UnB Departamento de Ciência da Computação

Modelagem de Classes de Projeto

A perfeição (no projeto) é alcançada, não quando não há nada mais para adicionar, mas quando não há nada mais para retirar
Eric Raymond, The Cathedral and the Bazaar

Edison Ishikawa, D. Sc.

Introdução

- Objetivo
  - Descrever algumas das transformações por que passam as classes e objetos da análise para o modelo de classes de projeto

Sumário

- Introdução
- Desenvolvimento
  - Especificações de classes
  - Especificação de atributos
  - Especificação de operações
  - Especificações de associações
  - Herança
- Referências

Especificação de Classes de Fronteira

- Servem como ponto
  - de captação de informações a partir do ambiente, ou
  - de apresentação de informações que o sistema (modelo) processou
- **Não** se deve atribuir a esta classe responsabilidades relativas à lógica do negócio
  - Facilita a migração em caso de mudança de ambiente
  - Permite o suporte a diversas formas de interação
    - Interface gráfica
    - Interface texto, etc..
  - Melhora a coesão a classe de fronteira

Especificação de Classes de Fronteira

- Na análise, considera-se que há uma única classe de fronteira para cada ator que participa de um caso de uso
- No projeto, cada classe de fronteira pode gerar várias outras classes, considerando diferentes tipos de classes de fronteira
  - Classes de interface com o usuário
  - Classes de interface com equipamentos
  - Classes de interface com outros sistemas

Especificação de Classes de Fronteira

- Principais formas de interação com o ambiente
  - Clientes WEB Clássicos
    - Página HTML, normalmente dinâmicas
      - Note que neste caso, embora na análise esta interface seja representada como classe, no projeto ela é realizada como página HTML
  - Clientes móveis
    - Classes de fronteira que implementem algum protocolo específico como o WML (Wireless Markup Language)
  - Clientes stand-alone
    - Classes que implementem uma interface gráfica
  - Serviços WEB
    - Web services – forma de permitir que uma aplicação ofereça serviços

Especificação de Classes de Entidade

- Representam
  - Os conceitos do domínio que o sistema vai processar
  - As informações e regras do negócio que manipulam estas informações
- Também chamadas de Classes de Negócio
- Exemplos:
  - Em um sistema escolar temos as seguintes classes de entidade
    - Aluno, professor, disciplina

Especificação de Classes de Entidade

- Em geral, normalmente:
  - são as primeiras a serem identificadas na fase de análise
  - permanecem na passagem da análise para o projeto
  - são as que necessitam de persistência
    - Mapeamento para algum Banco de Dados
- Aspecto importante
  - Representação dos relacionamentos entre estas classes
    - Associação, agregação e composição
  - Identificação de cada objeto de forma única
    - chave

Especificação de Classes de Controle

- Identificadas na modelagem de classes da aplicação, i.e., não são objetos do domínio
- Responsabilidade
  - Coordenar a interação entre outros objetos
- Exemplo: Classes que
  - Controlam o preenchimento da interface gráfica
  - Fazem a autenticação do usuário
  - Controlam o acesso a funcionalidades do sistema

Especificação de Classes de Controle

- Classe controlador de caso de uso
  - Coordena a realização do caso de uso
  - Serve como canal de comunicação entre objetos de fronteira e objetos de entidades
  - Comunica-se com outros controladores, sfc
  - Mapeia ações dos atores para atualização ou mensagens a serem enviadas a objetos entidades
  - Manipula exceções provenientes de classes de entidades

Especificação de Classes de Controle

- Objeto front controller
  - Usado em aplicações Web
  - Responsável por receber todas as requisições de um cliente (navegador WEB ou app móvel)
  - Ponto central de entrada para funcionalidades do sistema

Especificação de outras classes

- Quando o sistema é distribuído
  - Necessidade de comunicação entre os objetos
    - Segurança, autenticação, autorização, etc
    - Classes procuradoras
      - Usam tecnologias como CORBA, RMI, SOA
- Quando o armazenamento persistente é necessário
  - Classes DAO – (Data Access Object)
  - Uso de SGBD OO ou framework de persistência

Especificação de outras Classes

- Durante o desenvolvimento o projetista deve usar o máximo de sua experiência e criatividade para definir uma solução a mais adequada possível, mas não precisa "reinventar a roda"
- Pode reutilizar código, evitando desenvolvimento/codificação desnecessário
  - Padrões de projeto
  - Bibliotecas de classe
  - Frameworks

Especificação de outras Classes

- Padrão de projeto
  - Descrição da essência da solução para um problema que ocorre de forma recorrente no desenvolvimento de software
  - Seu uso permite que desenvolvedores sejam mais produtivos, evitando que eles “reinventem a roda”
- Framework
  - Possui uma grande quantidades de classes já implementadas, prontas ara reuso
- Bibliotecas de classe
  - Coleção de classes com objetivo específico

Especificação de Atributos

- Notação UML

  [/]*visibilidade nome: tipo = valor_inicial*

- *visibilidade*
  - \+ pública – visível externamente
  - \# protegida – visível apenas nas subclasses
  - \- privativa – invisível externamente
    - é a *default*, se nada for explicitado
  - ~ pacote – visível a qualquer classe do pacote

Especificação de Atributos

- Notação UML

  [/]*visibilidade nome: tipo = valor_inicial*

- *nome*
  - Nome do atributo
  - Única parte obrigatória da sintaxe

Especificação de Atributos

- Notação UML

  [/]*visibilidade nome: tipo = valor_inicial*

- *tipo*
  - Tipo do atributo
  - Dependente da linguagem de programação
    - Primtivo
    - TAD – Tipo Abstrato de Dado
      - Exemplos
        - Data – dia mês ano
        - CPF e CNPJ para armazenamento e validação
        - Endereço

Especificação de Atributos

- Notação UML

  [/]*visibilidade nome: tipo = valor_inicial*

- *valor_inicial*
  - Ao instanciar um objeto, o valor inicial declarado é automaticamente atribuído ao atributo
  - Seu valor depende da linguagem de programação

Especificação de Atributos

- Notação UML

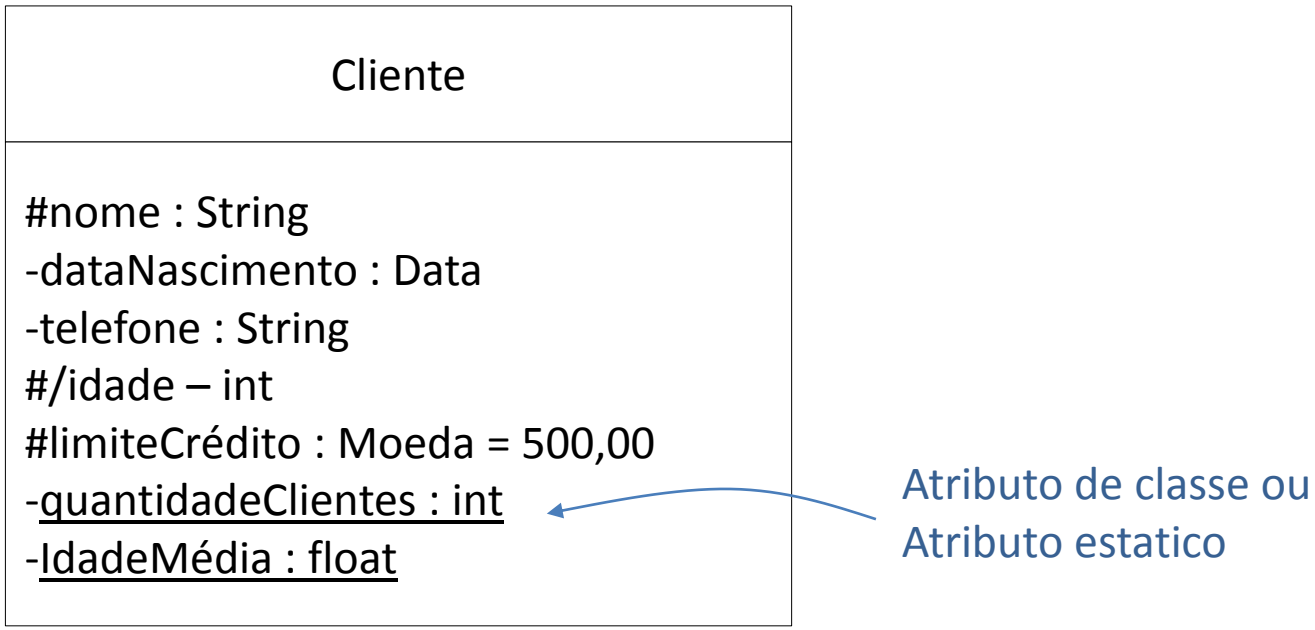
  [/]*visibilidade nome: tipo = valor_inicial*

- */*
  - Atributo derivado
  - Seu valor pode ser obtido a partir do valor de outros atributos
  - Exemplo
    - Idade pode ser obtido da data de nascimento

Especificação de Atributos

- Notação UML

  [/]*visibilidade nome: tipo = valor_inicial*



Especificação de Operações

- Notação UML

  visibilidade nome (parâmetros) : tipo-retorno {propriedades}

Cliente
+obterNome( ) : String +definirNome(in umNome : String) +obterDataNascimento( ) : Data +definirDataNascimento(in umaData : Data) +obterTelefone( ) : String +definirTelefone(in umTelefone : String) +obterLimiteCrédito( ) : Moeda +definirLimiteCrédito(in umLimiteCrédito : Moeda) +obterIdade( ) : Int +obterQuantidadeClientes( ) : Int +obterIdadeMédia( ) : float

Especificação de Operações

- Notação UML

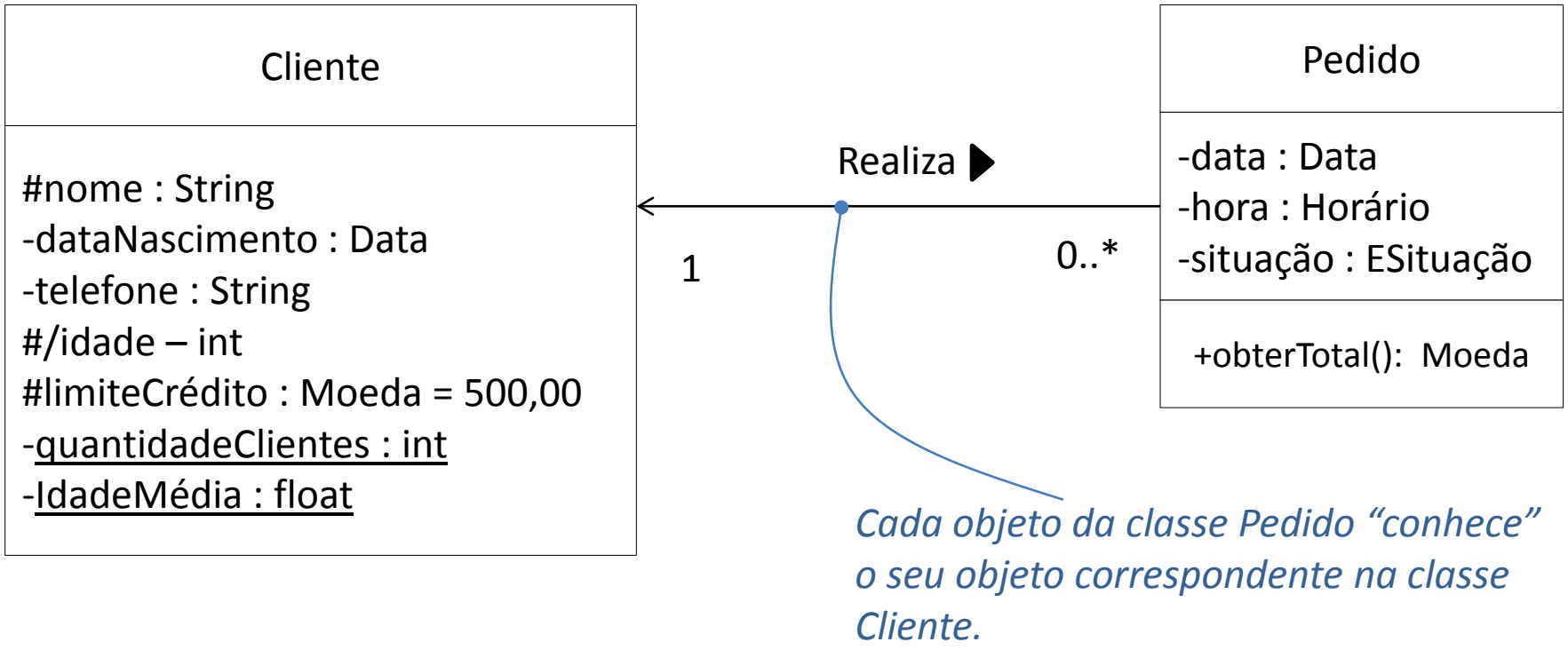
  visibilidade nome (parâmetros) : tipo-retorno {propriedades}

- Possíveis direções na definição de um parâmetro

Direção	Significado UML
in	Parâmetro de entrada: não pode ser modificado pela operação. Serve somente como informação para objeto receptor.
out	Parâmetro de saída: pode ser modificado ela operação para fornecer alguma informação ao objeto remetente.
inout	Parâmetro de entada que pode ser modificado

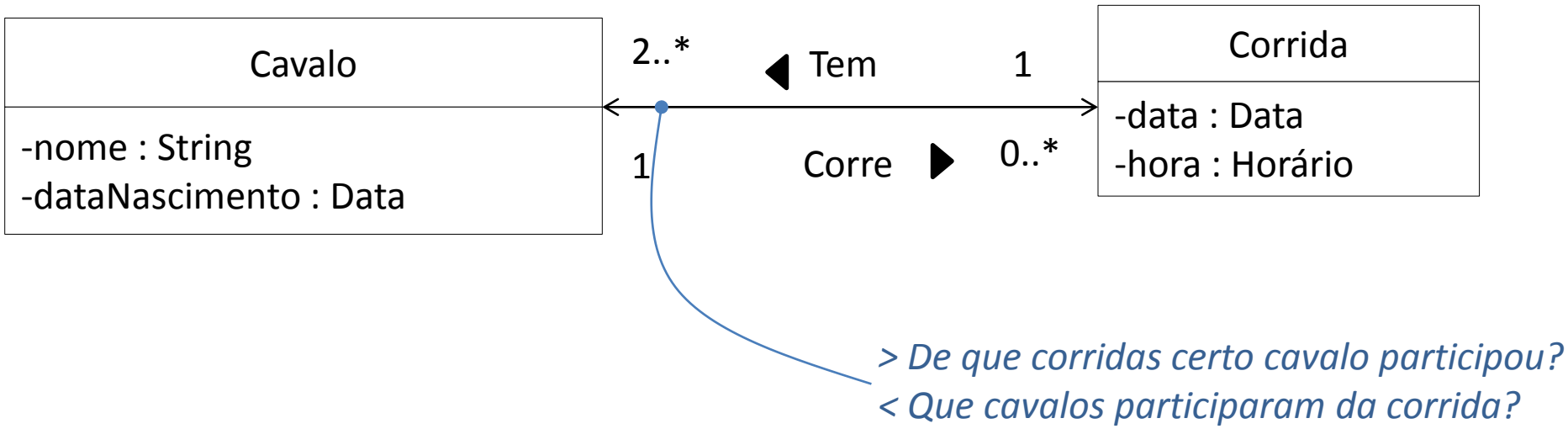
Especificação de Associações

- Navegabilidade de associações
  - Associação unidirecional



Especificação de Associações

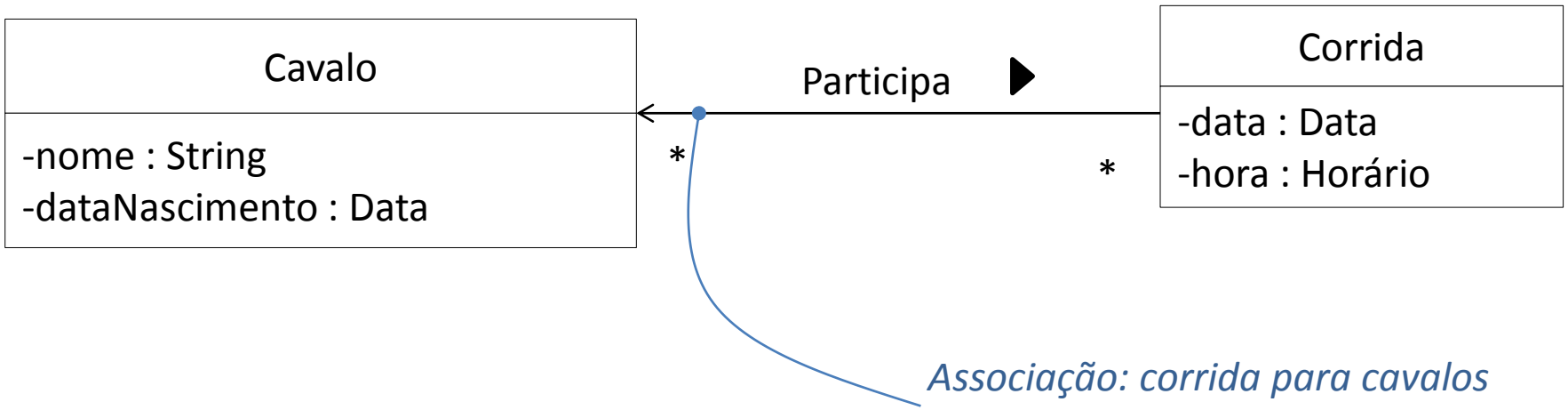
- Navegabilidade de associações
  - Associação bidirecional



Deve-se evitar associações bidirecionais, por ser mais difícil de implementar e manter (acoplamento é maior na associação bidirecional)

Especificação de Associações

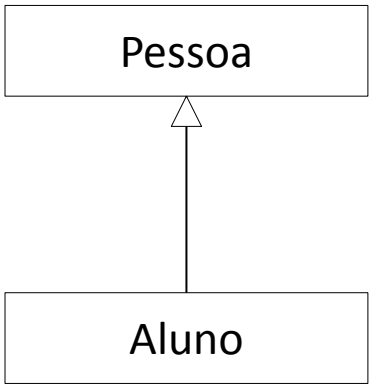
- Navegabilidade de associações
  - Substituindo uma associação bidirecional por unidirecional



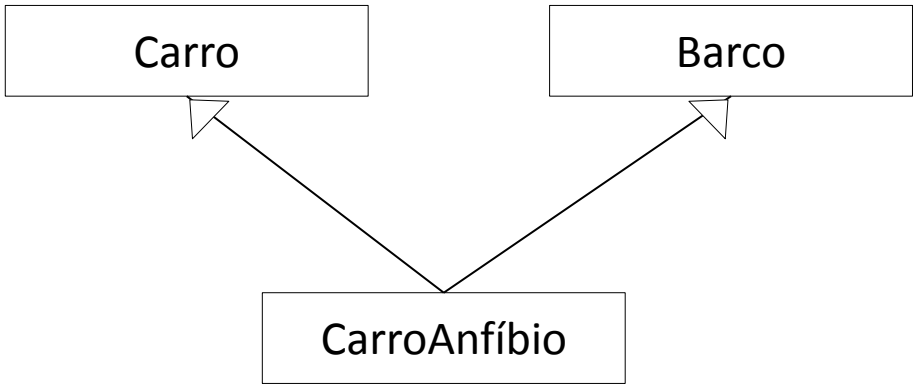
Com esta associação é fácil determinar que cavalos participaram de uma corrida.
Para saber de quais corridas o cavalo participou, é preciso a coleção de corridas e,
Para cada uma, verificar se o cavalo em questão participou da corrida.
Trade-off entre simplicidade de implementação e desempenho, caso a coleção de Corridas for muito grande.

Herança

- Simples



- Múltipla



Deve-se evitar herança múltipla

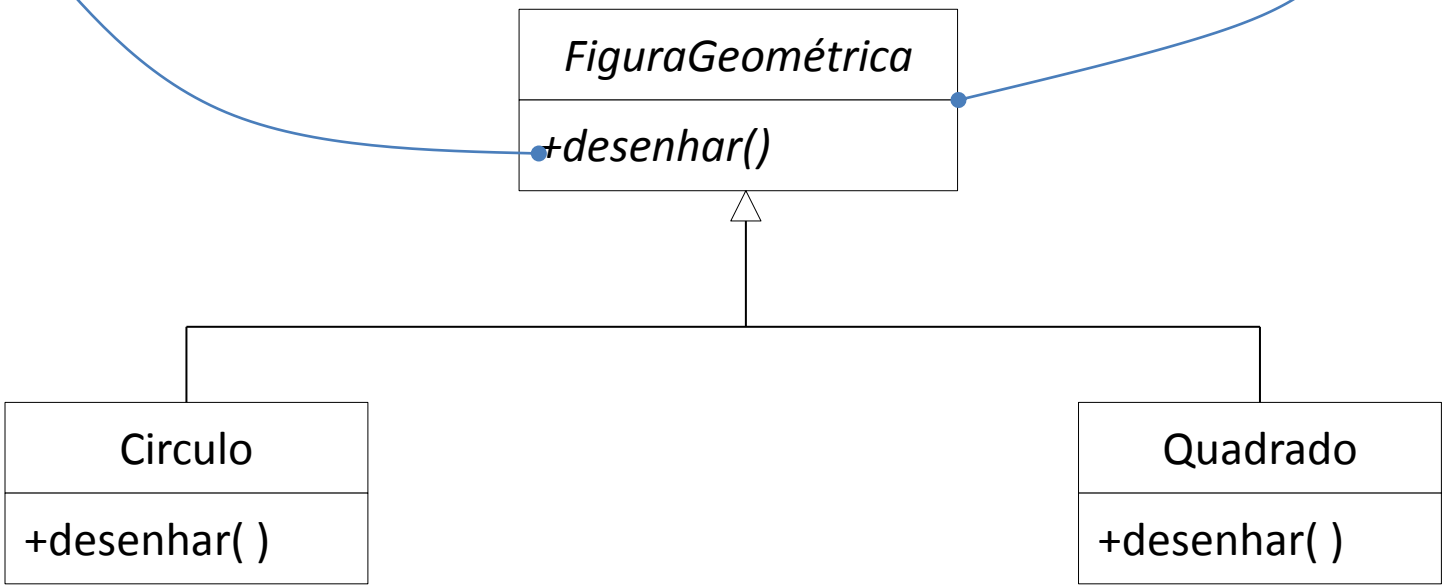
1. Difícil de entender
2. Nem todas as linguagens suportam (Java, C# e Smalltalk)
3. Linguagens de programação mais modernas possuem a alternativa de *interfaces*

Herança

- Classes abstratas

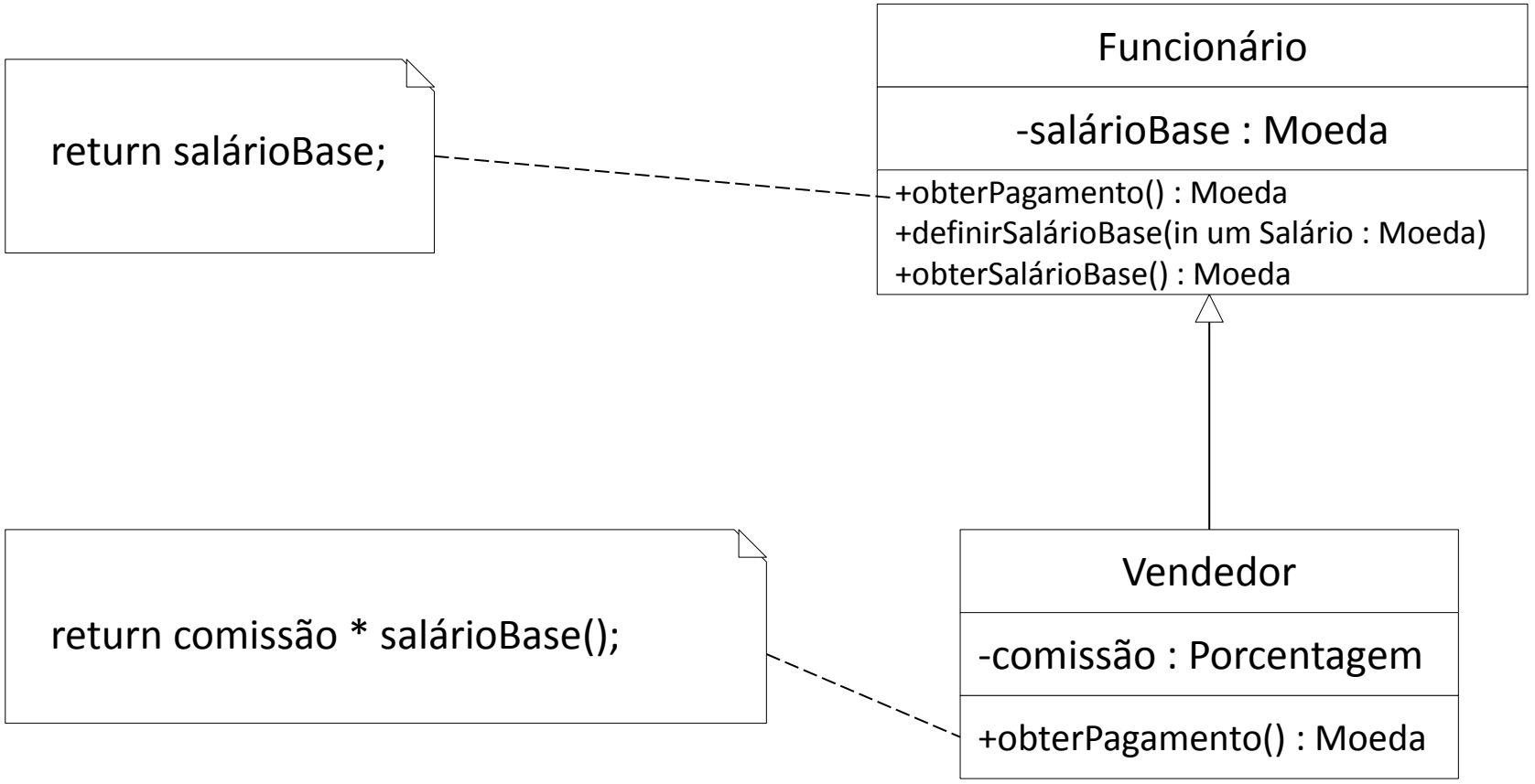
Subclasses devem implementar
Esta operação para serem concretas

Classe e operação abstrata
Em itálico



Herança

- Operações polimórficas



Herança

- Interfaces
  - Representa a “cara” de um objeto, i.e., serviços que um objeto fornece
  - Objeto pode ter várias interfaces
    - “caras diferentes” a diferentes membros da comunidade

Herança

- Interfaces
  - Exemplo: classe Bicicleta, que pertence a uma hierarquia de classes (veículos)
    - Classe bicicleta define o que uma bicicleta pode fazer em termos do comportamento do veículo
    - Mas ela pode interagir com o mundo de outras formas. Pode ser um dos produtos de um sistema de vendas, com outros tipos de informação
    - Para ser manipulado pelo sistema de vendas, um objeto Bicicleta deve estar em concordância com o protocolo ou contrato de comportamento predefinido

Herança

- Interfaces
  - Protocolo
    - Conjunto não vazio de assinaturas de operações
    - Cada assinatura apresenta o nome da operação, juntamente com a definição de seus eventuais parâmetros e tipo de retorno.
    - Não contém a implementação das operações nela definidas
    - Em uma linguagem de programação, estes protocolos são denominados interfaces

Herança

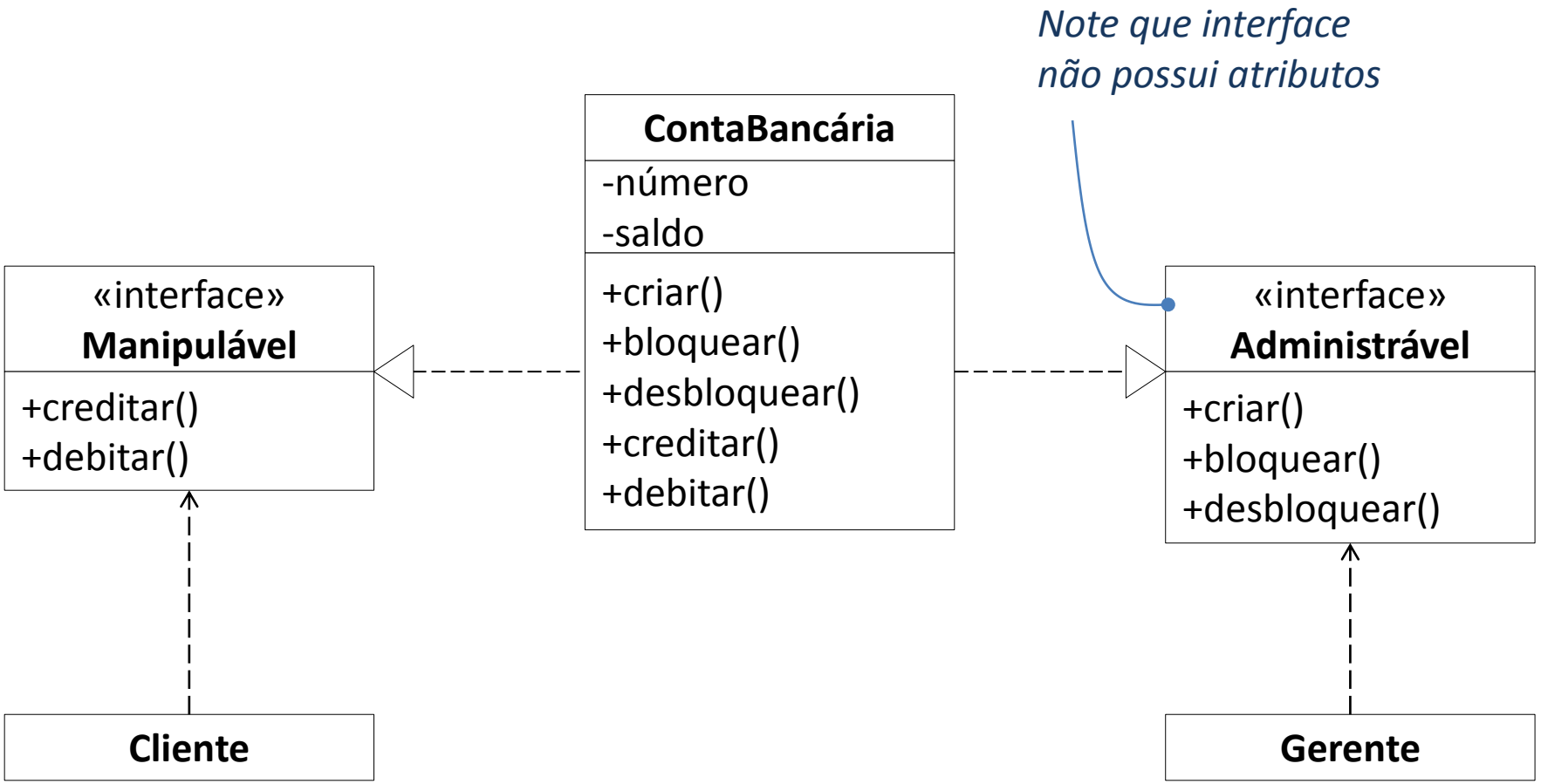
- Interfaces
  - Classe Bicicleta possui uma interface produto
  - Classe Geladeira possui uma interface produto
  - i.e., o sistema de vendas pode tratar de forma indistinta sem conhecer a classe dos objetos
  - Conclusão
    - Não há necessidade de se definir relacionamentos diretos entre classes que, em situações normais, não estariam relacionadas
    - Em vez disso, pode-se definir interfaces para que essas classe se comuniquem

Herança

- Interfaces
  - Objetivos
    - Capturar semelhanças entre classes não relacionadas sem forçar relacionamentos entre elas
    - Declarar operações que uma ou mais classes devem implementar
    - Revelar as operações de um objeto, sem revelar a sua classe
    - Facilitar o desacoplamento entre elemntos de um sistema

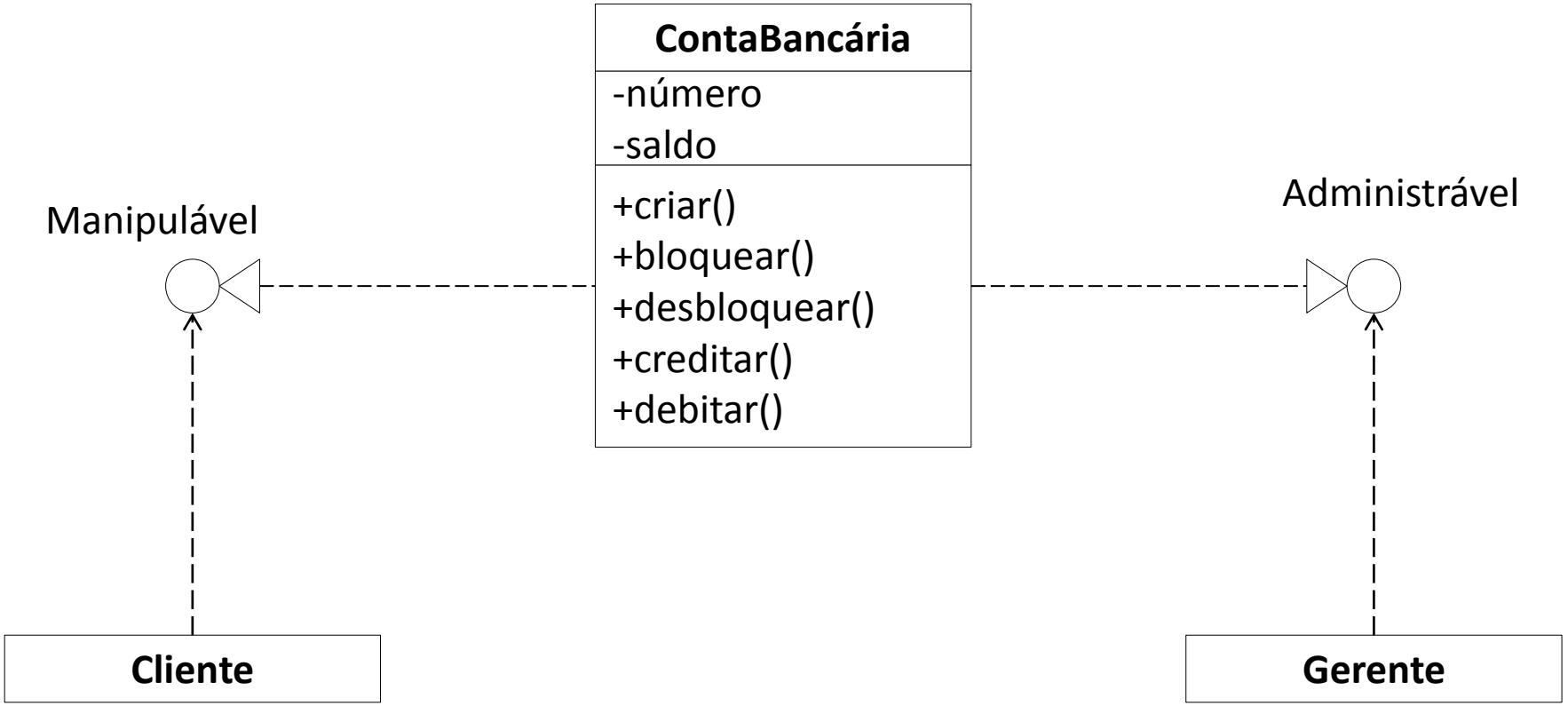
Herança

- Interfaces

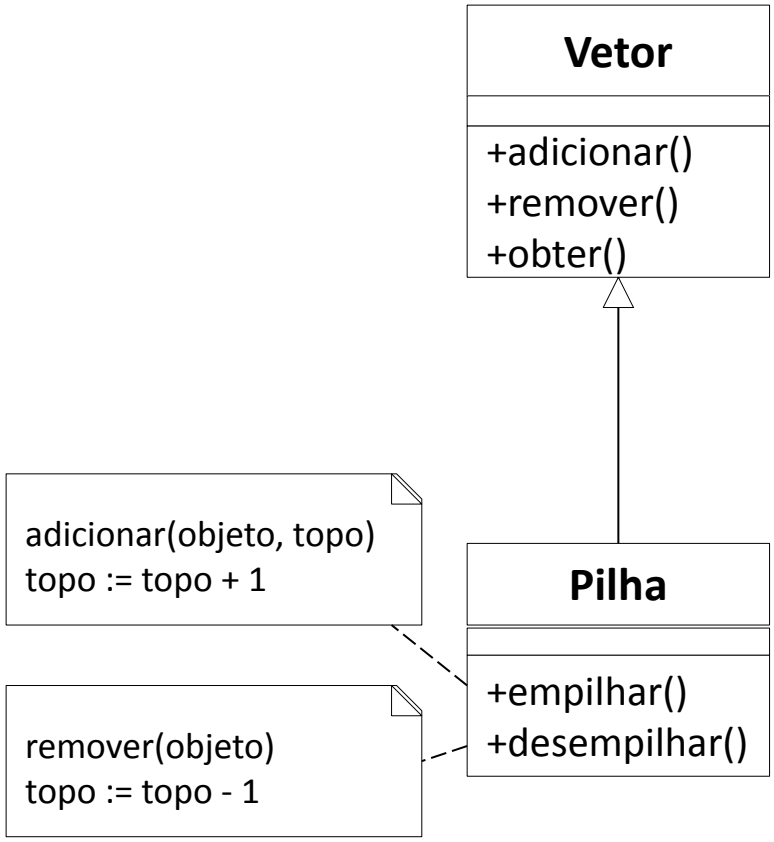


Herança

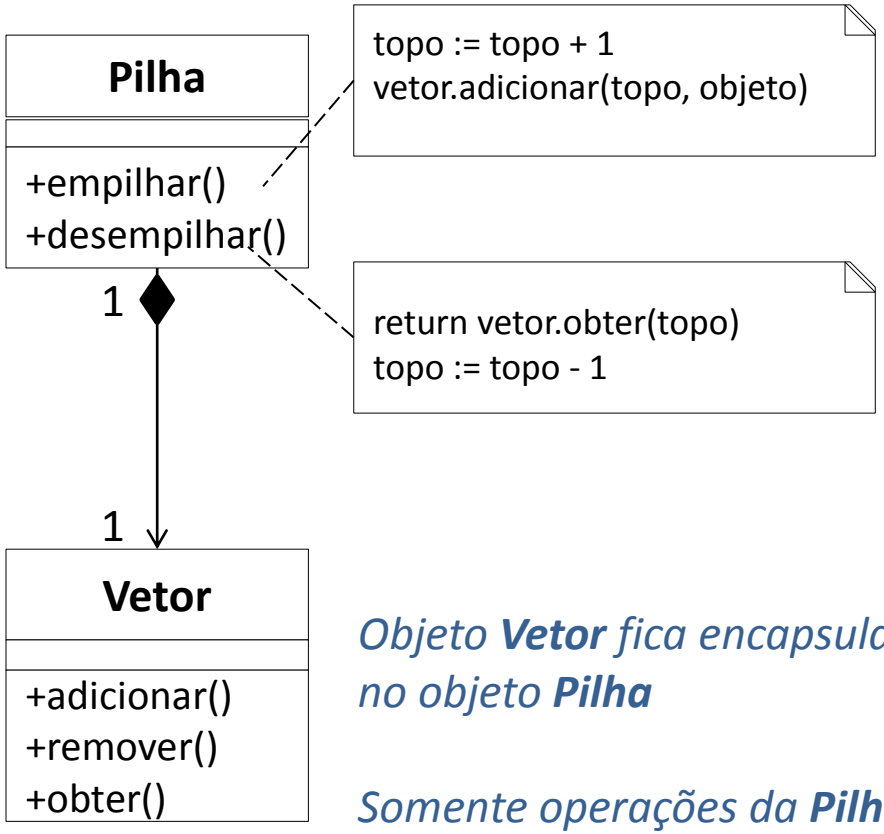
- Interfaces – notação simplificada



Herança



Reuso através da delegação



*Objeto **Vetor** fica encapsulado no objeto **Pilha***

*Somente operações da **Pilha** são acessadas*

Herança
Reuso através da delegação

- A delegação é mais genérica que a herança
  - Objeto pode reutilizar o comportamento de outro sem que precise ser subclasse
  - Compartilhamento de comportamento e o reuso podem ser realizadas em tempo de execução
    - Na herança o reuso é especificado estaticamente
    - No entanto delegação pode apresentar erda de desempenho, uma vez que precisa de mensagem de um objeto para outro

Herança
Reuso através da delegação

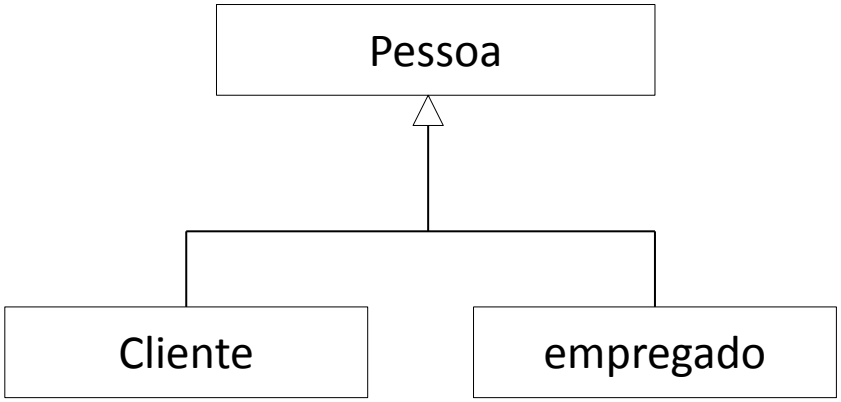
- Não se recomenda usar herança
  - Para representar papéis de uma superclasse
  - Quando a subclasse for herdar propriedades que não se aplicam a ela
  - Quando um objeto de uma subclasse pode se transformar em um objeto de outra subclasse
    - Ex: um objeto da classe Cliente se transforma em um objeto da classe Funcionário

Herança
Classificação Dinâmica

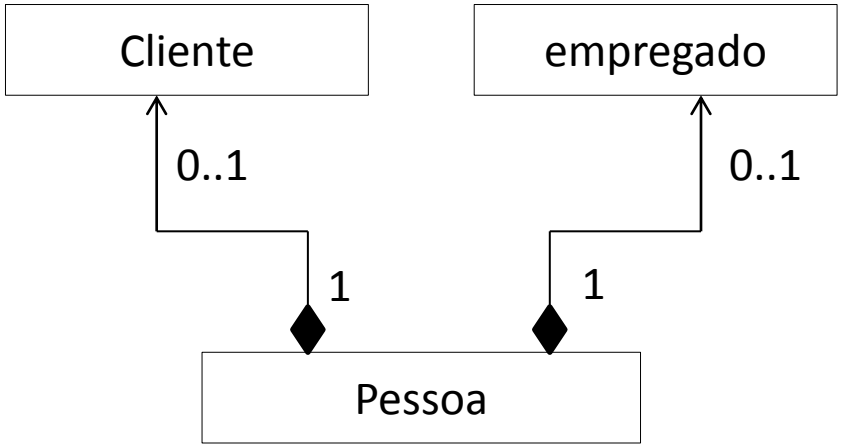
- Herança = classificação estática
  - Desde o momento em que um objeto é instanciado até o momento em que ele é destruído, sua classe permanece a mesma
  - Problema
    - Como especificar e implementar um relacionamento de herança com classificação dinâmica (metamorfose)
      - Ex: empresa com empregados e clientes. Pessoa pode ser em um momento cliente, em outro, empregado. Pode ser que este empregado seja desligado da empresa e continue como cliente.
    - Um mesmo objeto pode pertencer a diferentes classes durante a sua vida, mais do que isso, pode pertencer a múltiplas classes simultaneamente

Herança
Classificação Dinâmica

- Solução
  -



Usar delegação



Design Pattern

- Um padrão de projeto corresponde a um esboço de uma solução reusável para um problema comumente encontrado em um contexto particular. Estudar padrões é uma maneira eficaz de aprender com a experiência dos outros.

Design Pattern

- 3 categorias
  - Criacionais
    - Procuram separar a operação de uma aplicação de como seus objetos são criados
    - Abstract factory, builder, factory method, prototype, singleton
  - Estruturais
    - Proveem generalidade para que a estrutura da solução possa ser estendida no futuro
    - Adapter, bridge, composite, decorator, façade, flyweight, proxy
  - Comportamentais
    - Utilizam herança para distribuir o comportamento entre subclasses, ou agregação e composição para construir comportamento complexo a partir de componentes mais simples
    - Chain of responsability, command, interpreter, iterator, mediator, memento, observer, state, strategy, template method, visitor

Referências

- Ferramentas de modelagem visual
  - Rational Rose (www.rational.com)
  - ASTAH Community (astah.net/editions/community)
- Livros
  - The Unified Modeling Language User Guide, Grady Booch et al
  - Engenharia de software – uma abordagem profissional, Roger S. Pressman
  - Princípios de análise e projetos de sistemas com UML, Eduardo Bezerra
- Especificações
  - www.omg.org

Dúvidas