

ECM251 – Linguagens de Programação I

Aula 09 – L1/1, L2/1 e L3/1

Engenharia da Computação – 3ª série

***Polimorfismo, Vetores e ArrayList*
*(L1/1, L2/1 e L3/1)***

2025

ECM251 – Linguagens de Programação I

Aula 09 – L1/1, L2/1 e L3/1

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro*.

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Tópico

- Polimorfismo

Polimorfismo

Definição



- Conceito importante da POO - Programação Orientada a Objetos, que permite que objetos de diferentes classes sejam tratados de forma uniforme;
- Se refere à capacidade de um objeto ser referenciado de maneiras diferentes e responder de maneiras diferentes a chamadas de métodos com o mesmo nome;
- Permite que um objeto seja tratado como se fosse de uma classe diferente, desde que ele suporte a interface necessária;

Polimorfismo

Definição



- Com ele, pode-se escrever um código que funcione com vários tipos de objetos, ao invés de escrever códigos separados para cada tipo;
- O polimorfismo é um dos princípios básicos da programação orientada a objetos e é frequentemente usado em combinação com herança e interfaces para criar hierarquias de classes flexíveis e reutilizáveis;
- Em Java, existem dois tipos: polimorfismo de sobrecarga (sobrecarga, ou *overload*) e polimorfismo de subtipo (sobreposição, ou *override*).

Polimorfismo

Exemplo



- Dada uma classe chamada "Animal" e duas subclasses, "Cachorro" e "Gato";
- Ambas as subclasses herdam da classe Animal, mas têm seus próprios comportamentos específicos;
- No entanto, como ambas as classes são animais, podem ser tratadas de forma polimórfica;
- Isso significa que pode ser escrito um método que aceita um objeto do tipo Animal como argumento e pode passar um objeto do tipo Cachorro ou Gato para esse método, que funcionará corretamente, chamando o método específico daquela classe.

Polimorfismo

Exemplo



- A classe **Animal** é a classe base, que define um atributo "nome" e um método **fazerBarulho()** que será sobrescrito nas subclasses **Cachorro** e **Gato**.
- As classes **Cachorro** e **Gato** herdam de **Animal** e adicionam seu próprio comportamento específico, implementando o método **fazerBarulho()** de acordo com o som que cada animal faz.

Polimorfismo

Exemplo



```
1 // Classe Animal
2 public class Animal
3 {   private String nome;
4     public Animal(String nome)
5     {   this.nome = nome;
6     }
7     public void fazerBarulho()
8     {   System.out.println("Barulho de animal genérico");
9     }
10    public String getNome()
11    {   return nome;
12    }
13 }
14
```

```
1 // Classe Cachorro, que herda de Animal
2 public class Cachorro extends Animal
3 {   public Cachorro(String nome)
4     {   super(nome);
5     }
6     public void fazerBarulho()
7     {   System.out.println("Au au!");
8     }
9 }
10
```

```
1 // Classe Gato, que também herda de Animal
2 public class Gato extends Animal
3 {   public Gato(String nome)
4     {   super(nome);
5     }
6     public void fazerBarulho()
7     {   System.out.println("Miau!");
8     }
9 }
10
```


Tópico

- Sobrecarga

Sobrecarga

Definição



- Também conhecida por *overload*, a sobrecarga é um conceito da POO que permite que um mesmo nome de método seja utilizado para executar diferentes operações, de acordo com os argumentos que são passados para ele;
- Em outras palavras, a sobrecarga de métodos permite que um método tenha diferentes versões com o mesmo nome, mas com parâmetros de entrada diferentes, ou seja, com assinaturas diferentes:

int calcula(double a, double b)

int calcula(int a, int b)

Sobrecarga

Definição



- A sobrecarga é útil para criar métodos que realizam tarefas semelhantes, mas que podem receber argumentos de tipos diferentes, como em um método que realiza uma soma entre dois números inteiros e um outro que realiza a soma entre dois números decimais;
- Para exemplificar, considerando um método "soma" que recebe dois números inteiros como argumentos e retorna a soma entre eles;
- Pode-se criar uma sobrecarga desse método, que recebe dois números decimais como argumentos e também retorna a soma entre eles.

Sobrecarga

Exemplo



- Sobrecarga: na mesma classe, métodos com o mesmo nome, porém com assinaturas diferentes:

```
1 public class OperacoesMatematicas
2 {   public int soma(int a, int b)
3     {   return a + b;
4     }
5     public double soma(double a, double b)
6     {   return a + b;
7     }
8 }
9
```

Tópico

- Sobreposição

Sobreposição

Definição



- Também conhecida por *override*, a sobreposição é um conceito da POO que permite que uma classe filha (subclasse) forneça uma implementação específica de um método que já é definido em sua classe pai (superclasse);
- Em outras palavras, a sobreposição permite que a subclasse altere ou estenda o comportamento de um método herdado da superclasse, fornecendo sua própria implementação para o método;
- Para isso, o método na subclasse deve ter a mesma assinatura (nome e parâmetros) que o método herdado da superclasse;

Sobreposição

Definição



- Um exemplo comum de sobreposição é o método ***toString()*** da classe **Object**, que é herdado por todas as outras classes do Java;
- Esse método retorna uma representação em *String* do objeto, mas a implementação padrão da classe **Object** é geralmente insatisfatória para a maioria delas;
- Por isso, muitas classes em Java sobrepõem o método ***toString()*** para fornecer uma representação em *String* mais útil ou personalizada do objeto.

Sobreposição

Exemplo



- Sobreposição: permite que a subclasse altere ou estenda o método herdado da superclasse, em métodos com os mesmos nomes e com as mesmas assinaturas:

```
1 public class Pessoa
2 {   private String nome;
3     private int idade;
4     // construtor, getters e setters aqui
5     public String toString()
6     {   return "Pessoa [nome=" + nome + ", idade=" + idade + "]";
7     }
8 }
9
```


Sobreposição

Exemplo



- Sobreposição: No trecho abaixo, em uma outra classe:

```
.  
. Um d = new Um();  
Dois e = new Dois();  
int a,b;  
a = d.calc(10);  
b = e.calc(10);  
. .
```

```
1 public class Um  
2 { public int calc(int x)  
3   { return (x * x);  
4   }  
5 }  
6
```

```
1 public class Dois extends Um  
2 { public int calc(int x)  
3   { return (x + 5);  
4   }  
5 }  
6
```

Tópico

- Vetores

Vetores

Definição



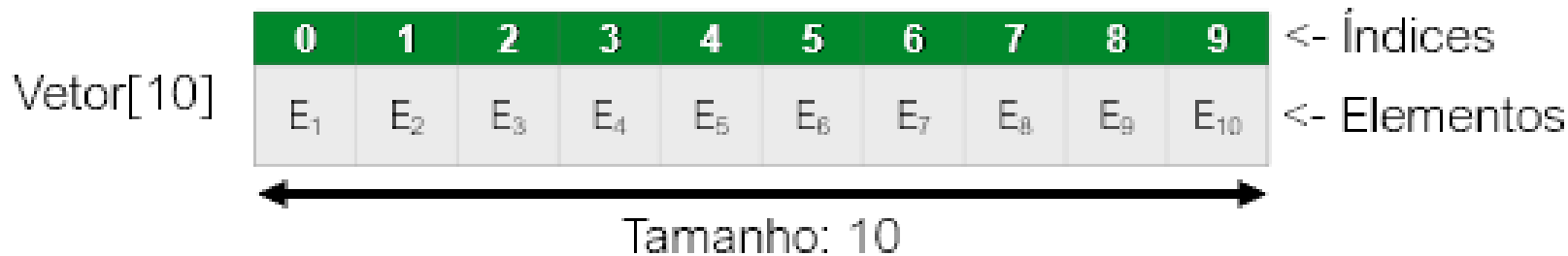
- Vetores, ou *Arrays*, são estruturas de dados homogêneas de acesso indexado, que:
 - É um conjunto de variáveis, base para estruturas de dados;
 - Possuem todas as variáveis do mesmo tipo, homogêneas;
 - Têm suas variáveis acessadas por posição, através do acesso indexado (índices);
 - A primeira posição de suas variáveis tem o índice 0; e
 - A última posição de suas variáveis tem o índice igual ao tamanho do vetor subtraído de uma unidade ($length - 1$);

Vetores

Exemplo



- Representação gráfica de um vetor:



Vetores

Exemplos



- Declaração de vetores, em Java:

```
//cria o vetor vetor de 10 posições contendo um inteiro em cada posição  
//a primeira é 0 e a última é 9
```

```
private int [] vetor = new int[10];
```

```
//cria o vetor livre de 5 posições contendo um true ou false em cada uma  
boolean livre[] = new boolean[5];
```

```
//cria um vetor de String de duas posições chamado nomes  
String[] nomes;  
nomes = new String[2];
```

Vetores

Exemplos



- Atribuição de valores a vetores, em Java:

```
vetor[0] = 1;  
vetor[9] = 10;  
livre[3] = false;
```

Vetores

Exemplos



- Atribuição de valores a vetores usando listas, em Java:

```
int array[] = {10, 20, 30};
```

```
String[] nomes = {"João", "Maria"};
```

Vetores

Exemplo



- Leitura de valores de vetores (elementos), em Java:

```
int x = vetor[7];  
  
if(livre[2]){  
    System.out.println(nomes[2]);  
}
```


Vetores

Exemplo



- Percorrendo um vetor em atribuição, em Java:
 - Existe uma propriedade dos vetores, *length*, que indica o tamanho do vetor em inteiro.

```
for(int i = 0; i < vetor.length; i++){  
    vetor[i] = Integer.parseInt(  
        JOptionPane.showInputDialog("Digite o valor"));  
}
```

Vetores

Exemplo



- Percorrendo um vetor em leitura dos elementos, em Java:
 - Existe uma propriedade dos vetores, *length*, que indica o tamanho do vetor em inteiro.

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

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Tópico

- Busca Sequencial com Vetores

Definição



- Como encontrar um elemento específico em um vetor?
 - Percorra o vetor todo e pergunte, em cada posição, se o elemento específico, ou chave de busca, é igual ao elemento que está armazenado no vetor, naquela posição;
 - Se for igual, retorne o índice da posição em que está o elemento procurado;
 - Se não for igual, repita o procedimento para o próximo índice, até encontrar, ou até chegar no final do vetor;
 - No final, se não encontrou a chave de busca depois de ter percorrido o vetor todo, retorne -1 (índice inexistente).

Busca Sequencial com Vetores

Exemplo



- Fazendo uma busca sequencial em um vetor, em Java:

```
public class Exemplo{
    int[] vetor;

    public Exemplo(int[] v){
        vetor = v;
    }
    public int busca(int chave){
        for(int i = 0; i < vetor.length; i++){
            if (vetor[i] == chave){
                return i;
            }
        }
        return -1;
    }
}
```

Tópico

- Vetores Bidimensionais

Vetores Bidimensionais

Definição



- Um vetor de duas dimensões é um vetor que contém dois índices, independentes;
- Normalmente, representam uma matriz, da matemática;
- O primeiro índice faz referência a um outro vetor e é o índice das linhas;
- O segundo índice faz referência a um elemento do vetor determinado pela sua linha e é o índice das colunas.

Vetores Bidimensionais

Exemplo



- Atribuindo números reais aleatórios a cada uma das posições de uma matriz 10x5 (10 linhas por 5 colunas) , em Java:

```
//instanciação
double[][] matriz = new double[10][5];

//atribuição a uma posição
matriz[3][2] = 10.0;

//atribuição a todas as posições
for(int i = 0; i < matriz.length; i++){
    for(int j = 0; j < matriz[i].length; j++){
        matriz[i][j] = Math.random();
    }
}

//leitura
double x = matriz[0][0];
```


Tópico

- Vetores Multidimensionais

Vetores Multidimensionais

Definição



- Estendendo-se o conceito anterior, podem ser criadas estruturas de armazenamento de elementos com **n** dimensões, em Java;
- Para se criar um cubo, por exemplo, pode ser criada uma matriz com um vetor para cada uma de suas posições, ou seja, 3 (três) dimensões: altura; largura e profundidade.

Vetores Multidimensionais

Exemplo



- Criando-se uma estrutura tridimensional de armazenamento de elementos, representando um cubo, em Java:

```
String[][][] cubo = new String[10][10][10];
```

- Lembrando-se que serão necessários 3 (três) índices independentes, para controlar esse cubo.

Tópico

- Tipos de Vetores

Tipos de Vetores

Definição



- Assim como acontece com qualquer outra variável em Java, um vetor pode ser declarado com um tipo primitivo de dado ou de um tipo objeto, específico;
- Vetores podem, também, assumir os papéis de variáveis locais, variáveis de instância, parâmetros de métodos ou tipos de retornos de métodos.

Tipos de Vetores

Exemplo



- Vetores como parâmetros e retornos de métodos e, também, como variáveis, em Java:

```
public Aluno[] getTurma(Aluno[][][] campus, int ano, int curso){  
    Aluno[] alunos = campus[curso][ano];  
    return alunos;  
}  
  
private Professor[] docentes;
```

Tópico

- ArrayList

ArrayList

Definição



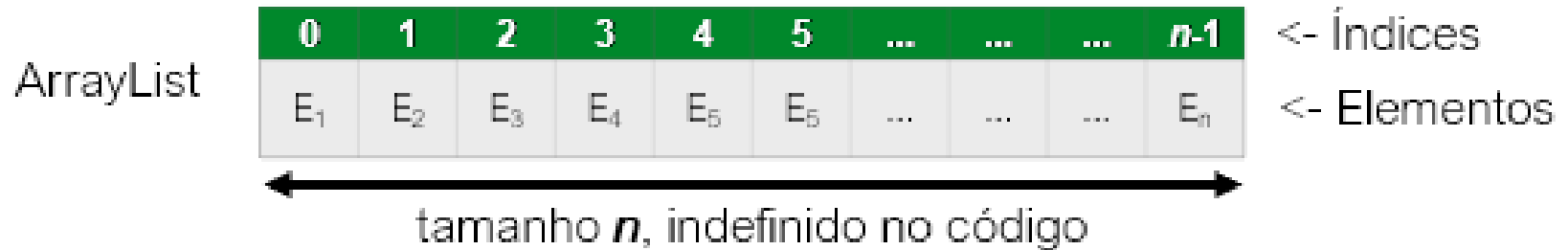
- O **ArrayList** é uma classe do pacote *java.util*, que representa uma coleção homogênea de elementos, com acesso indexado;
- O **ArrayList** “não tem limitação de tamanho” (dentro da quantidade de memória disponível para a aplicação), crescendo conforme os elementos vão sendo adicionados, um a um.

ArrayList

Exemplo



- Representação gráfica de um ArrayList:



ArrayList

Definição



- Um **ArrayList** não pode ser criado com tipos primitivos, somente com objetos;
- Um **ArrayList** de **Double** pode adicionar elementos *double* no formato literal (ex.: 10.0), ou variáveis *double*, pois serão automaticamente convertidos para objetos **Double**, pela *wrapper class* **Double**;
- *Wrapper classes*, as classes empacotadoras, existem uma para cada tipo primitivo de dados em Java, ou seja, **Double**, **Integer**, **Character**, **Boolean**, **Long**, **Float**, **Byte** e **Short**., sendo os seus nomes autoexplicativos.

Exemplo



- Construtor para instanciar um ArrayList de objetos:

```
ArrayList lista = new ArrayList();
```

- Construtor para instanciar um ArrayList de *String*:

```
ArrayList<String> lista = new ArrayList<String>();
```

- Construtor para instanciar um ArrayList de *double*:

```
ArrayList<Double> lista = new ArrayList<Double>();
```

ArrayList

Definição



- Quando um **ArrayList** é instanciado, seu tamanho é 0 (zero);
- Para saber o tamanho atual de um **ArrayList**, deve-se invocar o método *size()*;
- Quando se adiciona um elemento a um **ArrayList**, o tamanho é automaticamente incrementado;
- Quando se remove um elemento de um **ArrayList**, o tamanho é automaticamente decrementado;

ArrayList

Definição



- Quando um elemento é adicionado a um **ArrayList**, através do método ***add(elemento)***, o mesmo é colocado no final, imediatamente após o último elemento antes armazenado, e o tamanho é acrescido em uma unidade;
- O parâmetro ***elemento*** deve ser do mesmo tipo do **ArrayList**, por exemplo, se for um **ArrayList<String>** só serão permitidas *Strings*;
- Se um **ArrayList** não tiver seu tipo definido, entende-se que esse **ArrayList** é do tipo **Object** e qualquer objeto poderá a ele ser adicionado, exceto os tipos primitivos;

ArrayList

Definição



- Para se remover um elemento de um ArrayList, invoca-se o método ***remove(posição)***:
- Onde $0 \leq \textit{posição} < \textit{size}()$, ou seja, como se fosse seu índice;
- Caso contrário é gerado um erro de exceção, pelo ***ArrayIndexOutOfBoundsException***;

ArrayList

Definição



- Quando se remove um elemento de um **ArrayList**, seu tamanho é decrementado em um;
- Se for removido um elemento de um **ArrayList** que está antes de outros elementos, o índice dos demais é recalculado;
- Por exemplo, removendo-se o elemento do índice 2 de um **ArrayList** com 5 elementos: o elemento do índice 3 passa a ser o elemento do índice 2; e o elemento do índice 4 passa a ser o elemento do índice 3; além disso, o tamanho do **ArrayList** passa de 5 para 4.

ArrayList

Definição



- Para se obter um elemento de uma determinada posição de um *ArrayList*, invoca-se o método ***tipo get(posicao)***;
- Onde ***tipo*** é o tipo do *ArrayList*, por exemplo, se for ***<String>***, ***tipo*** é ***String***; se for ***Object***, ***tipo*** é ***Object***;
- O método ***get()*** não remove o elemento, apenas recebe uma referência dele.

ArrayList

Exemplos



- Obter o tamanho de um **ArrayList**:

```
int size();
```

- Adicionar um elemento ao final de um **ArrayList**:

```
void add(elemento);
```

- Remover um elemento de um **ArrayList**:

```
remove(int posição);
```

- Obter um elemento de um **ArrayList**:

```
tipo get(int posicao);
```

Exemplo



- Percorrendo um **ArrayList** com um comando **for**:

```
ArrayList<String> lista = new ArrayList<String>();  
lista.add("palavra 1");  
lista.add("palavra 2");  
String s;  
for(int i = 0; i < lista.size(); i++){  
    s = lista.get(i);  
    System.out.println(s);  
}
```

O resultado da execução desse código é a impressão na tela:

palavra 1

palavra 2

Definição



- O comando ***for-each*** é um laço simplificado para coleções;
- O ***for-each*** percorre a coleção do início ao fim e a única forma de interrompê-lo, antecipadamente, é usando um comando ***break***;
- Importante: uma coleção não pode ter seu tamanho alterado durante a execução de um comando ***for-each***, o que causa um erro de ***exception*** se isso ocorrer.

ArrayList

Exemplo



- Percorrendo um **ArrayList** com um comando *for-each*:

```
ArrayList<String> lista = new ArrayList<String>();  
lista.add("palavra 1");  
lista.add("palavra 2");  
for(String s:lista){  
    System.out.println(s);  
}
```

O resultado da execução desse código é a impressão na tela:

palavra 1

palavra 2

ArrayList

Exemplo



- Contagem quando elementos do **arraylist1** também estão no **arraylist2**, assumindo-se não haver elementos repetidos, encerrando o laço interno com um comando **break** se ocorrer:

```
3      public int contaIguais(ArrayList<Integer> lista1,
4                             ArrayList<Integer> lista2){
5
6          int count = 0;
7          for(int elemento1: lista1){
8              for(int elemento2: lista2){
9                  if(elemento1 == elemento2){
10                     count++;
11                     break;
12                 }
13             }
14         }
15         return count;
16     }
```

Exercício

- Digite o código abaixo em sua I.D.E., entenda-o e depois execute-o para ver os resultados e comprovar seu entendimento.



Exercício



- Sentindo-se solitário em um sábado à noite, o aluno de computação pensa consigo mesmo "Que bom seria se houvesse um jeito de conversar com meus amigos sem precisar sair do computador!". Então, ele resolve criar um *software* chamado Rede de Amigos, onde você possa adicionar seus amigos e conversar com eles. Para isso ele criou uma classe chamada **Amigo**, com os atributos **nome** (*String*), **sexo** (*String*), **idade** (*int*) e **mensagem** (*String*), que armazena a última mensagem enviada para o amigo. Claro que, sendo bom programador, respeitou o encapsulamento e criou os métodos de acesso e os modificadores.

Exercício



- Depois o aluno de computação criou a classe **Rede**, que contém um **ArrayList** de amigos. Ele fez um método para adicionar amigos e um para bloquear amigos (remover). Fez um método que encontra um amigo pelo nome, retornando sua posição no **Arraylist** (ou -1 se não achar). E um método para enviar uma mensagem para o amigo, que encontra o amigo pelo nome e altera seu atributo mensagem (a mensagem pode ter no máximo 144 caracteres; se for maior o sistema trunca em avisar nada). Ele fez também um método que retorna um vetor com o(s) amigo(s) mais velho(s).

Exercício



- Para testar seu sistema, o aluno de computação fez uma classe **Teste** com o método *main()* que, usando o **JOptionPane**, possui um loop com as opções 1. add amigo, 2. block amigo, 3. procura amigo, 4. envia mensagem, 5. lista velhos e 6. sair.

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercício



```
1 public class Amigo
2 { private String nome, sexo, mensagem;
3   private int idade;
4   public String getNome()
5   { return nome;
6   }
7   public String getSexo()
8   { return sexo;
9   }
10  public String getMensagem()
11  { return mensagem;
12  }
13  public int getIdade()
14  { return idade;
15  }
16  public void setNome(String nome)
17  { this.nome = nome;
18  }
19  public void setSexo(String sexo)
20  { this.sexo = sexo;
21  }
22  public void setMensagem(String mensagem)
23  { //tamanho da mensagem e no maximo 144
24    if(mensagem.length() <= 144)
25    { this.mensagem = mensagem;
26    }
27    else
28    { //trunca se for maior
29      this.mensagem = mensagem.substring(0, 144);
30    }
31  }
32  public void setIdade(int idade)
33  { this.idade = idade;
34  }
35  public String toString()
36  { return "[Nome: "+nome+"] [Sexo: "+sexo+"] [Idade: "+
37    idade+"]\n[Mensagem: "+mensagem+"]";
38  }
39 }
40
```

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercício



```
1 import java.util.ArrayList;
2 public class Rede
3 { private ArrayList<Amigo> amigos;
4   public Rede()
5   { amigos = new ArrayList<Amigo>();
6   }
7   public void addAmigo(Amigo amigo)
8   { amigos.add(amigo);
9   }
10  public boolean blockAmigo(String nome)
11  { int posicao = buscar(nome);
12    if(posicao >= 0)
13    { amigos.remove(posicao);
14      return true;
15    }
16    else
17    { return false;
18    }
19  }
20  public int procurarAmigo(String nome)
21  { return buscar(nome);
22  }
```

```
23  public boolean enviarMensagem(String nome, String mensagem)
24  { int posicao = buscar(nome);
25    if(posicao >= 0)
26    { Amigo amigo = amigos.get(posicao);
27      amigo.setMensagem(mensagem);
28      return true;
29    }
30    else
31    { return false;
32    }
33  }
34  public Amigo[] procurarVelhos()
35  { if(amigos.size() == 0)
36    { return new Amigo[0];
37    }
38    int maior = amigos.get(0).getIdade();
39    //encontrar a maior idade
40    for(Amigo amigo:amigos)
41    { if(amigo.getIdade() > maior)
42      { maior = amigo.getIdade();
43      }
44    }
```

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercício



```
45 //contar quantos tem a maior idade
46 int qtde = 0;
47 for(Amigo amigo:amigos)
48 { if(amigo.getIdade() == maior)
49   { qtde++;
50   }
51 }
52 //criar vetor de amigos
53 Amigo[] velhos = new Amigo[qtde];
54 //popular o vetor de mais velhos
55 int k = 0;
56 for(int i = 0; i < amigos.size(); i++)
57 { Amigo amigo = amigos.get(i);
58   if(amigo.getIdade() == maior)
59   { velhos[k++] = amigo;
60   }
61 }
62 return velhos;
63 }
```

```
64 private int buscar(String nome)
65 { for(int i = 0; i < amigos.size(); i++)
66   { Amigo amigo = amigos.get(i);
67     String nomeAmigo = amigo.getNome();
68     if(nome.equals(nomeAmigo))
69     { return i;
70     }
71   }
72   return -1; //nao achou
73 }
74 public void addAmigo(String nome, String sexo, int idade)
75 { Amigo amigo = new Amigo();
76   amigo.setNome(nome);
77   amigo.setSexo(sexo);
78   amigo.setIdade(idade);
79   amigos.add(amigo);
80 }
81 public void listarAmigos()
82 { for(Amigo amigo:amigos)
83   { System.out.println(amigo);
84   }
85 }
86 }
87 }
```

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercício



```
1 import javax.swing.JOptionPane;
2 public class Teste
3 { public static void main(String[] args)
4   { Rede rede = new Rede();
5     int menu;
6     String nome = null;
7     String sexo = null;
8     do
9     { menu = Integer.parseInt(JOptionPane.showInputDialog(
10      "1 add amigo\n2 block amigo\n3 procura amigo"+
11      "\n4 envia mensagem\n5 lista velhos\n6 sair"+
12      "\n7 listar todos"));
13
14     if(menu == 1)
15     { nome = JOptionPane.showInputDialog("Nome:");
16       sexo = JOptionPane.showInputDialog("Sexo:");
17       int idade = Integer.parseInt(
18        JOptionPane.showInputDialog("Idade:"));
19       rede.addAmigo(nome, sexo, idade);
20     }
21     else if(menu == 2)
22     { nome = JOptionPane.showInputDialog("Nome para remover:");
```

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercício



```
23         if(rede.blockAmigo(nome))
24         { JOptionPane.showMessageDialog(null, "Removido");
25         }
26         else
27         { JOptionPane.showMessageDialog(null, "Nao encontrado");
28         }
29     }
30     else if(menu == 3)
31     { nome = JOptionPane.showInputDialog("Nome para"+
32       " procurar:");
33       int posicao = rede.procurarAmigo(nome);
34       if(posicao >= 0)
35       { JOptionPane.showMessageDialog(null, "Encontrado em "
36         +posicao);
37       }
38       else
39       { JOptionPane.showMessageDialog(null, "Nao encontrado");
40       }
41     }
42     else if(menu == 4)
43     { String mensagem = JOptionPane.showInputDialog(
44       "Mensagem:");
```

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercício



```
45         nome = JOptionPane.showInputDialog("Nome para enviar:");
46         if(rede.enviarMensagem(nome, mensagem))
47         {   JOptionPane.showMessageDialog(null,
48             "Mensagem enviada");
49         }
50         else
51         {   JOptionPane.showMessageDialog(null,
52             "Nao encontrado");
53         }
54     }
55     else if(menu == 5)
56     {   Amigo[] amigos = rede.procurarVelhos();
57         for(int i = 0; i < amigos.length; i++)
58         {   System.out.println(amigos[i]);
59         }
60     }
61     else if(menu == 6)
62     {
63     }
64     else if(menu == 7)
65     {   rede.listarAmigos();
66     }
67     else
68     {   JOptionPane.showMessageDialog(null,
69         "Opcao invalida");
70     }
71     }while(menu != 6);
72 }
73 }
74 }
```


Exercícios Extras

- Propostos pelo professor em aula, utilizando os conceitos abordados neste material...



Bibliografia Básica



- MILETTO, Evandro M.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, javascript e PHP (Tekne). Porto Alegre: Bookman, 2014. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/9788582601969>
- WINDER, Russel; GRAHAM, Roberts. Desenvolvendo Software em Java, 3ª edição. Rio de Janeiro: LTC, 2009. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9>
- DEITEL, Paul; DEITEL, Harvey. Java: how to program early objects. Hoboken, N. J: Pearson, c2018. 1234 p. ISBN 9780134743356.

Continua...

Bibliografia Básica (continuação)



- HORSTMANN, Cay S; CORNELL, Gary. Core Java. SCHAFRANSKI, Carlos (Trad.), FURMANKIEWICZ, Edson (Trad.). 8. ed. São Paulo: Pearson, 2010. v. 1. 383 p. ISBN 9788576053576.
- LIANG, Y. Daniel. Introduction to Java: programming and data structures comprehensive version. 11. ed. New York: Pearson, c2015. 1210 p. ISBN 9780134670942.
- TURINI, Rodrigo. Desbravando Java e orientação a objetos: um guia para o iniciante da linguagem. São Paulo: Casa do Código, [2017]. 222 p. (Caelum).

Bibliografia Complementar



- HORSTMANN, Cay. Conceitos de Computação com Java. Porto Alegre: Bookman, 2009. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/9788577804078>
- MACHADO, Rodrigo P.; FRANCO, Márcia H. I.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java (Tekne). Porto Alegre: Bookman, 2016. E-book. Referência Minha Biblioteca:
<https://integrada.minhabiblioteca.com.br/#/books/9788582603710>
- BARRY, Paul. Use a cabeça! Python. Rio de Janeiro: Alta Books, 2012. 458 p.
ISBN 9788576087434.

Continua...

ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Bibliografia Complementar (continuação)



- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTfulem Java na nuvem do Google. São Paulo: Novatec, c2015. 431 p.
ISBN 9788575224540.
- SILVA, Maurício Samy. JQuery: a biblioteca do programador. 3. ed. rev. e ampl. São Paulo: Novatec, 2014. 544 p.
ISBN 9788575223871.
- SUMMERFIELD, Mark. Programação em Python 3: uma introdução completa à linguagem Phython. Rio de Janeiro: Alta Books, 2012. 506 p.
ISBN 9788576083849.

Continua...

Bibliografia Complementar (continuação)



- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.

ECM251 – Linguagens de Programação I

Aula 09 – L1/1, L2/1 e L3/1

FIM

ECM251 – Linguagens de Programação I

Aula 09 – L1/2, L2/2 e L3/2

Engenharia da Computação – 3ª série

Polimorfismo, Vetores e ArrayList
(L1/2, L2/2 e L3/2)

2025

ECM251 – Linguagens de Programação I

Aula 09 – L1/2, L2/2 e L3/2

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Menezes*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro*.

Exercícios



1. Dadas as classes **Pessoa**, **FuncionarioAposentado** e **Funcionario**, sabendo que existem os atributos **salario**, **salarioAposentadoria**, **nome**, **idade** e **cargo** e sabendo que o método **categoria()**, dado abaixo, pertence à classe **Funcionario**, pede-se:
 - a) Construa as classes em Java, lembrando de considerar construtores e incluir alguns métodos, implementando o encapsulamento;
 - b) Monte um programa de acesso;
 - c) Modifique o código para que seja implementado o Polimorfismo (Sobrecarga e Sobreposição);

Exercícios



d) Dado o método *categoria()*:

```
public int categoria()
{  if(idade > 20)
    {  return 30;
      }
    else
    {  return 10;
      }
}
```

Exercícios



2. Dadas as classes **Shape**, **TwoDimensionalShape**, **Circle**, **Square**, **Triangle**, **ThreeDimensionalShape**, **Sphere**, **Cube** e **Tetrahedron**, apresentadas no diagrama de classes a seguir, implementar o polimorfismo (Sobrecarga e Sobreposição) para o cálculo de área e perímetro (duas dimensões) e de volume e área total das faces (três dimensões), para entradas, cálculos e apresentações com medidas em números inteiros, apenas, e em ponto flutuante, distintamente.

Criar a classe de teste, com a entrada e apresentação dos valores de todos os atributos envolvidos, respectivamente, via teclado e via monitor.

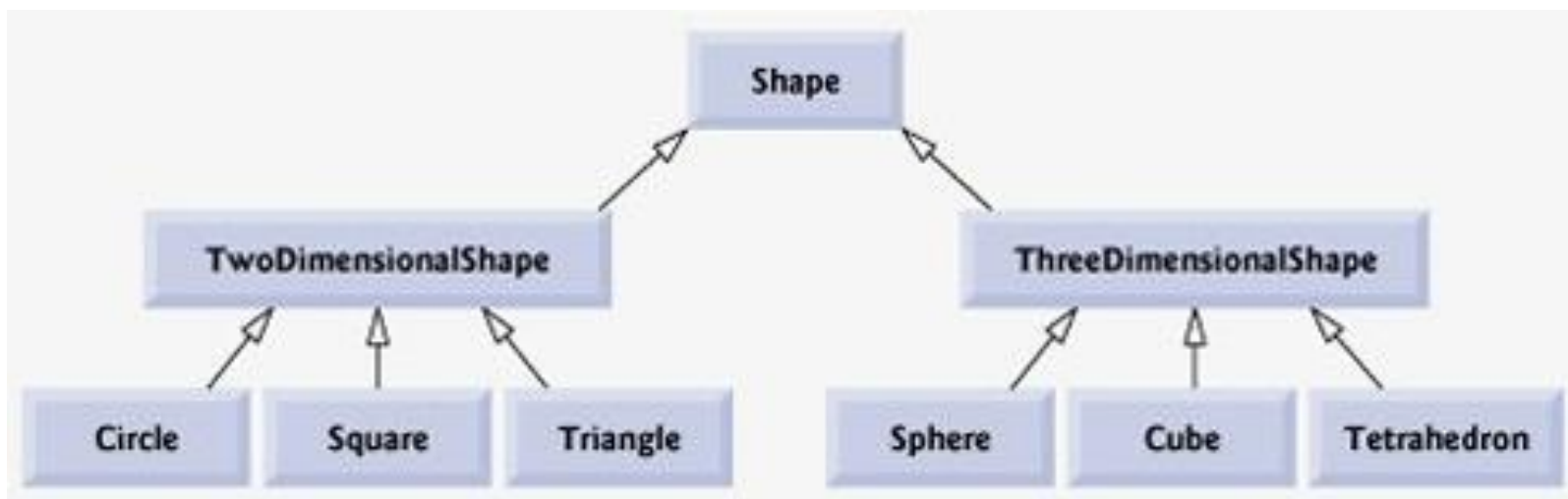
ECM251 – Linguagens de Programação I

Polimorfismo, Vetores e ArrayList

Exercícios



Diagrama de Classes dado para o exercício 2:



Exercícios



3. Crie a classe **BlocoDeNotas** que possui como atributo um **ArrayList<String>** chamado *notas*. Crie métodos para inserir, remover e buscar notas. Crie um método que imprima todas as notas;

Crie a classe **AppBloco**, com o método *main()* e um menu que:

- 1) Insira uma nota;
- 2) Remova uma nota;
- 3) Altere uma nota;
- 4) Listar todas as notas; e
- 5) Saia do sistema.

Exercícios



4. Você vai gerenciar um depósito e resolveu criar um sistema para isso. Para isso criou uma classe chamada **Caixa**, com os atributos **corredor** (*String*), **posicao** (*int*), **peso** (*double*) e **dono** (*String*), que armazena o nome do dono da caixa. Respeitou o encapsulamento e criou os métodos de acesso e os modificadores;

Exercícios



- Depois criou a classe **Deposito**, que contém um **ArrayList** de caixas. Fez um método para adicionar caixas e um para remover (pelo dono). Fez um método que encontra uma caixa pelo dono, retornando sua posição no **ArrayList** (ou -1 se não achar). E um método para mudar o corredor e a posição de uma caixa, que encontra a caixa pelo dono e altera seu atributos. Ele fez também um método que retorna um vetor com a(s) caixa(s) que pesam mais do que um valor passado por parâmetro;

Exercícios



- Para testar seu sistema fez uma classe **Teste** com o método ***main()*** que, usando o **JOptionPane**, possui um laço com as opções:

1. adiciona caixa;
2. remove caixa;
3. procura caixa;
4. muda caixa;
5. lista mais pesadas que 10.0; e
6. sair.

Exercícios



5. Crie a classe **Cliente** com os atributos privados do tipo *String* **nome** e **fone** e com o atributo inteiro **id**. Crie um construtor que receba valores para os atributos como parâmetros e os métodos de acesso e modificadores;
 - Crie a classe **BancoDeClientes** com um atributo privado do tipo **ArrayList<Cliente>** chamado **clientes**. Crie métodos para inserir um cliente, remover um cliente, alterar um cliente, listar os dados de um cliente e listar os dados de todos os clientes;

Exercícios



- Crie a classe **CadastroApp**, com o método *main()*, e que tenha um menu que insira um cliente, remova um cliente, altere um cliente, liste os dados de um cliente e liste os dados de todos os clientes.

Bibliografia (apoio)

- LOPES, ANITA. GARCIA, GUTO. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice-Hall (Pearson), 2010;
- BARNES, David J.; KÖLLING, Michael. Programação orientada a objetos com Java: uma introdução prática usando o BlueJ . 4. ed. São Paulo: Pearson Prentice Hall, 2009.

ECM251 – Linguagens de Programação I

Aula 09 – L1/2, L2/2 e L3/2

FIM