ELE2 – LPG3

Eletiva 2 – Linguagem de Programação 3 (JAVA)
Orientação a Objetos – Revisão Básica

Conceitos

- Classe
  - Objeto
  - Instância
- Construtor e Destrutor
- Encapsulamento (Membros)
  - Atributos
  - Métodos
- Herança
- Polimorfismo

Classe

- Abstrai um conjunto de objetos com características e comportamentos similares
- Define o comportamento de seus objetos através de:
  - Métodos
  - Atributos (estados possíveis destes objetos)
- "Descreve" a funcionalidade de seus objetos

Estrutura de um Classe

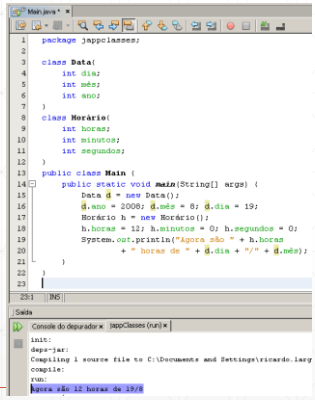
- Atributos
- Métodos
- Construtores
- Destrutor

- Sintaxe:

```
class NomeDaClasse{
    modificador tipo atributo
    modificador tipo método
    (tipo parâmetro1,
    tipo parâmetro2...,
    tipo parâmetroN){
        //Código...
    }
}
```

Atributos

- São declarações das classes que representam características de instância que possam assumir estados
- No exemplo, dia, mês e ano são atributos do tipo inteiro da classe Data, enquanto horas, minutos e segundos são atributos do tipo inteiro da classe Horário. Na classe Main, nenhum atributo foi declarado.



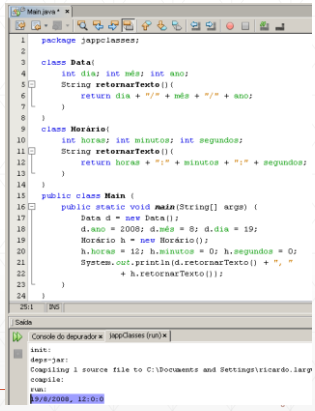
```
package jappclasses;

class Data{
    int dia;
    int mês;
    int ano;
}
class Horário{
    int horas;
    int minutos;
    int segundos;
}
public class Main {
    public static void main(String[] args) {
        Data d = new Data();
        d.ano = 2008; d.mês = 8; d.dia = 19;
        Horário h = new Horário();
        h.horas = 12; h.minutos = 0; h.segundos = 0;
        System.out.println("Agora são " + h.horas
                + " horas de " + d.dia + "/" + d.mês);
    }
}
```

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\ricardo.larg
compile:
run:
Agora são 12 horas de 19/8
```


Métodos

- São declarações das classes que representam ações que a mesma pode executar em função dos valores de seus atributos
- No exemplo, um método chamado retornarTexto() em cada classe retorna uma String representando a data e a hora em função dos valores dos atributos de suas respectivas instâncias



```
package jappclasses;

class Data{
    int dia; int mês; int ano;
    String retornarTexto(){
        return dia + "/" + mês + "/" + ano;
    }
}
class Horário{
    int horas; int minutos; int segundos;
    String retornarTexto(){
        return horas + ":" + minutos + ":" + segundos;
    }
}
public class Main {
    public static void main(String[] args) {
        Data d = new Data();
        d.ano = 2008; d.mês = 8; d.dia = 19;
        Horário h = new Horário();
        h.horas = 12; h.minutos = 0; h.segundos = 0;
        System.out.println(d.retornarTexto() + ", "
                + h.retornarTexto());
    }
}
```

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\ricardo.larg
compile:
run:
19/8/2008, 12:0:0
```


Objetos e Instâncias

- Objeto é uma entidade que pode ser física, conceitual ou de software. Ou seja, uma representação genérica.
- Instância é usada com o sentido de exemplo. É a concretização da classe. Ou seja, são os objetos de fato criados e ocupando espaço na memória.

Construtor

- Método especial. É disparado automaticamente na instanciação da classe
- Utilizamos ele basicamente para inicializar os estados dos atributos
- No exemplo a seguir, foram criados dois construtores. Cada um deles é chamado de acordo com a criação da instância, pela lista de argumentos

Construtor - Exemplo

```
1  package jappclasses;
2
3  class Data{
4      int dia; int mês; int ano;
5      Data(){
6          dia = 0; mês = 0; ano = 0;
7      }
8      Data(int dia, int mês, int ano){
9          this.dia = dia; this.mês = mês; this.ano = ano;
10     }
11     public String retornarTexto(){
12         return dia + "/" + mês + "/" + ano;
13     }
14 }
15 public class Main {
16     public static void main(String[] args) {
17         Data d1 = new Data(19, 8, 2008);
18         Data d2 = new Data();
19         d2.ano = 2008; d2.mês = 8; d2.dia = 20;
20         System.out.println("Data 1: " + d1.retornarTexto());
21         System.out.println("Data 2: " + d2.retornarTexto());
22     }
23 }
```

```
Saída - jappClasses (run)
deps-jar:
Compiling 1 source
compile:
run:
Data 1: 19/8/2008
Data 2: 20/8/2008
```


Destrutor

- O método Destrutor não pode ser sobrecarregado porque não aceita parâmetros e é executado quando o objeto é excluído.

```
obj.finalize();
```

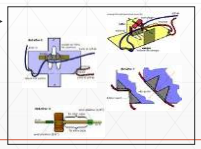

Encapsulamento

- Significa limitação de acesso aos membros da classe
- Utiliza-se de modificadores de acesso, tais como:
  - Public: Acessível a partir de qualquer outra classe, independente do package
  - Protected: Acessível a partir de qualquer classe derivada, no mesmo package
  - Private: Acessível apenas dentro da própria classe
  - Static: Membro compartilhado com todas as instâncias da classe
  - Final: Constante da classe
  - Friendly: Acessível a partir de qualquer outra classe do mesmo package

Encapsulamento

- Mecanismo que permite separar detalhes de funcionamento – características (atributos) e funções (métodos) – de sua interface.
- Exemplo: Para utilizarmos um liquidificador, não precisamos saber detalhes de seu funcionamento. A única interface que conhecemos são seus botões.

Classe Liquidicador →



Instância de um objeto de liquidificador: Muito mais fácil de usar.

Encapsulamento - Atributo

- São características (variáveis) da classe e podem ser divididos em dois tipos básicos:
  - Atributos de Instância: "determinam o estado de cada objeto."
  - Atributos de Classe: "possui um estado que é compartilhados por todos os objetos de uma classe."

```
class CartaDeBaralho{
        public char Simbolo;
        public image Naipe;
        public static ListaDeSimbolos[] = {0,1,2,3,4,5,6,7,8,9,J,Q,K,A};
        public static ListaDeNaipes[] = { ♥ , ♠ , ♦ , ♣ };
}
```

Atributos de Instância

Atributos de Classe

13

Encapsulamento - Método

- É uma rotina que é executada por um objeto ao receber uma mensagem: "É a descrição da forma como o objeto realiza uma tarefa."
- Os métodos determinam o comportamento dos objetos de uma classe e são análogos à funções ou procedimentos da programação estruturada.

```
class Chachorrro{
        //Atributos de instância:
        public String Nome;
        public Color CorDoPelo;
        public String Raça;
        :
        :
        //Atributos de classe:
        pubclic static ListaDeRaças[]
                = {Doberman, Pit Bull, ...};
        :
        :
        //Métodos:
        public void Correr(int Direção){
                //Código-fonte
        }
        public void Sentar(){
                //Código-fonte
        }
        public void Parar(){
                //Código-fonte
        }
        :
        :
}
```

14

Exemplo de encapsulamento

Main.java

```
1  package jappclasses;
2
3  class Data{
4      private int dia;
5      public int getDia() { return dia; }
6      public void setDia(int dia) {
7          if(dia > 0 && dia < 32) this.dia = dia;
8      }
9      private int mês;
10     public int getMês() { return mês; }
11     public void setMês(int mês) {
12         if(mês > 0 && mês < 13) this.mês = mês;
13     }
14     private int ano;
15     public int getAno() { return ano; }
16     public void setAno(int ano) { this.ano = ano; }
17     public String retornarTexto(){ return dia + "/" + mês + "/" + ano; }
18 }
19 public class Main {
20     public static void main(String[] args) {
21         Data d = new Data();
22         d.setAno(2008); d.setMês(8); d.setDia(19);
23         System.out.println("Data: " + d.retornarTexto());
24     }
25 }
```

Saída - jappClasses (run)

```
init:
deps-jar:
Compiling 1 sourc
compile:
run:
Data: 19/8/2008
```

15

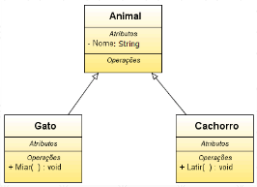
Herança - Definição

- Conhecida também como Generalização;
- Definição de uma classe a partir de outra, herdando os seus membros;
- A classe herdeira é chamada subclasse;
- A classe herdada é chamada superclasse;
- Pode ser utilizado em cascata, criando hierarquias com várias gerações de classes;
- Permite que as classes compartilhem atributos e operações baseados em um relacionamento.

16

Subclasse e superclasse

- Superclasse é a classe herdada. Ou seja, a classe Pai;
- Subclasse é a classe herdeira. Ou seja, a classe Filha;
- No exemplo, animal é a superclasse;
- No exemplo, cachorro e gato são subclasses.



17

Exemplo

```
public class Pessoa {
    /** Creates a new instance of Pessoa */
    public Pessoa() {
    }
    public String Nome;
    public Date nascimento = new Date();
}
```

```
public class Aluno extends Pessoa{
    /** Creates a new instance of Aluno */
    public Aluno() {
        disciplinas = new Vector();
    }
    public int matrícula;
    public Vector disciplinas;
}
```

```
public class Professor extends Pessoa{
    /** Creates a new instance of Professor */
    public Professor() {
        disciplinas = new Vector();
    }
    public float salário;
    public Vector disciplinas;
    public Date dataDeAdmissão = new Date();
}
```

18

Exemplo II

MeuForm.java

```
1  package br.com.fatecpg.heranca;
2
3  public class MeuForm extends javax.swing.JFrame {
4      javax.swing.JLabel MeuFormJLabel;
5
6      public MeuForm(){
7          this.setTitle("Meu Formulário JFrame");
8          MeuFormJLabel = new javax.swing.JLabel();
9          MeuFormJLabel.setText("Meu Formulário");
10         this.add(MeuFormJLabel);
11         this.setSize(300, 50);
12     }
13 }
```

Main.java

```
1  package br.com.fatecpg.heranca;
2
3  public class Main {
4      public static void main(String[] args) {
5          MeuForm f = new MeuForm();
6          f.setVisible(true);
7      }
8  }
```

Polimorfismo

- Grego: “muitas formas“
  - *poli* = muitas, *morphos* = formas
- Permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam;
- Um mesmo método pode apresentar várias formas, de acordo com seu contexto;

20

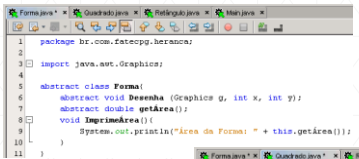
Polimorfismo

- Permite que a semântica de uma interface seja efetivamente separada da implementação que a representa;
- Basicamente, override e overload;
- Benefícios:
  - Clareza e manutenção do código;
  - Divisão da complexidade;
  - Aplicações flexíveis.

Sobreposição (Override)

- Utilizamos quando queremos reescrever um método na classe Filha que já existe e está implementado na classe Pai;
- Podemos chamar o método pai (se for desejado) através da palavra reservada super;
- Também chamado de sobrescrita ou polimorfismo vertical.

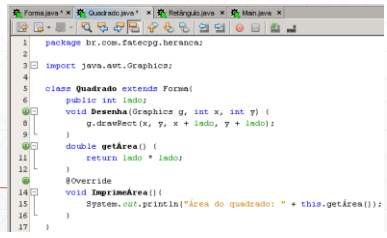
Exemplo



```
package br.com.fatecpg.heranca;

import java.awt.Graphics;

abstract class Forma{
    abstract void Desenha (Graphics g, int x, int y);
    abstract double getÁrea();
    void ImprimeÁrea(){
        System.out.println("Área da Forma: " + this.getÁrea());
    }
}
```

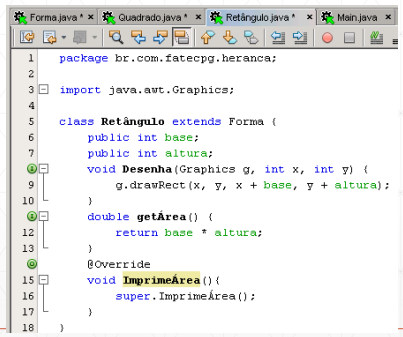


```
package br.com.fatecpg.heranca;

import java.awt.Graphics;

class Quadrado extends Forma{
    public int lado;
    void Desenha(Graphics g, int x, int y) {
        g.drawRect(x, y, x + lado, y + lado);
    }
    double getÁrea() {
        return lado * lado;
    }
    @Override
    void ImprimeÁrea(){
        System.out.println("Área do quadrado: " + this.getÁrea());
    }
}
```

Exemplo (utilizando o "super.")



```
package br.com.fatecpg.heranca;

import java.awt.Graphics;

class Retângulo extends Forma {
    public int base;
    public int altura;
    void Desenha(Graphics g, int x, int y) {
        g.drawRect(x, y, x + base, y + altura);
    }
    double getÁrea() {
        return base * altura;
    }
    @Override
    void ImprimeÁrea(){
        super.ImprimeÁrea();
    }
}
```

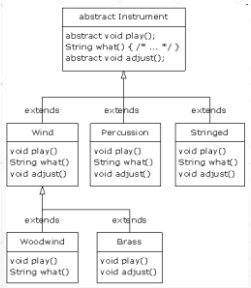
Sobrecarga (Overload)

- Quando há 2 ou mais métodos com o mesmo nome, mas com assinatura (e comportamento) diferente, declarados na mesma classe;
- Também conhecido como overload ou polimorfismo horizontal.

25

Classes abstratas

- São classes que abstraem um conjunto de características que podem ou não serem implementadas;
- Não podem ser instanciadas;
- "A implementamos quando sabemos o que fazer mas não como fazer".



26

Exemplo

Forma.java | Quadrado.java | Retângulo.java

```
1  package br.com.fatecpg.heranca;
2
3  import java.awt.Graphics;
4
5  abstract class Forma{
6      abstract void Desenha (Graphics g, int x, int y);
7      abstract double getÁrea();
8  }
```

Forma.java | Quadrado.java | Retângulo.java

```
1  package br.com.fatecpg.heranca;
2
3  import java.awt.Graphics;
4
5  class Quadrado extends Forma{
6      public int lado;
7      void Desenha(Graphics g, int x, int y) {
8          g.drawRect(x, y, x + lado, y + lado);
9      }
10     double getÁrea() {
11         return lado * lado;
12     }
13 }
```

Forma.java | Quadrado.java | Retângulo.java

```
1  package br.com.fatecpg.heranca;
2
3  import java.awt.Graphics;
4
5  class Retângulo extends Forma {
6      public int base;
7      public int altura;
8      void Desenha(Graphics g, int x, int y) {
9          g.drawRect(x, y, x + base, y + altura);
10     }
11     double getÁrea() {
12         return base * altura;
13     }
14 }
```

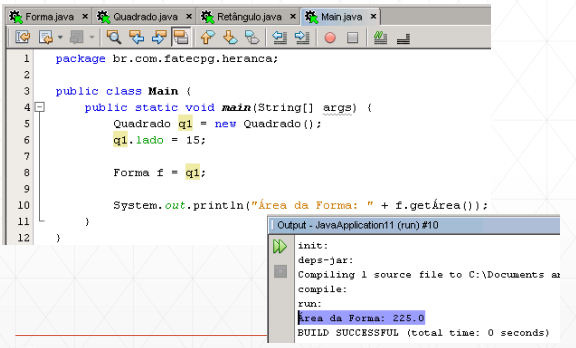
27

Upcasting e Downcasting

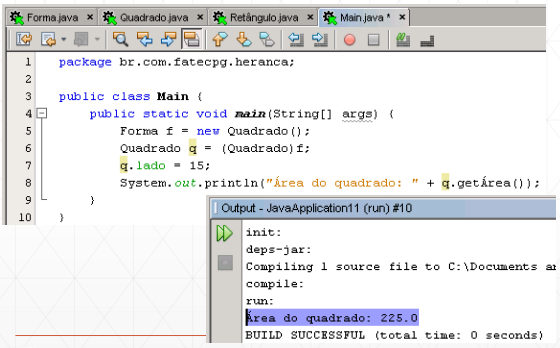
- Upcasting é quando transformamos (através de uma atribuição) um tipo de baixo para cima, na hierarquia;
- Downcasting é quando transformamos (também através de uma atribuição) um tipo de cima para baixo na hierarquia.

28

Exemplo de Upcasting



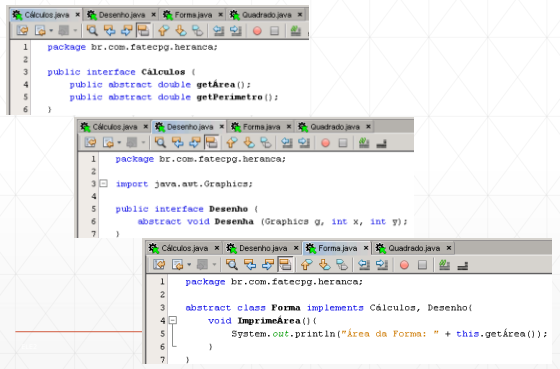
Exemplo de Downcasting



Interface

- Resolve (em partes) o problema da falta de Herança Múltipla;
- Uma classe pode implementar mais de uma interface;
- Modela um comportamento esperado;
- Na prática, uma interface assemelha-se a uma classe abstrata que contém apenas métodos abstratos.

Exemplo



Exemplo (Classe Quadrado)

Cálculos.java | Desenho.java | Forma.java | Quadrado.java

```
1  package br.com.fatecpg.heranca;
2
3  import java.awt.Graphics;
4
5  class Quadrado extends Forma{
6      public int lado;
7      public void Desenha(Graphics g, int x, int y) {
8          g.drawRect(x, y, x + lado, y + lado);
9      }
10     public double getÁrea() {
11         return lado * lado;
12     }
13     @Override
14     void ImprimeÁrea(){
15         System.out.println("Área do quadrado: " + this.getÁrea());
16     }
17     public double getPerimetro() {
18         return lado * 4;
19     }
20 }
```

33

Associação

- Vínculo que permite que objetos de uma ou mais classes se relacionem;
- É possível que um objeto convoque comportamentos e estados de outros objetos;
- As associações podem ser:
  - Unárias: Entre objetos de uma mesma classe;
  - Múltiplas: Entre mais de dois objetos.

34

Associação

- Cada associação possui características de:
  - Cardinalidade ou multiplicidade - determina quantos objetos no sistema são possíveis em cada vértice da associação;
  - Navegação - se é possível para cada objeto acessar outro objeto da mesma associação.

35

Exemplo de associação

- Basicamente podemos ter:
  - Agregação: quando duas ou mais classes formam outra classe. Neste caso, a ausência de uma delas não compromete a existência da classe resultante da associação.

```
package RegraNegocio;

public class Aula {
    private Aluno _aluno;
    private Materia _materia;
}
```

  - Composição: também ocorre quando duas ou mais classes compõem outra classe. Neste caso, porém, a ausência de uma das classes formadoras compromete a existência da classe resultante da associação, impedindo sua utilização.

```
package RegraNegocio;

public class Carro{
    private Chassi _chassi;
    private Motor _motor;
}
```

36

Exemplo prático

```
class Data{
    public int dia;
    public int mes;
    public int ano;
    public String getData(){
        return this.dia + "/" + this.mes + "/" + this.ano;
    }
}
class Pessoa{
    public String Nome;
    public Data nascimento = new Data();
}
public class Main {
    public static void main(String[] args) {
        // TODO code application logic here
        Pessoa p1 = new Pessoa();
        p1.Nome = "Roberval";
        p1.nascimento.ano = 2000;
        p1.nascimento.mes = 1;
        p1.nascimento.dia = 1;
        System.out.println(p1.Nome + " nasceu em " + p1.nascimento.getData());
    }
}
```

```
Output - JavaApplication24 (run)
init:
deps-jar:
Compiling 1 source file to C:\Documents s
compile:
run:
Roberval nasceu em 1/1/2000
BUILD SUCCESSFUL (total time: 0 seconds)
```

Outro exemplo

```
class Temperatura{
    public double grausCelsius;
    public double getCel(){return this.grausCelsius;}
    public double getFar(){return (9 / 5) * this.grausCelsius + 32;}
}
class CâmaraTérmica{
    public Temperatura minTemp = new Temperatura();
    public Temperatura maxTemp = new Temperatura();
    public CâmaraTérmica(int minTemp, int maxTemp){
        this.minTemp.grausCelsius = minTemp;
        this.maxTemp.grausCelsius = maxTemp;
    }
}
public class Main {
    public static void main(String[] args) {
        // TODO code application logic here
        CâmaraTérmica c12 = new CâmaraTérmica(56, 120);
        System.out.println(" A temperatura da câmara térmica deve ficar entre "
                + c12.minTemp.getCel() + ""C e " + c12.maxTemp.getCel() + ""C");
        System.out.println(" A temperatura da câmara térmica deve ficar entre "
                + c12.minTemp.getFar() + ""F e " + c12.maxTemp.getFar() + ""F");
    }
}
```

```
Output - JavaApplication24 (run)
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\ricardo.largues
compile:
run:
 A temperatura da câmara térmica deve ficar entre 56.0°C e 120.0°C
 A temperatura da câmara térmica deve ficar entre 88.0°F e 152.0°F
BUILD SUCCESSFUL (total time: 0 seconds)
```