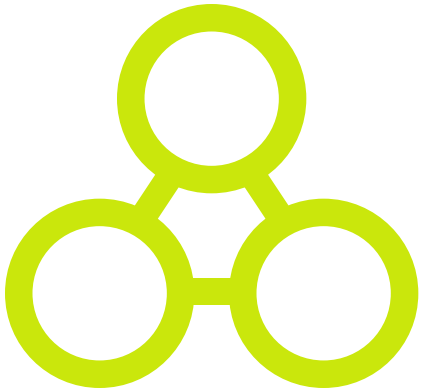


nemo

ontology & conceptual modeling research group

# Desenvolvimento OO com Java
# Orientação a objetos básica


Vítor E. Silva Souza

(vitor.souza@ufes.br)

http://www.inf.ufes.br/~vitorsouza



Departamento de Informática

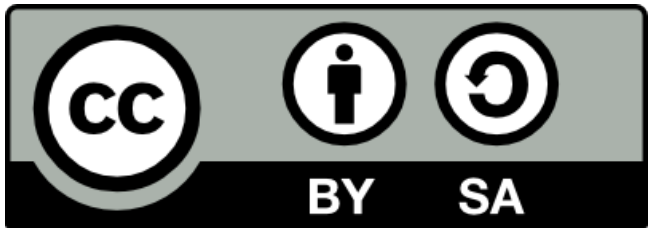
Centro Tecnológico

Universidade Federal do Espírito Santo

# Licença para uso e distribuição

- Este obra está licenciada com uma licença Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional;
- Você tem o direito de:
  - Compartilhar: copiar e redistribuir o material em qualquer suporte ou formato
  - Adaptar: remixar, transformar, e criar a partir do material para qualquer fim, mesmo que comercial.
- De acordo com os termos seguintes:
  - Atribuição: você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças foram feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de maneira alguma que sugira ao licenciante a apoiar você ou o seu uso;
  - CompartilhaIgual: se você remixar, transformar, ou criar a partir do material, tem de distribuir as suas contribuições sob a mesma licença que o original.


Mais informações podem ser encontradas em:
http://creativecommons.org/licenses/by-sa/4.0/



# Conteúdo do curso

- O que é Java;
- Variáveis primitivas e controle de fluxo;
- Orientação a objetos básica;
- Um pouco de vetores;
- Modificadores de acesso e atributos de classe;
- Herança, reescrita e polimorfismo;
- Classes abstratas;
- Interfaces;
- Exceções e controle de erros;
- Utilitários da API Java.

Estes slides foram baseados na apostila do curso FJ-11: Java e Orientação a Objetos da Caelum e na apostila Programação Orientada a Objetos em Java do prof. Flávio Miguel Varejão.

# Questão de responsabilidade

- Imagine um sistema...
  - Dezenas de formulários;
  - CPFs são validados – função `validar(cpf)` deve ser chamada em cada formulário;
  - Todos os desenvolvedores são responsáveis!
  - Define-se que idade deve ser >= 18. Validação simples (um *if*), mas onde adicioná-la?
- Procedural: responsabilidade espalhada;
- OO: responsabilidade concentrada, polimorfismo.

# Classes e objetos

- Desde Aristóteles que o ser humano classifica os objetos do mundo;
- Juntamos objetos com mesmas características em categorias que chamamos de “classes”:
  - Todas as contas de banco tem um saldo, mas cada conta pode ter um saldo diferente;
  - Todas as contas de banco podem sofrer depósitos ou serem encerradas.
- Classes são usadas por linguagens OO para modelar tipos compostos. São modelos abstratos que definem os objetos da classe.

# Classes e objetos

| Classes são projetos / especificações | Objetos são instâncias de verdade |
|---|---|
| Homo Sapiens | Um ser humano |
| Receita de bolo | Um bolo feito com a receita |
| Planta de uma casa | Uma casa construída a partir da planta |

Definem um conjunto de características e comportamentos comuns.

Possuem valores para as características (olhos verdes, calda de chocolate, cor azul) e podem realizar o comportamento (correr, ser comido, abrir a porta)

# Definição de uma classe

- Uso da palavra reservada `class`;
- Significado: “segue abaixo a especificação de como objetos deste tipo devem se comportar”;

```
class NomeDaClasse {
  /* Especificação da classe vai aqui. */
}
```

- Depois de definida a classe, podemos definir variáveis (referências) e criar objetos:

```
NomeDaClasse obj = new NomeDaClasse();
```

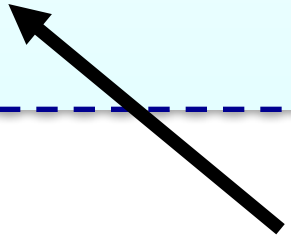
# Membros da classe

- Uma classe pode ter dois tipos de membro:
  - Variáveis (em jargão OO: “atributos”);
  - Funções (em jargão OO: “métodos”).
- Atributos são como partes de um tipo composto;
- Métodos são funções que são executadas no contexto de uma classe/objeto.

# Atributos

- Definidos como variáveis no escopo da classe:

```
class Conta {
  int numero;
  String dono;
  double saldo;
  double limite;
  // ...
}
```



Repare que as variáveis não são declaradas dentro de um bloco de função, mas diretamente no bloco da classe.

# Atributos

- Acesso via operador de seleção (“.”):

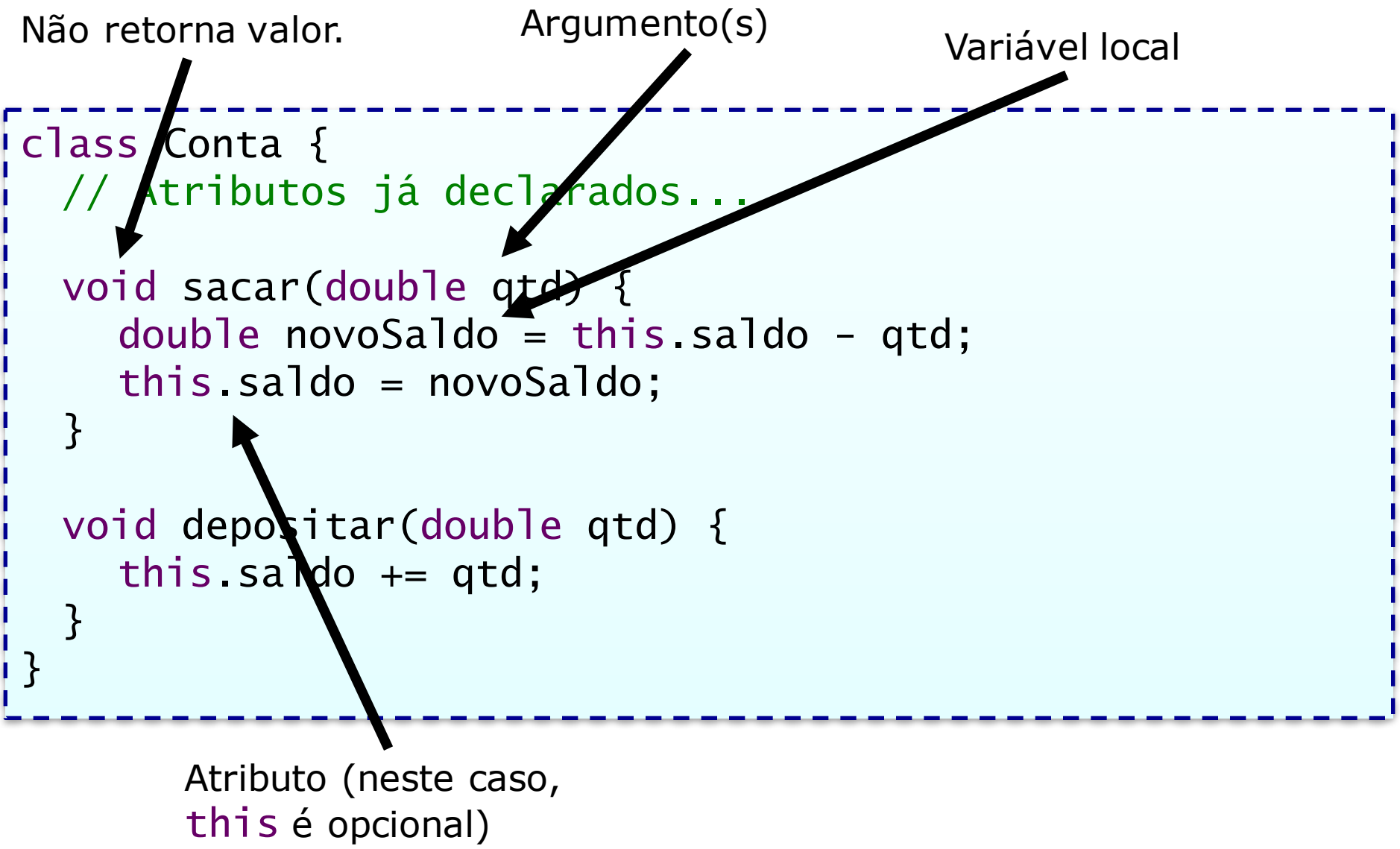
```
class Programa {
  public static void main(String[] args) {
     Conta minhaConta;
     minhaConta = new Conta();

     minhaConta.dono = "Duke";
     minhaConta.saldo = 1000.0;

     System.out.println("Saldo: " + minhaConta.saldo);
  }
}
```

# Métodos

- Um método é uma função que opera no contexto de uma classe (mensagem que o objeto recebe);
- É a maneira (método) de se fazer algo num objeto:

```
class Conta {
  // Atributos já declarados...

  void sacar(double qtd) {
     double novoSaldo = this.saldo - qtd;
     this.saldo = novoSaldo;
  }

  void depositar(double qtd) {
     this.saldo += qtd;
  }
}
```

# Métodos


Não retorna valor.
Argumento(s)
Variável local
class Conta {
// Atributos já declarados...
void sacar(double qtd) {
double novoSaldo = this.saldo - qtd;
this.saldo = novoSaldo;
}
void depositar(double qtd) {
this.saldo += qtd;
}
}
Atributo (neste caso,
this é opcional)

# Métodos

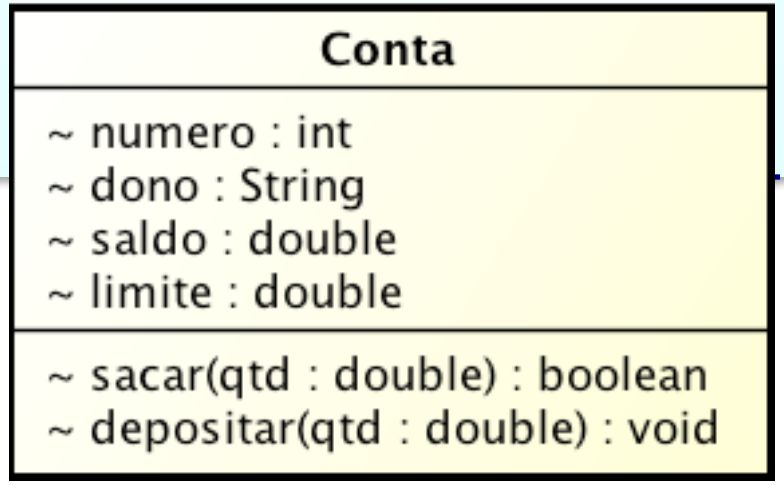
- E/Invocação também via operador de seleção (“.”):

```
class Programa {
  public static void main(String[] args) {
     Conta minhaConta = new Conta();
     minhaConta.dono = "Duke";
     minhaConta.saldo = 1000;

     minhaConta.sacar(200);
     minhaConta.depositar(500);

     // Saldo: 1300.0
     System.out.println("Saldo: " + minhaConta.saldo);
  }
}
```

# Métodos

- Um método pode retornar um valor:

```
class Conta {
  // Atributos já declarados...

  boolean sacar(double qtd) {
     if (saldo < qtd) return false;

     saldo = saldo - qtd;
     return true;
  }
}
```



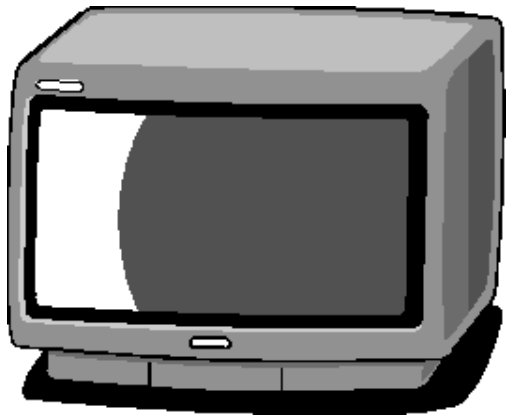
# Uma classe, múltiplas instâncias

- Podemos criar quantos objetos quisermos...

```
class Programa {
  public static void main(String[] args) {
     Conta minhaConta = new Conta();
     minhaConta.saldo = 1000;
     if (minhaConta.sacar(2000))
        System.out.println("Consegui");
     else System.out.println("Não deu...");

     Conta meuSonho = new Conta();
     meuSonho.saldo = 1_000_000_000.0;

     // Usando a mesma referência.
     minhaConta = new Conta();
  }
}
```

# Manipulação de objetos em Java

- Em Java trabalhamos com referências para objetos, ao contrário de C++ (manipulação direta ou ponteiros);
- Analogia:
  - A TV é o objeto;
  - O controle é a referência;
  - Você só carrega a referência;
  - A referência pode existir sem o objeto.



# Referência e objeto

```
public class Coordenadas {
  int x;
  int y;
  int z;

  public static void main(String[] args) {
    Coordenadas coord;  // Só a referência.
                        // Não dá pra fazer nada...

    // Agora temos um objeto, podemos usá-lo.
    coord = new Coordenadas();
    coord.x = 10;
    coord.y = 15;
    coord.z = 18;
  }
}
```

# Atribuição de valores

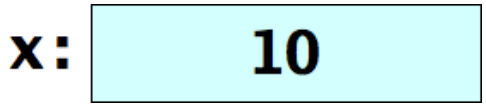
- Quando realizamos uma atribuição:

```
x = y;
```

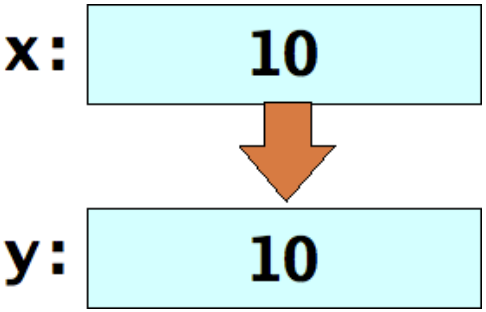
- Java faz a cópia do valor da variável da direita para a variável da esquerda;
  - Para tipos primitivos, isso significa que alterações em x não implicam alterações em y;
  - Para objetos, como o que é copiado é a referência para o mesmo objeto, alterações no objeto que x referencia altera o objeto que y referencia, pois é o mesmo objeto!

# Atribuição de valores primitivos

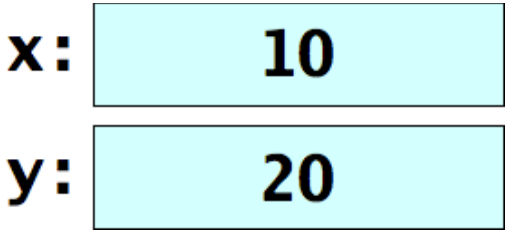
```
int x = 10;
```




```
int y = x;
```

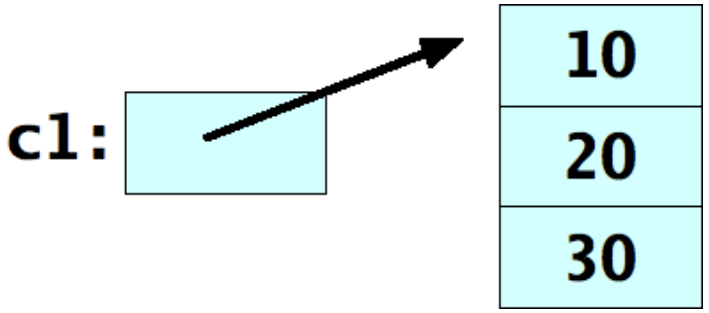



```
y = 20;
```

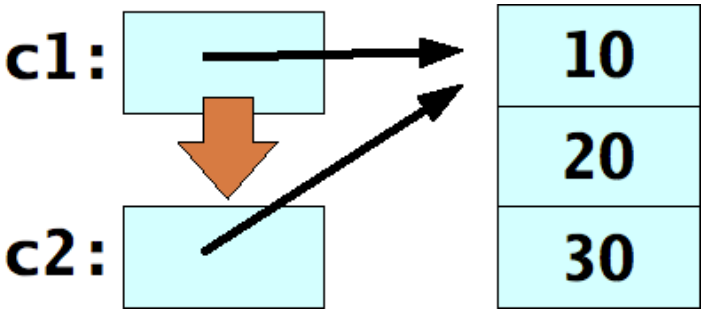


# Atribuição de objetos

```
Coordenada c1;
c1 = new Coordenada();
c1.x = 10;
c1.y = 20;
c1.z = 30;
```

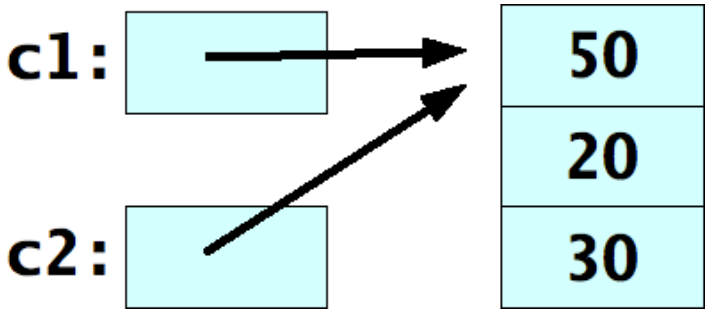



```
Coordenada c2;

// Erro comum:
// c2 = new Coordenada();

c2 = c1;
```



# Atribuição de objetos

```
c2.x = 50;
```




Tenha sempre em mente a diferença entre um tipo primitivo e um objeto (referência).

"É parecido com um ponteiro, porém você não pode manipulá-lo como um número e nem utilizá-lo para aritmética, ela é tipada." (Caelum FJ-11)

# Comparações entre objetos

```
public class Comparacoes {
  public static void main(String[] args) {
    Coordenadas c1 = new Coordenadas();
    c1.x = 10; c1.y = 15; c1.z = 20;

    Coordenadas c2 = new Coordenadas();
    c2.x = 10; c2.y = 15; c2.z = 20;

    // O que imprime?
    System.out.println(c1 == c2);
  }
}
```

false

# Comparações entre objetos

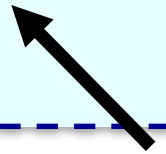
```
public class Comparacoes {
  public static void main(String[] args) {
    Coordenadas c1 = new Coordenadas();
    c1.x = 10; c1.y = 15; c1.z = 20;

    Coordenadas c2 = c1;
    c2.x = 11; c2.y = 16; c2.z = 21;

    // O que imprime?
    System.out.println(c1 == c2);
  }
}
```

true

# Métodos == funções de classe

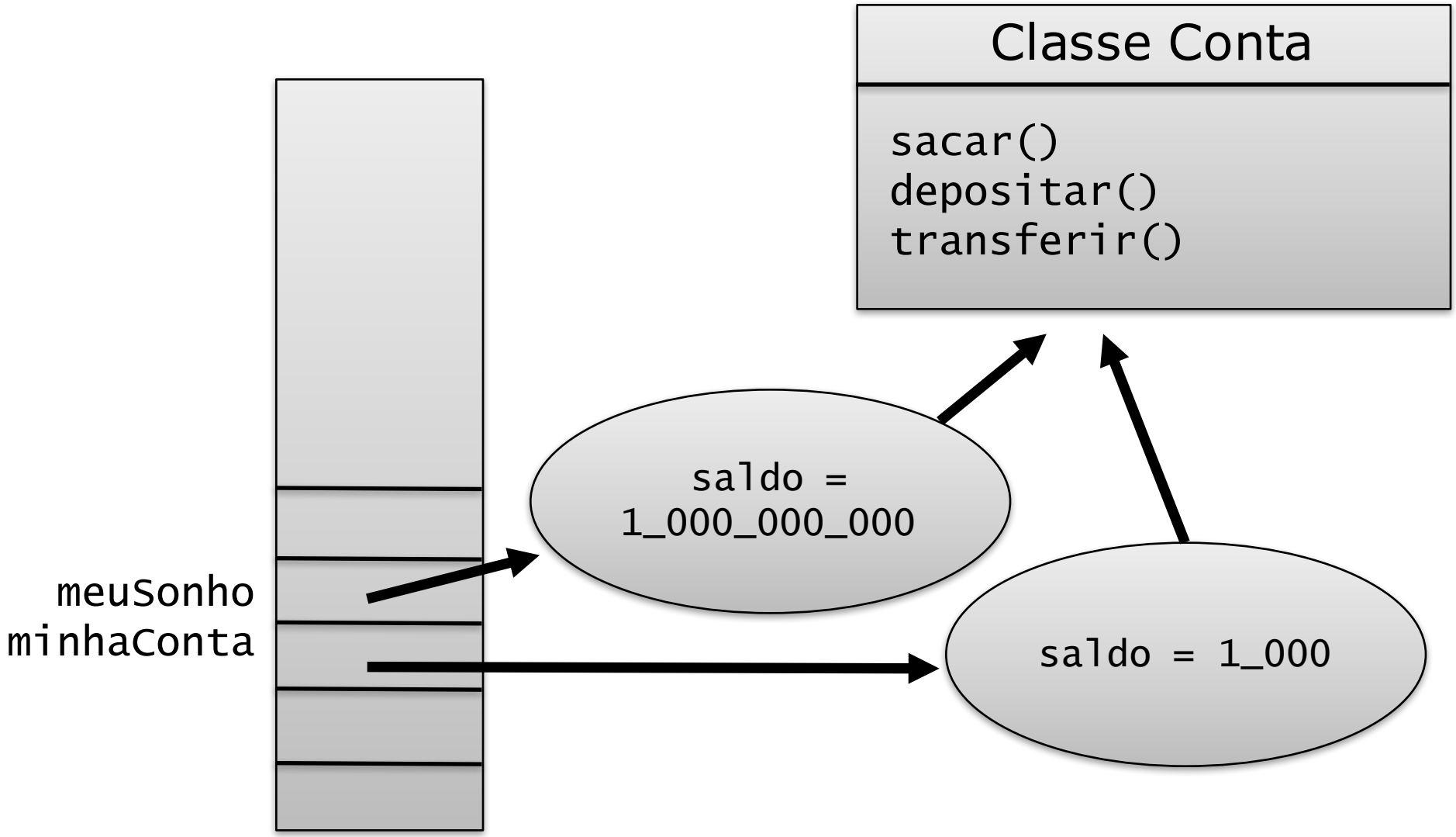
- Métodos são funções que executam no contexto de uma classe:

```
class Conta { //...
  boolean transferir(Conta destino, double qtd) {
     if (! this.sacar(qtd)) return false;
     destino.depositar(qtd);
     return true;
  }

  public static void main(String[] args) {
     // Copie aqui minhaConta e meuSonho do slide 15.

     meuSonho.transferir(minhaConta, 1_000_000);
  }
}
```



O destino é minhaConta,
mas e a origem?

# Armazenamento em memória



# Chamando métodos em objetos

- O código compilado dos métodos fica na área de memória da classe;
- Sendo assim, como Java sabe em qual objeto estou chamando um determinado método?

```
class Num {
  int i = 5;
  void somar(int j) { i += j; }
}
public class Teste
  public static void main(String[] args) {
     Num m = new Num(), n = new Num();
     m.somar(10); n.somar(5);
  }
}
```

# Chamando métodos em objetos

- Internamente é como se o método fosse:

```
// Imagina uma "função global", como em C:
void somar(Num this, int j) {
  this.i += j;
}
```

- E a chamada fosse:

```
// Como chamaríamos a função em programação estruturada.
somar(m, 10);
somar(n, 5);
```

# A palavra reservada this

- Java faz esta transformação para você, de forma que o objeto que "recebeu a mensagem" está disponível pela palavra-chave this:

```
class Num {
  int i = 5;
  void somar(int j) { this.i += j; }
}
```

- Não é necessário usar this quando acessamos membros do objeto de dentro do mesmo (como no exemplo acima).

# A palavra reservada this

- this pode ser usado para diferenciar um atributo do objeto de um parâmetro do método:

```
class Num {
  int i = 5;
  void somar(int i) { this.i += i; }
}
```

- Neste caso, o this é necessário!

# A palavra reservada this

```
class Num {
  int i = 5;
  Num somar(int j) {
     i += j;
     return this;  // Aqui, this é útil!
  }
}
public class Teste
  public static void main(String[] args) {
     Num m = new Num();
     m.somar(10).somar(5).somar(1);
     System.out.println(m.i); // 21
  }
}
```

# Valores *default* para atributos

- Um atributo pode ser inicializado:

```
class Conta {
  int numero;    // 0
  String dono;   // null
  double saldo; // 0.0
  double limite = 1000.0;
}
```

- Quando não inicializamos explicitamente, um valor *default* é atribuído a ele:

| Tipo | Valor |
|---|---|
| boolean | false |
| char | '\u0000' |
| byte | (byte) 0 |
| short | (short) 0 |

| Tipo | Valor |
|---|---|
| int | 0 |
| long | 0l |
| float | 0.0f |
| double | 0.0 |

# Implementando associações entre classes

- Atributos podem ser referências para objetos de outras classes (ou da mesma classe):

```
class Conta {
  int numero;
  double saldo;
  double limite = 1000.0;
  Cliente titular;
}

class Cliente {
  String nome;
  String sobrenome;
  String cpf;
}

// ...
```

# Implementando associações entre classes

- Atributos podem ser referências para objetos de outras classes (ou da mesma classe):

```
public class Teste
  public static void main(String[] args) {
     Cliente larry = new Cliente();
     larry.nome = "Larry";
     larry.sobrenome = "Ellison";

     Conta conta = new Conta();
     conta.saldo = 50_400_000_000.0;
     conta.titular = larry;

     // Navegando no grafo de objetos...
     System.out.println(conta.titular.nome);
  }
}
```

# O valor null

- O valor *default* para referências (objetos) é `null`;
- Um "objeto nulo" é uma referência que não aponta para nenhum objeto;
- Usar uma referência nula como se ela apontasse para um objeto causa `NullPointerException`.

```
public class Teste
  public static void main(String[] args) {
     Conta conta = new Conta();
     conta.saldo = 50_400_000_000.0;
     System.out.println(conta.titular.nome);
  }
}

// Exception in thread "main"
// java.lang.NullPointerException
```

# Valores *default* para variáveis locais

- Variáveis locais não são "zeradas" automaticamente e geram erros de compilação se utilizadas sem valor:

```
public class Teste
  public static void main(String[] args) {
     Cliente larry;
     Conta conta = new Conta();
     conta.saldo = 50_400_000_000.0;
     conta.titular = larry;
     System.out.println(conta.titular.nome);
  }
}

// error: variable larry might not have been initialized
//     conta.titular = larry;
//                     ^
```

# Exercitar é fundamental

- Apostila FJ-11 da Caelum:
  - Seção 4.12, página 51 (class Funcionario);
  - Seção 4.13, página 55 (recursividade / Fibonacci);
  - Seção 4.14, página 56 (fixando o conhecimento).


http://nemo.inf.ufes.br/