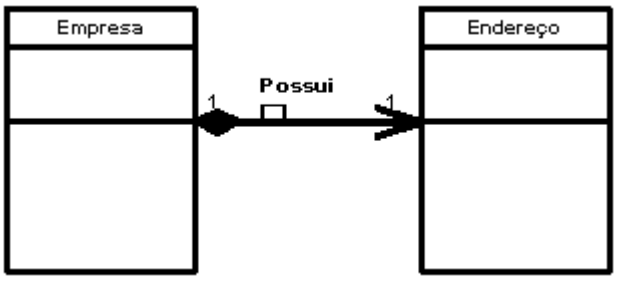
Padrão Básico de Projeto: Herança versus Composição

Composição e Herança

- Composição e herança são dois mecanismos para reutilizar funcionalidade
- Alguns anos atrás (e na cabeça de alguns programadores ainda!), a herança era considerada a ferramenta básica de extensão e reuso de funcionalidade
- A composição estende uma classe pela delegação de trabalho para outro objeto
  a herança estende atributos e métodos de uma classe
- Hoje, considera-se que a composição é muito superior à herança na maioria dos casos
  - A herança deve ser utilizada em alguns (relativamente poucos) contextos
- Vamos portanto desinflar um pouco a bola da herança ...

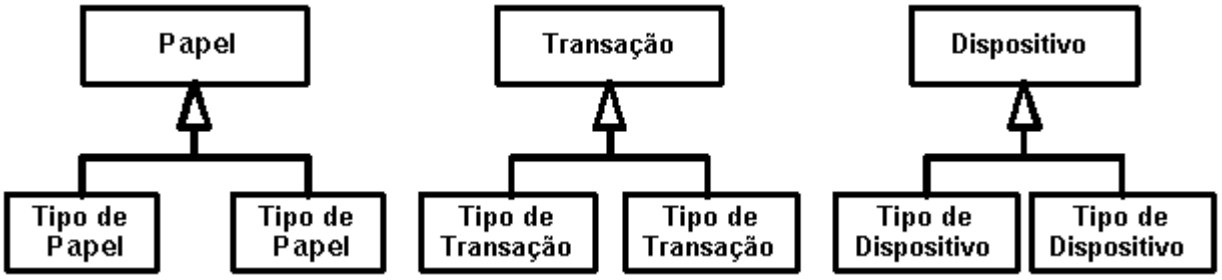
Um exemplo de composição

- Use composição para estender as responsabilidades pela delegação de trabalho a outros objetos
- Um exemplo no domínio de endereços
  - Uma empresa tem um endereço (digamos só um)
  - Uma empresa "tem" um endereço
  - Podemos deixar o objeto empresa responsável pelo objeto endereço e temos agregação composta (composição)

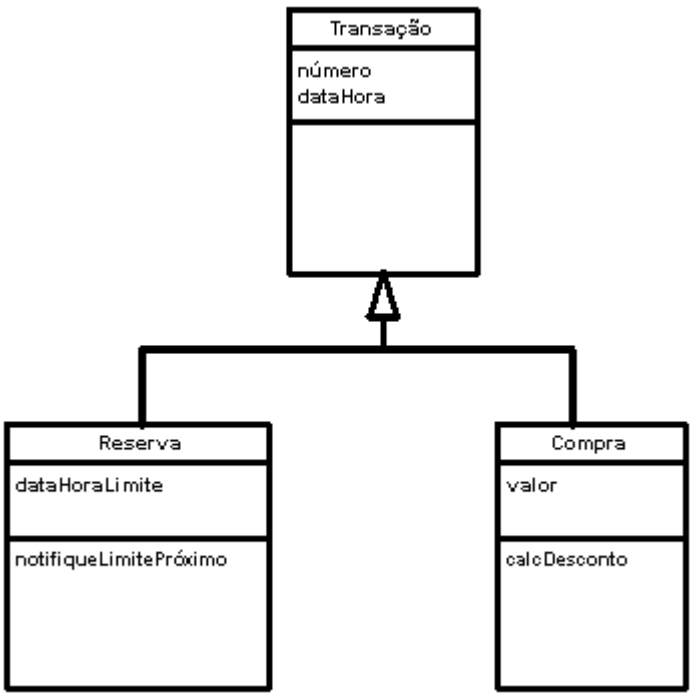


Um exemplo de herança

- Atributos, conexões a objetos e métodos comuns vão na superclasse (classe de generalização)
- Adicionamos mais dessas coisas nas subclasses (classes de especialização)
- Três situações comuns para a herança (figura abaixo)
  - Uma transação é um momento notável ou intervalo de tempo



- Exemplo no domínio de reserva e compra de passagens de avião

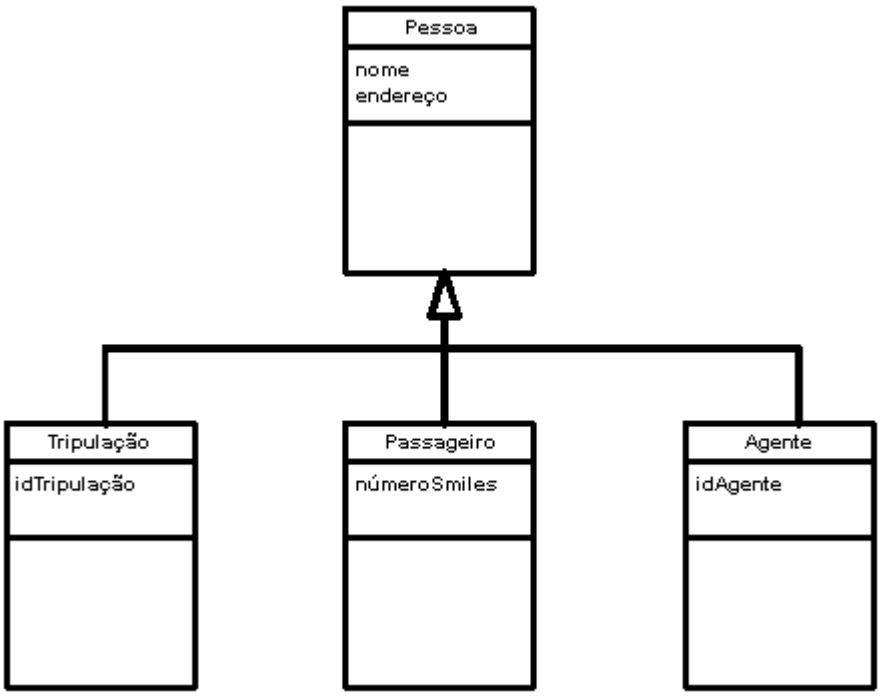


Benefícios da herança

- Captura o que é comum e o isola daquilo que é diferente
- A herança é vista diretamente no código

Problemas da herança

- O encapsulamento entre classes e subclasses é fraco (o acoplamento é forte)
  - Mudar uma superclasse pode afetar todas as subclasses
    - *The weak base-class problem*
  - Isso viola um dos princípios básicos de projeto O-O (manter fraco acoplamento)
- Às vezes um objeto precisa ser de uma classe diferente em momentos diferentes
  - Com herança, a estrutura está parafusada no código e não pode sofrer alterações facilmente em tempo de execução
  - A herança é um relacionamento estático que não muda com tempo
  - Cenário: pessoas envolvidas na aviação (figura abaixo)



  - Problema: uma pessoa pode mudar de papel a assumir combinações de papeis
  - Fazer papeis múltiplos requer 7 combinações (subclasses)

Solucionando o problema com composição: uma pessoa e vários papeis possíveis

- Estamos estendendo a funcionalidade de Pessoa de várias formas, mas sem usar herança
- Observe que também podemos inverter a composição (uma pessoa tem um ou mais papeis)
  - Pense na implicação para a interface de "pessoa"
- Aqui, estamos usando delegação: dois objetos estão envolvidos em atender um pedido (digamos setNome)
  - O objeto tripulação (digamos) delega setNome para o objeto pessoa que ele tem por composição
  - Técnica também chamada de forwarding
  - É semelhante a uma subclasse delegar uma operação para a superclasse (herdando a operação)
    - Delegação sempre pode ser usada para substituir a herança
  - Se usássemos herança, o objeto tripulação poderia referenciar a pessoa com this
  - Com o uso de delegação, tripulação pode passar this para Pessoa e o objeto Pessoa pode referenciar o objeto original se quiser

- Em vez de tripulação *ser* uma pessoa, ele *tem* uma pessoa
- A grande vantagem da delegação é que o comportamento pode ser escolhido em tempo de execução e vez de estar amarrado em tempo de compilação
- A grande desvantagem é que um software muito dinâmico e parametrizado é mais difícil de entender do que software mais estático

O resultado de usar composição

- Em vez de codificar um comportamento estaticamente, definimos pequenos comportamentos padrão e usamos composição para definir comportamentos mais complexos
- De forma geral, a composição é melhor do que herança normalmente, pois:
  - Permite mudar a associação entre classes em tempo de execução;
  - Permite que um objeto assuma mais de um comportamento (ex. papel);
  - Herança acopla as classes demais e engessa o programa

5 regras para o uso de herança (Coad)

- O objeto "é um tipo especial de" e não "um papel assumido por"
- O objeto nunca tem que mudar para outra classe
- A subclasse estende a superclasse mas não faz override ou anulação de variáveis e/ou métodos
- Não é uma subclasse de uma classe "utilitária"
  - Não é uma boa idéia fazer isso porque herdar de, digamos, HashMap deixa a classe vulnerável a mudanças futuras à classe HashMap
  - O objeto original não "é" uma HashMap (mas pode usá-la)
  - Não é uma boa idéia porque enfraquece a encapsulação
    - Clientes poderão supor que a classe é uma

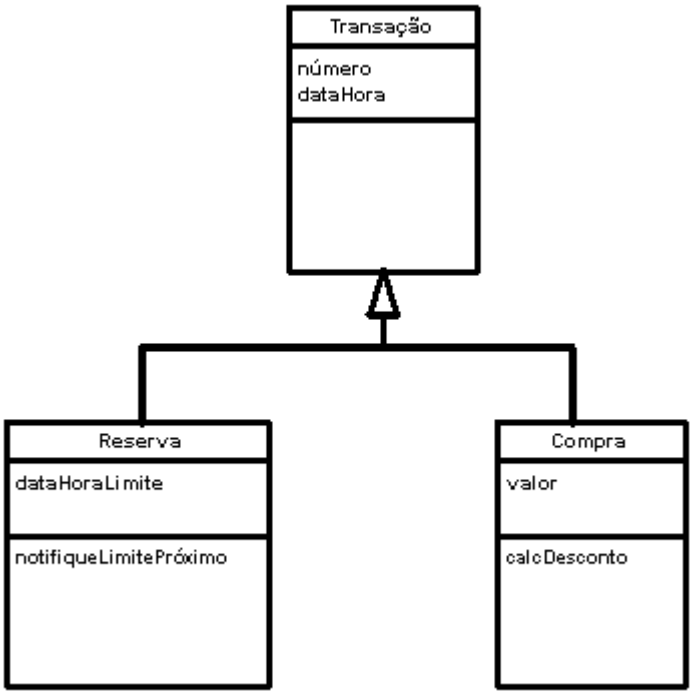
subclasse da classe utilitária e não funcionarão se a classe eventualmente mudar sua superclasse
    - Exemplo: x usa y que é subclasse de vector
        - x usa y sabendo que é um Vector
        - Amanhã, y acaba sendo mudada para ser subclasse de HashMap
        - x se lasca!
- Para classes do domínio do problema, a subclasse expressa tipos especiais de papeis, transações ou dispositivos

Exemplo da aplicação das regras

- Considere Agente, Tripulação e Passageiro como subclasses de Pessoa
    - Regra 1 (tipo especial): não passa. Um Passageiro não é um tipo especial de Pessoa: é um papel assumido por uma Pessoa
    - Regra 2 (mutação): não passa. Um Agente pode se transformar em Passageiro com tempo
    - Regra 3 (só estende): ok.
    - Regra 4: ok.
    - Regra 5: não passa. Passageiro está sendo modelado como tipo especial de Pessoa e não como tipo especial de papel

Outro exemplo: transações

- Reserva e Compra podem herdar de Transação?



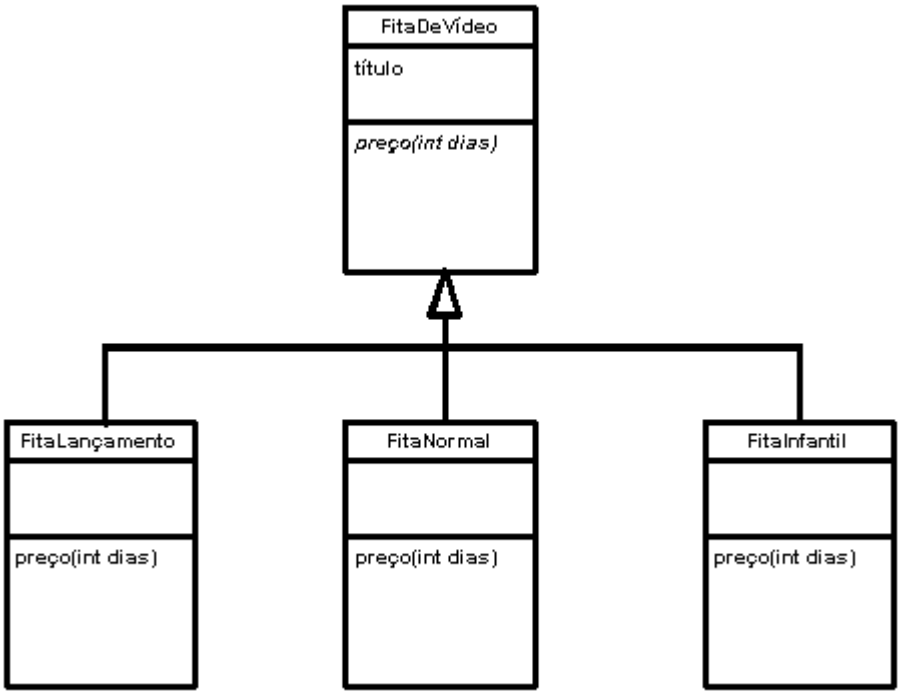
- Regra 1 (tipo especial): ok. Uma Reserva é um tipo especial de Transação e não um papel assumido por uma Transação
- Regra 2 (mutação): ok. Uma reserva sempre será uma Reserva, e nunca se transforma em Compra (se houver uma compra da passagem, será outra transação). Idem para Compra: sempre será uma Compra
- Regra 3 (só estende): ok. Ambas as subclasses estendem Transação com novas variáveis e métodos e não fazem override ou anulam coisas de Transação
- Regra 4 (não estende classe utilitária): ok.
- Regra 5 (tipo especial de papel/transação/dispositivo): ok. São tipos especiais de Transação

Para terminar ...

- Veremos muitos exemplos das vantagens da composição sobre a herança em Design Patterns, adiante

Exercício para casa

- Neste exemplo, que tal modelar as fitas de vídeo como mostrado na figura abaixo?


FitaDeVídeo
título
preço(int dias)
FitaLançamento
preço(int dias)
FitaNormal
preço(int dias)
FitaInfantil
preço(int dias)

programa