ORIENTAÇÃO A OBJETOS

AULA 3

Classes e Objetos

Vandor Roberto Vilardi Rissoli

APRESENTAÇÃO

- Programação Orientada a Objeto
- Classes e Objetos
- Sobrecarga
- Correção de leitura (*buffer*)
- Referências

Histórico

A Orientação a Objeto (OO) é proveniente da década de 70:

- Nos anos 60 a linguagem SIMULA67 já implementava alguns conceitos de OO
- SIMULA68 foi a primeira linguagem a suportar estes conceitos de OO
- Popularização da OO com Linguagem Smaltalk (elaborada pela Xerox)
- Incorporação destes conceitos por outras linguagens como: C++, Objetct Pascal (Delphi), Java, entre outras
- A grande popularização da OO aconteceu com Java

Programação Orientada a Objeto

- A OO possibilita uma nova forma de programação e desenvolvimento de sistemas computacionais, promovendo a programação a um novo "patamar" - POO
- A POO contribui, significativamente, com a qualidade na produção de software, principalmente quando todos seus conceitos são oferecidos pela tecnologia adotada
- A OO agrega a Programação o uso de "boas práticas" e padrões de projeto (*design patterns*)

Java

- Nativo em OO
- Suporta OO completamente (POO)
- Possuir recursos para uso dos padrões de projeto

Programação Orientada a Objeto

Programação Orientada a Objeto - POO

A **POO** é mais intuitiva e fácil de aprender porque trabalha conceitos pertinentes ao mundo real na solução de um problema computacional.

Entre suas vantagens se destacam:

- Aumento da produtividade
- Reuso ou reaproveitamento de código
- Uso de componentes
- Flexibilização do software
- Escalabilidade
- Facilidade de manutenção, entre outras.

Programação Orientada a Objeto

Principais Conceitos em POO

- Classe
- Objeto
- Abstração
- Mensagem
- **Encapsulamento**
- **Herança**
- **Polimorfismo**

→ Em POO, as estruturas de um programa são definidas usando conceitos do mundo real, o que permiti componentizá-los em partes separadas por responsabilidades e que se comunicam através de mensagens. Essas partes são chamadas de **OBJETOS**.

Classes e Objetos

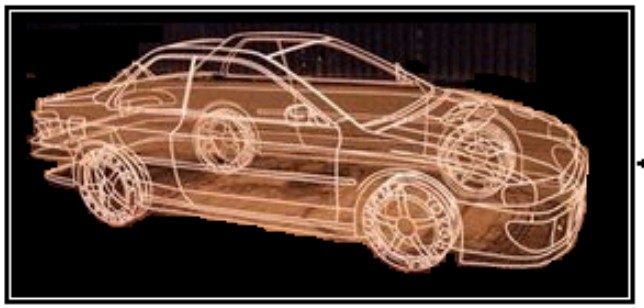
A vida de cada indivíduo é repleta de objetos de diversos tipos e formas, onde o contato com os mesmos é que nos permite identificar as suas características físicas, sua composição, sua forma, etc.

Por exemplo: observando um carro pode-se reconhecer sua forma, seu modelo, sua cor, enfim suas diversas características.

Qualquer outro carro é diferente deste primeiro observado, apesar de alguns deles possuírem forma, modelo, cor e diversas outras características comuns.

A esta situação, muito comum em nosso dia a dia, a POO tenta simular, trazendo seus aspectos principais para Programação Computacional.

Classes e Objetos



Classe
(modelo)

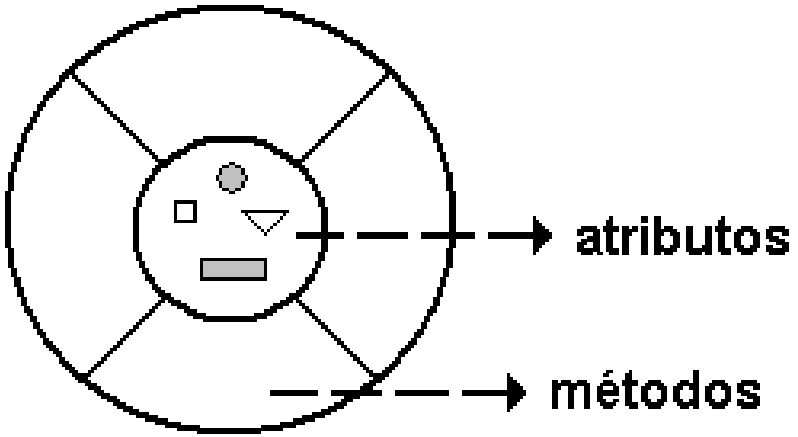
Objetos
(instanciações)

Classes e Objetos

CLASSE

- Consiste na **forma** (modelo) para criação de objetos moldados por esta **forma** (objetos do mesmo tipo)
- Define os atributos (ou variáveis) e métodos (ou funções) comuns aos objetos do mesmo tipo
- Os objetos são criados a partir de suas classes (modelos ou protótipos)

Representação Gráfica de Classe



Classes e Objetos

OBJETO

- Criações provenientes de uma classe que possuem estado independente, fornecido por suas variáveis, e comportamento definido por seus métodos
- Para cada objeto é alocada nova área de memória, coerentemente as características definidas em sua classe original
- O estado de um objeto revela seus dados, por exemplo:

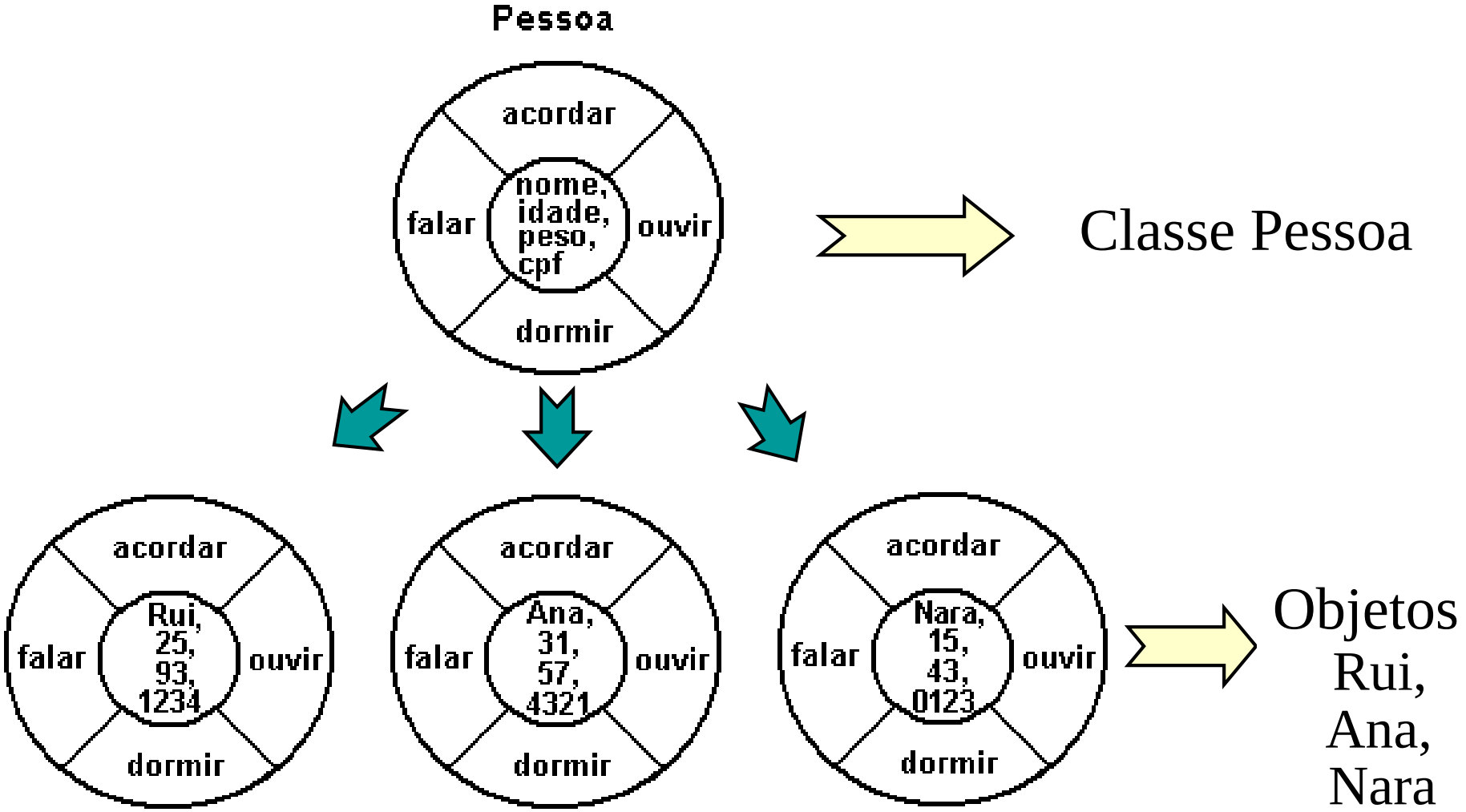
  Pessoa – nome, idade, peso, cpf

- O comportamento do objeto corresponde as suas ações que podem ser executadas, por exemplo:

  Pessoa – acordar, dormir, falar, ouvir

Classes e Objetos

Representação Gráfica

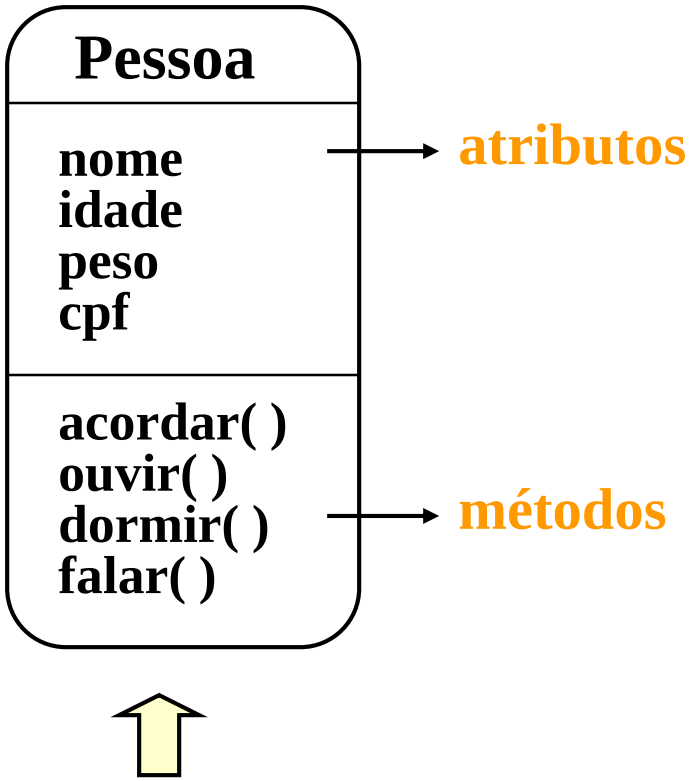


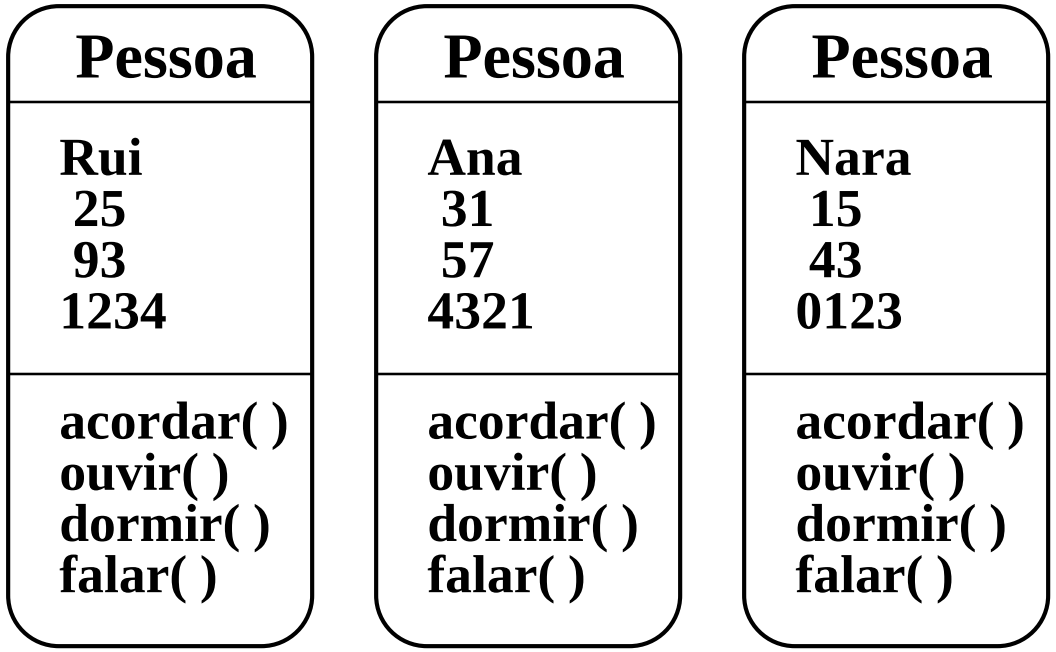
Classes e Objetos

- Objetos podem ser concretos (pessoa, carro, livro, etc.) e abstratos (conta corrente, venda, pessoa jurídica, etc.)
- Os objetos do mundo real são modelados e representados em Programação por meio de software
- Objetos de software também possuem estado e comportamento, como os objetos do mundo real, sendo
  - **Estado** armazenado em seus atributos (variáveis)
  - **Comportamento** implementado por seus métodos
- Um programa pode manter um ou vários objetos ativos, sendo cada um destes objetos ativos chamados de **instâncias** de determinada classe
- Cada instância possui seu próprio estado

Classes e Objetos

Observe as instâncias de Pessoa representadas anteriormente e conheça outra forma de representação de objetos mais simples na elaboração.





3 objetos ou 3 instâncias de Pessoa

Classes e Objetos

Programando Classes em Java

- A sintaxe genérica para criação de uma classe é:

```
<qualificador de acesso> class <identificador> {
        // atributos desta classe
        // métodos desta classe
}
```

- <qualificador de acesso> indica como a classe pode ser acessada por outras classes
- **<identificador>** nome que identifica a classe, sendo iniciado com caracter alfabético em maiúsculo por convenção
- **class** palavra reservada que indica descrição de uma classe

→ par de chaves que indica a descrição da classe (início/fim)

Classes e Objetos

Observe o trecho do exemplo de definição da classe Carro.

```
public class Carro { // similar estruturas heterogêneas
    int codigoCor;
    boolean status;
    String placa;
}
```

- A classe que possuir o método main() é executável (aplicação) e pode utilizar outras classes externas
- Um único arquivo **.java** pode conter várias classes, mas apenas uma delas pode ser pública (qualificador public)
- Ao se criar uma classe Java, seu arquivo físico (.java) deve possuir o mesmo nome da classe pública deste arquivo, com cuidado na precisão dos maiúsculos e minúsculos existentes no nome desta classe e o arquivo

Classes e Objetos - Abstração

O conceito de abstração está associado à característica de se observar somente os aspectos de interesse envolvidos com os elementos do mundo real que sejam importantes ao problema em questão.

Por exemplo:

Se pensarmos no conceito de **Empregado** e o abstrairmos poderemos identificar informações relevantes e comuns, porém é necessária a capacidade de utilização de informações importantes ao tratamento desejado. Observe qual a relevância da informação de cor dos olhos e dos cabelos para seleção de um Empregado no setor de Expedição de uma loja de magazine.

Classes e Objetos

Declaração de Atributos

Na criação da classe Carro foram definidos 3 atributos, respeitando a sintaxe de declaração de atributos (variáveis) para uma classe.

```
public class Carro {
    private int codigoCor;
    private booelan status;
    private String placa;
}
```

3 atributos da classe Carro
- **codigoCor**
- **status**
- **placa**

Forma Genérica

<qualificador> <tipo dado> <identificador>;

qualificador de acesso ao atributo

tipo de dado do atributo

nome do atributo (variável)

Classes e Objetos

- O processo de instanciação consiste na criação de um objeto a partir de uma classe
- Este processo aloca espaço de memória para armazenar o objeto, semelhante ao que ocorre na declaração de variáveis
- Para instanciar um objeto é usado o operador **new** (aloca área de memória para uso do objeto)
  - Classe – modelo abstrato de base aos objetos
  - Objetos – modelo concreto que ocupa memória
- Um programa pode manter um ou vários objetos ativos, sendo cada um destes objetos ativos chamados de instâncias de determinada classe
- Cada instância possui seu próprio estado, individualizando seus atributos (variáveis de instância)

Classes e Objetos

Observe o trecho do exemplo a seguir, mas atente a necessidade da classe Carro estar armazenada na mesma pasta (pacote) desta nova classe TemCarro.

```
public class TemCarro {
  public static void main(String[] args) {
   // Declarações
     Carro auto1, auto2;    // usa outra classe Carro
     auto1 = new Carro();   // aloca espaço de 1 Carro
     auto2 = new Carro();   // aloca espaço outro Carro
   // Instruções
     System.out.println(auto1.placa + "\t" +
                auto1.codigoCor + "\t" + auto1.status);
     auto1.codigoCor = 2;
     auto1.status = true;
     auto1.placa = "BDB1020";
     System.out.println(auto1.placa + "\t" +
                auto1.codigoCor + "\t" + auto1.status);
```

Classes e Objetos

```
        // continuação do exemplo anterior
        auto2.codigoCor = 1;
        auto2.status = true;
        System.out.println(auto2.placa + "\t" +
                    auto2.codigoCor + "\t" + auto2.status);
    }
}
```

- As primeiras instruções de saída de dados (System) mostram as variáveis sem atribuição nenhuma de valor
- **private** é um qualificador de proteção de acesso em Carro
- Quando um objeto é criado (**new**) e suas variáveis não recebem nenhum valor inicial, lhes serão atribuídos os **valores padrões** (*default*) de Java
  - Tipos numéricos: inicia com **zero** (0)
  - Tipos lógicos: inicia com **false**
  - Objetos String: inicia **null** (sem valor)

Classes e Objetos

Atributos Estáticos

Cada objeto (instância) trata seus atributos, provenientes da classe, de forma exclusiva, onde cada variável esta atrelada a seu próprio objeto.

No entanto, pode existir a necessidade de variáveis atreladas a classe e não a individualidade dos objetos.

A implementação desta funcionalidade de **compartilhamento de um atributo** (variável) entre todos seus objetos é possível em Java por meio da declaração static da variável, tornando-a estática, ou seja, controlada pela classe. Por exemplo:

```
static int codigoCor;
```

Classes e Objetos

Suponha estas alterações nas classes Carro e TemCarro.

```
public class Carro {
    static int codigoCor;
    boolean status;
    String placa;
}

public class TemCarro {
  public static void main(String[] args) {
    // Declarações
      Carro auto1, auto2;      // usa outra classe Carro
      auto1 = new Carro();
      auto2 = new Carro();
    // Instruções
      auto1.codigoCor = 2;    // primeira atribuição
      auto1.status = true;
      auto1.placa = "BDB1020";
```

Arquivos físicos diferentes, sendo um para cada classe

Classes e Objetos

```
    // continuação do exemplo anterior
      System.out.println("Carro 1: " + auto1.placa +
         "\t" + auto1.codigoCor + "\t" + auto1.status);
      System.out.println("Carro 2: " + auto2.placa +
         "\t" + auto2.codigoCor + "\t" + auto2.status);
      auto2.codigoCor = 1;      // segunda atribuição
      auto2.status = true;
      auto2.placa = "JPX0770";
      System.out.println("\nCarro 1: " + auto1.placa +
         "\t" + auto1.codigoCor + "\t" + auto1.status);
      System.out.println("Carro 2: " + auto2.placa +
         "\t" + auto2.codigoCor + "\t" + auto2.status);
    }
}
```

→ após a primeira atribuição para `codigoCor` são apresentados os valores dos 2 carros iguais (2)

→ na segunda atribuição para `codigoCor` os 2 carros são alterados, pois este atributo é compartilhado pela classe

Classes e Objetos

Atributos de Referência ou de Valor

Estas características indicam se o conteúdo do atributo (variável) está sendo armazenado no endereço de memória alocado na declaração da variável realmente, ou se este endereço esta guardando outro endereço que indica onde tal conteúdo está na memória do computador.

VALOR

- armazena o **conteúdo** da variável no endereço que foi alocado para mesma
- variáveis de tipo primitivo guardam dados por valor

REFERÊNCIA

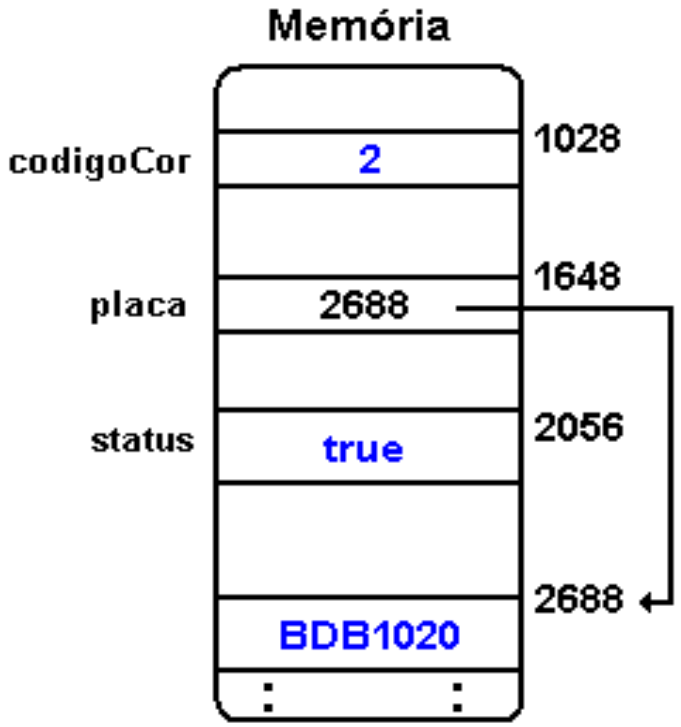
- armazena o **endereço** de memória onde o conteúdo desta variável está realmente armazenado na memória do computador
- geralmente, variáveis de objeto são de referência

Classes e Objetos

Observe a representação de alocação de memória para as declarações das 3 variáveis da classe Carro.

```
public class Carro {
   private int codigoCor = 2;
   private boolean status = true;
   private String placa = "BDB1020";  // objeto String
}
```

- codigoCor e status estão armazenando o próprio valor inicializado na variável
- placa é do tipo String, ou seja, um objeto na memória e esta guardando um outro endereço (referencia)
- conteúdo da placa está guardado em outro endereço de memória



Estrutura Homogênea - Array

Antes de prosseguir com o aprofundamento do estudo nas características das classes e seus objetos, torna-se imprescindível o estudo da estrutura de dados homogênea em Java disponível pela classe **Array**.

Assim, acesse a **Aula 4** para o estudo inicial deste conteúdo (**Array**) e após seu primeiro exercício ser concluído com êxito, esta aula (Aula 3) será retomada.

Estruturas Homogêneas

Classes e Objetos

Declaração de Métodos

A classe Carro mostrou como criar atributos (variáveis) em uma classe, sendo eles responsáveis pela identificação do **estado** dos objetos desta classe.

O **comportamento** de uma classe é definida pela implementação de seus métodos, que serão incorporados como exemplo nesta mesma classe Carro.

DEFINIÇÃO

Os métodos consistem em trechos de programa, com lógicas bem definidas, que recebem um identificador para serem acionados uma ou várias vezes, sempre que sua funcionalidade for necessária em sua classe original ou até em outras classes.

Classes e Objetos

Forma Genérica

<qualificador> <tipo retorno> <identificador>(argumentos);

qualificador de acesso ao método

tipo de dado de retorno do método

nome do método

lista de parâmetros

- <qualificador> indica como o método pode ser acessado por outras classes
- <tipo retorno> tipo de dado retornado pelo método, após seu processamento. Os métodos que não possuem retorno são sempre **void**
- **<identificador>** nome que identifica o método, sendo iniciado com caracter alfabético em minúsculo, por convenção
- **(argumentos)** parâmetros enviados ao método para processamento esperado e correto

Classes e Objetos

JAVA
(método)

- **Sem Retorno** - similar a procedimento
- Respeita a forma genérica de declaração, porém com tipo de retorno definido como void

- **Com Retorno** - como uma função
- Mantém a forma genérica de declaração, com tipo de retorno definido como um tipo de dado válido

→ Estes dois tipos de métodos permitem o reaproveitamento de código e facilitam a manutenção

→ Não existe em Java o conceito de método global

→ Observe o uso do método println da classe System

→ Imagine a elaboração de certo programa que precise do cálculo de juros em vários momentos (mais de cem) **e...**

Classes e Objetos

Suponha a classe Pessoa com 2 métodos: main() e limpaTela().

```
/** Síntese
 *    Objetivo: Preencher ficha cadastral de uma pessoa
 *    Entrada:  nome, idade e peso
 *    Saída:    confirmação do nome, idade e peso
 */
import java.util.Scanner;
import java.text.DecimalFormat;
 public class Pessoa {
  // Declarações globais
    private String nome;
    private int idade;
    private double peso;
  // Método main()
  public static void main(String[] args) {
   // Declarações
    Scanner ler = new Scanner(System.in);
    Pessoa p1 = new Pessoa();
    DecimalFormat forma = new DecimalFormat("00.00");
```

Classes e Objetos

```
// continuação do exemplo anterior
   // Instruções
    System.out.println("Informe seu nome: ");
    p1.nome = ler.nextLine();
    while (p1.idade <=  0) {
      System.out.println("Digite sua idade: ");
      p1.idade = ler.nextInt();
    }
    while (p1.peso <= 0) {
      System.out.println("Informe seu peso: ");
      p1.peso = ler.nextDouble();
    }
    limpaTela();  // aciona outro método desta classe
    System.out.println("Nome: " + p1.nome + "\tIdade:"+
         p1.idade + "\tPeso:" + forma.format(p1.peso));
}
// Método limpa tela (console)
public static void limpaTela() {
 // Instruções
   for(int aux=1;aux<=25;aux++)
     System.out.println();
}
}
```

Classes e Objetos

- Na classe Pessoa são criados 2 métodos e 3 atributos
- Nesta classe existe o método main(), o que torna esta classe executável
- Inicia a execução pelas instruções do método main() que aciona o método limpaTela() chamando-o pelo seu nome (identificador) e não envia nenhum parâmetro
- Suspende a execução do método acionador (main()) e inicia a execução do método limpaTela()
- Quando limpaTela() é encerrado o fluxo de execução retorna para o método acionador (main()), no ponto anteriormente suspenso (interrompido)
- O encerramento do main() termina a execução do programa, por ser este o método principal do programa elaborado

Classes e Objetos

Imagine esta mesma classe Pessoa com 3 métodos, sendo o main(), o limpaTela(), sem retorno de dados e o analise(), que recebe 2 parâmetros e retorna um único valor ao seu método acionador.

```
public static double analise(double pesos, int idades) {
   // Declarações
    double resultado;
   // Instruções
    resultado = pesos*2.5/idades;
    return(resultado);
}
```

- A instrução return encerra o bloco que estiver em execução imediatamente
- O método com retorno devolve um valor de retorno ao seu acionador, similar a uma função matemática

→ Inclua no método principal um condicional analisando se o retorno for maior que **5.23 o peso é Inadequado**.

Classes e Objetos

- A instrução **return** encerra um método imediatamente, após sua execução, além de retornar ao seu acionador **um único valor**, se este método não for do tipo **void**
- As seqüências de operações executadas usando métodos é a mesma, independente destes serem **com ou sem retorno**
- O cabeçalho ou declaração de um método também é chamado de **assinatura do método**
- Um método pode acionar outro método, que por sua vez também pode acionar um novo método, efetivando uma cascata entre todos eles
- Cada método pode ser acionado quantas vezes sejam necessárias para solução do problema em questão

Classes e Objetos

Propriedades das Variáveis (ou atributos)

- **Instância** – individual para cada objeto criado (instanciado) na memória do computador (por padrão)
- **Classe** – compartilha seu único valor com todos seus objetos e exige o uso do qualificador static
- **Valor** – armazena o dado no espaço exato da variável (guarda seu próprio conteúdo obtido em sua memória)
- **Referência** – armazena o endereço de memória onde o valor está guardado (comum para variáveis de objeto)

GLOBAL	**LOCAL**
- declarada fora de qualquer método, mas dentro de uma classe	- declarada dentro de um método ou bloco de instruções menor (auxiliar)

Classes e Objetos

Exemplos:

GLOBAL

```
public class Carro {
   private String placa;
   private int codigoCor;
   static booelan status;
}
```

placa: de instância global que referencia outra área de memória

codigoCor: de instância global de valor

status: de classe global de valor

LOCAL

```
static private double total;
for(int aux=1;aux<10;aux++)
{
   total = total+aux*0.35;
   System.out.println(total);
}
```

aux: de instância local de valor

total: de classe **global** de valor

Exercícios de Fixação

1) Solicite ao usuário o sexo válido de uma pessoa e acione o método adequado correspondente ao cálculo de seu peso ideal. Este peso é baseado também na altura de cada indivíduo, sendo para homens aplicada a fórmula pesoIdeal=72.7 * altura – 58 e para mulheres pesoIdeal = 62.1 * altura –44.7. O resultado deverá ser apresentado no método principal e sua solução deverá possuir um método de cálculo para cada sexo válido.

2) Faça um programa contendo um método que receba 3 notas de um aluno como parâmetros e uma letra. Se esta letra for A, o método deverá calcular a média aritmética destas notas e se for P o cálculo será de média ponderada, sendo seus pesos respectivamente definidos como 2, 3 e 5. A média desejada só será mostrada pelo principal e cada calculo de média terá seu próprio método definido, além do método de validação das notas entre 0 e 10.0.

Classes e Objetos

Qualificadores de Acesso

O controle da acessibilidade dos elementos de uma classe (**atributos e métodos**) é realizado por meio de seus qualificadores de acesso (não nas variáveis locais).

- **public**: acesso público, sendo este pelos métodos da própria classe e suas subclasses, além de qualquer outra classe que pertença a qualquer outro pacote
- **protected**: acesso protegido (somente pelos métodos da própria classe e de suas classes derivadas)
- **private**: qualificador mais restritivo, onde só podem ser acessados por métodos da própria classe
- **package**: qualificador não definido (padrão), podendo ser acessado por métodos da própria classe, suas classes derivadas (subclasses) ou qualquer outra classe definida no mesmo pacote

Classes e Objetos

Método Construtor

Operador new

A alocação de memória (instanciação) na criação de um novo objeto é realizada pelo simples operador **new**, que atribui valores padrões (default) para este novo objeto. Por exemplo:

```
auto1 = new Carro();     // constrói o objeto auto1
```

A declaração acima cria em memória um objeto proveniente da classe Carro (auto1) com seus valores sendo inicializados com valor padrão, caso não exista um **método construtor** nesta classe.

Classes e Objetos

Um método construtor consiste em um método definido em uma classe que é responsável por construir um objeto com determinados valores na memória.

- Método invocado pelo operador **new**
- O construtor pode conter chamadas para outros métodos, possibilitando a criação de objetos mais complexos
- Deve sempre possuir o nome **idêntico** ao de sua classe
- A ausência da declaração de um método construtor provoca a utilização de um construtor padrão, inicializado com valores padrões em Java
- O método construtor funciona como outro método qualquer, podendo receber parâmetros para que um novo objeto seja criado com sua inicialização
- Realiza a instanciação do objeto e seus valores

Classes e Objetos

Método Destrutor

- Responsáveis pela liberação da memória alocada por um método construtor padrão ou definido na classe
- "Devolve" ao sistema operacional os endereços alocados de memória para novo uso, evitando seu esgotamento

Observe o exemplo de alocação errada de memória.

```
// Declarações
  Carro auto1;  // usa classe Carro
  auto1 = new Carro("JPX0770“);
  auto1 = new Carro("BDB1020");
```

- 2 alocações (instanciações) foram feitas acima, porém para mesma variável, resultando na perda da primeira
- Processo automático de coleta de lixo na memória**:** sistema de tempo e marcação para possível destruição

Classes e Objetos

Suponha as classes abaixo e verifique a elaboração de um método construtor de Carro. A existência deste método não possibilita que o programa crie este objeto com valores padrões, como foi feito anteriormente.

```
public class Carro {     // classe de dados
  String placa;
  int codigoCor;
  boolean status;
  // Método construtor de Carro
  Carro (String registro, int cor, boolean situacao)
  {
    placa = registro;
    codigoCor = cor;
    status = situacao;
  }
}
```

Classes e Objetos

```
// continuação do exemplo anterior
 public class NovoCarro {
   public static void main(String[] args) {
      Carro auto1 = new Carro("DBX2008",3,true);
      System.out.println("Carro:" + auto1.placa +
         "\tCor:" + auto1.codigoCor + "\tSituação:"
                                    + auto1.status);
   }

 }
```

→ O método construtor deve ter o mesmo nome (identificador) da classe e nunca retorna valor (**void**)

→ O qualificador static em um método indica que o mesmo pode ser acionado, mesmo que seu objeto não seja criado

Classes e Objetos

Sobrecarga

A sobrecarga consiste na declaração de métodos com o mesmo identificador, porém realizando alguma forma de processamento diferente devido aos seus tipos de parâmetros e/ou retorno.

Por exemplo: até onde já foi estudado o símbolo + pode executar operações diferentes de acordo com seu contexto, ou seja, seus valores serão analisados para sua funcionalidade correta ser aplicada (executada).

2 + 3 → realiza operação de adição entre os valores

"Ciência" + "Computação" → efetiva a concatenação

Neste exemplo se pode concluir que o operador + é sobrecarregado (mesmo símbolo usado em diferentes operações)

Classes e Objetos

Normalmente, a sobrecarga em métodos acontece sobre diferentes tipos ou quantidades de parâmetros definidos em vários métodos com o mesmo nome (identificador), podendo seu retorno também ser diferente e possibilitar a sobrecarga de um método.

```
/** Síntese
 *    Objetivo: calcular a área de uma figura
 *    Entrada:  quantidade de lados e seus tamanhos
 *    Saída:    área
 */
import java.util.Scanner;
public class Sobrecarga {
  public static void main(String[] args) {
  // Declarações
   byte quantidadeLados;
   Scanner ler = new Scanner(System.in);
```

Classes e Objetos

```
// continuação do exemplo anterior

 // Instruções
  System.out.print("Qual o número de lados (1-3):");
  do {
      quantidadeLados = ler.nextByte();
      if(quantidadeLados < 1 || quantidadeLados >3) {
        System.out.print("\nValor Inválido. Informe"+
                                          " novamente: ");
         quantidadeLados = ler.nextByte();
      }
  } while (quantidadeLados<1 || quantidadeLados>3);
  switch(quantidadeLados) {
    case 1 : {
     int lado1;  // declaração de variável
     System.out.println("Informe o lado: ");
     lado1 = ler.nextInt();
     System.out.print("\nQuadrado = " + area(lado1));
     break;
    }
```

Classes e Objetos

```
// continuação do exemplo anterior
    case 2 : {
      int lado1,lado2;  // declaração de variáveis
      System.out.println("Informe os 2 lados: ");
      lado1 = ler.nextInt();
      lado2 = ler.nextInt();
      System.out.print("\nRetangulo = " +
                                        area(lado1,lado2));
      break;
    }
    default : {
      int lado1,lado2,lado3;  // declaração variáveis
      System.out.println("Informe os 3 lados: ");
      lado1 = ler.nextInt();
      lado2 = ler.nextInt();
      lado3 = ler.nextInt();
      System.out.print("\n\nCubo = " +
                                   area(lado1,lado2,lado3));
      break;
    }
  }
}
```

Classes e Objetos

```
// finalizando o exemplo anterior

 // Métodos (sobrecarga em area)
  public static long area(int lado) {
     return (lado * lado);
  }

  public static long area(int lado1, int lado2) {
    return (lado1 * lado2);
  }

  public static long area(int lado1,int lado2,int lado3) {
    return (lado1 * lado2 * lado3);
  }

} // encerra a classe
```

→ Observe a sobrecarga no método area(), existindo três métodos diferentes, porém com parâmetros distintos.

Classes e Objetos

Acesso a Métodos de Outras Classes

- É possível acessar um método elaborado por uma classe diferente daquela que esta em execução e sendo este método de outra classe não pertencente a essência da linguagem Java como o println na java.lang
- Qualquer método pode ser usado pelas classes elaboradas na construção de um programa, porém estas classes devem estar armazenadas no mesmo diretório (pacote), que deverá estar configurado corretamente em seu ambiente de execução (variável de ambiente classpath)

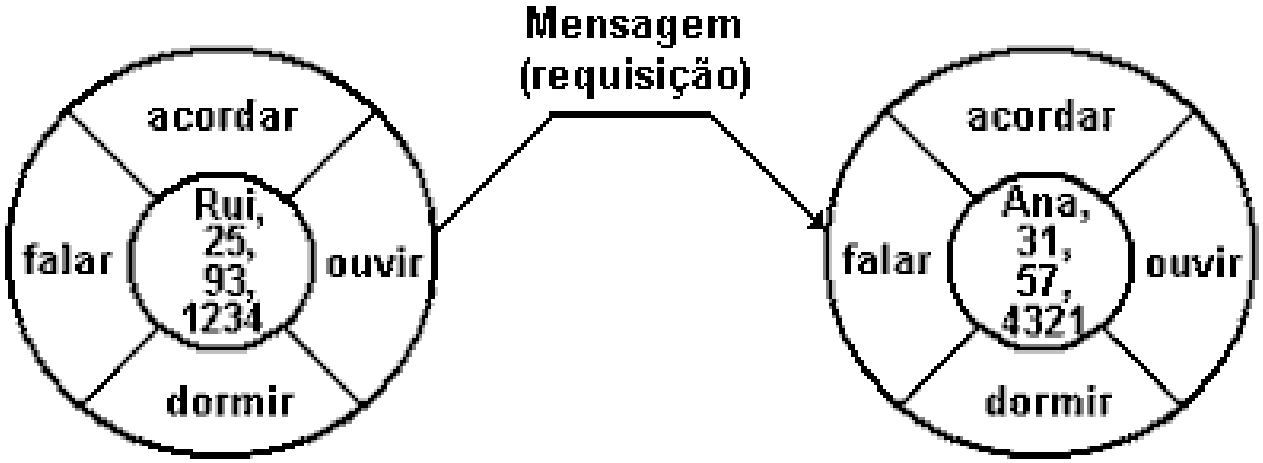
  <nomeClasse>.<nomeMétodo>

- O método é criado uma única vez em uma classe e todas as demais usam o mesmo método

Classes e Objetos

Mensagens

Os diferentes objetos interagem através de requisições (mensagens) trocadas entre eles, onde métodos são acionados para o acesso aos dados associados (atributos).



- Enviar uma mensagem significa executar um método
- As mensagens são compostas por 3 partes: objeto destinatário; identificador do método a ser acionado; parâmetro recebidos pelo método

Classes e Objetos

Acompanhe este exemplo e verifique uma situação de acesso a outras classes com uso de seus métodos.

```
/** Síntese
 *    Atributos:
 *    Métodos: limpaTela(), ValidaIdade(int)
 */
public class VariosMetodos {     // classe de serviços
  public static void limpaTela() {
   // Instruções
    for(int aux=1;aux<=25;aux++)
       System.out.println();
  }

  public static boolean validaIdade(int idade) {
   // Instruções
    if(idade > 0 && idade <= 150)
      return false;
    else
      return true;
  }
}
```

Classes e Objetos

```
/** Síntese
 *    Objetivo: armazena uma idade válida de pessoa
 *    Entrada:  idade
 *    Saída:    confirmação da idade válida armazenada
 */
import java.util.Scanner;
public class VerificaIdade {
  public static void main(String[] args) {
    // Declarações
     int idade = 0;
     Scanner ler = new Scanner(System.in);
    // Instruções
     do {
         System.out.println("Informe sua idade: ");
         idade = ler.nextInt();
     } while(VariosMetodos.validaIdade(idade));
     VariosMetodos.limpaTela();
     System.out.println("Sucesso no registro de " +
                              idade + " anos de idade.");
  }
}
```

Classes e Objetos

Atente a classe VariosMetodos que não é executável, pois não possui método main(). Apesar desta classe possuir vários métodos em sua definição, ela só pode ser executada a partir do acesso (acionamento) de outras classes.

Esta característica de uso de várias classes para solução de problema computacional é muito comum na linguagem Java e permite a organização de métodos e funcionalidades como "**Serviços**" que estão disponíveis para o uso em seu programa

Classes e Objetos

Uma orientação inicial para compreensão do problema e criação de classes/objetos coerentes com uma possível solução em Programação Orientada a Objeto (POO) seria:

1. Sempre mantenha os dados (atributos) protegidos, por meio do qualificador de acesso adequado (privado ou protegido)
2. Inicialize sempre suas variáveis, pois as variáveis de instância são inicializadas sempre, mas as locais não, e podem ocasionar erros no programa
3. Não use tipos de dados primitivos em demasia (exemplo do endereço residencial que poderia ser uma nova classe com o conjunto de dados que o forma)

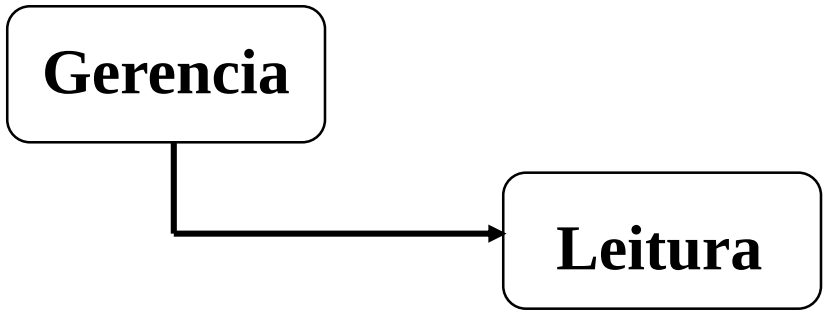
Classes e Objetos

4. Nem todos os atributos precisam de assessores (*get*) e modificadores (*set*) – (exemplo data de contratação que pode ser lida do computador)
5. Padronize a definição dos dados (públicos, pacote, etc.)
6. Divida classes com muitas tarefas, mas sem exagerar (exemplo do jogo de cartas)
7. Identificadores significativos, com substantivos para classes e *get* e *set* para os métodos assessores e modificadores
8. Procure priorizar o uso de objetos de classes, evitando o trabalho com métodos estáticos (*static*)

Classes e Objetos – corrigindo a leitura

Uma dificuldade na leitura de dados usando a classe Scanner esta vinculada a manipulação do ***buffer*** de memória na coleta de cadeia de caracteres que podem possuir espaços em branco como seu conteúdo, por exemplo: nome completo de pessoa, frases, etc.

Uma maneira alternativa de superar este problema na memória temporária (*buffer*) poderia ser a elaboração de uma classe específica para leitura dos diversos tipos de dados que sejam interessante para sua aplicação, senda esta classe novos "**Serviços**" disponíveis num programa.



Classes e Objetos – corrigindo a leitura

```
/** Síntese
 *   Objetivo: gerenciar leitura de dados sem problema
 *             de buffer (memória temporária)
 *   Entrada:  valores distintos para teste
 *   Saída: cada um dos valores lidas e armazenados
 */
 public class Gerencia {
  public static void main(String[] args) {
    char opcao;
    do {
       System.out.print("Informe uma string: ");
       System.out.println("\n\t" +
                                  MeuScanner.getString());
       System.out.print("\n\nDigite valor inteiro: ");
       System.out.println("\n\t" +
                                  MeuScanner.getInt());
       System.out.print("\n\nDigite um double(com ,): ");
       System.out.println("\n\t" +
                                  MeuScanner.getDouble());
```

Classes e Objetos – corrigindo a leitura

```
// continuação do exemplo anterior
      System.out.print("\n\nDigite S para continuar: ");
      opcao = MeuScanner.getString().charAt(0);
   } while ((opcao == 's') || (opcao == 'S'));
   System.out.println("\n\nPrograma Encerrado.");
 }
}
```

//

```
/** Síntese
 *    Atributos:
 *    Métodos: getInt(), getDouble(), getShort(),
 *             getString()
 */
import java.util.Scanner;
public class MeuScanner {
   public static int getInt(){
      Scanner ler = new Scanner(System.in);
      return ler.nextInt();
   }
```

Classes e Objetos – corrigindo a leitura

```
// continuação do exemplo anterior
    public static double getDouble(){
        Scanner ler = new Scanner(System.in);
        return ler.nextDouble();
    }
    public static short getShort(){
        Scanner ler = new Scanner(System.in);
        return ler.nextShort();
    }
    public static String getString(){
        Scanner ler = new Scanner(System.in);
        return ler.nextLine(); // sem problema com buffer
    }
}
```

Exercícios de Fixação

3) Elabore um programa que crie uma classe de dados que definirá os atributos: nome (para guardar o nome completo de uma pessoa); idade (para guardar a idade de uma pessoa); altura (para guardar a altura em metros de uma pessoa. Uma outra classe (Leitura) deverá ser responsável por todos os tipos de leituras, só dos tipos de dados necessários a este problema, enquanto a classe Validacao será responsável pela validação de cada dado de entrada, conforme abaixo:

    a) nome tem que ser no mínimo de 3 caracteres;

    b) idade tem que ser maior que zero e menor que 150;

    c) altura tem que ser maior que 0.4 e menor que 2.6 metros

Sua solução deve cadastrar pessoas enquanto o usuário quiser, validando o desejo da repetição por meio do método ***validaContinua*** que também estará disponível na classe de Validacao. O método ***mostraDados***, que só apresenta na console um registro que tenha acabado de acontecer e outros métodos interessante a sua solução modularizada deverão estar disponíveis em outra classe denominada Servicos.

Exercícios de Fixação

4) Uma loja de entretenimento fornece desconto (50%) para pessoas do sexo feminino na participação de seus eventos, enquanto que para os homens o benefício oferecido é o desconto (10%) sobre a bebida consumida durante estes eventos. Faça um programa que cadastre o CPF, Nome e Sexo de uma pessoa em um destes eventos e acompanhe seus gastos até que o mesmo informe que deixou o evento e irá pagar sua conta. Solicite os dados pessoais do usuário e os armazene em variáveis globais de instanciação na classe Cliente (similar a struct em C). Em outra classe, denominada Validacao deverão existir os métodos que validam o cadastro do sexo do participante e seu CPF, que não poderá ser menor que cem mil e nem maior que 9999999999. Enquanto o usuário do programa estiver no evento ele poderá consumir bebidas, além de pagar seu ingresso. Quando este informar que vai embora seu sistema deve apresentar o valor final de sua conta no método principal. O valor do ingresso (sem desconto) deve ser solicitado no início do programa e o preço de qualquer bebida deverá ser definida como uma constante Java de R$2,00 no principal. Este problema exige a criação do método construtor Cliente sem o uso dos valores padrões da linguagem Java para sua inicialização.

Exercícios de Fixação

5) Faça um programa que armazene o valor do expoente inteiro e da base real de forma que você possa desenvolver sua codificação específica para o cálculo de um valor (base) elevado ao expoente informado por seu usuário. Em sua solução deverão existir somente 2 classes: **Calculo** que instancia a base e o expoente, sendo construído seu objeto com valores padrões em Java; e **Console** que possuirá 2 métodos (***limpaTela*** recebendo como parâmetro um valor do tipo byte que define a quantidade de saltos de linha, além do ***mostraResultado*** que deverá somente apresentar uma mensagem e o resultado deste cálculo centralizado, horizontalmente, em uma linha). A classe **Calculo** será executável e deverá sobrecarregar o método **pontencia**, tendo 2 parâmetros (base e expoente) quando expoente for positivo ou zero ou 3 parâmetros (base, expoente e 1) quando expoente for negativo. Realize os cálculos da potencia enquanto o usuário desejar e todos os métodos da classe **Calculo** não poderão ser estáticos (ter qualificador **static**).

Exercícios de Fixação

6) No intuito de colaborar no processo de avaliação de uma disciplina, faça um programa que coletará a matrícula e o nome de cada aluno informado pelo professor e suas notas de a 0 até 10 pontos. Esse programa terá pelo menos duas classes, onde na de dados **Aluno** serão definidos só os atributos instanciados para cada aluno e o código, do tipo short, para disciplina que este professor leciona. Este código será solicitado no início do programa e será armazenado em um atributo pertencente a classe, compartilhando seu valor entre todos os alunos cadastrados. Como cada aluno pode ter feito de 1 até 4 avaliações (provas), sua solução deverá SOBRECARREGAR o método ***fazMediaFinal*** como:
   a) 1 prova: só aciona outro método (***statusAluno***) sem retorno que mostre a situação de **aprovado** (nota>=7) ou **reprovado** (nota<7);
   b) 2 provas: realizar a média ponderada de peso 4 e peso 6, respectivamente, para cada nota, e mostrar a média e a situação do aluno (***statusAluno***);
   c) 3 provas: encontra a maior das 3 notas e apresenta-la como sua média final, além de situação do aluno pelo acionamento do método ***statusAluno***;
   d) 4 provas: média aritmética das 4 notas e a situação (***statusAluno***).

   A quantidade de avaliações efetuadas por cada aluno deve ser validada por um método especifico, denominado ***validaQuantidade***. A leitura de dados de alunos deverá ser feita até o professor informar que não tem mais alunos, sendo a análise de cada um apresentada imediatamente após o cadastro de cada aluno, mostrando o código da disciplina, nome e matrícula do aluno, sua média final, formatada com 2 casas inteiras e 2 fracionárias, além da situação **Aprovado** ou **Reprovado**. Sua solução NÃO poderá possui métodos estáticos na classe de dados.

Referência de Criação e Apoio ao Estudo

Material para Consulta e Apoio ao Conteúdo

- HORSTMANN, C. S., CORNELL, G., Core Java2 , volume 1, Makron Books, 2001.
  - Capítulo 4
- FURGERI, S., Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações, São Paulo: Érica, 372 p., 2002.
  - Capítulo 7
- Universidade de Brasília (UnB Gama)
  - http://cae.ucb.br/conteudo/unbfga

  (escolha a disciplina **Orientação a Objetos** no menu superior)