Disciplina: ED

Java e Orientação a Objetos

Objetos são acessados por referências

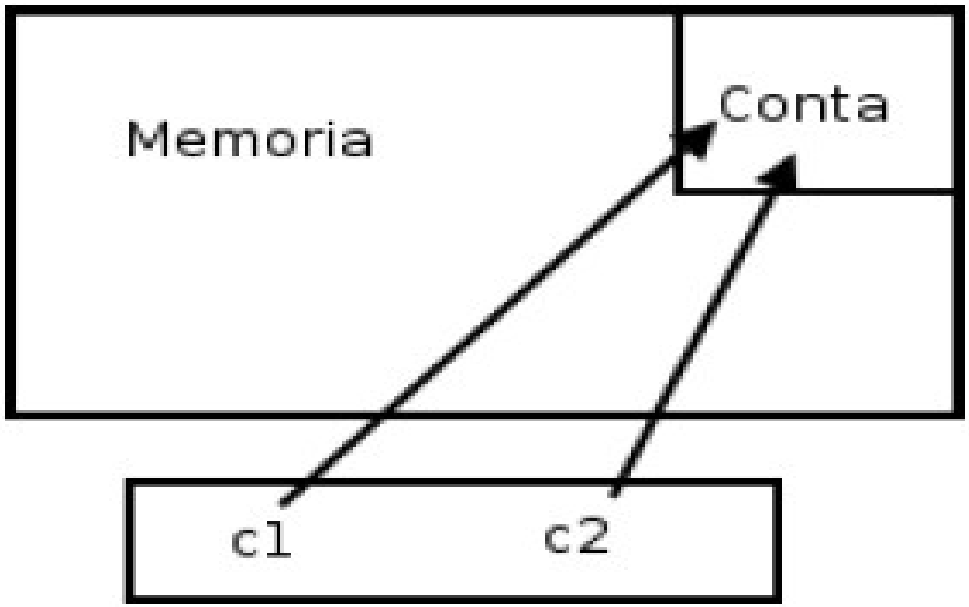
```
1 public static void main(String args[]) {
2       Conta c1;
3       c1 = new Conta();
4
5       Conta c2;
6       c2 = new Conta();
7 }
```

"Tenho uma **referência c** a um **objeto** do tipo **Conta**".

Objetos são acessados por referências

```
1  class TestaReferencias {
2      public static void main(String args[]) {
3          Conta c1 = new Conta();
4          c1.deposita(100);
5
6          Conta c2 = c1;  // linha importante!
7          c2.deposita(200);
8
9          System.out.println(c1.saldo);
10          System.out.println(c2.saldo);
11     }
12 }
```

Qual é o resultado do código acima? O que aparece ao rodar?



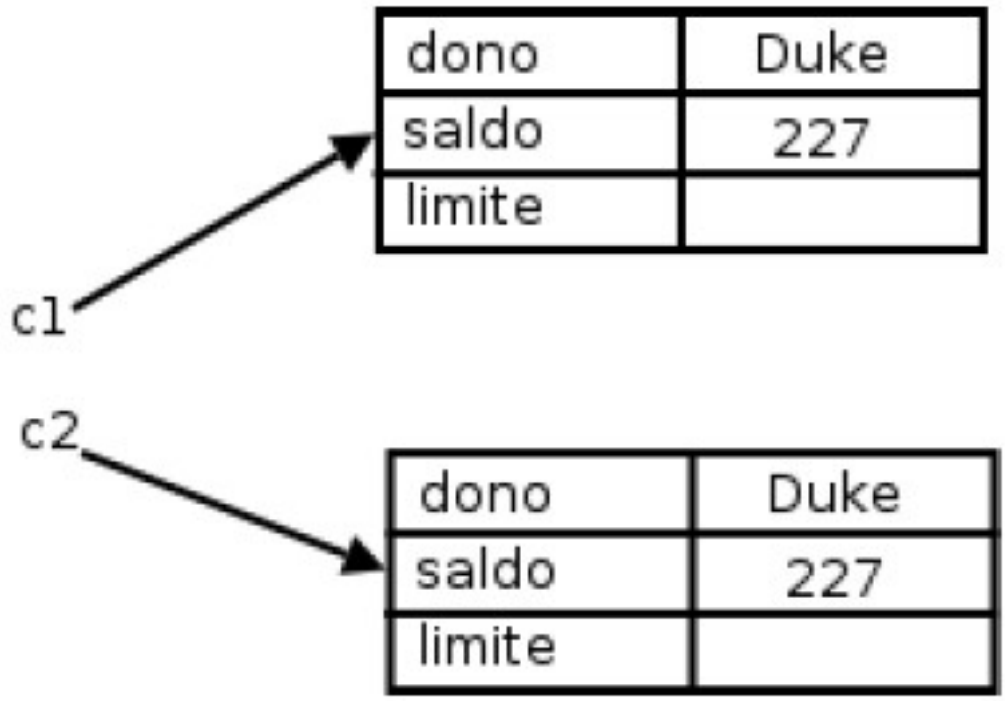
`c2 = c1`, `c2` passa a fazer referência para o mesmo objeto que `c1` referencia nesse instante.

Outra situação

```
1  public static void main(String args[]) {
2      Conta c1 = new Conta();
3      c1.dono = "Duke";
4      c1.saldo = 227;
5
6      Conta c2 = new Conta();
7      c2.dono = "Duke";
8      c2.saldo = 227;
9
10     if (c1 == c2) {
11         System.out.println("Contas iguais");
12     }
13 }
```

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, eles 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.

Para saber se dois objetos têm o mesmo conteúdo, você precisa comparar atributo por atributo.



```java
class Carro {
    String cor;
    String modelo;
    double velocidadeAtual;
    double velocidadeMaxima;

    //liga o carro
    void liga() {
        System.out.println("O carro está ligado");
    }

    //acelera uma certa quantidade
    void acelera(double quantidade) {
        double velocidadeNova = this.velocidadeAtual + quantidade;
        this.velocidadeAtual = velocidadeNova;
    }

    //devolve a marcha do carro
    int pegaMarcha() {
        if (this.velocidadeAtual < 0) {
            return -1;
        }
        if (this.velocidadeAtual >= 0 && this.velocidadeAtual < 40) {
            return 1;
        }
        if (this.velocidadeAtual >= 40 && this.velocidadeAtual < 80 {
            return 2;
        }
        return 3;
    }
}
```

```java
class TestaCarro {
    public static void main(String[] args) {
        Carro meuCarro;
        meuCarro = new Carro();
        meuCarro.cor = "Verde";
        meuCarro.modelo = "Fusca";
        meuCarro.velocidadeAtual = 0;
        meuCarro.velocidadeMaxima = 80;

        // liga o carro
        meuCarro.liga();

        // acelera o carro
        meuCarro.acelera(20);
        System.out.println(meuCarro.velocidadeAtual);
    }
}
```

```
1 class Motor {
2       int potencia;
3       String tipo;
4 }
```

```
1 class Carro {
2       String cor;
3       String modelo;
4       double velocidadeAtual;
5       double velocidadeMaxima;
6       Motor motor;
7
8       // ..
9 }
```

Podemos, agora, criar diversos Carros e mexer com seus atributos e métodos.

Percorrendo uma array

```
public static void main(String args[]) {
    int[] idades = new int[10];
    for (int i = 0; i < 10; i++) {
        idades[i] = i * 10;
    }
    for (int i = 0; i < 10; i++) {
        System.out.println(idades[i]);
    }
}
```

array como argumento em um método

```
void imprimeArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.println(array[i]);
    }
}
```

Percorrendo uma array no Java 5.0

```
class AlgumaClasse{
    public static void main(String args[]) {
        int[] idades = new int[10];
        for (int i = 0; i < 10; i++) {
            idades[i] = i * 10;
        }

        // imprimindo toda a array
        for (int x : idades) {
            System.out.println(x);
        }
    }
}
```

Exercícios: Orientação a Objetos

1) Modele um funcionário. Ele deve ter o nome do funcionário, o departamento onde trabalha, seu salário (`double`), a data de entrada no banco (`String`), seu RG (`String`) e um valor booleano que indique se o funcionário ainda está ativa na empresa no momento ou se já foi mandado embora.

   Você deve criar alguns métodos de acordo com sua necessidade. Além deles, crie um método `bonifica` que aumenta o `salario` do funcionário de acordo com o parâmetro passado como argumento. Crie, também, um método `demite`, que não recebe parâmetro algum, só modifica o valor booleano indicando que o funcionário não trabalha mais aqui.

   A idéia aqui é apenas modelar, isto é, só identifique que informações são importantes e o que um funcionário faz. Desenhe no papel tudo o que um `Funcionario` tem e tudo que ele faz.

2) Transforme o modelo acima em uma classe Java. Teste-a, usando uma outra classe que tenha o `main`. Você deve criar a classe do funcionário chamada `Funcionario`, e a classe de teste você pode nomear como quiser. A de teste deve possuir o método `main`.

Um esboço da classe:

```
class Funcionario {


    double salario;
    // seus outros atributos e métodos

    void bonifica(double aumento) {
        // o que fazer aqui dentro?
    }

    void demite() {
        // o que fazer aqui dentro?
    }
}
```

Você pode (e deve) compilar seu arquivo java sem que você ainda tenha terminado sua classe `Funcionario`. Isso evitará que você receba dezenas de erros de compilação de uma vez só. Crie a classe `Funcionario`, coloque seus atributos e, antes de colocar qualquer método, compile o arquivo java. `Funcionario.class` será gerado, não podemos "executá-la" pois não há um `main`, mas assim verificamos que nossa classe `Funcionario` já está tomando forma.

Esse é um processo incremental. Procure desenvolver assim seus exercícios, para não descobrir só no fim do caminho que algo estava muito errado.

Um esboço da classe que possui o `main`:

```
 1 class TestaFuncionario {

 2
 3     public static void main(String[] args) {
 4         Funcionario f1 = new Funcionario();
 5
 6         f1.nome = "Fiodor";
 7         f1.salario = 100;
 8         f1.bonifica(50);
 9
10         System.out.println("salario atual:" + f1.salario);
11
12     }
13 }
```

Incremente essa classe. Faça outros testes, imprima outros atributos e invoque os métodos que você criou a mais.

Lembre-se de seguir a convenção java, isso é importantíssimo. Isto é, `nomeDeAtributo`, `nomeDeMetodo`, `nomeDeVariavel`, `NomeDeClasse`, etc...

1) Programa 1

   Classe: Pessoa Atributos: nome, idade. Método: void fazAniversario()

   Crie uma pessoa, coloque seu nome e idade iniciais, faça alguns aniversários (aumentando a idade) e imprima seu nome e sua idade.

2) Programa 2

   Classe: Porta Atributos: aberta, cor, dimensaoX, dimensaoY, dimensaoZ Métodos: void abre(), void fecha(), void pinta(String s), boolean estaAberta()

   Crie uma porta, abra e feche a mesma, pinte-a de diversas cores, altere suas dimensões e use o método `estaAberta` para verificar se ela está aberta.

3) Programa 3

   Classe: Casa Atributos: cor, porta1, porta2, porta3 Método: void pinta(String s), int quantasPortasEstaoAbertas()

   Crie uma casa e pinte-a. Crie três portas e coloque-as na casa; abra e feche as mesmas como desejar. Utilize o método `quantasPortasEstaoAbertas` para imprimir o número de portas abertas.

This

This é usado para fazer auto-referência ao próprio contexto em que se encontra. Resumidamente, this sempre será a própria classe ou o objeto já instanciado.