Introdução a Programação – IF669
http://www.cin.ufpe.br/~if669

Conceitos Básicos de Orientação a Objetos e Strings

AULA 08

Ricardo Massa F. Lima
rmfl@cin.ufpe.br

Sérgio C. B. Soares
scbs@cin.ufpe.br

Até aqui . . .

- Conceitos gerais de programação
- Tipos primitivos, entrada e saída de dados, operadores
- Comando condicional, seleção e repetição
- **HOJE:** Introdução a orientação a objetos

Objetivos

- Compreender
  - Benefícios da programação orientada a objetos
  - Conceitos básicos e terminologia da programação orientada a objetos
    - objeto, classe, atributo, método

Benefícios da tecnologia de objetos

- Acelerar o tempo de desenvolvimento
- Reduzir o tempo de manutenção
- Mais fácil de entender e adaptar
- Código de melhor qualidade

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
- Objetos e atividades do mundo real

O que é um objeto?

- É o agrupamento dos dados e operações que representam um conceito
  - Conta bancária
    - número e saldo
    - creditar e debitar
  - Aluno da UFPE (cadastrado no Sig@)
    - nome, cpf, endereço ...
    - corrigir nome, atualizar endereço
  - Produto (de supermecado)
    - código, descrição, valor ...
    - atualizar estoque, remarcar preço...

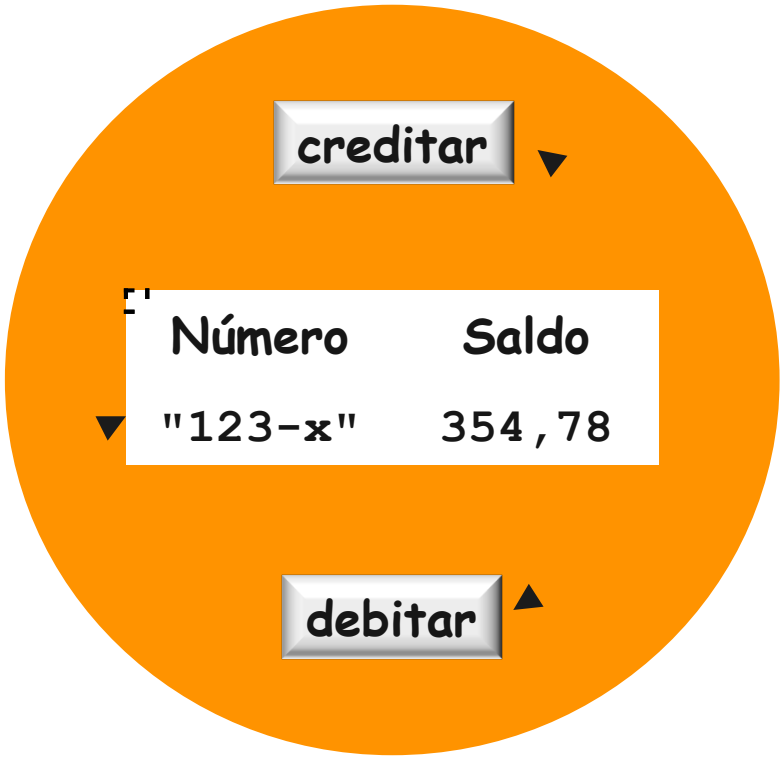
Objetos

- Blocos básicos para construção de um programa
- Contém dados que podem ser usados e modificados
- Possuem
  - Identidade (identificação única)
  - Estado (os valores armazenados)
  - Interface (como se comunicar com ele)
  - Comportamento (operações que pode executar)

Objetos

- Um carro pode ser considerado um objeto
  - Identidade ("1")
  - Estado (sua cor, tipo de pneu, etc)
  - Interface (volante, pedal do freio, etc)
  - Comportamento (respostas ao giro do volante, ao pisar o pedal do freio)
- Muitos textos definem um objeto como possuindo duas características apenas: estado e comportamento
  - Nesses casos, a identidade é parte do estado e a interface é parte do comportamento

Objeto Conta Bancária

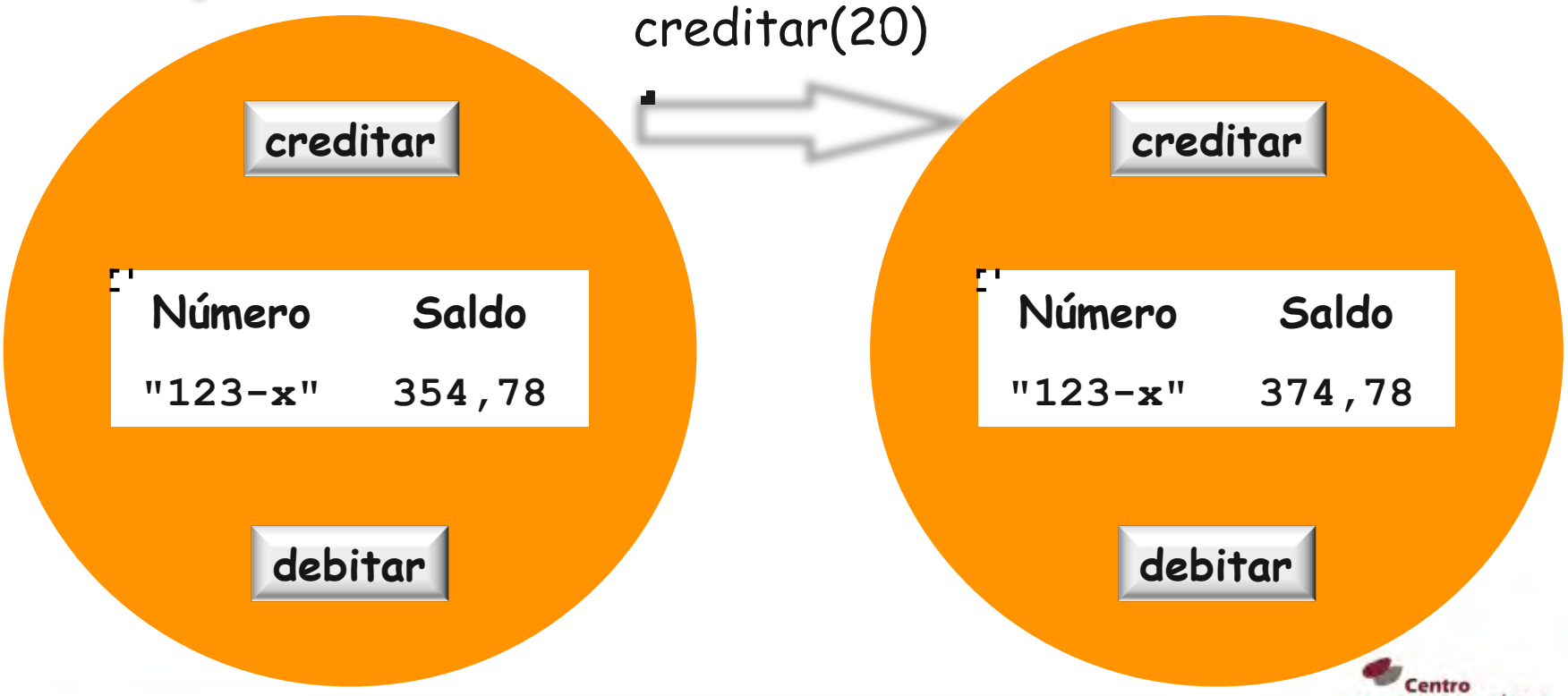


O estado atual da conta

Comportamento: operações que uma conta pode executar

Estados do Objeto Conta

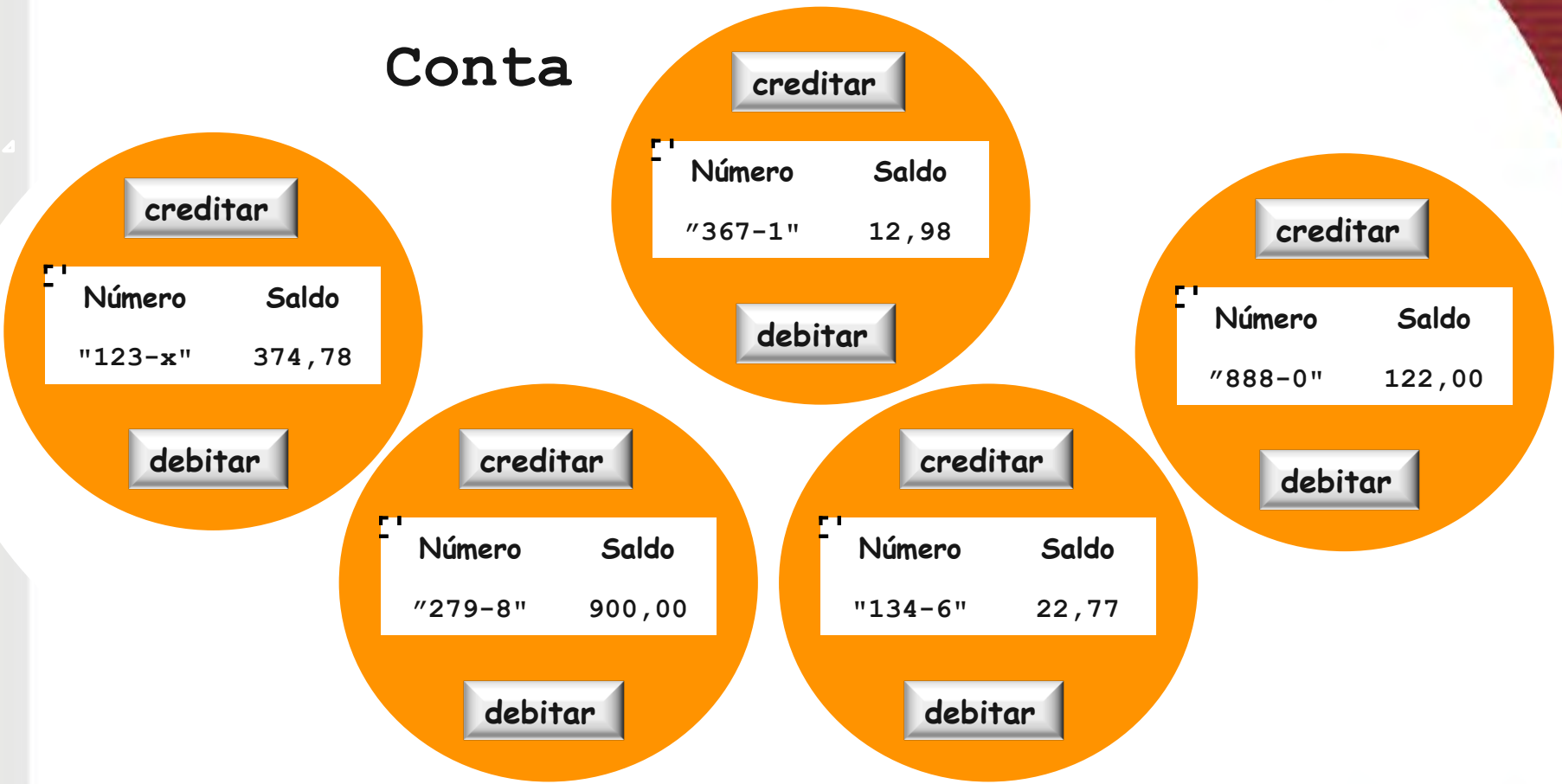
- Comportamento mudou o estado do objeto conta



Classe

- Agrupamento de objetos que têm propriedades comuns e realizam as mesmas operações
- Descreve como objetos pertencentes à classe são estruturados internamente (propriedades e operações)
- Classe é um conceito, o objeto é uma instância deste conceito
- Portanto, podemos ter vários objetos pertencentes à mesma classe
  - Os objetos conta de um banco (um para cada conta)

Classe x Objeto



Múltiplos objetos podem ser criados à partir da mesma classe

Centro de Informática

Mas como declarar uma classe em Java?

- Exemplo:
  - Temos uma aplicação bancária que deverá armazenar os dados de todas as contas correntes de um banco
  - Contas têm saldo e número e podemos realizar créditos e débitos nas mesmas

Definindo Classes em Java

```
public class Conta {
    CorpoDaClasse
}
```

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

Definindo Atributos em Java

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

- cada atributo tem um tipo específico que caracteriza as propriedades dos objetos da classe
- `double` e `String` denotam os tipos cujos elementos são reais e strings (texto)

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"` `"CIn\nUFPE"`
  - Operadores: + (concatenação)

```
"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

```
String a ...
String b ...
if ( a.equals(b) ) {
   ...
```

- Tamanho de um string a

```
int i =  a.length();
```

Usando String

```
String s1 = "ricardo";
String s2 = "Ricardo";
if (s1.equals(s2)) {
   System.out.println("igual");
} else {
   System.out.println("diferente");
}
if (s1.equalsIgnoreCase(s2)) {
   System.out.println("igual");
} else {
   System.out.println("diferente");
}
```

Mais operadores sobre `Strings`

- Extrair uma substring de uma string maior

```
String saudacao = "Bem-vindo";
String s = saudacao.substring(0,3);
// s = "Bem"
// caracteres das posições 0, 1 e 2
```

A classe `String` em Java contém mais de 50 métodos
http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html

Information Hiding

Não use preposições

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

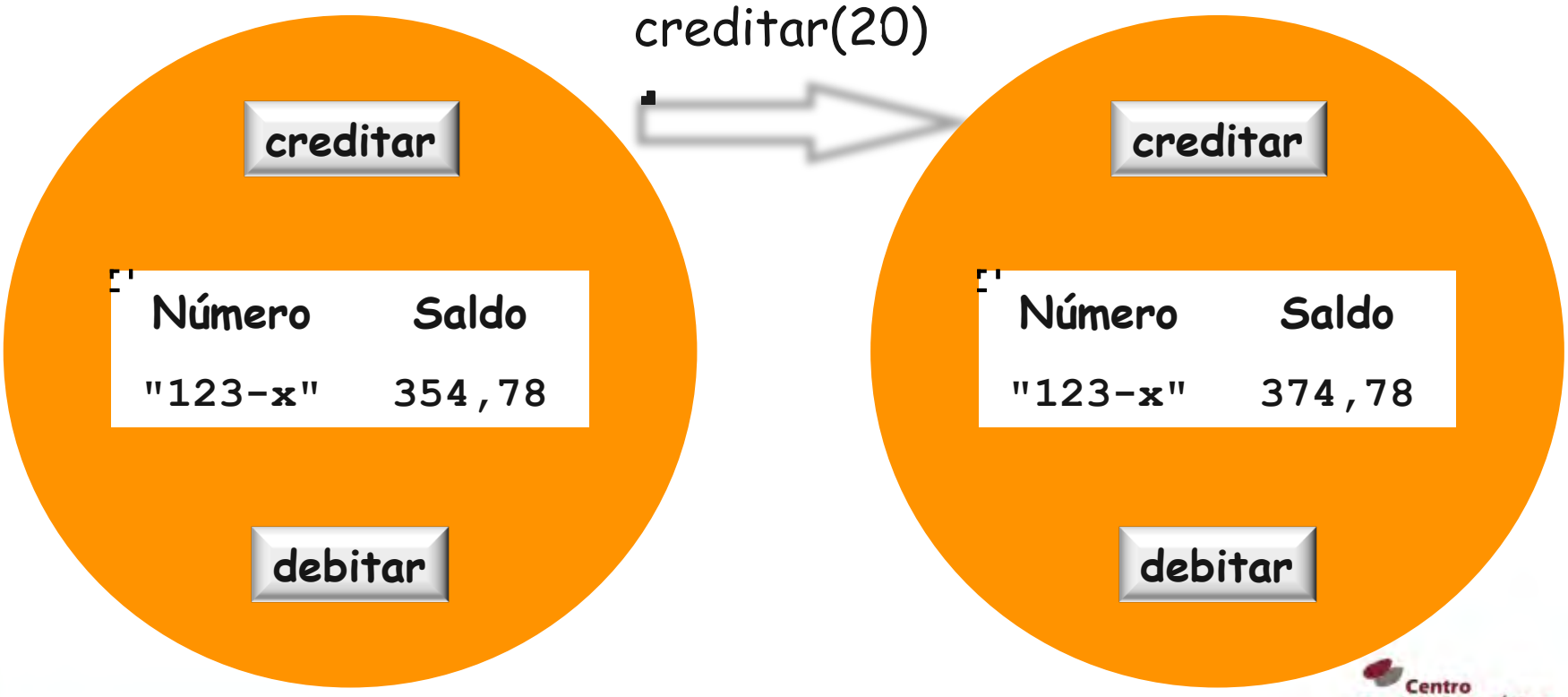
A palavra reservada **`private`** indica que os atributos só podem ser acessados (isto é, lidos ou modificados) pelas operações da classe onde foram definidos

Information Hiding e Java

- Java não obriga o uso de `private`, mas vários autores consideram isto uma pré-condição para programação orientada a objetos
- Grande impacto em extensibilidade
- Usem `private`!

Definindo métodos em Java

- Como definir o método `creditar`?



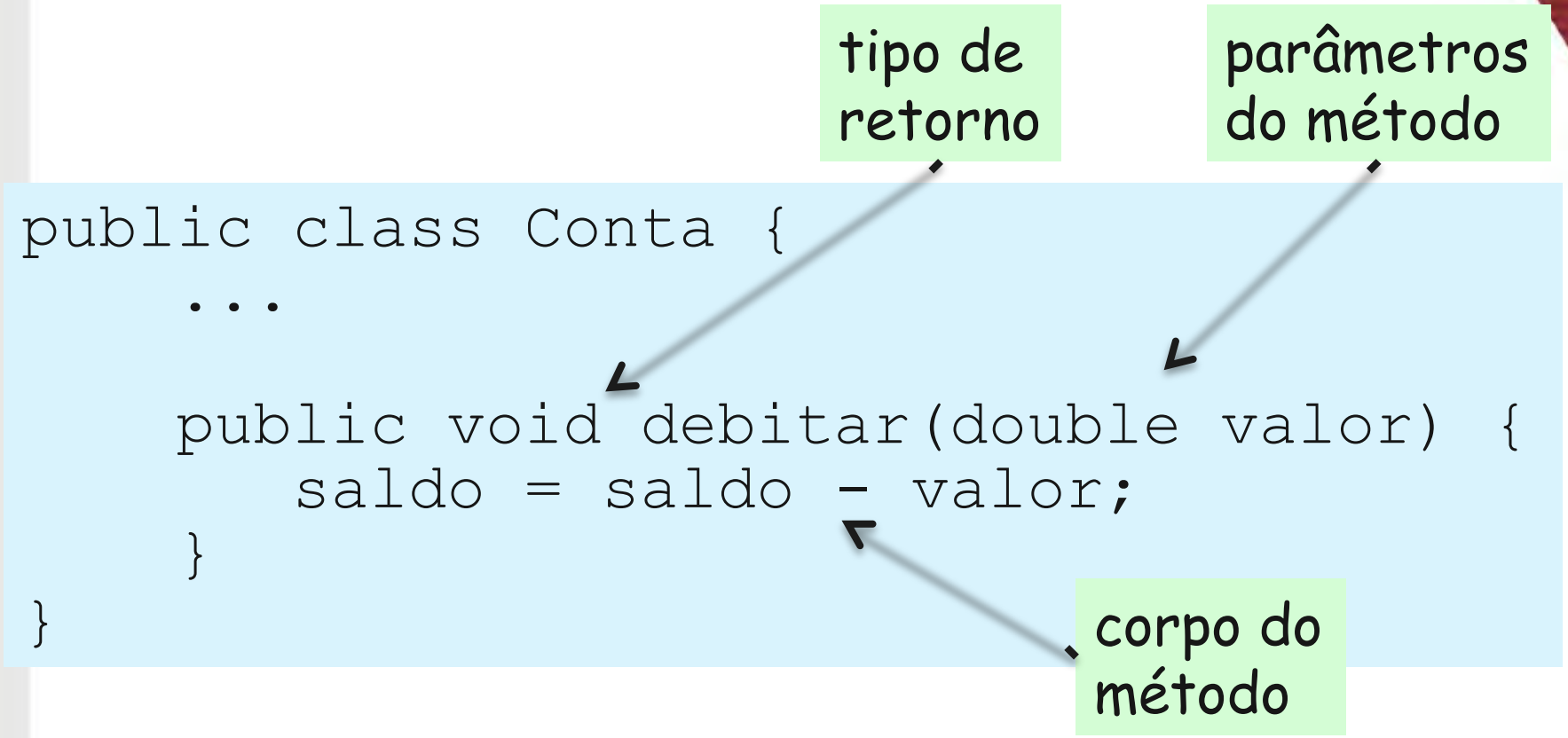
Definindo métodos em Java

```
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 que o método `debitar` não tem como parâmetro o número da conta?

Centro de Informática

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 double getSaldo() {
      return saldo;
   }
   public void setSaldo(double saldo) {
      this.saldo = 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 c = ...
c.creditar(45.30);
```

variável contendo referência para objeto

nome do método a ser executado

Exercício 1 (10 minutos)

- Faça uma classe chamada `Exercicio1` para ler o nome completo de uma pessoa e imprimir o primeiro e último nomes
  - o nome completo deve ter pelo menos dois nomes

```
Digite o nome completo: Sérgio Castelo Branco Soares
Primeiro: Sérgio
Último: Soares
```

Caractere espaço em unicode = 32

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html

Utilizando OO para desenvolver programas

- Desenvolvimento de uma solução é mais fácil quando quebramos esta solução em módulos gerenciáveis
- Desenvolver módulos separados, onde cada um é responsável por uma certa parte da solução
  - **Programação Modular**
- OO facilita a programação modular
  - Módulos são as classes e objetos

Algumas Considerações sobre OO

- Orientação a Objetos ➡ Modularidade
  - Reusabilidade
  - Extensibilidade

- Linguagens OO têm em objetos, classes, ..., elementos fundamentais para construir programas
  - Estruturas da linguagem permitem mapeamento direto dos conceitos de OO

Os conceitos de Orientação a Objetos são independentes da linguagem de programação

Exercício 2 (10 minutos)

- Crie a classe `Conta` conforme apresentado em sala
  - atributos (`numero` e `saldo`)
  - métodos (`creditar`, `debitar`, `getSaldo`, `getNumero`)

Exercício (extra aula)

- Crie uma classe `Produto` para representar produtos de supermecado
- Cada produto tem uma descrição (texto), valor (real) e quantidade em estoque (inteiro)
- Lembre-se de definir métodos para ler e alterar os atributos dos produtos

`String` é uma classe e as variáveis do tipo `String` armazenam referências para objetos `String`

Mas como criamos um objeto?

Antes de criar objetos...

- Precisamos criar um método especial nas classes que será responsável por inicializar os atributos dos objetos que criaremos
- Estes métodos especiais são chamados de construtores

Construtores

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

construtores

definindo como os atributos de um objeto são 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 numeroInicial,
                double saldoInicial) {
      numero = numeroInicial;
      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",100.0);
```

Atribui à variável `c` a referência para o objeto criado

cria um objeto do tipo `Conta` em memória

responsável por inicializar os atributos do objeto criado

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

Exemplo de classe

```
public class Conta {
   private String numero;
   private double saldo;
   public Conta(String numeroInicial) {
      numero = numeroInicial;
      saldo = 0.0;
   }
   public void creditar(double valor) {
      saldo = saldo + valor;
   } ...
   public String getNumero() {
      return numero;
   } ...
}
```

- Essa classe não tem `main`!!!
- Método `main` inicia a execução de toda aplicação Java
- Portanto, essa classe não é uma aplicação

Classe que usa/testa Conta

```
public class Programa {
  public static void main(String[] args) {
    Conta c = new Conta("123-X”, 8.0);
    c.creditar(10.0);
    c.debitar(5.0);
    System.out.print("Conta "+c.getNumero());
    System.out.print(" saldo "+c.getSaldo());
  }
}
```

Por exemplo

- Crie uma classe `Curso` com código e nome
- Crie uma classe `Aluno`, contendo nome, cpf, idade e o `Curso` que o aluno frequenta
- Lembre-se de criar um construtor, bem como os métodos `get` e `set` para as classes
- Crie uma classe `TestaAluno` para testar as classes criada

Classe Curso

Sem o uso do `this` no construtor e no `setNome`, `nome` e `codigo` seriam os parâmetros, não os atributos

```
public class Curso {
   private String nome;
   private String codigo;
   public Curso(String nome, String codigo) {
      this.nome = nome;
      this.codigo = codigo;
   }
   public String getNome() {
      return this.nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
   // ... get/set para código
}
```

A palavra reservada `this` representa uma referência para o objeto em execução

Classe Aluno

Note que a classe `Aluno` tem um relacionamento com a classe `Curso`

```
public class Aluno {
  private String nome;
  private String cpf;
  private int idade;
  private Curso curso;
  public Aluno(String nome, String cpf,
               int idade, Curso curso) {
    this.nome = nome;
    this.cpf = cpf;
    this.idade = idade;
    this.curso = curso;
  }
  public Curso getCurso() {
    return this.curso;
  }
  // ... os outros get/set
}
```

Teste de Aluno

```
public class TestaAluno {
  public static void main(String[] args) {
    Curso curso = new Curso("Ciência da Computação", "18");
    Aluno a = new Aluno("Sergio Soares",
                        "239.234.111-98", 32, curso);
    System.out.println("Dados do aluno: "+a.getNome());
    System.out.println("CPF: "+a.getCpf());
    System.out.println("Idade: "+a.getIdade());
    System.out.println("Curso: "+a.getCurso().getNome());
    a.setCpf("NOVO_CPF");
    a.setIdade(20);
    a.setNome("NOVO_NOME");
    curso = new Curso("Engenharia da Computação", "21");
    a.setCurso(curso);
    System.out.println("Dados do aluno: "+a.getNome());
    System.out.println("CPF: "+a.getCpf());
    System.out.println("Idade: "+a.getIdade());
    System.out.println("Curso: "+a.getCurso().getNome());
  }
}
```

Métodos (boas práticas)

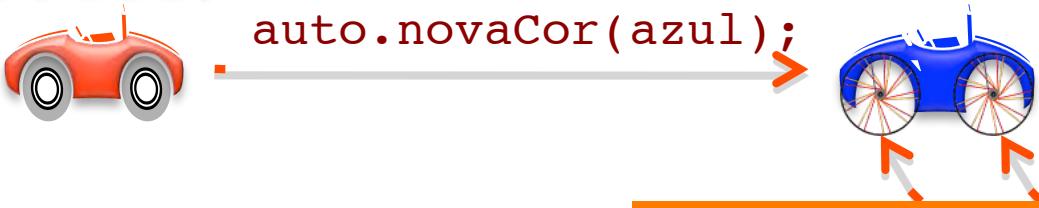
- Crie métodos para
  - Encapsular complexidade e fazer seu código mais legível

```
1. inches = centimeters / 2.54;
2. inches = Metric.centimetersToInches(centimeters);
```

  - Evitar código duplicado. Exemplo: O que é mais interessante?
    - Promover reutilização de código
    - Isolar operações e estruturas de dados complexas

Métodos (boas práticas)

- Métodos devem ter **forte coesão**: tudo dentro de um método deve ser relacionado ao seu propósito central. Se há dois propósitos, deve haver dois métodos.



Não deveria ter um método específico para mudar as rodas

Métodos (boas práticas)

- Acoplamento é um termo usado para descrever quão dependente um método é de outro(s)
- O ideal é ter **fraco acoplamento**
- Existe uma longa disputa sobre qual é o tamanho ideal para métodos:
  - Muitos acham que devemos nos restringir a uma única página
  - Muitos acham que vários pequenos métodos podem diminuir a legibilidade do código.
  - Às vezes, métodos longos são bons, desde que possuam uma unicidade lógica.

Múltiplos Construtores

- Objetos da classe `Conta` podem ser inicializados de duas formas:

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

  public Conta(String numero, double saldo) {
      this.numero = numero;
      this.saldo = saldo;
   }
   public Conta(String numero) {
      this(numero, 0.0);
   }...
}
```

Chama o outro construtor da classe

Method Overloading

- Métodos com mesmo nome e diferentes listas de argumentos
- Da mesma forma que os dois construtores de `Conta`

Exercício 3 (10 minutos)

- Defina os dois construtores da classe `Conta` conforme mostrado na aula de hoje
  - Utilize a classe `Conta` resultante do exercício da aula passada
    - disponível na solução!

Variáveis e métodos estáticos

- Até aqui variáveis e métodos de instância
  - Para acessar o atributo (variável de instância) ou chamar o método é preciso ter um objeto
  - Cada objeto tem seu atributo
- Variáveis e métodos estáticos são da classe
  - Todos os objetos compartilham uma mesma variável estática
  - Acesso através do nome da classe que os contém

Variáveis e métodos estáticos

```
public class ContaComGerador {
   private int numero;
   private double saldo;
   private static int proximo = 0;
   public ContaComGerador() {
      this.numero = ContaComGerador.getProximo();
      this.saldo = 0.0;
}
private static int getProximo() {
      proximo = proximo + 1;
      return proximo;
}
// ...
}
```

Variáveis e métodos estáticos

- O método `main` é estático
  - Por onde se inicia a execução
  - A execução não inicia de um objeto, mas da classe que contém o `main`
- Métodos estáticos só acessam váriáveis e outros métodos estáticos
  - Atributos e métodos de instância apenas através de objetos, como qualquer outro

Variáveis e métodos estáticos

```
public class ContaComGerador {
   private int numero;
   private double saldo;
   private static int proximo = 0;
   public ContaComGerador() {
      this.numero = ContaComGerador.getProximo();
      this.saldo = 0.0;
}
private static int getProximo() {
      proximo = proximo + 1;
      this.saldo = 0.0;
      return proximo;
}
// ...
}
```

Erro de compilação `saldo` é atributo, não pode ser acessado por método estático

Exercício (extra aula)

- Defina pelo menos um construtor para a classe `Produto` resultante do exercício extra anterior