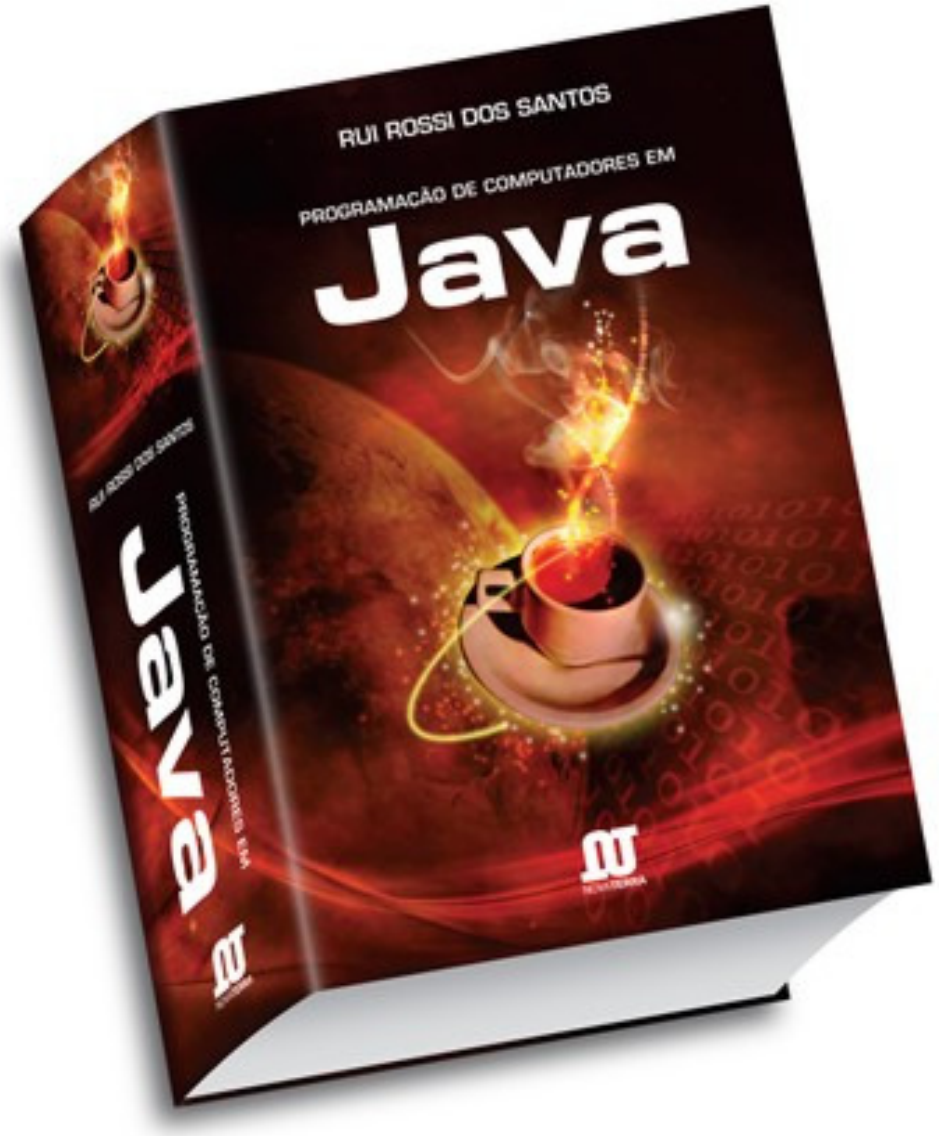


Capítulo 14

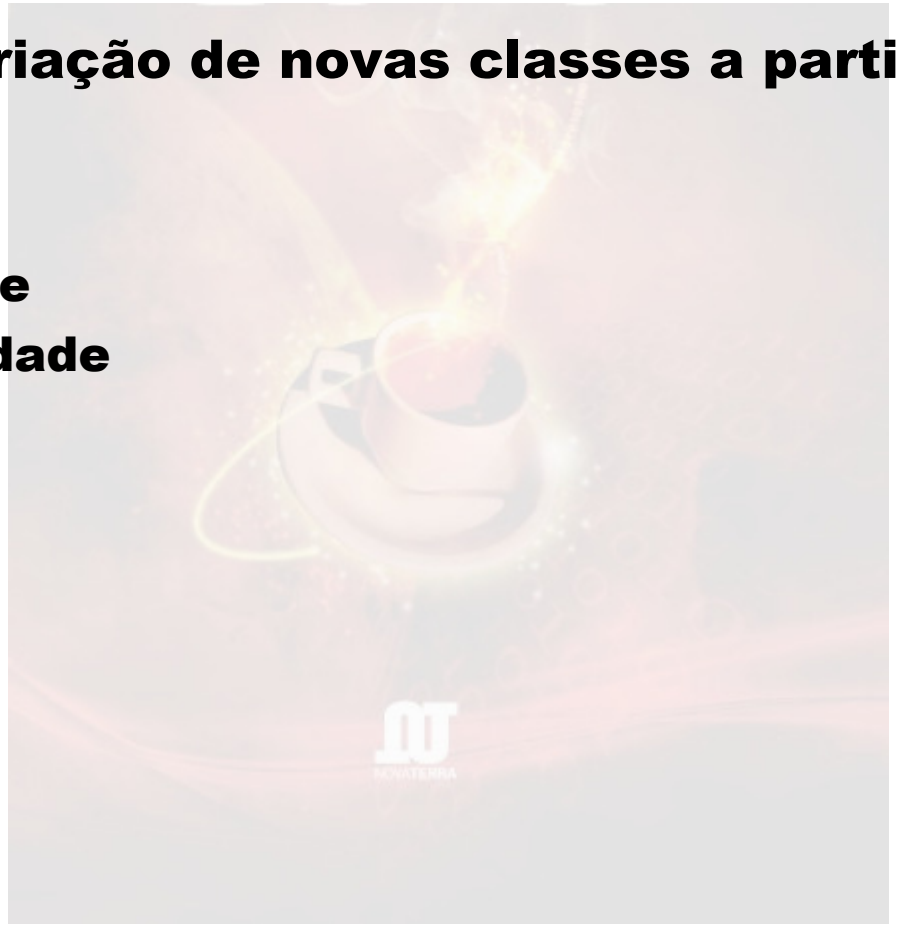
Herança e Polimorfismo

Objetivos do Capítulo

- Apresentar os conceitos de herança e de polimorfismo.
- Explorar os diversos recursos disponíveis no Java para a aplicação da herança e do polimorfismo na construção de estruturas complexas de classes.
- Analisar o uso conjugado da herança e do polimorfismo.
- Evidenciar os benefícios que podem ser obtidos através da aplicação da herança e do polimorfismo.

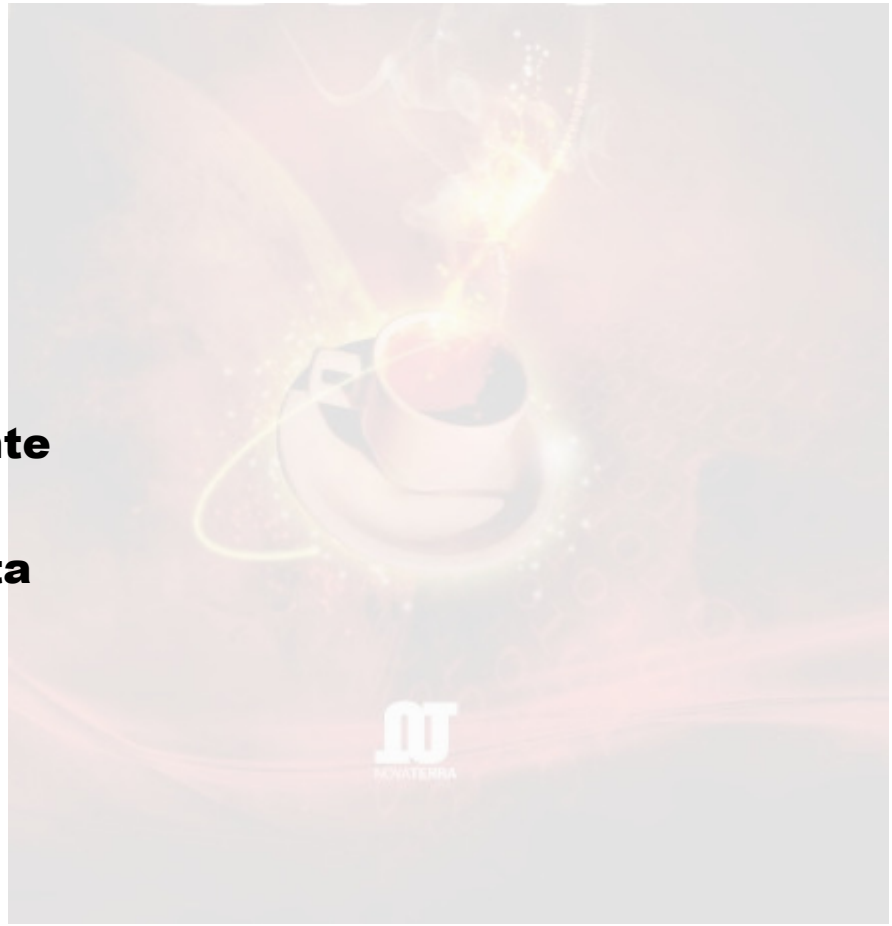
O Conceito de Herança

- Mecanismo fundamental da POO
- Possibilita a criação de novas classes a partir das existentes
- Finalidade:
  - Reusabilidade
  - Manutenibilidade



O Conceito de Herança

- **Papéis:**
  - **Superclasse**
    - **Ancestral**
    - **Mãe**
    - **Genérica**
  - **Subclasse**
    - **Descendente**
    - **Filha**
    - **Especialista**



A declaração extends

- **Onde**
  - **Na declaração de uma classe**
- **Para que**
  - **Especificar a superclasse**
- **Se omitida**
  - **Classe java.lang.Object é assumida**

A declaração extends

- **Sintaxe:**

```
[public] class <subclasse> extends <superclasse> {
}
```

- **Exemplo:**

```
public class Aluno extends Pessoa {
}
```

A referência this

- **Onde aplicar:**
  - **Construtores e métodos**
- **Para que:**
  - **Referência ao objeto atual**
  - **Acessar atributos**
  - **Executar construtores**
- **Quando:**
  - **Parâmetros e atributos com nomes idênticos**
  - **Construtor que executa outro construtor**

A referência this

- **Acessar atributos: sintaxe**

```
this.<atributo>
```

- **Exemplo:**

```
public class Pessoa {
  private String nome;

  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

A referência this

- **Executar outro construtor: sintaxe**

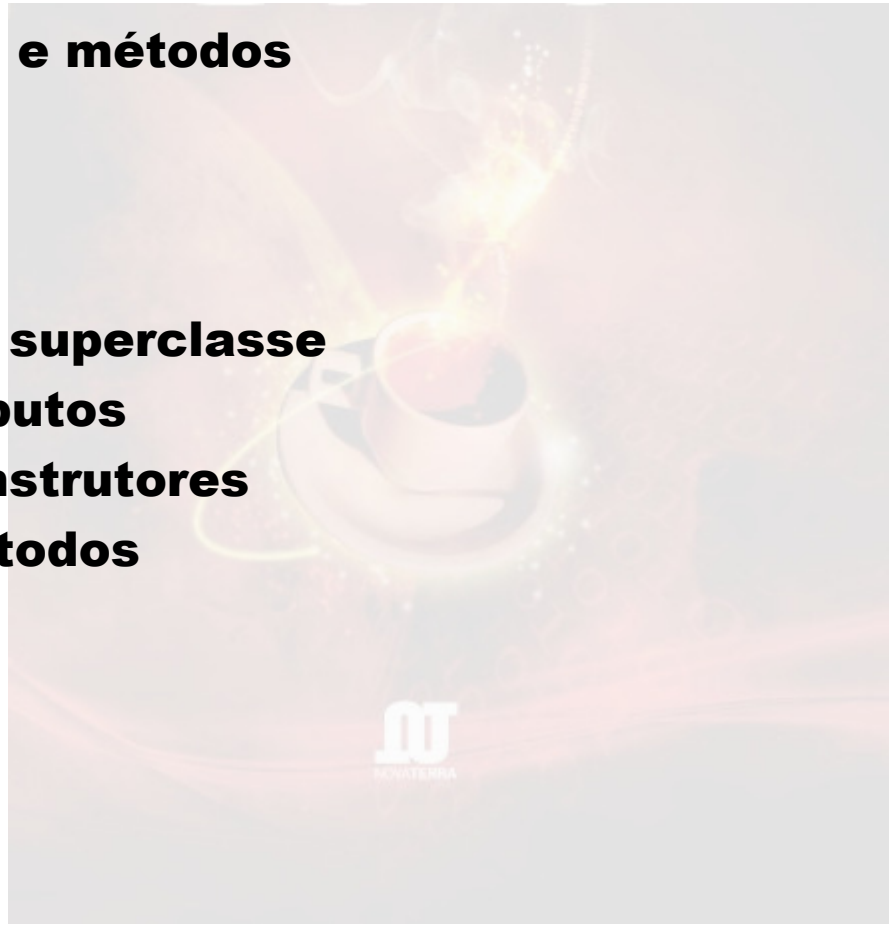
```
this([parâmetros])
```

- **Exemplo:**

```
public class Amigo {
  private String nome;
  private String fone;

  public Amigo( ) {
    this("","");
  }

  public Amigo(String nome, String fone) {
    this.nome = nome;
    this.fone = fone;
  }
}
```

A referência super

- **Onde aplicar:**
  - **Construtores e métodos**
- **Para que:**
  - **Referência à superclasse**
  - **Acessar atributos**
  - **Executar construtores**
  - **Executar métodos**



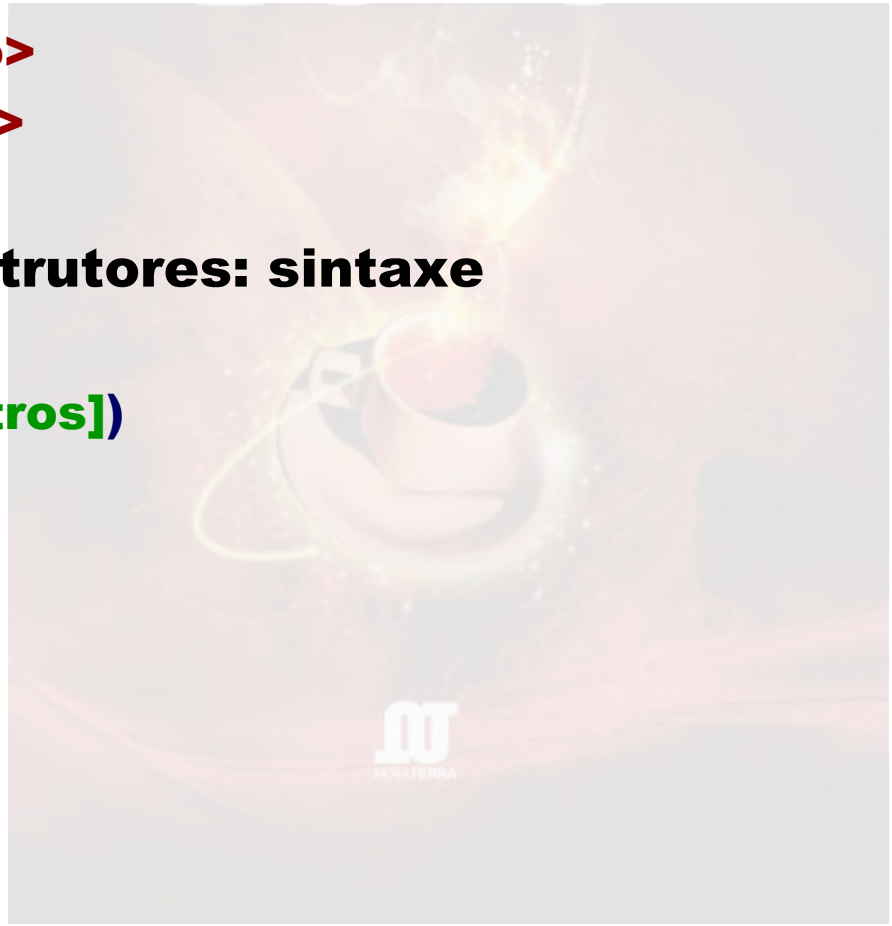
A referência super

- **Acessar atributos e métodos: sintaxe**

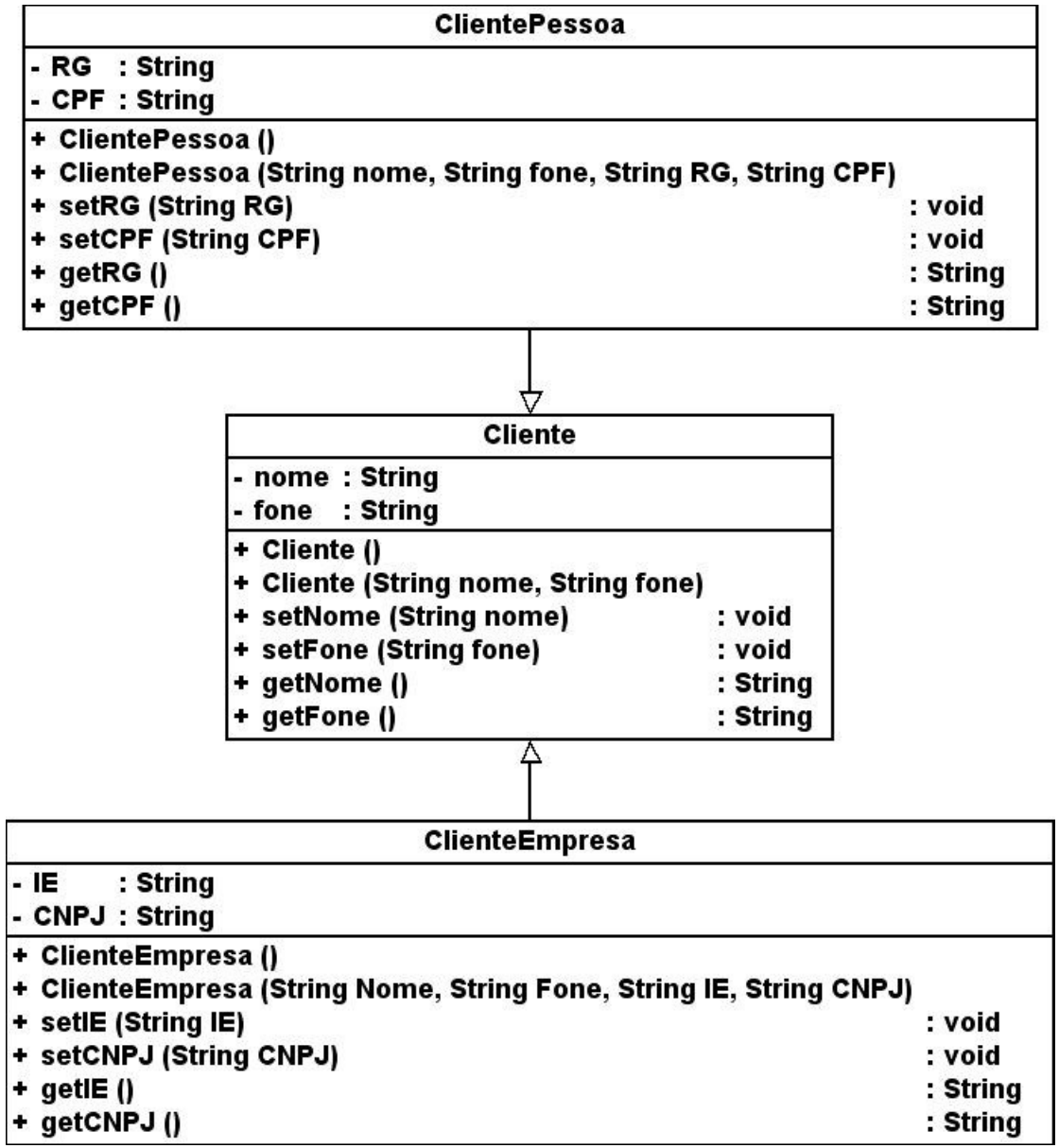
  super.<atributo>
  super.<método>

- **Executar construtores: sintaxe**

  super([parâmetros])



Exemplo de Especialização



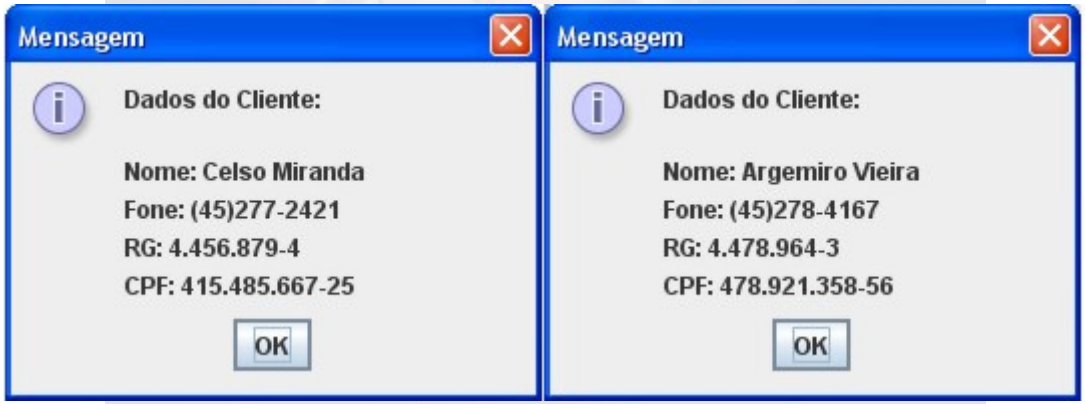
Exemplo de Especialização

- **Código 14.1 – Cliente.java**
  - **Construtor Cliente( )**
    - **Invocar o construtor Cliente(String,String)**
    - **Inicializar atributos com: “Sem nome” e “(00)0000-0000”**
  - **Analisar o uso dos termos:**
    - **this( )**
    - **this.<atributo>**

- **Código 14.2 – ClientePessoa.java**
  - **Construtor ClientePessoa( )**
    - **Invocar o construtor ClientePessoa(String,String,String,String)**
    - **Inicializar atributos com strings vazias**
  - **Construtor ClientePessoa(String,String,String,String)**
    - **Invocar o construtor da superclasse: super(nome, fone)**

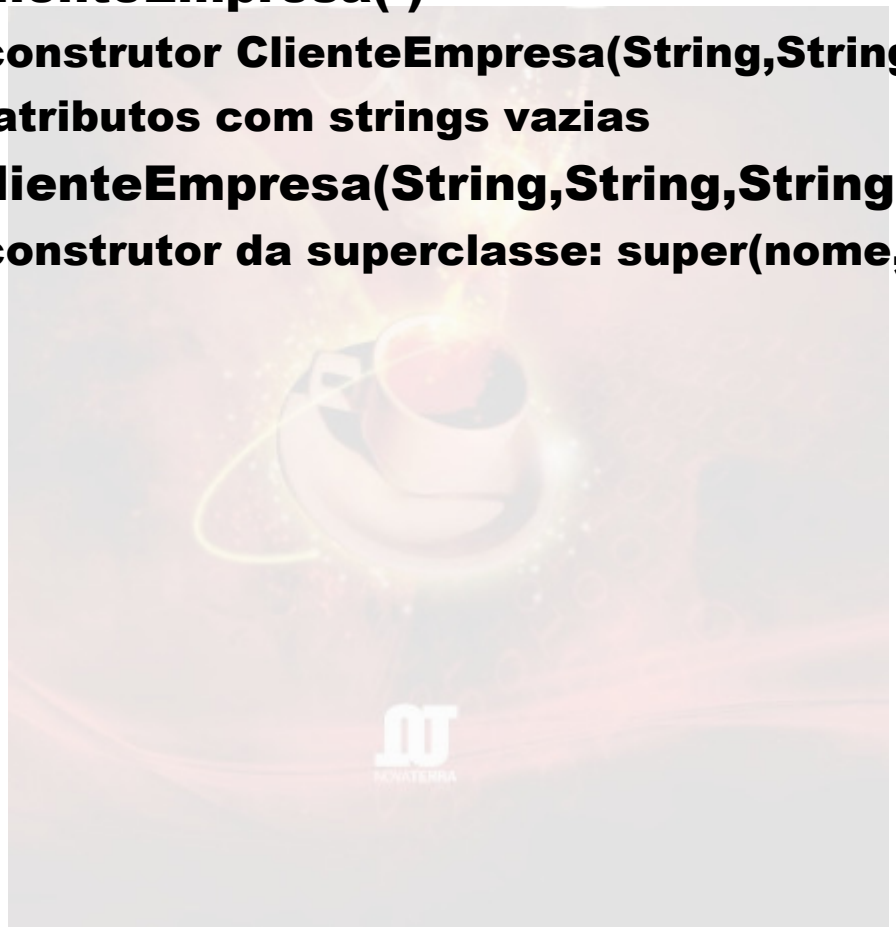
Exemplo de Especialização

- **Código 14.3 – CadastroClientePessoa.java**
  - **Crie uma instância da classe ClientePessoa.**
    - **Utilize o construtor padrão.**
    - **Inicialize seus atributos com os métodos de escrita.**
  - **Recupere e exiba o conteúdo de seus atributos.**
  - **Crie outra instância da classe ClientePessoa.**
    - **Utilize o construtor alternativo.**
  - **Recupere e exiba o conteúdo de seus atributos.**



Exemplo de Especialização

- **Código 14.4 – ClienteEmpresa.java**
  - **Construtor ClienteEmpresa( )**
    - **Invocar o construtor ClienteEmpresa(String,String,String,String)**
    - **Inicializar atributos com strings vazias**
  - **Construtor ClienteEmpresa(String,String,String,String)**
    - **Invocar o construtor da superclasse: super(nome, fone)**



Exemplo de Especialização

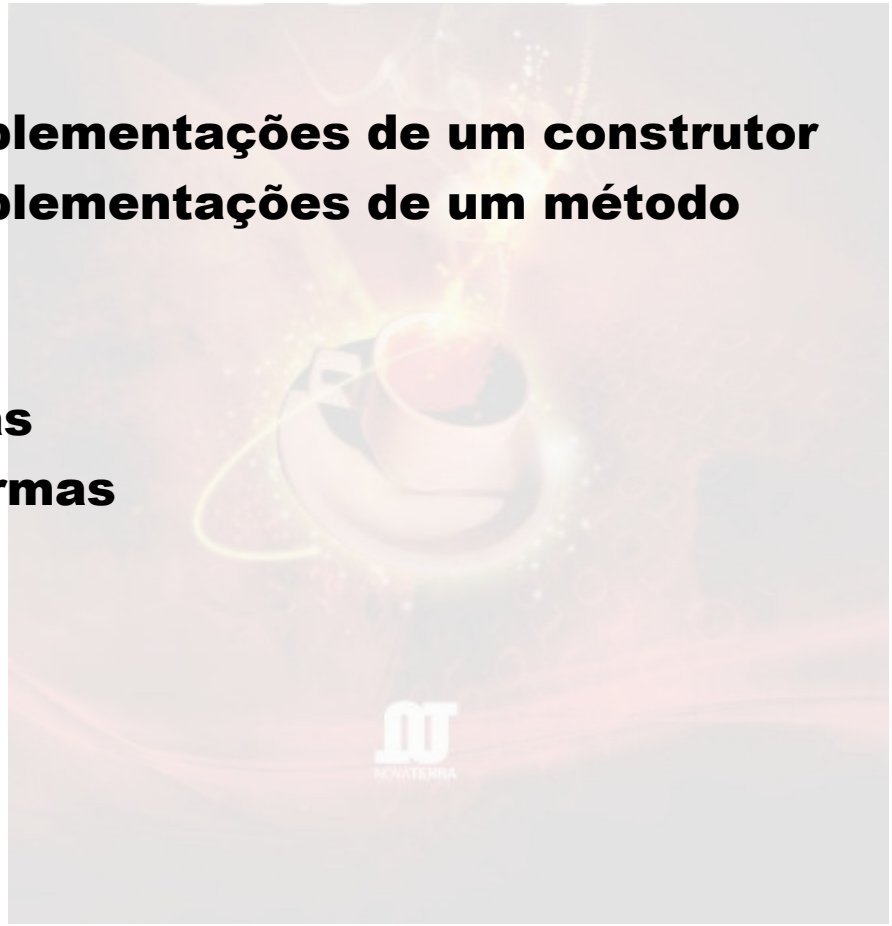
- **Código 14.5 – CadastroClienteEmpresa.java**
  - **Crie uma instância da classe ClienteEmpresa.**
    - **Utilize o construtor padrão.**
    - **Inicialize seus atributos com os métodos de escrita.**
  - **Recupere e exiba o conteúdo de seus atributos.**
  - **Crie outra instância da classe ClienteEmpresa.**
    - **Utilize o construtor alternativo.**
  - **Recupere e exiba o conteúdo de seus atributos.**

O Conceito de Polimorfismo

- Mecanismo fundamental da POO

- Possibilita:
  - Múltiplas implementações de um construtor
  - Múltiplas implementações de um método

- Termo
  - Poli: múltiplas
  - Morfismo: formas



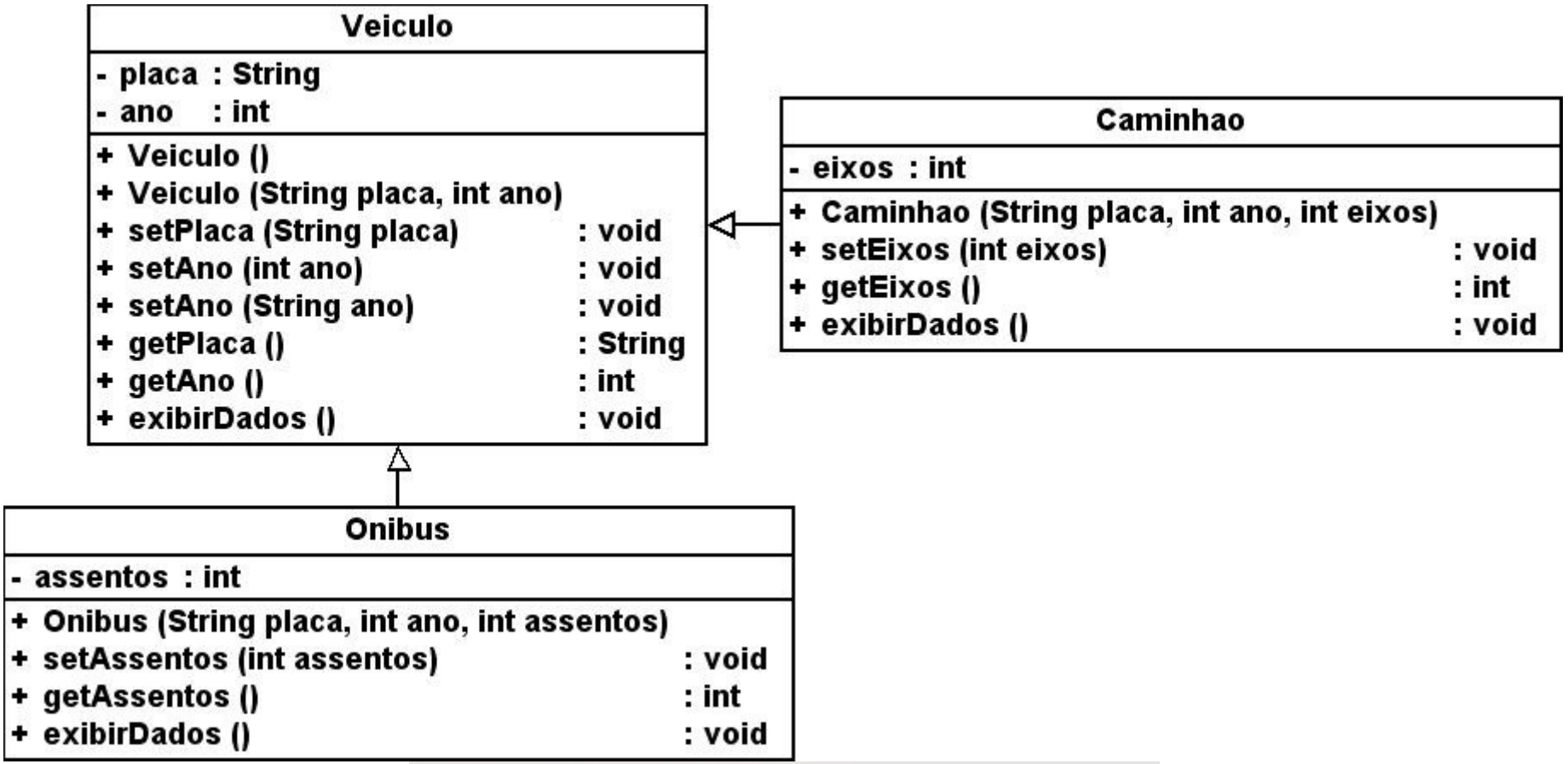
O Conceito de Polimorfismo

- **Tipos**
  - **Sobrecarga**
  - **Sobreposição**

- **Sobrecarga**
  - **Variações nos parâmetros**
  - **Na mesma classe**
  - **Em subclasses**

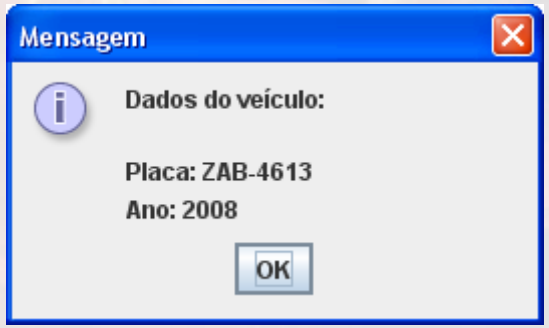
- **Sobreposição**
  - **Variações na implementação**
    - **Implementação nova**
    - **Implementação complementar**
  - **Em subclasses**

Exemplo de Polimorfismo



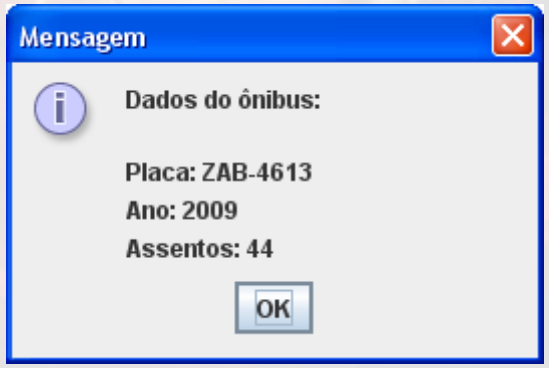
Exemplo de Polimorfismo

- **Código 14.6 – Veiculo.java**
  - **Construtor Veiculo()**
    - **Invocar o construtor Veiculo(String placa,int ano)**
    - **Inicializar atributos com: “” e 0**
  - **Método exibirDados()**
    - **Produzir uma mensagem gráfica**
    - **Exibir os dados do veículo conforme figura abaixo.**



Exemplo de Polimorfismo

- **Código 14.7 – Onibus.java**
  - **Construtor Onibus(String placa, int ano, int assentos)**
    - **Invocar o construtor da superclasse: Veiculo(String placa,int ano)**
    - **Complementar com a inicialização do atributo assentos**
  - **Método exibirDados()**
    - **Produzir uma mensagem gráfica**
    - **Exibir os dados do ônibus conforme figura abaixo.**

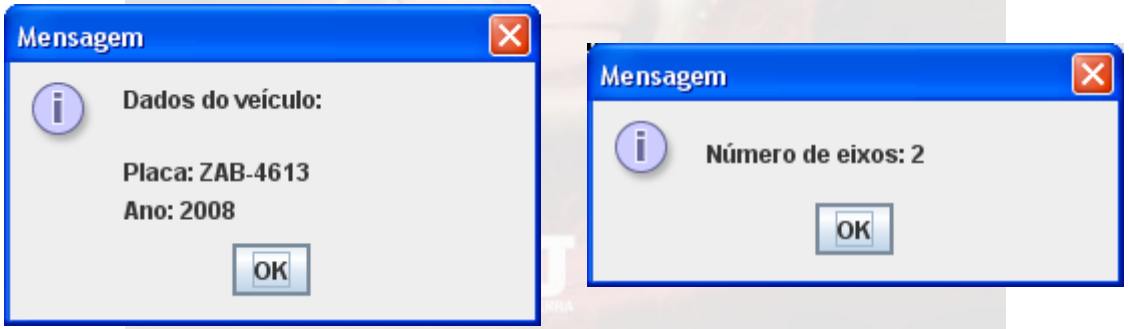


Exemplo de Polimorfismo

- **Código 14.8 – CadastroOnibus.java**
  - **Crie uma instância da classe Onibus.**
    - **Utilize o construtor alternativo.**
  - **Recupere e exiba o conteúdo de seus atributos.**
  - **Altera o conteúdo de seus atributos.**
    - **Utilize os métodos de escrita.**
  - **Recupere e exiba o conteúdo de seus atributos.**

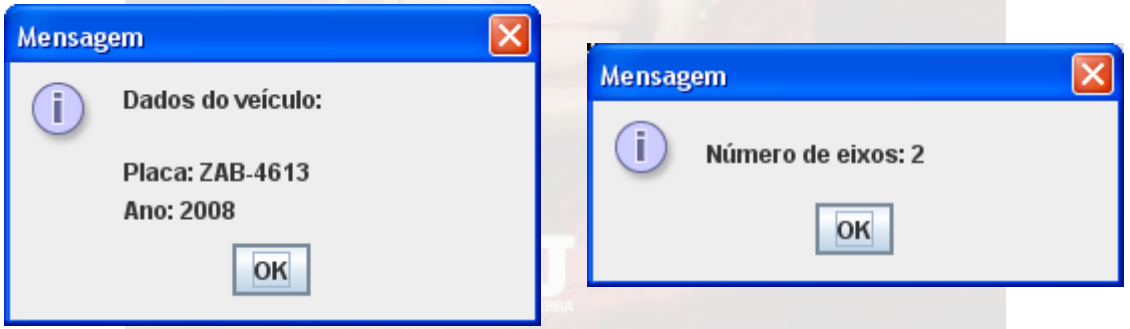
Exemplo de Polimorfismo

- **Código 14.9 – Caminhao.java**
  - **Construtor Caminhao(String placa, int ano, int eixos)**
    - **Invocar o construtor da superclasse: Veiculo(String placa,int ano)**
    - **Complementar com a inicialização do atributo eixos**
  - **Método exibirDados()**
    - **Produzir uma mensagem gráfica**
    - **Exibir os dados do caminhão em duas mensagens separadas.**



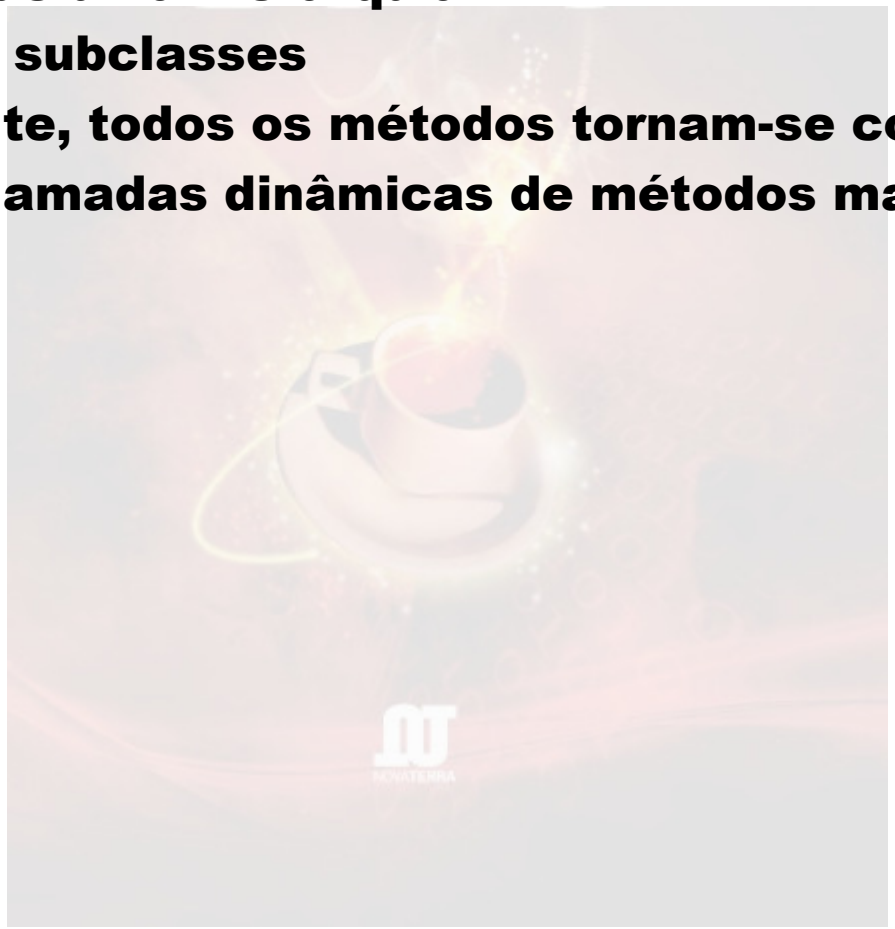
Exemplo de Polimorfismo

- **Código 14.10 – CadastroCaminhao.java**
  - **Crie uma instância da classe Caminhao.**
    - **Utilize o construtor alternativo.**
  - **Recupere e exiba o conteúdo de seus atributos.**



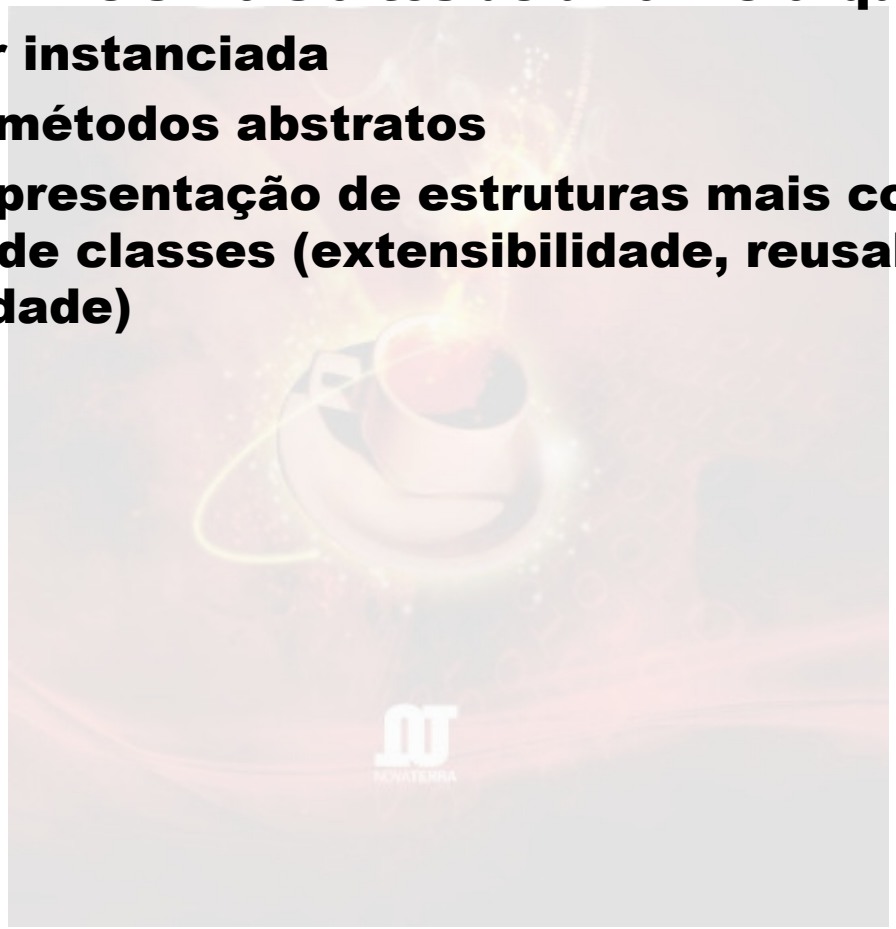
Classes Finais e Abstratas

- **Classe final (constante, terminal)**
  - **Último nível de uma hierarquia**
  - **Não pode ter subclasses**
  - **Implicitamente, todos os métodos tornam-se constantes**
  - **Benefício: chamadas dinâmicas de métodos mais eficientes**



Classes Finais e Abstratas

- **Classe abstrata**
  - **Utilizada nos níveis mais altos de uma hierarquia**
  - **Não pode ser instanciada**
  - **Pode conter métodos abstratos**
  - **Benefício: representação de estruturas mais complexas e sofisticadas de classes (extensibilidade, reusabilidade, manutenibilidade)**



Classes Finais e Abstratas

- **Classe final: sintaxe**

```
public final class <nome>
```

- **Classe abstrata: sintaxe**

```
public abstract class <nome>
```

- **Exemplos:**

```
public abstract class Veiculo
public final class Onibus extends Veiculo
public final class Caminhao extends Veiculo
```

Métodos Constantes e Abstratos

- **Métodos constantes**
  - **Incluem o qualificador final**
  - **Não podem ser sobrepostos em subclasses**

- **Métodos abstratos**
  - **Incluem o qualificador abstract**
  - **Não possui um corpo**
  - **Só é permitido em classes abstratas**
  - **Precisa ser implementado em todas as subclasses**

Manipulação Polimórfica de Objetos

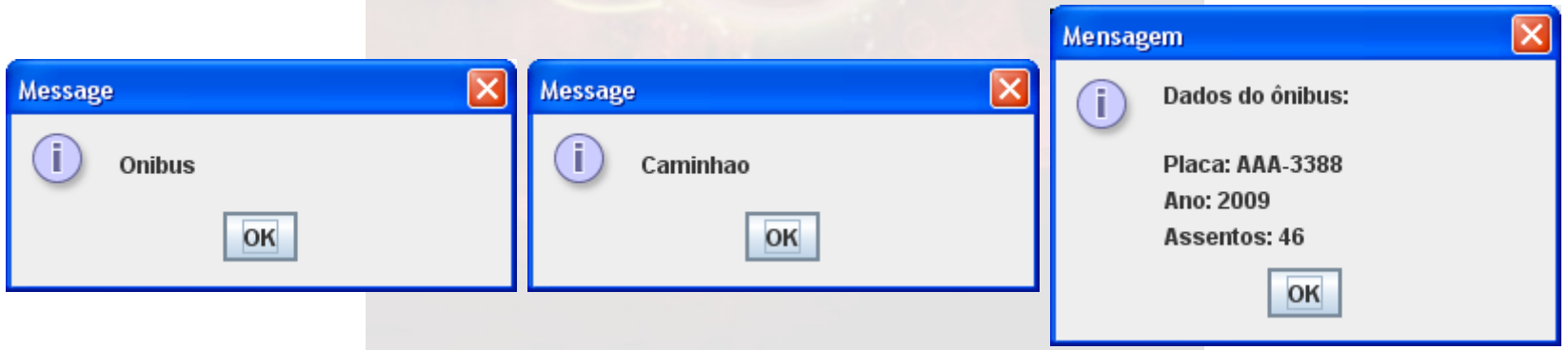
- **Conceitos importantes: objeto**
  - Ocupa espaço de memória
  - Mantém o conteúdo de seus atributos

- **Conceitos importantes: referência**
  - Ponteiro de memória
  - Aponta para um objeto
  - Possui tipo e nome
  - O tipo pode ser:
    - A classe do objeto
    - Uma das superclasses do objeto

Manipulação Polimórfica de Objetos

- **Exemplo:**

```
Veiculo v1 = new Onibus("AAA-3388",2009,46);
Veiculo v2 = new Caminhao("BBB-3498",2009,2);
Veiculo v3 = v1;

JOptionPane.showMessageDialog(null, v1.getClass().getName());
JOptionPane.showMessageDialog(null, v2.getClass().getName());
v3.exibirDados();
```



Manipulação Polimórfica de Objetos

- **Inspeção de tipo: instanceof**
  - Permite testar o tipo de um objeto em tempo de execução
  - Precedido pelo nome do objeto
  - Seguido do nome da classe testada
  - Produz um valor booleano

- **Exemplo:**

```
Veiculo v1 = new Onibus("AAA-3388",2009,46);
if (v1 instanceof Onibus) System.out.println("Ônibus");
else if (v1 instanceof Caminhao) System.out.println("Caminhão");
```

Manipulação Polimórfica de Objetos

- **Conversão de tipo:**
  - Uma referência pode ser convertida em tempo de execução
  - Só é possível converter para uma de suas subclasses
  - Pode gerar ums ClassCastException

- **Exemplo 1:**

```
Veiculo v1 = new Onibus("AAA-3388",2009,46);
Onibus bus = (Onibus)v1;

String str = "Dados do ônibus: " +
      "\nPlaca: " + v1.getPlaca() +
      "\nAno: " + v1.getAno() +
      "\nAssentos: " + bus.getAssentos();
JOptionPane.showMessageDialog(null, str);
```

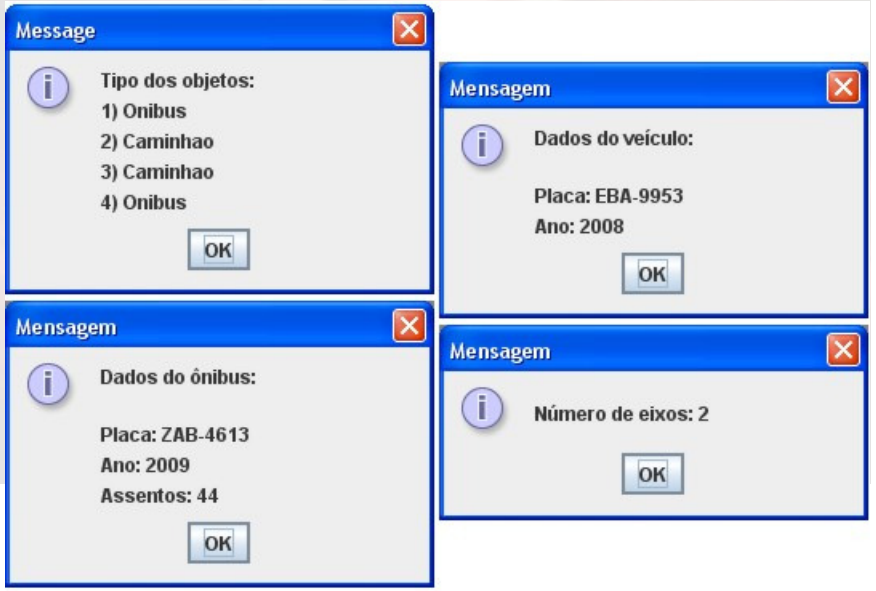
Manipulação Polimórfica de Objetos

- **Exemplo 2: ClassCastException**

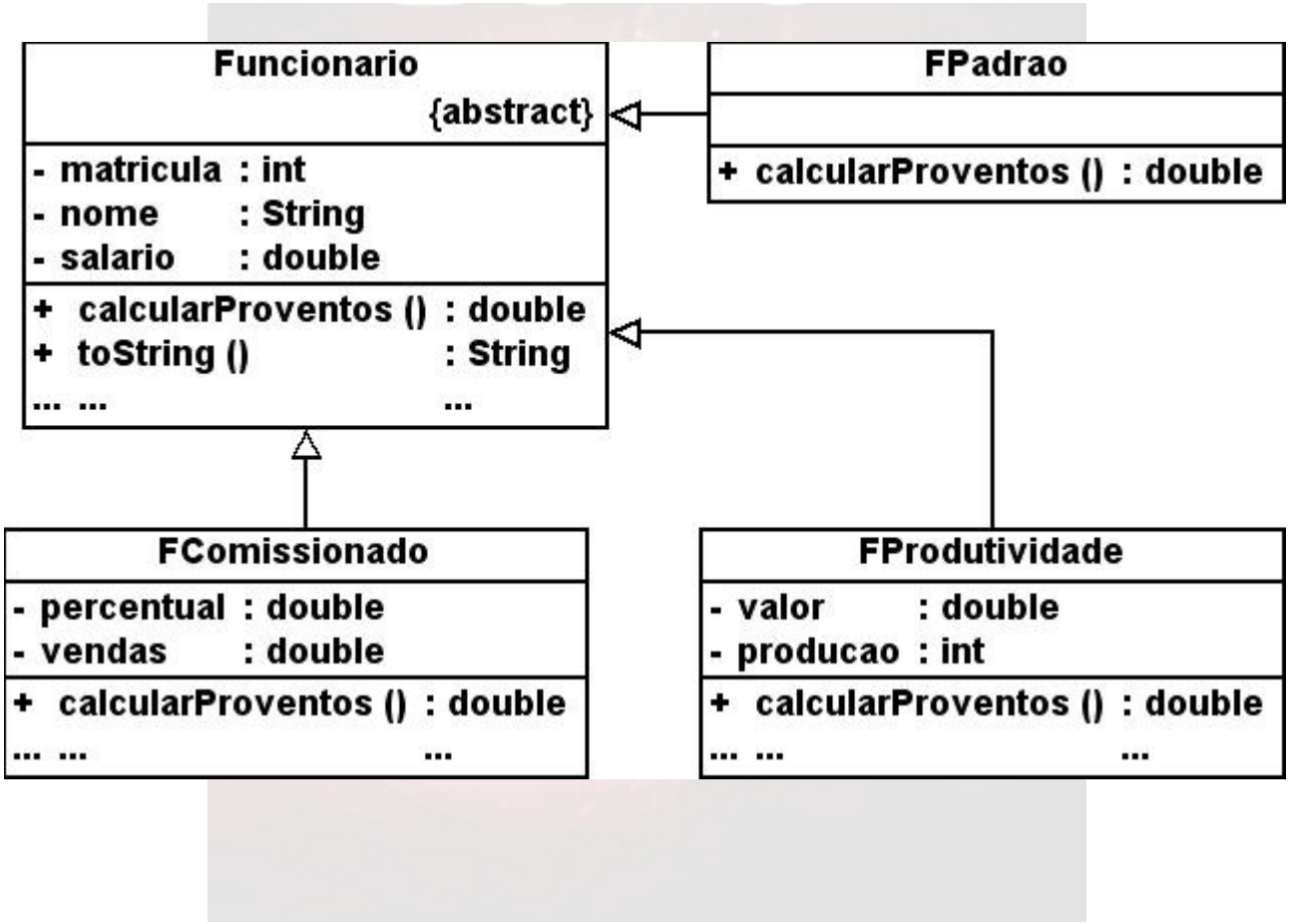
```
Veiculo v1 = new Onibus("AAA-3388",2009,46);
Caminhao cam = (Caminhao)v1;

String str = "Dados do ônibus: " +
      "\nPlaca: " + v1.getPlaca() +
      "\nAno: " + v1.getAno() +
      "\nEixos: " + cam.getEixos();
JOptionPane.showMessageDialog(null, str);
```

Manipulação Polimórfica de Objetos

- **Código 14.17 – CadastroVeiculos.java**
  - Crie um vetor capaz de armazenar até 100 cadastros de veículos.
  - Crie algumas instâncias da classe Onibus e grave-as no vetor.
  - Crie algumas instâncias da classe Caminhao e grave-as no vetor.
  - Escreva uma estrutura de repetição que percorra o vetor e recupere o nome da classe de cada objeto.
  - Escreva outra estrutura de repetição que invoque o método exibirDados( ) de cada objeto do vetor.



Estudo de Caso: Folha de Pagamento

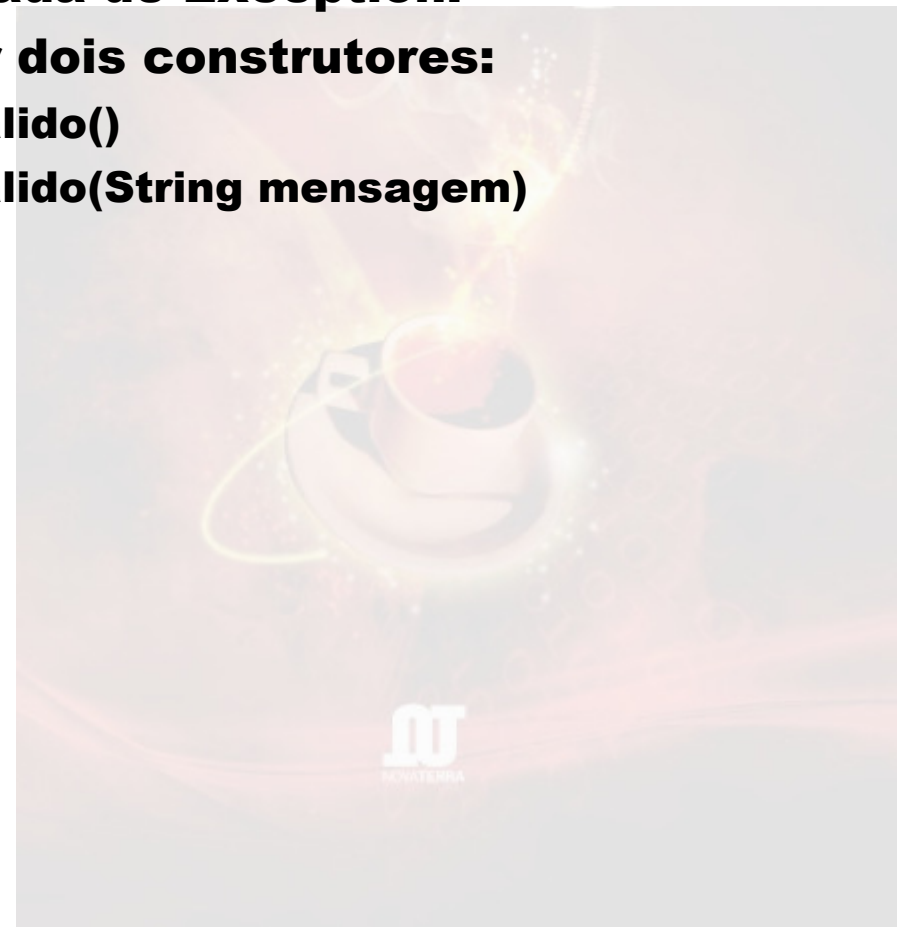


Estudo de Caso: Folha de Pagamento

- **Descrição do estudo de caso:**
  - **Objetivo: cadastrar os funcionários de uma empresa e calcular os proventos devidos a cada um deles por um mês de trabalho.**
  - **Proventos: representam quaisquer valores devidos aos funcionários a título de remuneração.**
  - **Serão considerados apenas três tipos de proventos:**
    - **Salário fixo: devido a todos os funcionários.**
    - **Comissão: adicional devido pelas vendas realizadas pelos funcionários que são comissionados**
    - **Produtividade: adicional devido aos funcionários que trabalham diretamente na produção e que recebem determinado valor por unidade produzida**

Estudo de Caso: Folha de Pagamento

- **Código 14.18 – EDadoInvalido.java**
  - **Classe derivada de Exception.**
  - **Implementar dois construtores:**
    - **EDadoInvalido()**
    - **EDadoInvalido(String mensagem)**



Estudo de Caso: Folha de Pagamento

- **Classe Funcionario:**
  - Representará as características comuns a todos os funcionários.
  - Será abstrata.
  - Representará três atributos do funcionário:
    - A matrícula
    - O seu nome completo
    - O salário fixo
  - O método calcularProventos( ) será um método abstrato.
    - Deverá calcular o valor total a ser pago
  - O método toString( ) será sobrescrito (observe figura abaixo).
    - Deve invocar o método calcularProventos( )

Estudo de Caso: Folha de Pagamento

- **Classe FPadrao:**
  - Representa os funcionários que recebem apenas o salário fixo.
  - Seus proventos equivalem ao valor deste salário fixo.
    - Valor retornado pelo método calcularProventos( ).
- **Classe FComissionado:**
  - Representa os funcionários que recebem o salário fixo e um percentual sobre as vendas realizadas.
  - Adiciona dois novos atributos: percentual e vendas.
  - O método calcularProventos( ) soma o salário fixo com o valor devido a título de comissão
- **Classe FProdutividade:**
  - Representa os funcionários que recebem o salário fixo e um adicional por cada unidade produzida.
  - Adiciona dois novos atributos: valor e producao.
  - O método calcularProventos( ) deve somar o salário fixo com o que é devido a título de produtividade.

Estudo de Caso: Folha de Pagamento

- **Código 14.23 – TesteFolha.java**
  - **Crie um vetor que suporte até 10 cadastros de funcionários.**
  - **Crie uma instância de cada uma das seguintes classes: FPadrao, FComissionado e FProdutividade.**
  - **Grave todas estas instâncias no vetor.**
  - **Escreva uma estrutura de repetição que percorra o vetor e recupere a representação textual de cada objeto.**
  - **Exiba todos os dados recuperados através de um diálogo.**

Estudo de Caso: Folha de Pagamento

- **Código 14.24 –FolhaPagamento.java**
  - **Crie um vetor que suporte até 1000 cadastros de funcionários.**
  - **Permita que seja cadastrado qualquer tipo de funcionário.**
    - **O tipo deverá ser indicado antes de iniciar o cadastro**
  - **Ao final, gere a folha de pagamento para todos os funcionários cadastrados.**

Exercício 1

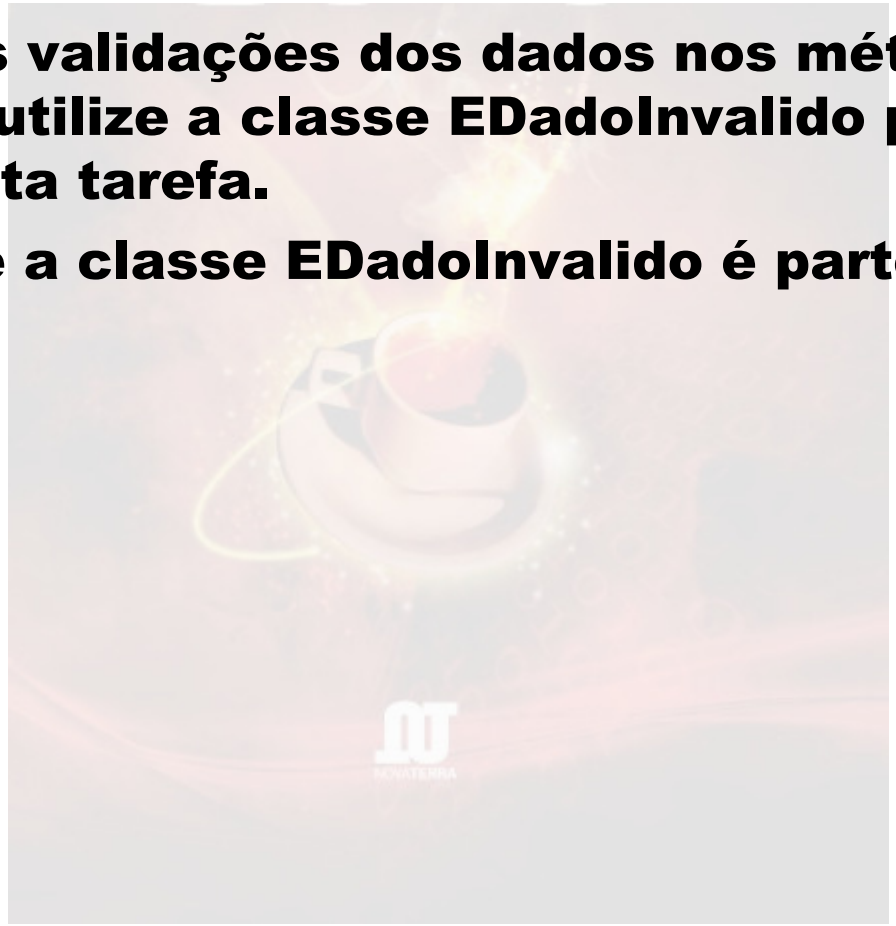
- Suponha que você deve desenvolver um sistema para ser utilizado por empresas que comercializam materiais de construção. O objetivo do sistema é simplesmente permitir o cadastro de seus clientes.
- Entretanto, estas empresas vendem materiais de construção tanto para pessoas físicas quanto para pessoas jurídicas (outras empresas). O sistema que você deve implementar precisa manter certos dados cadastrais que são comuns a todos os clientes e também precisa manter dados cadastrais que somente as pessoas físicas possuem ou que somente as pessoas jurídicas possuem.

Exercício 1

- Os dados que deverão ser registrados nos cadastros de todos os clientes são os seguintes:
  - Código: um número inteiro que servirá para identificar cada cliente no sistema.
  - Nome: será o nome completo no caso de pessoas físicas e o nome fantasia no caso de empresas.
  - Telefone: será o telefone residencial no caso de pessoas físicas e o telefone comercial no caso de pessoas jurídicas.
  - E-mail: será qualquer e-mail que possa ser utilizado para entrar em contato com o cliente.
- Os dados que somente deverão ser registrados nos cadastros de clientes que são pessoas físicas são os seguintes: o número de seu telefone celular, o número de seu RG e o número de seu CPF.
- Os dados que somente deverão ser registrados nos cadastros de clientes que são pessoas jurídicas são os seguintes: sua razão social, o número de sua Inscrição Estadual (IE) e seu número no Cadastro Nacional de Pessoas Jurídicas (CNPJ).

Exercício 1

- Este sistema não deve permitir que dados inválidos sejam gravados no cadastro de um cliente.
- Implemente as validações dos dados nos métodos de escrita das classes e utilize a classe EDadoInvalido para a realização desta tarefa.
- Lembre-se que a classe EDadoInvalido é parte do último exemplo.

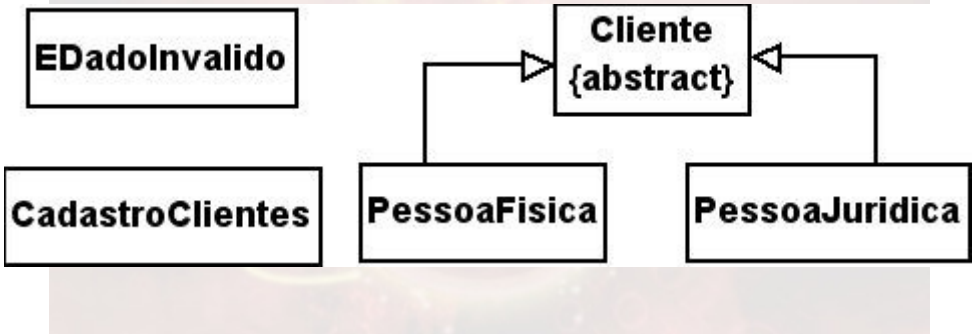


Exercício 1

- As regras que devem ser implementadas para validar os dados cadastrais dos clientes são as seguintes:
  - Código: deve ser um número inteiro de 1 a 1.000.000.
  - Nome: deve ser um texto contendo pelo menos duas palavras com pelo menos 2 caracteres cada uma.
  - Telefone: deve ser composto pelo código de área, pelo prefixo e pelo número e ser informado exatamente no formato (99)9999-9999.
  - E-mail: deve ser um texto contendo pelo menos dois caracteres separados por um símbolo de arroba e não pode conter mais que um símbolo de arroba.
  - Celular: segue as mesmas restrições descritas para o telefone.
  - RG: deve conter de 5 a 15 dígitos numéricos e o último dígito deve estar separado por um hífen.
  - CPF: deve ser gravado no formato 999.999.999-99.
  - Razão social: segue as mesmas restrições descritas para o nome.
  - IE: deve ser um texto contendo de 1 a 15 caracteres.
  - CNPJ: deve ser informado no formato 00.000.000/0000-00.

Exercício 1

- As classes que devem ser implementadas para compor este sistema são aquelas apresentadas através da figura abaixo.



Exercício 2

- Suponha que você deve desenvolver um sistema para ser utilizado por farmácias.
- O objetivo do sistema é permitir o cadastro de três diferentes tipos de parceiros dos quais estas empresas dependem: os funcionários, os clientes e os fornecedores.
- Considerar-se-á que todos os funcionários e clientes devem ser pessoas físicas e que todos os fornecedores devem ser pessoas jurídicas.
- O sistema que você deve implementar precisa manter certos dados cadastrais que são comuns a todos os parceiros e também precisa manter dados cadastrais que são específicos para funcionários, clientes e fornecedores.

Exercício 2

- Os dados que deverão ser registrados nos cadastros de todos os parceiros são os seguintes:
  - Código: um número inteiro que servirá para identificar cada cliente no sistema.
  - Nome: será o nome completo no caso de funcionários e clientes e o nome fantasia no caso dos fornecedores.
  - Telefone: será o telefone residencial no caso de funcionários e clientes e o telefone comercial no caso dos fornecedores.
  - E-mail: será qualquer e-mail que possa ser utilizado para entrar em contato com o parceiro.
- Os dados que somente deverão ser registrados nos cadastros de funcionários são os seguintes: o número de sua CTPS (Carteira de Trabalho e Previdência Social) e a série de sua CTPS.
- Os dados que somente deverão ser registrados nos cadastros de clientes são os seguintes: o número de seu telefone celular, o número de seu RG e o número de seu CPF.

Exercício 2

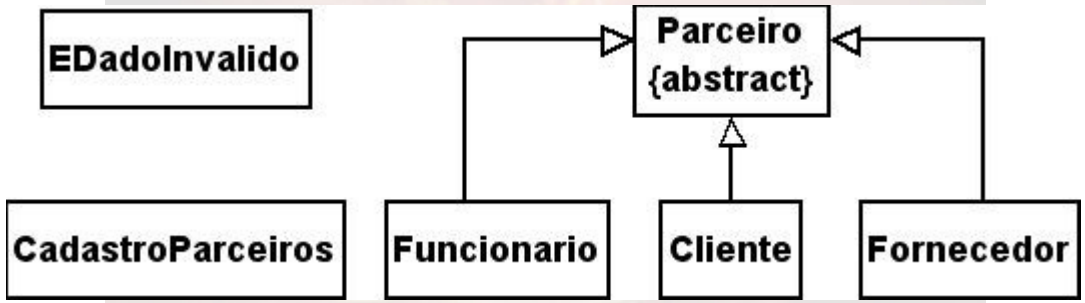
- Os dados que somente deverão ser registrados nos cadastros de fornecedores são os seguintes: sua razão social, o número de sua Inscrição Estadual (IE) e seu número no Cadastro Nacional de Pessoas Jurídicas (CNPJ).
- Este sistema não deve permitir que dados inválidos sejam gravados no cadastro de um parceiro.
- Implemente as validações dos dados nos métodos de escrita das classes e utilize o classe EDadoInvalido para a realização desta tarefa.

Exercício 2

- As regras que devem ser implementadas para validar os dados cadastrais dos parceiros são as seguintes:
  - Código: deve ser um número inteiro de 1 a 1.000.000.
  - Nome: deve ser um texto contendo pelo menos duas palavras com pelo menos 2 caracteres cada uma.
  - Telefone: deve ser composto pelo código de área, pelo prefixo e pelo número e ser informado exatamente no formato (99)9999-9999.
  - E-mail: deve ser um texto contendo pelo menos dois caracteres separados por um símbolo de arroba e não pode conter mais que um símbolo de arroba.
  - Número da CTPS: deve ser um número inteiro de 1 a 1.000.000.000.
  - Série da CTPS: deve ser um número inteiro de 1 a 30.000.
  - Celular: segue as mesmas restrições descritas para o telefone.
  - RG: deve conter de 5 a 15 dígitos numéricos e o último dígito deve estar separado por um hífen.
  - CPF: deve ser gravado no formato 999.999.999-99.
  - Razão social: segue as mesmas restrições descritas para o nome.
  - IE: deve ser um texto contendo de 1 a 15 caracteres.
  - CNPJ: deve ser informado no formato 00.000.000/0000-00.

Exercício 2

- As classes que devem ser implementadas para compor este sistema são aquelas apresentadas através da figura abaixo.



Contato

Com o autor:

Rui Rossi dos Santos
E-mail: livros@ruirossi.pro.br
Web Site: http://www.ruirossi.pro.br

Com a editora:

Editora NovaTerra
Telefone: (21) 2218-5314
Web Site: http://www.editoranovaterra.com.br