EN05170 - Programação de Computadores II

Introdução a Orientada a Objetos

Prof. Dr. Lidio Mauro Lima de Campos

limadecampos@gmail.com

Universidade Federal do Pará – UFPA
ICEN/FACOMP/BSI

Agenda

- Introduzir conceitos de OO.
- Mostrar como escrever classes reutilizáveis que podem realizar tarefas não triviais.
- Mostrar como a linguagem Java implementa conceitos de POO.
- Entender como os objetos são criados
- Entender como as variáveis de referência são atribuídas
- Criar métodos, retornar valores e usar parâmetros
- Retornar um valor de um método
- Adicionar parâmetros a um método
- Utilizar construtores e criar construtores parametrizados.
- Entender new

Introdução

- OO é uma técnica de programação baseada na construção e utilização de objetos.
- Um objeto combinada dados e operações específicas.
- Um sistema OO é um conjunto de objetos que se relacionam para produzir os resultados específicos

Introdução

- Orientação a objetos é uma técnica para modelagem de sistemas;
- A O.O. contrasta da programação convencional na qual a estrutura e os dados são fracamente acoplados;
- Classe é a essência de Java , sendo o fundamento sobre qual toda linguagem Java se estrutura.

Introdução

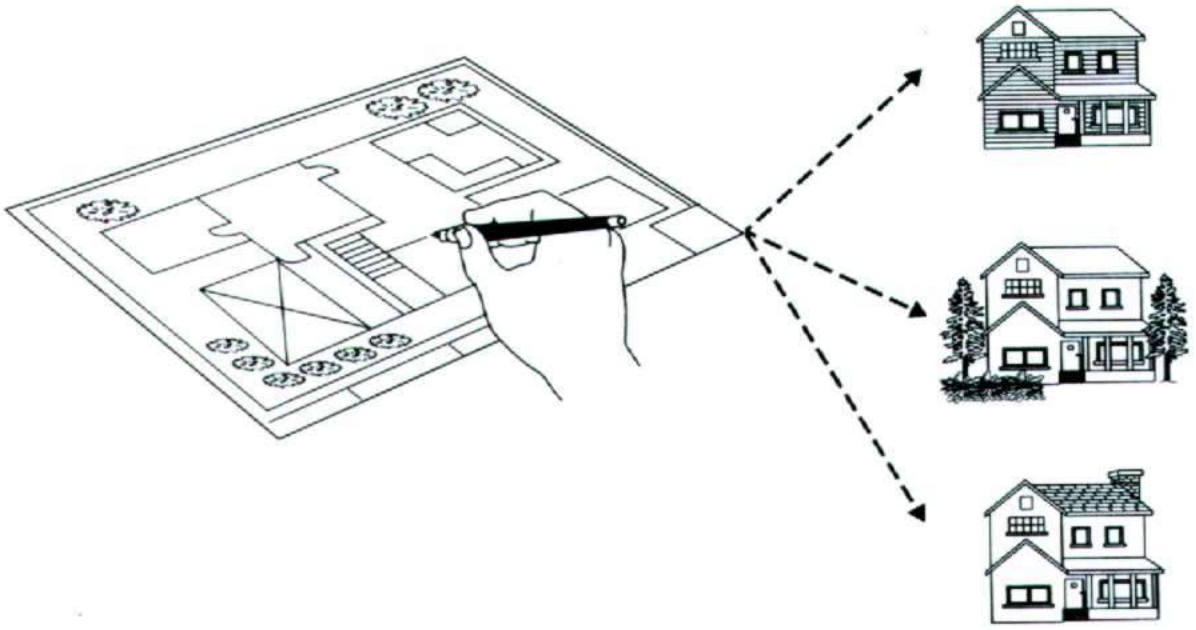
- QUAIS AS VANTAGENS? Orientação a objetos irá ajudá-lo bastante a se organizar e escrever menos, além de concentrar as responsabilidades nos pontos certos, flexibilizando sua aplicação e encapsulando na lógica de negócios.

Fundamentos OO

- Toda a atividade dos programas Java ocorre dentro de uma classe, no capítulo anterior usamos o conceito de classe.
- Um classe é um modelo que define a forma de um objeto.
  - Ela especifica tanto os dados quanto o código que operará sobre eles.
  - Java usa uma especificação de classe para construir objetos.

Fundamentos OO

- Classes são formas de objetos
  - Objetos são **instâncias** de classes
  - Uma classe pode ter várias instâncias.



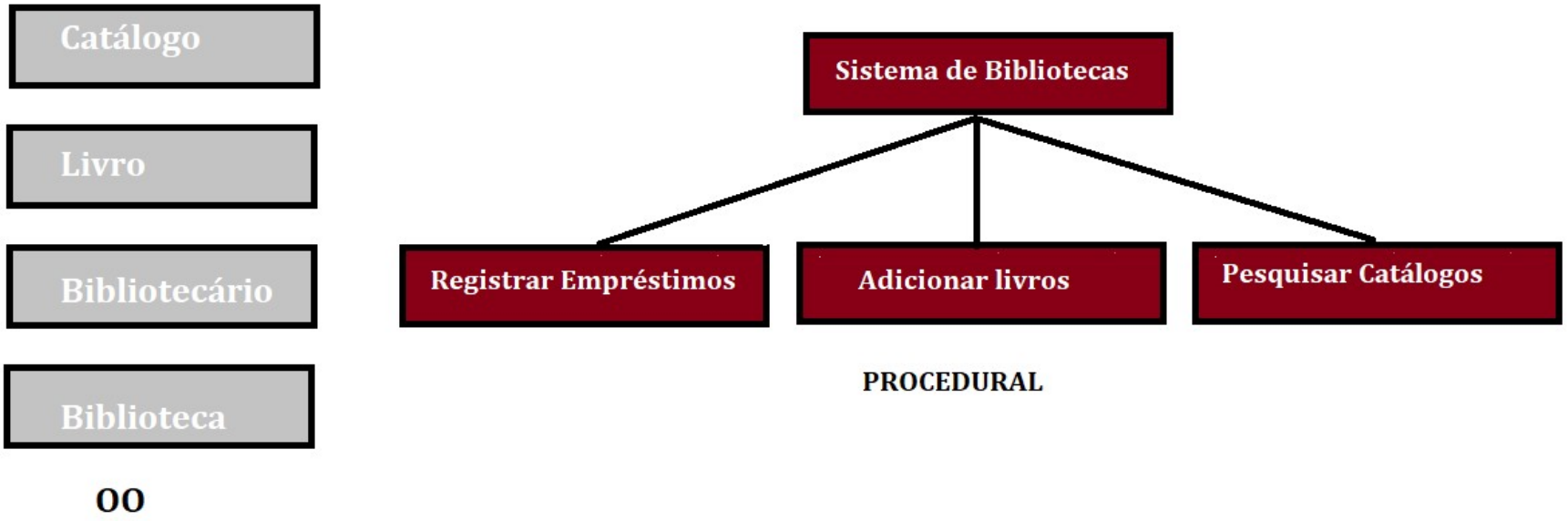
Fundamentos OO

- Propósito da OO
    - Facilitar a modelagem de problemas do mundo real com a utilização de programas computacionais.

- Objetos no mundo real tem:
    - Comportamentos
    - Estados
        - Ex : cachorro
            - Estados (cor, nome, com fome....) e
            - Comportamentos (comer, latir, sujar a casa...)

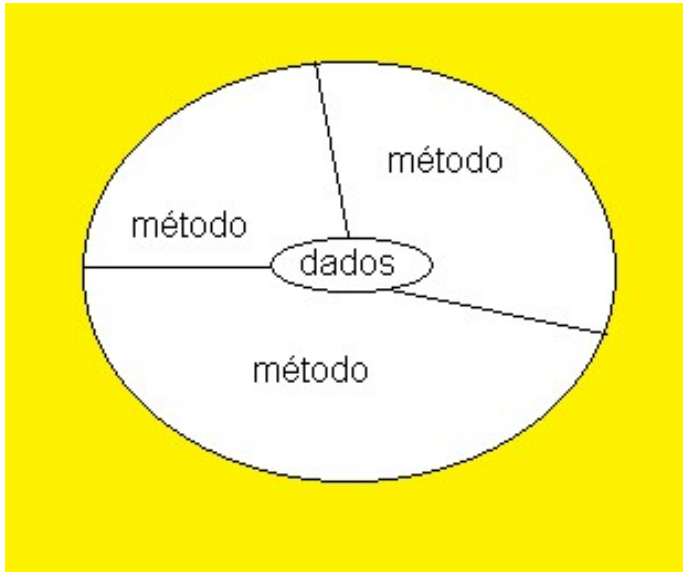
Fundamentos OO

- Forma de abstrair o problema – dividir & conquistar;
- Comportamento e dados fortemente Acoplados;
- Abstrações próximas do mundo real.



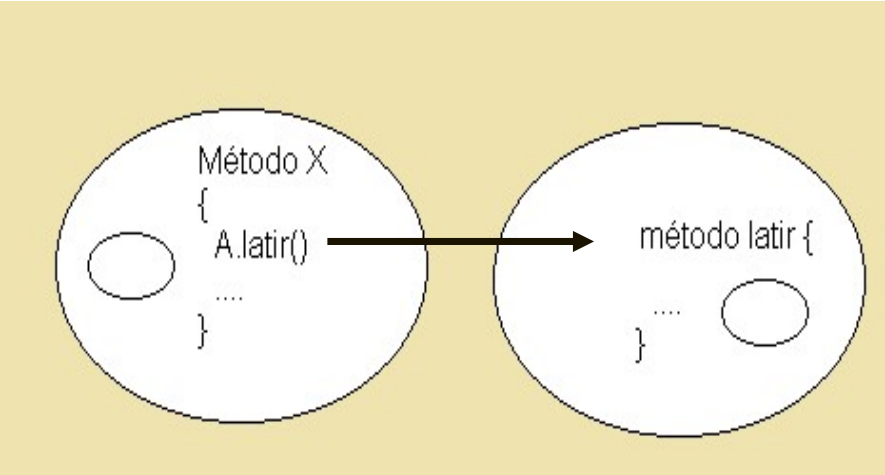
Fundamentos OO

- Objetos de software tem seus **estados encapsulados** em **atributos**.
- Seus **comportamentos** estão descritos em **métodos** que podem acessar os valores dos atributos.



Fundamentos OO

- Os objetos comunicam-se pela passagem de mensagens, que são chamadas a métodos do objeto.
    - Ex: A é um objeto que tem um método latir B pode pedir para A latir chamando este método.



Fundamentos OO

- Allan Kay, um dos criadores do SmallTalk:
  - *Tudo* são objetos;
- Um *programa* é um grupo de objetos enviando mensagens uns aos outros.
- *Um objeto é qualquer coisa, real ou abstrata, na qual nós armazenamos dados e as operações que manipulam os dados* [Martin]

Orientação a Objetos

- A forma geral de uma **definição de classe** aparece a seguir.

```
class nome-classe extends nome-superclasse
{
  tipo var-instancia1;
  tipo var-instancia2;
   .......
  tipo var-instanciaN;

  tipo nome-metodo1(lista-de-parâmetros)
    {corpo-método;}
   ........
  tipo nome-métodoN(lista-de-parâmetros)
    {corpo-método;}
}
```

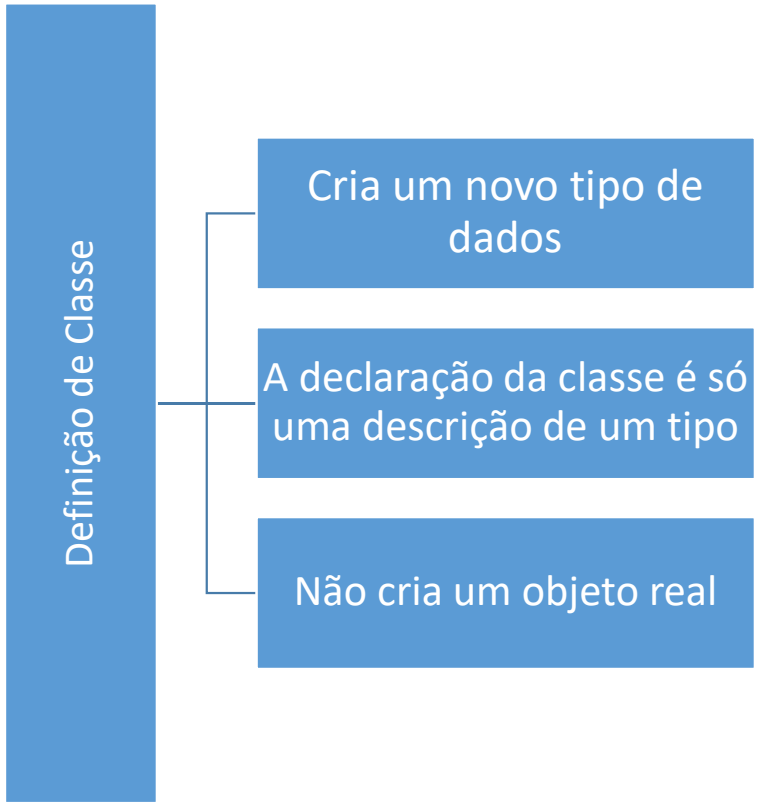
Orientação a Objetos

- Definindo uma classe
- Desenvolver uma classe que encasula informações sobre veículos, como carros, furgões e caminhões.
- Essa classe se chama Veiculo e conterá três informações sobre o veículo:
  - Número passageiros (passageiros)
  - Capacidade de armazenamento de combustível (ccombustivel)
  - Consumo médio de combustível (consumo)

Orientação a Objetos

- Definindo uma classe – A Classe veículo abaixo define as três variáveis de instância.

```
class Veiculo
{  int passageiros;
    int ccombustível;
    int consumo;
}
```



Orientação a Objetos

- ...Declarando objetos em Java

Etapas:

1)Declarar uma variável do tipo

```
Veiculo minivan;
```

2)Instanciar o objeto

```
Veiculo minivan=new Veiculo();
```

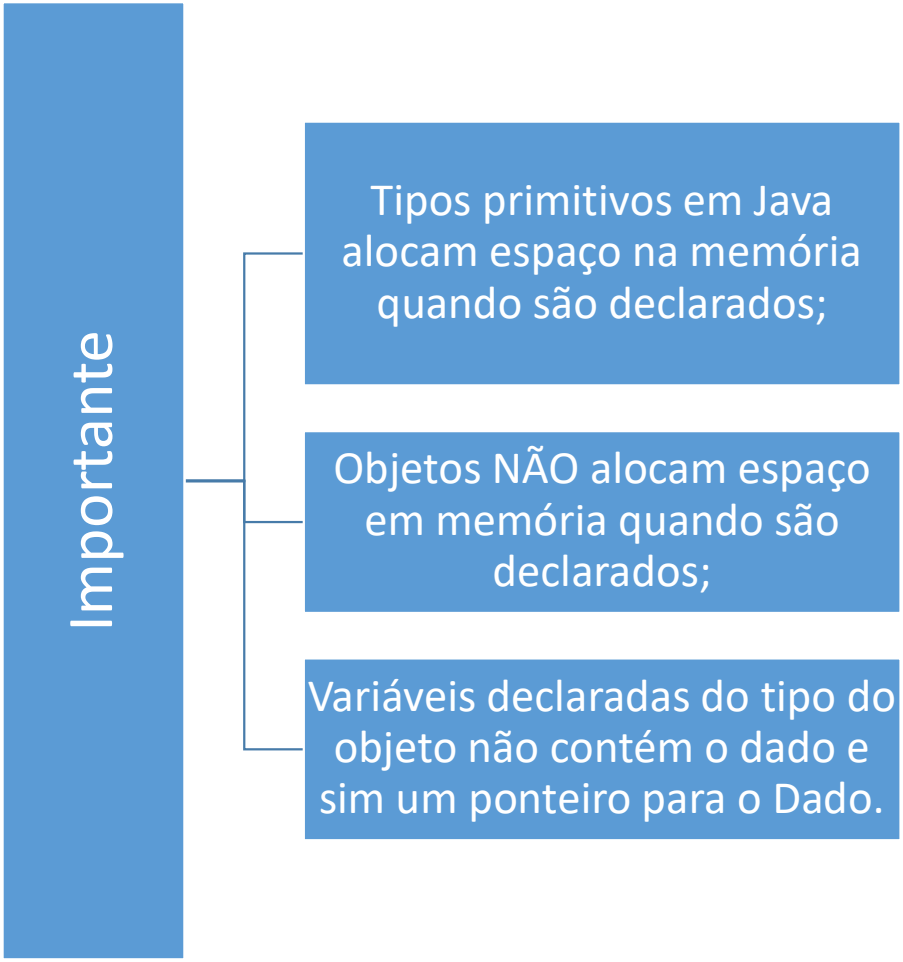
3)Acessar o objeto

```
minivan.passageiros  = 7;
minivan. ccombustivel = 50;
minivan.consumo=11
```

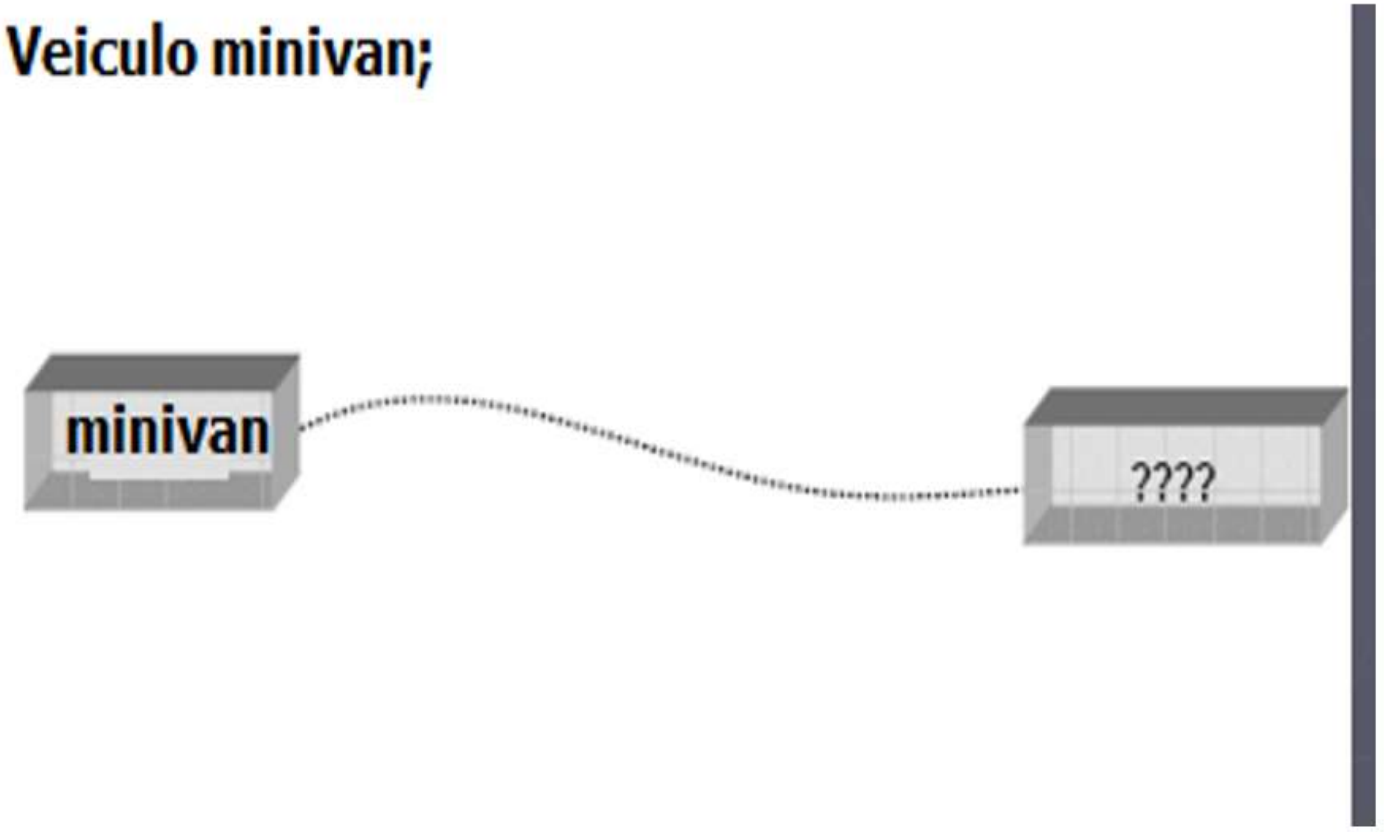
Importante

- Tipos primitivos em Java alocam espaço na memória quando são declarados;
- Objetos NÃO alocam espaço em memória quando são declarados;
- Variáveis declaradas do tipo do objeto não contém o dado e sim um ponteiro para o Dado.

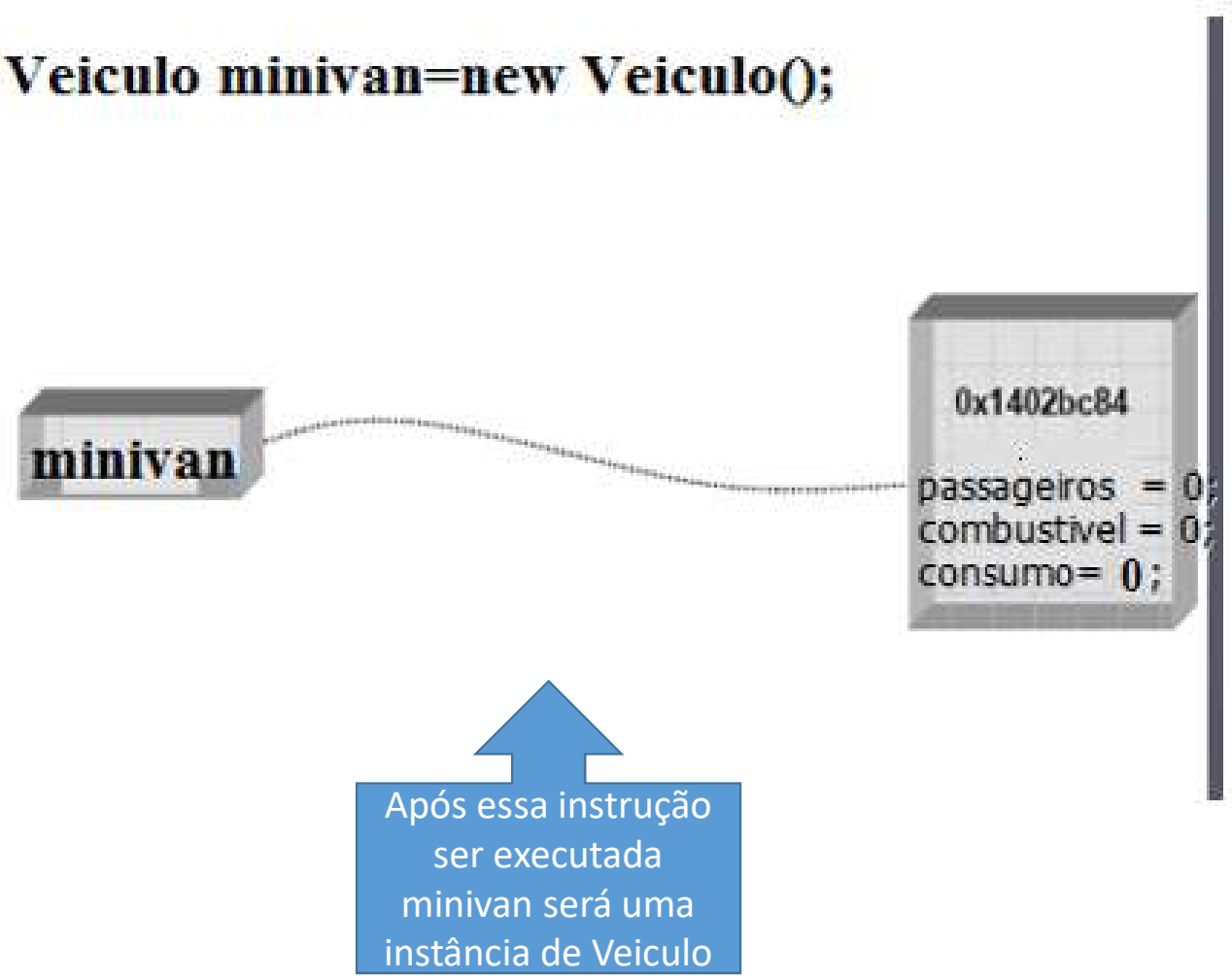
Orientação a Objetos



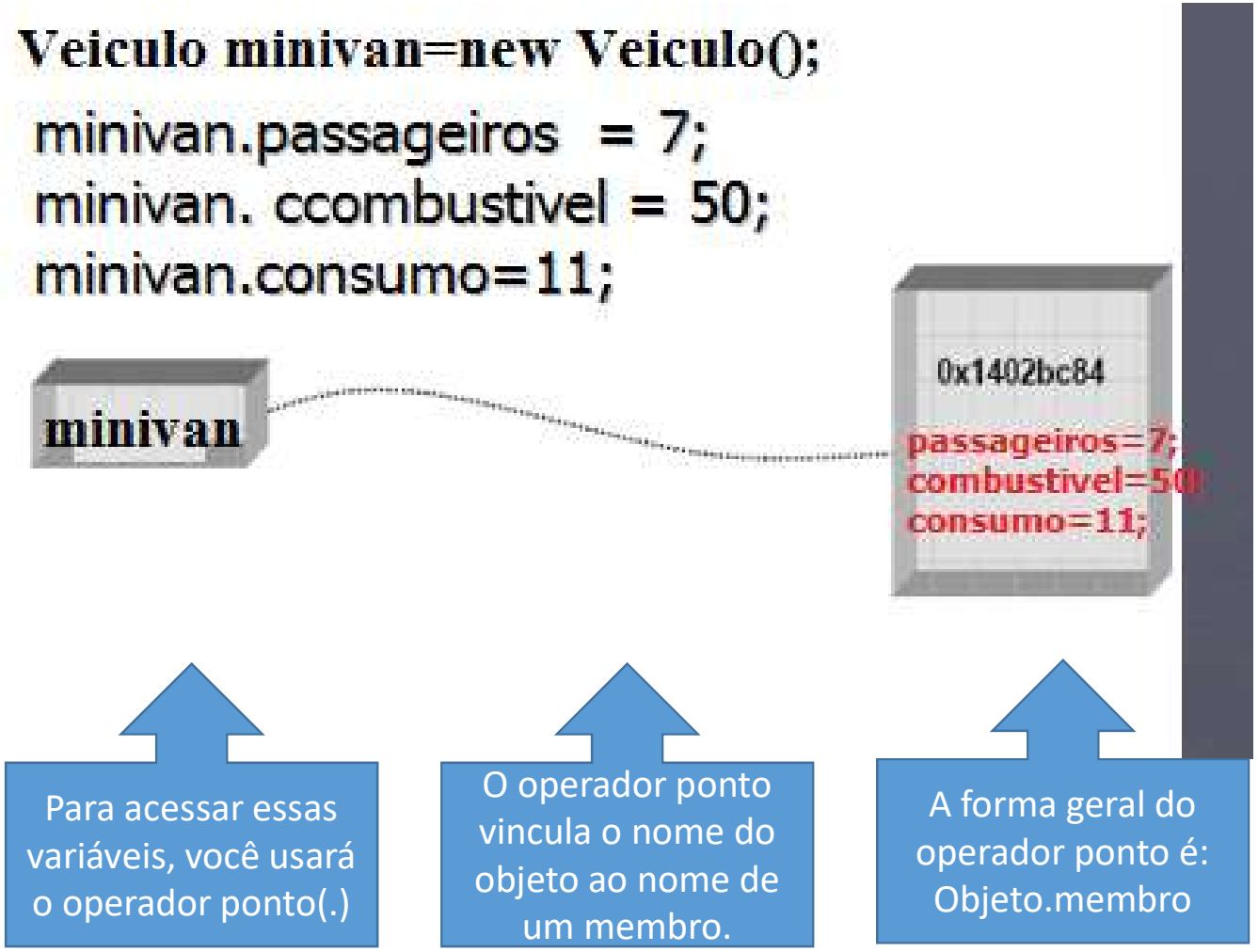
Orientação a Objetos



Orientação a Objetos



Orientação a Objetos



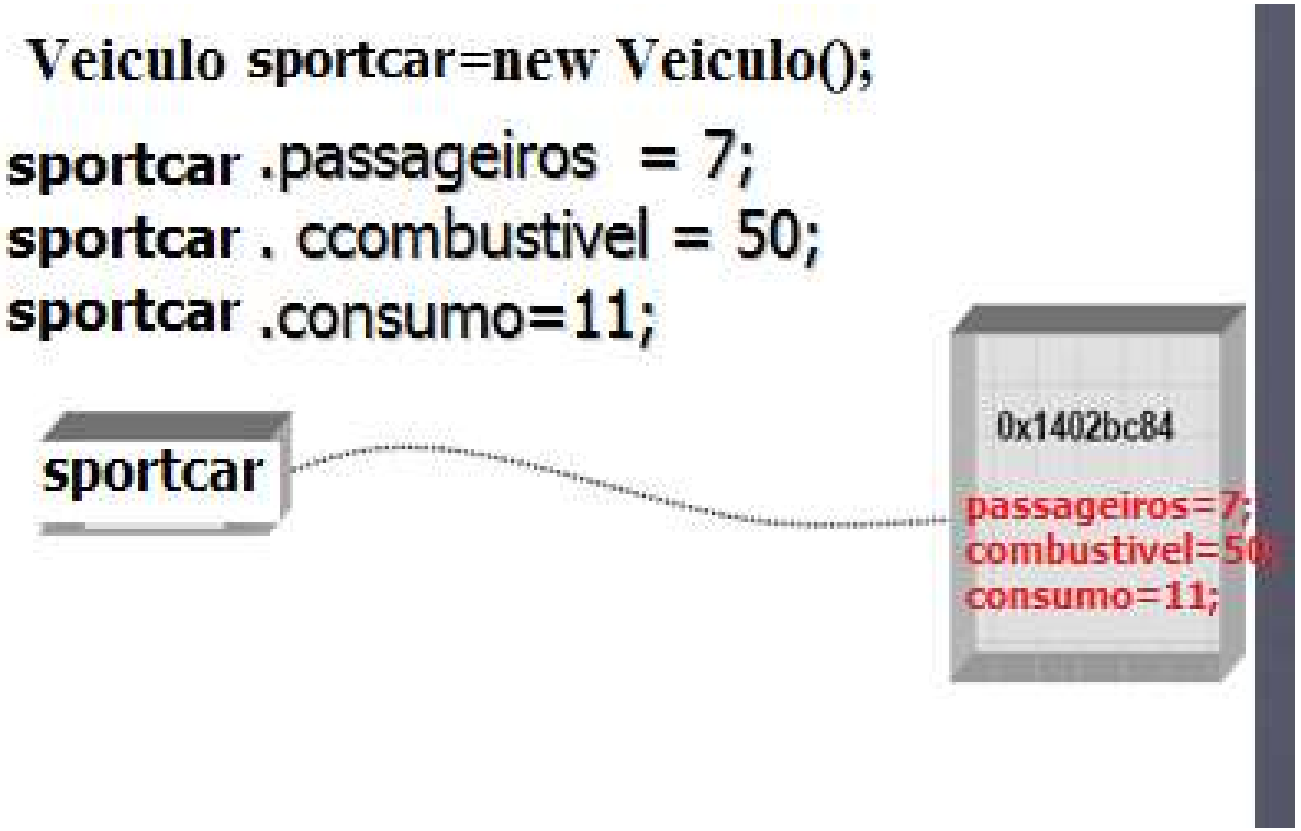
Orientação a Objetos

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

        Veiculo minivan=new Veiculo();
        minivan.passageiros=7;
        minivan.ccombustível=50;
        minivan.consumo=11;
        int autonomia=minivan.ccombustível*minivan.consumo;

        System.out.println("Minivan pode transportar:"+minivan.passageiros+
                "com uma autonomia :"+minivan.consumo);
    }
}
```

Orientação a Objetos



Orientação a Objetos

```
Veiculo carro1=new Veiculo();
Veiculo carro2=carro1;
carro1.passageiros=5;
carro1.ccombustível=50;
carro1.consumo=13;

System.out.println(carro1.ccombustível);
System.out.println(carro2.ccombustível);
```

Exibirão o mesmo valor : 50

Orientação a Objetos

- **Métodos** – contém uma ou mais instruções, cada método tem um nome e é esse que é usado para chamá-lo.

- A forma geral de um método é:

```
tipo-retorno nome(lista-parâmetros)
{
 //corpo do métodos
}
```

- **tipo-retorno** pode ser void.

- **lista de parâmetros** – é uma sequencia de pares separados por vírgulas compostos por tipo e identificador.
  - Os parâmetros são variáveis que recebem o valor dos *argumentos* passados para o método quando ele é chamado.
  - Se não tiver parâmetros a lista será vazia.

Orientação a Objetos

- **1)Adicionando um método à classe Veiculo**

```
public class Veiculo1
{
    int passageiros; //numero de passageiros
    int ccombustível; //capacidade de combustível
    int consumo; //consumo de combustível em km por litro
    void autonomia()
   {Sytem.out.println(“ Autonomia e”+ccombustível*consumo); }
}
```

Orientação a Objetos

- **1)Modificando o método main**

```
public class InstanciaVeiculo1
{
   public static void main(String[] args)
   {
     Veiculo minivan=new Veiculo();
      minivan.passageiros=7;  minivan.ccombustível=50; minivan.consumo=11;
     Veiculo sportscar=new Veiculo();
     sportscar.passageiros=6;  minivan.ccombustível=70; minivan.consumo=10;
     System.out.println("Minivan pode transportar:"+minivan.passageiros);
     minivan.autonomia();
     System.out.println(“Sportcar pode transportar:"+sportcar.passageiros);
     sportcar.autonomia();
   }
}
```

Orientação a Objetos

- 2)Método quem Retornam um valor

```
public class Veiculo2
{
   int passageiros; //numero de passageiros
   int ccombustível;//capacidade de combustível
   int consumo;//consumo de combustível em km por litro

   int autonomia()
   {
     return ccombustível*consumo;
   }

}
```

Orientação a Objetos

- **2)Modificando o método main**

```
public class InstanciaVeiculo2
 {    public static void main(String[] args) {
      Veiculo minivan=new Veiculo();
      minivan.passageiros=7;  minivan.ccombustível=50; minivan.consumo=11;
      Veiculo sportscar=new Veiculo();
      sportscar.passageiros=6;  minivan.ccombustível=70; minivan.consumo=10;
      int aut1,aut2;
      aut1=minivan.autonomia();
      aut2=sportcar.autonomia();
      System.out.println("Minivan pode transportar:"+minivan.passageiros+”com autonomia de:”+aut1);
      System.out.println(“Sportcar pode transportar:"+sportcar.passageiros+
       +”com autonomia de:”+aut2);
  }
}
```

Orientação a Objetos

- **1)Exercício defina uma classe chamada ChkNum, que possuirá um método ePar que recebe como parametro um número inteiro e retorna true caso o valor passado na classe principal seja par. Na classe principal (ParDemo), intanciar três objetos ChkNum passando três valores inteiros nas chamadas do método ePar, após o retorno ao ponto de chamada, imprimir se o numero passado é par ou impar.**

```
class ChkNum {
  // return true if x is even
  boolean ePar(int x)
  {
      return (x%2) == 0;
   }
}
```

Orientação a Objetos

```
class ParmDemo {
  public static void main(String args[]) {
    ChkNum e = new ChkNum();

    if(e.ePar(10)) System.out.println("10 e par.");

    if(e.ePar(9)) System.out.println("9 e par.");

    if(e.ePar(8)) System.out.println("8 e par.");

  } }
```

Orientação a Objetos

- **2)Construir uma classe Veiculo 3, a partir da classe Veículo2 construída anteriormente, que além do método autonomia , tenha outro método chamado necombustivel, que retorna um double, esse valor é a necessidade de combustível de um veículo em litros ao realizar um determinado percurso em Km. Na classe principal , fazer instâncias da classe veiculo 3, imprimindo a necessidade de combustivel de um veiculo , dada uma distância percorrida em km e o consumo do mesmo em (Km/l).**

Orientação a Objetos

- **3)Usando Parâmetros na classe Veículo**

```
public class Veiculo3
{
   int passageiros;  //numero de passageiros
   int ccombustível; //capacidade de combustível
   int consumo; //consumo de combustível em km por litro

   int autonomia()
   {return ccombustível*consumo;}

   double necombustivel(int distkm) {
   return (double) distkm / consumo;
}
}
```

Orientação a Objetos

```
public class InstanciaVeiculo3
{       public static void main(String[] args)
        {
         int dist=1000; double litros;
         Veiculo minivan=new Veiculo();
          minivan.passageiros=7;  minivan.ccombustível=50; minivan.consumo=11;
          Veiculo sportscar=new Veiculo();
          sportscar.passageiros=6;  minivan.ccombustível=70; minivan.consumo=10;
          litros=minivan.necombustivel(dist);
         System.out.println(“Para percorrer:"+dist+”km minivan necessita:”+litros+”de
combustivel”);
        litros=sportscar.necombustivel(dist);
       System.out.println(“Para percorrer:"+dist+”km sportscar necessita:”+litros+”de
combustivel”); } }
```

Orientação a Objetos

- **Métodos Construtores**
- Um construtor inicializa um objeto quando este é criado.
- **Características:**
  - Tem o mesmo nome de sua classe.
  - Sintaticamente semelhante a qualquer método.
  - Não tem um tipo de retorno explicito.
  - Normalmente são usados para fornecer os valores iniciais para as variáveis de instancia definidas pela classe.
  - Todas as classes tem construtores, mesmo quando não definidos.
  - Java fornece um construtor padrão que inicializa todas as variáveis membros com seus valores padrão : zero, null e false, para tipos numericos, tipos referencia e booleans, respectivamente.

Orientação a Objetos

- **Exemplo de Construtor**

```
class MinhaClasse
{
  int x;
  MinhaClasse()
  {
    x = 10;
  }
}
```

Orientação a Objetos

- **Exemplo de Construtor**

```
class ConsDemo
{
  public static void main(String args[]) {
    MinhaClasse t1 = new MinhaClasse();
    MinhaClasse t2 = new MinhaClasse();

    System.out.println(t1.x + " " + t2.x);
  }
}
```

Orientação a Objetos

- **3)Alterar a classe MinhaClasse, construida anteriormente, para que o método construtor da mesma receba um valor inteiro e inicialize a variável de instância x com esse valor, após isso, na classe principal, realizar instâncias de MinhaClasse passando valores inteiros como parâmetros ao método construtor de MinhaClasse ao ser instanciado. Imprimir na classe principal os valores de x de cada objeto instanciado.**

Orientação a Objetos

Exemplo de Construtor parametrizado

```
class MinhaClasse
{
  int x;
  MinhaClasse(int i)
 {
    x = i;
  }
}
```

Orientação a Objetos

Exemplo de Construtor parametrizado

```
class ParmConsDemo
{
  public static void main(String args[]) {
    MinhaClasse t1 = new MinhaClasse(10);
    MinhaClasse t2 = new MinhaClasse(88);

    System.out.println(t1.x + " " + t2.x);
  }
}
```

Orientação a Objetos

4)Modificar a classe Veiculo3, adicionando um construtor a mesma, esse método deve inicializar as variaveis de instância int passageiros; int ccombustível; int consumo. A classe principal deve instanciar os objetos, passando os parâmetros para o método construtor, a seguir imprimir a autonomia de cada objeto criado e a necessidade de combustivel para relizar uma viagem de x Km , considerando o consumo do veiculo.

Orientação a Objetos

4)Adicionando um Construtor a classe veiculo

```
public class Veiculo4
{
   int passageiros;  int ccombustível; int consumo;

   Veiculo(int pass,int comb,int cons)
   {passageiros=pass; ccombustível=comb; consumo=cons;}

   int autonomia()
  {return ccombustível*consumo;}

  double necombustivel(int distkm)
  {return (double) distkm / consumo;}
}
```

Orientação a Objetos

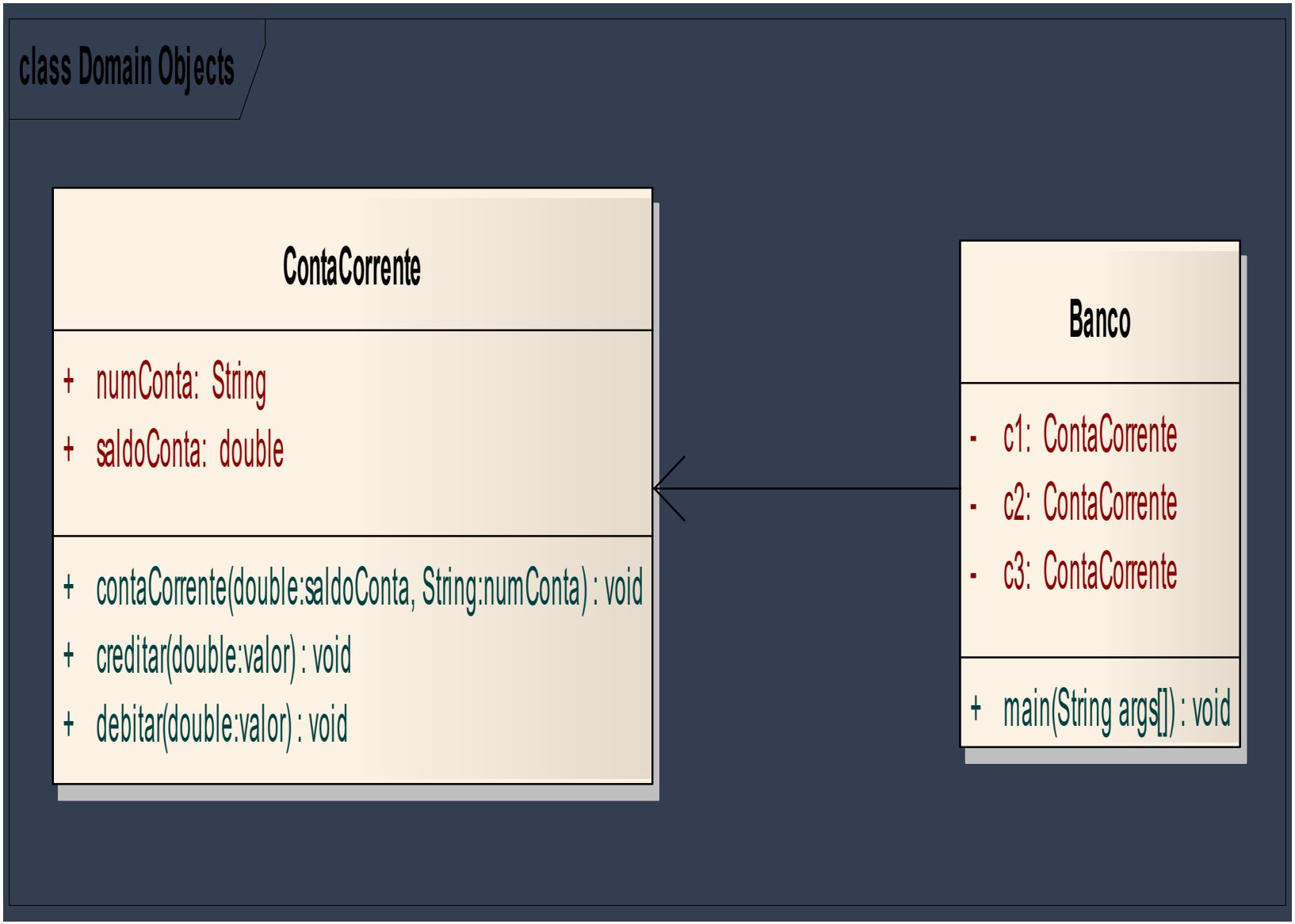
- **4)Modificando o método main**

```
public class InstanciaVeiculo4
{ public static void main(String[] args)
 { int dist=1000; double litros;
   Veiculo minivan=new Veiculo(7,50,11);
   Veiculo sportscar=new Veiculo(6,70,10);
    litros=minivan.necombustivel(dist);
    System.out.println(“Para percorrer:"+dist+”km minivan necessita:”+litros+”de combustivel”);
    litros=sportscar.necombustivel(dist);
    System.out.println(“Para percorrer:"+dist+”km sportscar necessita:”+litros+”de combustivel”);
 }
}
```

Exercício-Estudo de Caso 1

- Crie uma classe chamada ContaCorrente que contém as seguintes características:
  - Propriedades: numero (String), saldo (double)
  - Métodos:
    - debitar e creditar: recebendo um parâmetro double e fazendo a devida operação com o saldo;
    - Construtor: Recebendo o número da conta e o saldo inicial.
- Para testar, declare uma classe chamada Banco e chame os métodos da classe ContaCorrente.

Diagrama UML



Estudo de Caso 1.1

```
public class ContaCorrente
{ String numero;  double saldo;
  public ContaCorrente(String numConta,double saldoConta)
  {numero=numConta;saldo=saldoConta; }
  public void creditar(double valor)
  {saldo=saldo+valor; }
  public void debitar(double valor)
  {if (saldo >= valor)
  {saldo = saldo - valor;  }
   }}
```

Estudo de Caso 1.1

```
public class Banco {
   public static void main(String args[])
  {ContaCorrente c1=new ContaCorrente("1001",1090);
   ContaCorrente c2=new ContaCorrente("1002",1290);
   ContaCorrente c3=new ContaCorrente("1003",1340);
   ContaCorrente c4=new ContaCorrente("1004",1870);
   c1.creditar(100); c2.creditar(2100);c3.debitar(100);
   c4.debitar(2100);
   System.out.println("Numero Conta:"+c1.numero+"
Saldo="+c1.saldo);}}
```

- ONDE PESQUISAR
  - CAPÍTULO 4 – COREJAVA

- Referências

- Peter Jandl Junior. **Java - Guia do Programador: Atualizado Para Java 16 - 4ª edição. NOVATEC. 2021.**
- Herbert Schildt. Java: a referência completa. ALTA BOOKS Editora. 2020.