Centro Tecnológico
Departamento de Informática

Prof. Vítor E. Silva Souza
http://www.inf.ufes.br/~vitorsouza

[Desenvolvimento OO com Java]

Classes abstratas e interfaces

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conteúdo do curso

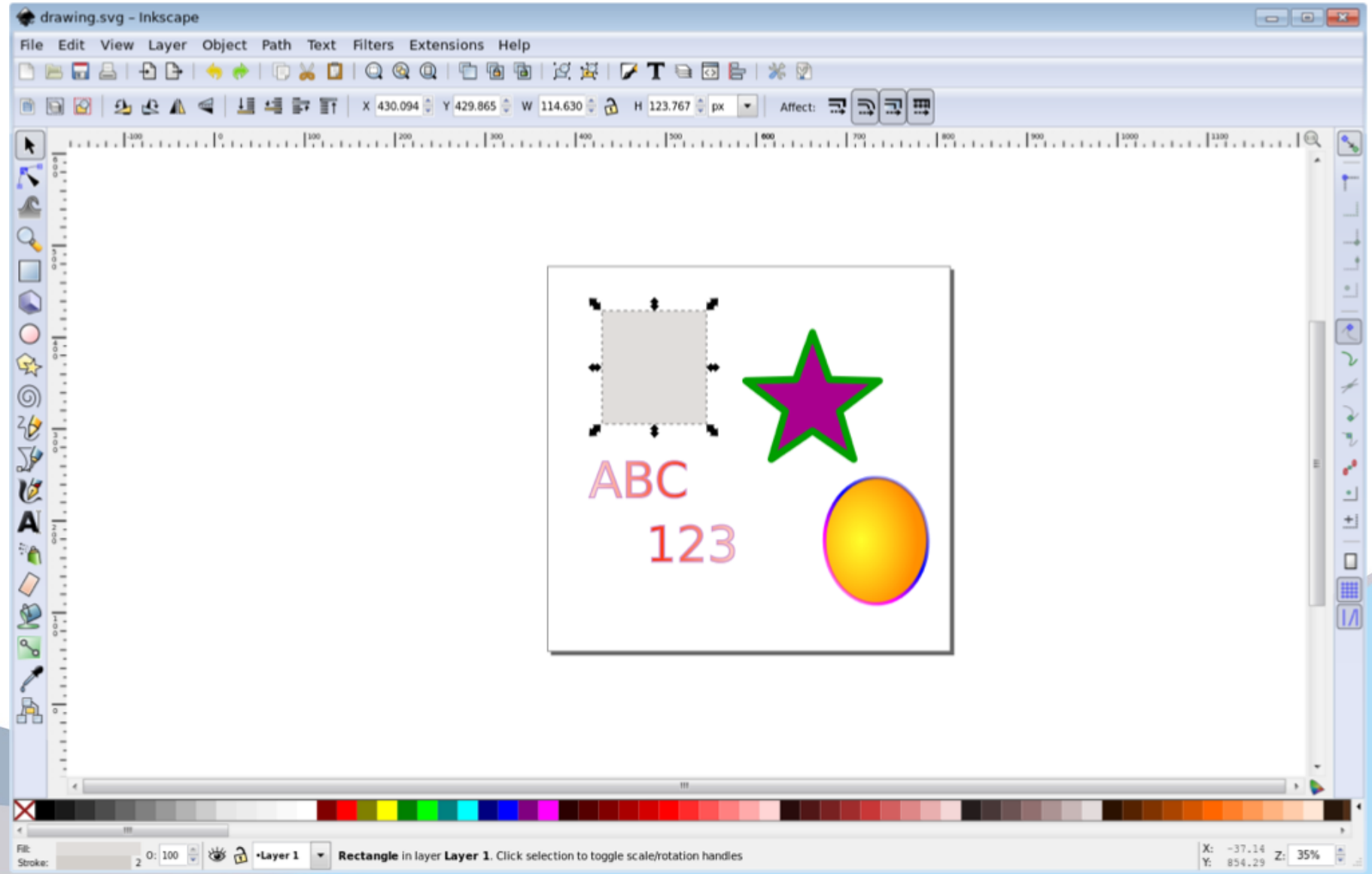
- O que é Java;
- Variáveis primitivas e controle de fluxo;
- Orientação a objetos básica;
- Um pouco de vetores;
- Modificadores de acesso e atributos de classe;

- Herança, reescrita e polimorfismo;
- Classes abstratas e interfaces;
- Exceções e controle de erros;
- Organizando suas classes;
- Utilitários da API Java.

Estes slides foram baseados na apostila do curso FJ-11: Java e Orientação a Objetos da Caelum e na apostila Programação Orientada a Objetos em Java do prof. Flávio Miguel Varejão.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplo: um aplicativo de desenho



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Classes e métodos abstratos

- Classes no topo da hierarquia podem ser muito gerais:
  - *O que é uma forma?*
  - *Como se desenha uma forma?*
  - *Como se aumenta uma forma?*
- Tais operações não fazem sentido. Queremos apenas definir que elas existam, mas não implementá-las;
- A solução: métodos abstratos.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Classes e métodos abstratos

- Uma classe que possui métodos abstratos deve ser declarada como abstrata:

```
abstract class Forma {
  public abstract void desenhar();
}

class Circulo extends Forma {
  @Override
  public void desenhar() {
    System.out.println("Círculo");
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

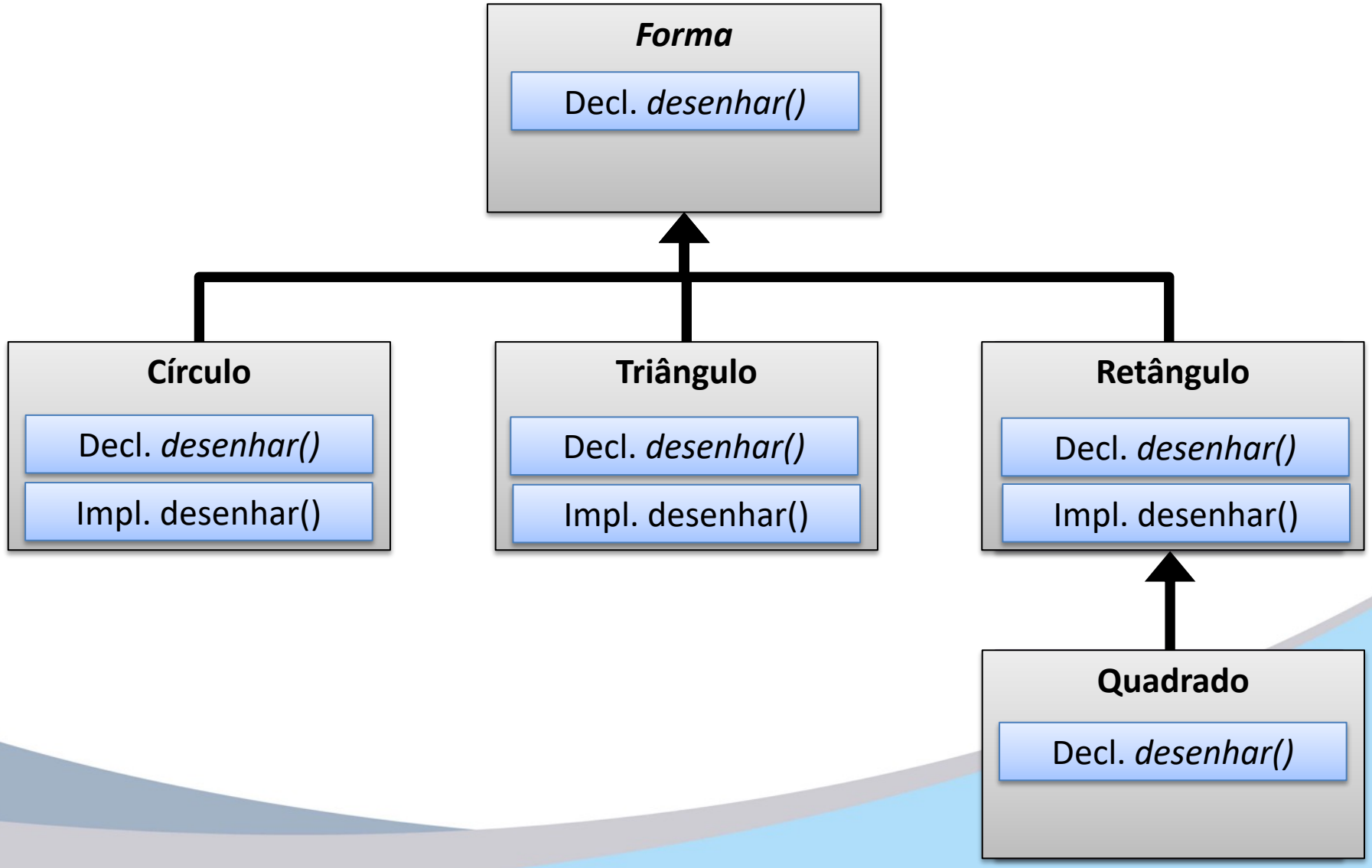
Classes abstratas

- Não permitem criação de instâncias (objetos):
  - *Um método abstrato não possui implementação, portanto não pode ser chamado.*
- Para ser útil, deve ser estendida:
  - *Suas subclasses devem implementar o método ou declararem-se como abstratas.*
- Servem para definir interfaces e prover algumas implementações comuns.

Polimorfismo!

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Construindo a hierarquia de formas



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Classes e métodos abstratos - Quiz

Métodos estáticos podem ser abstratos?	Não
Construtores podem ser abstratos?	Não
Classes abstratas podem ter construtores?	Sim

Lembre-se: construtores são chamados pelas subclasses!

Métodos abstratos podem ser privativos?	Não
Uma classe abstrata podem estender uma normal?	Sim
Posso ter uma classe abstrata sem nenhum método abstrato?	Sim

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Classes abstratas (puras) e concretas

```
// Classe abstrata pura.
abstract class Forma {
  public abstract void desenhar();
  public abstract void aumentar(int t);
}

// Classe abstrata.
abstract class Poligono extends Forma {
  private int lados;
  public Poligono(int lados) {
    this.lados = lados;
  }
  public int getLados() { return lados; }
  public abstract void pintar(int cor);
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Classes abstratas (puras) e concretas

```
// Classe concreta.
class Retangulo extends Poligono {
  public Retangulo() {
    super(4);
  }
  @Override public void desenhar() {
    System.out.println("Retangulo.desenhar");
  }
  @Override public void aumentar(int t) {
    System.out.println("Retangulo.aumentar");
  }
  @Override public void pintar(int cor) {
    System.out.println("Retangulo.pintar");
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Interfaces

- Uma classe abstrata é pura quando:
  - *Possui métodos abstratos;*
  - *Não possui métodos concretos;*
  - *Não possui atributos (não-static).*
- Java oferece a palavra reservada `interface`:
  - *Cria uma classe abstrata pura;*
  - *Chamaremos pelo nome de interface;*
  - *Ao conversar com outros programadores, cuidado para não confundir com "interface com o usuário".*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Interfaces

```
interface Forma {
  void desenhar();
  void aumentar(int t);
}

abstract class Poligono implements Forma {
  private int lados;

  public Poligono(int lados) {
    this.lados = lados;
  }

  public int getLados() { return lados; }
  public abstract void pintar(int cor);
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Interfaces

```
class Linha implements Forma {
  private double x1, y1, x2, y2;

  @Override
  public void desenhar() {
    /* ... */
  }

  @Override
  public void aumentar(int t) {
    /* ... */
  }
}
```

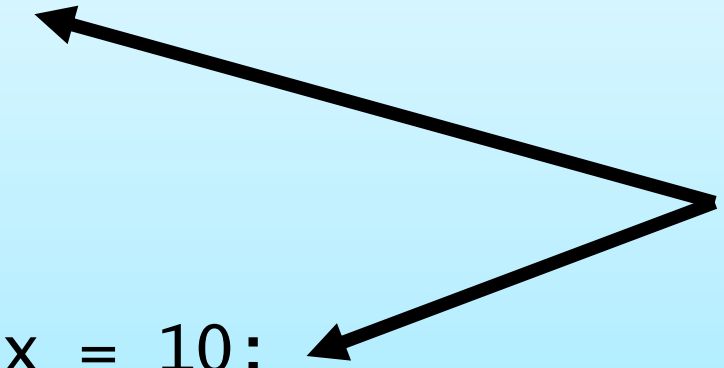
Tudo é público na interface

- Métodos definidos na interface são automaticamente públicos e abstratos;

Exceção: default methods (Java 8)

- Atributos definidos na interface são automaticamente públicos e estáticos.

```
interface Forma {
  int x = 10;
  void desenhar();
}

interface Forma {
  public static int x = 10;
  public abstract void desenhar();
}
```

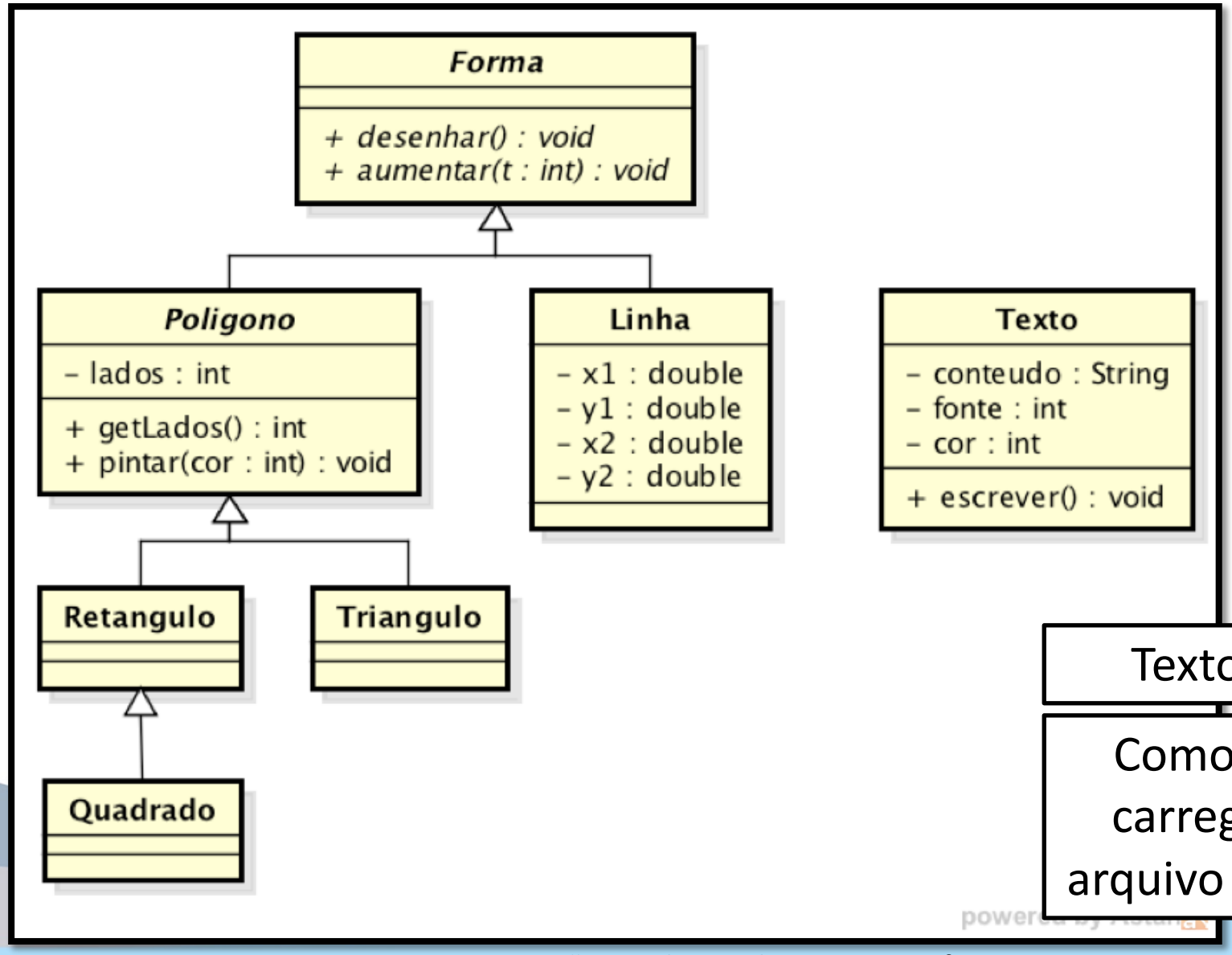


UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Por isso, cuidado com erros

```
interface Forma {
  void desenhar();
  void aumentar(int t);
}

class Linha implements Forma {
  // Erro: reduziu de público para priv. ao pacote!
  void desenhar() {
    /* ... */
  }

  // Erro: reduziu de público para privativo!
  private void aumentar(int t) {
    /* ... */
  }
}
```

Motivação



Texto não é forma, OK!

Como fica o método que carrega meu desenho do arquivo que eu salvei no HD?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Não-solução 1

```
public class AplicativoDesenho {
  private static void desenhar(Forma[] fs) {
    for (int i = 0; i < fs.length; i++)
      fs[i].desenhar();
  }
}
```

Texto não é forma! Esse método não serve...

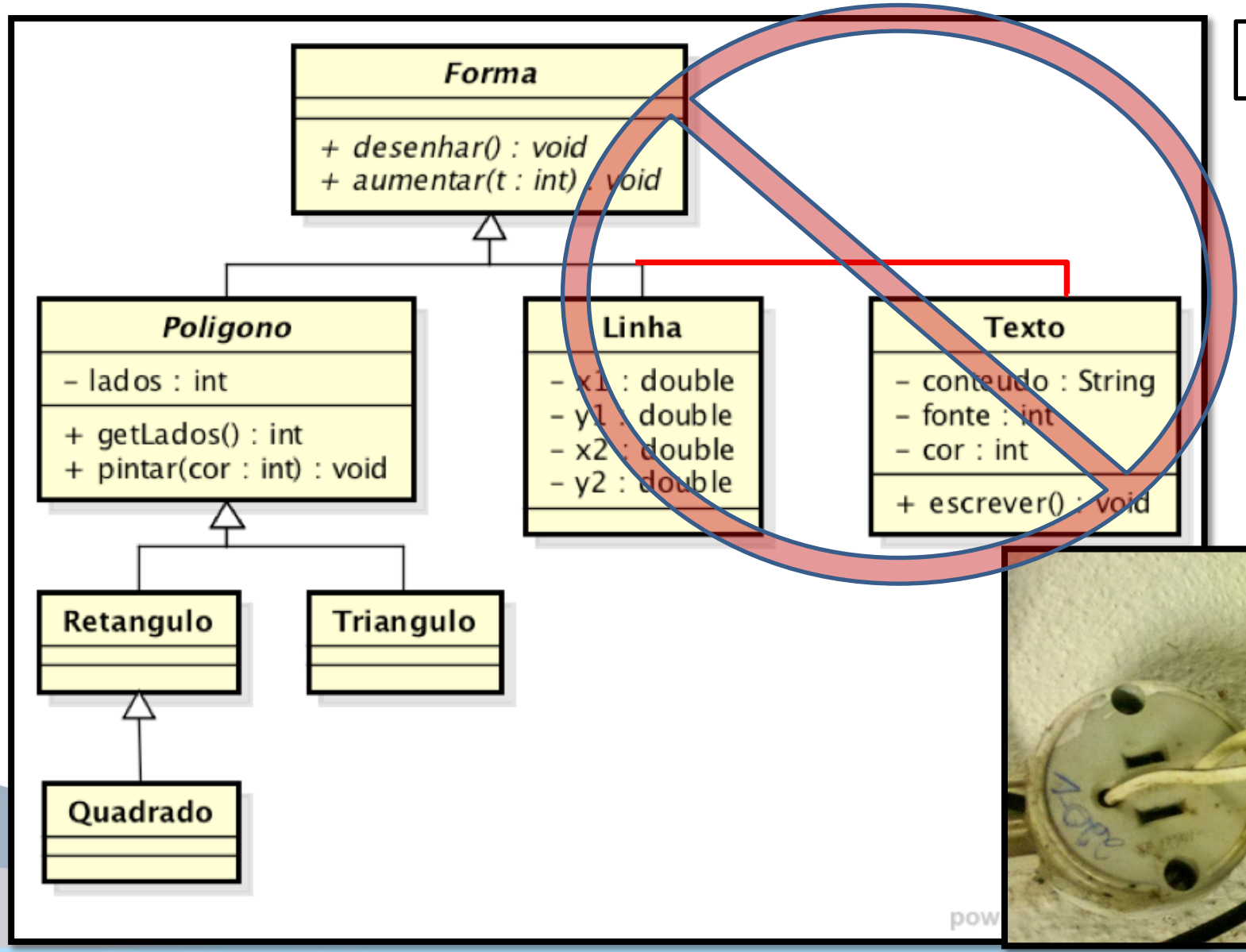
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Não-solução 2

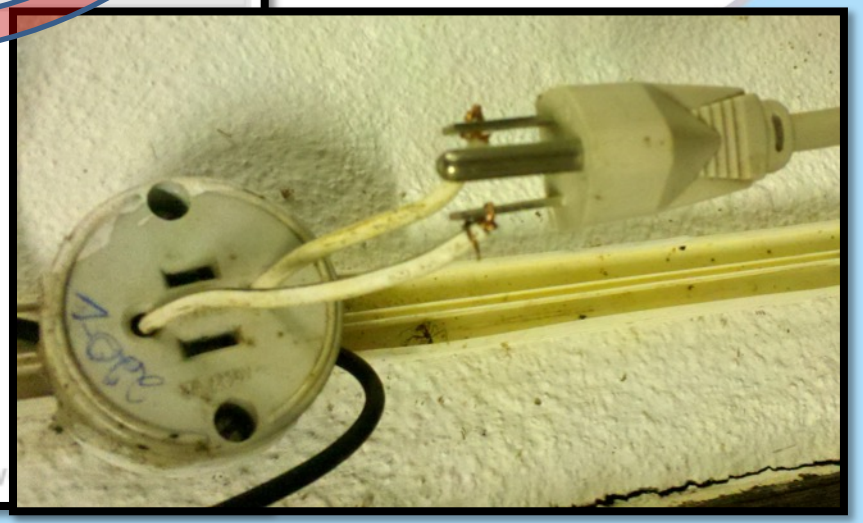
```
public class AplicativoDesenho {
  private static void desenhar(Forma[] fs) {
    for (int i = 0; i < fs.length; i++)
      fs[i].desenhar();
  }

  private static void desenhar(Texto[] ts) {
    for (int i = 0; i < ts.length; i++)
      ts[i].escrever();
  }
}
```

Já vimos, com polimorfismo, que essa não é a solução...

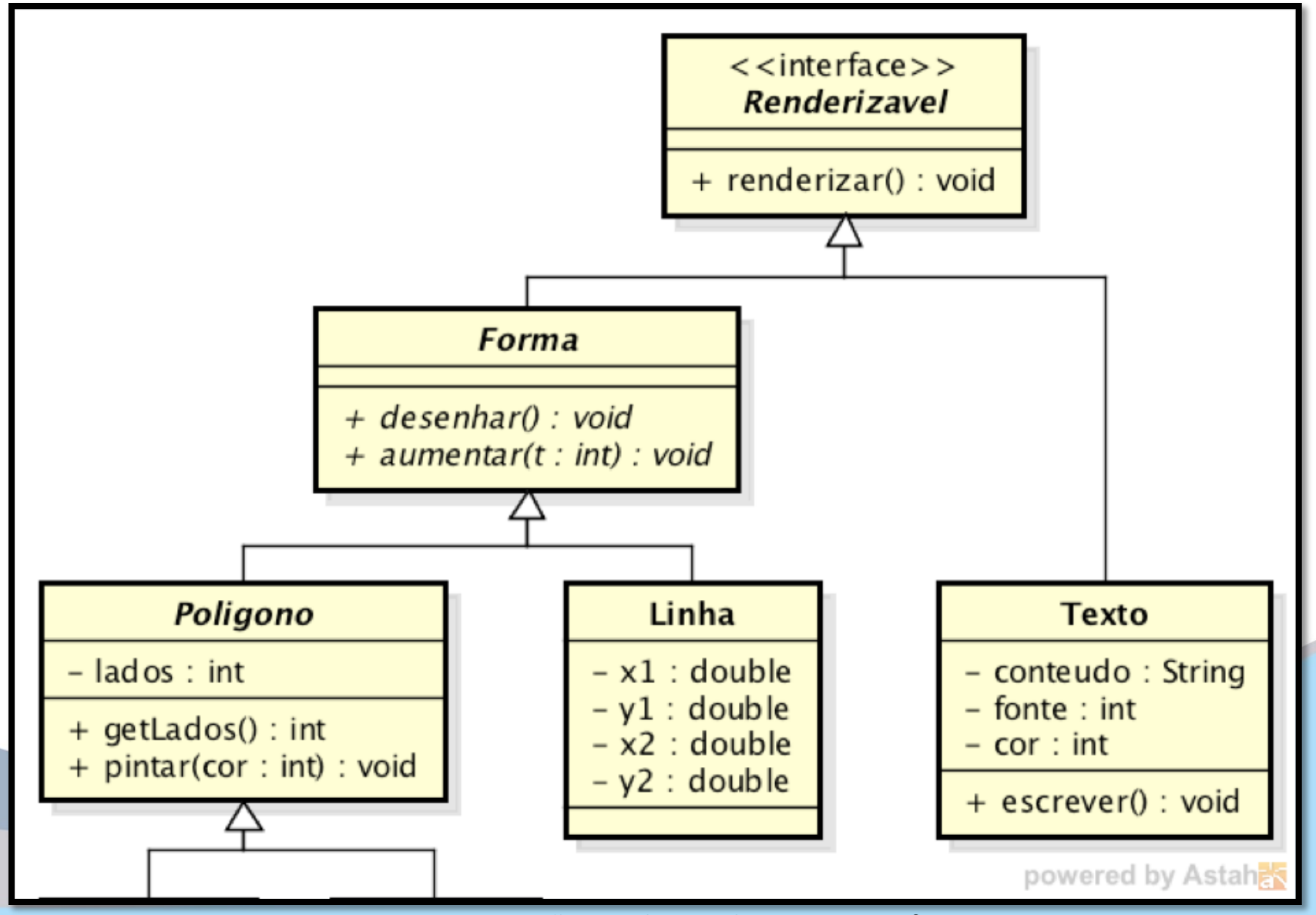
Não-solução 3



Já sei!



Solução: interfaces



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Solução: interfaces

```
abstract class Forma implements Renderizavel {
  /* ... */
  @Override
  public void renderizar() {
    desenhar();
  }
}

class Texto implements Renderizavel {
  /* ... */
  @Override
  public void renderizar() {
    escrever();
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Solução: interfaces (alternativa)

```
interface Forma extends Renderizavel {
  /* ... */

  // As diferentes implementações de forma agora tem
  // que implementar renderizar() e não desenhar().
}

class Texto implements Renderizavel {
  /* ... */

  @Override
  public void renderizar() {
    escrever();
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Solução: interfaces

- O polimorfismo se amplia: mais um modo de referenciar uma forma: `Renderizavel`;
- `AplicativoDesenho` não precisa saber a classe real do objeto, apenas que ele implementa a interface;
  - *Se implementa a interface, ele implementa o método `renderizar()`!*
- Novas classes podem ser renderizáveis!

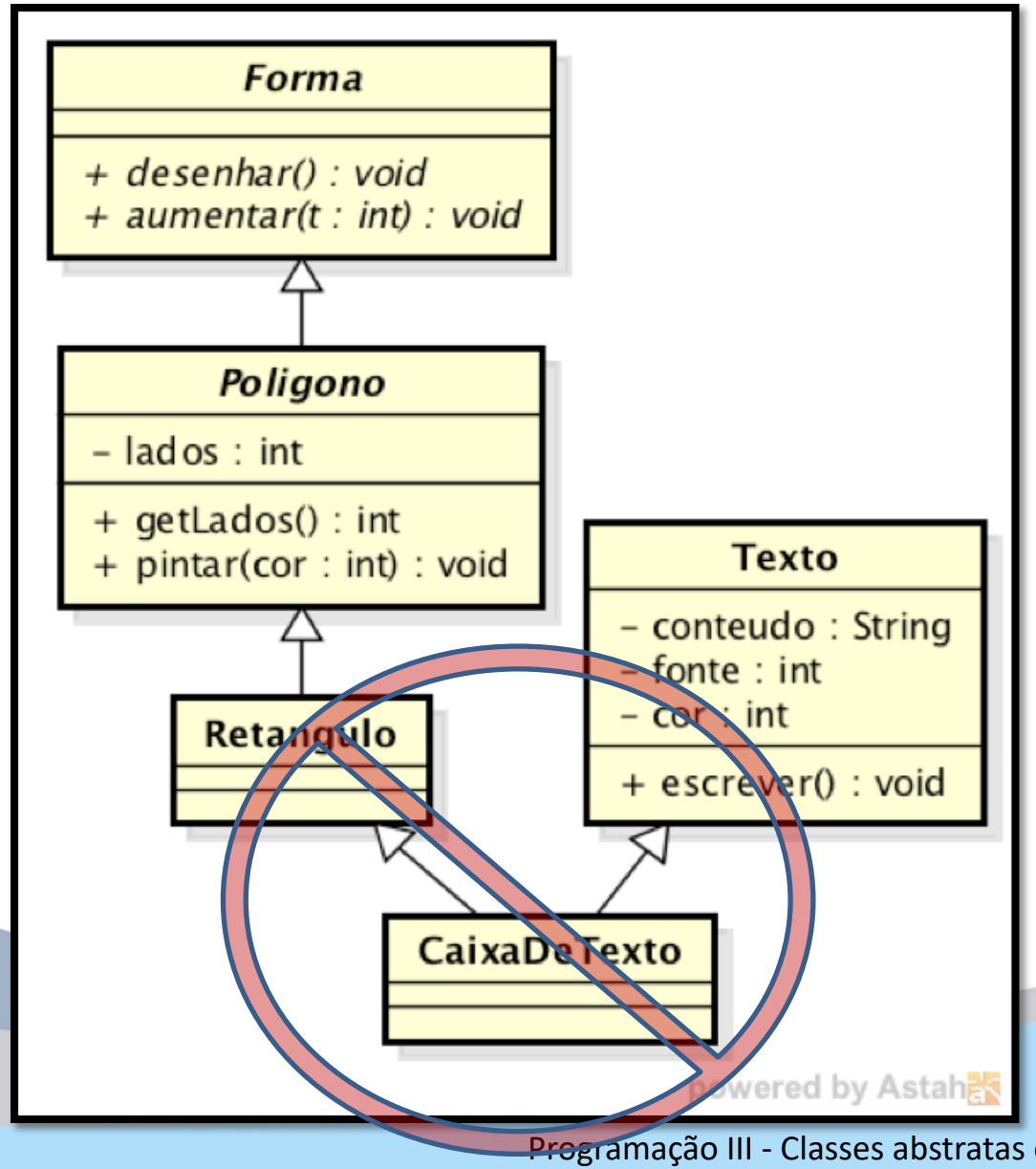
```
public class AplicativoDesenho {
  private static void desenhar(Renderizavel[] fs) {
    for (int i = 0; i < fs.length; i++)
      fs[i].renderizar();
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Interface = contrato

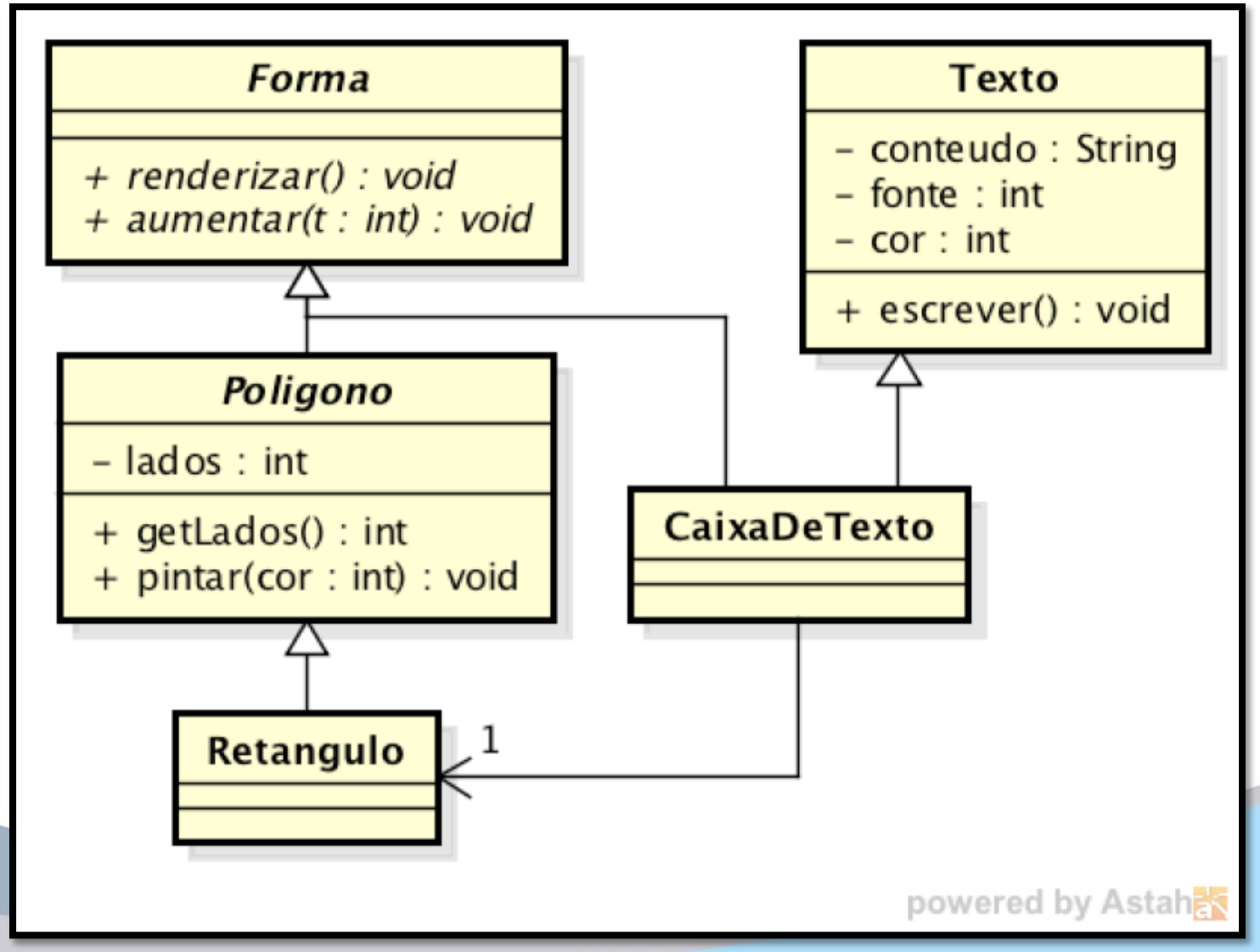
- `Renderizavel` define que todas as classes que a implementam saibam se `renderizar()` – "o que";
  - *A implementação define "o como";*
- Define um contrato: "quem desejar ser renderizável precisa saber se `renderizar()`";
  - *A classe que quiser, assina o contrato e se responsabiliza a cumpri-lo;*
- Programe voltado a interfaces e não a implementações (mas sem exageros).

Herança múltipla em Java



Pode isso, Arnaldo?

Herança múltipla em Java



A regra é clara...

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Herança múltipla em Java

```
class CaixaDeTexto extends Texto implements Forma {
  private Retangulo caixa;
  /* ... */
  public CaixaDeTexto() {
    // Parâmetros foram omitidos para simplificar...
    caixa = new Retangulo();
  }

  public void renderizar() {
    // Desenha a caixa.
    caixa.renderizar();
    // Escreve o texto.
    escrever();
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

A interface Comparable

- Um exemplo de interface na API Java é a interface `Comparable`;
- Define o método `compareTo(Object obj)`:
  - *Compara o objeto atual (`this`) com o objeto informado (`obj`);*
  - *Retorna 0 se `this` = `obj`;*
  - *Retorna um número negativo se `this` < `obj`;*
  - *Retorna um número positivo se `this` > `obj`.*
- Métodos genéricos a utilizam para ordenar coleções de elementos.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

A interface Comparable

```
class Valor implements Comparable {
  int valor;

  public Valor(int v) { valor = v; }

  @Override
  public int compareTo(Object obj) {
    return valor - ((Valor)obj).valor;
  }

  @Override
  public String toString() {
    return "" + valor;
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

A interface Comparable

```
public class Teste {
  static void imprimir(Object[] vetor) {
    for (int i = 0; i < vetor.length; i++)
      System.out.print(vetor[i] + "; ");
    System.out.println();
  }
  public static void main(String[] args) {
    Valor[] vetor = new Valor[] {
      new Valor(10), new Valor(3),
      new Valor(15), new Valor(7)
    };
    imprimir(vetor);    // 10; 3; 15; 7;
    Arrays.sort(vetor);
    imprimir(vetor);    // 3; 7; 10; 15;
  }
}
```

O mecanismo de RTTI

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Polimorfismo e extensão

- Com polimorfismo, podemos esquecer a classe de um objeto e trabalhar com a superclasse:
  - *A interface de ambas é a mesma;*
  - *A amarração dinâmica garante que o método da classe correta será executado.*
- O que acontece se a subclasse estende a superclasse (adiciona mais funcionalidade)?
- Se a superclasse não possui aquela funcionalidade, não podemos chamá-la!

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Polimorfismo e extensão

```
interface Animal {
  void comer();
}

class Cachorro implements Animal {
  @Override public void comer() {
    System.out.println("Comendo um osso...");
  }

  public void latir() {
    System.out.println("Au Au!");
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Polimorfismo e extensão

```
class Gato implements Animal {
  @Override public void comer() {
    System.out.println("Comendo um peixe...");
  }

  public void miar() {
    System.out.println("Miau!");
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Polimorfismo e extensão

```
public class Teste {
  public static void main(String[] args) {
    Animal[] vet = new Animal[] {
        new Cachorro(), new Gato(),
        new Gato(), new Cachorro()
    };

    for (int i = 0; i < vet.length; i++) {
      vet[i].comer();
      // Erro: vet[i].latir();
    }
  }
}
```

#comofas?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Estreitamento (*downcast*)

- Precisamos relembrar a classe específica do objeto para chamarmos métodos que não estão na interface da superclasse;
- Para isso faremos estreitamento:

Ampliação (upcast)	Estreitamento (downcast)
`int` para `long`	`long` para `int`
`float` para `double`	`double` para `float`
`Cachorro` para `Animal`	`Animal` para `Cachorro`
`Gato` para `Animal`	`Animal` para `Gato`

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

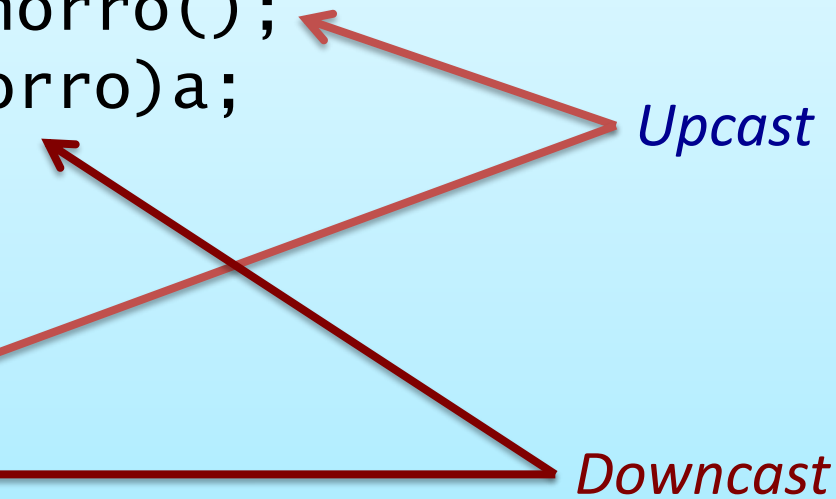
Upcast vs. *downcast*

- Ampliação é automática e livre de erros:
  - *A classe base não pode possuir uma interface maior do que a classe derivada;*
  - *Não é necessário explicitar o upcast.*
- Estreitamento é manual e pode causar erros:
  - *A classe base pode ter várias subclasses e você está convertendo para a classe errada;*
  - *É necessário explicitar o downcast;*
  - *Pode lançar um erro (`ClassCastException`);*
  - *Pode haver perda de informação (tipos primitivos).*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Upcast vs. *downcast*

```
public class Teste {
  public static void main(String[] args) {
    Animal a = new Cachorro();
    Cachorro c = (Cachorro)a;
    c.latir();

    // Forma resumida:
    a = new Gato();
    ((Gato)a).miar();
  }
}
```



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

RTTI: Run-Time Type Identification

- O mecanismo que verifica o tipo de um objeto em tempo de execução chama-se RTTI;
- RTTI = *Run-Time Type Identification* ou Identificação de Tipos em Tempo de Execução;
- Este mecanismo garante que as conversões são sempre seguras;
- Não permite que um objeto seja convertido para uma classe inválida:
  - *Fora da hierarquia: erro de compilação;*
  - *Dentro da hierarquia: erro de execução.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

RTTI: Run-Time Type Identification

```
public class Teste {
  public static void main(String[] args) {
    Animal a = new Cachorro();

    // Sem erro nenhum:
    Cachorro c = (Cachorro)a;

    // Erro de execução (ClassCastException):
    Gato g = (Gato)a;

    // Erro de compilação:
    String s = (String)a;
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

O operador `instanceof`

- O mecanismo de RTTI permite que você consulte se um objeto é de uma determinada classe;
- Operador `instanceof`:
  - *Sintaxe: <objeto> instanceof <Classe>*
  - *Retorna `true` se o objeto for instância (direta ou indireta) da classe especificada;*
  - *Retorna `false` caso contrário.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

O operador instanceof

```
public class Teste {
  public static void main(String[] args) {
    Animal[] vet = new Animal[] {
        new Cachorro(), new Gato(),
        new Gato(), new Cachorro()
    };

    for (int i = 0; i < vet.length; i++) {
      if (vet[i] instanceof Cachorro)
        ((Cachorro)vet[i]).latir();
      else if (vet[i] instanceof Gato)
        ((Gato)vet[i]).miar();
    }
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

O uso de `instanceof` deve ser raro

- Não é uma boa prática usar `instanceof`:
  - *Use polimorfismo;*
  - *Use classes genéricas (veremos adiante).*
- Use `instanceof` apenas quando não há outra solução.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Trocando `instanceof` por polimorfismo

```
interface Animal {
  void comer();
  void falar();
}

class Cachorro extends Animal {
  @Override public void comer() { /* ... */ }
  @Override public void falar() { /* ... */ }
}

class Gato extends Animal {
  @Override public void comer() { /* ... */ }
  @Override public void falar() { /* ... */ }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Trocando instanceof por genéricos

```
public class Teste {
  public static void main(String[] args) {
    Cachorro c;

    List lista = new ArrayList();
    lista.add(new Cachorro());
    Object o = lista.get(0);
    if (o instanceof Cachorro) c = (Cachorro)o;

    // Com genéricos.
    List<Cachorro> listaGen;
    listaGen = new ArrayList<Cachorro>();
    listaGen.add(new Cachorro());
    c = listaGen.get(0);
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exercitar é fundamental

- Apostila FJ-11 da Caelum – Classes Abstratas:
  - *Seção 9.6, página 121 (conta corrente);*
  - *Seção 9.7, página 123 (desafios);*

- Apostila FJ-11 da Caelum – Interfaces:
  - *Seção 10.5, página 134 (formas, conta corrente);*
  - *Seção 10.6, página 138 (exercícios avançados);*
  - *Seção 10.7, página 139 (discussão).*