Curso de Java

Orientação a objetos e a Linguagem JAVA

- A linguagem Java e a máquina virtual
- Objetos e Classes
- Encapsulamento, Herança e Polimorfismo
- Primeiro Exemplo

- Principais características:
  - Orientação ao objetos
  - Exceções
  - '*multi-thread*'
  - Liberação automática de memória
  - Independência de Plataforma

Ao se aprender uma nova linguagem de programação, normalmente se estuda

- sintaxe
- tipos de dados (básicos, compostos, definidos pelo usuário)
- operadores
- comandos de controle
- funções
- entrada e saída
- bibliotecas

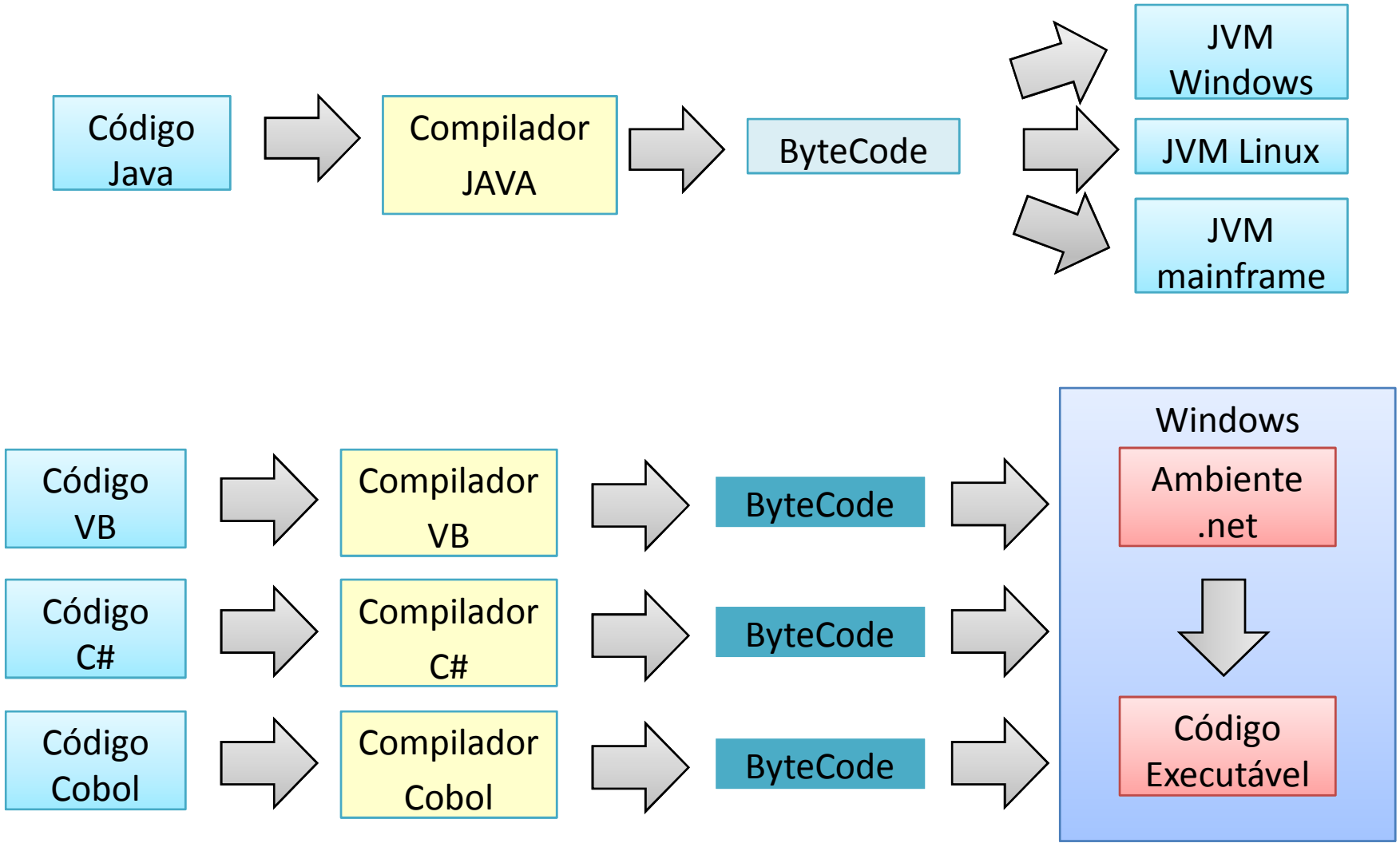
Essas características são comuns à maioria das linguagens ditas "de uso geral". Esse é o caso de C, C++, Pascal, VB e também de C#.

- Um programa, para ser executado, precisa de um conjunto de 'serviços de apoio' que normalmente são oferecidos pelo sistema operacional.
- Em VB, por exemplo, as funções para criação da interface de usuário são exemplos de funções que acessam esses 'serviços de apoio'.
- Normalmente os serviços oferecidos pelo sistema operacional são disponibilizados através de funções de biblioteca.

- O conjunto de serviços oferecidos pelo sistema operacional
  - não é padronizado
  - afeta a estrutura das aplicações
- Essa dependência compromete a portabilidade dos sistemas.
- Por essa razão, alguns fabricantes têm trabalhado na linha de oferecer um 'ambiente de execução' aos programas que seja independente da máquina e do sistema operacional.

- A máquina virtual Java, JVM, constitui o 'ambiente' de execução para programas Java.
- É através da JVM que se consegue a independência de plataforma: para cada plataforma (hardware + sistema operacional) é implementada uma versão da JVM.
- Uma vez compilado, o código pode ser executado em qualquer JVM.

JVM x Ambiente .net

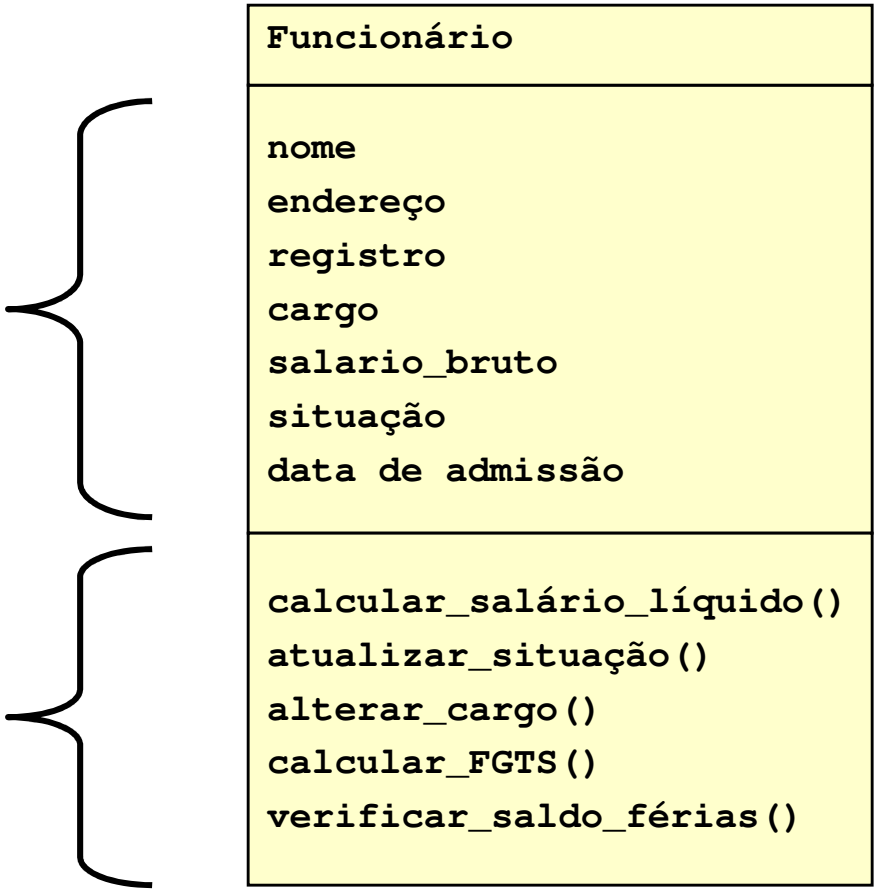


- Objetos reunem num único elemento as *características estáticas* e *comportamentais*.
- Características estáticas são representadas como *dados* associados ao *objeto*. Elas descrevem o *estado* atual do objeto.
- Características dinâmicas são descritas através de operações ou *métodos* executados pelo objeto.
- *Estado* e *comportamento* são partes integrantes do objeto.

- Um exemplo

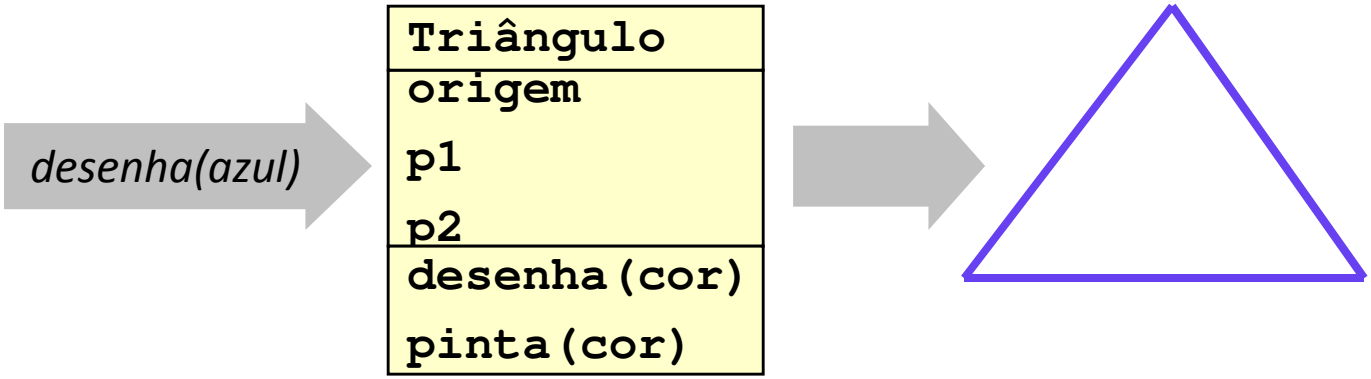
Dados: descrevem o ‘estado atual’ do objeto.

Operações: métodos que descrevem os aspectos dinâmicos do objeto.

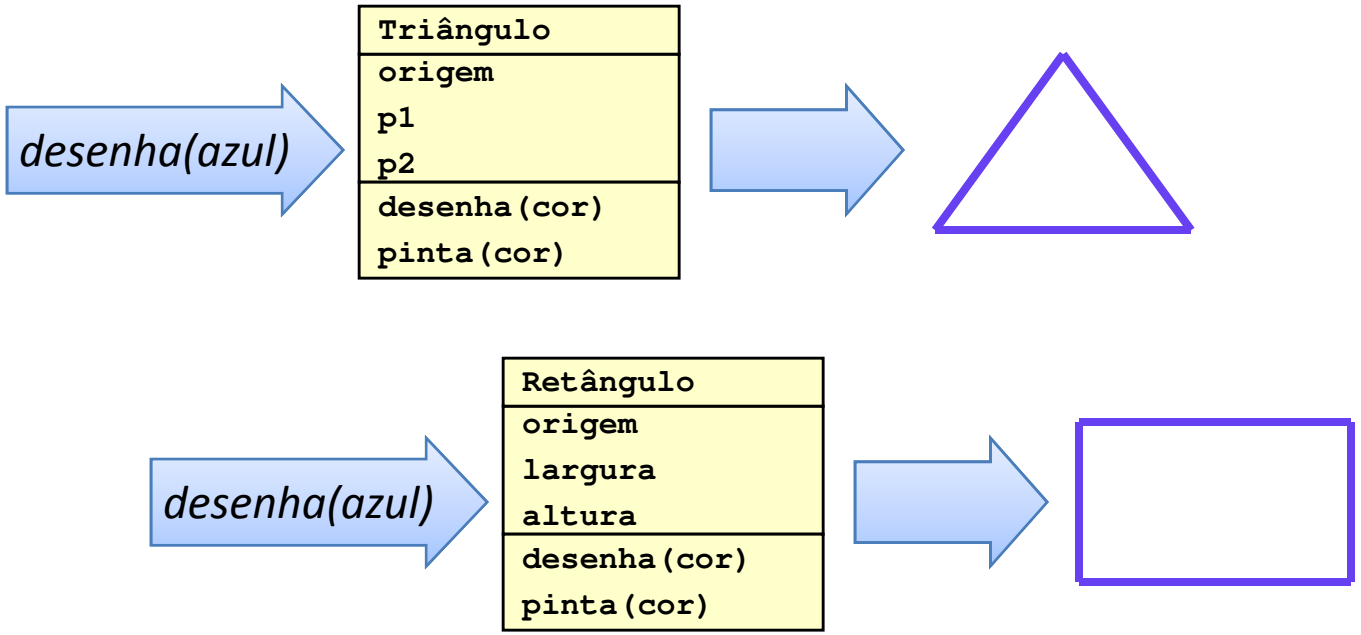


- Um objeto *encapsula* numa única entidade o seu estado e os métodos que definem o seu comportamento.
- Consequência
  - acoplamento mais fraco entre as diversas partes do sistema
  - menor probabilidade de interferências espúrias entre as partes do sistema

- A execução de um *método* é disparada pelo envio de uma *mensagem* ao objeto.
- A execução do método é a *interpretação* que o objeto dá à mensagem.



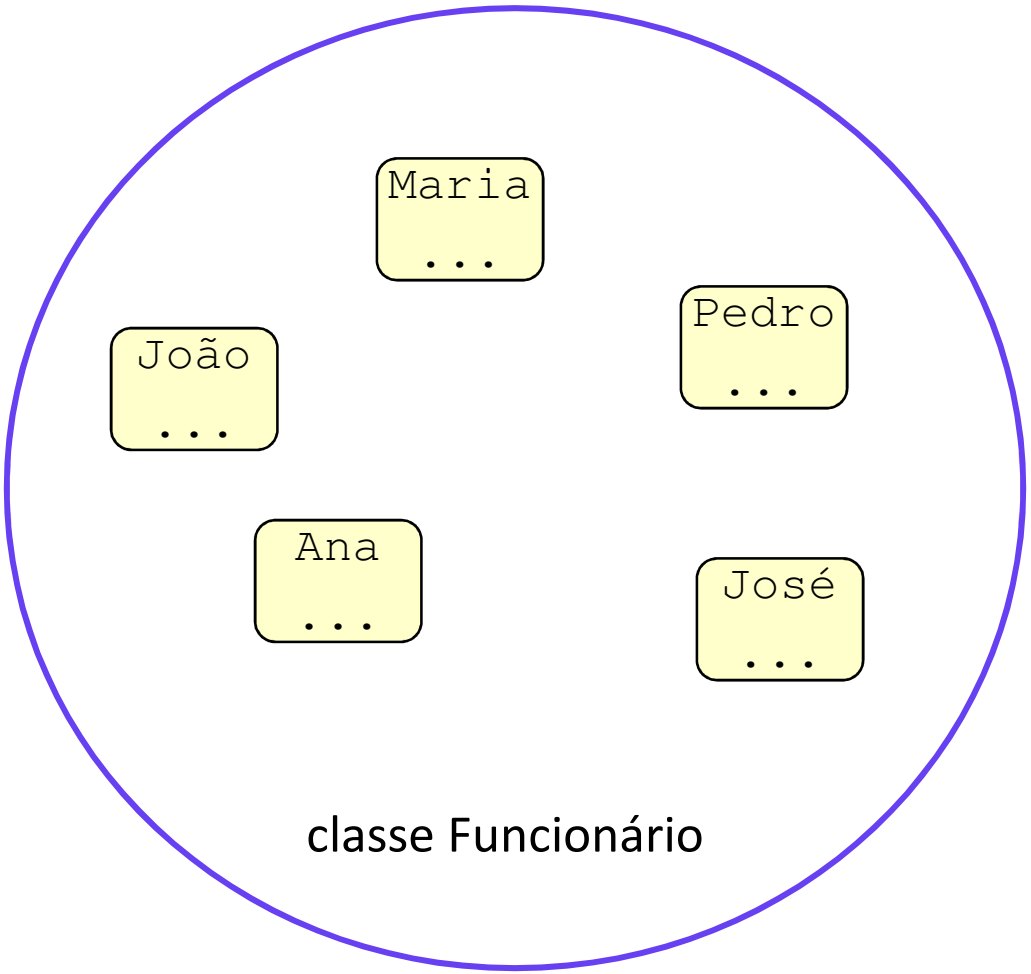
- Objetos diferentes podem dar interpretações diferentes a uma mesma mensagem



- Um *objeto* é um *conceito*, *abstração* ou algo que tenha um significado bem definido para o problema em questão
- uma *classe* descreve um *grupo de objetos* com o mesmo *conjunto de propriedades* (atributos), os mesmos *relacionamentos* com outros objetos e a mesma *semântica* (operações ou métodos)

Um objeto é uma *instância* (ou um exemplar) de uma classe.

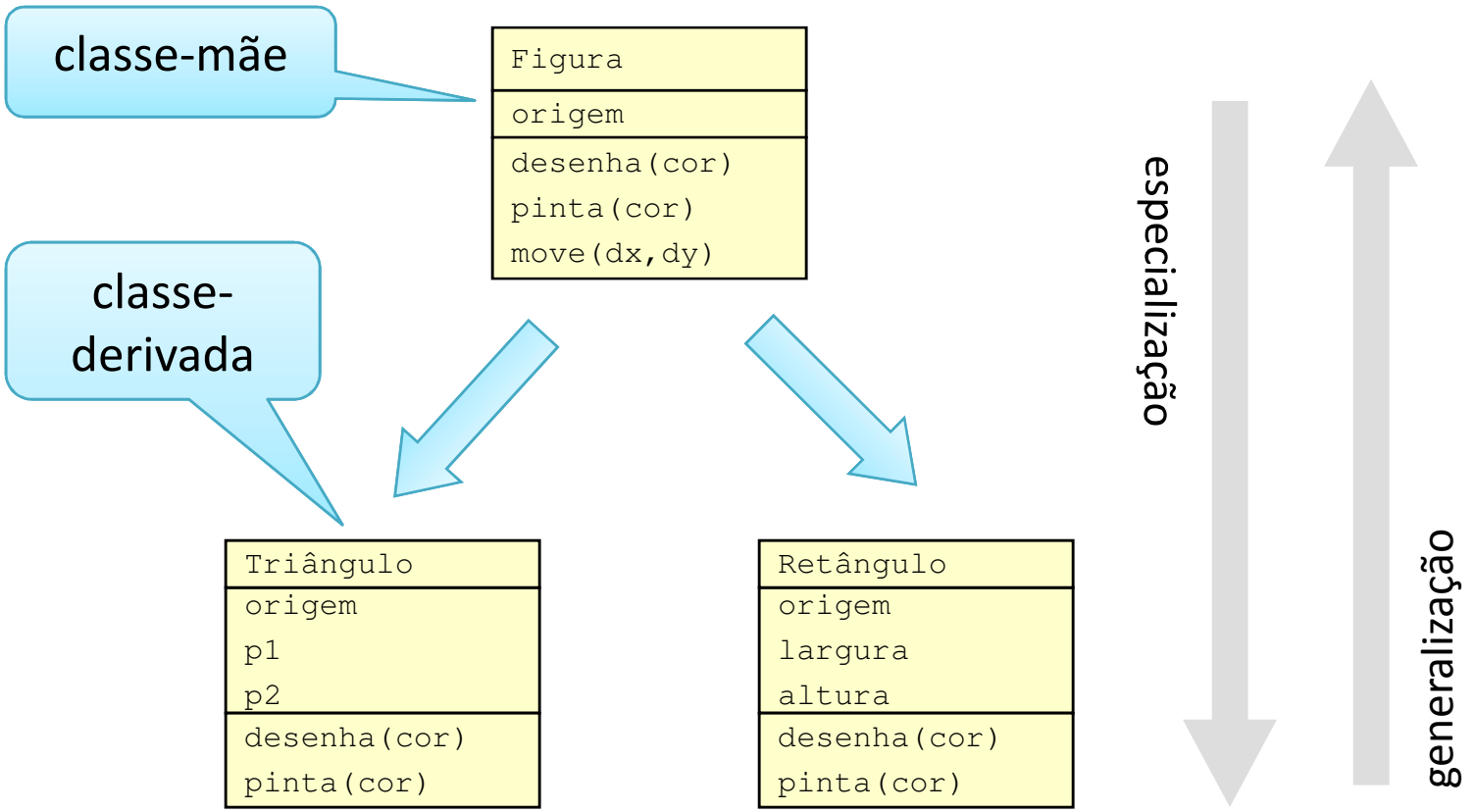
Objetos e Classes



- Todos os objetos da classe funcionário oferecem a mesma interface e mantêm o mesmo conjunto de atributos.
- Cada instância no entanto tem os seus próprios valores para os cada um dos atributos

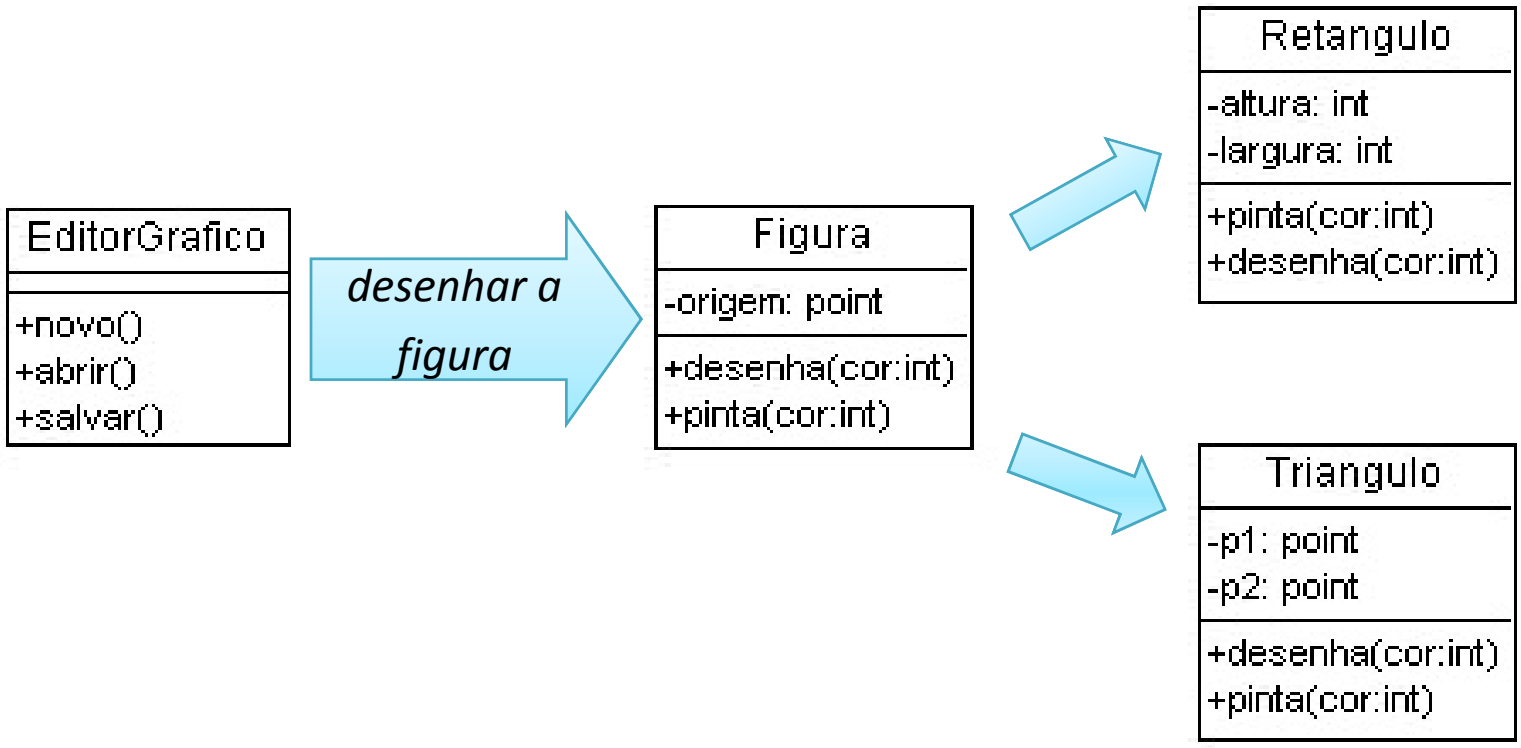
- Uma classe pode dar origem a outras classes através de *herança* (ou *especialização*).
- Os objetos da *classe derivada* herdam da classe origem (ou *classe-mãe*) todos os atributos e métodos.
- A classe derivada pode estender a classe-mãe agregando novos atributos e métodos. Ela pode também redefinir métodos da classe-mãe.

Herança



Klais

- Numa situação onde se espera um objeto de uma determinada classe C é sempre possível utilizar um objeto de uma classe derivada de C.



Os pilares da OO

- Encapsulamento, herança e polimorfismo constituem os pilares da orientação a objetos.
- Encapsulamento
  - melhora a modularidade do sistema
  - disciplina as interfaces
  - reduz interferências entre as partes
- Herança e polimorfismo
  - fatoração de elementos comuns
  - favorece o *reuso*

Objetos

- Um tipo de dado define
  - um conjunto de valores
  - um conjunto de operações sobre esses valores.
- Uma classe define
  - um conjunto de dados
  - um conjunto de métodos
- Um objeto é um 'elemento' de uma classe
- Cada objeto tem os seus dados associados

Um exemplo

- Suponha uma classe que define frações. Cada objeto dessa classe tem associado
  - um valor correspondente ao numerador
  - um valor correspondente ao denominador
  - um conjunto de operações (soma, subtração, etc.)

Um exemplo


```
public class Fracao {
  int num;    // numerador
  int den;    // denominador

  /* construtor */
  public Fracao(int n, int d){
    int m = mdc(n,d);
    num = n/m; den = d/m;
  }

  public String toString() { return "("+num+"/"+den+")"; }

  private static int mdc(int a, int b){
    while (a != b) if(a > b) a = a - b; else b = b - a;
    return a;
  }

  public Fracao soma(Fracao f){
    return new Fracao(f.num*den + num*f.den, f.den*den);
  }

  public Fracao mult(Fracao f){
    return new Fracao(f.num*num,f.den*den);
  }

  public float toFloat(){ return (float)num / (float)den; }

}
```

- Nesse exemplo
  - o método

```
public Fracao(int n, int d){
        int m = mdc(n,d);
        num = n/m; den = d/m;
}
```

é o construtor dessa classe: utilizado para a criação de objetos dessa classe.

- tem o mesmo nome que a classe
- inicia os atributos do objeto sendo criado (neste caso i e r).
- exemplo de uso: `Fracao f = new Fracao(12,32);`

- o modificador public indica que o método pode ser utilizado por outras classes que façam parte do projeto ou usem o pacote que contém a classe.

- Nesse exemplo
  - a declaração das 'variáveis'

```
int num;
int den;
```

define os dados ou *atributos* associados a cada objeto da classe.

  - os atributos definidos dessa forma só podem utilizados pelos métodos da classe onde são definidos.

- Nesse exemplo
  - o método

```
public string toString(){ return "("+num+"/"+dem+")";
```

é responsável pela conversão de um objeto Fracao para string. Exemplo:

- Supondo a declaração `Fracao f = new Fracao(2,3);`
- ao fazermos `c.toString()`, o valor retornado será o string `"(2/3)"`
- o operador + na expressão `"("+num+"/"+den+")"` é usado para realizar a concatenação de strings.
- o modificador `public` permite que o método `toString()` seja usado por outras classes.

Um exemplo


```
public class Teste1 {

  /**
   * @param args
   */
  public static void main(String[] args) {
    Fracao f1 = new Fracao(1,2);
    Fracao f2 = new Fracao(2,3);
    System.out.println("f1:"+f1.toString()+ " = " + f1.toFloat());
    System.out.println("f2:"+f2.toString()+ " = " + f2.toFloat());
    System.out.println("f1+f2:"+f1.soma(f2).toString());
    System.out.println("f1*f1:"+f1.mult(f2).toString());
  }

}
```

- Nesse exemplo
  - os objetos **f1** e **f2** são criados através da chamada ao construtor. Essas variáveis são apenas referências a objetos.
  - quando declaramos uma variável da forma

```
Fracao f1;
```

estamos declarando uma referência a um objeto que ainda não está definido.

ao fazermos a atribuição

```
f1 = new Fracao(1,2);
```

estamos indicando que **c1** passa a se referir ao objeto criado através do construtor. Se fizermos uma atribuição da forma

```
f2 = f1;
```

fazemos com que **f1** e **f2** sejam referências aos mesmos objetos.