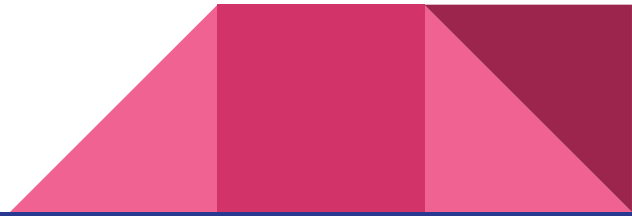
Linguagens de Programação

Java Orientado a Objetos

Prof. Waldeck Lindoso Jr.

Declarando métodos

- Dentro da classe, também declararemos o que cada conta faz e como isto é feito - os comportamentos que cada classe tem, isto é, o que ela faz. Por exemplo, de que maneira que uma Conta saca dinheiro? Especificaremos isso dentro da própria classe Conta, e não em um local desatrelado das informações da própria Conta. É por isso que essas "funções" são chamadas de métodos. Pois é a maneira de fazer uma operação com um objeto.



Declarando métodos

- Queremos criar um método que saca uma determinada quantidade e não devolve nenhuma informação para quem acionar esse método:

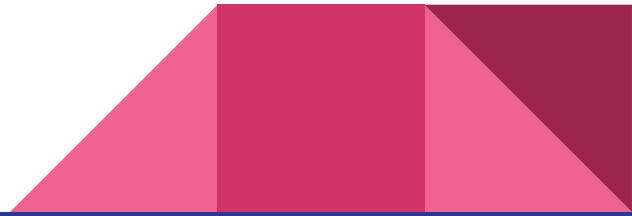
```
class Conta {
    double salario;
    // ... outros atributos ...

    void saca(double quantidade) {
        double novoSaldo = this.saldo - quantidade;
        this.saldo = novoSaldo;
    }
}
```

Declarando métodos

- A palavra chave void diz que, quando você pedir para a conta sacar uma quantia, nenhuma informação será enviada de volta a quem pediu.
- Quando alguém pedir para sacar, ele também vai dizer quanto quer sacar. Por isso precisamos declarar o método com algo dentro dos parênteses -o que vai aí dentro é chamado de argumento do método(ou parâmetro). Essa variável é uma variável comum, chamada também de temporária ou local, pois, ao final da execução desse método, ela deixa de existir.

Declarando métodos

- Dentro do método,estamos declarando uma nova variável. Essa variável, assim como o argumento, vai morrer no fim do método, pois este é seu escopo. No momento que vamos acessar nosso atributo, usamos a palavra chave this para mostrar que esse é um atributo, e não uma simples variável. (veremos depois que é opcional).
- Repare que, nesse caso, a conta poderia estourar um limite fixado pelo banco. Mais para frente, evitaremos essa situação, e de uma maneira muito elegante.



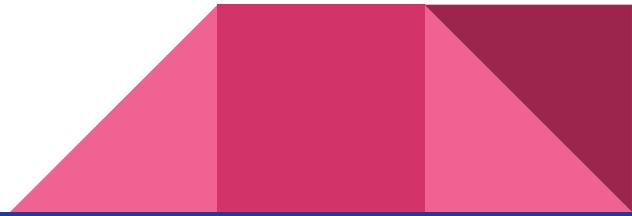
Declarando métodos

- Da mesma forma, temos o método para depositar alguma quantia:

```
class Conta {
    // ... outros atributos e métodos ...

    void deposita(double quantidade) {
        this.saldo += quantidade;
    }
}
```

Declarando métodos

- Observe que não usamos uma variável auxiliar e, além disso, usamos a abreviação **+=** para deixar o método bem simples. O **+=** soma quantidade ao valor antigo do saldo e guarda no próprio saldo, o valor resultante.
- Para mandar uma mensagem ao objeto e pedir que ele execute um método, também usamos o ponto. O termo usado para isso é invocação de método.



Declarando métodos

- O código a seguir saca dinheiro e depois deposita outra quantia na nossa conta:

```
class TestaAlgunsMetodos {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();

        // alterando os valores de minhaConta
        minhaConta.titular = "Duke";
        minhaConta.saldo = 1000;

        // saca 200 reais
        minhaConta.saca(200);

        // deposita 500 reais
        minhaConta.deposita(500);
        System.out.println(minhaConta.saldo);
    }
}
```

Declarando métodos

- Uma vez que seu saldo inicial é 1000 reais, se sacarmos 200 reais, depositarmos 500 reais e imprimirmos o valor do saldo, o que será impresso?

```
class TestaAlgunsMetodos {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();

        // alterando os valores de minhaConta
        minhaConta.titular = "Duke";
        minhaConta.saldo = 1000;

        // saca 200 reais
        minhaConta.saca(200);

        // deposita 500 reais
        minhaConta.deposita(500);
        System.out.println(minhaConta.saldo);
    }
}
```

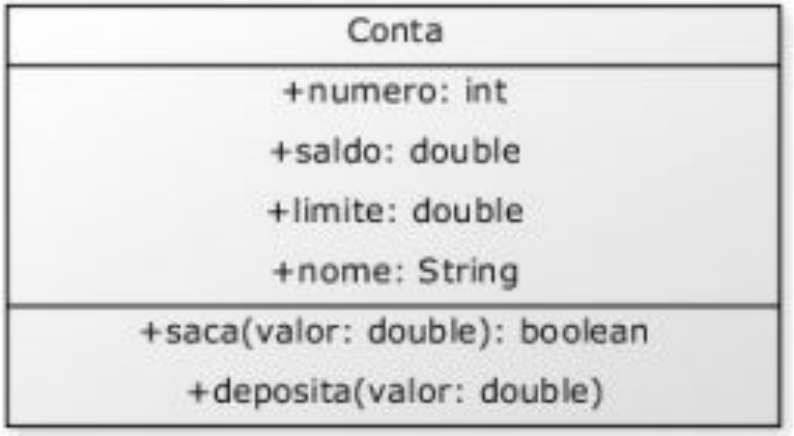
Métodos com retorno

- Um método sempre tem que definir o que retorna, nem que defina que não há retorno, como nos exemplos anteriores onde estávamos usando o ***void.***
- Um método pode retornar um valor para o código que o chamou. No caso do nosso método saca, podemos devolver um valor booleano indicando se a operação foi bem sucedida.

```
class Conta {
    // ... outros métodos e atributos ...

    boolean saca(double valor) {
        if (this.saldo < valor) {
            return false;
        }
        else {
            this.saldo = this.saldo - valor;
            return true;
        }
    }
}
```

Métodos com retorno

- A declaração do método mudou! O método saca não tem void na frente. Isto quer dizer que, quando é acessado, ele devolve algum tipo de informação. No caso, um boolean . A palavra chave return indica que o método vai terminar ali, retornando tal informação.



Métodos com retorno

- Exemplo de uso:

```
minhaConta.saldo = 1000;
boolean consegui = minhaConta.saca(2000);
if (consegui) {
    System.out.println("Consegui sacar");
} else {
    System.out.println("Não consegui sacar");
}
```

Métodos com retorno

- Ou então, posso eliminar a variável temporária, se desejado:

```
minhaConta.saldo = 1000;
if (minhaConta.saca(2000)) {
    System.out.println("Consegui sacar");
} else {
    System.out.println("Não consegui sacar");
}
```

Métodos com retorno

- Meu programa pode manter na memória não apenas uma conta, como mais de uma:

```
class TestaDuasContas {
    public static void main(String[] args) {

        Conta minhaConta;
        minhaConta = new Conta();
        minhaConta.saldo = 1000;

        Conta meuSonho;
        meuSonho = new Conta();
        meuSonho.saldo = 1500000;
    }
}
```

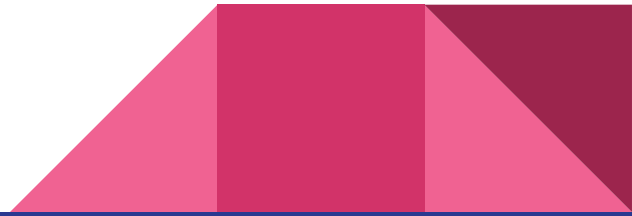
Acesso a objetos por referência

- Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma maneira de acessá-lo, chamada de referência.
- É por esse motivo que, diferente dos tipos primitivos **int** e **long**, precisamos dar **new** depois de declarada a variável:

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

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

Acesso a objetos por referência

- O correto aqui, é dizer que c1 se refere a um objeto. **Não é correto** dizer que c1 é um objeto, pois c1 é uma variável referência, apesar de, depois de um tempo, os programadores Java falarem "Tenho um **objeto c** do tipo **Conta**", mas apenas para encurtar a frase "Tenho uma **referência c** a um objeto do tipo **Conta**".



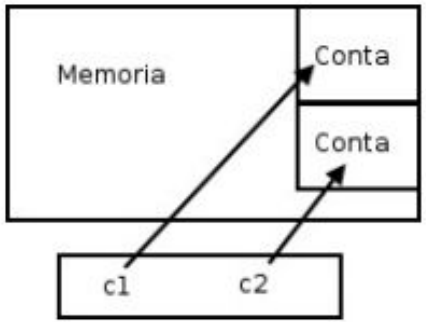
Acesso a objetos por referência

- Basta lembrar que, em Java, uma variável nunca é um objeto. Não há, no Java, uma maneira de criarmos o que é conhecido como "objeto pilha" ou "objeto local", pois todo objeto em Java, sem exceção, é acessado por uma variável referência.
- Esse código nos deixa na seguinte situação:

```
Conta c1;
c1 = new Conta();

Conta c2;
c2 = new Conta();
```

Acesso a objetos por referência

- Internamente, c1 e c2 vão guardar um número que identifica em que posição da memória aquela Conta se encontra. Dessa maneira, ao utilizarmos o ‘.’ para navegar, o Java vai acessar a Conta que se encontra naquela posição de memória, e não uma outra.
- Para quem conhece, é parecido com um ponteiro, porém você não pode manipulá-lo como um número e nem utilizá-lo para aritmética, ela é tipada.



Acesso a objetos por referência

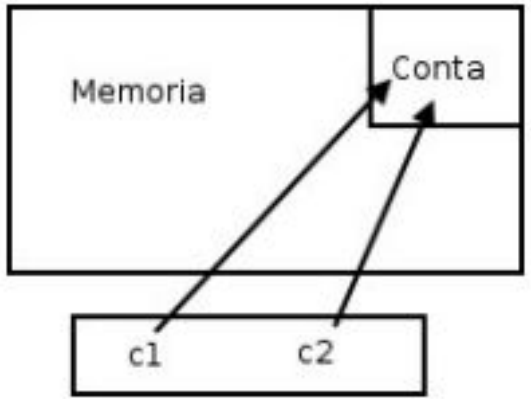
- Um outro exemplo:
  - Qual seria o resultado do código abaixo? o que aparece ao rodar?

```
class TestaReferencias {
    public static void main(String[] args) {
        Conta c1 = new Conta();
        c1.deposita(100);

        Conta c2 = c1;  // linha importante!
        c2.deposita(200);

        System.out.println(c1.saldo);
         System.out.println(c2.saldo);
    }
}
```

Acesso a objetos por referência

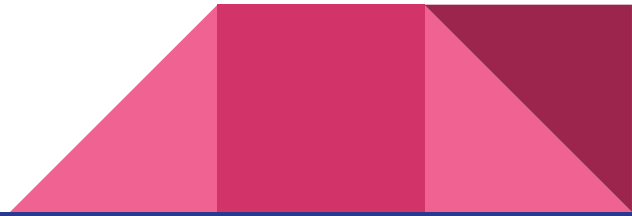
- O que acontece aqui? O operador = copia o valor de uma variável. Mas qual é o valor da variável c1 ? É o objeto? Não. Na verdade, o valor guardado é a referência (endereço) de onde o objeto se encontra na memória principal.
  - Na memória, o que acontece nesse caso:

```
Conta c1 = new Conta();
Conta c2 = c1;
```



Acesso a objetos por referência

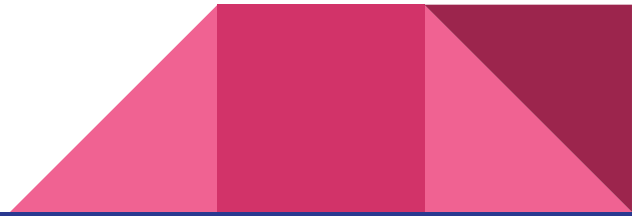
- Então, nesse código em específico, quando utilizamos c1 ou c2 estamos nos referindo exatamente ao mesmo objeto! Elas são duas referências distintas, porém apontam para o mesmo objeto! Compará-las com "**==**" vai nos retornar **true** , pois o valor que elas carregam é o mesmo! Outra forma de perceber, é que demos apenas um **new**, então só pode haver um objeto Conta na memória.



Acesso a objetos por referência

NEW

- O que exatamente faz o new ?
- O new executa uma série de tarefas, que veremos mais adiante.
- Mas, para melhor entender as referências no Java, saiba que o new, depois de alocar a memória para esse objeto, devolve uma "flecha", isto é, um valor de referência. Quando você atribui isso a uma variável, essa variável passa a se referir para esse mesmo objeto.



Acesso a objetos por referência

- Podemos então ver outra situação:

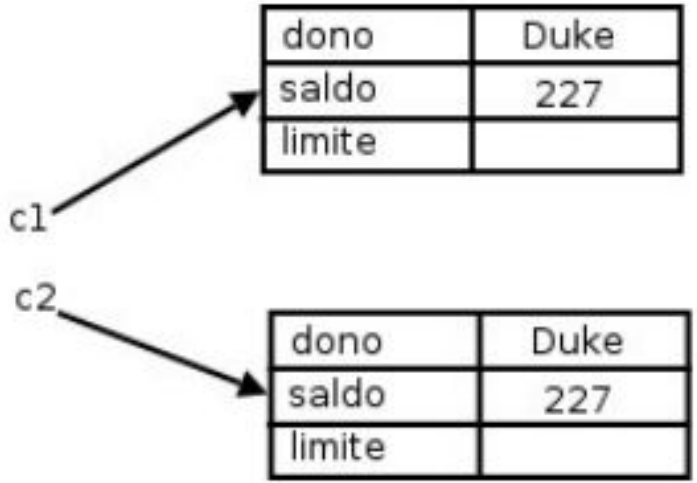
```
public static void main(String[] args) {
    Conta c1 = new Conta();
    c1.titular = "Duke";
    c1.saldo = 227;

    Conta c2 = new Conta();
    c2.titular = "Duke";
    c2.saldo = 227;

    if (c1 == c2) {
        System.out.println("Contas iguais");
    }
}
```

Acesso a objetos por referência

- O operador “**==**“ compara o conteúdo das variáveis, mas essas variáveis não guardam o objeto, e sim o endereço em que ele se encontra. Como em cada uma dessas variáveis guardamos duas contas criadas diferentemente, elas estão em espaços diferentes da memória, o que faz o teste no **if** valer **false**. As contas podem ser equivalentes no nosso critério de igualdade, porém elas não são o mesmo objeto. Quando se trata de objetos, pode ficar mais fácil pensar que o “**==**” compara se os objetos (referências, na verdade) são o mesmo, e não se são iguais.

Acesso a objetos por referência

- Para saber se dois objetos têm o mesmo conteúdo, você precisa comparar atributo por atributo. Veremos uma solução mais elegante para isso também.



Hands on!

Obrigado!