Introdução à Programação Orientada a Objetos

Programação Orientada a Objetos — QXD0007

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

1º semestre/2022

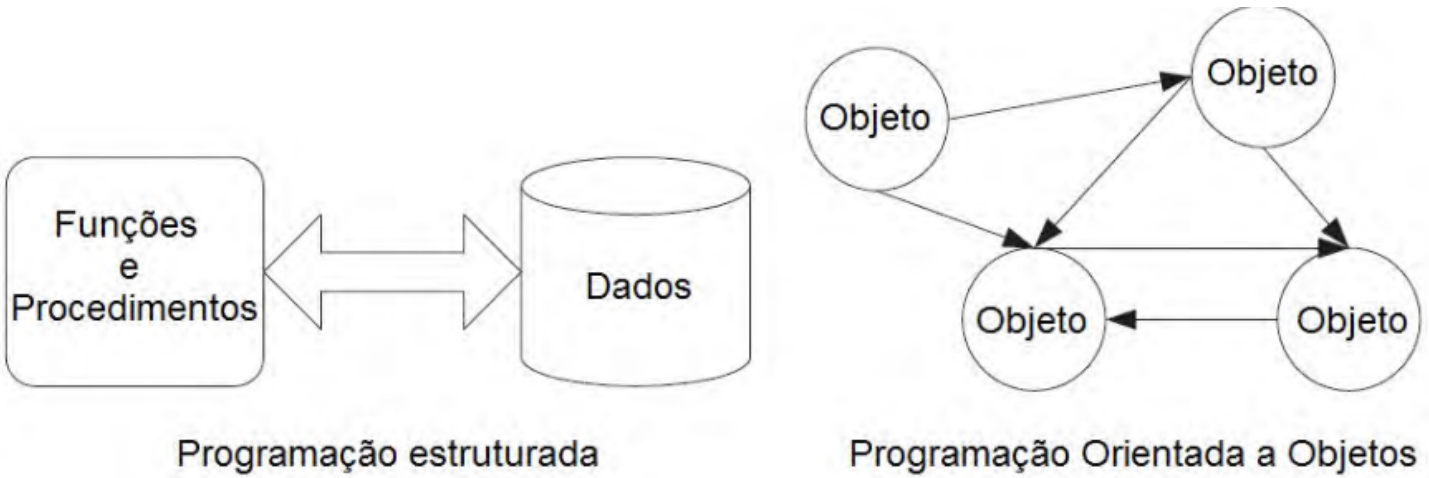
Resumo deste Tópico

- Modelos
- Modelos e Programação Orientada a Objetos
- Classes, atributos e métodos
- Objetos: comportamento, estado e identidade
- Criação de Classes em Java

Introdução

Paradigma Orientado a Objetos

- A orientação a objetos é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.



Paradigma Orientado a Objetos

- A Programação Orientada a Objetos (POO) tem como principal característica uma melhor e maior expressividade das necessidades do mundo real.
  - Possibilita criar unidades de código mais próximas da forma como pensamos e agimos, assim facilitando o processo de transformação das necessidades diárias para uma linguagem orientada a objetos.

Paradigma Orientado a Objetos

- A Programação Orientada a Objetos (POO) tem como principal característica uma melhor e maior expressividade das necessidades do mundo real.
  - Possibilita criar unidades de código mais próximas da forma como pensamos e agimos, assim facilitando o processo de transformação das necessidades diárias para uma linguagem orientada a objetos.

- POO considera que os dados a serem processados e os mecanismos de processamento desses dados devem ser considerados em conjunto.
  - Precisamos de modelos que representem conjuntamente dados e operações sobre estes dados.

Modelos

O que são modelos?

Modelos são representações simplificadas de objetos, pessoas, itens, processos, conceitos, ideias, etc. usados comumente por pessoas no seu dia-a-dia, independente do uso de computadores.

O que são modelos?

Modelos são representações simplificadas de objetos, pessoas, itens, processos, conceitos, ideias, etc. usados comumente por pessoas no seu dia-a-dia, independente do uso de computadores.

Exemplo: Considere o **Restaurante Caseiro Dona Maria**, que serve refeições por quilo, e onde o gerente, que também é a pessoa que fica na balança e no caixa, anota os pesos dos pratos dos clientes e os pedidos que os garçons trazem em um quadro branco.

O que são modelos?

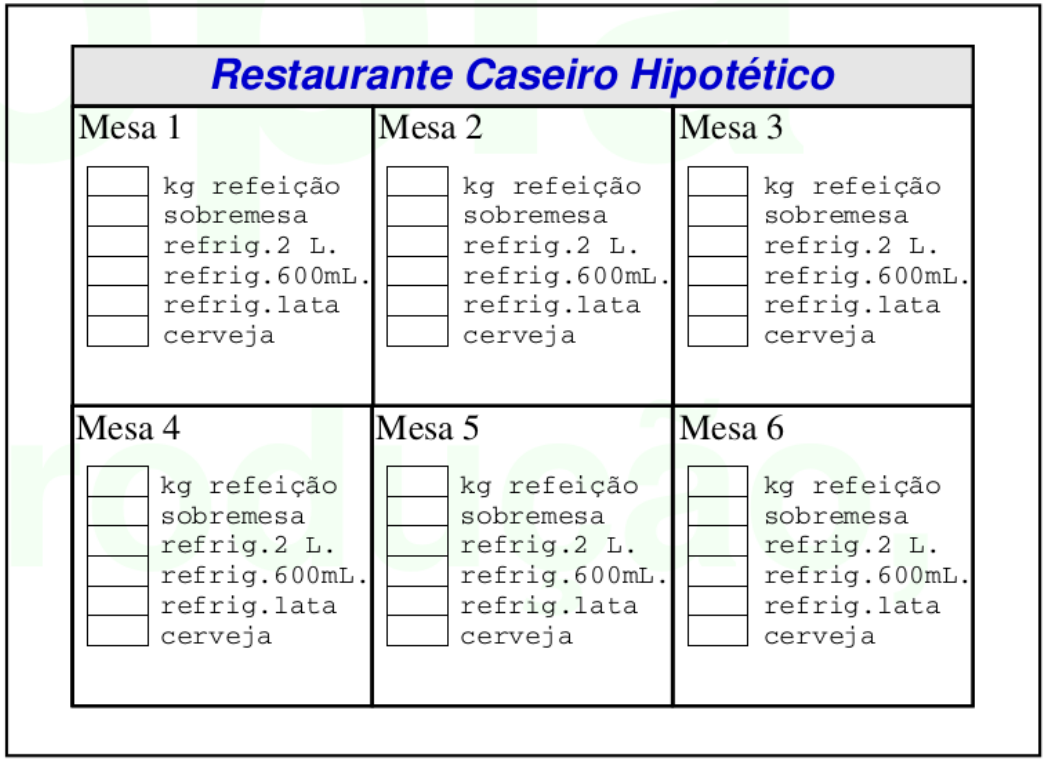
Modelos são representações simplificadas de objetos, pessoas, itens, processos, conceitos, ideias, etc. usados comumente por pessoas no seu dia-a-dia, independente do uso de computadores.

Exemplo: Considere o **Restaurante Caseiro Dona Maria**, que serve refeições por quilo, e onde o gerente, que também é a pessoa que fica na balança e no caixa, anota os pesos dos pratos dos clientes e os pedidos que os garçons trazem em um quadro branco.

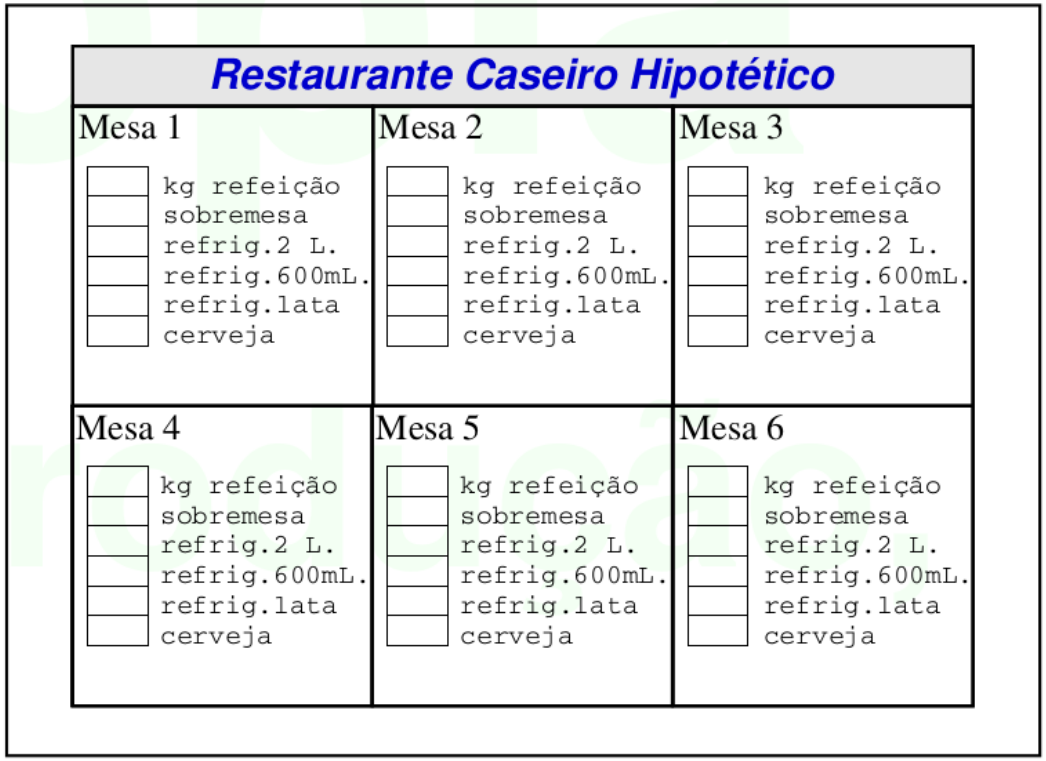
Quando os itens dos pedidos são servidos, o gerente anota, ao lado do item no quadro-branco, o número de itens ou o peso do prato.

Quando o cliente pede a conta, o gerente se refere ao quadro-branco para calcular o valor devido.

Modelo do quadro-branco do Restaurante



Modelo do quadro-branco do Restaurante



- O quando-branco é um modelo do restaurante. Representa de forma simplificada as informações que são necessárias para a contabilização dos pedidos feitos pelos clientes.
  - Quais informações podemos extrair desse modelo?

Características dos modelos

(1) Um modelo comumente contém operações associadas a ele, que são listas de comandos que processarão os dados do modelo.

Por exemplo, no Restaurante Caseiro, algumas operações seriam a inclusão de um pedido para uma mesa, modificação do *status* de uma mesa (pedido servido ou não), encerramento dos pedidos dos clientes de uma mesa e a apresentação da conta para os clientes.

Características dos modelos

(2) Como o modelo é uma simplificação do mundo real, os dados contidos no modelo devem ser relevantes à abstração do mundo real sendo feita. $\Longrightarrow$ Depende do contexto

Características dos modelos

(2) Como o modelo é uma simplificação do mundo real, os dados contidos no modelo devem ser relevantes à abstração do mundo real sendo feita. $\Longrightarrow$ Depende do contexto

Por exemplo, a representação das informações sobre uma pessoa pode ser feita de maneira diferente dependendo do contexto:

- **Pessoa como empregado de empresa:** para fins de processamento de folha de pagamento, seria necessária a representação de: `nome`, `cargo`, `salario` e `horasExtrasTrabalhadas`. Este modelo poderia conter as operações `calculaSalario` e `aumentaSalario`.

Características dos modelos

(2) Como o modelo é uma simplificação do mundo real, os dados contidos no modelo devem ser relevantes à abstração do mundo real sendo feita. $\Longrightarrow$ Depende do contexto

Por exemplo, a representação das informações sobre uma pessoa pode ser feita de maneira diferente dependendo do contexto:

- **Pessoa como empregado de empresa:** para fins de processamento de folha de pagamento, seria necessária a representação de: `nome`, `cargo`, `salario` e `horasExtrasTrabalhadas`. Este modelo poderia conter as operações `calculaSalario` e `aumentaSalario`.

- **Pessoa como paciente de clínica médica:** seria necessário representar o `nome`, `sexo`, `idade`, `altura`, `peso` e o `históricoDeConsultas` do paciente. Este modelo poderia conter as operações `verificaObesidade` e `adicionaInformaçãoAoHistorico`.

Características dos modelos

(3) Modelos podem conter submodelos e ser parte de outros modelos

- O quadro-branco que representa um restaurante pode ser composto de diversos quadrados no quadro que representam mesas ou comandas, cada um contendo os dados relativos aos pedidos daquela mesa e ações correspondentes.

Características dos modelos

(3) Modelos podem conter submodelos e ser parte de outros modelos

- O quadro-branco que representa um restaurante pode ser composto de diversos quadrados no quadro que representam mesas ou comandas, cada um contendo os dados relativos aos pedidos daquela mesa e ações correspondentes.

(4) Modelos podem ser reutilizados para representar diferentes objetos, pessoas ou itens.

Características dos modelos

(3) Modelos podem conter submodelos e ser parte de outros modelos

- O quadro-branco que representa um restaurante pode ser composto de diversos quadrados no quadro que representam mesas ou comandas, cada um contendo os dados relativos aos pedidos daquela mesa e ações correspondentes.

(4) Modelos podem ser reutilizados para representar diferentes objetos, pessoas ou itens.

A criação e o uso de modelos é uma tarefa natural e a extensão desta abordagem à programação deu origem ao paradigma *Programação Orientada a Objetos*.

POO

- Programação orientada a objetos (POO) é um paradigma de programação de computadores onde se usam classes (modelos) e objetos, criados a partir destas classes, para representar e processar dados usando programas de computador.

POO

- Programação orientada a objetos (POO) é um paradigma de programação de computadores onde se usam classes (modelos) e objetos, criados a partir destas classes, para representar e processar dados usando programas de computador.

- Os dados pertencentes aos modelos são representados por tipos de dados nativos ou também podem ser representados por classes já existentes na linguagem ou por outras classes criadas pelo programador.

POO

- Programação orientada a objetos (POO) é um paradigma de programação de computadores onde se usam classes (modelos) e objetos, criados a partir destas classes, para representar e processar dados usando programas de computador.

- Os dados pertencentes aos modelos são representados por tipos de dados nativos ou também podem ser representados por classes já existentes na linguagem ou por outras classes criadas pelo programador.

- Os dados e as operações que os manipulam são considerados em conjunto, como se fossem uma unidade.

Mais exemplos de modelos

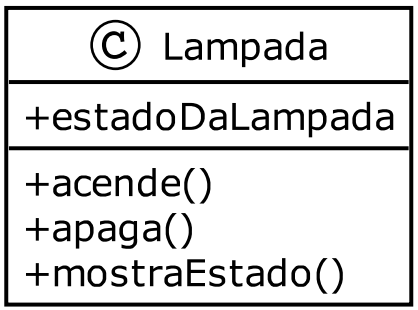
Lâmpada Incandescente

- Consideremos uma lâmpada incandescente que tem um dado básico, que é seu estado (“ligada” ou “desligada”).
- As operações que podem ser efetuadas na lâmpada são simples: podemos ligá-la ou desligá-la
- Para saber se uma lâmpada está ligada ou desligada podemos pedir que uma operação mostre o valor do estado.

Lâmpada Incandescente

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

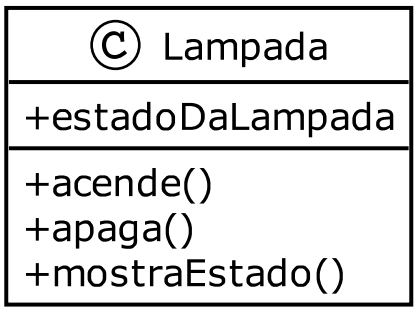
- Consideremos uma lâmpada incandescente que tem um dado básico, que é seu estado ("ligada" ou "desligada").
- As operações que podem ser efetuadas na lâmpada são simples: podemos ligá-la ou desligá-la
- Para saber se uma lâmpada está ligada ou desligada podemos pedir que uma operação mostre o valor do estado.
- A figura abaixo mostra uma variante do diagrama de classes da Linguagem Unificada de Modelagem (*Unified Modeling Language*, UML)



Lâmpada Incandescente

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Consideremos uma lâmpada incandescente que tem um dado básico, que é seu estado ("ligada" ou "desligada").
- As operações que podem ser efetuadas na lâmpada são simples: podemos ligá-la ou desligá-la
- Para saber se uma lâmpada está ligada ou desligada podemos pedir que uma operação mostre o valor do estado.
- A figura abaixo mostra uma variante do diagrama de classes da Linguagem Unificada de Modelagem (*Unified Modeling Language*, UML)



- Se as lâmpadas a serem representadas neste modelo fossem usadas em uma aplicação de controle de qualidade, quais dados seriam úteis?

Lâmpada Incandescente

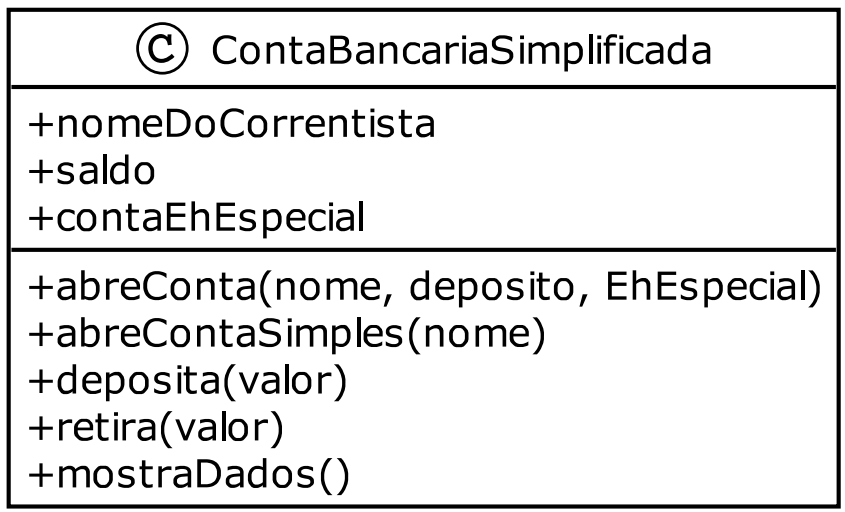
Pseudocódigo do modelo

```
1  modelo Lampada // representa uma lampada em uso
2  inicio do modelo
3    dado estadoDaLampada;
4
5    operacao acende() // acende a lampada
6      inicio
7        estadoDaLampada = aceso;
8      fim
9
10   operacao apaga() // apaga a lampada
11     inicio
12       estadoDaLampada = apagado;
13     fim
14
15   operacao mostraEstado() // mostra estado da lampada
16     inicio
17       se (estadoDaLampada == aceso)
18         imprime "A lampada esta acesa";
19       senao
20         imprime "A lampada esta apagada";
21     fim
22 fim do modelo
```

Uma conta bancária simplificada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Este modelo de conta bancária somente representa o nome do correntista, o saldo da conta e se a conta é especial ou não
- Se a conta for especial, o correntista terá o direito de retirar mais dinheiro do que tem no saldo (ficar com o saldo negativo)



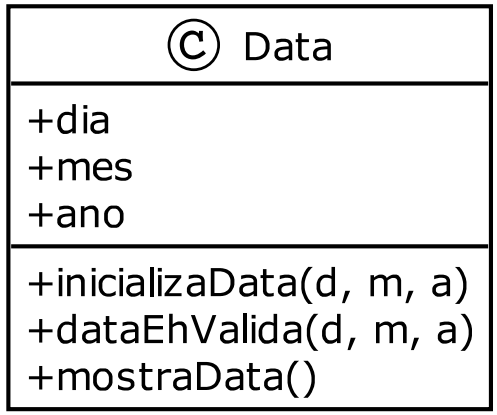
Conta bancária simplificada


```
1  modelo ContaBancariaSimplificada
2  inicio do modelo
3      dado nomeDoCorrentista, saldo, contaEhEspecial;
4
5      // Inicializa todos os dados do modelo
6      operacao abreConta(nome, deposito, ehEspecial)
7          inicio
8              nomeDoCorrentista = nome;
9              saldo = deposito;
10             contaEhEspecial = ehEspecial;
11         fim
12
13     operacao abreContaSimples(nome)
14         inicio
15             nomeDoCorrentista = nome;
16             saldo = 0.0;
17             contaEhEspecial = falso;
18         fim
19
20     // Deposita um valor na conta
21     operacao deposita(valor)
22         inicio
23             saldo = saldo + valor;
24         fim
```

Conta bancária simplificada (Cont.)


```
25      // Retira um valor da conta
26      operacao retira(valor)
27          inicio
28              se(contaEhEspecial == falso) // nao eh especial
29                  inicio
30                      se(valor <= saldo) // se existe saldo
31                          saldo = saldo - valor;
32                  fim
33              senao // Conta especial, pode retirar a vontade
34                  saldo = saldo - valor
35          fim
36
37      // Mostra os dados da conta, imprimindo seus valores
38      operacao mostraDados()
39      inicio
40          imprime "O nome do correntista eh: ";
41          imprime nomeDoCorrentista;
42          imprime "O saldo eh: ";
43          imprime saldo;
44          se(contaEhEspecial) imprime "A conta eh especial";
45          senao imprime "A conta eh comum";
46      fim
47  fim do modelo
```

Modelo Data



Uma variante do diagrama UML para o modelo Data

Modelo: Data

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  modelo Data
2  inicio do modelo
3      dado dia, mes, ano; // componentes da data
4
5      // Inicializa simultaneamente todos os dados do modelo
6      operacao inicializaData(d, m, a)
7          inicio
8              se(dataEhValida(d,m,a))
9                  inicio
10                     dia = d;
11                     mes = m;
12                     ano = a;
13                 fim
14             senao
15                 inicio
16                     dia = 0;
17                     mes = 0;
18                     ano = 0;
19                 fim
20         fim
```

Modelo: Data (Cont.)


```
21      // Verifica se a data eh valida
22      operacao dataEhValida(d, m, a)
23      inicio
24          se((dia>=1) e (dia<=31) e (mes>=1) e (mes<=12))
25              retorna verdadeiro;
26          senao
27              retorna falso;
28      fim
29
30      // Mostra a data imprimindo valores de seus dados
31      operacao mostraData()
32          inicio
33              imprime dia;
34              imprime "/";
35              imprime mes;
36              imprime "/";
37              imprime ano;
38          fim
39  fim do modelo
```

Classes e Objetos

Classes

Classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de **métodos** e estados possíveis destes objetos através de **atributos**. Resumidamente, classe é o modelo a partir do qual os objetos são feitos.

Classes

Classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de **métodos** e estados possíveis destes objetos através de **atributos**. Resumidamente, classe é o modelo a partir do qual os objetos são feitos.

- Analogia: Pense em uma classe como a planta de uma casa. Os objetos são as próprias casas.

Classes

Classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de **métodos** e estados possíveis destes objetos através de **atributos**. Resumidamente, classe é o modelo a partir do qual os objetos são feitos.

- Analogia: Pense em uma classe como a planta de uma casa. Os objetos são as próprias casas.

- Para representação de dados específicos usando classes é preciso criar **objetos** ou **instâncias** da classe.
- Um objeto ou instância é uma materialização da classe e, assim, pode ser usado para representar dados e executar operações.

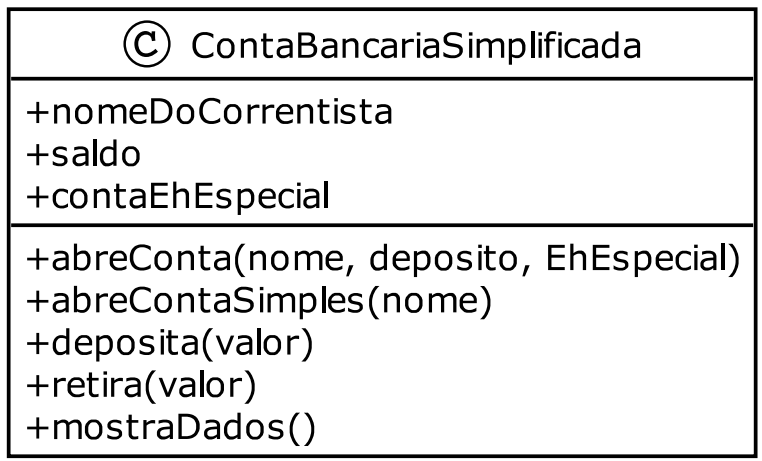
Classes e atributos

- Considerando os exemplos de modelos mostrados anteriormente, os modelos seriam as classes e a partir destas classes poderíamos criar instâncias.

Classes e atributos

- Considerando os exemplos de modelos mostrados anteriormente, os modelos seriam as classes e a partir destas classes poderíamos criar instâncias.

- Os dados contidos em uma classe são chamados de campos ou atributos daquela classe.
  - Cada campo deve ter um nome e ser de um tipo, que será um tipo nativo do Java ou uma classe existente na linguagem ou definida pelo programador.



Classes e métodos

- As operações contidas em uma classe são chamadas de métodos.

- Métodos são geralmente chamados explicitamente a partir de outros trechos de código na classe que os contém ou a partir de outras classes.

Classes e métodos

- As operações contidas em uma classe são chamadas de métodos.
- Métodos são geralmente chamados explicitamente a partir de outros trechos de código na classe que os contém ou a partir de outras classes.
- Métodos podem receber argumentos na forma de valores de tipos nativos de dados ou referências a instâncias de classes.
- Métodos podem retornar nenhum valor ou retornar um único valor.

Identificando as classes em um problema

- Ao projetar um sistema orientado a objetos, devemos começar identificando as classes que farão parte do nosso sistema e, em seguida, adicionando atributos e métodos a estas classes.

Identificando as classes em um problema

- Ao projetar um sistema orientado a objetos, devemos começar identificando as classes que farão parte do nosso sistema e, em seguida, adicionando atributos e métodos a estas classes.

- Uma regra simples na identificação de **classes** é procurar **substantivos** na análise de problemas. **Métodos**, por outro lado, correspondem a **verbos**.

Identificando as classes em um problema

- Ao projetar um sistema orientado a objetos, devemos começar identificando as classes que farão parte do nosso sistema e, em seguida, adicionando atributos e métodos a estas classes.

- Uma regra simples na identificação de **classes** é procurar **substantivos** na análise de problemas. **Métodos**, por outro lado, correspondem a **verbos**.

- Por exemplo, em um sistema de processamento de pedidos, alguns substantivos que surgem são: Item, Pedido, Endereço de entrega, Forma de pagamento e Conta. Esses substantivos podem levar às classes `Item`, `Pedido`, `Pagamento`, `Conta` e assim por diante.

Identificando as classes em um problema

- Ao projetar um sistema orientado a objetos, devemos começar identificando as classes que farão parte do nosso sistema e, em seguida, adicionando atributos e métodos a estas classes.

- Uma regra simples na identificação de **classes** é procurar **substantivos** na análise de problemas. **Métodos**, por outro lado, correspondem a **verbos**.

- Por exemplo, em um sistema de processamento de pedidos, alguns substantivos que surgem são: Item, Pedido, Endereço de entrega, Forma de pagamento e Conta. Esses substantivos podem levar às classes `Item`, `Pedido`, `Pagamento`, `Conta` e assim por diante.

- Depois, procuramos por verbos. Itens são adicionados a Pedidos, Pedidos são enviados ou cancelados. Pagamentos são aplicados a pedidos. A partir de cada verbo, identificamos os métodos enviar, adicionar, cancelar, aplicar, e identificamos o objeto que tenha a maior responsabilidade de executá-lo.

Criando Classes

Criando Classes em Java

Sintaxe Básica

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

- O nome da classe deve ser um identificador válido no Java.
  - **Obs.:** tradicionalmente, os nomes de classes começam com caracteres maiúsculos e alternam entre palavras.
    Exemplo: `RegistroAcademico` e `ContaDeLuz`.

- O conteúdo da classe é delimitado por um bloco de comandos que inicia com { e termina com }.

Exemplo de classe em Java


```
1 /**
2  * A classe Vazia não possui atributos ou métodos
3  */
4 class Vazia { // esta é a declaração da classe
5
6     /* Se houvesse atributos ou métodos para a classe Vazia,
7     eles deveriam estar aqui dentro */
8
9 } // fim da classe Vazia
```

- Esta classe deve ser salva em um arquivo chamado `Vazia.java`

Declarando atributos de classes em Java

- A declaração de campos é simples: basta declarar o tipo de dado, seguido dos nomes dos campos que serão daquele tipo.

```
1  /**
2   * A classe RegistroAcademicoSimples contem somente alguns
3   * campos que exemplificam as declaracoes de campos e tipos
4   * de dados em Java. Esta classe depende da classe Data para
5   * ser compilada com sucesso.
6   */
7  class RegistroAcademicoSimples { // declaracao
8                                   // da classe
9
10     String nomeDoAluno;
11     int numeroDeMatricula;
12     Data dataDeNascimento = new Data();
13     boolean ehBolsista;
14     short anoDeMatricula;
15
16     /* Se houvesse metodos, eles seriam declarados aqui */
17
18 } // fim da classe RegistroAcademicoSimples
```

Escopo

- O escopo dos atributos de uma classe é toda a classe, mesmo que os campos estejam declarados depois dos métodos que os usam.

- Variáveis e instâncias declaradas dentro de métodos só serão válidas dentro desses métodos

- Dentro de métodos e blocos de comandos, a ordem de declaração de variáveis e referências a instâncias é considerada. Variáveis dentro de métodos só podem ser usadas depois de declaradas.

Exemplo ilustrando escopo de variáveis

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  /**
2   * A classe Triangulo representa os três
3   * lados de um triângulo qualquer
4   */
5  public class Triangulo {
6      /**
7       * Declaração dos outros campos da classe
8       */
9      private float lado2, lado3; // só para ilustrar.
10                                 // eles devem estar todos juntos
11     // declaração de um dos campos da classe
12     private float lado1;
13
14     Triangulo(int lado1, int lado2, int lado3) {
15         this.lado1 = lado1;
16         this.lado2 = lado2;
17         this.lado3 = lado3;
18     }
```

Exemplo ilustrando escopo de variáveis

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
20      /**
21       * O método equilátero verifica se o triângulo
22       * é equilátero ou não
23       * @return true se triangulo equilátero, false se não
24       */
25      boolean ehEquilatero() {
26          if((lado1 == lado2) && (lado2 == lado3))
27              return true;
28          else
29              return false;
30      }
31
32      /**
33       * O método perimetro calcula o peimetro do
34       * triângulo usando seus lados.
35       * @return o perímetro do triângulo representado por esta
        classe
36       */
37      float perimetro() {
38          int x = 3;
39          return lado1 + lado2 + lado3 + x;
40      }
41
42  }
```

Métodos em classes Java

Os métodos também seguem regras rígidas para a sua criação:

- Nomes de métodos seguem as mesmas regras de nomes de atributos.
- Métodos não podem ser criados dentro de outros métodos, nem fora da classe à qual pertencem.
- os nomes de métodos devem seguir o padrão camelCase.
  - Exemplo: `mostraDados`, `acende` e `inicializaData`.

Sintaxe de criação de método:

```
<modificador de acesso> <tipo de retorno> <nome do método>
(lista de argumentos) {
    .... comandos ....
}
```

Exemplo de Classe Java


```
1  class DataSimples { // declaracao da classe
2
3      byte dia, mes; // atributos ou campos
4      short ano;
5
6      /**
7       * O metodo inicializaDataSimples recebe argumentos
8       * para inicializar os campos da classe DataSimples.
9       * @param d o argumento correspondente ao campo dia
10      * @param m o argumento correspondente ao campo mes
11      * @param a o argumento correspondente ao campo ano
12      */
13     void inicializaDataSimples(byte d, byte m, short a) {
14         if(dataEhValida(d,m,a)) {
15             dia = d;
16             mes = m;
17             ano = a;
18         }
19         else {
20             dia = mes = 0;
21             ano = 0;
22         }
23     }
```

Exemplo de Classe Java (continuação)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
24      /**
25       * O metodo dataEhValida recebe tres valores como
        argumentos
26       * e verifica de maneira simples se os dados
27       * correspondem a uma data valida.
28       * Vale a pena notar que este algoritmo eh simples e
29       * incorreto.
30       * @param d o argumento correspondente ao campo dia
31       * @param m o argumento correspondente ao campo mes
32       * @param a o argumento correspondente ao campo ano
33       * @return true se a data for valida, false caso contrario
34       */
35      boolean dataEhValida(byte d, byte m, short a) {
36          if(d >= 1 && d <= 31 && m >= 1 && m <= 12)
37              return true;
38          else
39              return false;
40      }
```

Exemplo de Classe Java (continuação)


```
41      /**
42       * O metodo ehIgual recebe uma instancia da propria classe
43       * DataSimples como argumento e verifica se a data
44       * representada pela classe e pela intancia que foi
45       * passada eh a mesma.
46       * @param data uma instancia da propria classe DataSimples
47       * @return true se a data encapsulada for igual a passada,
         false caso contrario
48       */
49      boolean ehIgual(DataSimples data) {
50          if( (dia == data.dia) && (mes == data.mes) &&
51              (ano == data.ano) )
52              return true;
53          else
54              return false;
55      }
56
57      /**
58       * O metodo mostraData imprime a data na tela
59       */
60      void mostraData() {
61          System.out.println(dia + "/" + mes + "/" + ano);
62      }
63  }
```

Observações sobre a classe DataSimples

- Um ponto interessante: Observe que o método `ehIgual` recebe uma instância da própria classe `DataSimples`, para comparação de igualdade. Dentro deste método, os campos `dia`, `mês` e `ano` da classe são comparados, respectivamente, com os mesmos campos da instância que foi passada como argumento.

Atividade

Fazer um programa para ler as medidas dos lados de dois triângulos X e Y. Em seguida, mostrar o valor do perímetro e o valor da área dos dois triângulos. Além disso, dizer se os triângulos são equiláteros e dizer qual dos dois triângulos possui a maior área.

A fórmula para calcular a área de um triângulo a partir das medidas de seus lados a, b e c é a seguinte (fórmula de Heron):

$$area = \sqrt{p(p-a)(p-b)(p-c)} \text{ onde } p = \frac{a+b+c}{2}$$

Exemplo:

```
Enter the measures of triangle X: 3.00 4.00 5.00
Enter the measures of triangle Y: 7.50 4.50 4.02
Triangle X perimeter: 12.00
Triangle Y perimeter: 16.02
Triangle X area: 6.0000
Triangle Y area: 7.5638
Larger area: Y
X is not equilateral
Y is not equilateral
```

Métodos e passagem de parâmetros

Métodos e passagem de parâmetros

- **Passagem por valor (call by value):** o método obtém o valor do parâmetro que é passado para ele.
- **Passagem por referência (call by reference):** o método obtém o endereço de memória do argumento que é passado para ele.

Métodos e passagem de parâmetros

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Passagem por valor (call by value):** o método obtém o valor do parâmetro que é passado para ele.
- **Passagem por referência (call by reference):** o método obtém o endereço de memória do argumento que é passado para ele.

A linguagem Java **SEMPRE** usa passagem por valor.
Isso significa que o método obtém uma cópia dos valores dos parâmetros passados.

Métodos e passagem de parâmetros

- **Passagem por valor (call by value):** o método obtém o valor do parâmetro que é passado para ele.
- **Passagem por referência (call by reference):** o método obtém o endereço de memória do argumento que é passado para ele.

A linguagem Java **SEMPRE** usa passagem por valor.
Isso significa que o método obtém uma cópia dos valores dos parâmetros passados.

- Existem dois tipos de parâmetros para métodos:
  - **Parâmetros de tipos nativos:** os oito tipos nativos do Java.
  - **Parâmetros de tipos definidos por classes (referências a objetos):** variáveis que guardam o endereço de um objeto na memória.

Exemplo de uma aplicação Java

Passagem de parâmetro de tipo nativo

- Qual o valor impresso pelo programa abaixo?

```
1  public class Triplo {
2      public static void main(String[] args) {
3          double value = 10.0;
4          tripleValue( value );
5          System.out.println(value);
6      }
7
8      public static void tripleValue(double x) {
9          x = 3 * x;
10     }
11 }
```

Parâmetros de tipo referência a objeto

Uma prova de que Java não usa chamada por referência:

```
1  public class Swap {
2      public static void main(String[] args) {
3          Empregado a = new Empregado();
4          Empregado b = new Empregado();
5          a.nome = "Alice";
6          b.nome = "Bob";
7          swap(a, b);
8          System.out.printf("%s %s%n", a.nome, b.nome);
9      }
10
11     public static void swap(Empregado x, Empregado y) {
12         Empregado temp = x;
13         x = y;
14         y = temp;
15     }
16 }
17
18 class Empregado {
19     String nome;
20 }
```

Resumo

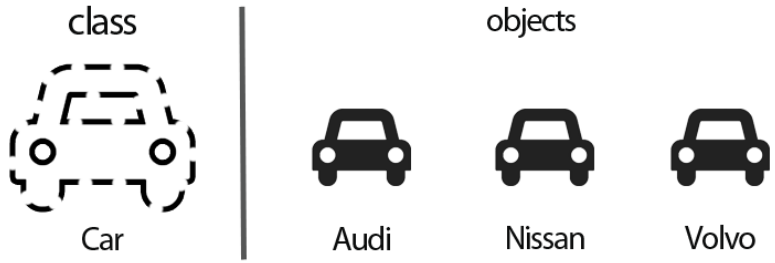
Resumo do que você pode e não pode fazer com os argumentos passados para os métodos em Java:

- Um método não pode modificar um argumento do tipo nativo.
- Um método **pode mudar o estado** de um argumento do tipo objeto.
- Um método não pode fazer um argumento objeto (que está fora do método) referenciar um outro objeto.

Objetos

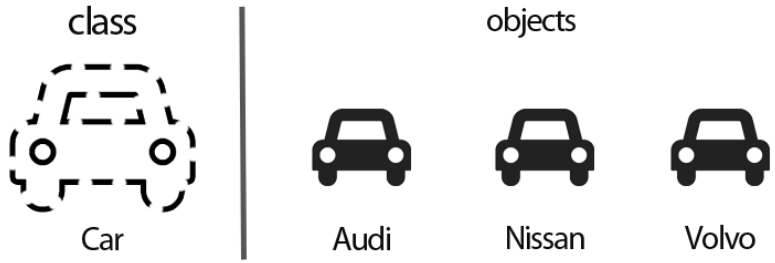
Objetos

- Um objeto ou instância é uma materialização da classe e, assim, pode ser usado para representar dados e executar operações.



Objetos

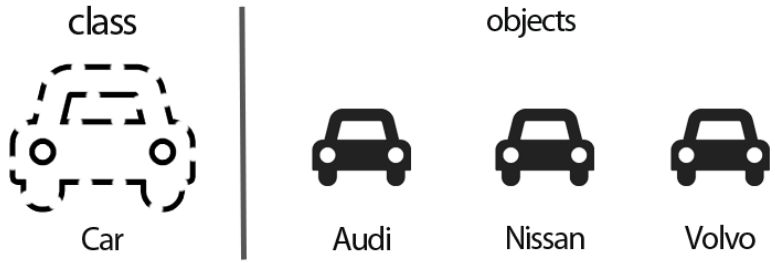
- Um objeto ou instância é uma materialização da classe e, assim, pode ser usado para representar dados e executar operações.



- Em Java, para que os objetos possam ser manipulados, é necessária a criação de referências a estes objetos, que são basicamente variáveis do tipo da classe.
  - Os objetos são armazenados na pilha (heap) do sistema e as referências apontam para os objetos.

Objetos

- Um objeto ou instância é uma materialização da classe e, assim, pode ser usado para representar dados e executar operações.



- Em Java, para que os objetos possam ser manipulados, é necessária a criação de referências a estes objetos, que são basicamente variáveis do tipo da classe.
  - Os objetos são armazenados na pilha (heap) do sistema e as referências apontam para os objetos.
- Para programadores em C++, referências podem ser consideradas como ponteiros, mas referências em Java não têm a complexidade de alocação em manipulação existente em outras linguagens.

Objetos são acessados por referência

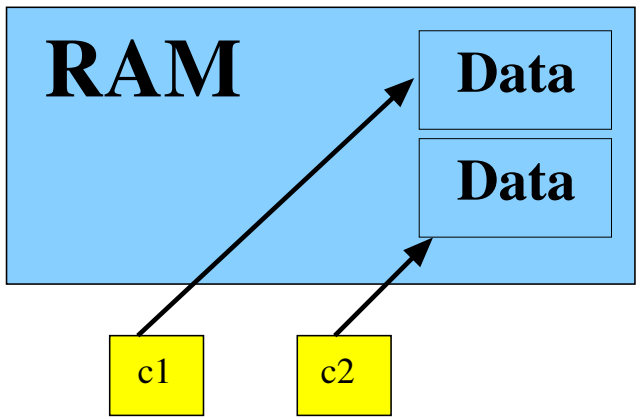
- Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma **referência** para ele.
  - Diferente dos tipos primitivos, como o `int` e `long`, precisamos chamar **new** depois de declarada a variável.

Objetos são acessados por referência

- Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma **referência** para ele.
  - Diferente dos tipos primitivos, como o `int` e `long`, precisamos chamar **new** depois de declarada a variável.

```
public static void main(String[] args) {
    Data c1;
    c1 = new Data();

    Data c2;
    c2 = new Data();
}
```

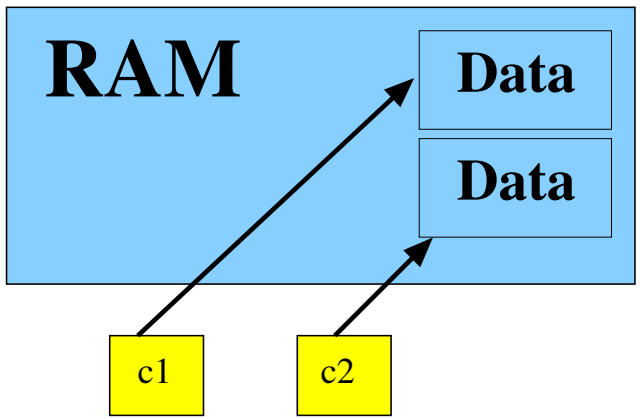


Objetos são acessados por referência

- Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma **referência** para ele.
  - Diferente dos tipos primitivos, como o `int` e `long`, precisamos chamar **new** depois de declarada a variável.

```
public static void main(String[] args) {
    Data c1;
    c1 = new Data();

    Data c2;
    c2 = new Data();
}
```



- Internamente, c1 e c2 guardam o endereço de memória (**referência**) em que o respectivo objeto do tipo Data se encontra.
  - Uma referência parece um ponteiro; porém, não podemos manipular uma referência como se fosse um número e nem utilizá-la para aritmética pois ela é tipada.

Objetos são acessados por referência

- O que é impresso na tela ao executar o programa abaixo?

```
1  public class AppTestaRef {
2      public static void main(String[] args) {
3          DataSimples c1 = new DataSimples();
4          byte dia = 1, mes = 1;
5          short ano = 2021;
6          c1.inicializaDataSimples(dia, mes, ano);
7
8          DataSimples c2 = c1;
9          c2.mes = 7;
10
11         c1.mostraData();
12         c2.mostraData();
13
14         if(c1 == c2)
15             System.out.println("Datas iguais");
16         else
17             System.out.println("Datas diferentes");
18     }
19 }
```

Objetos são acessados por referência

- O que é impresso na tela ao executar o programa abaixo?

```
1  public class AppTestaRef2 {
2      public static void main(String[] args) {
3          DataSimples c1 = new DataSimples();
4          byte dia = 1, mes = 1;
5          short ano = 2021;
6          c1.inicializaDataSimples(dia, mes, ano);
7
8          DataSimples c2 = new DataSimples();
9          dia = 1; mes = 1; ano = 2021;
10         c2.inicializaDataSimples(dia, mes, ano);
11
12         if(c1 == c2)
13             System.out.println("Datas iguais");
14         else
15             System.out.println("Datas diferentes");
16     }
17 }
```

Objetos

Comportamento

- Um objeto possui três características principais: **comportamento, estado e identidade**.

Objetos

Comportamento

- Um objeto possui três características principais: **comportamento, estado e identidade**.
- **Comportamento do objeto**: o que você pode fazer com esse objeto ou quais métodos você pode aplicar a ele. O comportamento de um objeto é definido pelos seus métodos.

Objetos

Comportamento

- Um objeto possui três características principais: **comportamento, estado e identidade**.
- **Comportamento do objeto**: o que você pode fazer com esse objeto ou quais métodos você pode aplicar a ele. O comportamento de um objeto é definido pelos seus métodos.

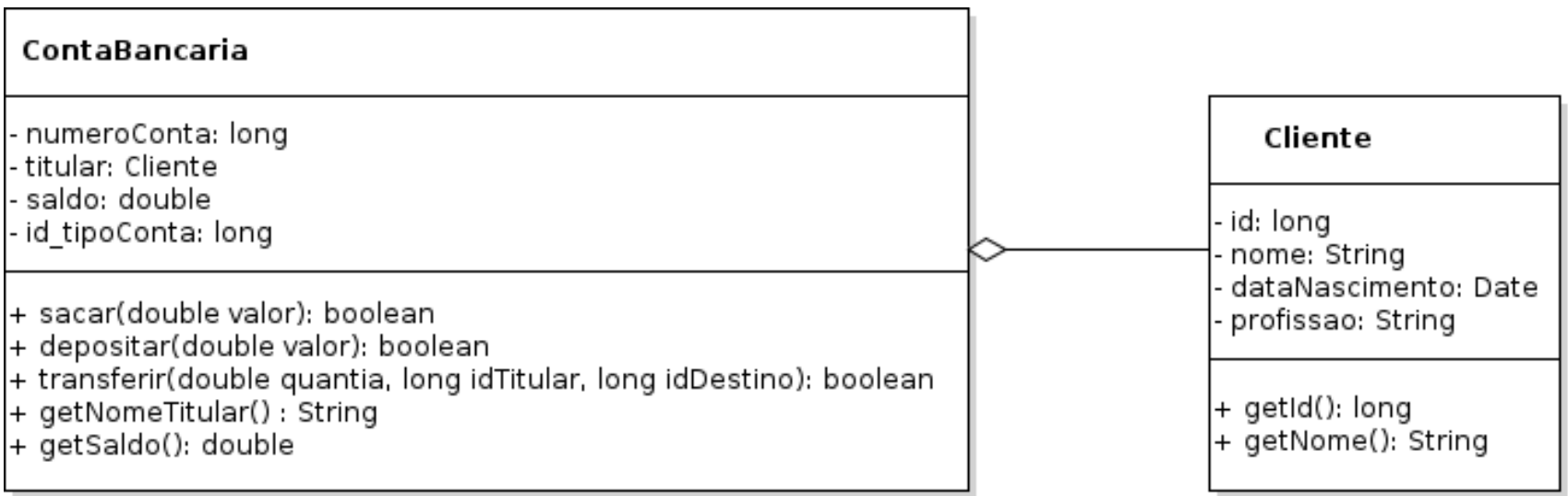
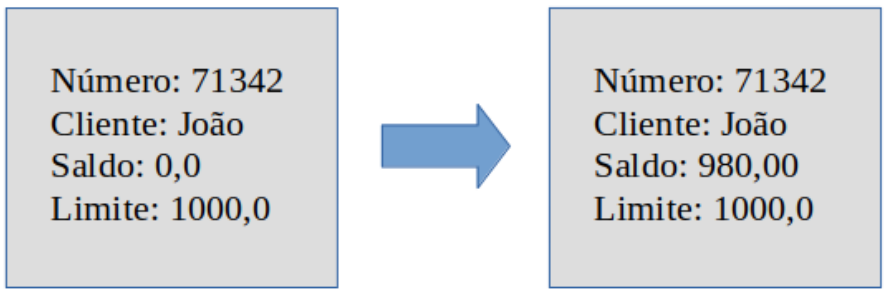


Diagrama de classes com duas classes

Objetos

Estado

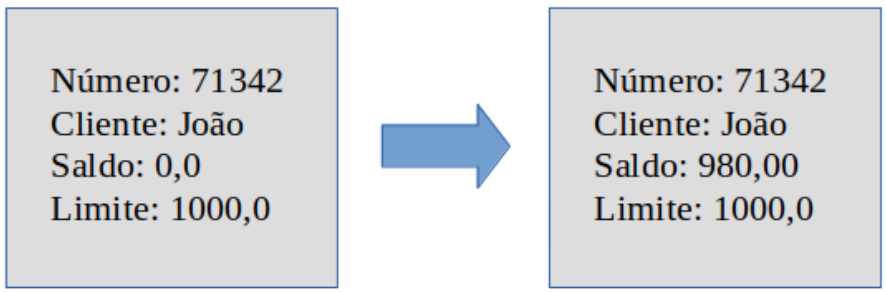
- **Estado do objeto**: conjunto dos valores armazenados em seus atributos.



Objetos

Estado

- **Estado do objeto**: conjunto dos valores armazenados em seus atributos.



- O estado de um objeto pode mudar com o tempo, mas não de forma espontânea. Uma mudança no estado de um objeto deve ser uma consequência de chamadas de método.
- Se o estado do objeto foi alterado sem uma chamada de método nesse objeto, então o encapsulamento não foi bem programado.

Objetos

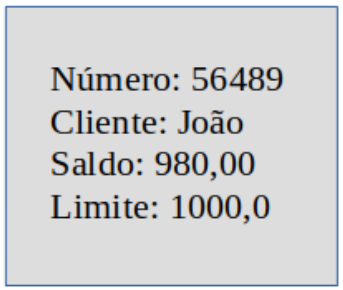
Identidade

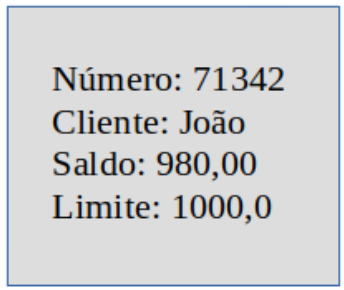
- O estado de um objeto não o descreve completamente, porque cada objeto possui uma **identidade** distinta.

Objetos

Identidade

- O estado de um objeto não o descreve completamente, porque cada objeto possui uma **identidade** distinta.
- Por exemplo, em um sistema de processamento de pedidos, dois pedidos são distintos mesmo que solicitem itens idênticos.

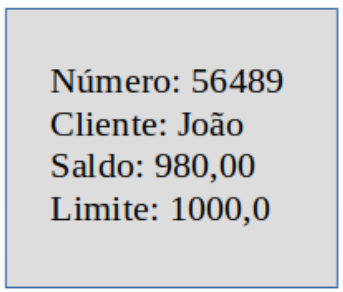


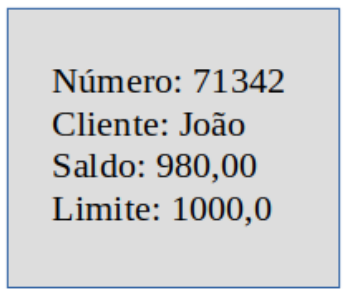


Objetos

Identidade

- O estado de um objeto não o descreve completamente, porque cada objeto possui uma **identidade** distinta.
- Por exemplo, em um sistema de processamento de pedidos, dois pedidos são distintos mesmo que solicitem itens idênticos.





- Os objetos individuais que são instâncias de uma classe sempre diferem em sua identidade e geralmente diferem também em seu estado.

Criando e usando um objeto

- Para criar um objeto do tipo DataSimples, precisamos criar uma aplicação (uma classe com a função main) que instancie DataSimples e invoque os seus métodos.

- Mostrar o projeto DataSimples

- Quais problemas o programa AppDataSimples.java tem? Quais boas práticas de programação orientada a objetos não são respeitadas?

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM