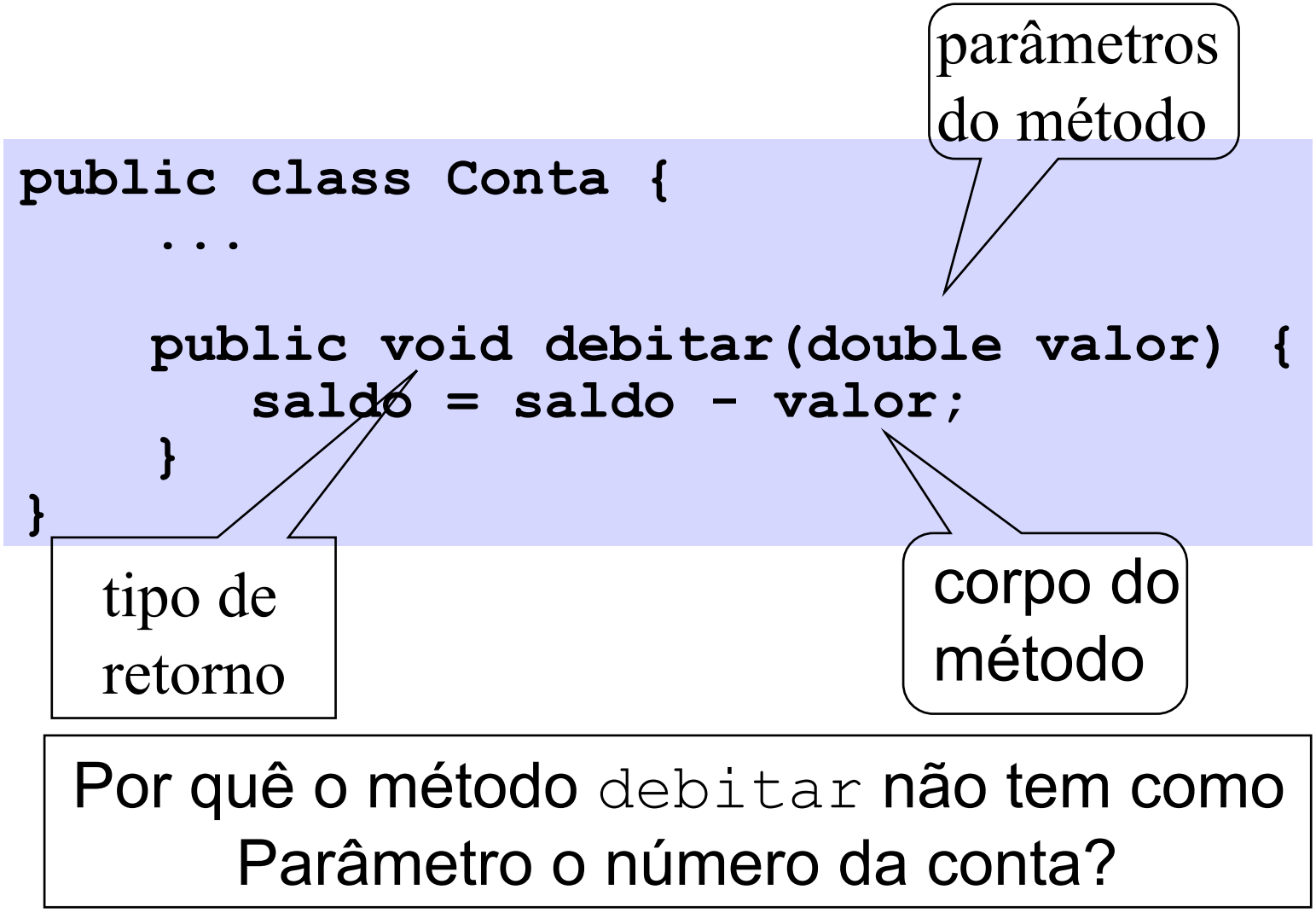
Definindo Métodos em Java

```
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



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 projetos da disciplina

Construtores

Além de métodos e atributos, o corpo de uma classe pode conter

construtores

definindo como os atributos de um objeto devem ser inicializados

```
<nome da classe> (<lista de parâmetros>) {
   <corpo do construtor>
}
```

Construtor default

- Um construtor sem parâmetros

```
Conta() {
   saldo = 0; ...
}
```

- Caso não seja definido um construtor, um construtor implícito *default*, equivalente a

```
<nome da classe>(){}
```

é fornecido, inicializando os atributos com seus valores *default*

Valores default para atributos

- **0 para `int`, `double`, etc.**
- **`false` para `boolean`**
- **`null` para tipos referência**

`null` denota uma referência nula, não existente, para um objeto de qualquer tipo

Outros construtores

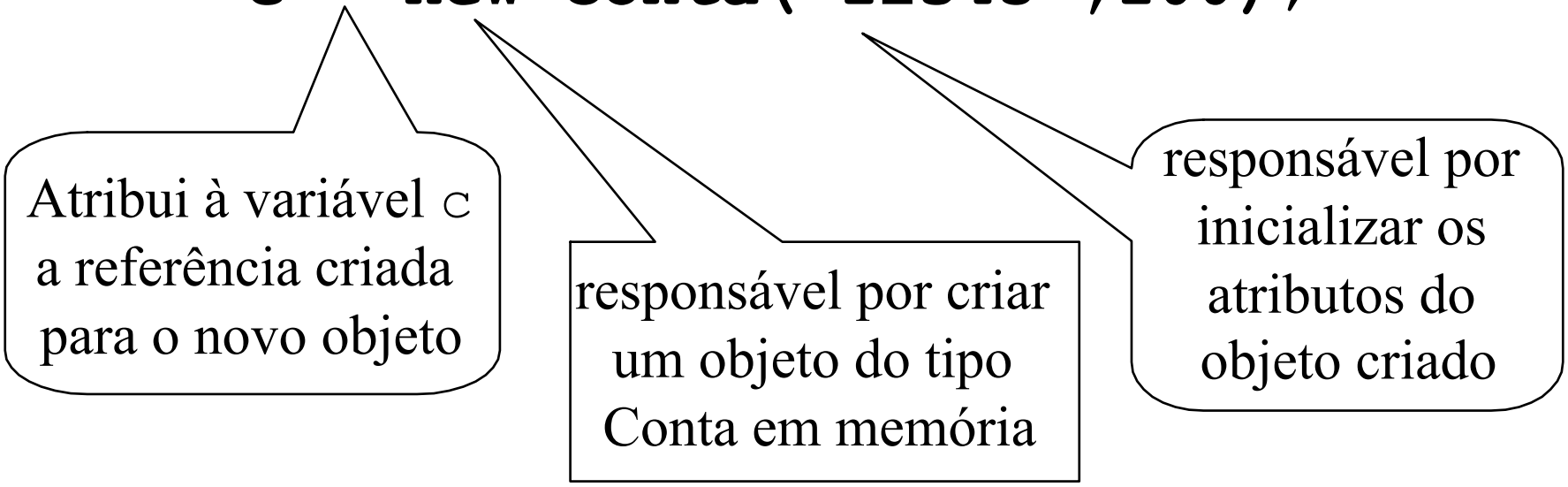
```
public class Conta {
   ...
   public Conta(String numeroConta,
                double saldoInicial) {
      numero = numeroConta;
      saldo = saldoInicial;
   }
}
```

Neste caso, o construtor implícito é descartado!

Criação de objetos

- Um objeto é criado através do operador `new`

```
Conta c;...
c = new Conta("12345",100);
```



`new` ***<nome da classe>(lista de argumentos)***

Referências

Objetos são manipulados através de referências

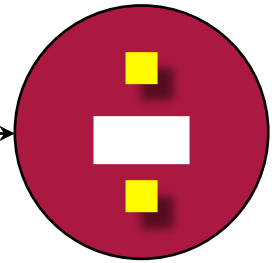
```
Conta c;

c = new Conta("1287",0);

c.getSaldo();
```

```
c == null ?
```

c



envia a mensagem `getSaldo()` ao objeto referenciado pela variável `c`

Aliasing

Mais de uma variável armazenando a mesma referência para um dado objeto

```
Conta a = new Conta("123-4",34);
Conta b;

b = a;

b.creditar(100);
System.out.println(a.getSaldo());
```

a e b passam a referenciar a mesma conta

qualquer efeito via b é refletido via a

Remoção de objetos

- Não existe mecanismo de remoção explícita de objetos da memória em Java (`free()` de C++)
- O *Garbage Collector* de Java elimina estes objetos da memória quando não são mais referenciados
- É possível liberar recursos quando o objeto está na iminência de ser destruído

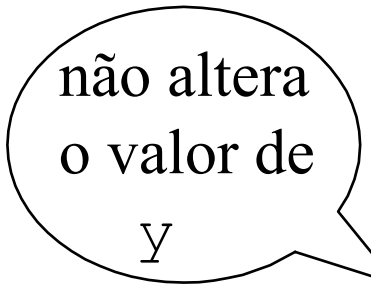
```
public class Conta {
    public void finalize() {
        ...
    } ...
}
```

Passagem de parâmetro

- Em Java, a passagem de parâmetro é por valor
  - o valor, e não o endereço, da expressão é passado para o método chamado
  - variáveis primitivas armazenam um valor do tipo
    - `'a', 1, true, 50.89`
  - variáveis referência armazenam a referência, não o objeto!
  - modificações no parâmetro formal não são refletidas no parâmetro real

Passagem de parâmetro por valor

```
class PassagemPorValor {
    void incrementa(int x)  {
        x = x + 1;
        System.out.println ("x = " + x);
    }
}
```




```
PassagemPorValor p;
p = new PassagemPorValor();
int y = 1;
System.out.println("y = " + y);
p.incrementa(y);
System.out.println("y = " + y);
```

Referências são valores!

```
class Referencia {
    void redefine (Conta a)  {
      Conta b = new Conta("567-8",55);
      a.creditar(100);
      a = b;
      a.creditar(100);
    }
}
```

não altera o valor de c

altera o estado do objeto referenciado por c

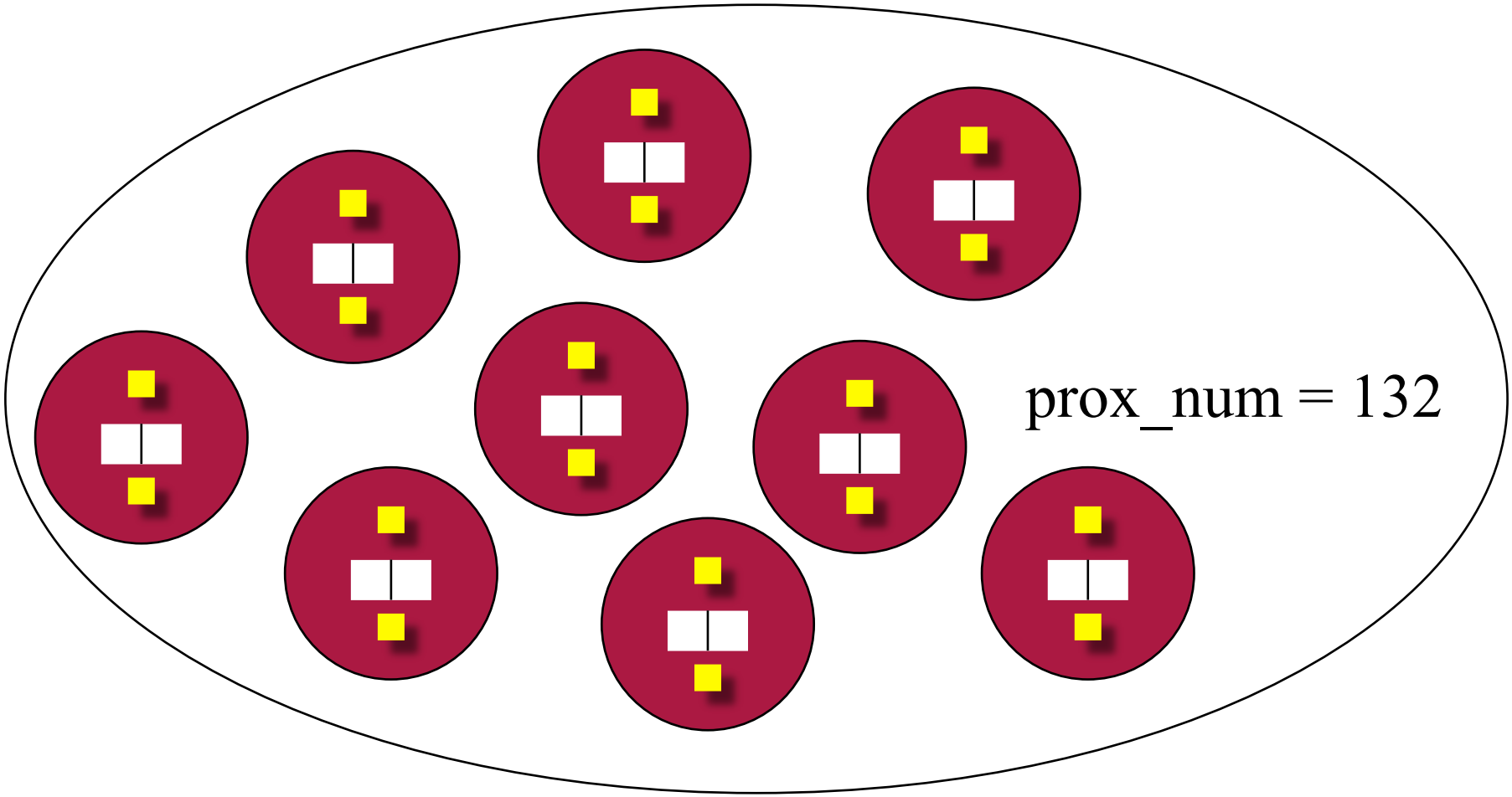
```
Referencia r;
r = new Referencia();
Conta c = new Conta("123-4",12);
r.redefine(c);
System.out.println(c.getSaldo());
```

Variáveis estáticas versus atributos

Enquanto cada instância da classe (objeto) tem seus próprios atributos, variáveis estáticas (ou de classe) são compartilhadas por todas as instâncias da classe

Cuidado!
Podem ser vistas como variáveis globais, fugindo do paradigma orientado a objetos...

Classe Conta com um gerador de números de conta



Em Java ...

```
public class ContaComGerador {
   private int numero;
   private double saldo;
   private static int prox_num = 1;

   public ContaComGerador(){
      numero = prox_num;
      saldo = 0;
      prox_num = prox_num + 1;
   }
   ...
}
```

Na prática, não deve ser feito assim!

Métodos estáticos

- Da mesma forma que há variáveis estáticas (de classe) e variáveis de instância (atributos), há métodos estáticos (de classe) e métodos de instância
  - um método estático só tem acesso as variáveis de classe (estáticas)
  - um método estático pode ser acrescentado à classe `ContaComGerador` para retornar o valor corrente de `prox_num`

O método main

```
public class Sistema {
  public static void main(String[] args) {
    Conta a = new Conta("123-4",34);
    Conta b;
    b = a;
    b.creditar(100);
    System.out.println(a.getSaldo());
  }
}
```

Só as classes com um método **`main`** podem ser executadas por um interpretador Java

Arrays

Arrays

- São objetos especiais de Java
- Uma variável do tipo array é definida usando a notação:

```
Tipo[] arrayTipo;
```

- `Tipo[]` é uma classe, mas não pode-se herdar dela

Criação de arrays

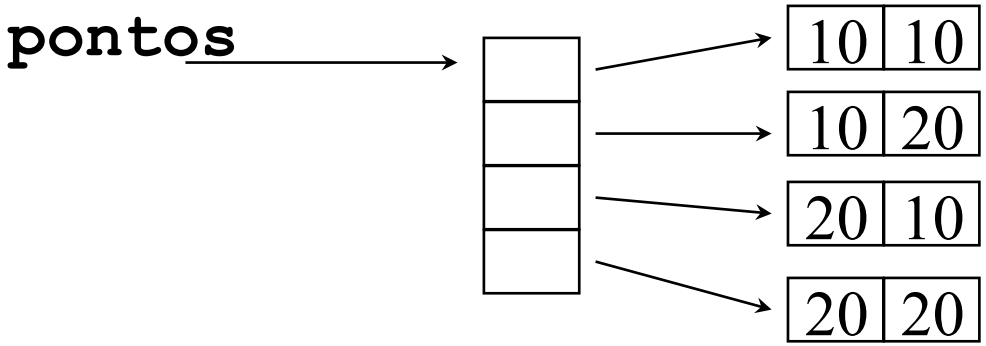
- O Operador **`new X[tamanho]`** cria um objeto array, não os objetos do tipo **`X`** por ele referenciado
- O primeiro elemento do array tem índice **0** e o último tem índice **`tamanho - 1`**
- O comprimento do array é acessível pela variável de instância (atributo) final e pública **`length`**

Inicializadores

Inicializadores de arrays são representados da seguinte forma: `{<expressões>}`. Onde ***expressões*** representam expressões de tipos válidos separadas por vírgulas

- Exemplo: Declara, cria e inicializa um array de pontos

```
int[][] pontos = {{10,10},{10,20},
                  {20,10},{20,20}};
```



Acesso

***variável*[*expressão_inteira*]**

- Acesso a array é checado em tempo de execução. A exceção **`java.lang.IndexOutOfBoundsException`** é levantada na tentativa de acesso fora dos limites do array (0..TAMANHO-1)
- Exemplo:

Representa o envio da mensagem `getNumero()` para um objeto do tipo `Conta`, referenciado pelo i-ésimo elemento do array `contas`.

```
if (contas[i].getNumero().equals(numero))
    achou = true;
else
    ...
```

Classe CadastroContas: Assinatura

```
public class CadastroContas {
   CadastroContas(){}
   void cadastrar(Conta conta){}
   void remover(String numero){}
   double getSaldo(String numero){}
   void debitar(String numero, double valor){}
   void creditar(String numero, double valor){}
   void transferir(String numeroOrigem,
                   String numeroDestino,
                   double valor){}
}
```

Todos os métodos são `public`

Classe CadastroContas: Descrição

```
public class CadastroContas {
    private Conta[] contas;
    private int indice;

    public CadastroContas(int tamanho) {
        contas = new Conta[tamanho];
        indice = 0;
    }

    public void cadastrar(Conta conta) {
       contas[indice] = conta;
       indice = indice + 1;
    }
```

```
public void debitar(String numero,
                    double valor) {
  Conta c;
  c = this.procurar(numero);
  c.debitar(valor);
}

public void creditar(String numero,
                     double valor) {
  Conta c;
  c = this.procurar(numero);
  c.creditar(valor);
}
```

```
  private Conta procurar(String numero) {
    int i = 0;
    boolean achou = false;
    Conta resposta = null;
    while((! achou) && (i < indice)) {
      if (contas[i].getNumero().equals(numero))
        achou = true;
      else
        i = i + 1;
    }
    if (achou)
      resposta = contas[i];
    else
      throw new RuntimeException("Conta não existe!");
    return resposta;
  }
}
```

Exercício

- Defina a classe **Banco** com o construtor e os métodos **creditar**, **remover**, **transferir** e **getSaldo**. A classe **Banco** utiliza a classe **CadastroContas** para manipular as contas do banco.

```
public class Banco {
    private CadastroContas contas;
    ...
}
```

Especialização em Engenharia de Software

Programação Orientada a Objetos

Conceitos gerais e array

Sérgio Soares
scbs@cin.ufpe.br