Análise e Projeto de Sistemas

Universidade Federal do Ceará – UFC
Campus de Quixadá
Prof. Enyo José

Índice

- Introdução
- Transformação de classes de análise em classes de projeto
- Especificação de atributos
- Especificação de operações
- Especificação de associações
- Herança

INTRODUÇÃO

Da Análise ao Projeto

- Os modelos de análise são insuficientes para se ter uma visão completa do sistema para que a implementação comece
  - Antes disso, diversos aspectos referentes à solução a ser utilizada devem ser definidos
  - É na fase de **projeto** que essas definições são realmente feitas

Após a realização do projeto de um sistema OO, os modelos que resultarem estarão em um nível de detalhamento suficiente para que o sistema possa ser codificado.

Da Análise ao Projeto

- A principais atividades realizadas na fase de projeto são
  - Refinamento dos aspectos estáticos e estruturais do sistema
  - Detalhamento dos aspectos dinâmicos do sistema
  - Detalhamento da arquitetura do sistema
  - Definição das estratégias para armazenamento, gerenciamento e persistência dos dados manipulados pelo sistema
  - Realização do projeto da interface gráfica com o usuário
  - Definição dos algoritmos a serem utilizados na implementação

Principais atividades realizadas na fase de projeto

- O modelo de classes de análise descreve as classes do sistema em um nível alto de abstração, através da definição de suas responsabilidades.
  - Pode haver uma classe de análise que resulte em várias classes de projeto ou mais raramente algumas classes de análise resultam em uma única classe de projeto
- Dessa atividade resulta o modelo de classes do projeto

TRANSFORMAÇÃO DE CLASSES DE ANÁLISE EM CLASSES DE PROJETO

O papel da arquitetura do sistema

- Exemplo de documento de arquitetura

Especificação de Classes de Entidade

- A maioria das classes de entidade normalmente permanece na passagem da análise ao projeto
  - Na verdade, classes de entidade são normalmente as primeiras classes a serem identificadas, na análise de domínio
- Durante o projeto, um aspecto importante a considerar sobre classes de entidade é identificar quais delas geram objetos que devem ser persistentes
  - Para essas classes, o seu mapeamento para algum mecanismo de armazenamento persistente deve ser definido

Especificação de Classes de Entidade

- Um aspecto importante é a forma de representar associações, agregações e composições entre objetos de entidade.
  - Essa representação é função da navegabilidade e da multiplidade definidas para a associação, conforme visto mais adiante
- Outro aspecto relevante para classes de entidade é modo como podemos identificar cada um de seus objetos unicamente
  - Isso porque, principalmente em sistemas de informação, objetos de entidade devem ser armazenados de modo persistente
  - Por exemplo, um objeto da classe Aluno é unicamente identificado pelo valor de sua matrícula (atributo do domínio)

Especificação de Classes de Entidade

- A manipulação dos diversos atributos identificadores possíveis em uma classes pode ser bastante trabalhosa
- Para evitar isso, um ***identificador de implementação*** é criado, que não tem correspondente com atributo algum do domínio
  - Possibilidade de manipular identificadores de maneira uniforme e eficiente
  - Maior facilidade quando objetos devem ser mapeados para um SGBDR

Especificação de Classes de Controle

- Com relação às classes de controle, no projeto devemos identificar a real utilidade das mesmas
- É comum a situação em que uma classe de controle de análise ser transformada em duas ou mais classes no nível de especificação
- No refinamento de qualquer classe proveniente da análise, é possível a aplicação de padrões de projeto (*design patterns*)

Especificação de Classes de Controle

- Normalmente, cada classe de controle deve ser particionada em duas ou mais outras classes para controlar diversos aspectos da solução
  - **Objetivo**: de evitar a criação de uma única classe com baixa coesão e alto acoplamento
- Alguns exemplos dos aspectos de uma aplicação cuja coordenação é de responsabilidade das classes de controle
  - Produção de valores para preenchimento de controles da interface gráfica
  - Autenticação de usuários
  - Controle de acesso a funcionalidades do sistema

Especificação de Classes de Controle

- Um tipo comum de controlador é o ***controlador de caso de uso***, responsável pela coordenação da realização de um caso de uso

Responsabilidade de um Controlador
Coordenar a realização de um caso de uso do sistema
Servir como canal de comunicação entre objetos de fronteira e objetos de entidade
Se comunicar com outros controladores, quando necessário
Mapear ações do usuário (ou atores de uma forma geral) para atualizações ou mensagens a serem enviadas a objetos de entidade
Estar apto a manipular exceções provenientes das classes de entidades

Especificação de Outras Classes

- Além do refinamento de classes preexistentes, diversas outros aspectos demandam a identificação de novas classes durante o projeto
  - Persistência de objetos
  - Distribuição e comunicação (e.g., RMI, CORBA, DCOM)
  - Autenticação/Autorização
  - Logging
  - Configurações
  - Threads
  - Classes para testes (*Test Driven Development*)
  - Uso de bibliotecas, componentes e frameworks

ESPECIFICAÇÃO DE: ATRIBUTOS E OPERAÇÕES

Refinamento de Atributos e Métodos

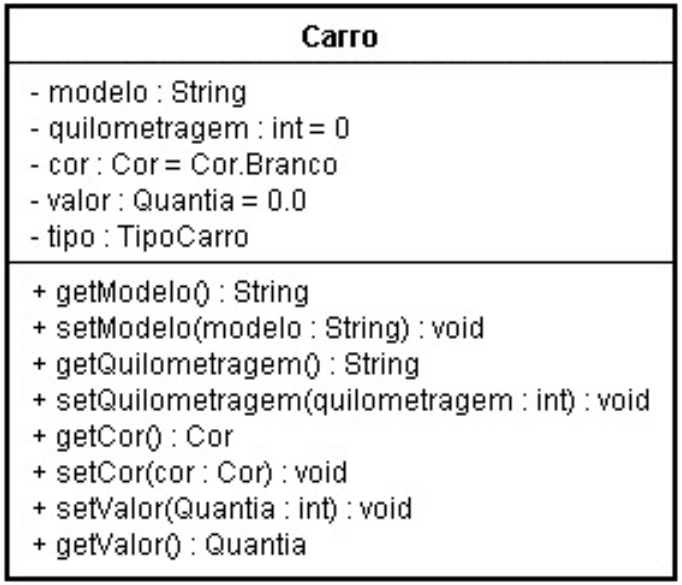
- Os atributos e métodos de uma classe a habilitam a cumprir com suas **responsabilidades**
  - **Atributos**: permitem que uma classe armazene informações necessárias à realização de suas tarefas

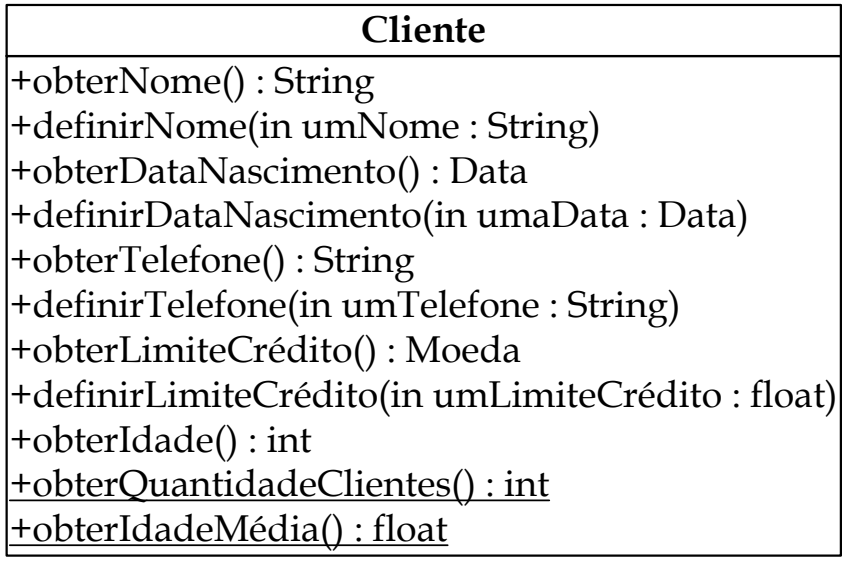
[/] [*visibilidade*] *nome* [multiplicidade] [*: tipo*] [*= valor-inicial*]

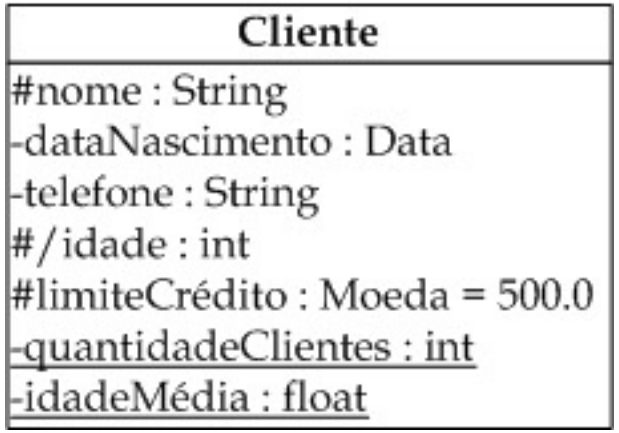
  - **Métodos**: são funções que manipulam os valores do atributos, com o objetivo de atender às **mensagens** que o objeto recebe

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

Sintaxe para Atributos e Operações







OBS: Classes utilitárias podem ser utilizadas como tipos para atributos. (e.g., Moeda, Quantia, TipoCarro, Endereco)

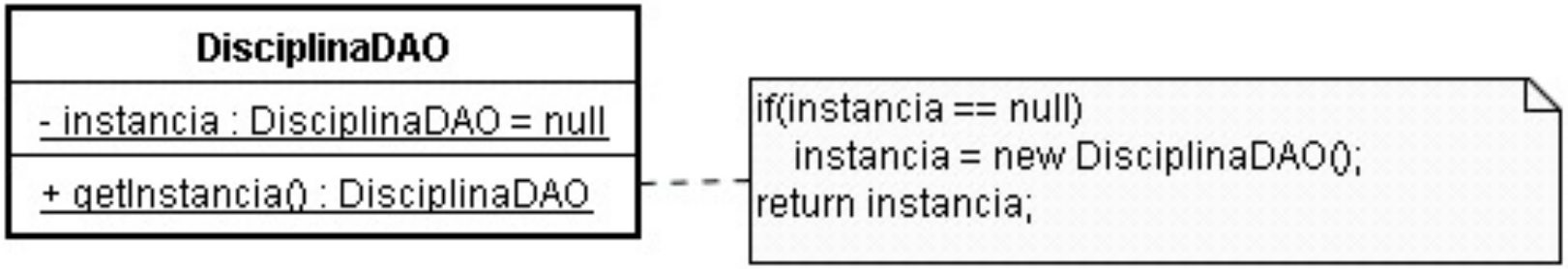
Visibilidade e Encapsulamento

Visibilidade	Símbolo	Significado
Pública	+	Qualquer objeto externo pode obter acesso ao elemento, desde que tenha uma referência para o objeto
Protegida	#	O elemento protegido é visível somente para objetos das subclasses da classe em que foi definido
Privativa	-	O elemento privativo é invisível externamente à classe em que este está definido
Pacote	~	O elemento é visível para qualquer instância de classes que pertençam ao mesmo pacote no qual está a classe proprietária do elemento esta definida

Usualmente, o conjunto das operações públicas de uma classe são chamadas de **interface** dessa classe.

Membros Estáticos

- São representados no diagrama de classes por declarações sublinhadas
  - **Atributos estáticos** (variáveis de classe) são aqueles cujos valores valem para a classe de objetos como um todo
  - **Métodos estáticos** são os que não precisam da existência de uma instância da classe a qual pertencem para serem executados



Projeto de Métodos

- Métodos de construção (criação) e destruição de objetos
- Métodos de acesso (getX/setX) ou propriedades
- Métodos para manutenção de associações (conexões) entre objetos
- Outros métodos
  - Valores derivados, formatação, conversão, cópia e clonagem de objetos, etc.

Projeto de Métodos

- Alguns métodos devem ter uma operação inversa óbvia
  - Exemplo: habilitar e desabilitar; tornarVisível e tornarInvisível; adicionar e remover; depositar e sacar, etc.
- Operações para desfazer ações anteriores

Operações para Manutenção de Associações (Exemplo)

```
public class Turma {
    private Set<OfertaDisciplina> ofertasDisciplina = new HashSet();

    public Turma() {
    }

    public void adicionarOferta(OfertaDisciplina oferta) {
        this.ofertasDisciplina.add(oferta);
    }

    public boolean removerOferta(OfertaDisciplina oferta) {
        return this.ofertasDisciplina.remove(oferta);
    }

    public Set getOfertasDisciplina() {
        return Collections.unmodifiableSet(this.ofertasDisciplina);
    }
}
```

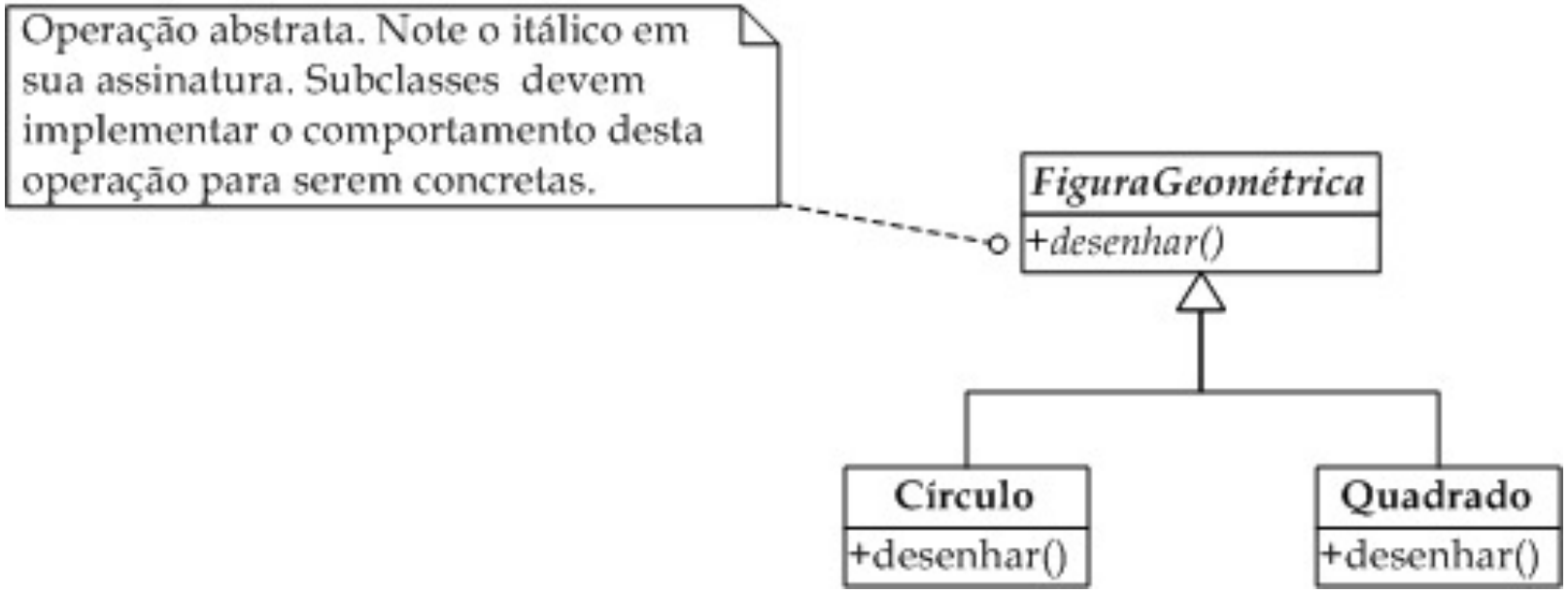
Detalhamento de Métodos

- Diagramas de interação fornecem um indicativo sobre como métodos devem ser implementados
- Como complemento, notas explicativas também são úteis no esclarecimento de como um método deve ser implementado
- O diagrama de atividades também pode ser usado para detalhar a lógica de funcionamento de métodos mais complexos

Operações Abstratas

- Uma classe abstrata possui ao menos uma ***operação abstrata***, que corresponde à especificação de um serviço que a classe deve fornecer (sem método)
- Quando uma subclasse herda uma operação abstrata e não fornece uma implementação para a mesma, esta classe também é abstrata

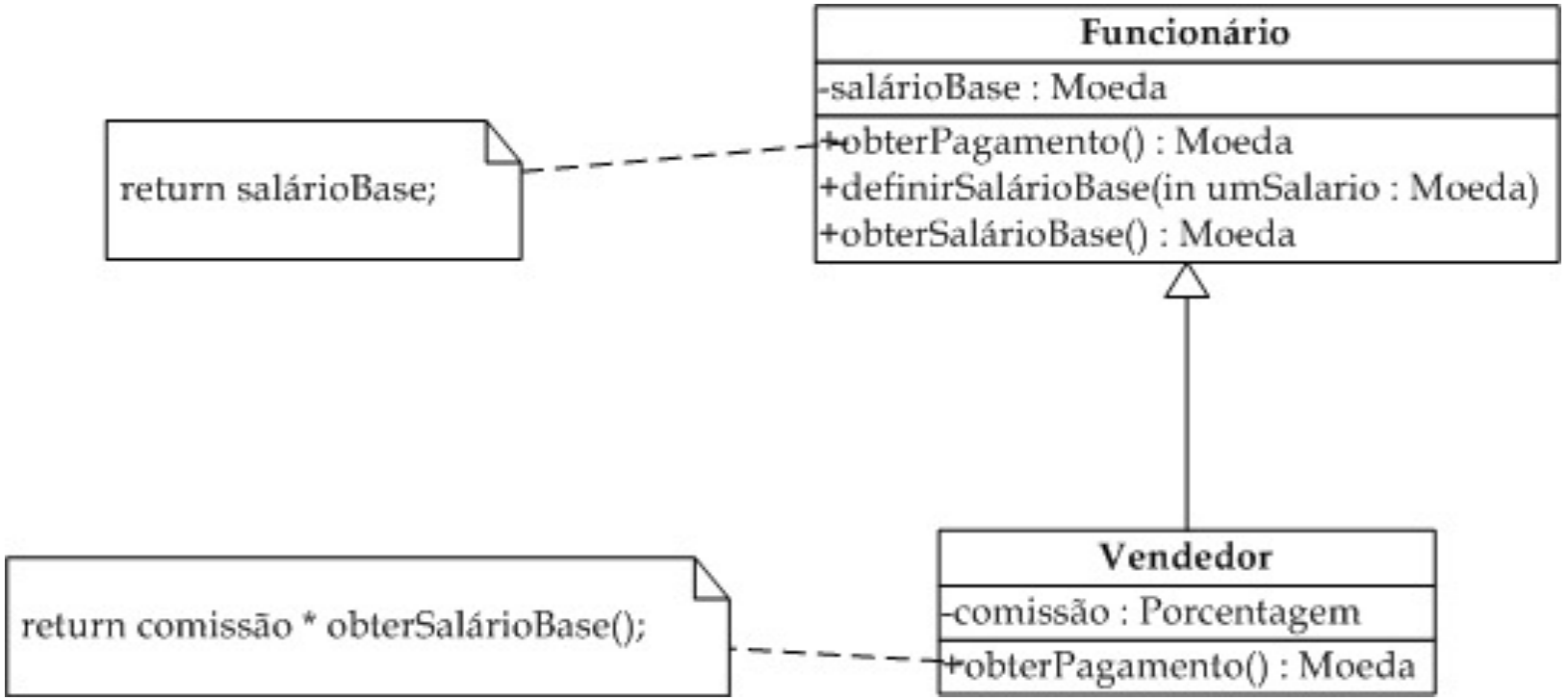
Operações Abstratas



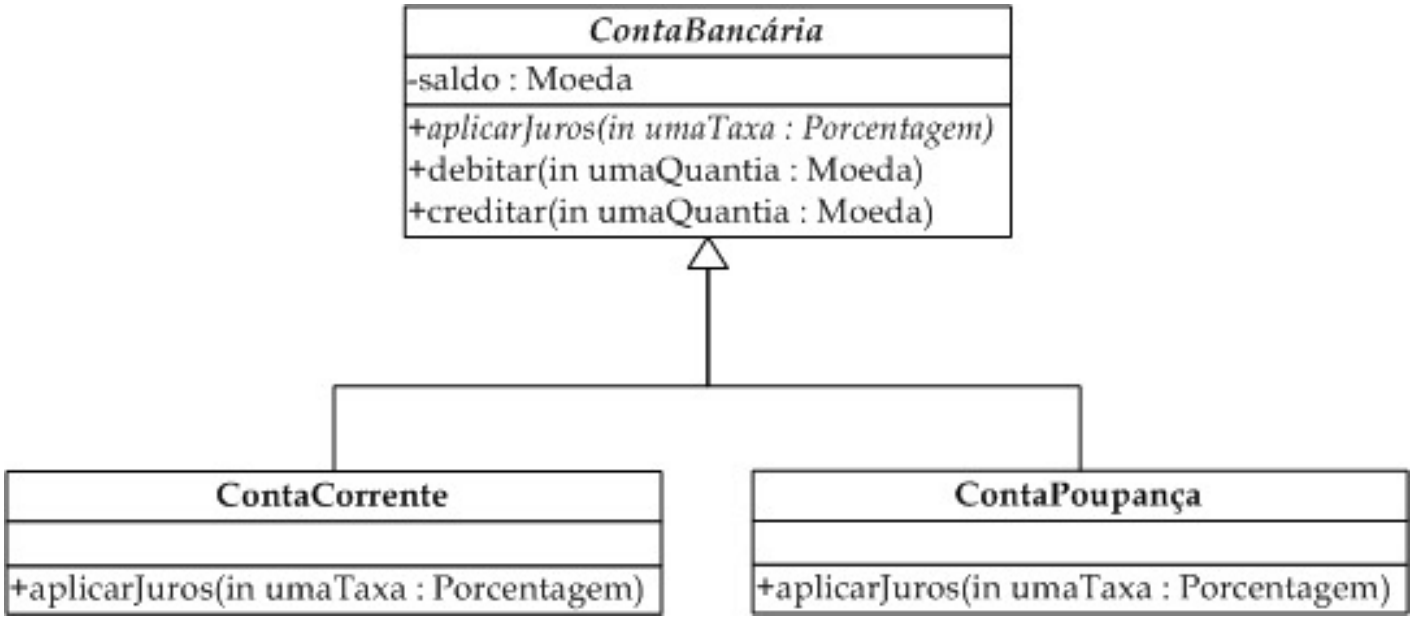
Operações Polimórficas

- Uma subclasse herda todas as propriedades de sua superclasse que tenham visibilidade pública ou protegida
- Entretanto, pode ser que o comportamento de alguma operação herdada seja diferente para a subclasse
- Nesse caso, a subclasse deve redefinir o comportamento da operação
  - A assinatura da operação é reutilizada
  - Mas, a implementação da operação (ou seja, seu método) é diferente

Operações Polimórficas



Operações Polimórficas



ESPECIFICAÇÃO DE ASSOCIAÇÕES, AGREGAÇÕES E COMPOSIÇÕES

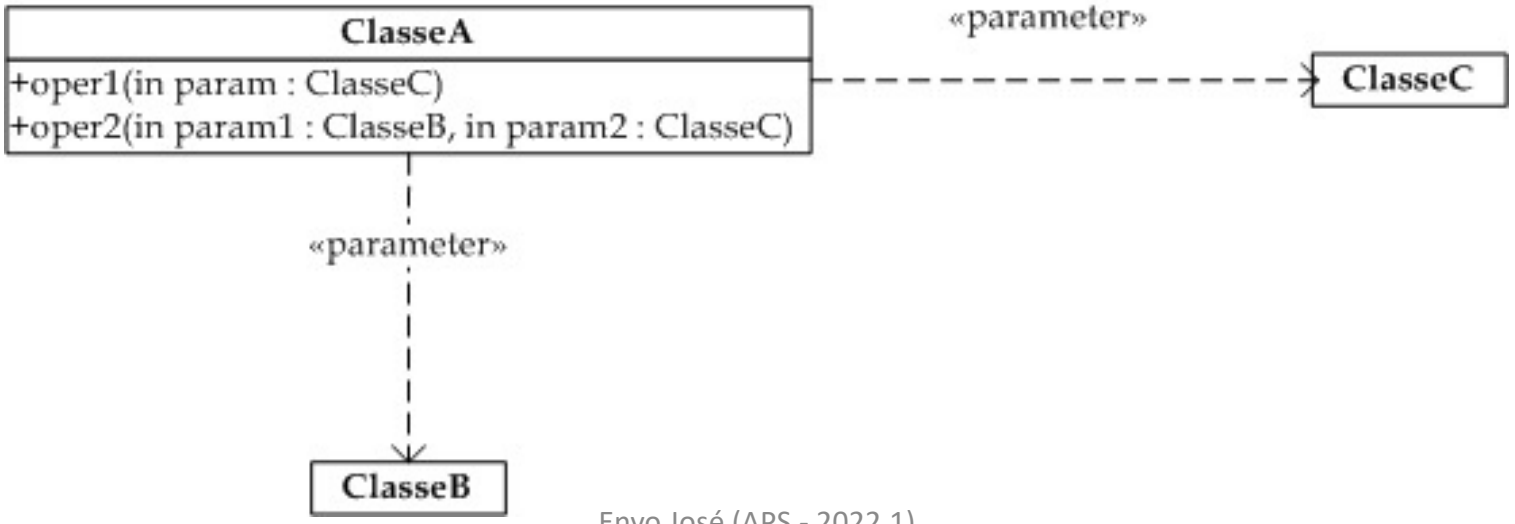
O Conceito de Dependência

- O ***relacionamento de dependência*** indica que uma classe depende dos serviços (operações) fornecidos por outra classe

- Entretanto, há também as ***dependências não estruturais***
  - Na **dependência por variável global**, um objeto de escopo global é referenciado em algum método da classe dependente
  - Na **dependência por variável local**, um objeto recebe outro como retorno de um método, ou possui uma referência para o outro objeto como uma variável local em algum método
  - Na **dependência por parâmetro**, um objeto recebe outro como parâmetro em um método

O Conceito de Dependência

- Dependências não estruturais são representadas na UML por uma linha tracejada direcionada e ligando as classes envolvidas
  - A direção é da classe dependente (***cliente***) para a classe da qual ela depende (***fornecedora***).
  - Estereótipos predefinidos: <<global>>, << local>>, << parameter>>

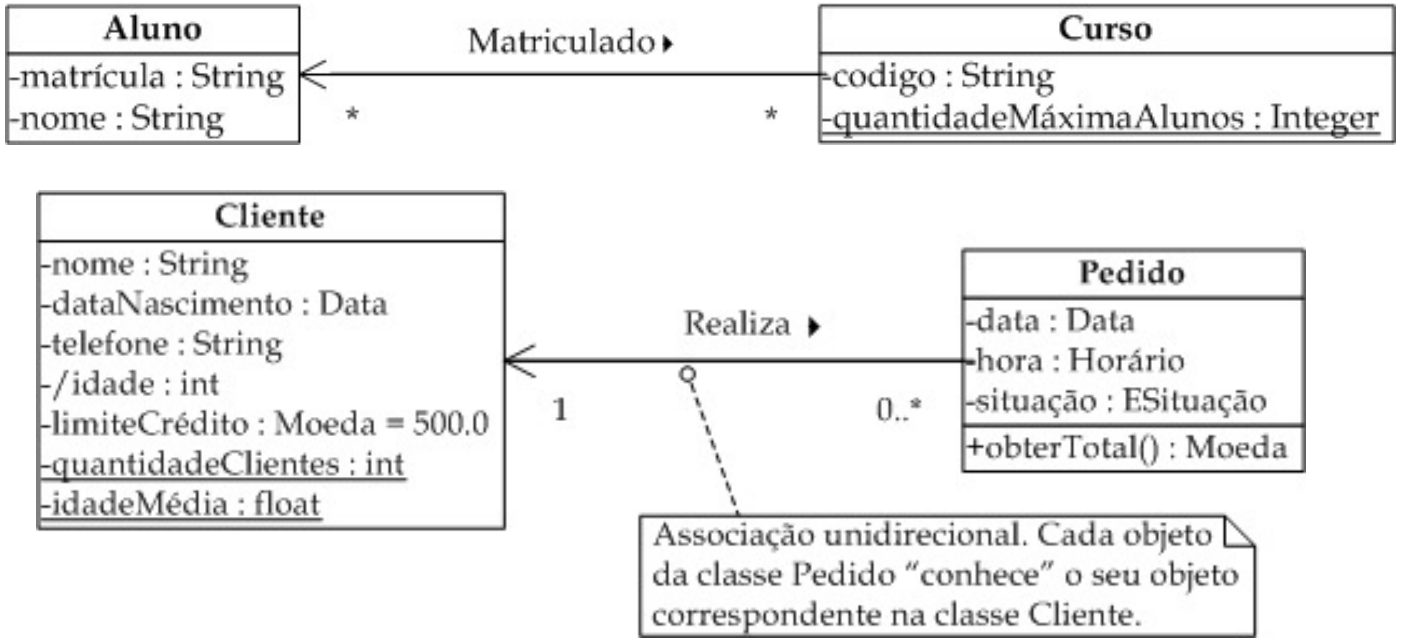


Navegabilidade de Associações

- Associações podem ser **bidirecionais** ou **unidirecionais**
  - Uma **associação bidirecional** indica que há um conhecimento mútuo entre os objetos associados
  - Uma **associação unidirecional** indica que apenas um dos extremos da associação tem ciência da existência da mesma

Navegabilidade de Associações

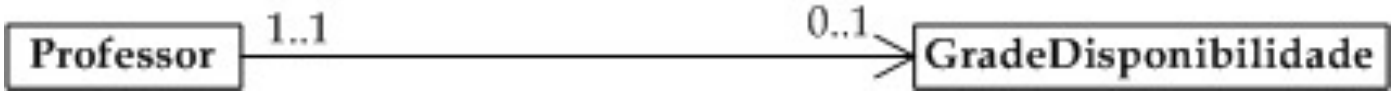
- O sentido de envio das mensagens entre objetos influencia na necessidade ou não de navegabilidade em cada um dos sentidos
  - A classe de onde parte a seta é quem fica com a informação



Implementação de Associações

- Há três casos, em função da conectividade
  - 1:1, 1:N e N:M
- Para uma associação 1:1 entre duas classes A e B
  - Se a navegabilidade é unidirecional no sentido de A para B, é definido um atributo do tipo B na classe A
  - Se a navegabilidade é bidirecional, podemos aplicar o procedimento acima para as duas classes

Associação 1:1




```
public class Professor {
  private GradeDisponibilidade grade;
  ...
}
```

Implementação de Associações

- Para uma associação 1:N ou N:M entre duas classes A e B
  - São utilizados atributos cujos tipos representam coleções de elementos
  - É também comum o uso de classes parametrizadas
    - Idéia básica: definir uma classe parametrizada cujo parâmetro é a classe correspondente ao lado *muitos* da associação
    - O caso N:M é bastante semelhante ao refinamento das associações um para muitos

Agregação

- Agregação é uma associação em que um objeto é parte de outro, de tal forma que a parte pode existir sem o todo


```
public class A {
    private B b;
    public A( ){
    }
    public void setB( B b ){
        this.b = b;
    }
    public B getB( ) {
        return b;
    }
}
```

```
public class B {
    public B( ){
    }
}
```

Composição

- O todo *contém* as partes (e não *referências* para as partes). Quando o todo desaparece, todas as partes também desaparecem


```
public class A {
    private B b;
    public A( ){
        b = new B();
    }
}
```

```
public class B {
    public B( ){
    }
}
```

Discussão Agregação x Composição

- Eu consigo criar instâncias da classe B no exemplo de composição
- Diferença de composição e agregação seria algo mais conceitual?

HERANÇA

Tipos de Herança

- Com relação à quantidade de superclasses que certa classe pode ter
  - ***Herança múltipla*** ou ***Herança simples***

- Com relação à forma de reutilização envolvida
  - Na ***herança de implementação***, uma classe reusa alguma implementação de um "ancestral"

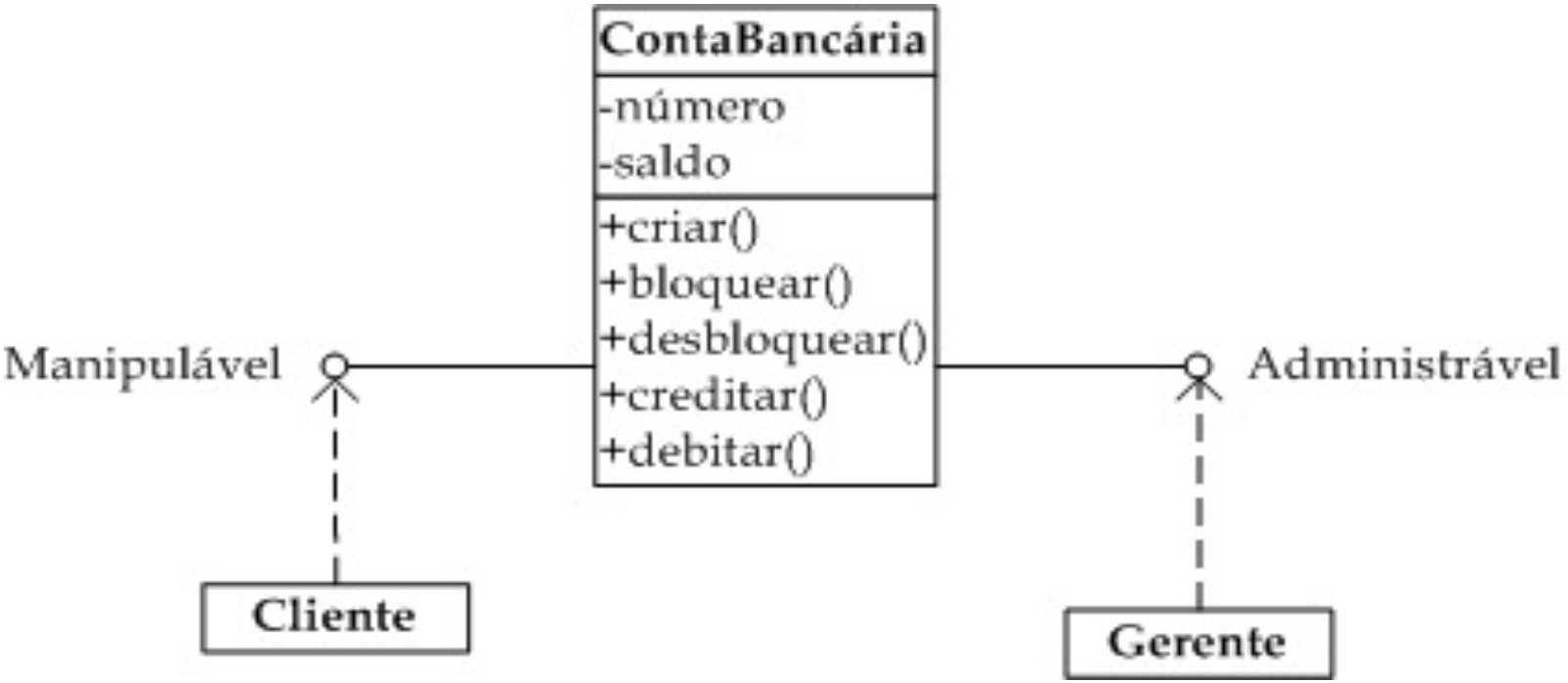
  - Na ***herança de interface*****,** uma classe reusa a interface de um "ancestral" e se compromete a implementar essa interface

Interfaces

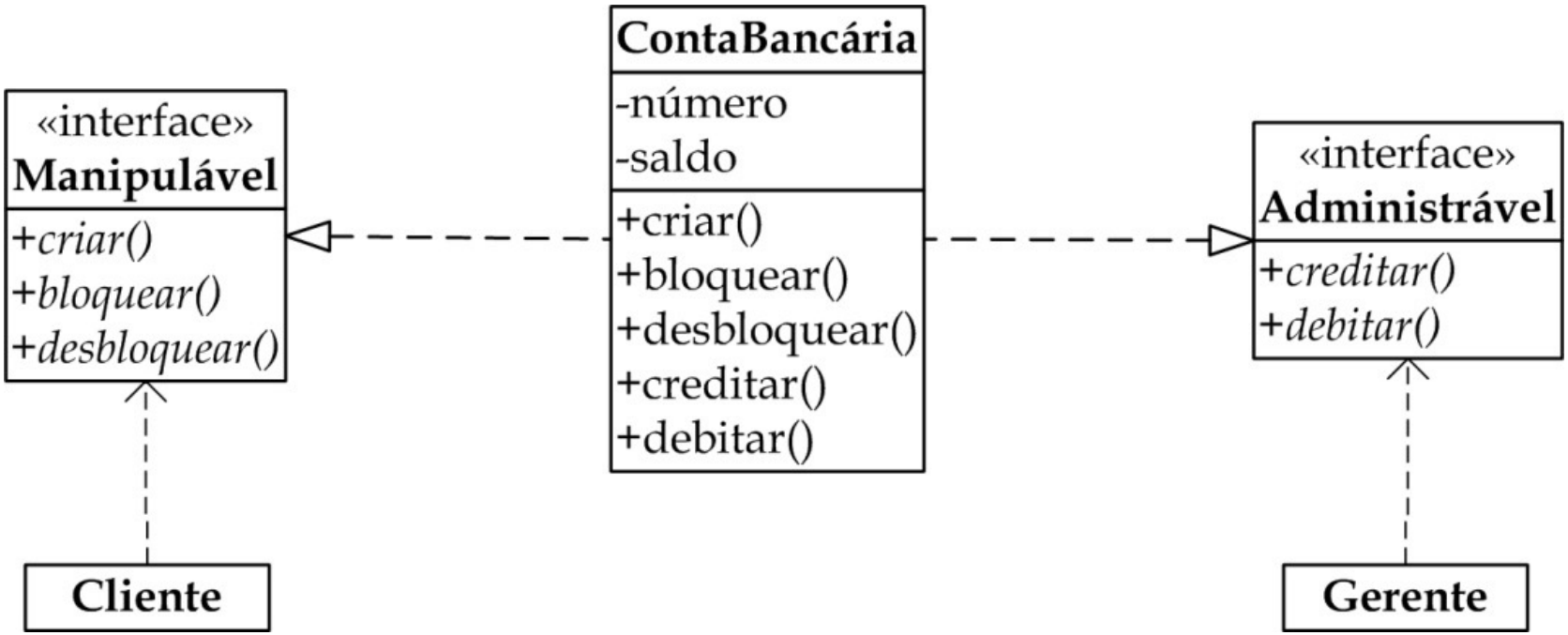
- Uma ***interface*** entre dois objetos compreende um conjunto de **assinaturas de operações** correspondentes aos serviços dos quais a classe do objeto cliente faz uso

- Uma interface pode ser interpretada como um ***contrato de comportamento*** entre um objeto cliente e eventuais objetos fornecedores de um determinado serviço

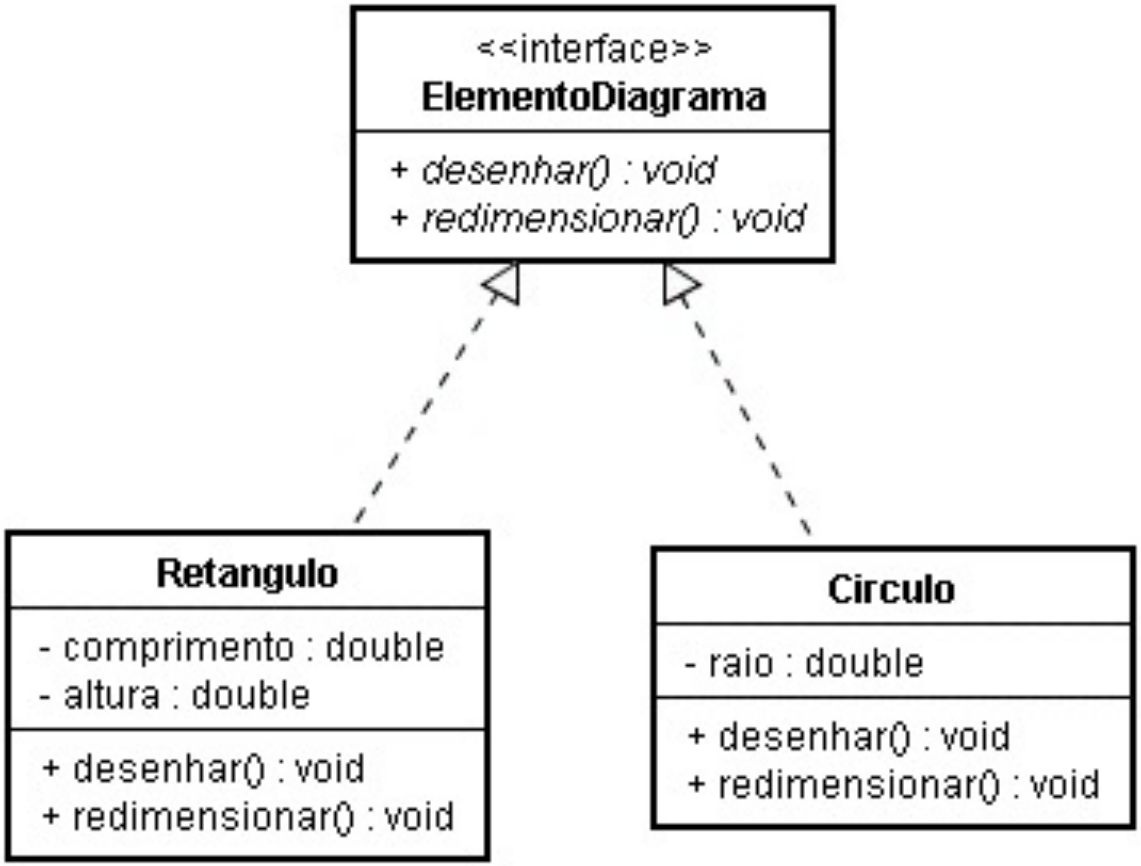
Interfaces


ContaBancária
-número
-saldo
+criar()
+bloquear()
+desbloquear()
+creditar()
+debitar()
Manipulável
Administrável
Cliente
Gerente

Interfaces


«interface»
Manipulável
+criar()
+bloquear()
+desbloquear()
ContaBancária
-número
-saldo
+criar()
+bloquear()
+desbloquear()
+creditar()
+debitar()
«interface»
Administrável
+creditar()
+debitar()
Cliente
Gerente

Interfaces


<<interface>>
ElementoDiagrama
+ desenhar() : void
+ redimensionar() : void
Retangulo
- comprimento : double
- altura : double
+ desenhar() : void
+ redimensionar() : void
Circulo
- raio : double
+ desenhar() : void
+ redimensionar() : void

Atividade

- Modelar o diagrama de classes da especificação de caso de uso que você escreveu nas aulas anteriores de acordo com o padrão de arquitetura apresentado

Referências

- BEZERRA, E. Princípios de Análise e Projeto de Sistemas com UML. 2ª ed. Rio de Janeiro: Elsevier, 2007.

- FOWLER, M. 3. UML Essencial. 3. ed. Porto Alegre: Bookman, 2007.