ORIENTAÇÃO A OBJETOS

AULA 5

Encapsulamento e Estrutura Homogênea Dinâmica

Vandor Roberto Vilardi Rissoli

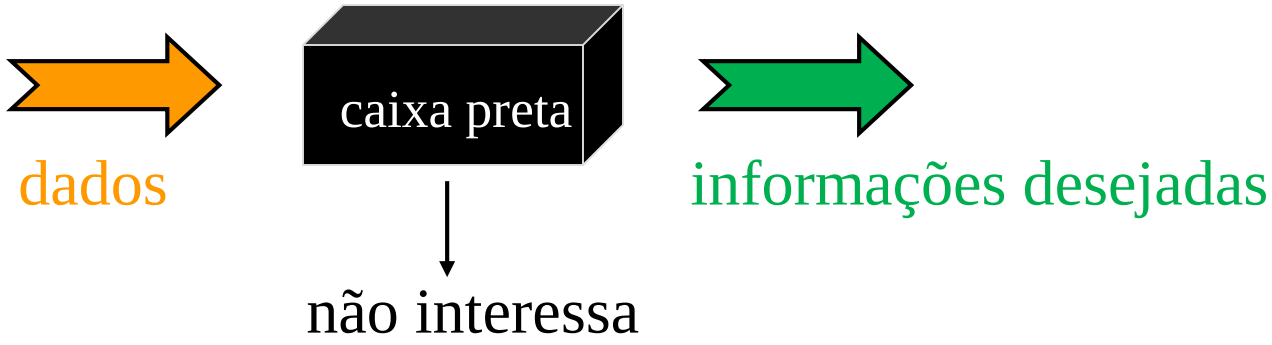
APRESENTAÇÃO

- Encapsulamento
- Superclasse ***Object*** (suprema)
- Estrutura Homogênea Dinâmica
  - ***Vector***
- Referências

Encapsulamento

Este conceito, também chamado de acessibilidade ou ocultamento por alguns estudiosos, corresponde a uma característica importante ao se trabalhar com objetos.

Por meio dele são combinados dados e comportamentos em um objeto para controlar e fornecer certa estabilidade e segurança ao estado do objeto (instância de uma classe).

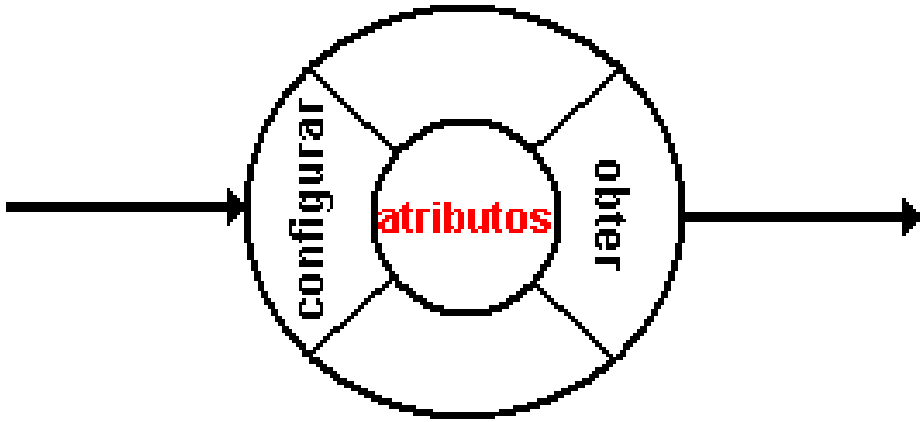


→ O encapsulamento promove o comportamento de caixa preta nos objetos

Encapsulamento

A incorporação deste conceito em POO consiste na implementação de objetos que nunca fornecerão acesso direto aos seus atributos (instâncias).

Este acesso sempre acontecerá através da troca de mensagens com acionamento de métodos específicos da classe que instanciam (possui) estes atributos.



→ O acesso aos atributos da classe só acontece por meio dos métodos da própria classe, tanto no fornecimento quanto na alteração de seus valores

Encapsulamento

Este conceito garante

- facilita e agiliza a **manutenção** do programa
- possibilita **reutilização**
- fornece **confiabilidade**

porque o próprio objeto pode modificar totalmente como seus dados são armazenados, porém seus métodos continuarão fornecendo suas informações desejadas sem que os recursos solicitantes destes dados se importem como estes dados foram alcançados.

comportamento

Encapsulamento

Métodos Privados

Os atributos e métodos são membros de uma classe, podendo cada um deles possuir seu qualificador de acesso adequado a lógica implementada em um programa Java.

Conforme o encapsulamento, os atributos de uma classe deverão ter acesso restrito aos métodos da classe, podendo alguns métodos também serem encapsulados.

Esta situação ocorre com certa frequência, pois é comum a "quebra" de um código em vários métodos, onde alguns deles não terão utilidade "pública", mas somente específica as operações que um método específico desta classe tenha que realizar para chegar ao seu resultado esperado.

Encapsulamento

Normalmente, os métodos privados são definidos por:

- não serem de interesse dos usuários e serviços da aplicação desenvolvida;
- não serem facilmente suportados para uma possível modificação na implementação de sua classe

Para realizar a implementação de um método privado, simplesmente troque seu qualificador de acesso para private e este método só poderá ser acessado por outros métodos definidos na sua própria classe, por exemplo:

```
private boolean verificaStatusSaldo (float valor) {
                :
}
```

Encapsulamento

Métodos *getters* e *setters*

Sendo os atributos de uma classe privados o acesso aos mesmos só acontecerá através de métodos existentes nesta própria classe. Assim, um padrão adotado na POO em Java é a elaboração de métodos de acesso ao conteúdo dos atributos (get) e de modificação dos mesmos (set).

A elaboração destes métodos recebem o prefixo correspondente (get ou set) em seus identificadores (nomes), por exemplo:

```
public void setAno(int ano) {
            :
}
```

Encapsulamento

O método **set**Ano realiza a alteração do estado do atributo ano em um objeto que o declarou como uma variável de instância private.

Observe que este método recebe um parâmetro (**int ano**) que será o responsável pelo novo valor que este atributo receberá.

A simples obtenção do valor armazenado neste atributo poderia ser conseguida através do acionamento do método denominado **getAno**, que possivelmente não receberia nenhum parâmetro, mas retornaria o valor da variável de instância denominada ano.

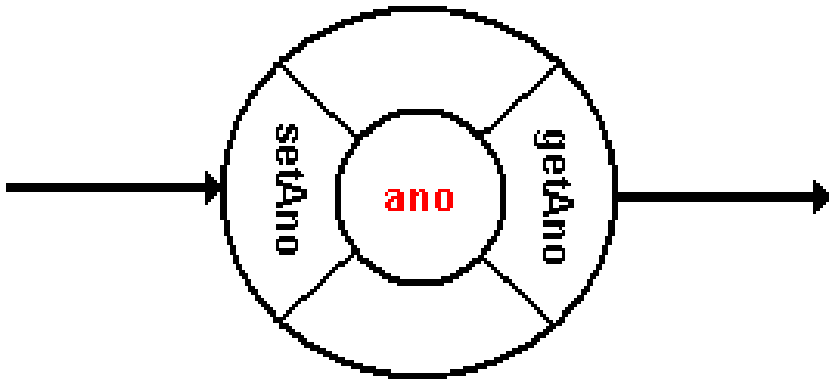
```
public int getAno( ) {
            :
}
```

Encapsulamento

Observe que o qualificador de acesso destes dois métodos (setAno e getAno) são públicos, apesar do atributo acessado (ano) ser privado (private).

Dessa forma, o atributo só poderá ser acessado por outras classes através de seus próprios métodos, respectivamente acionados de acordo com a necessidade do processamento a ser realizado:

- acesso ou obtenção de seu conteúdo (get);
- alteração ou configuração de seu conteúdo (set).



Encapsulamento

```
/** Síntese
  *    Objetivo: registrar uma data
  *    Entrada:  dia, mês, ano
  *    Saída:    data completa com dia da semana
 */
import java.util.Scanner;
 public class RegistraData {
  public static void main(String[] args) {
   int valor;   // usada na leitura de dia, mês e ano
   Data data = new Data();
   Scanner ler = new Scanner(System.in);
   do {
      System.out.print("Informe o ano: ");
      valor = ler.nextInt();
   } while (valor < 1);
   data.setAno(valor);

   do {
      System.out.print("Informe o mês: ");
      valor = ler.nextInt();
   } while ((valor < 1) | (valor > 12));
   data.setMes(valor);
```

Encapsulamento

```
// continuação do exemplo anterior

  System.out.print("Informe o dia: ");
  do {
     valor = ler.nextInt();
     if (!data.validaDia(valor))
       System.out.print("Dia inválido, digite outro: ");
     else
       data.setDia(valor);
  } while (!data.validaDia(valor));

  mostraData(data.getDiaSemana(),data.getDia(),
                              data.getMes(), data.getAno());
 } // encerra o método principal

 // Outro método da classe RegistraData
 public static void mostraData(String diaSemana,
                                  int dia,int mes,int ano){
    System.out.print("\nDia " + dia +"/"+ mes +"/" + ano
                                      + " - " + diaSemana);
 }
}
```

//

Encapsulamento

```
/** Síntese
 *    Atributos: dia, mês, ano, dia da semana
 *    Métodos: getAno(), getMes(),getDia(),setAno(int),
 *             getDiaSemana(),setMes(int), setDia(int),
 *             validaDia(int), semana(String)
 */
import java.util.Date; // classe de manipulação de data
import java.util.Scanner;

public class Data {
  // define registro de Data (atributos)
   private int dia;
   private int mes;
   private int ano;
   private String diaSemana;

  // métodos de acesso aos atributos da classe Data

   // obtém ou fornece o conteúdo do atributo ano
   public int getAno() {
     return this.ano;
   }
```

similar a **struct** em **C**

Encapsulamento

```
// continuação do exemplo anterior
  // configura ou altera o conteúdo do atributo ano
   public void setAno(int anoParametro) {
     this.ano = anoParametro;
   }
   public int getMes() {
     return this.mes;
   }
   public void setMes(int mesParametro) {
     this.mes = mesParametro;
   }
   public int getDia() {
     return this.dia;
   }
   public void setDia(int diaParametro) {
     this.dia = diaParametro;
   }
   public String getDiaSemana() {
      return this.diaSemana;
   }
```

Encapsulamento

```
// continuação...
protected boolean validaDia(int dia) {
       boolean valida = false;
      switch (this.mes) {
       case 1:
       case 3:
       case 5:
       case 7:
       case 8:
       case 10:
       case 12:
         if (dia > 0 && dia <= 31)
            valida = true;
         break;
       case 4:
       case 6:
       case 9:
       case 11:
         if (dia > 0 && dia <= 30)
            valida = true;
         break;
       case 2 :
         if (dia > 0 && dia <= 28)
            valida= true;
```

Encapsulamento

```
// continuação...
          else
            if (dia == 29) {
              if (((ano%4 == 0) && (ano%100 != 0)) ||
                                                (ano%400 == 0))
                   valida = true;
            }
      } // encerra switch
      if(valida) {
        StringBuilder hoje = new StringBuilder();
        hoje.append(this.mes);
        hoje.append("/");
        hoje.append(dia);
        hoje.append("/");
        hoje.append(this.ano);
        semana(hoje.toString());
      }
      return valida;
    }
    private void semana(String dataDesejada) {
      Date data = new Date(dataDesejada);
     switch(data.getDay()) {
       case 0:
         diaSemana = "Domingo";
```

Encapsulamento

```
// continuação do exemplo anterior
      case 1:
        diaSemana = "Segunda-feira";
        break;
      case 2:
        diaSemana = "Terça-feira";
        break;
      case 3:
        diaSemana = "Quarta-feira";
        break;
      case 4:
        diaSemana = "Quinta-feira";
        break;
      case 5:
         diaSemana = "Sexta-feira";
         break;
      case 6:
         diaSemana = "Sábado";
    }
  }
} // fim da classe Data
```

Encapsulamento

"**DEPRECATED**"

Observe em seu programa, no IDE Eclipse, que a classe Date e o método getDay() aparecem de maneira diferente das demais instruções. Como informação deste ambiente é realizado um alerta de que estes recursos são desencorajados ao uso (*deprecated*), normalmente, por sua descontinuidade em novas versões Java, onde estas devem ter sido substituídas por novas classes e métodos.

Por exemplo no trecho do código anterior:

```
switch(data.getDay()) {
   case 0:
      diaSemana = "Domingo";
      break;
       :       // getDay() é "Deprecated"
```

Encapsulamento

Observe no programa anterior que o acesso aos atributos da classe Data acontecem somente por meio de seus métodos, onde cada um deles respeita o padrão de acesso e modificação através do prefixo **set** e **get** em seus identificadores.

Analisando a classe Data também é possível observar como Java cria **Estruturas de Dados Compostas Heterogêneas** (registros). Estas estruturas são criadas e manipulados pelo programa por meio de suas classes.

```
public class Data() {
        private int dia;
        private int mes;
        private int ano;
        private String diaSemana;
}
```

elementos de **Data** (registro ou estrutura hete-rogênea)

Encapsulamento

Analisando o registro Data, do exemplo anterior, é possível observar que os métodos ***get*** e ***set*** foram elaborados com êxito, porém um atributo privado ficou sem a especificação destes métodos padrões (diaSemana).

No entanto, o valor coerente a ser armazenado neste elemento do registro é gerado e apresentado na execução do programa.

Exemplo da console (execução)

```
Informe o ano: 2000
Informe o mês: 2
Informe o dia: 29

Dia 29/2/2000 - Terça-feira
```

Encapsulamento

A prática do desenvolvimento destes métodos ***get*** e ***set*** na programação em Java deve ser condicionada a situação lógica do programa que atenderá as necessidades existentes no problema envolvido.

Dessa forma, estes métodos devem ser planejados antes de serem implementados, não sendo uma "boa prática" de programação, e nem da realização do encapsulamento, a simples implementação destes métodos para qualquer atributo existente em uma classe.

Exemplo: suponha que o usuário deste programa tivesse informado que **sexta-feira** fosse o dia da semana para **29/2/2000**, através de seu método setDiaSemana, enquanto o getDiaSemana mostraria este dia gerando um **dado inconsistente** (errado) a nossa realidade.

Encapsulamento

Instrução *this*

A utilização do ***this*** em um método referência o objeto no qual este método esteja operando. Como exemplo observe o método setAno que altera o valor do atributo private ano definido na classe Data através do ***this***.

O uso desta instrução referencia os atributos existentes no objeto de Data onde o método foi acionado (objeto alvo).

Por exemplo:

> Suponha a execução da chamada de método **obj1.calculaTotal(obj2)**, onde cada ocorrência do ***this*** representa a referência ao objeto referido por **obj1**; a expressão ***this.aux*** representa o acesso a variável ***aux*** do objeto referido por **obj1** (objeto corrente).

Encapsulamento

Algumas outras aplicações do ***this*** são possíveis na POO em Java, mas neste momento é interessante conhecer seu uso também sobre métodos construtores.

Entre as características especiais deste método ainda existe a possibilidade dele acionar um **outro método construtor** na mesma classe, em sua primeira linha de código. Por exemplo:

```
class Aluno {
   Aluno (String nome) {
      this(nome, Matricula.getNovaMatricula());
   }
   public Aluno(String nome, int matricula) {
      chamada = nome;
      codigo = matricula;
        :
   }
}
```

Exercício de Fixação

1) Desenvolva um programa orientado a objeto que possua um método construtor para classe Aluno. Esta classe é composta por dois atributos (matrícula inteira e nome – String) e pode criar quantos alunos o usuário desejar, respeitando o limite máximo de 100 cadastros de Alunos. Seu programa deverá encapsular esta classe Aluno e fazer as respectivas validações para estes dados em métodos específicos para cada atributo (serviços). O valor máximo de cadastro estará definido na constante MAXIMO que estará declarada em outra classe executável que gerenciará a aplicação elaborada e que possibilitará sua execução. Quando o usuário desejar encerrar estes cadastros, ou o limite for atingido, seu programa deverá realizar um salto de 20 linhas na console e apresentar de maneira tabelar (forma de tabela) todos os cadastros realizados. Todos os métodos presentes na classe Aluno não poderão ser estáticos.

Encapsulamento

Compreendendo alguns Qualificadores de Acesso

Os membros de uma classe (atributos e métodos) podem ser definidos como **static** ou não. Os criados como static pertencem a classe e não estão condicionado a criação de seus objetos para serem utilizados.

A sintaxe para o acesso a um membro de uma classe é:

identificador**Objeto**.identificador**Atributo**

ou

identificador**Objeto**.identificador**Método**()

Para membros **static** o acesso pode ser pela classe:

identificador**Classe**.identificador**Atributo**

ou

identificador**Classe**.identificador**Método**()

Encapsulamento

```
/** Síntese
 *    Objetivo: cadastro de produto de uma loja
 *    Entrada:  sem entrada
 *    Saída:    código,preço unitário e nome do produto
 */
public class Loja {
  private static class Produto {  // define estrutura
    public int codigo;            // heterogênea
    public double preco;
    public String nome;
  }

  public static void main(String[] args) {
    Produto prod;   // cria uma referência a Produto
    prod = new Produto(); // aciona construtor alocação
    prod.codigo = 1;
    prod.preco = 300.00;
    prod.nome = "filmadora";
    System.out.print("Código: " + prod.codigo + " =>");
    System.out.print("\t" + prod.nome + "\tR$" +
                                          prod.preco);
  }
}
```

Encapsulamento

- A classe **Loja** possui dentro de si a definição privada da classe **Produto**
- **Produto** esta definida como **private** (acesso somente de códigos existentes dentro da classe Loja)
- Como **main()** é **static** ele só pode usar variáveis e classes que tenham sido declaradas dentro dele ou que também sejam **static**, sendo por esta razão que **Produto** é **static**
- Os atributos de **Produto** foram definidos como **public**, o que indica que quem conseguir acesso a esta classe poderá manipular seus atributos diretamente (sem get/set)
- Estes atributos não são **static**, sendo necessária a criação de um objeto **Produto** para que os mesmos sejam utilizados
- A variável **prod** é criada para receber o endereço do novo objeto **Produto**, sendo somente por meio dela efetuada a manipulação direta dos atributos de **Produto**

Encapsulamento

Acesso a Membros da Classe e do Objeto

Um método pode acessar os dados privados do objeto no qual é chamado, além de também conseguir acessar todos os dados privados de todos os objetos gerados por sua mesma classe.

Outra informação importante, e que confundi muitos programadores, esta relacionada a passagem de parâmetros em Java quando envolve variáveis de referência.

Muitos indivíduos acreditam que esta passagem é feita por referência, porém **Java só passa parâmetros por valor**, mesmo as variáveis que fazem referência de memória a outros objetos.

Encapsulamento

No intuito de elucidar a POO incorporando o uso de estruturas de dados e o encapsulamento acompanhe a evolução do exemplo anterior sobre o registro de produtos em uma loja de magazine.

```
/** Síntese
 *    Objetivo: cadastro de produto de uma loja
 *    Entrada:  sem entrada
 *    Saída:    código,preço unitário e nome do produto
 */
public class RegistraProduto {
  public static void main(String[] args) {
    Produto prod = new Produto(); // cria referência e
    prod.setCodigo(1);            // o objeto prod
    prod.setPreco(300);
    prod.setNome("filmadora");
    System.out.print("Código: " + prod.getCodigo() +
                                                " =>");
    System.out.print("\t" + prod.getNome() + "\tR$" +
                                          prod.getPreco());
  }
}
```

Encapsulamento

```
/** Síntese
 *   Atributos: código, preço, nome
 *   Métodos: getCodigo(), getPreco(), getNome()
 *      setCodigo(int),setPreco(double),setNome(String)
 */
public class Produto {
  // define estrutura heterogênea Produto
   private int codigo;
   private double preco;
   private String nome;

 // métodos set e get para Produto
  public int getCodigo() {
    return codigo;
  }
  public void setCodigo(int vCodigo) {
    this.codigo = vCodigo;
  }
  public double getPreco() {
    return preco;
  }
  public void setPreco(double vPreco) {
    this.preco = vPreco;
  }
```

Encapsulamento

```
// continuação do exemplo anterior
  public String getNome() {
    return nome;
  }
  public void setNome(String vNome) {
    this.nome = vNome;
  }
}
```

- Programa implementado com 2 arquivos físicos distintos e 2 classes públicas (RegistraProduto e Produto)
- Apesar da classe Produto ser pública seus atributos são privados e só podem ser acessados pela própria classe
- Os métodos **get** e **set** da classe Produto são públicos
- Na classe RegistraProduto foi criado um objeto Produto identificado por **prod**, o que possibilitou acesso a todos os métodos públicos desta classe

→ Observe que o acesso aos métodos é sempre pelo objeto **prod**

Encapsulamento

Evoluindo este mesmo exemplo para o uso das estruturas de dados em conjunto (homogêneas e heterogêneas), além da leitura dos dados fornecidas pelo usuário, tem-se a seguinte implementação na classe **RegistraProduto**, pois a classe **Produto** se mantém **sem nenhuma alteração** (bem encapsulada).

```
/** Síntese
 *   Objetivo: cadastros de produtos de uma loja
 *   Entrada:  código,preço unitário,nome dos produtos
 *   Saída: código,preço unitário,nome de cada produto
 */
import java.util.Scanner;
public class RegistraProduto2 {
  public static void main(String[] args) {
    Scanner ler = new Scanner(System.in);
    // matriz 2x3 de referencia a Produto
    Produto prod[][] = new Produto[2][3];
    int automatico = 0;  // códigos gerados automáticos
```

Encapsulamento

```
// continuação...
    for(int aux=0;aux < 2; aux++) {
      for(int cont=0; cont < 3; cont++) {
        // cria um objeto Produto por vez
         prod[aux][cont] = new Produto();
         String auxNome = new String();
         double auxPreco;
        // código gerado automaticamente
         prod[aux][cont].setCodigo(++automatico);
        // Leitura do nome do Produto
         do {
           System.out.print("Digite nome do produto: ");
           auxNome = ler.next();
           auxNome = auxNome.trim();
         } while (auxNome == null);
         prod[aux][cont].setNome(auxNome);
        // Leitura do preço de cada Produto
         do {  System.out.print("Informe seu preço: ");
               auxPreco = ler.nextDouble();
         } while (auxPreco <= 0);
         prod[aux][cont].setPreco(auxPreco);
      }
    }
```

Encapsulamento

```
// continuação...
    for(int aux=0;aux < 2; aux++) {
      for(int cont=0; cont < 3; cont++) {
        System.out.print("Código: " +
                    prod[aux][cont].getCodigo() + " =>");
        System.out.println("\t" +
                     prod[aux][cont].getNome()+ "\tR$"+
                              prod[aux][cont].getPreco());
      }
    }
  }
}
```

- Programa implementado com 2 arquivos físicos distintos e 2 classes públicas (RegistraProduto2 e Produto)
- Os atributos de Produto são privados e só podem ser acessados por seus métodos públicos **get** e **set**
- Na classe RegistraProduto2 é criada uma matriz 2x3 indicando que 6 produtos poderão ser criados e suas referências serão armazenadas nas posições desta matraiz

→ O acesso ao Produto é sempre pelo objeto **prod** e seus índices

Exercícios de Fixação

2) Elabore um programa que realize o cadastro de contas bancárias através do registros dos dados: número da conta; nome do cliente (principal proprietário da conta); saldo atual. Este cadastro será efetuado em um posto de atendimento bancário que pode possuir até 50 contas abertas com números completamente diferentes. Crie um menu com as opções relacionadas abaixo e as implemente adequadamente ao encapsulamento.
    a) Cadastrar nova conta respeitando o limite máximo definido em uma constante em seu programa que deverá sempre ser usada em seu código, onde for conveniente;
    b) Mostrar todas as contas de um determinado cliente;
    c) Fechar (desativar) uma conta especifica;
    e) Consultar todas as contas desativadas (máximo 50);
    d) Encerrar o sistema de gerenciamento de contas.

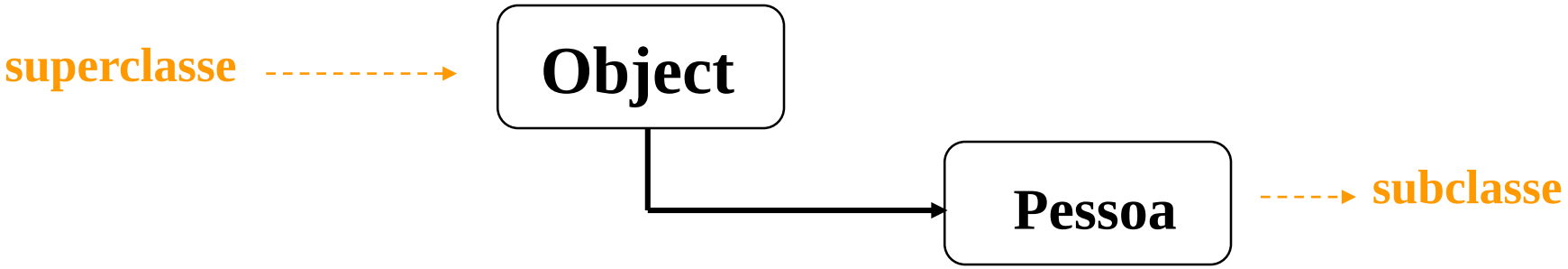
Exercícios de Fixação

3) Elabore um programa seguindo todas as características de encapsulamento que permita a uma prefeitura efetuar uma pesquisa entre os habitantes de sua cidade, coletando informações sobre o salário, idade, sexo e número de filhos. Este programa deverá ler os dados de uma quantidade indeterminada de pessoas (quantidade menor que mil, e armazenar em um registro cada dado registrado. Ao final deverá ser mostrado:
   - menor idade entre os entrevistados;
   - maior salário registrado;
   - média do número de filhos;
   - média do salário das pessoas registradas;
   - média dos homens com salário superior a R$300,00;
   - quantidade de pessoas que tem salário maior que a média de todas as pessoas pesquisadas.

Superclasse *Object*

Toda classe criada em Java é subclasse da classe Object, não sendo necessário incluir nenhuma instrução específica para que isso se estabeleça.

Exemplo:



- Object é a classe referida quando nenhuma superclasse (classe pai) é indicada
- Pode-se usar uma variável do tipo Object para **referenciar objetos** de qualquer tipo (outros objetos)
- O tipo Object só é útil como contêiner genérico de valores arbitrários, sendo necessário conhecer melhor o tipo original da classe para trabalhar sobre seus valores

Estrutura Homogênea Dinâmica

A criação de estruturas de dados compostas dinâmicas, para o armazenamento de objetos em memória, é possível em Java por meio da criação de objetos da classe ***Vector*** ou da classe ***ArrayList*** (estudada a frente).

Vector

- Disponível em **java.util.Vector**
- Consiste em um array de Object com dimensão dinâmica, conforme a necessidade de armazenamento
- Não guarda nenhum tipo de dado primitivo (embutidas)
- Possui um número menor ou igual a sua capacidade de armazenamento
- Sua expansão pode acontecer por meio de um valor informado em sua construção ou pelo valor padrão, que corresponde ao dobro de seu tamanho atual

Estrutura Homogênea Dinâmica

Vector

- tamanho variável
- armazena somente objetos (tipo ***Object***)
- Alteração sobre o mesmo objeto

Array

- tamanho fixo
- armazena valores de um mesmo tipo de dado
- Alteração cria um novo objeto em memória

Estrutura Homogênea Dinâmica

Métodos Importantes no Uso da Vector

- **vector**: cria um vector vazio (sem objetos)
- **vector(int)**: cria um vector vazio com tamanho inicial
- **vector(int, int)**: cria vector vazio com tamanho inicial e valor de incremento indicado
- **size**: obtém o número de elementos do vector
- **setSize(int)**: configura tamanho do objeto vector, sendo maior que o atual adiciona novos elementos nulos e menor descarta os itens além do tamanho informado
- **isEmpty**: verifica se vector esta vazio
- **contains(object)**: verifica se object é componente de vector
- **indexOf(object)**: retorna o índice da primeira ocorrência de object no vector ou –1 se ele não estiver no vector
- **element(int)**: obtém o componente do índice parametrizado
- **toString**: retorna objeto String representando o vector

Estrutura Homogênea Dinâmica

continuação... **Métodos Importantes no Uso da Vector**

- **setElementAt(object, int)**: copia object sobre o índice indicado
- **removeElement(int)**: remove o elemento de índice indicado, sendo vector reorganizado e diminuído em 1 posição
- **removeAllElements**: remove todos os elementos de vector e reduz seu tamanho para zero
- **insertElementAt(object, int)**: inseri um novo componente na posição indicada e reorganiza o vector em mais uma posição
- **addElement(object)**: vector é acrescido de uma nova posição que recebe o objeto indicado

Estrutura Homogênea Dinâmica

```
/** Síntese
 *    Objetivo: guardar e mostrar dados de pessoas
 *    Entrada:  nomes e idades
 *    Saída:    listagem das pessoas cadastradas
*/
import java.util.Vector;
public class CadastraPessoa {
  public static void main(String[] args) {
    Vector variasPessoas = new Vector();
    String nome;
    int idade, opcao;
    do {Pessoa pessoa = new Pessoa();
        System.out.println("Informe o nome: ");
        nome = Servicos.lerString();
        pessoa.setNome(nome);
        System.out.println("Digite a idade em anos: ");
        idade = Servicos.lerInt(1,130);
        pessoa.setIdade(idade);
        variasPessoas.add(pessoa);
        System.out.println("Novo cadastro (0-NÃO / 1-SIM)?");
        opcao = Servicos.lerInt(0, 1);
    } while (opcao == 1);
    Servicos.limparTela(20);
    Servicos.mostraTabela(variasPessoas);
  }
}
```

//

Estrutura Homogênea Dinâmica

```
// continuação do exemplo anterior
/** Síntese
 *    Atributos: nome, idade
 *    Métodos:  getIdade(), setIdade(String), getNome(),
 *                setNome(String)
 */
  public class Pessoa {
    private String nome;
    private int idade;

    public int getIdade() {
      return idade;
    }
    public String getNome() {
      return nome;
    }
    public void setNome(String nome) {
      this.nome = nome;
    }
    public void setIdade(int idade) {
      this.idade = idade;
    }
  }
```

//

Estrutura Homogênea Dinâmica

```
/** Síntese
 *    Atributos:
 *    Métodos: lerString(), lerInt(int, int),
 *             validaString(String, Scanner),
 *             validaInteiro(int, int, int)
 *             limparTela(int), mostraTabela(Vector)
 */
import java.util.Scanner;
import java.util.Vector;
 public class Servicos {
  // Métodos de Leitura
   public static String lerString(){
      Scanner ler = new Scanner(System.in);
      return validaString(ler.nextLine(),ler);
   }
   public static int lerInt(int min, int max){
      Scanner ler = new Scanner(System.in);
      return validaInteiro(ler.nextInt(),min,max);
   }
  // Métodos de validação
   public static String validaString(String str,
                                     Scanner ler){
     while(str.isEmpty()) {
       System.out.println("Inválido, informe novamente.");
       str = ler.nextLine();
     }
     return str;
   }
```

Estrutura Homogênea Dinâmica

```
// continuação do exemplo anterior
   public static int validaInteiro(int inteiro,
                                    int min, int max){
     while((inteiro < min) ||(inteiro > max)) {
       System.out.println("Inválido, informe novamente.");
       inteiro = lerInt(min,max);
     }
     return inteiro;
   }
  // Métodos de apresentação de dados
   public static void limparTela(int linhas) {
     for(int aux=1;aux<=linhas;aux++)
        System.out.println();
   }
   public static void mostraTabela(Vector pessoas) {
     System.out.println("NOME\t\tIDADE");
     for(int aux=0;aux < pessoas.size(); aux++) {
       Pessoa pes = (Pessoa) pessoas.get(aux);
       System.out.println(pes.getNome() + "\t" +
                                       pes.getIdade());
     }
   }
}
```

Exercícios de Fixação

4) Elabore um programa que permita ao usuário cadastrar quantos nomes ele quiser e quando terminar de cadastrar apresente todos os nomes cadastrados na ordem que o usuário escolher:

OPÇÕES DE APRESENTAÇÃO

1- Sequência de inserção;

2- Sequência inversa de inserção;

0- Encerra sem mostrar os nomes cadastrados.

Esta solução deverá possuir um método específico para cada tipo de apresentação (inserção ou inversa).

Exercícios de Fixação

5) Desenvolva um programa que armazene o nome completo de um aluno, sua matrícula na instituição, que nunca pode se repetir (valor chave sempre maior que a constante MAX com valor 1000 e nunca podendo ser igual a outra matrícula já existente), além de sua média final (igual ou maior que zero e até 10 pontos).

   Faça um programa respeitando TODAS as regras e definições para POO e cadastre todos os alunos desejados pelo usuário. Quando este usuário não quiser mais cadastrar alunos apresente um relatório do tipo de uma tabela (tabelar) contendo somente uma linha de cabeçalho seguida de uma linha para cada registro de aluno com todos registro efetuados neste programa.

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, 2002.
  - Capítulo 7
- ASCENCIO, A. F. G.; CAMPOS, E. A. V., Fundamentos da programação de computadores, Pearson Hall,2007.
  - Capítulo 10
- CAMARÃO, C., FIGUEIREDO, L., Programação de Computadores em Java, Rio de Janeiro: LTC, 2003.
  - Capítulo 13
- Universidade de Brasília (UnB FGA)
  - http://cae.ucb.br/conteudo/unbfga