MC302EF
Introdução a Programação Orientada a Objetos

Prof. Fernando Vanini
vanini@ic.unicamp.br

Motivação

- Desenvolvimento de software
  - Alto custo
  - Tempo de desenvolvimento e depuração
- Primeiras linguagens e ambientes de programação: preocupação com
  - Reduzir custos e tempo
  - Reuso
  - Generalização

Um exemplo em C

```
#include <stdio.h>
...
int main(...){
   ...
   FILE* arq = fopen("saida.txt","w");
   ...
   fprintf(arq,...);
   ...
   flose(arq);
   ...
}
```

Um exemplo em C

- Nesse exemplo
  - a biblioteca stdio cria o tipo abstrato de dados FILE
  - o arquivo 'stdio.h' contém apenas as definições necessárias para que uma aplicação consiga usar variáveis do tipo FILE e as operações associadas (como fopen(), fclose(), fprintf(), etc.)
  - os detalhes de implementação do tipo FILE e operações associadas ficam escondidos dos programas de aplicação que o utilizam.

Os conceitos envolvidos

Uma biblioteca como stdio se baseia em conceitos que

- norteiam a construção da maioria das bibliotecas padrão de C.
- são aplicáveis aos módulos de quaisquer aplicações.

Os conceitos

- abstração
- encapsulamento
- generalização

Abstração

- *abstração*: “eliminação do irrelevante e a amplificação do essencial” (Robert C. Martin).
- em programação, *abstração* é usada para separar os conceitos essenciais do problema a ser resolvido dos detalhes relacionados unicamente com programação.

Encapsulamento

- Consiste em manter dentro de cada módulo sistema os detalhes que só dizem respeito ao próprio.
- Cada módulo só 'exporta' as definições necessárias à utilização dos serviços que ele oferece.
- Em C, as definições exportadas por uma biblioteca, também chamada de 'interface', normalmente ficam num arquivo com extensão '.h' (de *header file*).

Generalização

- Consiste em projetar cada módulo ou biblioteca de forma que seja aplicável ao maior número de situações possível.
- Exemplo: uma biblioteca para ordenação que possa ser utilizada praticamente por qualquer tipo de dado para o qual se consiga definir uma função de comparação.

Aplicação

- Os conceitos de abstração, encapsulamento e generalização são aplicáveis às estruturas de dados de uso mais comuns:
  - listas
  - pilhas
  - filas
  - árvores binárias de busca
  - grafos
  - tabelas de hashing
  - etc.

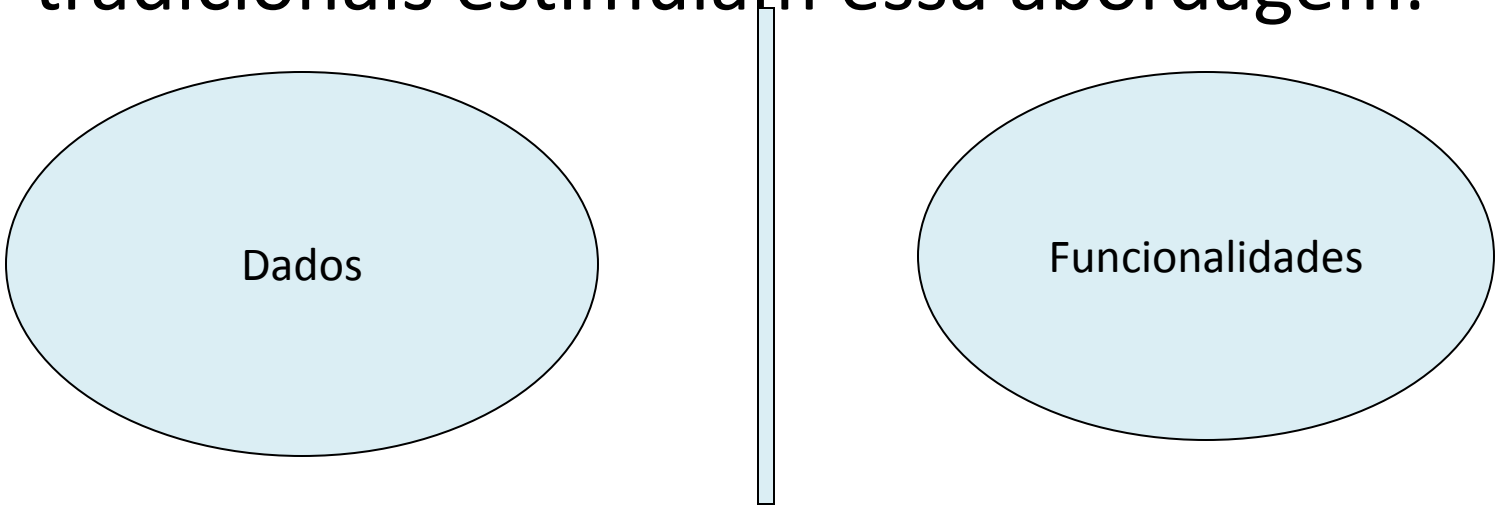
Aplicação

Numa linguagem como C, a aplicação desses conceitos tem um preço:

- O uso de apontadores para estruturas e funções viola as verificações de tipo oferecidas pelo compilador, comprometendo a confiabilidade do software.

Programação 'Tradicional'

- Dados e funcionalidades são tratados como elementos independentes.
- As linguagens e ambientes de programação tradicionais estimulam essa abordagem.



Programação ‘Tradicional’

- Um exemplo

Funcionário

nome
endereço
registro
cargo
salário
situação
data de admissão

Dados descrevem características estáticas do funcionário. Características dinâmicas (relativas ao negócio) são expressas pelos programas que implementam as funcionalidades

Diferentes aplicações que utilizem os mesmos dados devem implementar as ‘características dinâmicas’ da mesma maneira, para que não ocorram inconsistências.

Objetos

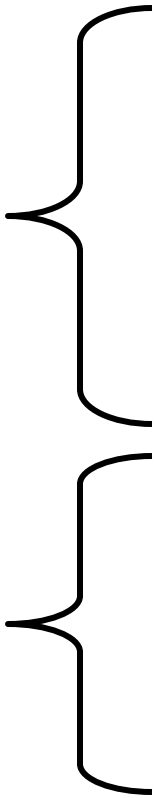
- Objetos reunem num único elemento as *características estáticas* e *comportamentais*.
- Características estáticas são representadas como *dados* associados ao *objeto*. Elas descrevem o *estado* atual do objeto.
- Características dinâmicas são descritas através de operações ou *métodos* executados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

Objetos

- Um exemplo

Estado: dados (ou atributos) que descrevem o 'estado atual' do objeto

Comportamento: métodos que descrevem os aspectos dinâmicos do objeto.



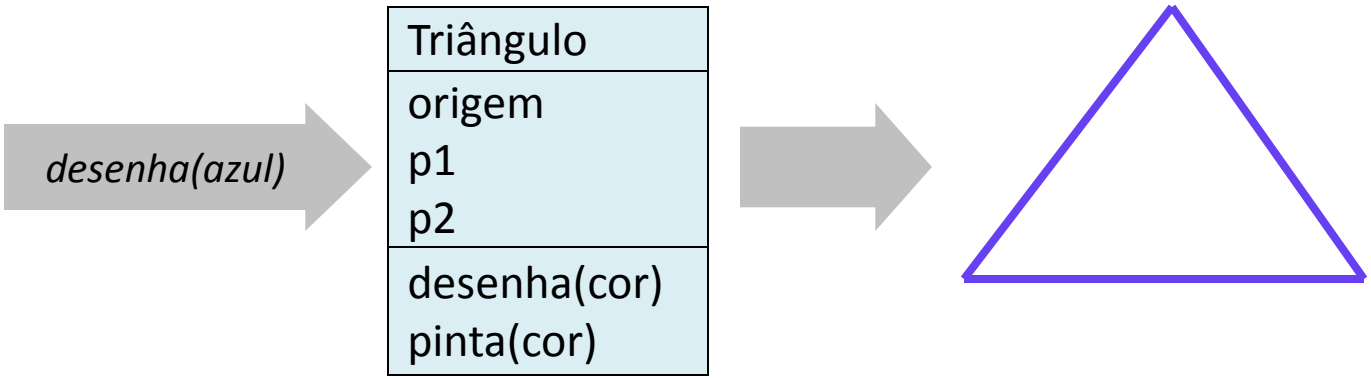
Funcionário
nome endereço registro cargo salario_bruto situação data de admissão
calcular_salário_líquido() atualizar_situação() alterar_cargo() calcular_FGTS() verificar_saldo_férias()

Encapsulamento

- 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

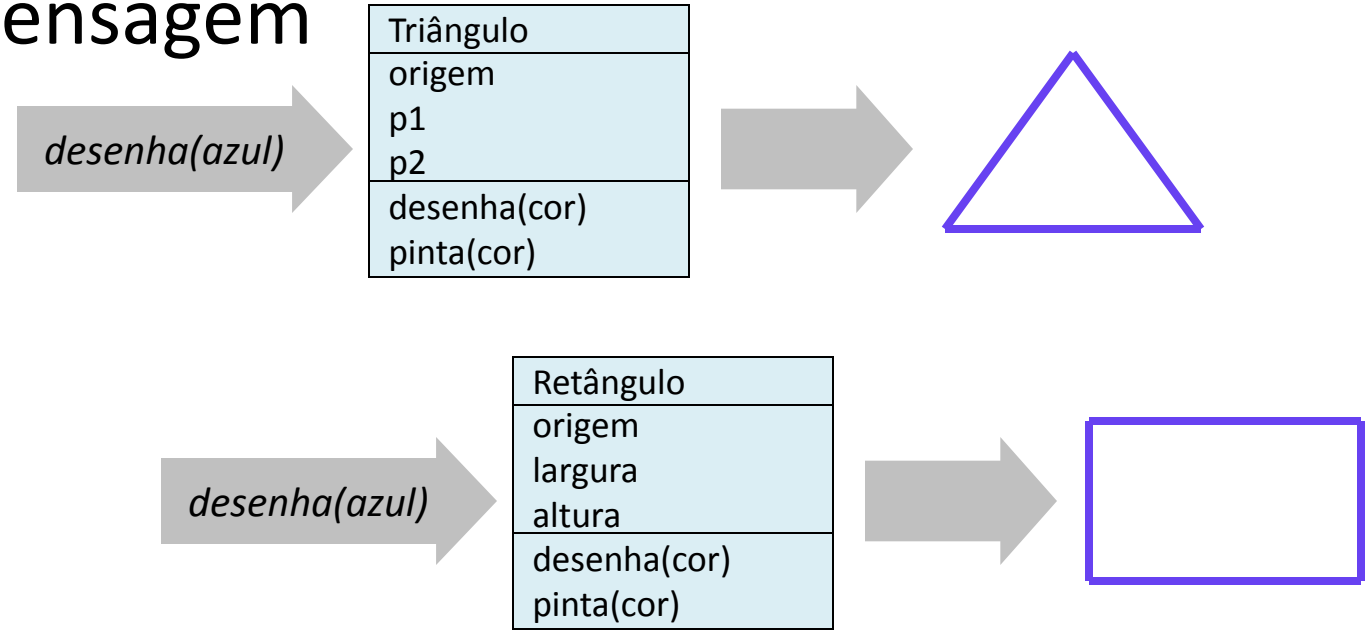
Métodos e mensagens

- 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.



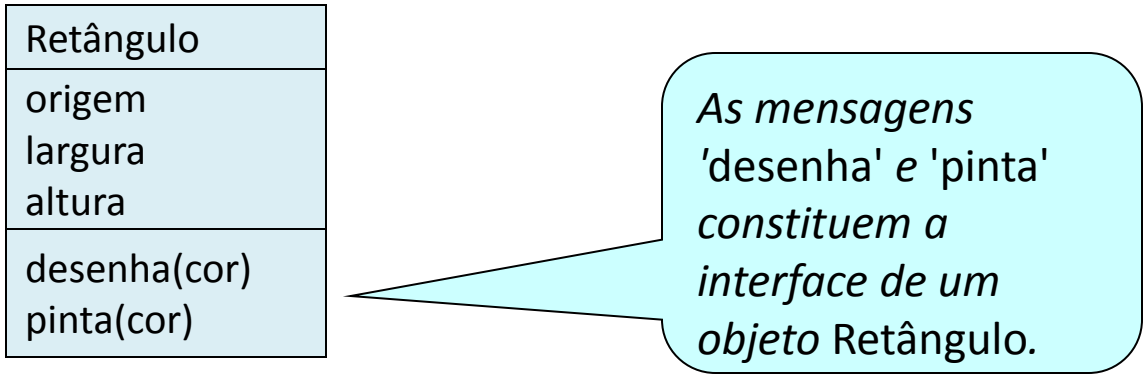
Métodos e mensagens

- Objetos diferentes podem dar interpretações diferentes a uma mesma mensagem



Interface

- A *interface* de um objeto é formada pelo conjunto de mensagens tratadas pelo objeto.

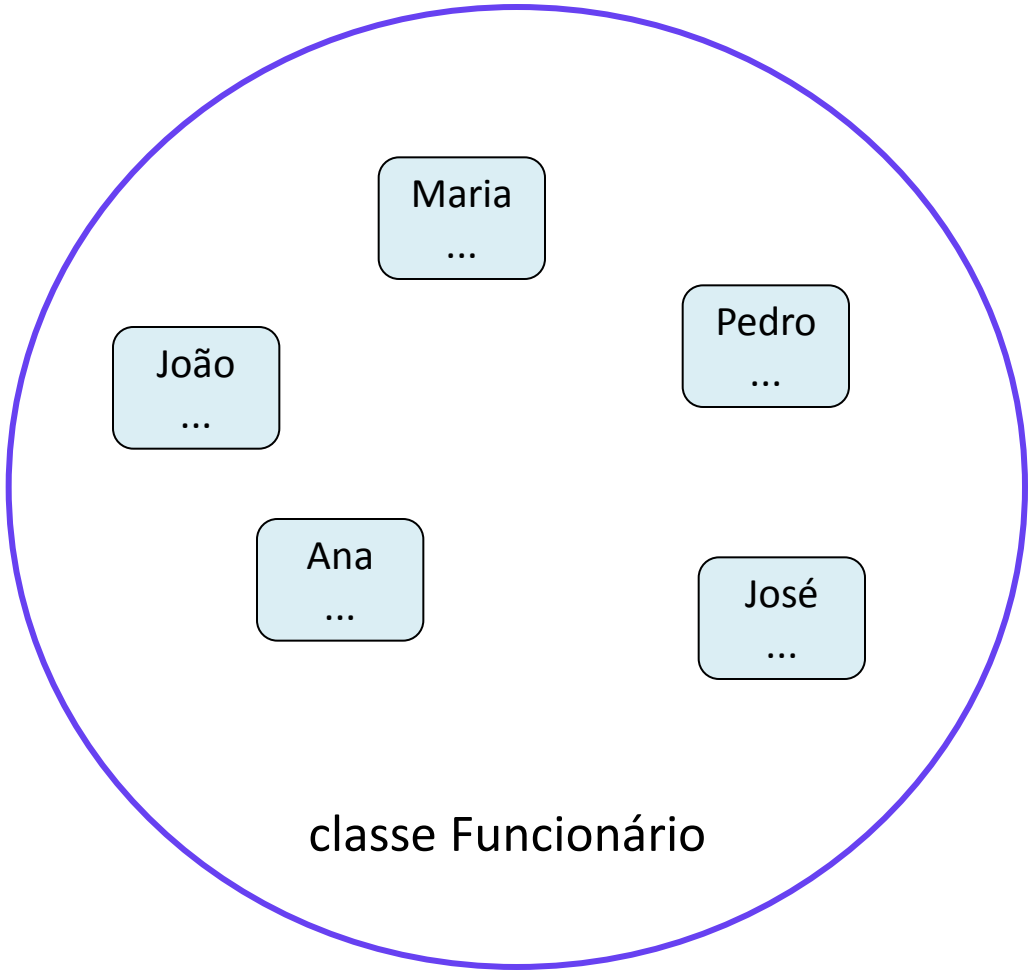


Objetos e Classes

- 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)

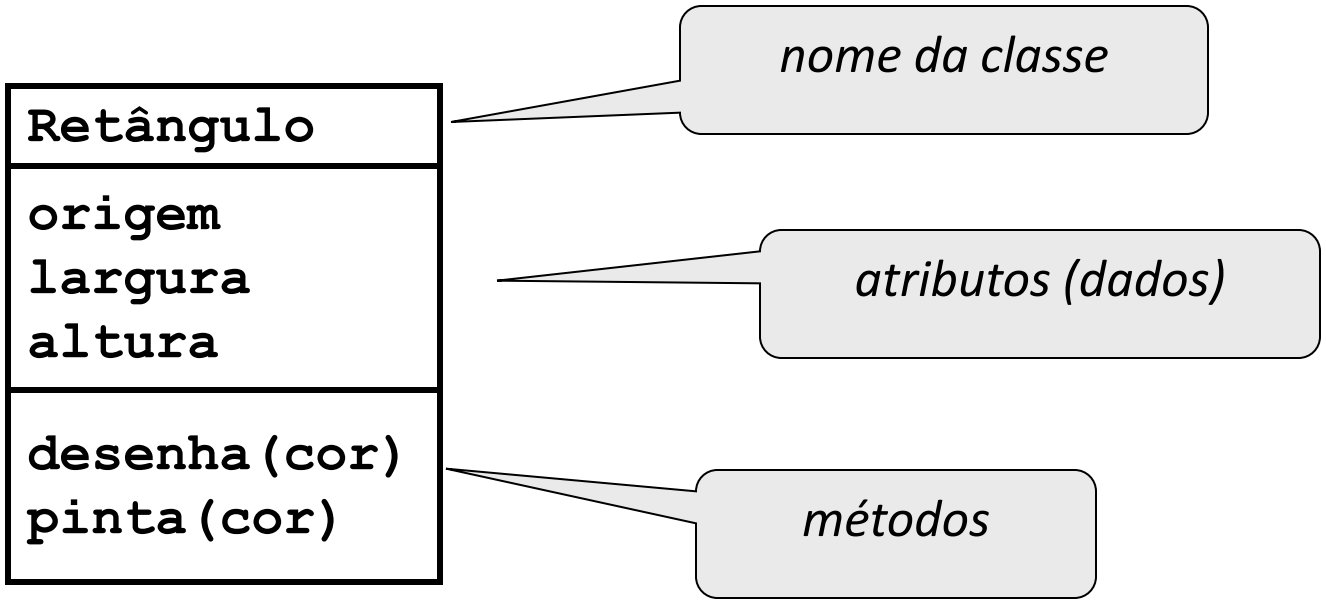
Um objeto é uma *instância* (ou um exemplar) de uma classe.

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

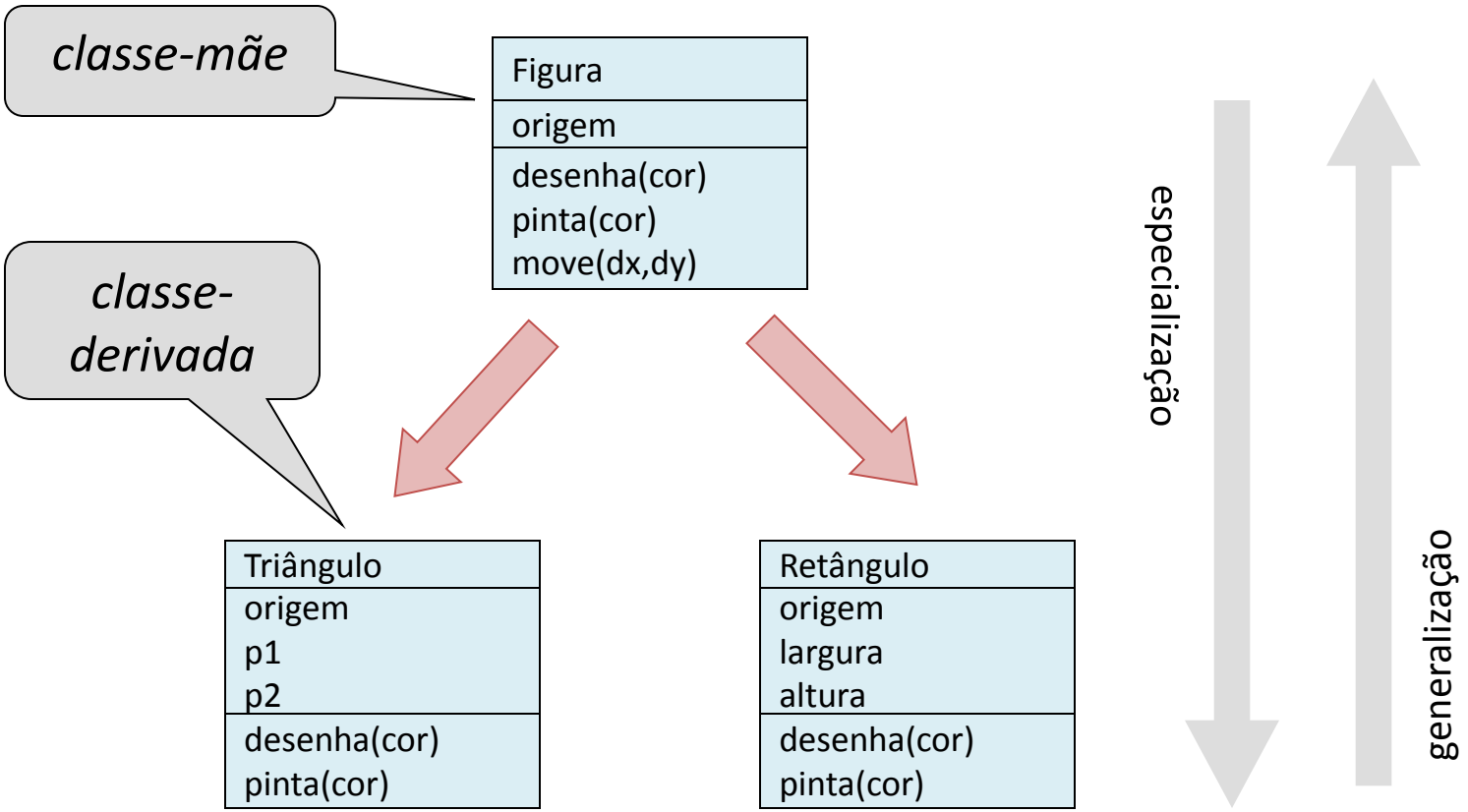
Representação Gráfica



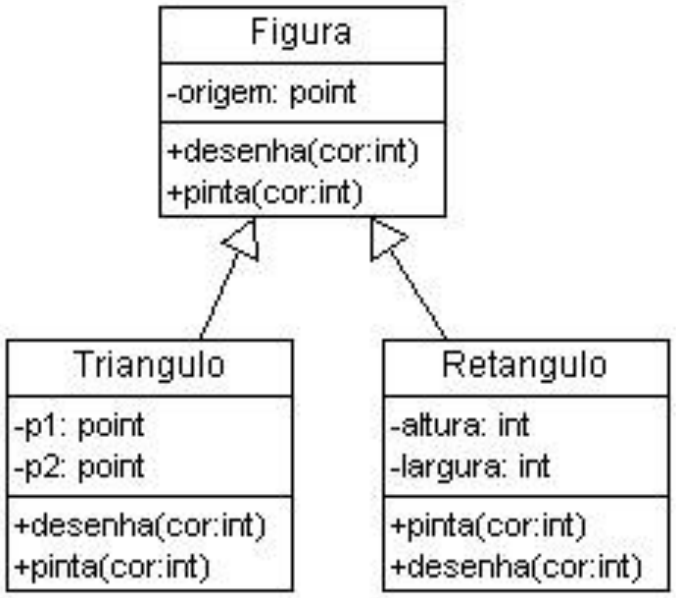
Herança

- 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.

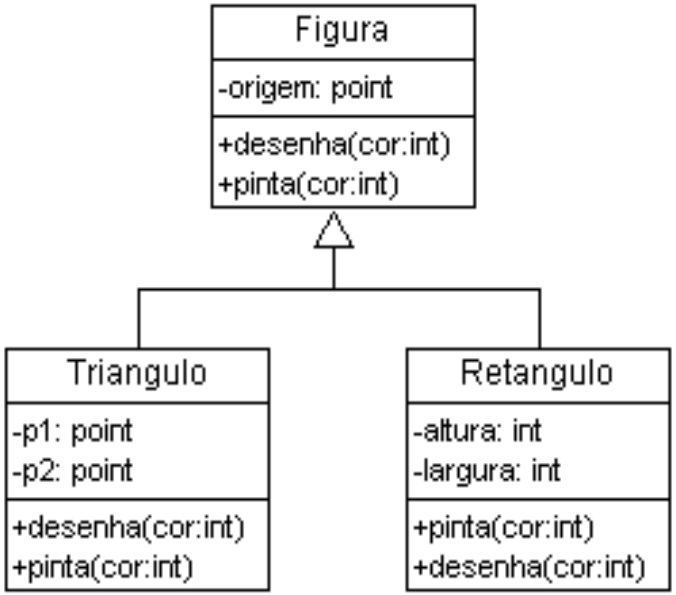
Herança


classe-mãe
Figura
origem
desenha(cor)
pinta(cor)
move(dx,dy)
classe-derivada
Triângulo
origem
p1
p2
desenha(cor)
pinta(cor)
Retângulo
origem
largura
altura
desenha(cor)
pinta(cor)
especialização
generalização

Representação Gráfica

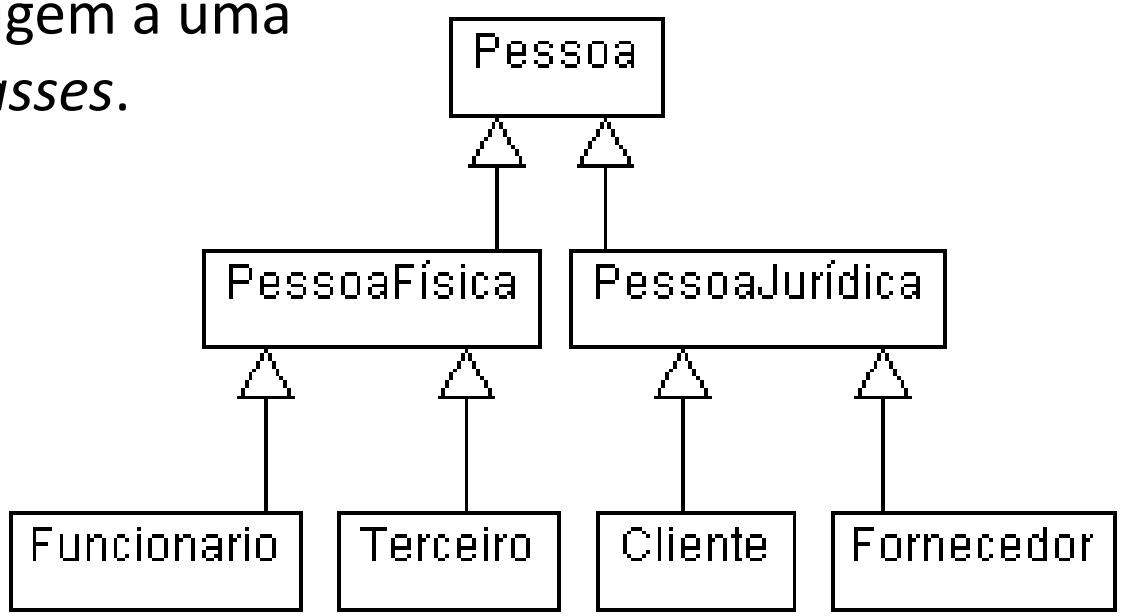


ou



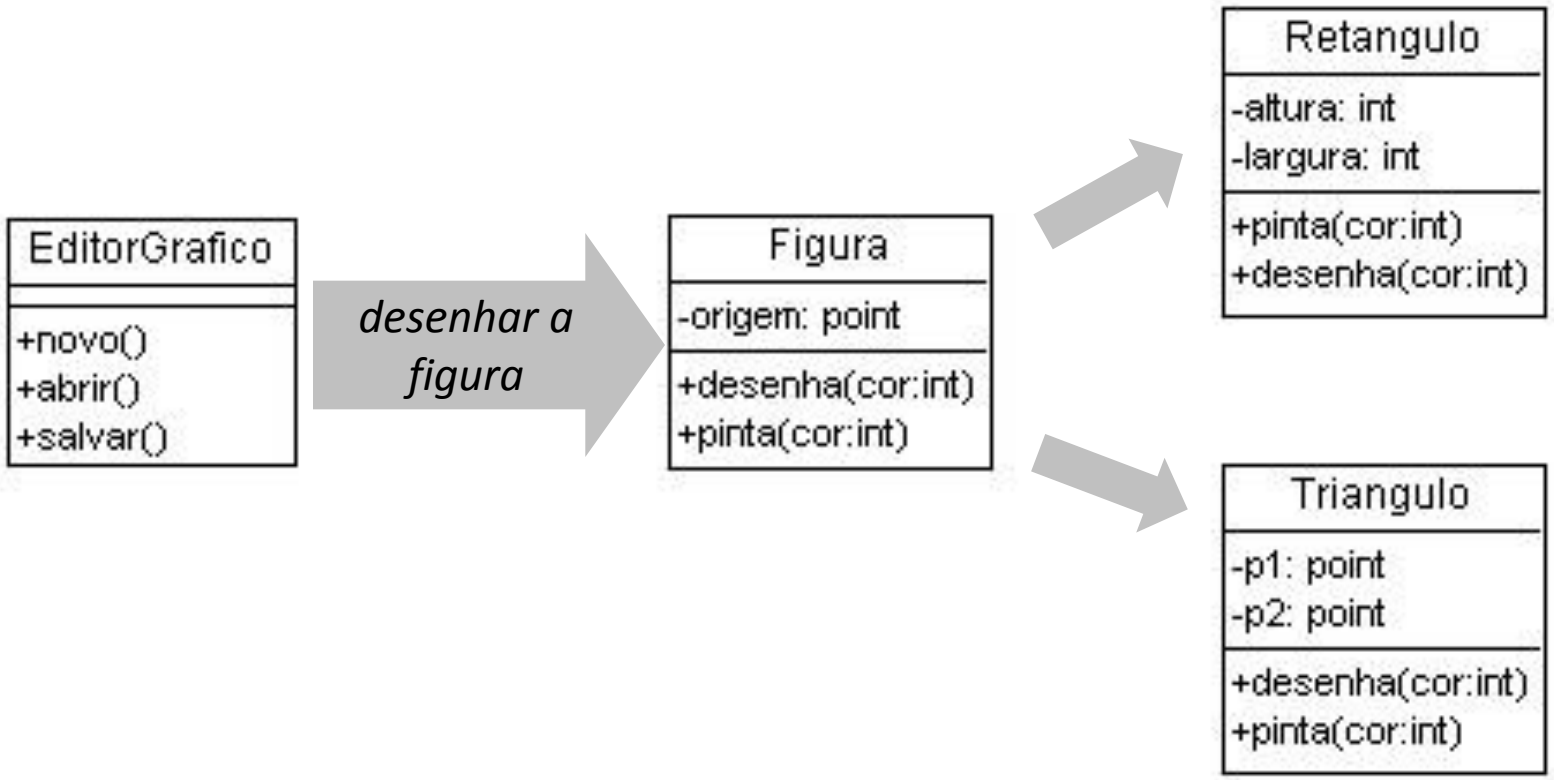
Hierarquia de Classes

- Herança também se aplica a classes derivadas de outras
- isso pode dar origem a uma *hierarquia de classes*.



Polimorfismo

- 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 e o 'mundo real'

- Objetos têm se mostrado convenientes para descrever elementos e situações do *mundo real*.
- Por essa razão, quando se usa objetos para modelar um sistema, é possível partir de conceitos pertencentes ao *domínio do problema*.
- Os detalhes no *domínio da solução* podem ser postergados até o momento em que se tenha uma idéia clara do que o sistema deve fazer.

Domínio do Problema

- Os conceitos no domínio do problema tendem a se manter estáveis ao longo da vida do sistema.
- Exemplo: num sistema de administração escolar
  - aluno, sala, turma, professor, disciplina, curso
- Um sistema cuja arquitetura é modelada com base nesses conceitos tende a suportar melhor a evolução (necessária em qualquer sistema).

Impacto no Ciclo de Vida do Software

- Pelo fato de encapsular estado e comportamento, objetos são adequados a representar conceitos ou elementos do mundo real.
- Como os requisitos evoluem ao longo do processo de desenvolvimento, a realimentação por parte do cliente final nesse período é importante e em geral influencia o sistema sendo desenvolvido.

Impacto no Ciclo de Vida do Software

- As características impostas à arquitetura, principalmente baixo acoplamento e reuso influenciam a fase de verificação e validação do sistema.
- Pelas mesmas razões, a evolução do sistema também é afetada.

Impacto na Arquitetura do Software

- baixo acoplamento: objetos, pelas sua característica de encapsulamento, permitem que se modele a arquitetura do sistema a partir de elementos com *baixo acoplamento* entre si. As partes do sistema que usam um dado objeto, não precisam saber como ele funciona, precisam saber apenas quais os serviços que ele oferece (expresso pelos métodos que ele exporta).
- reuso: herança e polimorfismo estimulam a reutilização do código que implementa as classes. O baixo acoplamento entre as parte por sua vez facilita o reuso.

Impacto na Arquitetura do Software

- foco *no domínio do problema* ao invés de *domínio da solução*.
- *componentes de software*: objetos têm sido usados para a criação de componentes de software, que além de oferecerem métodos para acesso aos seus serviços, oferecem métodos adicionais que permitem o seu registro, identificação e caracterização dos seus serviços.

Impacto nas Ferramentas de Desenvolvimento

- ambientes de desenvolvimento: novos ambientes disponibilizam ao projetista um conjunto de componentes que podem ser facilmente integrados às aplicações.
- extensibilidade: novos componentes podem, desenvolvidos por terceiros ou pela própria equipe, podem ser integrados ao ambiente de desenvolvimento, abrindo um leque de infinitas possibilidades.

Impacto na Equipe de Desenvolvimento

- a partir do domínio de bibliotecas e ferramentas de desenvolvimento, o programador passa a ser uma peça importante no processo de desenvolvimento.
- A definição da arquitetura, por ser influenciada pelo uso dos objetos, pelo reuso e pela farta disponibilidade de componentes, na maioria dos casos, necessita da participação do ‘programador-arquiteto’.

Impacto na Equipe de Desenvolvimento

- o programador necessita cada vez mais dominar todo o contexto formado pelos recursos de programação oferecidos pela linguagem e ambiente sendo utilizados, assim como bibliotecas de classes, componentes e módulos de software candidatos ao reuso.