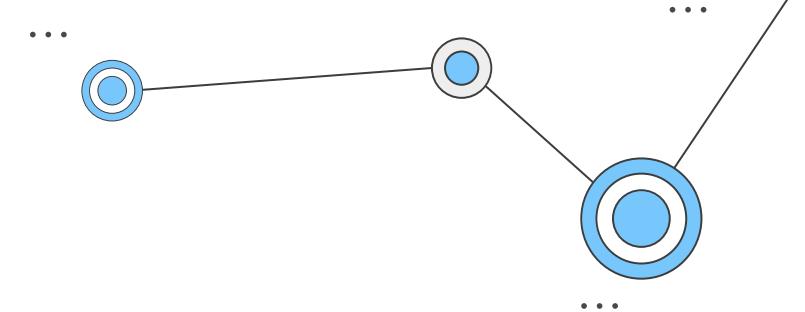
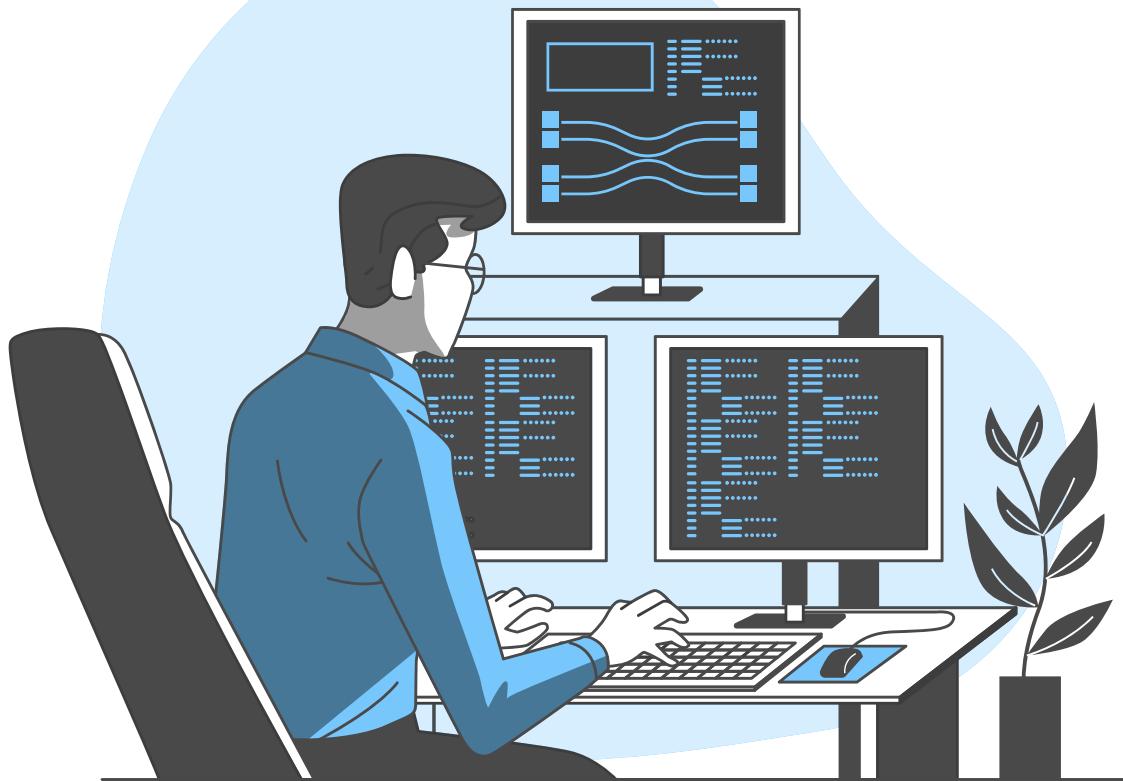




PUC Minas



# Projeto de Software

Prof. Dr. João Paulo Aramuni

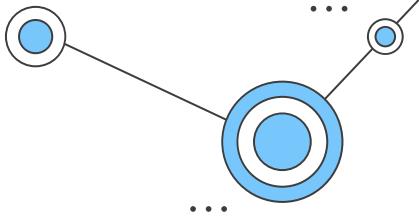
# **Unidade 5**

## **Projeto de Classes**

PDS - Manhã

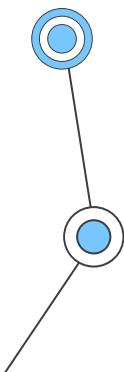
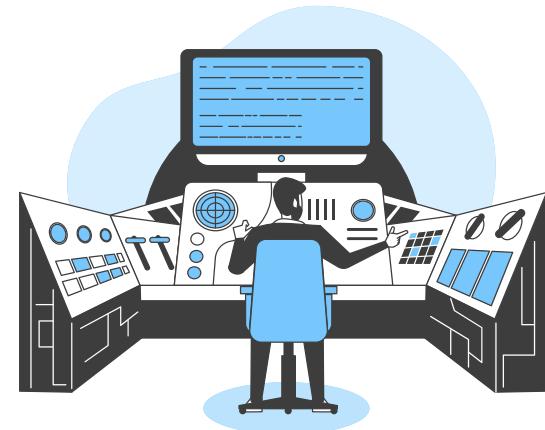
# Projeto de Classes

Diagrama de Classes de projeto



## Objetivos

- Definir o objetivo do projeto de classe e o ciclo de vida onde ela é realizada
- Identificar classes adicionais e relacionamentos necessários para apoiar a implementação dos mecanismos arquitetônicos escolhidos
- Identificar e analisar as transições de estado dos objetos
- Refinar relacionamentos, operações e atributos



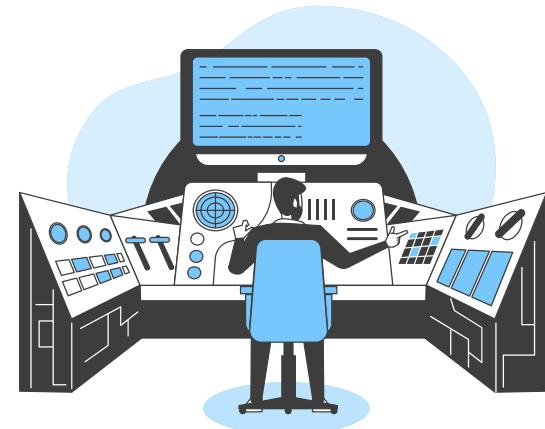
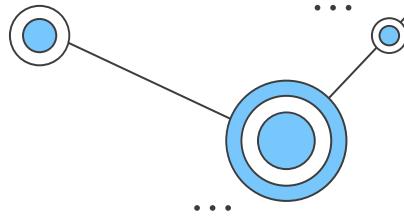
# Projeto de Classes

## Diagrama de Classes de projeto

### Objetivos

Um diagrama de classes na UML pode ser usado múltiplas perspectivas.

- Em um perspectiva conceitual: o diagrama de classes pode ser usado para visualizar um Modelo de Domínio.
- Na perspectiva de software ou projeto: o diagrama pode ser usado para representar um Modelo Lógico. Esse modelo é normalmente denominado de Diagrama de Classe de Projeto (DCP).
  - ✓ O DCP é criado a partir do Modelo de Domínio e de informações obtidas durante a modelagem dinâmica



# Modelo de Domínio x Modelo de Projeto

Modelo conceitual: abstração de conceitos do mundo real

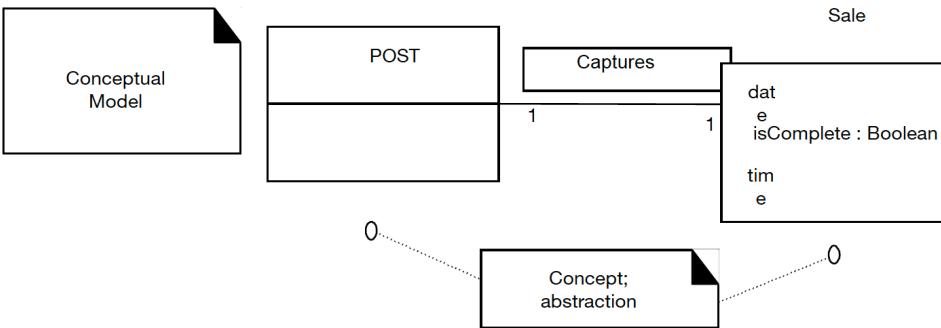
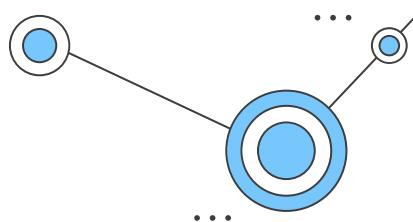
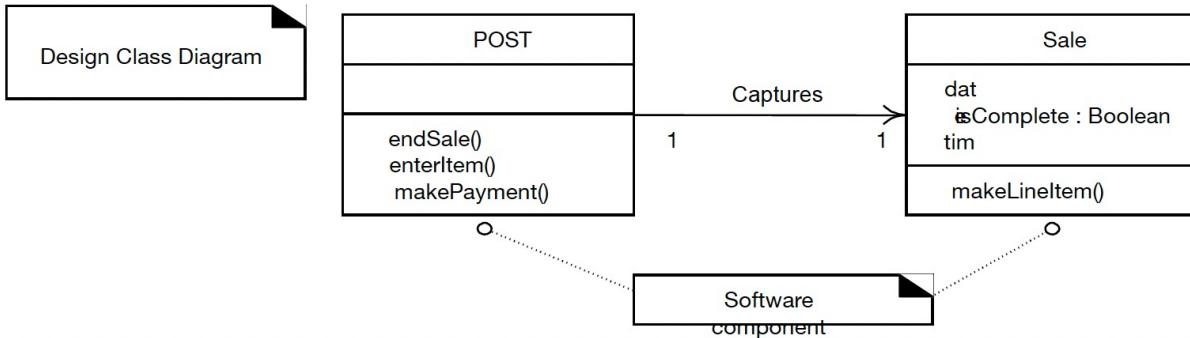
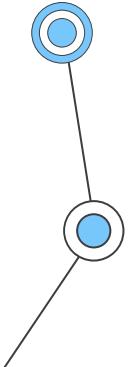


Diagrama de classe de projeto: especificação de componentes de software

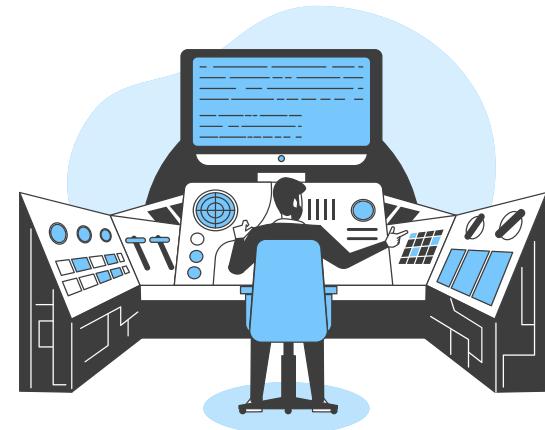
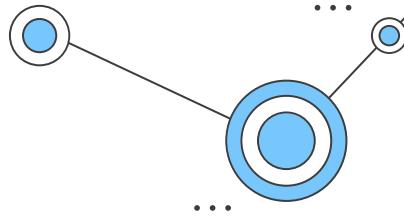


# Projeto de Classes

## Considerações de projeto de classes

Quando fazemos projeto de classes, devemos considerar:

- Como as classes que foram identificadas na análise (entidade, fronteira e controle) serão realizadas na implementação.
- Padrões de projetos aplicáveis
  - ✓ Como os padrões de projeto podem ser usados para ajudar a resolver questões de implementação.
- Mecanismos arquiteturais
  - ✓ Como os mecanismos de arquitetura serão realizados em termos de classes de design definidas.

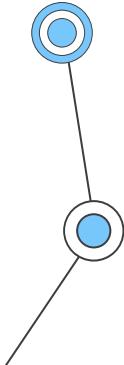
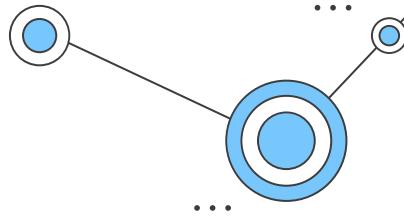


# Projeto de Classes

## Diagrama de Classe de Projeto

Diagrama de classes de projeto:

- A criação de classes de projeto deve acontecer em paralelo a criação de diagramas de interação.
- É construído inicialmente nas iterações da fase de elaboração e refinado nas iterações da fase de construção.
- É passado aos programadores para que eles o implementem.



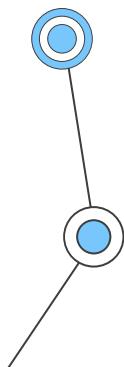
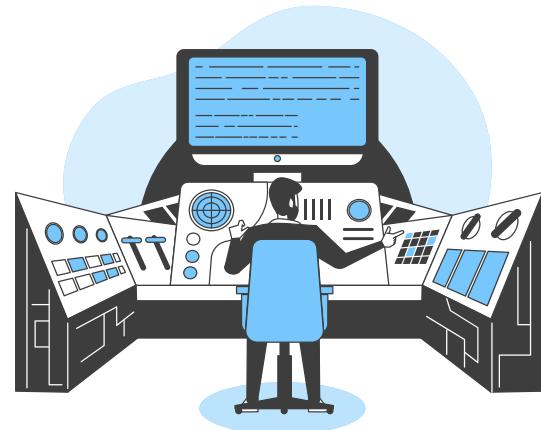
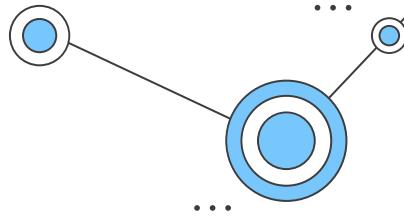
# Projeto de Classes

## Classe de Projeto

O DCP ilustram as especificações para classes de software e interfaces em um sistema.

Em relação ao Modelo Conceitual, o DCP apresenta:

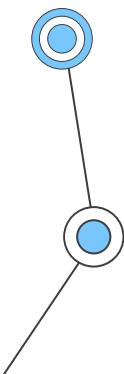
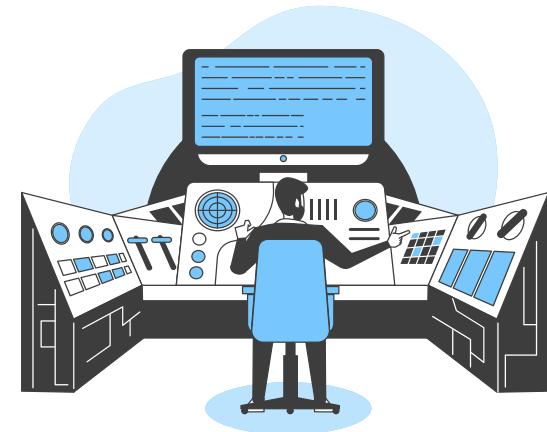
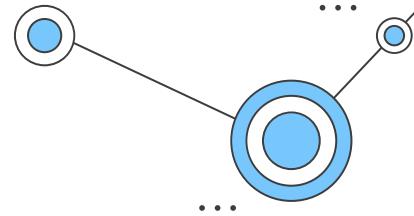
- Possível detalhamento dos atributos e operações
- Adição dos métodos
- Possível detalhamento das associações
- naveabilidade (Adição da direção das associações )
- Possível adição de relacionamentos de dependências
- Possível alteração na estrutura das classes (herança)
- interfaces com suas operações
- Possível criação de atributos privados ou protegidos
- Utilização de padrões de projeto (design patterns)



# Projeto de Classes

## Relacionamentos de classes

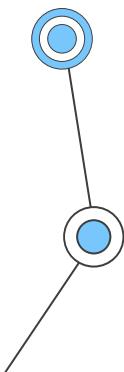
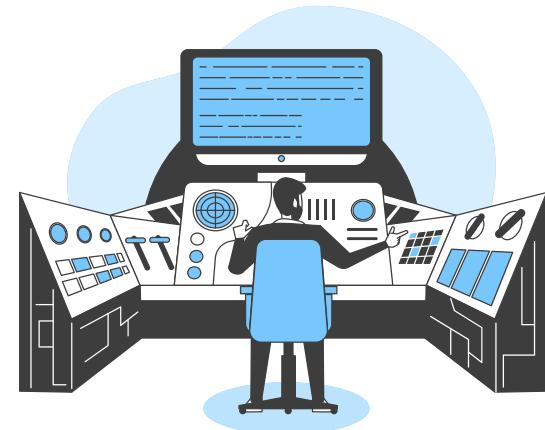
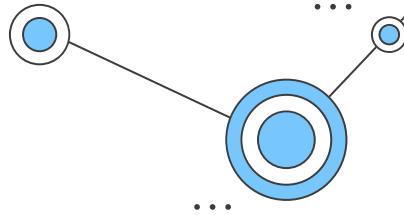
- Na UML, há três tipos de relacionamentos entre objetos de classes:
  - ✓ Associações
  - ✓ Dependências
  - ✓ Generalizações
- Normalmente no modelo de classes de domínio, os relacionamentos entre objetos foram identificados como associações e generalizações.
- É importante entender bem o significado das associações observando a possibilidade substitui-la por um outro relacionamento.



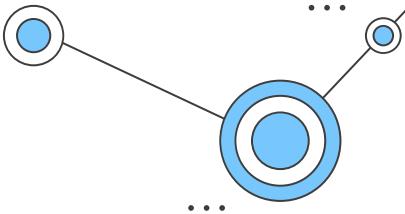
# Projeto de Classes

## Associações

- Associação:
  - São relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes. Pode-se ter associação unária, binária, etc.
- A associação pode existir entre classes ou entre objetos.
  - Uma associação entre a classe Professor e a classe disciplina (um professor ministra uma disciplina) significa que uma instância de Professor (um professor específico) vai ter uma associação com uma instância de Disciplina.
  - Esta relação significa que as instâncias das classes são conectadas, seja fisicamente ou conceitualmente.



# Projeto de Classes



## Associações

- Como saber da existência de uma associação: Sempre que um atributo de uma classe não é de tipo primitivo, temos uma associação desta classe com alguma outra.



Associação com notação textual para indicar que Cliente tem uma referência para uma instância de Endereço

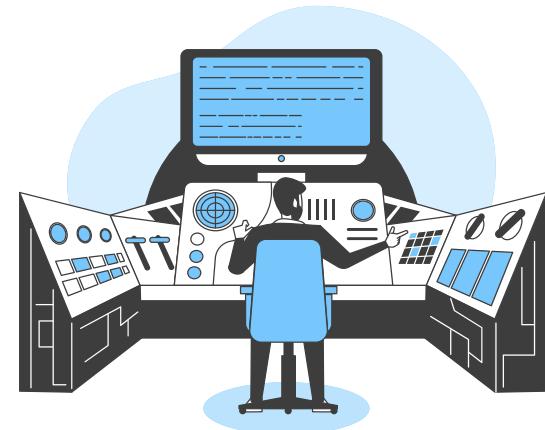
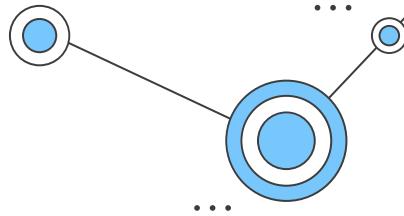


Associação com notação visual

# Projeto de Classes

## Associações

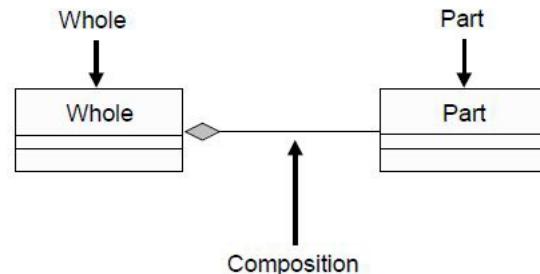
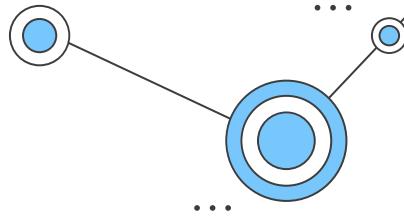
- As associações do DCP são transformadas em variáveis de instância, da mesma forma que os atributos, e terão métodos para alteração e consulta.
- Os atributos geram sempre variáveis cujos tipos são básicos (alfanuméricos).
- As associações geram tipos que são classes de objetos ou estruturas de dados.
- Considerando as diferentes multiplicidades de papel e outras características das associações, haverá algumas distinções a fazer quanto aos métodos associados.



# Projeto de Classes

## Agregações

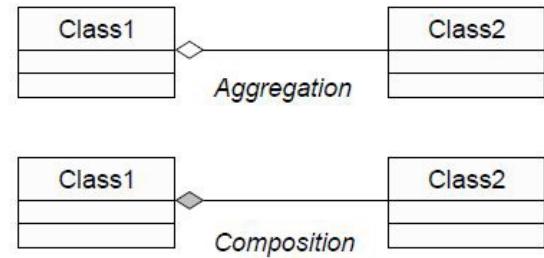
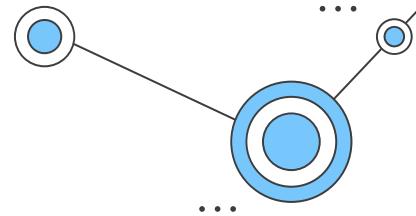
- Agregação - tipo de associação (é parte de, todo/parte) onde o objeto parte é um atributo do todo ; onde os objetos partes somente são criados se o todo ao qual estão agregados seja criado.
  - Pedidos é composto por itens de pedidos.
- Composição - Relacionamento entre um elemento (o todo) e outros elementos (as partes) onde as partes só podem pertencer ao todo e são criadas e destruídas com ele.
  - As partes não podem sobreviver sem o todo



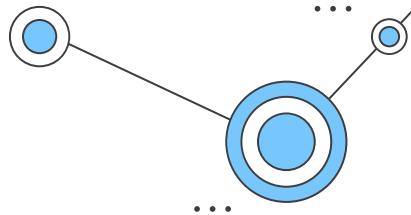
# Projeto de Classes

## Agregações x Composições

- O que difere é o tempo de vida das classe
- Na composição existe uma interdependência entre o 'todo' e as 'partes', em que a definição do 'todo' é incompleta sem as 'partes'.
  - Por exemplo, não faz sentido ter um Pedido se não há nenhum item de pedido a ser entregue.
- Composição deve ser utilizado quando o 'conjunto' e 'parte' devem ter tempos de vida coincidentes.
- Seleção de agregação ou composição irá determinar como a criação de objetos e exclusão são projetados.

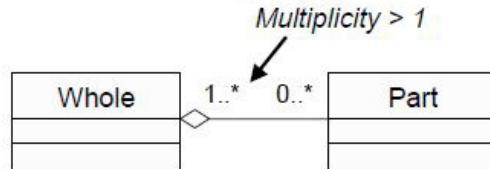


# Projeto de Classes

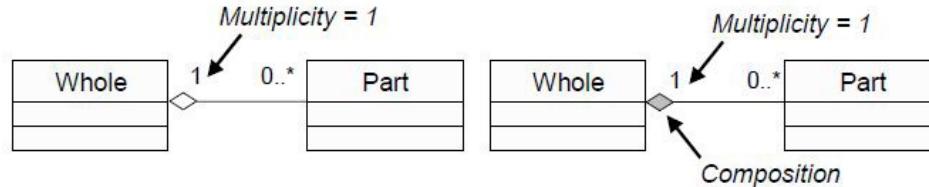


Agregações: compartilhada x não compartilhada

- Compartilhada



- Não compartilhada

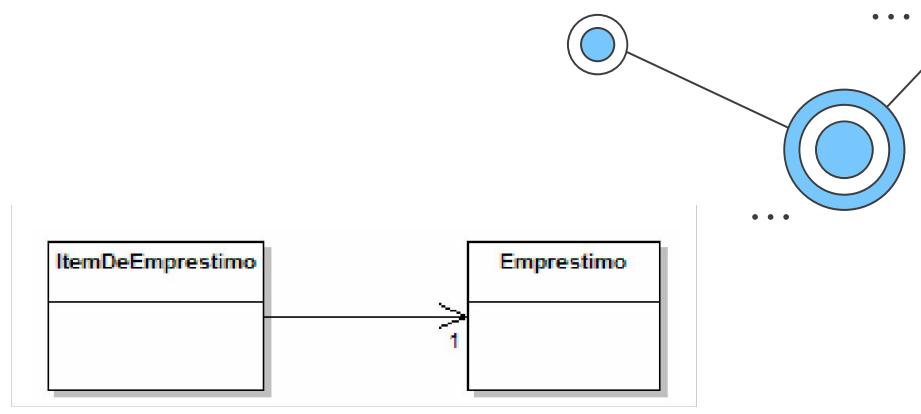


- Por definição uma composição é uma agregação não compartilhada

# Projeto de Classes

Associações devem implementar no mínimo:

- Um método para criar ou redefinir a associação (SET)
- Um método para obter objetos associados (GET)
- Um método para remover a associação (exceto para associações para 1)

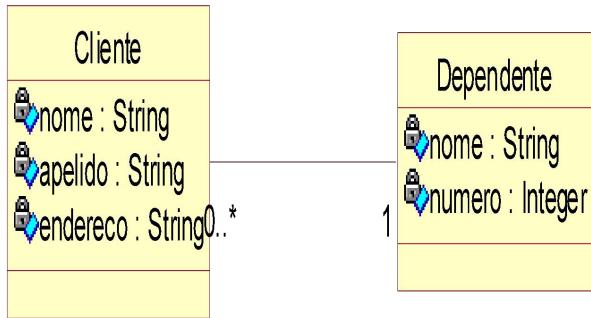


```
class ItemDeEmprestimo {  
    private Emprestimo emprestimo;  
  
    public ItemDeEmprestimo(Emprestimo emprestimo) {  
        this.associaEmprestimo(emprestimo )  
    }  
    public void associaEmprestimo(Emprestimo emprestimo) {  
        this.emprestimo = emprestimo;  
    }  
    public Emprestimo getEmprestimo() {  
        return emprestimo;  
    }  
}
```

# Projeto de Classes

## Associações exemplo 2:

- Suponha uma classe cliente relacionada com uma classe dependente:



```
public class Cliente {  
    //Declaração dos atributos  
    private String nome;  
    private String endereco;  
    private String apelido;  
    protected Dependente  
    dependente;  
    //Construtor padrão  
    public Cliente() {}  
    //Construtor com atributos  
    public Cliente(String nome,  
    String endereco, String  
    apelido) {  
        this.nome = nome;  
        this.endereco = endereco;  
        this.apelido = apelido;  
    }  
}
```

```
//Get  
public String getApelido() {  
    return apelido;  
}  
public String getEndereco() {  
    return endereco;  
}  
public String getNome() {  
    return nome;  
}  
public Dependente  
getDependente() {  
    return dependente;  
}  
//Set  
public void setApelido(String  
apelido) {  
    this.apelido = apelido;  
}  
public void  
setEndereco(String endereco) {  
    this.endereco = endereco;  
}  
public void setNome(String  
nome) {  
    this.nome = nome;  
}  
public void  
setDependente(Dependente  
dependente) {  
    this.dependente = dependente;  
}
```

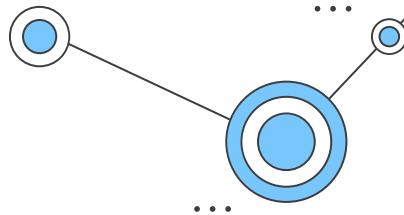
# Projeto de Classes

## Associações exemplo 2:

```
public class Dependente {  
    private String nomeDep;  
    private double numero;  
    //Criação de uma lista de clientes dentro de dependentes  
    //neste caso o dependente possui um relacionamento 1 para n  
    //ou seja o cliente pode ser de vários dependentes  
    protected List<Cliente> listaCliente = new ArrayList<Cliente>();  
    //Construtor padrão  
    public Dependente() {}  
    //Construtor com argumentos  
    public Dependente(String nomeDep, double numero){  
        this.nomeDep = nomeDep;  
        this.numero = numero;  
    }  
}
```

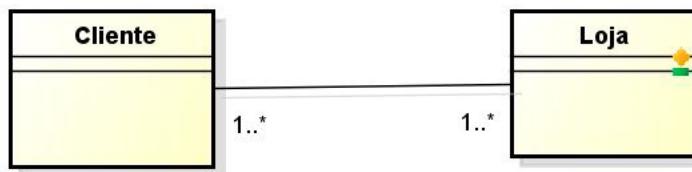
```
//Get  
public String getNomeDep() {  
    return nomeDep;  
}  
public double getNumero() {  
    return numero;  
}  
public List<Cliente> getListaCliente(){  
    return listaCliente;  
}  
//Set  
public void setNomeDep(String nomeDep) {  
    this.nomeDep = nomeDep;  
}  
public void setNumero(double numero) {  
    this.numero = numero;  
}  
public void setListaCliente(List<Cliente>  
    listaCliente) {  
    this.listaCliente = listaCliente;  
}
```

# Projeto de Classes



## Navegabilidade de associações

- Dada a associação abaixo, é necessário ter um atributo de Cliente em Loja e outro de Loja em Cliente?
  - Qual o impacto de ter os dois clientes sem ter necessidade?
    - ✓ Alocação de memória desnecessária
- Posso ter só a definição de um dos atributos?

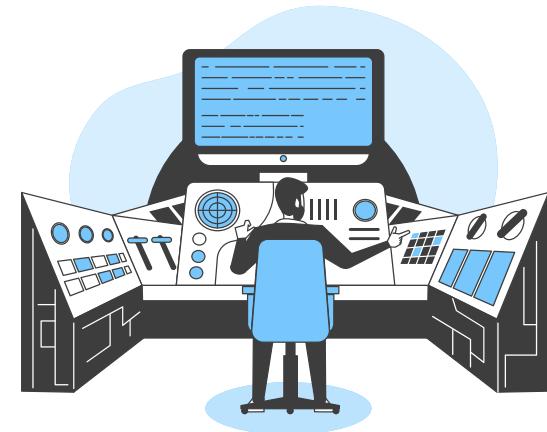
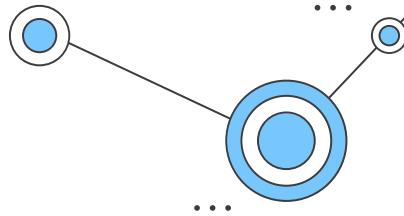


- Usar conceito de NAVEGABILIDADE

# Projeto de Classes

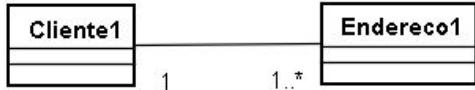
## Navegabilidade de associações

- Navegabilidade é a propriedade do papel que indica que é possível navegar unidirecionalmente por meio da associação de objetos da classe de origem para a classe destino.
- A navegabilidade implica geralmente em visibilidade por atributo (associação).
- Associações, agregações e composições podem ser bidirecionais e unidirecionais.
- Uma associação bidirecional indica que há um conhecimento mútuo entre os objetos associados.
- Na UML, associações são, por omissão, navegáveis em ambos os sentidos.
- Uma associação unidirecional é representada adicionando-se um sentido à seta da associação.



# Projeto de Classes

## Navegabilidade de associações



# Projeto de Classes

## Navegabilidade de associações

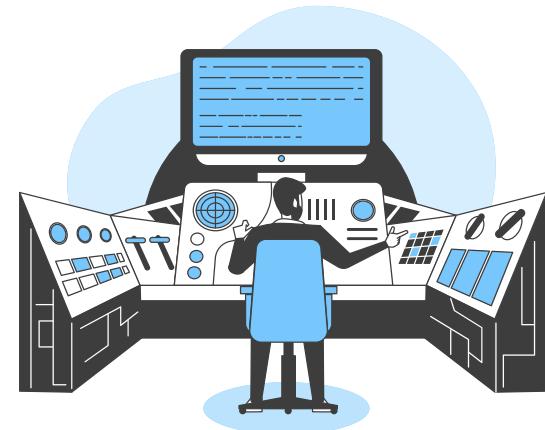
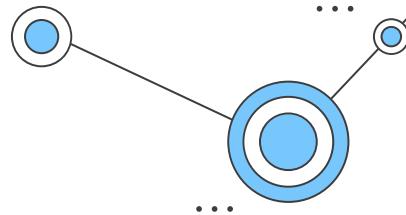
No Modelo de Domínio, as associações são bi-direcionais.

No modelo de classes de projeto, a navegabilidade de todas as associações deve ser definida.

- Algumas associações permanecem bidirecionais.
- Se não houver essa necessidade, recomenda-se transformar a associação em unidirecional.

A escolha do sentido da navegabilidade pode ser feita através dos diagramas de interação.

- Mensagens influenciam na existência ou não de navegabilidade em um sentido.



# Projeto de Classes

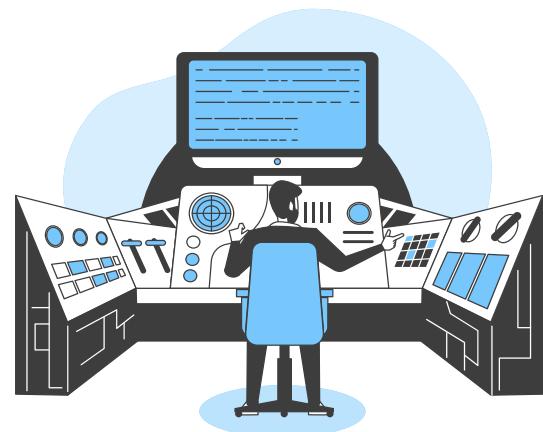
## Navegabilidade de associações

- No projeto, uma associação pode ser uni-direcional; uma seta é adicionada à associação para mostrar que a navegação é em uma direção.



O cliente pode “conversar” com pedido O pedido não pode “conversar” com o cliente.

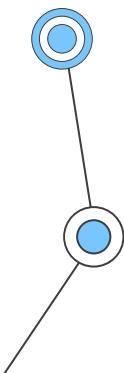
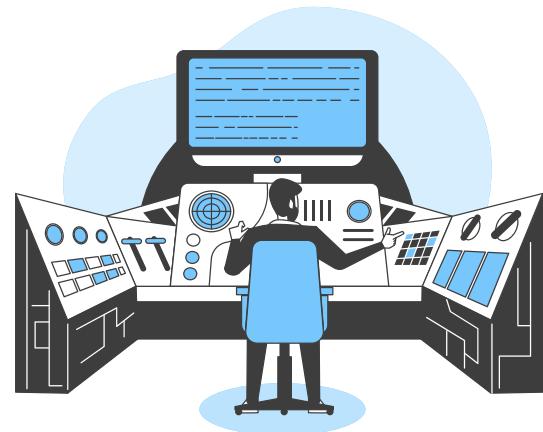
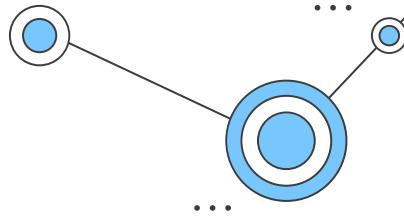
- A necessidade de navegação é revelada por casos de uso e cenários.
- dada uma instância de A, precisamos encontrar todas as instâncias associadas da classe B ou vice-versa?



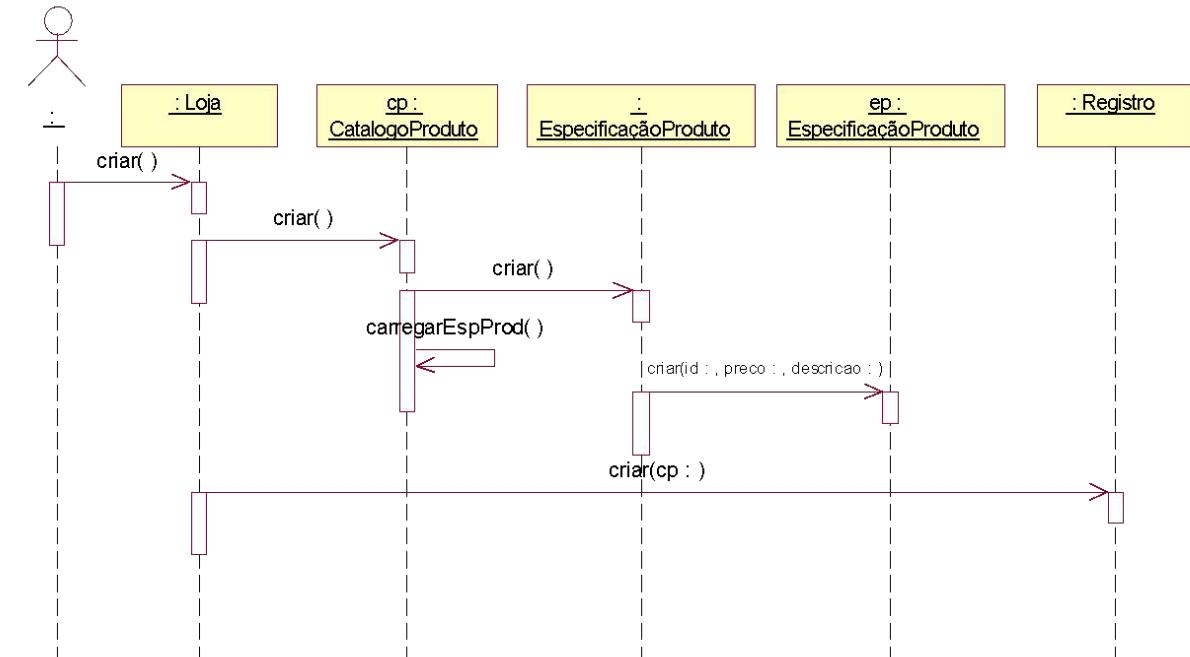
# Projeto de Classes

Navegabilidade: como identificar o sentido

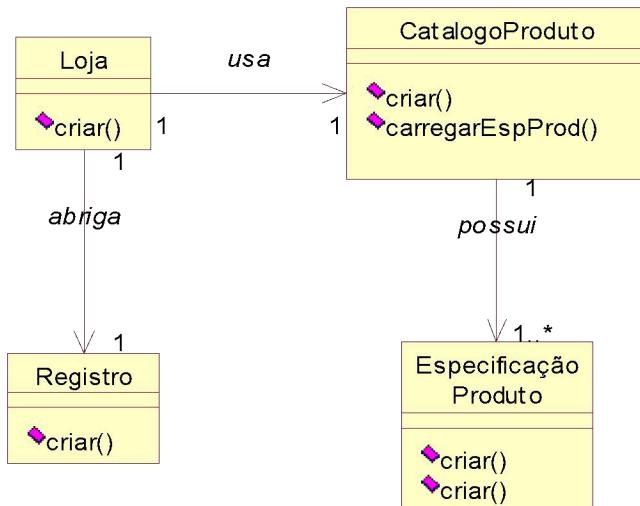
- Explorar diagramas de interação
- Situações comuns que indicam uma necessidade de definir uma associação com um adorno de navegabilidade de A para B
  - A envia uma mensagem para B
  - A cria uma instância B
  - A precisa manter uma conexão com B



# Navegabilidade: como identificar o sentido



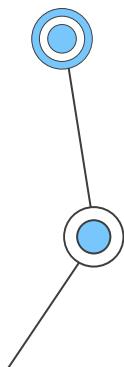
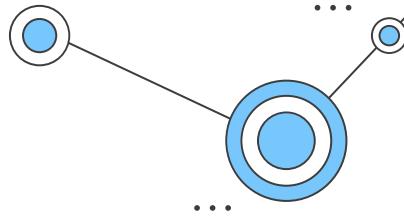
- Observando o diagrama é discernível que Loja tenha uma associação permanente com instâncias de registro e de CatalogoProduto que criou e que CatalogoProduto também tenha uma conexão permanente com EspecificaçãoProduto



# Projeto de Classes

## Relacionamentos de dependência

- O relacionamento de dependência indica que uma classe depende (ou tem conhecimento) dos serviços fornecidos por uma outra classe.
- São relacionamentos de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe.
  - características de implementação.



# Projeto de Classes

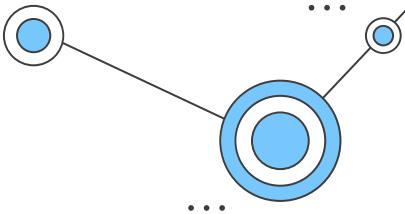
## Relacionamentos de dependência

- Representado através de uma linha direcionada e tracejada ligando as classes envolvidas.
- A direção é da classe dependente (cliente) para a classe da qual ela depende (fornecedor).
- Estereótipos: global, local, parameter.



- Um relacionamento de dependência indica que:
  - Operações da classe cliente criam objetos da classe fornecedor;
  - Operações da classe cliente possuem métodos cuja classe de retorno ou seus argumentos são instâncias da (ou se referenciam à) classe fornecedor.

# Projeto de Classes



Relacionamentos de dependência

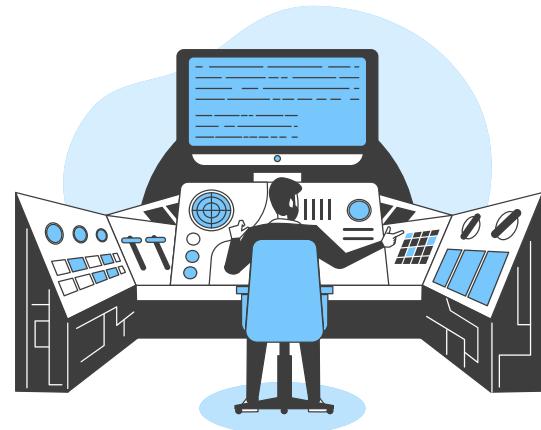
Propósito da análise das dependências

- Determinar se relacionamentos estruturais são necessários ou não



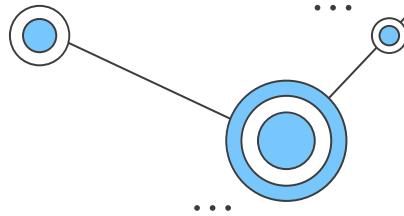
Considerar:

- O que causa o fornecedor estar visível para o cliente
  - ✓ Considerar a Visibilidade de objetos

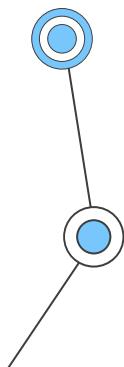


# Projeto de Classes

## Visibilidade entre Objetos



- Para que dois objetos possam trocar mensagens para realizar responsabilidades, é necessário que exista visibilidade entre eles.
  - Para um objeto A enviar uma mensagem para um objeto B, B deve ser visível para A.
- Capacidade de um objeto “ver” ou ter uma referência para outro objeto.
  - Necessária para comunicação (envio de mensagens) entre objetos.
- Estabelece uma dependência que indica que uma classe depende (ou tem conhecimento) dos serviços fornecidos por uma outra classe.
- Essa dependência pode ser expressa por mais de um tipo de relacionamento.

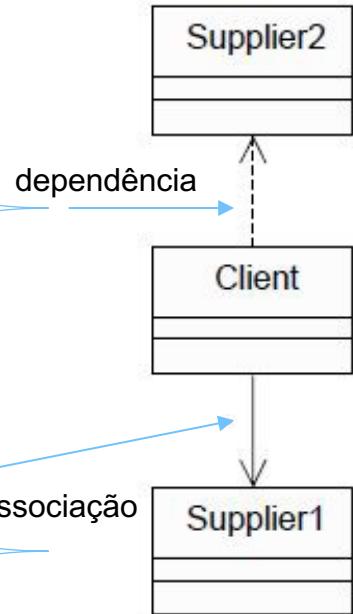
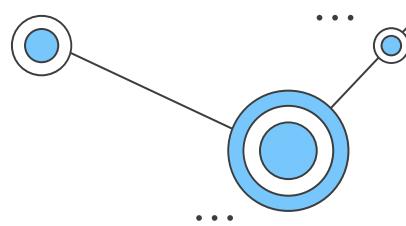


# Projeto de Classes

## Visibilidade entre Objetos

Algumas maneiras de B ser visível para A:

- Visibilidade de parâmetro – B é um parâmetro de um método de A (quando um objeto recebe outro como parâmetro em um de seus métodos)
- Visibilidade declarada localmente – B é declarado como objeto local de um método de A (quando um objeto recebe outro como retorno de um método)
- Visibilidade global – B é de algum modo visível globalmente
- Visibilidade de Classe – chamadas de métodos estáticos ou de classe
- Visibilidade de atributo (associação) – B é um atributo de A (as classes de dois objetos estão associadas)

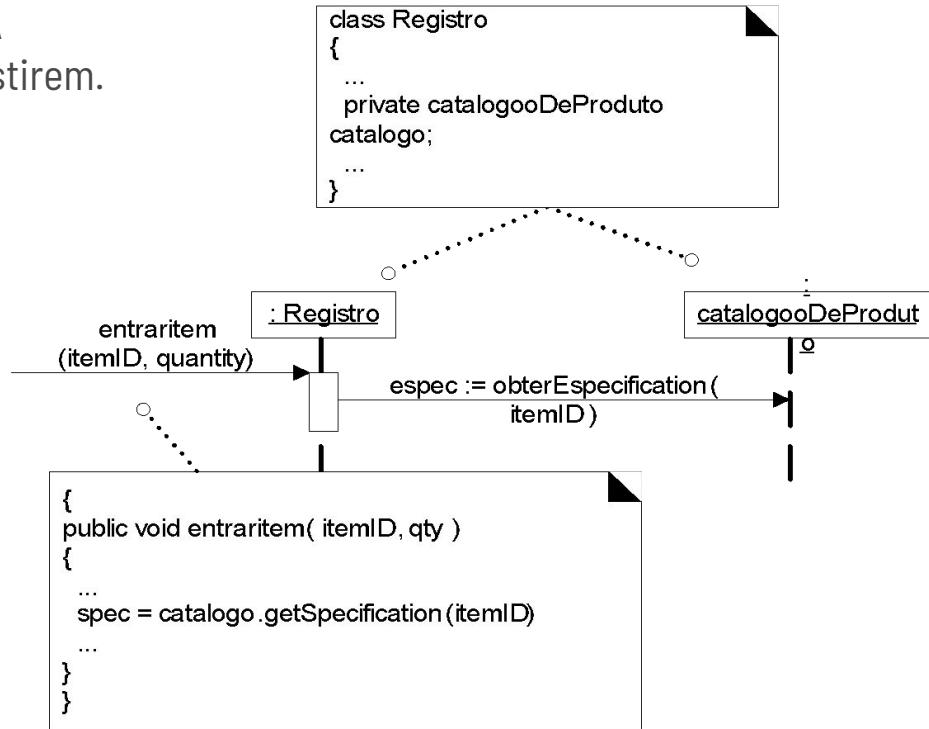
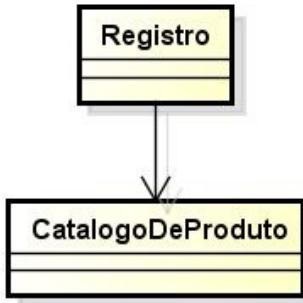


# Projeto de Classes

## Visibilidade de atributo (associação)

Existe de A para B quando B é um atributo de A

- Permanente – persiste enquanto A e B existirem.

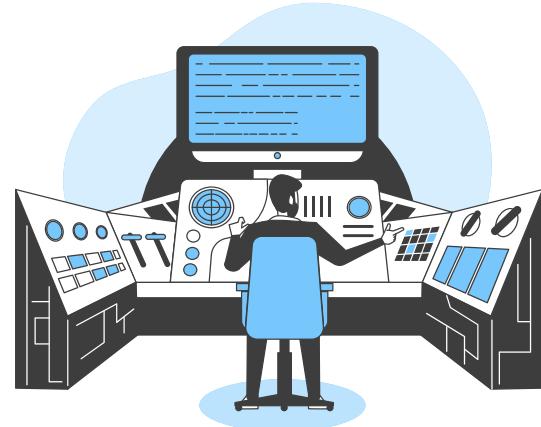
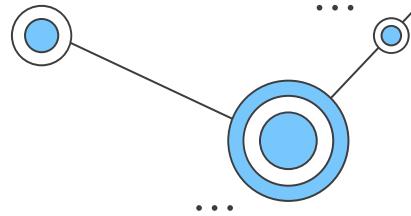


# Projeto de Classes

## Visibilidade por parâmetro

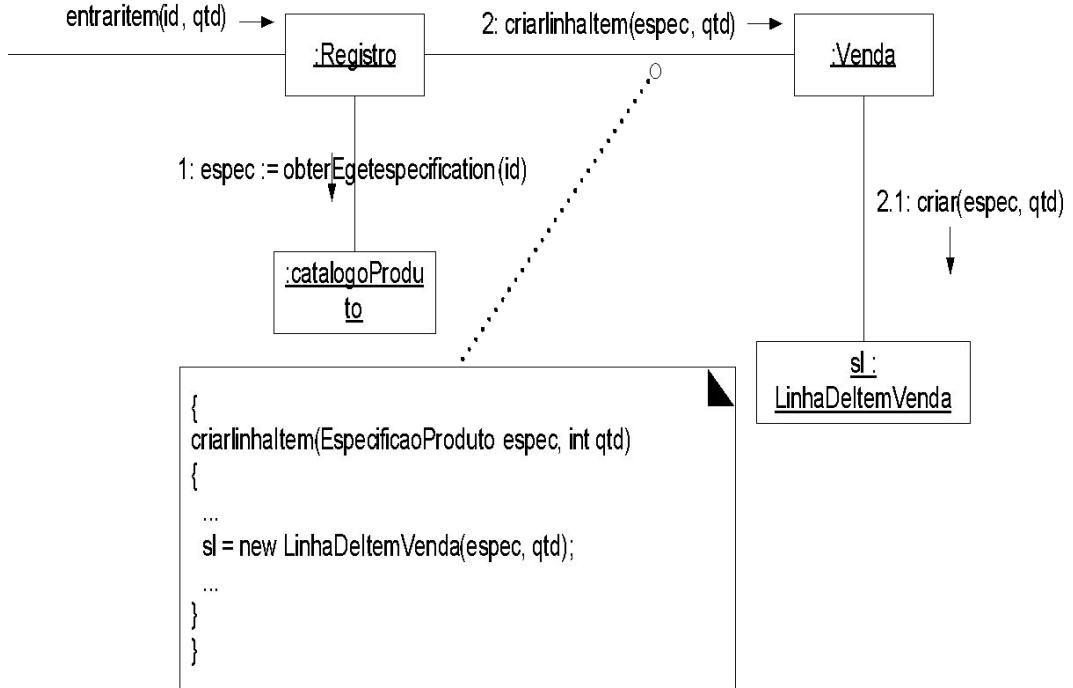
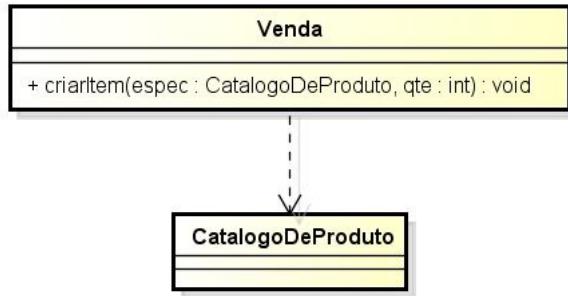
A visibilidade por parâmetro é obtida quando um objeto, ao executar um método, recebe outro objeto como parâmetro.

- Existe de A para B quando B é passado como um parâmetro para um método de A.
  - ✓ Temporária - persiste apenas dentro do escopo do método de A (permanente se B é atribuído a um atributo de A).



# Projeto de Classes

## Visibilidade por parâmetro

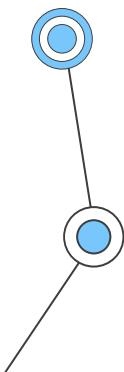
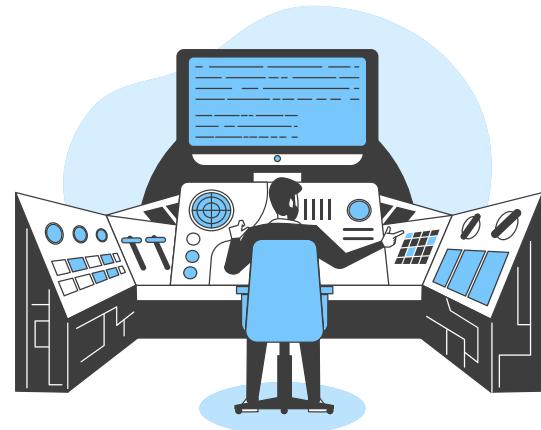
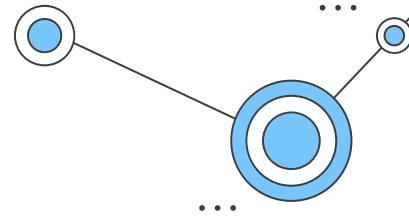


# Projeto de Classes

## Visibilidade Local

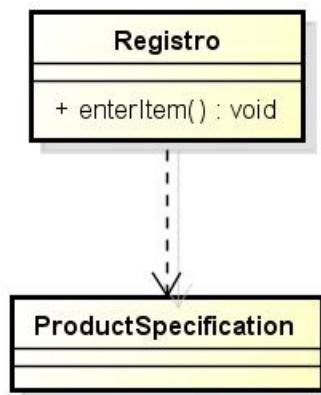
Existe de A para B quando B é declarado como um objeto local dentro de um método de A.

- Temporária – persiste apenas dentro do escopo do método de A (permanente se B é atribuído a um atributo de A).
- Duas maneiras comuns de alcançar:
  - 1. Criar nova instância e atribuir para variável local.
  - 2. Atribuir objeto de retorno de um método para variável local.



# Projeto de Classes

## Visibilidade Local

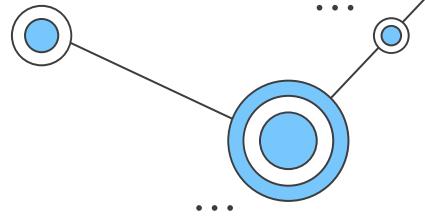


```
{  
    enterItem(id, qty)  
{  
    ...  
    // local visibility via assignment of returning object  
    ProductSpecification spec = catalog.getSpecification(id);  
    ...  
}
```



Dentro do método `enterItem()` existe uma definição de `ProductSpecification`

# Projeto de Classes



## Visibilidade Global

Existe de A para B quando B é global para A.

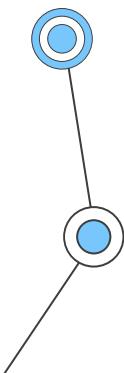
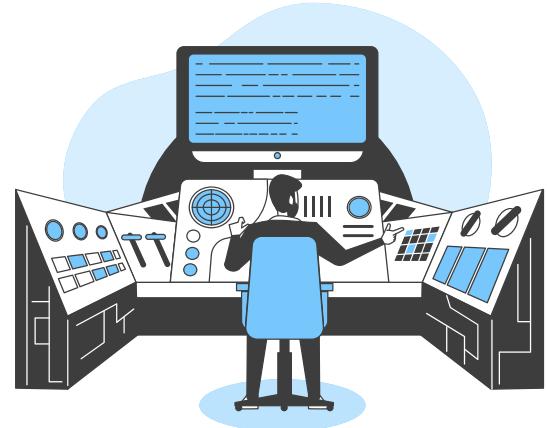
- Permanente – persiste enquanto A e B existirem.

Forma menos comum de visibilidade em sistemas desenvolvidos utilizando OO.

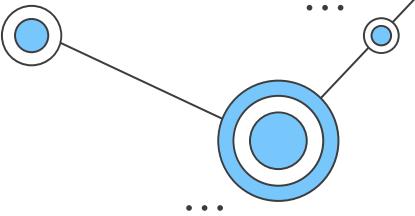
- Maneira mais comum (mas não recomendada) de atingir é atribuir nova instância a uma variável global.

Alternativa recomenda:

- Padrão Singleton (GoF).



# Projeto de Classes



## Visibilidade de Classes

Existe de A para B quando A executa um método estático de outra classe.

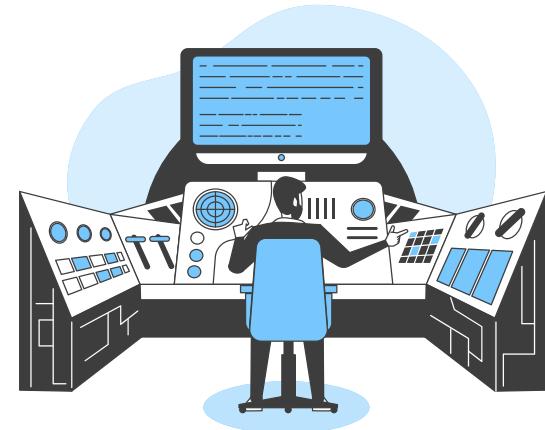
- Normalmente temporária



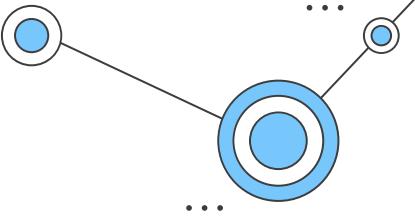
O método `obterQteCliente()` de `Loja` chama um método estático da classe `Cliente` (`qteCleinte()`). Assim, o objeto de `Loja` tem uma dependência por método estático com a classe `Cliente`.



```
Public class Loja {
    Public obterQteCliente() {
        Cliente.qteClientes();
    }
}
```



# Projeto de Classes



## Quiz

Qual tipo de visibilidade existe entre Cliente e loja? O relacionamento de associação ou dependência?

- Visibilidade de Classe. O objeto de Loja tem uma dependência por método estático com a classe Cliente.

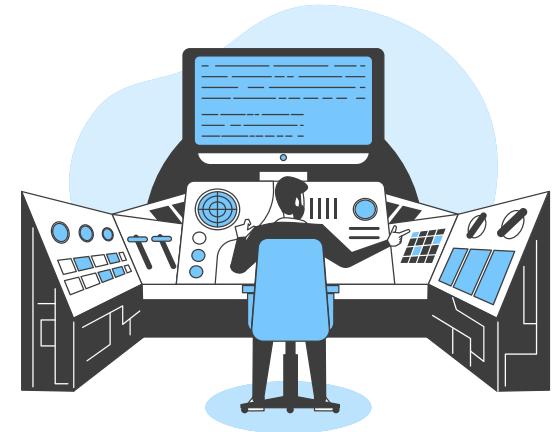
Método  
estático



O método `obterQteCliente()` de Loja chama um método estático da classe Cliente (`qteCleinte()`).



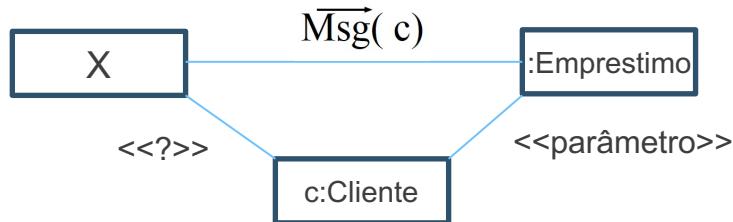
```
Public class Loja {  
    Public obterQteCliente() {  
        Cliente.qteClientes();  
    }  
}
```



# Projeto de Classes

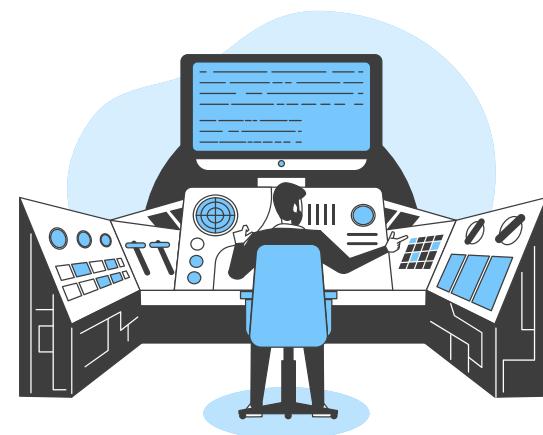
## Quiz

Que tipo de visibilidade uma instância de Empréstimo possui para a instância c?



Visibilidade por Parâmetro. A instância de Empréstimo apenas adquire visibilidade para a instância c de cliente durante a execução da mensagem Msg.

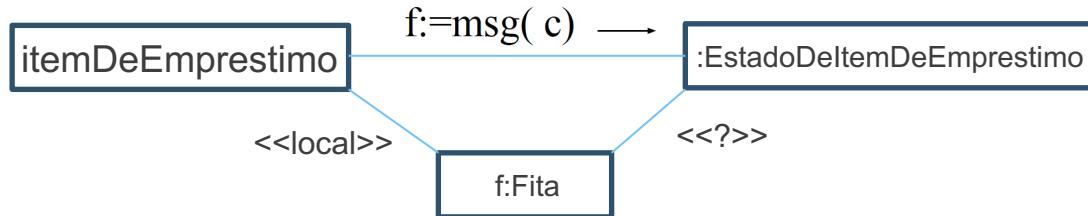
Para que o objeto X da esquerda pudesse passar C como parâmetro ele deveria ter algum tipo de visibilidade indicado por <<?>>



# Projeto de Classes

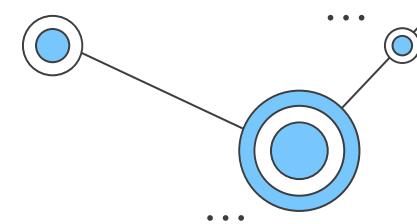
## Quiz

Que tipo de visibilidade uma instância de item de Empréstimo possui para a instância F de Fita?



Visibilidade local: A instância de ItemDeEmprestimo adquire visibilidade local para a instância f de Fita após enviar a mensagem MSG para a instância EstadoDelItemDeEmprestimo.

Para a instância EstadoDelItemDeEmprestimo poder retornar efetivamente f deve-se assumir que ela possua visibilidade por atributo ou local declarada para f indicado por <<?>>



Duas maneiras comuns de alcançar:

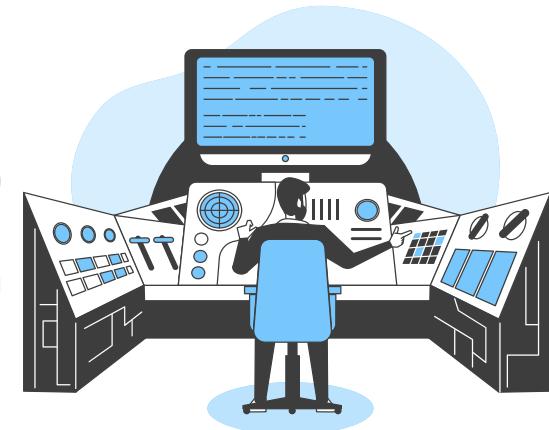
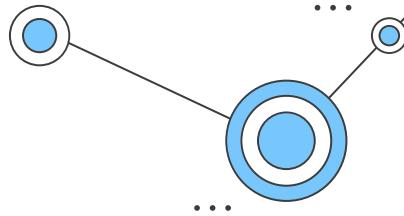
1. Criar nova instância e atribuir para variável local
2. Atribuir objeto de retorno de um método para variável local

# Projeto de Classes

## Quiz

Assinale a afirmação correta:

- A. Ao ser definida a navegabilidade em associações entre classes estamos, também, definindo, entre essas classes, uma dependência do tipo local.
- B. Se uma classe A tem uma associação de 0..1 com B e navegabilidade de A para B. Isso implica que no código da classe B deve haver uma variável de instância do tipo A;
- C. A navegabilidade pode ser 1, 1..\*, \* ou 0..\*
- D. Associação é o meio empregado para registrar o relacionamento entre instâncias de classes.
- E. Visibilidade por atributo pode ser representada por um relacionamento de dependência

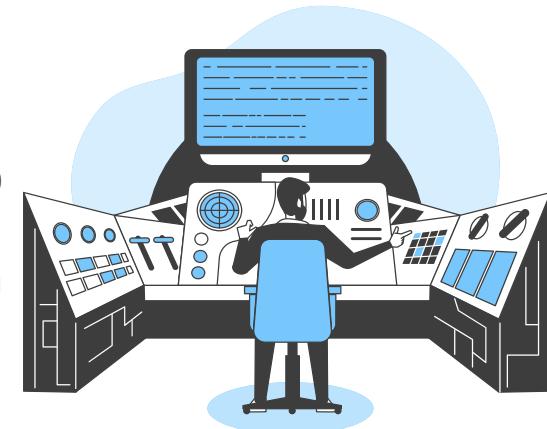
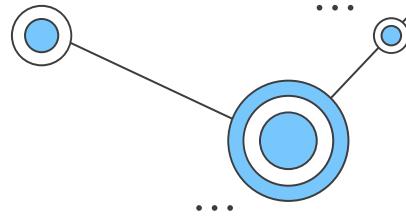


# Projeto de Classes

## Quiz

Assinale a afirmação correta:

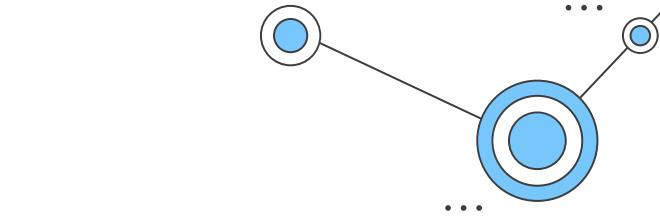
- A. Ao ser definida a navegabilidade em associações entre classes estamos, também, definindo, entre essas classes, uma dependência do tipo local.
- B. Se uma classe A tem uma associação de 0..1 com B e navegabilidade de A para B. Isso implica que no código da classe B deve haver uma variável de instância do tipo A;
- C. A navegabilidade pode ser 1, 1..\*, \* ou 0..\*
- D. Associação é o meio empregado para registrar o relacionamento entre instâncias de classes.**
- E. Visibilidade por atributo pode ser representada por um relacionamento de dependência



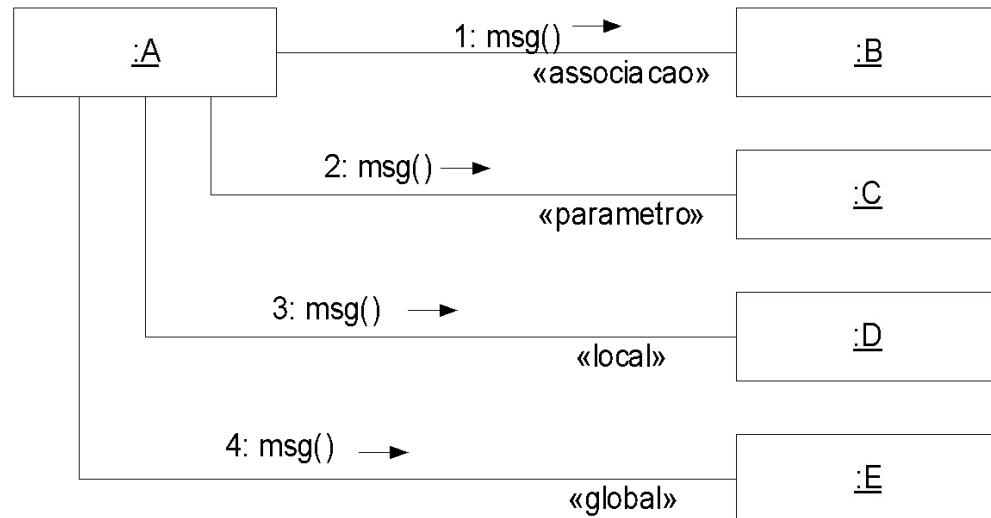
# Projeto de Classes

## Notação de Visibilidade na UML

Uso opcional de “estereótipos” específicos



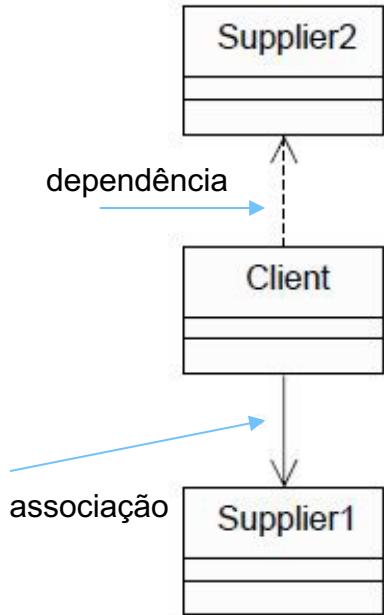
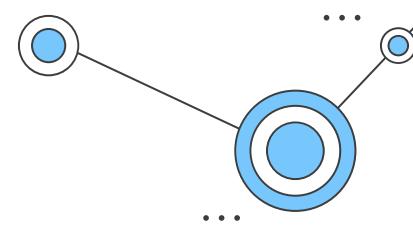
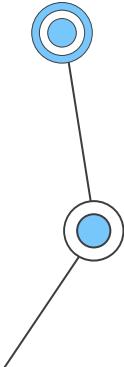
«associação» é usado para  
Visibilidade por atributo



# Projeto de Classes

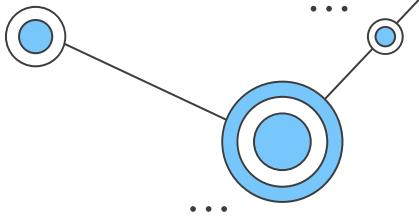
## Associação x Dependência

- Associações são relacionamentos estruturais.
- As dependências são relacionamentos não-estruturais.
- Dependência é um tipo de via de comunicação que é um tipo de relacionamento transitivo. Estes ocorrem quando a visibilidade é parâmetro, global, ou local.



# Projeto de Classes

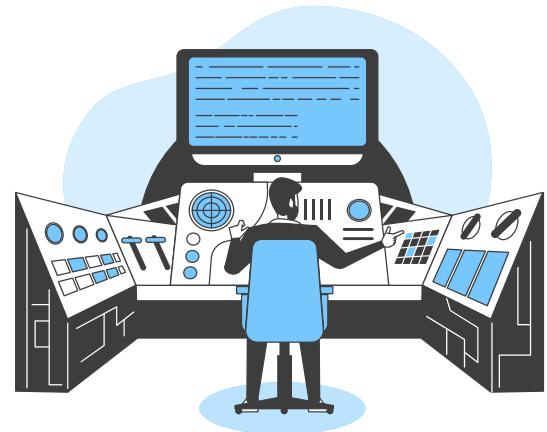
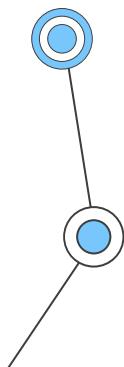
## Transformação de associações em dependências



Na passagem para o modelo de projeto, cada associação deve ser analisada para identificar se ela pode ser transformada em relacionamento de dependência.

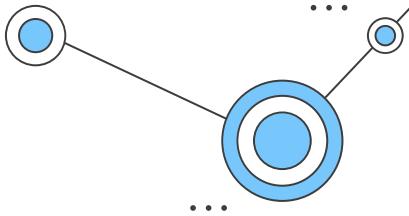
Razão: aumentar o encapsulamento das classes.

- Dependência por atributo torna as classes envolvidas mais dependentes uma da outra.
- Quanto menos dependências estruturais houver no modelo de classes, maior é o encapsulamento das classes constituintes.

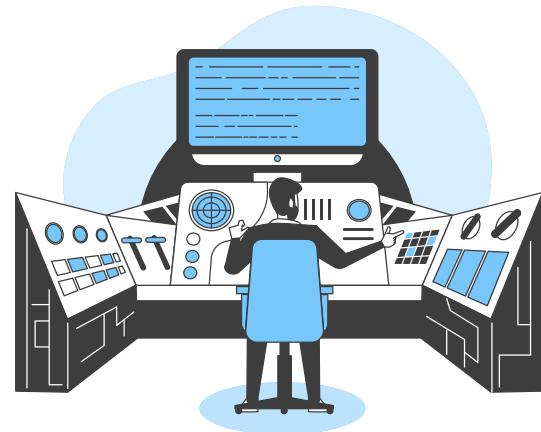
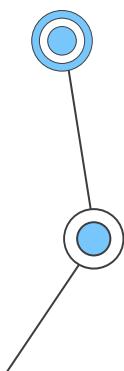


# Projeto de Classes

Transformação de associações em dependências



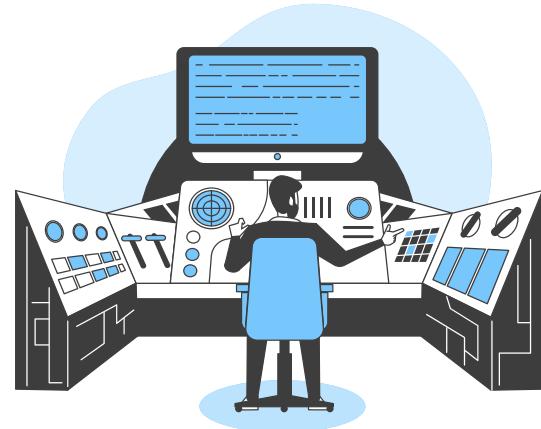
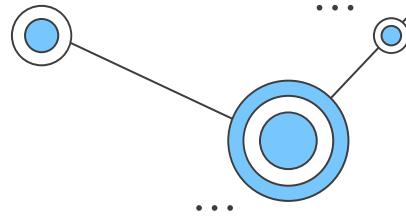
- Associações que ligam classes de entidade permanecem como associações.
- Associações entre controladores e classes de entidade: bastante suscetíveis à transformação.
- Associações entre classes de fronteira e controladores:
  - classes de fronteira para equipamentos ou outros sistemas: interfaces, dependência não estrutural.



# Projeto de Classes

## Interfaces

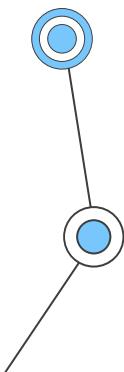
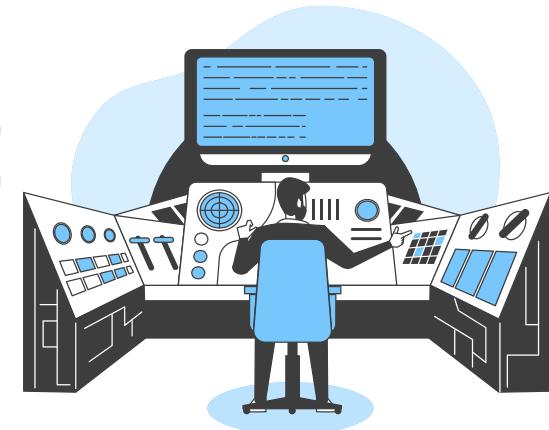
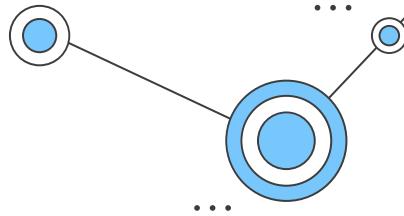
- LP 00 oferece uma capacidade que exige que classes não relacionadas implementem um conjunto de métodos comuns?
- Em outras palavras:
  - Há como forçar que a implementação de uma ação comum a classes de objetos diferentes sejam implementadas nessas classes?
  - Ex: mover para Anfíbio, Pássaro e Peixe.



# Projeto de Classes

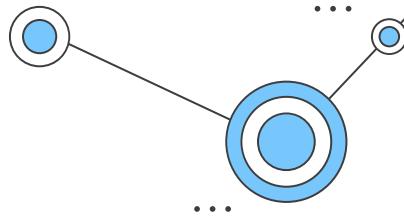
## Interfaces

- 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.
  - Contanto que um objeto fornecedor forneça implementação para a interface que o objeto cliente espera, este último não precisa conhecer a verdadeira classe do primeiro.



# Projeto de Classes

## Interfaces

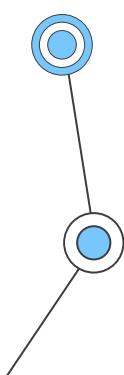


Interfaces são utilizadas com os seguintes objetivos:

- 1. Capturar semelhanças entre classes não relacionadas sem forçar relacionamentos entre elas.
- 2. Declarar operações que uma ou mais classes devem implementar.
- 3. Revelar as operações de um objeto, sem revelar a sua classe.
- 4. Facilitar o desacoplamento entre elementos de um sistema.

Nas LPOO modernas (Java, C#, etc.), interfaces são definidas de forma semelhante a classes.

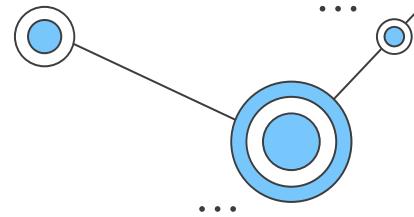
- Uma diferença é que todas as declarações em uma interface têm visibilidade pública.
- Adicionalmente, uma interface não possui atributos, somente declarações de assinaturas de operações e (raramente) constantes.



# Projeto de Classes

## Interfaces

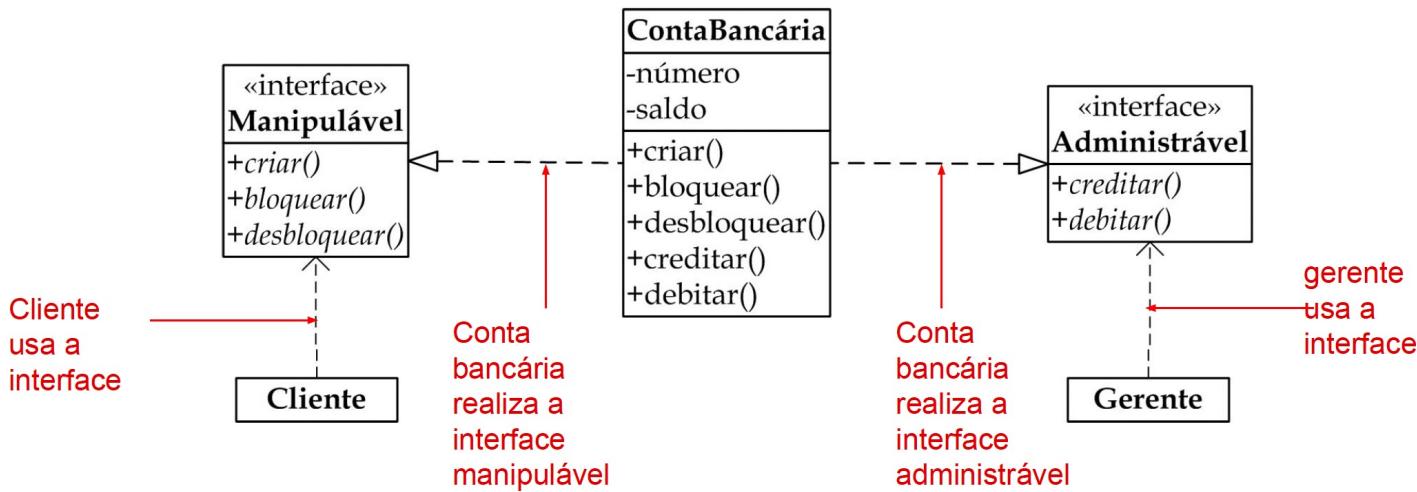
- Uma interface define um conjunto de métodos que uma classe deve implementar mas não define como esses métodos devem ser implementados.
- Define, na verdade, uma lista de métodos que toda a classe que segue determinada interface deve prover.
- Em JAVA, classes só podem ter um ancestral direto, ou seja, só podem ser derivadas de uma única classe (embora esta possa ser derivada de outra e assim por diante).
  - Existem situações, porém, onde pode ser necessário que uma classe herde características de mais de uma “superclasse” simultaneamente.
  - Como a herança múltipla não é permitida em JAVA, a linguagem oferece o conceito de interface como opção.



# Projeto de Classes

## Interfaces - Notação

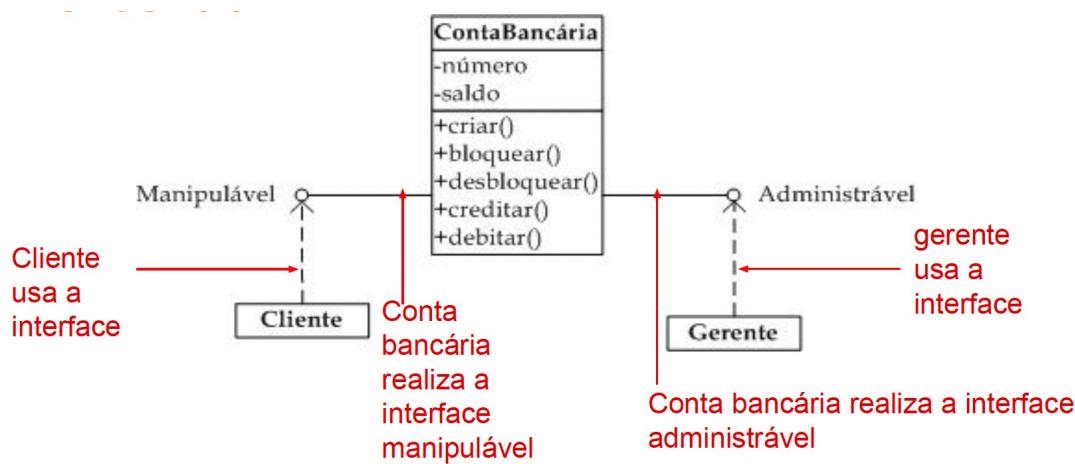
- A primeira notação: é a mesma para classes.
  - São exibidas as operações que a interface especifica.
  - Deve ser usado o estereótipo <<interface>>.



# Projeto de Classes

## Interfaces - Notação

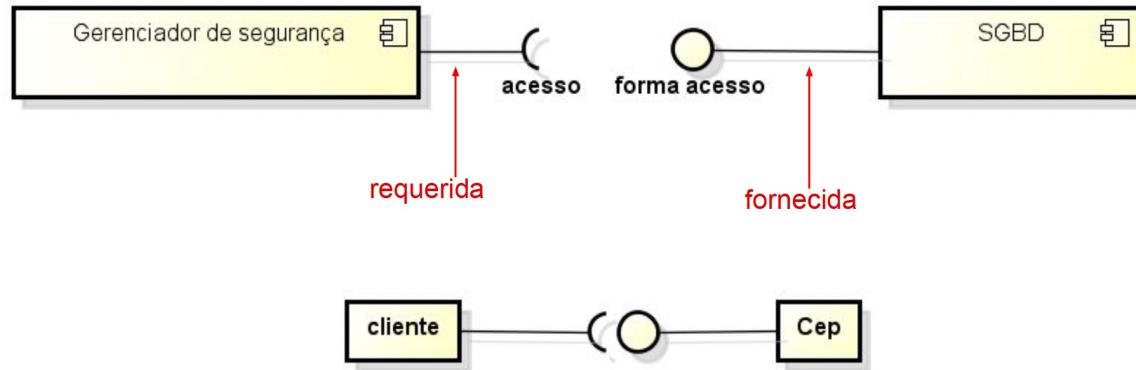
- Segunda notação: através de um segmento de reta com um pequeno círculo em um dos extremos e ligado ao classificador que a realiza no outro extremo.
  - ✓ Classes clientes conectadas à interface através de um relacionamento de dependência.



# Projeto de Classes

## Interfaces - Tipos

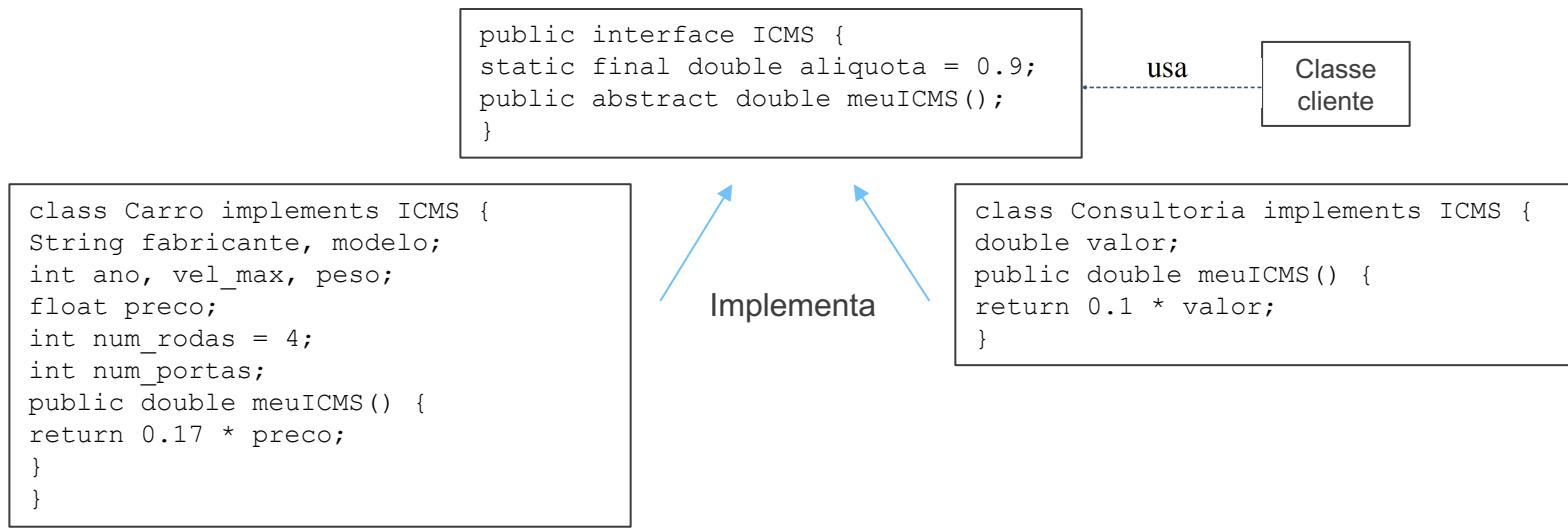
- Interface fornecida
  - Uma interface fornecida por um componente, significando uma interface em que o componente fornece um serviço para outros componentes
- Interface requerida
  - Uma interface utilizada pelo componente, significando uma interface à qual o componente se adapta quando solicita serviços de outros componentes



# Projeto de Classes

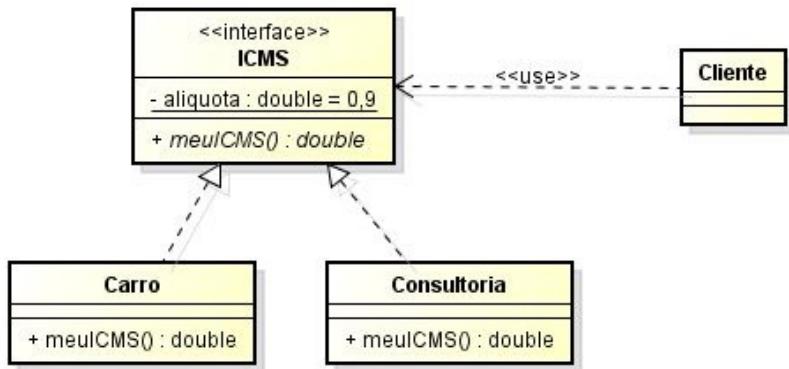
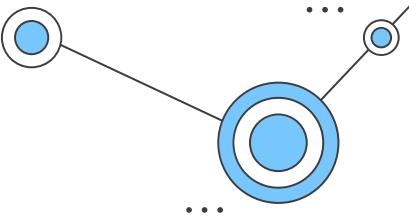
## Interfaces

Exemplo: ICMS pode ser diferenciado dependendo do tipo de produto. Para evitar que um programa chame cada classe específica para calcular o ICMS, pode-se criar uma interface que vai ser acionada pelo programa. A implementação dela está nas classes específicas de cada produto. ;

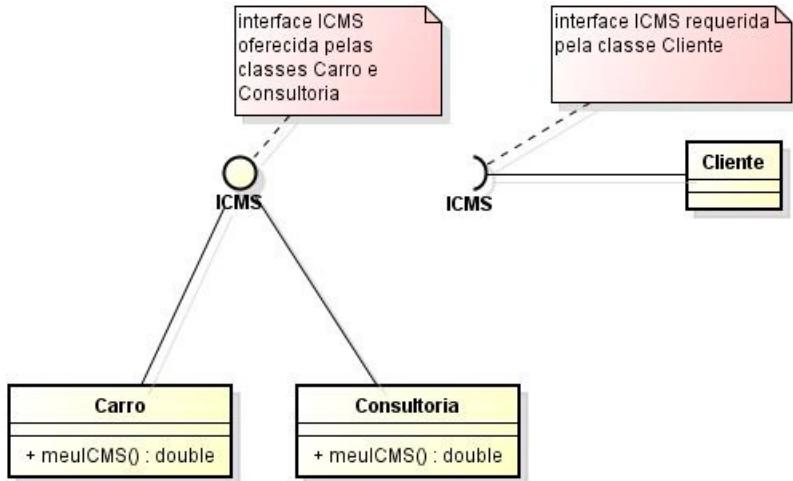


# Projeto de Classes

## Notações



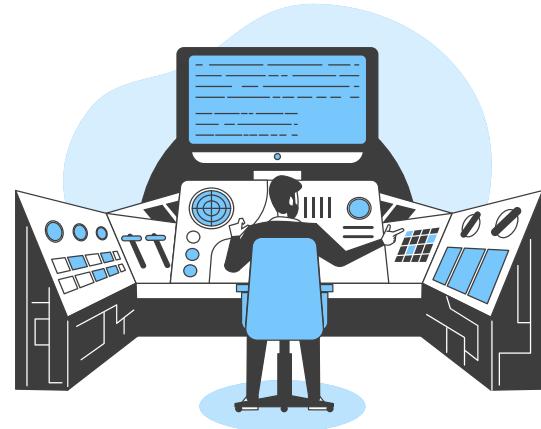
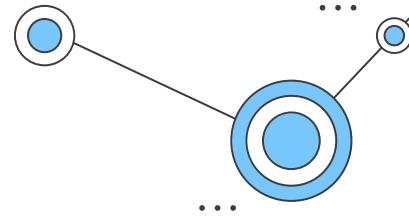
OU



# Projeto de Classes

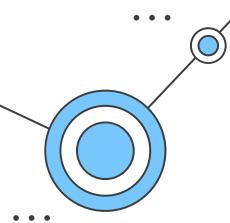
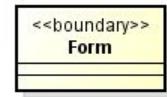
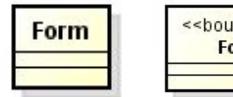
## Interfaces

- Uma interface é como um conjunto de regras (contrato) definidas e que devem ser rigorosamente seguidas (implementadas).
  - Interface é quando você apenas quer definir um "contrato" que as classes devem realizar (implementar)
  - uma interface é uma maneira de descrever o que a classe vai fazer, ao invés de como ela faria



# Projeto de Classes

Especificação de classes de fronteira



Não devemos atribuir a essas classes responsabilidades relativas à lógica do negócio.

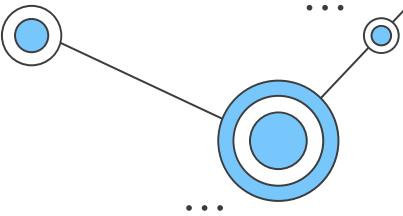
- Classes de fronteira devem apenas servir como um ponto de captação de informações, ou de apresentação de informações que o sistema processou.
- A única inteligência que essas classes devem ter é a que permite a elas realizarem a comunicação com o ambiente do sistema.

Há diversas razões para isso:

- Em primeiro lugar, se o sistema tiver que ser implantado em outro ambiente, as modificações resultantes sobre seu funcionamento propriamente dito seriam mínimas.
- Além disso, o sistema pode dar suporte a diversas formas de interação com seu ambiente (e.g., uma interface gráfica e uma interface de texto).
- Finalmente, essa separação resulta em uma melhor coesão.

# Projeto de Classes

## Especificação de classes de fronteira



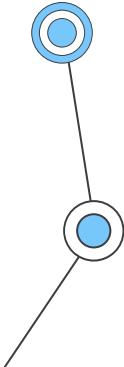
Durante a análise, considera-se que há uma única classe de fronteira para cada ator. No projeto, algumas dessas classes podem resultar em várias outras.

Interface com seres humanos: projeto da interface gráfica produz o detalhamento das classes.

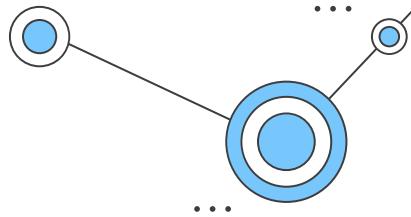
Outros sistemas ou equipamentos: devemos definir uma ou mais classes para encapsular o protocolo de comunicação.

- É usual a definição de um subsistema para representar a comunicação com outros sistemas de software ou com equipamentos.
- É comum nesse caso o uso do padrão Façade (mais adiante)

O projeto de objetos de fronteira é altamente dependente da natureza do ambiente...



# Projeto de Classes



## Especificação de classes de fronteira

### Cientes WEB clássicos

- Classes de fronteira são representadas por páginas HTML que, muitas vezes, representam sites dinâmicos.

### Cientes móveis

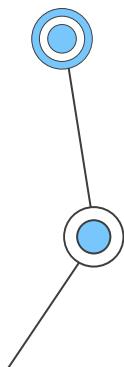
- Classes de fronteira implementam algum protocolo específico com o ambiente.
  - ✓ Um exemplo é a WML (Wireless Markup Language).

### Cientes stand-alone

- Nesse caso, é recomendável que os desenvolvedores pesquisem os recursos fornecidos pelo ambiente de programação sendo utilizado.
  - ✓ Um exemplo disso é o Swing/JFC da linguagem Java.

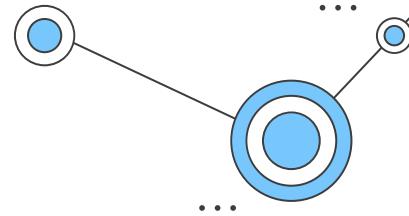
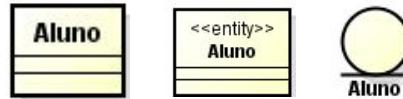
### Serviços WEB (WEB services)

- Um serviço WEB é uma forma de permitir que uma aplicação forneça seus serviços (funcionalidades) através da Internet.



# Projeto de Classes

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.

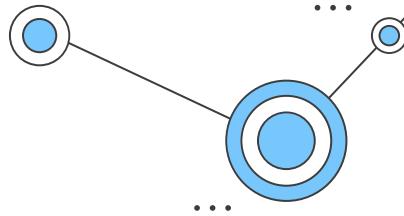
- Para essas classes, o seu mapeamento para algum mecanismo de armazenamento persistente deve ser definido.

Um aspecto importante é a forma de representar associações, agregações e composições entre objetos de entidade.

- Essa representação é função da naveabilidade e da multiplicidade definidas para a associação, conforme visto mais adiante.

# Projeto de Classes

## Especificação de classes de entidade



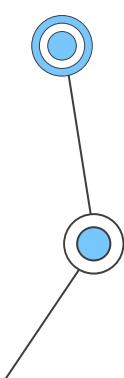
Outro aspecto relevante para classes de entidade é modo como podemos identificar cada um de seus objetos unicamente.

- 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 (um atributo do domínio).

A manipulação dos diversos atributos identificadores possíveis em uma classes pode ser bastante trabalhosa.

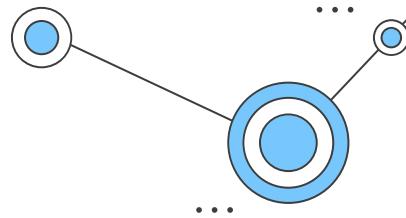
Para evitar isso, um identificador de implementação é criado, que não tem correspondente com atributo algum do domínio.

- Possibilidade de manipular identificadores de maneira uniforme e eficiente.
- Maior facilidade quando objetos devem ser mapeados para um SGBDR.



# Projeto de Classes

Especificação de classes de controle

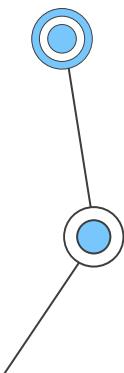


Com relação às classes de controle, no projeto devemos identificar a real utilidade das mesmas.

- Em casos de uso simples (e.g., manutenção de dados), classes de controle não são realmente necessárias. Neste caso, classes de fronteira podem repassar os dados fornecidos pelos atores diretamente para as classes de entidade correspondentes.

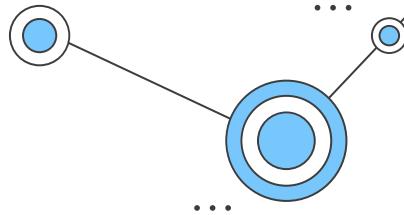
Entretanto, é comum a situação em que uma classe de controle de análise ser transformada em duas ou mais classes no nível de especificação.

No refinamento de qualquer classe proveniente da análise, é possível a aplicação de padrões de projeto (design patterns).



# Projeto de Classes

## Especificação de classes de controle

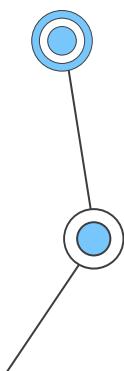


Normalmente, cada classe de controle deve ser particionada em duas ou mais outras classes para controlar diversos aspectos da solução.

- Objetivo: de evitar a criação de uma única classe com baixa coesão e alto acoplamento.

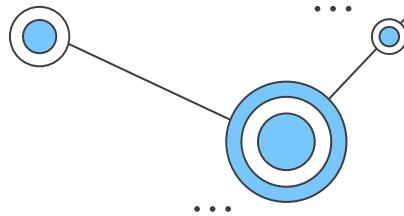
Alguns exemplos dos aspectos de uma aplicação cuja coordenação é de responsabilidade das classes de controle:

- produção de valores para preenchimento de controles da interface gráfica,
- autenticação de usuários,
- controle de acesso a funcionalidades do sistema, etc.



# Projeto de Classes

## Especificação de classes de controle



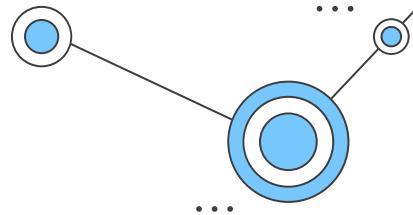
Um tipo comum de controlador é o controlador de caso de uso, responsável pela coordenação da realização de um caso de uso.

As seguintes responsabilidades são esperadas de um controlador de caso de uso:

- 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.



# Projeto de Classes



## Especificação de classes de controle

Em aplicações WEB, é comum a prática de utilizar outro tipo de objeto controlador chamado front controller (FC).

Um FC é um controlador responsável por receber todas as requisições de um cliente.

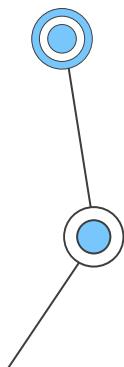
O FC identifica qual o controlador (de caso de uso) adequado para processar a requisição, e a despacha para ele.

Sendo assim, um FC é um ponto central de entrada para as funcionalidades do sistema.

- Vantagem: mais fácil controlar a autenticação dos usuários.

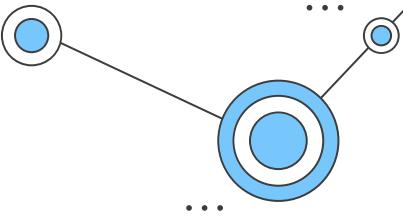
O FC é um dos padrões de projeto do catálogo J2EE

- <https://www.oracle.com/java/technologies/front-controller.html>



# Projeto de Classes

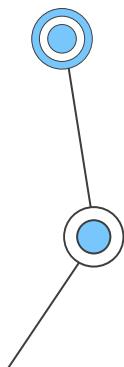
## Especificação de outras classes



Além do refinamento de classes preexistentes, diversas outros aspectos demanda a identificação de novas classe durante o projeto.

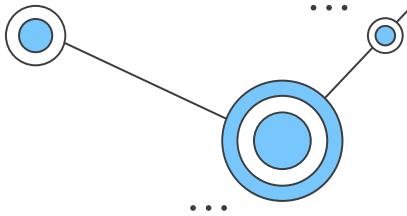
- Persistência de objetos
- Distribuição e comunicação (e.g., RMI, CORBA, DCOM, WEB)
- Autenticação/Autorização
- Logging
- Configurações
- Threads
- Classes para testes (Test Driven Development)
- Uso de bibliotecas, componentes e frameworks

Conclusão: a tarefa de identificação (reuso?) de classes não termina na análise.



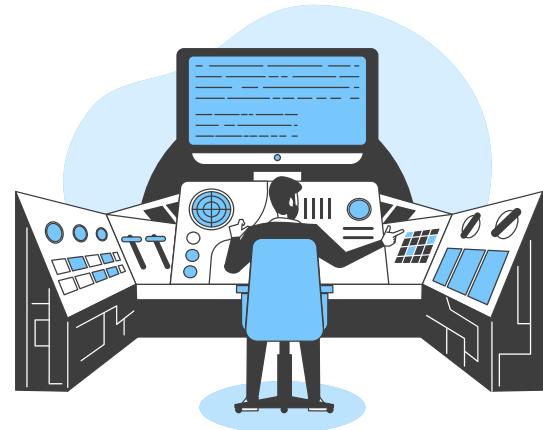
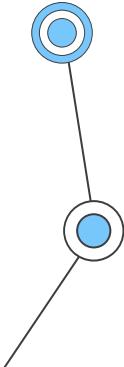
# Projeto de Classes

## Modelando as Classes



Após identificação de todas classes, uma alternativa para fazer os modelos é:

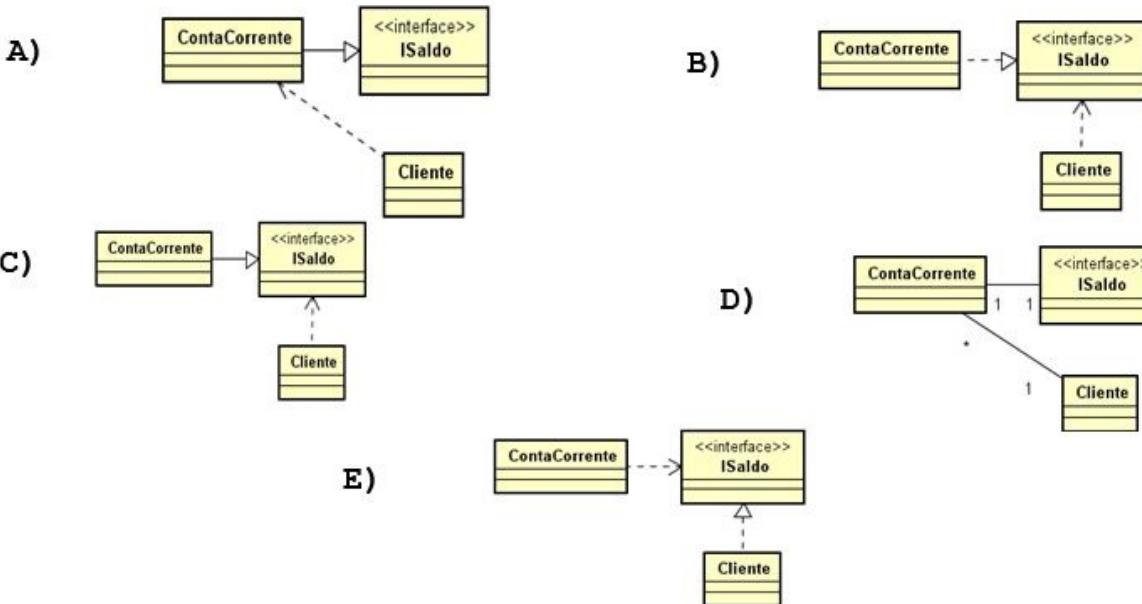
- Criar um diagrama para cada tipo de classes (fronteira, entidade e controle)
- Colocar esses diagramas em pacotes
- Criar relacionamentos entre os pacotes



# Projeto de Classes

## Quiz

Um arquiteto de software deseja declarar que a classe ContaCorrente implementa a interface ISaldo e que a classe Cliente usa essa interface. Qual dos diagramas UML abaixo representa corretamente essa declaração?

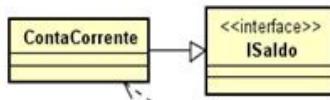


# Projeto de Classes

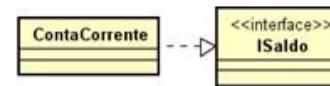
## Quiz

Um arquiteto de software deseja declarar que a classe ContaCorrente implementa a interface ISaldo e que a classe Cliente usa essa interface. Qual dos diagramas UML abaixo representa corretamente essa declaração?

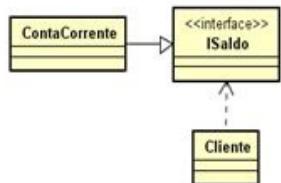
A)



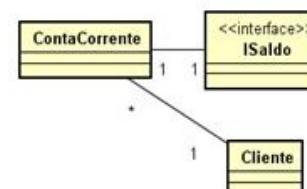
B)



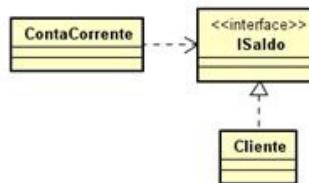
C)



D)



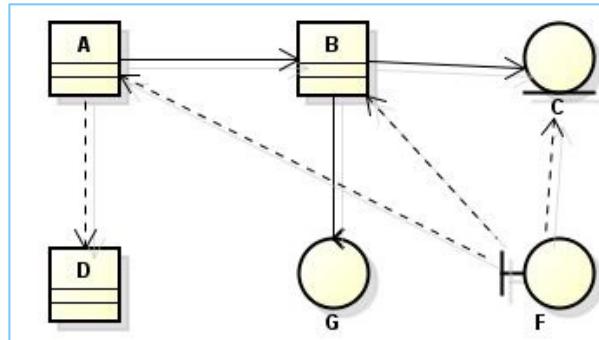
E)



# Projeto de Classes

## Quiz

Considere o modelo conceitual ao lado:



Pode-se afirmar que:

- a) A navegabilidade da associação entre A e B é unidirecional e deve haver um atributo de referencia de A em B
- b) Como C é uma classe do tipo fronteira (form), não é aconselhável que uma classe de domínio (entidade) B tenha um relacionamento de associação com ela, principalmente se considerarmos o padrão MVC
- c) O relacionamento entre B e C Não está representado corretamente, pois ambas a classe é do tipo controle e em B não deveria ter um atributo de referência do tipo C
- d) A navegabilidade da associação entre B e C é unidirecional e deve haver um atributo de referencia de C em B
- e) O relacionamento entre A e D implica na criação de um atributo do tipo D em A

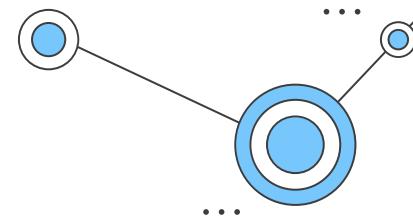
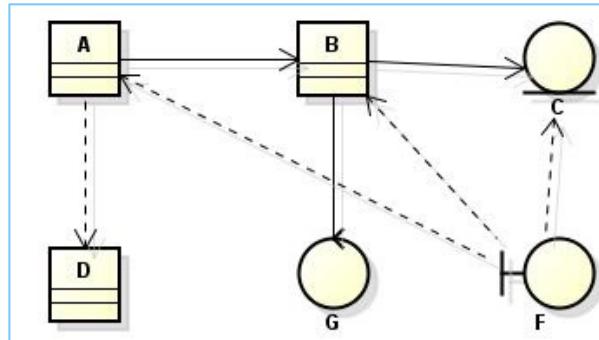
# Projeto de Classes

## Quiz

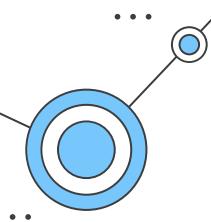
Considere o modelo conceitual ao lado

Pode-se afirmar que:

- a) A navegabilidade da associação entre A e B é unidirecional e deve haver um atributo de referencia de A em B
- b) Como C é uma classe do tipo fronteira (form), não é aconselhável que uma classe de domínio (entidade) B tenha um relacionamento de associação com ela, principalmente se considerarmos o padrão MVC
- c) O relacionamento entre B e C Não está representado corretamente, pois ambas a classe é do tipo controle e em B não deveria ter um atributo de referência do tipo C
- d) A navegabilidade da associação entre B e C é unidirecional e deve haver um atributo de referencia de C em B**
- e) O relacionamento entre A e D implica na criação de um atributo do tipo D em A



# Projeto de Classes



Hackles



<http://hackles.org>

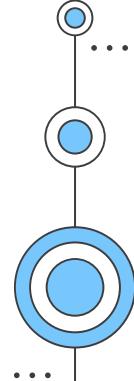
Copyright © 2003 Drake Emko & Jen Brodzik



# Projeto de Software

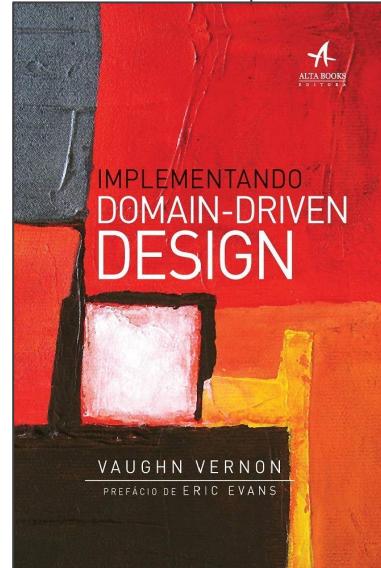
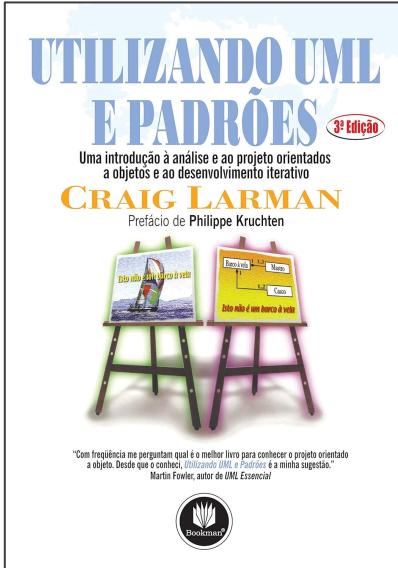
## Referências básicas:

- **ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY.** New York, N.Y., USA: Association for Computing Machinery, 1992-. Trimestral. ISSN 1049-331X. Disponível em: <https://dl.acm.org/toc/tosem/1992/1/2>. Acesso em: 19 jul. 2024. (Periódico On-line).
- LARMAN, Craig. **Utilizando UML e padrões:** uma introdução á análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007. E-book. ISBN 9788577800476. (Livro Eletrônico).
- SILVEIRA, Paulo et al. **Introdução à arquitetura e design de software:** uma visão sobre a plataforma Java. Rio de Janeiro, RJ: Elsevier, Campus, 2012. xvi, 257 p. ISBN 9788535250299. (Disponível no Acervo).
- VERNON, Vaughn. **Implementando o Domain-Driven Design.** Rio de Janeiro, RJ: Alta Books, 2016. 628 p. ISBN 9788576089520. (Disponível no Acervo).



# Projeto de Software

## Referências básicas:



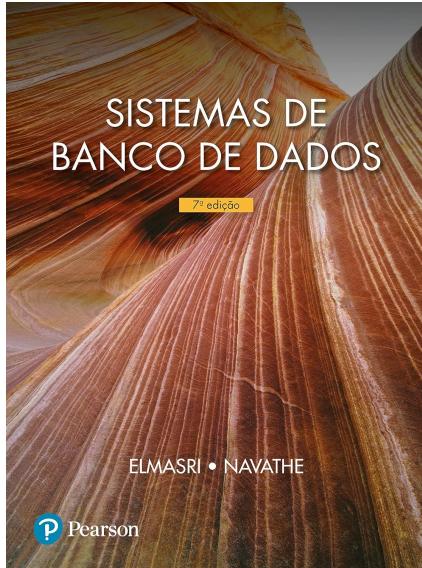
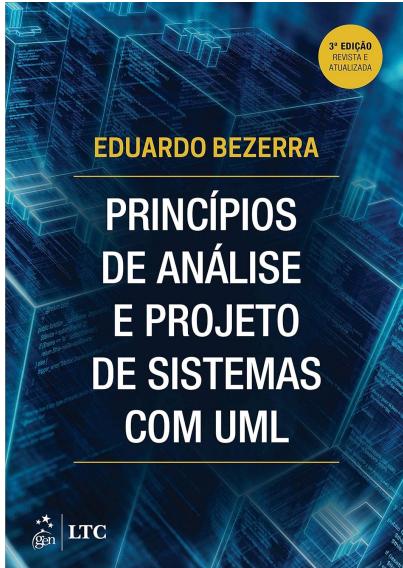
# Projeto de Software

## Referências complementares:

- BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 3. ed. rev. e atual. Rio de Janeiro: Elsevier, 2015. xvii, 398 p. ISBN 9788535226263. (Disponível no Acervo).
- ELMASRI, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados**, 7<sup>a</sup> ed. Editora Pearson 1152 ISBN 9788543025001. (Livro Eletrônico).
- GUEDES, Gilleanes T. A. **UML 2**: uma abordagem prática. 2. ed. São Paulo: Novatec, c2011. 484 p. ISBN 9788575222812. (Disponível no Acervo).
- **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING**. New York: IEEE Computer Society, 1975-. Mensal,. ISSN 0098-5589. Disponível em: <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=32>. Acesso em: 19 jul. 2024. (Periódico On-line).
- SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, c2019. xii, 756 p. ISBN 9788543024974. (Disponível no Acervo).
- WAZLAWICK, Raul Sidnei. **Análise e design orientados a objetos para sistemas de informação**: modelagem com UML, OCL e IFML. 3. ed. Rio de Janeiro, RJ: Elsevier, Campus, c2015. 462 p. ISBN 9788535279849. (Disponível no Acervo).

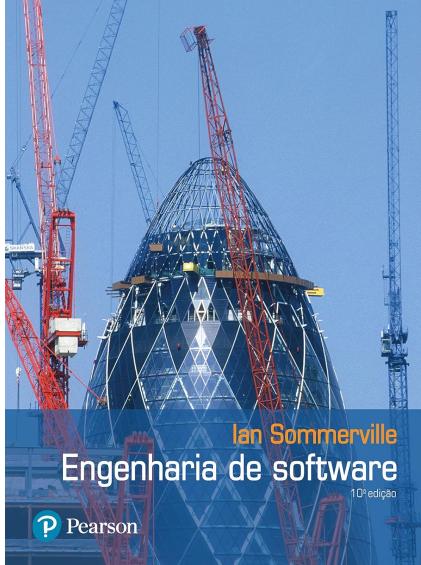
# Projeto de Software

## Referências complementares:



# Projeto de Software

## Referências complementares:



# Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com



[GitHub](#)



[LinkedIn](#)



[Lattes](#)

...



...