

Reutilização de código em Programação Orientada a Objectos

Herança, Composição e Componentes

Paulo Moura, UBI, Março de 2004

Estrutura da palestra

- Uma apresentação breve
- Seguida de um problema de programação para debate entre todos

Para a palestra correr bem...

- Não fumar...
- Não soltar gases... (sim, sabemos que acabaram de almoçar)...
- E, pelas alminhas, não tirar os sapatos!!!

Reutilização de código

- Entre aplicações
- Dentro da mesma aplicação

Reutilização de código

- Entre aplicações
 - Bibliotecas de objectos
 - Bibliotecas de componentes
 - Design Patterns

Reutilização de código

- Dentro da mesma aplicação
 - Herança
 - Composição
 - Componentes

Reutilização de código

- Duas facetas:
 - Reutilização de interface/protocolos
 - Reutilização de implementação

Reutilização de interfaces/protocolos

- Implica que a linguagem suporte o conceito de interface ou protocolo
- Exemplos: Java, D

Herança

- Baseada em hierarquias de objectos
- Quatro facetas:
 - Interface versus implementação
 - Simples versus múltipla
 - Uma hierarquia ou múltiplas hierarquias
 - Pública, protegida e privada

Herança

- Algumas linguagens suportam hierarquias de interfaces/protocolos
- Vantagem: podemos definir um protocolo minimal que pode ser especializado para definirmos um protocolo extendido
- Exemplos: Java, D, Logtalk, ...

Herança simples

- Cada objecto tem apenas um antecessor
- Exemplos: Smalltalk, Java
- Geralmente associada a linguagens que suportam apenas uma única hierarquia de objectos
- Pode forçar a migração de métodos para o topo das hierarquias

Herança múltipla

- Cada objecto pode ter múltiplos antecessores
- Exemplos: C++, Eiffel
- Geralmente associada a linguagens que suportam várias hierarquias
- Maior flexibilidade, possibilidade de conflitos

Herança de implementação

- Chamar um método herdado
- Especializar um método herdado
- Definição de métodos "abstractos": métodos constituídos por chamadas a métodos definidos nos descendentes

Composição

- Baseado na utilização de variáveis que “contêm” outros objectos que queremos reutilizar
- O protocolo dos objectos reutilizados é utilizado indirectamente

Componentes

- Noção de componente: unidade de encapsulamento funcionalmente coesa
- Um objecto importa um ou mais componentes
- Uma forma de composição
- O protocolo dos componentes é adicionado ao protocolo do objecto

Exemplo:
Pontos geométricos em
2D

Ponto

posição()
mover()
escrever()

Ponto simples em 2D

Ponto_h

Ponto c/ história

história()

Ponto_ml

Ponto c/ movimento limitado

limites()

Ponto_hml

Ponto c/ história e movimento limitado

Problemas

- ponto_h e ponto_ml:
 - especializam os métodos `mover()` e `escrever()`
- ponto_hml:
 - como implementar os métodos `mover()` e `escrever()`?

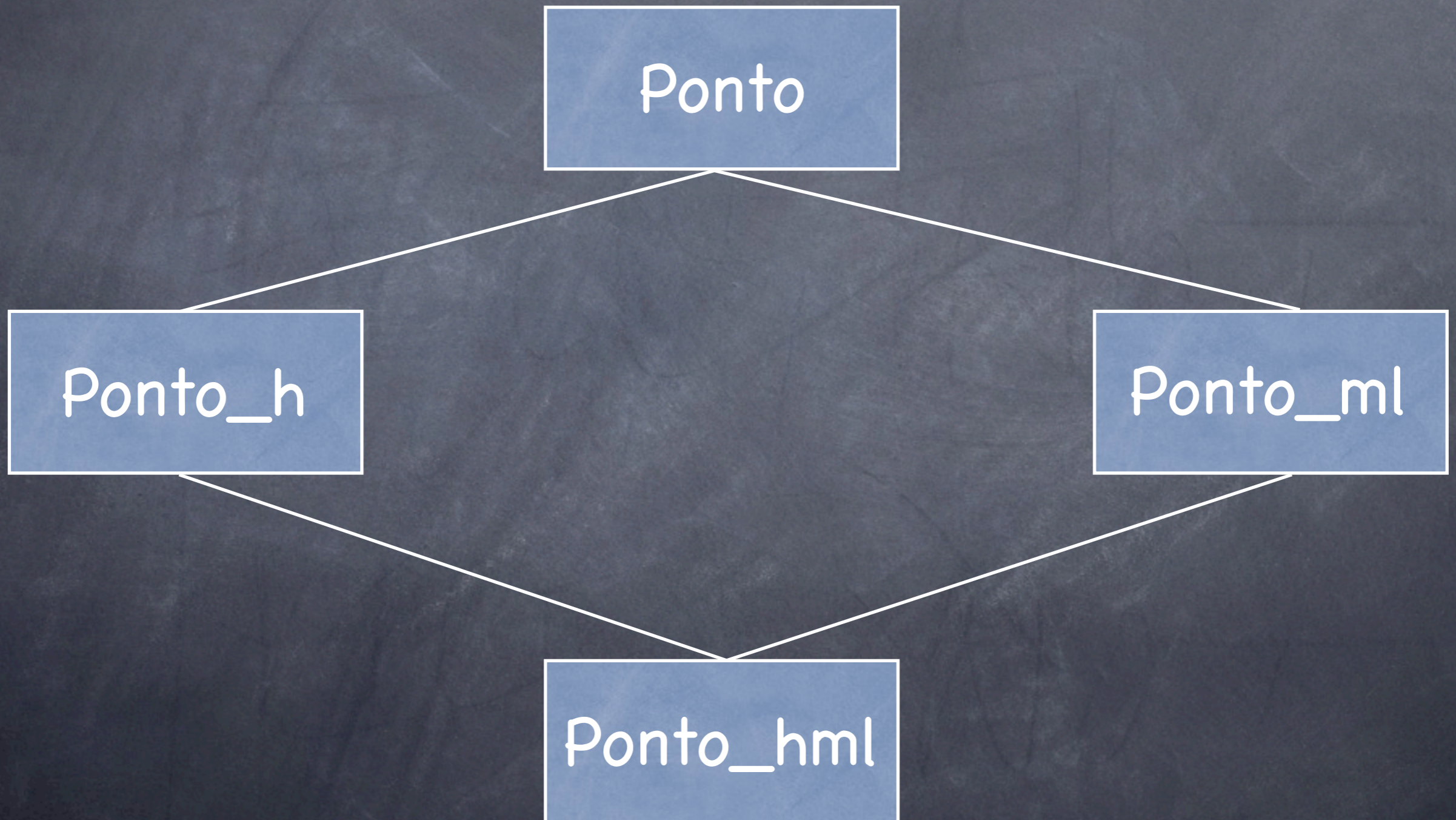
Problemas

- Queremos reutilizar os métodos específicos de `ponto_h` e `ponto_ml`, mas...
- Não queremos escrever as coordenadas do ponto duas vezes!
- Não queremos mover o ponto duas vezes!

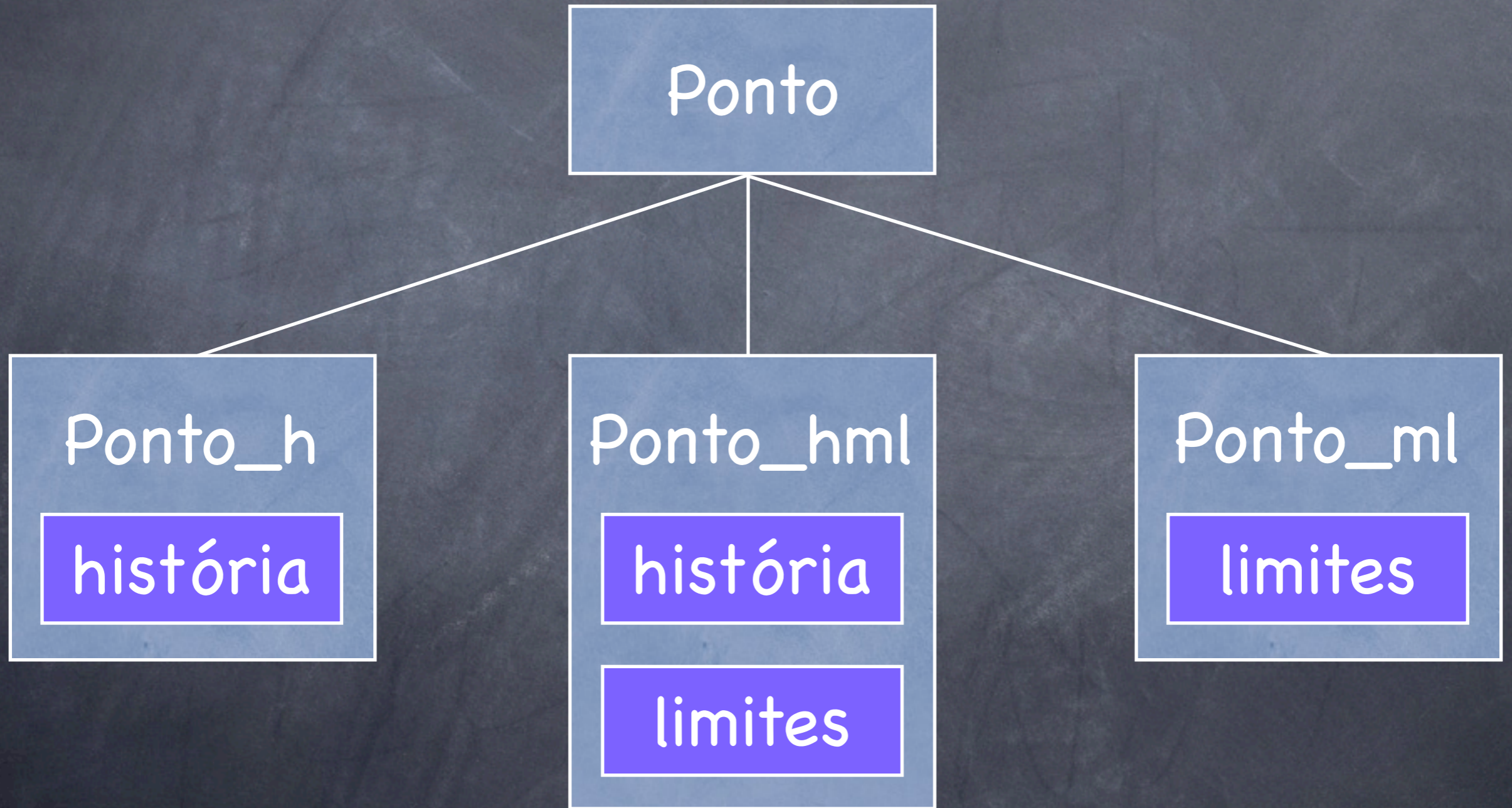
Possíveis soluções

- Herança múltipla (C++, Eiffel, ...)
- Herança simples + Composição (Smalltalk, Java, ...)
- Componentes (Aspect/J, Logtalk, ...)

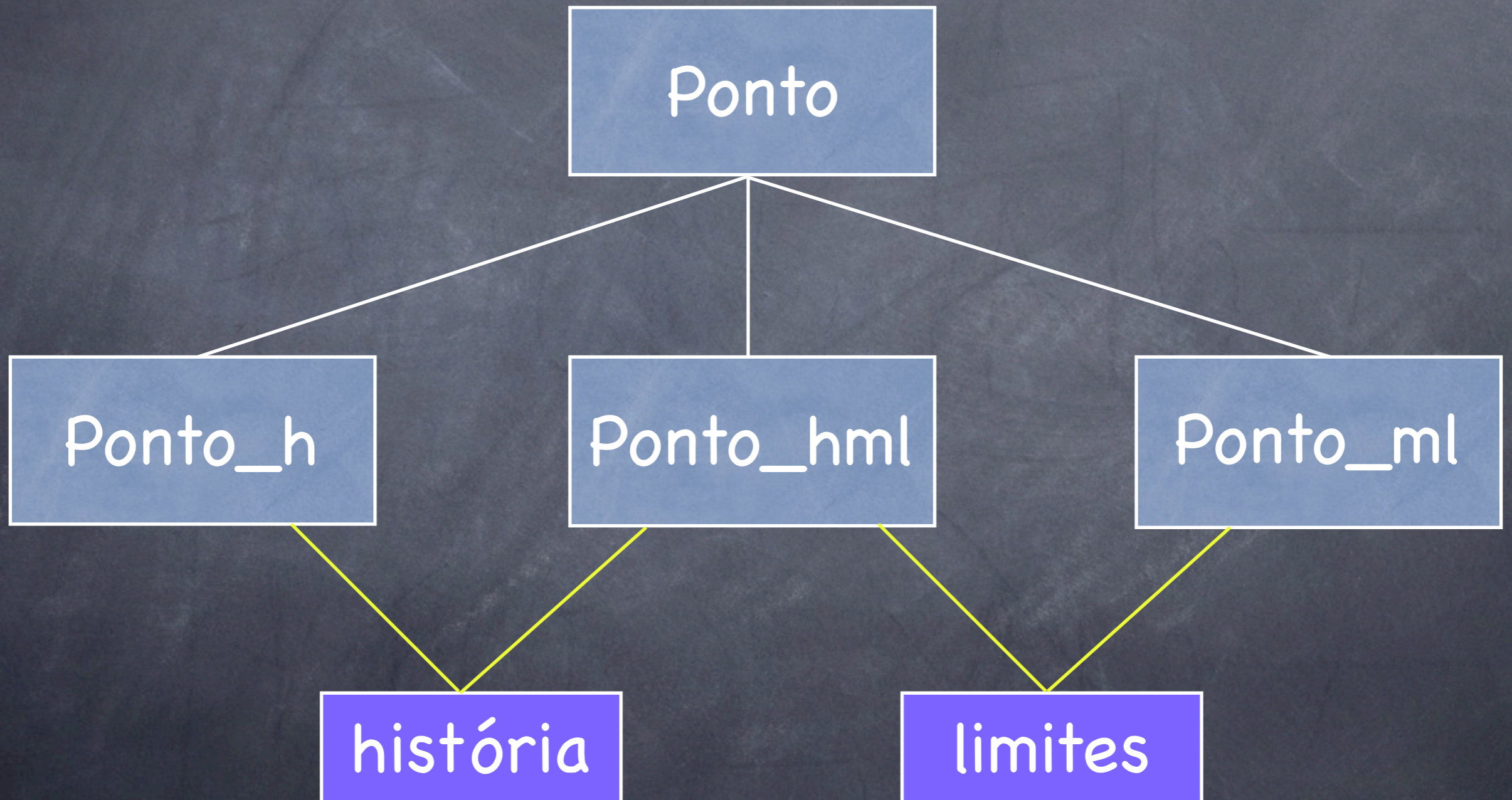
Herança múltipla



Herança simples + composição



Componentes



Questões?

Aplausos?

Donativos?