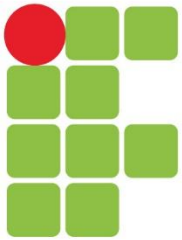


INSTITUTO FEDERAL
SÃO PAULO
Câmpus Araraquara

Disciplina de Programação Orientada a Objetos
Pacotes, constantes e membros estáticos
3º Semestre - Curso Superior em ADS

Prof. Msc. Dênis L. Zaniro
deniszaniro@ifsp.edu.br

Sumário

- Objetivos
- Criação de pacotes
- Definição de constantes
- Definição de membros estáticos
- Exercícios de fixação
- Referências bibliográficas

Objetivos

- Entender os benefícios oriundos da criação de pacotes.
- Entender o significado de constantes e membros estáticos e saber utilizá-los para resolver certos tipos de problemas.

Criação de pacotes

- Problema: Crie uma aplicação que permita gerenciar empregados (e seus dependentes) em uma empresa (cadastros, cálculos de aumento salarial, geração de relatórios, etc.). Essa aplicação deve fornecer uma interface gráfica, e deve possuir conexão com um banco de dados para que diversas operações sejam realizadas.
- Sob a perspectiva estrutural dessa aplicação, quais seriam algumas de suas classes?

Criação de pacotes

- Algumas classes seriam:
  - Classe **Empregado** (objetos empregados).
  - Classe **Dependente** (objetos dependentes).
  - Classe **GerenciaEmpregados** (conjuntos de empregados).
  - Classe **TelaPrincipal** (janela principal da aplicação).
  - Classe **TelaEmpregados** (tela de empregados).
  - Classe **ConexaoBD** (para encapsular todas as instruções para a conexão com o banco de dados).
  - Classe **EmpregadoBD** (para encapsular todas as operações a serem executadas no banco de dados).

Criação de pacotes

- Hipótese:
  - Todas essas classes devem ser armazenadas em um único lugar para que sejam distribuídas.
  - Há problemas nessa abordagem?
  - A resposta é sim. Observa-se que há um número razoável de classes com responsabilidades distintas.
- Um dos problemas é a falta de organização geral desse projeto.

Criação de pacotes

- Outro problema é que a aplicação pode ser expandida e pode haver diversos desenvolvedores que participem do projeto na organização.
- Dessa forma, existe a possibilidade de que duas ou mais classes sejam criadas com o mesmo nome.
- Então, como resolver esses problemas?
  - A partir da **criação de pacotes**.

Criação de pacotes

- O que são pacotes?
  - Pacotes permitem armazenar e organizar classes em uma estrutura igual à estrutura de diretórios.
  - Todas as classes da API Java são organizadas em pacotes.
  - Normalmente, as classes são agrupadas em cada pacote de acordo com suas responsabilidades.
  - Para indicar que uma classe pertence a um dado pacote, deve-se usar a instrução **package**.

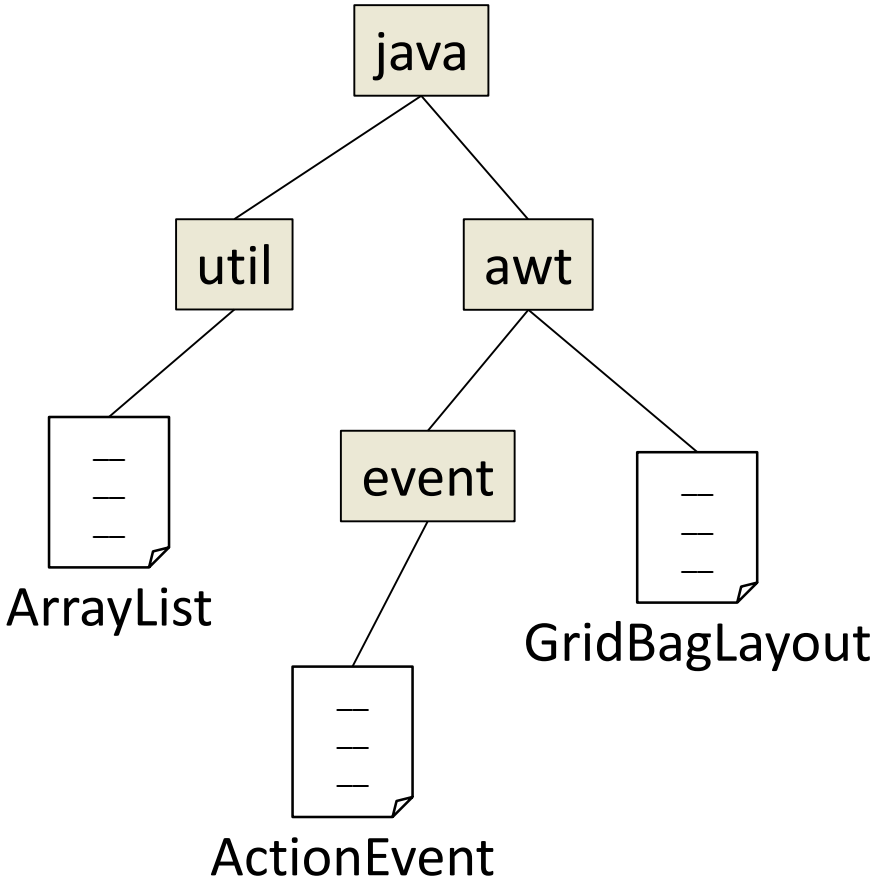
Criação de pacotes

- Ilustração:

```
package java.util;
...
public class ArrayList ... { ... }
```

```
package java.awt;
...
public class GridBagLayout ... { ... }
```

```
package java.awt.event;
...
public class ActionEvent ... { ... }
```



Criação de pacotes

- Quando uma classe é armazenada em um pacote, deve-se considerar seu nome completamente qualificado.
- Exemplo:
  - Para a classe ArrayList, seu nome completamente qualificado é: java.util.ArrayList.
- Portanto, além de melhorar a organização geral do projeto, pacotes permitem evitar conflitos de nomes.

Criação de pacotes

- Mas o que acontece se dois desenvolvedores criarem pacotes idênticos?
  - Esse risco é evitado seguindo-se convenções para nomear pacotes.
  - Uma convenção é nomear pacotes utilizando-se o nome do domínio invertido de uma organização.
  - Por exemplo:
    - Para o domínio **ifsp.edu.br**, temos, como exemplos:
      - br.edu.ifsp.telas, br.edu.ifsp.util, br.edu.ifsp.entidades, br.edu.ifsp.entidades.alunos, etc.

Criação de pacotes

- Analise o código dado a seguir:

```
package br.edu.ifsp.teste1;
...
public class Teste { ... }
```

```
package br.edu.ifsp.teste2;
...
public class Teste { ... }
```

```
package br.edu.ifsp.principal;

import br.edu.ifsp.teste1.Teste;   X
import br.edu.ifsp.teste2.Teste;

public class Principal
{
   public static void main(String args[]) {
      Teste teste = new Teste();
   }
}
```

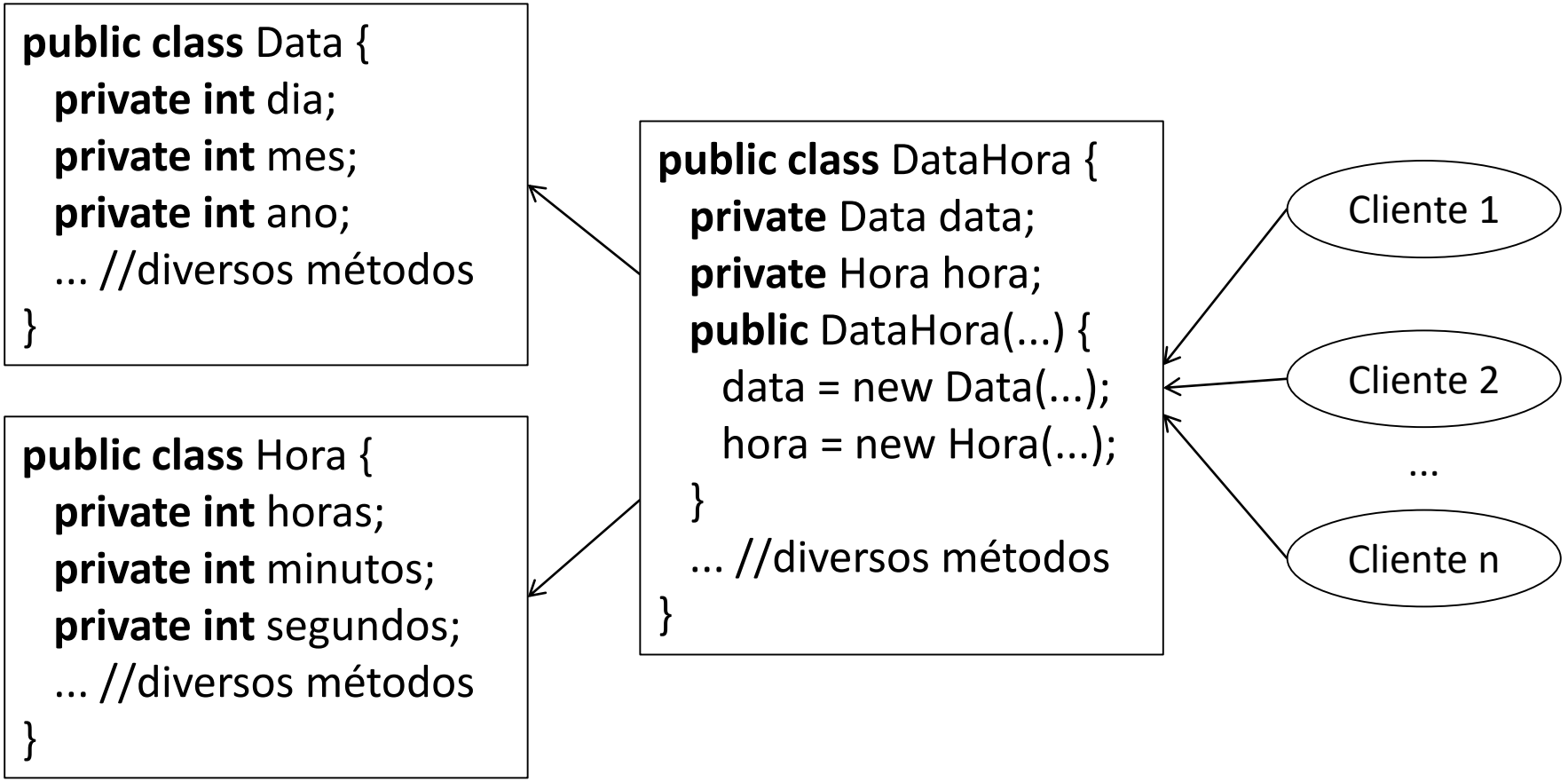
O problema desse código é que haverá conflito entre as instruções **import** no arquivo da classe Principal.

Criação de pacotes

- Há mais algum motivo para criar pacotes?
  - Há outro motivo que é importante para a criação de pacotes.
- Suponha que seja necessário criar um conjunto simples de classes que forneça operações com datas e horas. Portanto, qualquer outra classe que necessite trabalhar com datas poderá reutilizar esse conjunto de classes.
- Analisando-se os requisitos, três classes foram definidas: **Data**, **Hora** e **DataHora**.

Criação de pacotes

- Ilustração:



Criação de pacotes

- De acordo com a ilustração, das três classes, a única que deve disponibilizar às classes clientes as operações com datas é a classe DataHora.
- Nesse momento, surge a seguinte questão:
- Como “esconder” as classes Data e Hora das classes clientes?
  - A solução envolve criar pacotes para as três classes, e especificar que as classes Data e Hora devem ser visíveis apenas em nível de pacote.

Criação de pacotes

- Esse é o momento para revisar os modificadores de visibilidade estudados até o momento.
  - Modificador ***public***: membros *public* podem ser enxergados pela classe que os contém e por todas as outras classes, independentemente de seu pacote.
  - Modificador ***private***: membros *private* podem ser enxergados apenas pela classe que os contém.
    - Para refletir: Uma classe poderia ser declarada como *private*?

Criação de pacotes

- Modificador ***default***: membros *default* podem ser enxergados pela classe que os contém e também pelas classes que pertencem ao pacote da classe em questão.

- Como se declara um membro como *default*?
  - Para declarar um elemento (classe, atributo ou método) como *default*, é necessário apenas não especificar outro modificador (*private*, *public* ou *protected*).

Criação de pacotes

- Solução:

```
package br.edu.ifsp.datas;
class Data {
  ...
}
```

```
package br.edu.ifsp.datas;
class Hora {
  ...
}
```

```
package br.edu.ifsp.datas;
public class DataHora {
  ...
}
```

```
package br.edu.ifsp.principal;
import br.edu.ifsp.datas.*;
public class Principal
{
  public static void main(String args[]) {
    DataHora dh;
    Data d;   X
    Hora h;   X
  }
}
```

Criação de pacotes

- Analise o código dado a seguir:

```
package abc;
public class X {
  X() {
    ...
  }
}
```

```
package def;
import abc.X;
public class Y {
  public Y() {
    X x;
    x = new X();  X
  }
}
```

```
package abc;
import def.Y;
public class Z {
  Z() {
    Y y;
    y = new Y();
  }
}
```

Criação de pacotes

- Faça o mesmo para o código dado a seguir:

```
package abc;
public class X {
  private X() {
    ...
  }
  public X(int v) {
    ...
  }
}
```

```
package def;
import abc.X;
public class Y {
  Y() {
    X x;
    x = new X(2);  OK
  }
}
```

Definição de constantes

- Problema: Escreva uma classe que encapsule um *array* de *strings*. O tamanho máximo desse *array* é 50. Uma vez que esse tamanho tenha sido estabelecido, seu valor não poderá mais ser modificado em qualquer outro lugar do código.
- Hipóteses:
  - Hipótese 1: Utilizar o valor 50 (literal) em todos os pontos do código que necessitem desse valor.
  - Problemas?

Definição de constantes

- Hipótese 2: Armazenar o valor 50 em uma variável (atributo).
- Ainda há problemas?
- **Hipótese 3:** Usar uma **constante** para armazenar o valor 50.

- O que são constantes?
  - Constantes permitem armazenar valores em nomes de tal forma que se garanta que, uma vez definido o valor em um ponto do código, ele não poderá mais ser modificado em outros pontos.

Definição de constantes

- Em Java, constantes são declaradas usando-se o termo **final**.
- Tanto dados de instância (atributos) quanto dados locais podem ser armazenados em constantes.
- Por convenção, nomes de constantes são escritos em letras maiúsculas. Para separar uma palavra da outra, deve-se usar o caractere '_'. Exemplos: QUANTIDADE_MAXIMA, LIM_MIN, etc.

Definição de constantes

- Solução:

```
public class GerenciaArray
{
   private String[] palavras;
   private int cont;
   private final int MAX = 50;
   public GerenciaArray() {
      palavras = new String[MAX];
   }
   public void adicionaPalavra(String palavra) {
      if(cont < MAX)
          palavras[cont++] = palavra;
   }
}
```

Definição de constantes

- Reflexão:
  - De acordo com a solução anterior, percebe-se que, para todo objeto do tipo GerenciaArray, será criado um *array* de tamanho 50.
  - E se a classe GerenciaArray tivesse que fornecer condições para permitir criar *arrays* de tamanhos diferentes?
  - Essa solução seria adequada?
    - A resposta é não.

Definição de constantes

- Nova solução:

```
public class GerenciaArray
{
   private String[] palavras;
   private int cont;
   private final int MAX;
   public GerenciaArray(int max) {
      this.MAX = max;
      palavras = new String[MAX];
   }
   ...
}
```

Definição de constantes

- Mais reflexão:
  - A nova solução permite que classes clientes criem *arrays* cujo tamanho máximo esteja de acordo com as suas necessidades.
  - Entretanto, alguns clientes gostariam de criar *arrays* de uma forma tal que não seja necessário informar um valor de tamanho.
  - Portanto, precisamos de mais flexibilidade.

Definição de constantes

- Nova solução:

```
public class GerenciaArray
{
  private String[] palavras;
  private final int MAX;
  public GerenciaArray() {
    this.MAX = 50;
    palavras = new String[MAX];
  }
  public GerenciaArray(int max) {
    this.MAX = max;
    palavras = new String[MAX];
  }
  ...
}
```

```
public class GerenciaArray
{
  private String[] palavras;
  private final int MAX;
  public GerenciaArray() {
    this(50);
  }
  public GerenciaArray(int max) {
    this.MAX = max;
    palavras = new String[MAX];
  }
  ...
}
```

Definição de membros estáticos

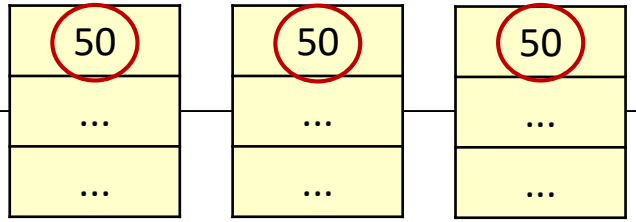
- Vamos supor que a classe GerenciaArray deva ser reutilizada em um novo projeto.
- Nesse contexto, não se deseja permitir que *arrays* sejam instanciados a partir de qualquer tamanho.
- Em outras palavras, o tamanho de todos os *arrays* deve ser fixo (por exemplo, tamanho 50).
- Hipótese:
  - Já resolvemos esse problema (primeira solução). A solução é atribuir um valor para uma constante em sua declaração.

Definição de membros estáticos

- De fato, essa seria uma solução. Mas ela não poderia ser melhorada?
  - Analisemos o código mais uma vez:

```
public class GerenciaArray
{
  private String[] palavras;
  private int cont;
  private final int MAX = 50;
  public GerenciaArray() {
    palavras = new String[MAX];
  }
  ...
}
```

```
public class Principal
{
  public static void main(String args[]) {
    GerenciaArray g1, g2, g3;
    g1 = new GerenciaArray();
    g2 = new GerenciaArray();
    g3 = new GerenciaArray();
  }
}
```



Definição de membros estáticos

- Se, nesse novo contexto, todos os *arrays* devem possuir o mesmo tamanho, por que estamos armazenando o tamanho de cada *array* de forma repetitiva?
- A solução para isso é armazenar o tamanho de cada *array* em um atributo estático.
- O que são atributos estáticos?
  - Atributos estáticos permitem armazenar valores que se apliquem a todas as instâncias de uma dada classe.

Definição de membros estáticos

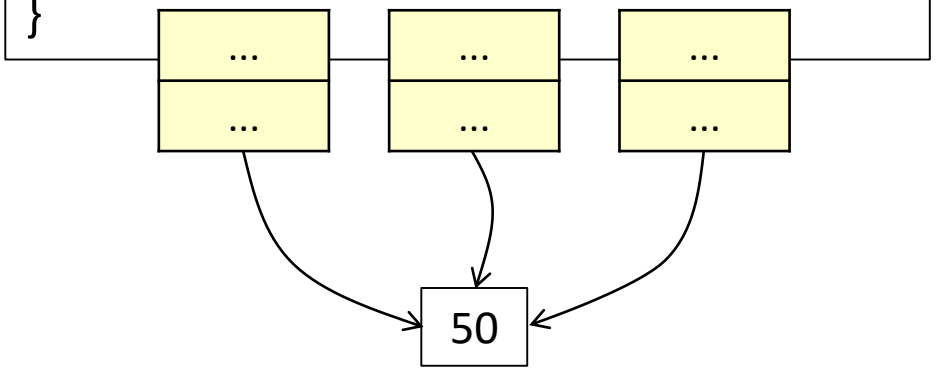
- Atributos estáticos são conhecidos como **variáveis de classe**.
- O valor armazenado por uma variável de classe é enxergado por todas as instâncias daquela classe.
- Observe, nesse ponto, a diferença fundamental entre variáveis de classe e variáveis de instância.
- Membros estáticos (atributos ou métodos) são definidos usando-se o termo ***static***.

Definição de membros estáticos

- Nova solução:

```
public class GerenciaArray
{
  private String[] palavras;
  private int cont;
  private static int max = 50;
  public GerenciaArray() {
    palavras = new String[max];
  }
  ...
}
```

```
public class Principal
{
  public static void main(String args[]) {
    GerenciaArray g1, g2, g3;
    g1 = new GerenciaArray();
    g2 = new GerenciaArray();
    g3 = new GerenciaArray();
  }
}
```



Definição de membros estáticos

- Há algo errado com o código anterior.
- Ao se definir o tamanho máximo do *array* em um membro estático, esse valor poderá ser alterado em qualquer parte do código.
- A solução é transformar a variável estática *max* em uma constante estática.
- Declaração de uma constante estática:
  - **private static final int** MAX = 50;

Definição de membros estáticos

- Há alguma implicação em se definir um membro que seja estático e, ao mesmo tempo, constante?
  - A implicação é que o valor definido em um membro estático e constante será enxergado por todas as instâncias de uma classe, e não poderá ser alterado.
  - Portanto, o valor de uma constante estática deve ser definido obrigatoriamente em sua declaração.
  - Normalmente constantes estáticas são declaradas como públicas.

Definição de membros estáticos

- Definição e uso de um membro estático:

```
public class GerenciaArray
{
   private String[] palavras;
   private int cont;
   public final static int MAX = 50;
   public GerenciaArray() {
      palavras = new String[MAX];
   }
   ...
}
```

```
public class Principal
{
   public static void main(String args[]) {
      int tam;
      tam = GerenciaArray.MAX;
      //armazena em tam o valor de MAX
   }
}
```

Definição de membros estáticos

- Prática 1: Crie uma classe chamada *Saida* que permita imprimir dados na tela usando a classe *System* ou a classe *JOptionPane*. Portanto, essa classe deve possuir um método que receba, como parâmetros, uma *string* (a ser impressa) e a informação sobre a classe a ser usada para realizar a impressão (*JOptionPane* ou *System*). Suponha que a classe *JOptionPane* corresponda à opção 1 e a classe *System* corresponda à opção 2. Crie a classe principal do programa para testar a classe *Saida*. Defina uma estratégia para que essa classe não precise informar o valor 1 ou o valor 2 para realizar a impressão, mas apenas um nome sugestivo que indique qual será a classe de impressão. Ao final de sua execução, o método deverá retornar um valor indicando se a impressão foi efetuada ou não.

Definição de membros estáticos

- Observamos que, caso exista um membro estático público, seu valor poderá ser acessado de forma global usando-se o próprio nome da classe.
- Mas, e se o membro estático for uma variável que esteja protegida pela classe que a contém?
  - Nesse caso, será necessário criar métodos que permitam modificar e acessar seu valor.
  - Portanto, os métodos também devem ser estáticos.

Definição de membros estáticos

- Definição e chamada de um método estático:

```
public class Pessoa
{
   private static int total;
   public static void setTotal(int total) {
      Pessoa.total = total;
   }
   public static int getTotal() {
      return total;
   }
   ...
}
```

```
public class Principal
{
   public static void main(String args[]) {
      Pessoa.setTotal(10);
      int total = Pessoa.getTotal(); //10
   }
}
```

Definição de membros estáticos

- Quer dizer que métodos estáticos somente são criados para permitir manipular variáveis que sejam estáticas?
  - A resposta é não.
- Métodos estáticos devem ser criados quando a sua tarefa não depende de dados de instâncias específicas.
- Observe que métodos estáticos somente podem acessar outros membros estáticos.

Definição de membros estáticos

- Suponha que seja necessário criar uma classe que calcule o fatorial de um valor qualquer informado por classes clientes.
- A única necessidade dessas classes é o resultado do cálculo, que é apenas um valor.
- Portanto, seria possível criar um método estático que receba, como parâmetro, um valor, e calcule e retorne o fatorial para esse valor.
- Como exemplo, a classe Math da API Java possui diversos métodos estáticos.

Definição de membros estáticos

- Analise o código dado a seguir:

```
public class Pessoa
{
  private String nome;
  private int idade;
  private static int total = 50;
  public static int getTotal() {
    return total;
  }
  public static void setTotal(int total) {
    Pessoa.total = total;
  }
  ...
}
```

```
public class Principal
{
  public static void main(String args[]) {
     Pessoa p1, p2;
     p1 = new Pessoa();
     int total = Pessoa.getTotal();
     p1.setTotal(10);
     total = Pessoa.getTotal();
     p2 = null;
     p2.setTotal(15);
     total = Pessoa.getTotal();
  }
}
```

Exercícios de fixação

- Escreva um programa que permita a realização de alguns serviços relacionados a pessoas e endereços. Toda pessoa possui nome, idade, endereço residencial e endereço comercial. Apenas o endereço comercial não é obrigatório durante a instanciação de uma pessoa, mas deve-se considerar a possibilidade de que ele seja informado nesse momento. Endereços são compostos de número, logradouro e bairro. Ao ser executado, o programa deve instanciar duas pessoas, e exibir uma mensagem informando se as pessoas são iguais ou diferentes. Nesse contexto, pessoas iguais são aquelas cujos valores de seus atributos são iguais entre si. Caso as pessoas sejam diferentes, o programa deve imprimir os dados da pessoa mais velha ou informar que as pessoas possuem a mesma idade. As classes clientes não devem saber da existência da classe de endereços. Forneça os serviços da forma mais flexível possível.

Exercícios de fixação

- Escreva um programa que gerencie pessoas. Toda pessoa deve possuir nome e código. O nome é um dado que deve ser informado obrigatoriamente durante a instanciação de uma pessoa. Já o código de uma pessoa deve ser um número inteiro sequencial gerado pelo próprio programa. Assim, a primeira pessoa criada no programa deve possuir código igual a 1, a segunda pessoa deve possuir código igual a 2, e assim por diante. Além da classe de pessoas, crie uma classe que permita instanciar três pessoas e imprimir o nome e o código de todas elas.

Exercícios de fixação

- **Desafio**: Com base na solução anterior, considere que, além de nome e código, toda pessoa possui idade. A idade deve ser um valor positivo e, para ser válido, o nome deve ser um valor diferente de *null* e deve ser diferente de vazio. Portanto, nomes como “”, “ “, e assim por diante, não devem ser aceitos como valores válidos. Para validar nomes, use dois métodos da classe String: trim() e isEmpty(). Nesse contexto, uma pessoa somente deve ser instanciada se seus dados forem válidos. Portanto, defina uma estratégia que controle a instanciação de objetos pessoas nesse programa.

Referências bibliográficas

- SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Elsevier, 2003.
- SIERRA, Kathy; BATES, Bert. **Use a cabeça! Java**. 2. ed. Alta Books, 2009.