


# RELACIONAMENTOS ENTRE CLASSES

## Conteúdo

Introdução.....	2
Tipos de Relacionamentos .....	3
O Relacionamento de Dependência ( “ precisa de ” ).....	5
O Relacionamento de Generalização ( “ é um tipo de ” ).....	6
O Relacionamento de Associação.....	9
Multiplicidade. ....	12
Quadro 4 a) – Código C++ e Java do Exemplo da Figura 8 a) .....	13
Quadro 4 b) – Código C++ e Java do Exemplo da Figura 8 b).....	14
Relações Unidirecionais de Associação.....	14
Quadro 5 – Código C++ e Java do Exemplo da Figura 9 .....	15
Composição ( “...pertence exclusivamente a ...” )  .....	16
Quadro 6 – Códigos C++ e Java do Exemplo da Figura 10.....	17
Propriedades Da Relação .....	17
Dicas .....	19
Resumo.....	20
Exercício:.....	21
Referência Bibliográfica Utilizada.....	22

# RELACIONAMENTOS ENTRE CLASSES

## Introdução

Classes são representações de elementos, conceitos, idéias, grupo de objetos ou qualquer outra entidade conhecida. Para o engenheiro, Classes são peças de um quebra-cabeça, ou engrenagens. Essas peças devem ser “juntadas” ou “associadas” de tal forma a solucionar uma questão ou problema.

Outra forma de visualizar Classes é imaginar que elas definem um vocabulário básico. Esses elementos do vocabulário, por sua vez, necessitam ser estruturados ou relacionados de tal forma a representar uma ou mais idéias.

Podendo as Classes serem consideradas engrenagens ou peças de um quebra-cabeça, uma questão surge naturalmente: Como essas peças poderiam ser relacionadas?

Este material procura descrever, de maneira reduzida, como as classes podem ser relacionadas. O que é apresentado não é um conjunto de regras que busque delimitar a criatividade do engenheiro, mas sim, orientações ou recomendações.

Nesse contexto, analisam-se relacionamentos entre duas classes. Apesar dessa limitação, os conceitos apresentados neste material podem ser aplicados para definir relacionamentos entre três ou mais classes.

Este material utiliza a linguagem UML e códigos C++ e Java para representar os relacionamentos entre as classes. Os exemplo foram produzidos na ferramenta ArgoUML.

De maneira geral, espera-se desenvolver no aluno a habilidade de ser capaz de traduzir uma representação gráfica de um modelo para uma linguagem de programação.

## Tipos de Relacionamentos

Os relacionamentos que duas ou mais classes podem ter são:

- Dependência
- Generalização
- Associação

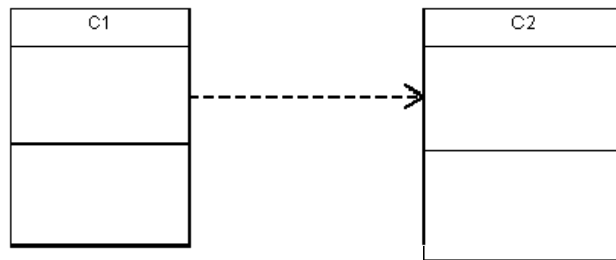
Se desejarmos desmontar um automóvel, por exemplo, verificaremos que ele é formado por peças que apresentam relações muito definidas. O automóvel possui: motor, rodas, banco, volante e diversas outras engrenagens. Esses elementos, associados ou conectados de maneira correta, permitem a existência e o correto funcionamento do automóvel.

São alguns exemplos de relacionamento entre as peças que compõem um automóvel:

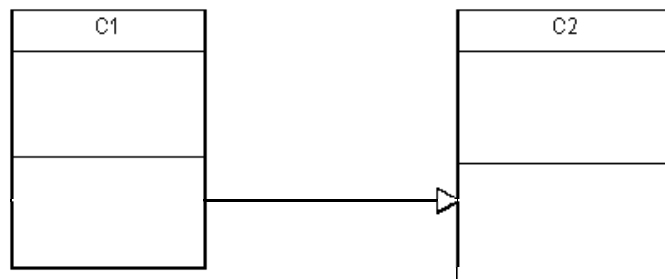
- **Dependência:** Motor depende de Bomba de Gasolina para bombear a gasolina para o seu interior.
- **Generalização:** Filtro de Ar, Filtro de Óleo e Filtro de Combustível são tipos específicos de Filtro.
- **Associação:** Pneus, Rodas, Amortecedores e Freios compõem a Suspensão do Automóvel.

A representação gráfica do tipo de associação entre duas classes, por exemplo, é feita com alguns elementos específicos. As Figuras 1, 2 e 3, respectivamente, representam graficamente a relação de Dependência, Generalização e Associação entre duas classes C1 e C2.

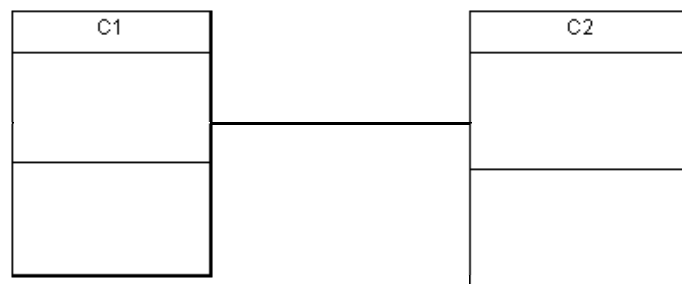
Na Figura 1, utiliza-se uma reta tracejada com terminal em seta para indicar a relação de Dependência. Na Figura 2, uma reta não tracejada com terminal em triângulo representa a Generalização. Finalmente, na Figura 3, uma reta, sem terminação representa o relacionamento de Associação.



**Figura 1- Representação de Dependência**



**Figura 2-Relação de Generalização**



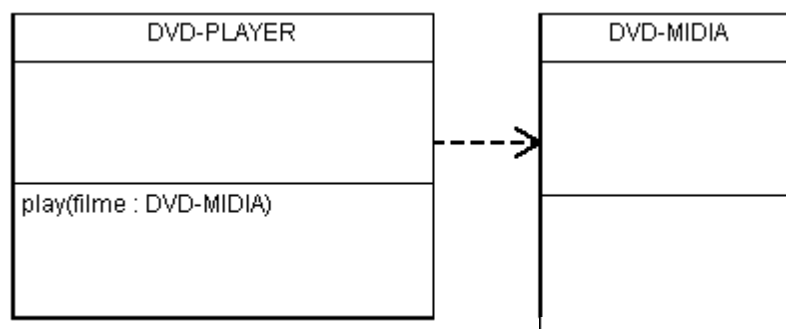
**Figura 3-Relação de Associação**

## O Relacionamento de Dependência ( “ precisa de ” )

Duas classes possuem um relacionamento de Dependência quando uma alteração em uma dessas classes poder afetar a outra classe. O inverso não é verdade. Nesse contexto, diz-se que uma classe utiliza a outra como argumento em sua assinatura.

Representa-se uma relação de Dependência apontando de uma classe que possui um método ou operação para a classe que é utilizada como um parâmetro para essa operação.

Por exemplo, na Figura 4, as classes DVD-PLAYER e DVD-MIDIA apresentam um relacionamento de dependência. A assinatura do método *play* da classe DVD-PLAYER recebe como parâmetro um objeto ou instância da classe DVD-MIDIA. Nesse exemplo da Figura 4, esse objeto é rotulado por “filme”. O sentido da seta indica quem depende de quem.



**Figura 4-Exemplo de Dependência**

No quadro a seguir, as duas classes e seu relacionamento representadas na Figura 4 são codificadas ou traduzidas para as linguagens C++ (arquivos “.cpp” e “.h”) e Java (arquivo “.java”). Esse código gerado é representado nos arquivos .h e .cpp dessas duas classes.

**Quadro 1 – Código C++ e Java do Exemplo da Figura 4**

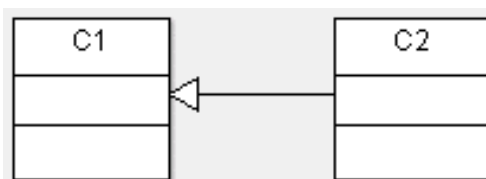
	<b>DVD-PLAYER</b>	<b>DVD-MIDIA</b>
<b>.h</b>	<pre>#ifndef DVD-PLAYER_h #define DVD-PLAYER_h #include "DVD-MIDIA.h"  class DVD-PLAYER { public:     void play(DVD-MIDIA filme); };  #endif // DVD-PLAYER_h</pre>	<pre>#ifndef DVD-MIDIA_h #define DVD-MIDIA_h  class DVD-MIDIA {};  #endif // DVD-MIDIA_h</pre>
<b>.cpp</b>	<pre>#include "DVD-PLAYER.h"  void DVD-PLAYER::play(DVD-MIDIA filme) {    }</pre>	<pre>#include "DVD-MIDIA.h"</pre>
<b>.java</b>	<pre>public class DVD-PLAYER {     public play(DVD-MIDIA filme)     {    } }</pre>	<pre>public class DVD-MIDIA { }</pre>

O relacionamento de Dependência é caracterizado quando uma operação precisa de uma outra classe para ser executada. O seu significado semântico é “**precisa de**”.

## **O Relacionamento de Generalização ( “ é um tipo de ” )**

Duas classes possuem um relacionamento de Generalização quando uma das classes especializa ou detalha a outra. A classe genérica é denominada de SuperClasse ou Classe Pai e a outra classe de SubClasse ou classe Filha.

A relação de Generalização é identificada por meio do texto “é um”. Na Figura 5, a seguir, simplifica-se um modelo genérico de generalização. Nesse exemplo, a seta terminadora aponta sempre para a classe mais genérica.



**Figura 5 -Relacionamento de Generalização**

O Quadro 2 apresenta os arquivos gerados para as classes C1 e C2 da Figura 5. Destaca-se que no código de definição da classe C2, informa-se de maneira explícita qual é a classe Pai. Na linguagem C++ isso é feito por meio do caracter “:”, da forma de acesso (*public* nesse caso) e do nome da classe genérica. Por outro lado, na linguagem Java, utiliza-se apenas a palavra reservada “**extends**” seguida do nome da classe genérica ou classe-pai.

**Quadro 2 – Código C++ e Java do Exemplo da Figura 6**

	<b>C1</b>	<b>C2</b>
<b>.h</b>	<pre> #ifndef C1_h #define C1_h  class C1 {};  #endif // C1_h                     </pre>	<pre> #ifndef C2_h #define C2_h  #include "C1.h"  <b>class C2 : public C1 {};</b>  #endif // C2_h                     </pre>
<b>.cpp</b>	<pre> #include "C1.h"                     </pre>	<pre> #include "C2.h"                     </pre>
<b>.java</b>	<pre> public class C1 { }                     </pre>	<pre> public class C2 <b>extends</b> C1 { }                     </pre>

Como outro exemplo de generalização, a Figura 6 a seguir, três classes são representadas: Pessoa, Engenheiro e Musico. Nessa representação, a generalização permite dizer “Engenheiro é um(a) Pessoa” e “Musico é um(a) Pessoa”. Ou seja, os objetos do mundo real representados pela classe Engenheiro e pela classe Pessoa, possuem algumas características comuns ( atributos e método ). Essas características comuns são os atributos e métodos da classe Pessoa.

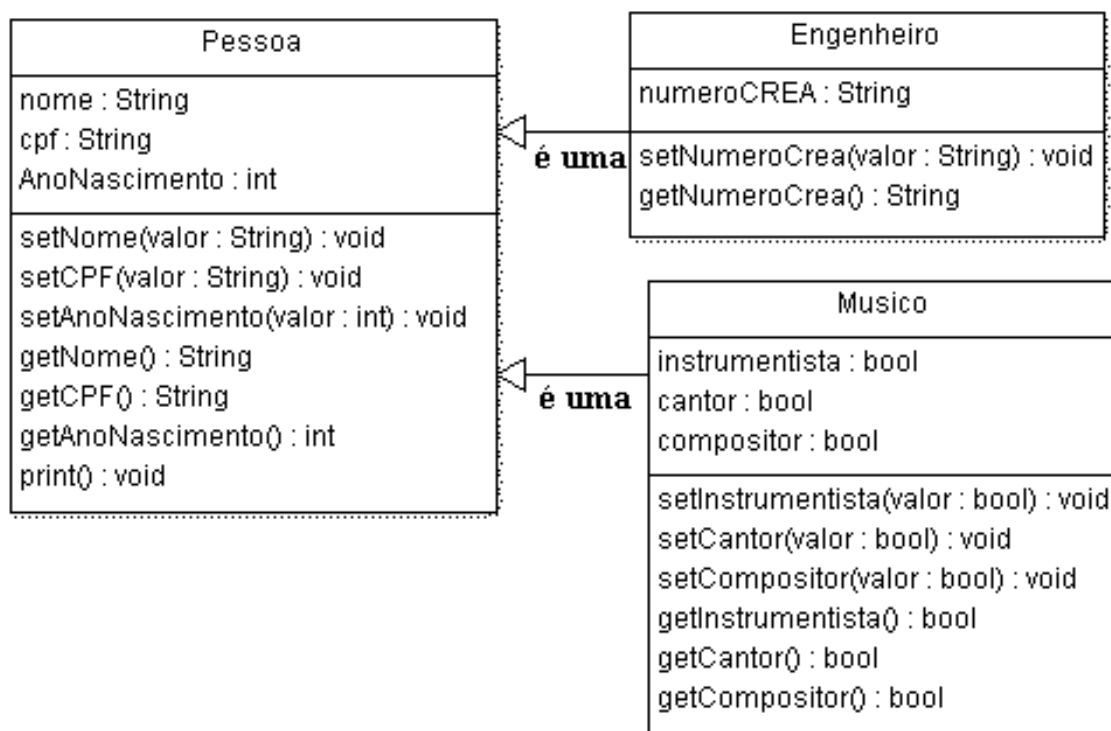


Figura 5-Exemplo de Generalização





## O Relacionamento de Associação

Uma Associação é uma relação estrutural. Ou seja, ela informa que uma classe faz parte da estrutura de outra. Por exemplo: o Motor faz parte da classe Automóvel, um empregado está associado a uma empresa ou um músico está associado a uma banda. Definida uma Associação entre duas classes, conforme mostrado na Figura 3, pode-se navegar de um objeto de uma classe para a outra e vice e versa.

O Relacionamento de Associação ainda pode ser subdividido em:

- **Plana** : representa uma relação estrutural onde as classes possuem a mesma importância. Uma linha ligando duas classes representa graficamente essa relação. A Figura 3 ilustra esse tipo de relacionamento.
- **Agregação**: representa a estrutura todo-parte. Ela é representada por uma linha ligando as duas classes da relação e a presença de um símbolo diamante (losango). A figura de um diamante é colocado na conexão entre a linha e a classe que é considerada a mais importante da relação.

Esse diamante pode ser aberto ou escuro, a saber:

-  : Se aberto, tem-se **Agregação Simples**.
-  : Se escuro, têm-se Agregação por Composição, ou simplesmente **Composição**. Essa relação será explicada em uma seção específica deste material.

Uma Associação pode também ter um nome ou rótulo. Nesse caso, esse nome deve representar a natureza da relação. As Figuras 7a) e b) exemplificam isso. A Figura 7a) representa a relação “Saxofonista *toca\_com* Baterista”. Essas duas classes são igualmente importantes. Nesse exemplo, tanto a classe Baterista quanto a classe Saxofonista possuem a mesma importância.

A Figura 7b) representa que “Musico *toca\_em* Orquestra”. Nesse contexto, deseja-se destacar que a classe Orquestra é “mais importante ou maior” que a classe Música.

Os Quadros 3a) e 3b), apresentam respectivamente os códigos Java e C++ dessas classes representadas respectivamente nas Figuras 7a) e 7b).

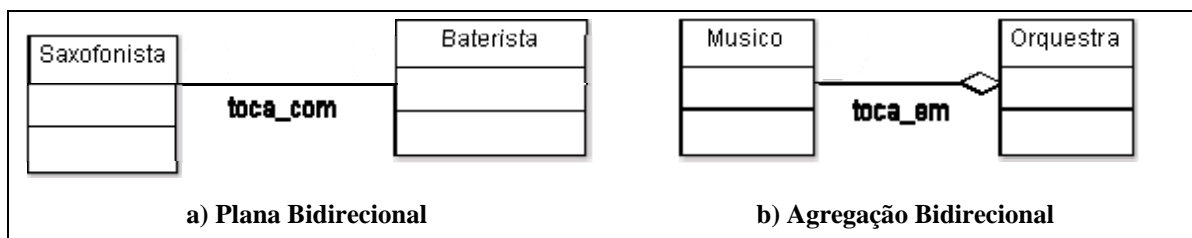


Figura 6-Exemplo de Associação

Quadro 3a) – Códigos C++ e Java do Exemplo da Figura 7a)

	Saxofonista	Baterista
.h	<pre>#ifndef Saxofonista_h #define Saxofonista_h  class Bateria;  class Saxofonista {  public:     Bateria *toca_com; };  #endif // Saxofonista_h</pre>	<pre>#ifndef Bateria_h #define Bateria_h  class Saxofonista;  class Bateria {  public:     Saxofonista *toca_com; };  #endif // Bateria_h</pre>
.cpp	<pre>#include "Saxofonista.h"</pre>	<pre>#include "Bateria.h"</pre>
.java	<pre>import java.util.Vector;  public class Saxofonista {     public Vector toca_com; }</pre>	<pre>import java.util.Vector;  public class Bateria {     public Vector toca_com; }</pre>

Quadro 3b) – Códigos C++ e Java do Exemplo da Figura 7b)

	Musico	Orquestra
.h	<pre> #ifndef Musico_h #define Musico_h  class Orquestra;  class Musico {  public:      Orquestra *toca_em; };  #endif // Musico_h                     </pre>	<pre> #ifndef Orquestra_h #define Orquestra_h  class Musico;  class Orquestra {  public:      Musico* toca_em; };  #endif // Orquestra_h                     </pre>
.cpp	<pre> #include "Musico.h"                     </pre>	<pre> #include "Orquestra.h"                     </pre>
.java	<pre> import java.util.Vector;  public class Musico {     public Vector toca_em; }                     </pre>	<pre> import java.util.Vector;  public class Orquestra {     public Vector toca_em; }                     </pre>

Nos quadros 3 a) e b), pode-se observar a referência cruzada entre as duas classes da relação. A classe Bateria possui uma referência para a classe Saxofonista e vice e versa no Quadro 3a). A classe Musico possui uma referencia para a classe Orquestra e vice e versa.

Por padrão, a relação de Associação é bidirecional. Apesar disso, pode-se também definir relações de Associação unidirecional.

O que significa ser bidirecional? No exemplo apresentado na Figura 3a), significa que a classe Saxofonista tem registrado com quem o saxofonista toca e

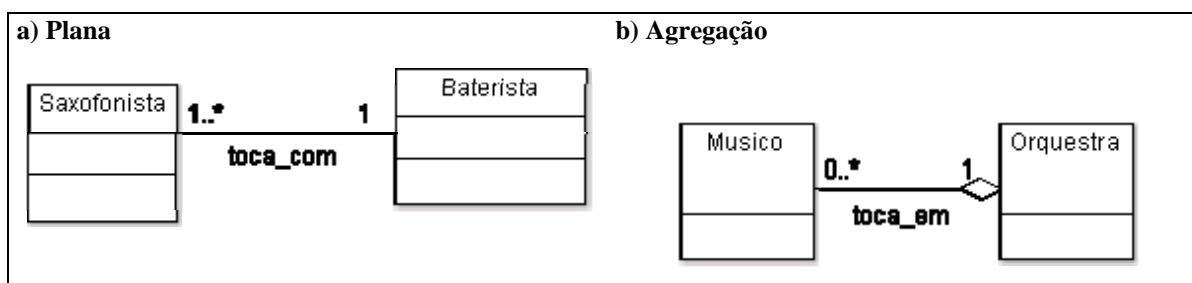
que o mesmo acontece com a classe Bateria. A classe Bateria também tem registrado com quem ele toca.

Destaca-se aqui, que a simples inspeção dos códigos produzidos nas linguagens C++ e Java não permite identificar “o lado do losango” nem se ele está presente na associação.

### ***Multiplicidade.***

A relação de Associação pode também informar a quantidade de elementos que são necessários para a estrutura. Por exemplo, A Figura 8a) registra que um Bateria toca com um ou mais saxofonistas. Por sua vez, essa figura também representa que um saxofonista toca apenas com um único bateria.

A Figura 8b) que uma Orquestra é formada por nenhum ou vários músicos e que um Musico obrigatoriamente toca\_em uma Orquestra.



**Figura 7-Exemplo de Associação com Multiplicidade**

No Quadro 4a), observa-se que a classe Bateria armazena em um vetor ( `std::Vector < Saxofonista * >` ) ponteiros para elementos da classe Saxofonistas que estão associados a ela.

No Quadro 4 b), a representação é similar ao Quadro 4 a). A classe mais importante semanticamente falando, possui um vetor de ponteiros que indicam quem são os elementos que estão associados a ela.

Caso seja necessário especificar uma quantidade mínima diferente de um, é só informar. Por exemplo, pode-se especificar que a relação deve ser delimitada entre 4 e 8 integrantes. Nesse caso, ao invés de registrar “1..\*”, seria “4..8”.

Quadro 4 a) – Código C++ e Java do Exemplo da Figura 8 a)

	<b>Saxofonista</b>	<b>Baterista</b>
<b>.h</b>	<pre>#ifndef Saxofonista_h #define Saxofonista_h  class Bateria;  class Saxofonista {  public:      <b>Bateria *toca_com;</b> };  #endif // Saxofonista_h</pre>	<pre>#ifndef Bateria_h #define Bateria_h  #include &lt;vector&gt;  class Saxofonista;  class Bateria {  public:      <b>std::vector&lt; Saxofonista* &gt; toca_com;</b> };  #endif // Bateria_h</pre>
<b>.cpp</b>	<pre>#include "Saxofonista.h"</pre>	<pre>#include "Bateria.h"</pre>
<b>.java</b>	<pre>public class Saxofonista {     public Bateria toca_com; }</pre>	<pre>import java.util.Vector;  public class Bateria {     public Vector  toca_com; }</pre>

**Quadro 4 b) – Código C++ e Java do Exemplo da Figura 8 b)**

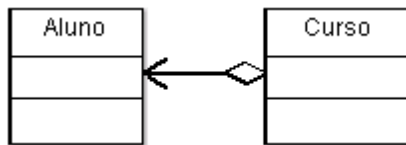
	<b>Musico</b>	<b>Orquestra</b>
<b>.h</b>	<pre>#ifndef Musico_h #define Musico_h  class Orquestra;  class Musico {  public:      Orquestra *toca_em; };  #endif // Musico_h</pre>	<pre>#ifndef Orquestra_h #define Orquestra_h #include &lt;vector&gt;  class Musico;  class Orquestra {  public:      std::vector&lt; Musico* &gt; toca_em; };  #endif // Orquestra_h</pre>
<b>.cpp</b>	<pre>#include "Musico.h"</pre>	<pre>#include "Orquestra.h"</pre>
<b>.java</b>	<pre>public class Musico {     public Orquestra toca_em; }</pre>	<pre>import java.util.Vector;  public class Orquestra {     public Vector  toca_em; }</pre>

### ***Relações Unidirecionais de Associação***

As relações de Associação apresentadas até o presente momento são bidirecionais. Ou seja, todas as classes que participam da relação tem a consciência disso.

Em algumas situações, pode-se desejar que uma das classes que participa da relação não tenha consciência disso. Nesses casos, têm-se uma relação Unidirecional.

A representação gráfica da relação Unidirecional consiste em utilizar um terminador seta apontando para a classe que deve desconhecer a relação. A Figura 9, a seguir representa essa possibilidade.



**Figura 8-Exemplo de Relação Unidirecional**

No exemplo da Figura 9, a classe Aluno desconhece a relação que a classe Curso tem com ela. O quadro 5, a seguir, apresenta o correspondente código dessa relação.

Nos códigos do Quadro 5, o arquivo Aluno.h não possui referência alguma da classe Curso. Por outro lado, a classe Curso possui em seu código, referências à classe Aluno.

**Quadro 5 – Código C++ e Java do Exemplo da Figura 9**

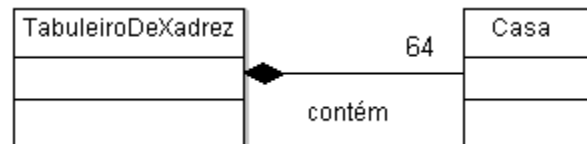
	<b>Aluno</b>	<b>Curso</b>
<b>.h</b>	<pre>#ifndef Aluno_h #define Aluno_h  class Aluno {  };  #endif // Aluno_h</pre>	<pre>#ifndef Curso_h #define Curso_h  class Aluno;  class Curso { public:     Aluno *myAluno; };  #endif // Curso_h</pre>
<b>.cpp</b>	<pre>#include "Aluno.h"</pre>	<pre>#include "Curso.h"</pre>
<b>.java</b>	<pre>public class Aluno { }</pre>	<pre>import java.util.Vector;  public class Curso {     public Vector myAluno; }</pre>

### **Composição ( “...pertence exclusivamente a ...” ) —◆**

A Agregação Simples possui uma variante, a Composição. Essa variação adiciona um grau de importância semântica à relação. Ela define a relação de posse ou possessão. Isso significa que um objeto da classe pertence apenas e exclusivamente ao objeto da outra classe.

Como a classe mais importante (ou todo) é responsável pela disposição das partes. Isso significa que ela precisa gerenciar a criação e destruição das partes que a compõe.

Por exemplo, na Figura 10, a seguir, representa-se a classe TabuleiroDeXadrez que deve conter instâncias da classe Casa. A classe Casas, que pode ter o atributo cor como sendo “preta” ou “branca”, compõe um Tabuleiro de Xadrez. Mais especificamente, um tabuleiro de Xadrez é formado, composto por 32 casas pretas e 32 casas brancas.



**Figura 9-Exemplo de Relacionamento de Composição**

No Quadro 6, os respectivos códigos das classes da Figura 10 são apresentados. Destaca-se nesse relacionamento, a inexistência de ponteiro na classe TabuleiroDeXadrez. Como os elementos da classe Casa compõe apenas a classe TabuleiroDeXadrez e existem apenas com essa principal finalidade, não há a necessidade de ponteiro. A classe TabeuleiroDeXadrez deverá instanciar os 64 elementos dessa classe.



**Quadro 6 – Códigos C++ e Java do Exemplo da Figura 10**

	<b>TabuleiroDeXadrez</b>	<b>Casa</b>
<b>.h</b>	<pre> #ifndef TabuleiroDeXadrez_h #define TabuleiroDeXadrez_h  #include "Casa.h"  class TabuleiroDeXadrez {  public:      Casa contem[64];  };  #endif // TabuleiroDeXadrez_h                     </pre>	<pre> #ifndef Casas_h #define Casas_h  class TabuleiroDeXadrez;  class Casa {  public:      TabuleiroDeXadrez *contem;  };  #endif // Casa_h                     </pre>
<b>.cpp</b>	<pre> #include "TabuleiroDeXadrez.h"                     </pre>	<pre> #include "Curso.h"                     </pre>
<b>.java</b>	<pre> public class TabuleiroDeXadrez {     Casas contem = new Casa[64]; }                     </pre>	<pre> public class Casa {     TabuleiroDeXadrez contem; }                     </pre>

De maneira similar aos outros relacionamentos, a Composição também pode ser unidirecional. Nesse caso, a classe que desconhece a relação não faz referência à ela em seu código.

## Propriedades Da Relação

Em algumas situações, a relação entre duas classes possui propriedades. Por exemplo, a relação entre a classe Paciente e AgendaDoMédico é a Consulta. Essa classe só existe para representar as propriedades da relação entre as duas classes Paciente e AgendaDoMedico.

Nesses casos, representa-se em UML as três classes, conforme exemplifica a Figura 11 a seguir.



**Figura 10-Classe Propriedades da Relação**

A classe que deverá registrar as propriedades da relação tem uma ligação pontilhada com as classes da relação.

O Quadro 7 apresenta o código C++ de cada uma das classes genéricas ilustradas na Figura 11. Destaca-se nesse quadro que as classes da relação C1 e C2 devem registrar por meio de ponteiros qual é a classe que registra os detalhes da relação.

A classe que detalha a relação, PropriedadesDaRelacao, neste caso, registra quais são as classes detalhadas sem o uso de ponteiros.

**Quadro 7 – Código C++ do Exemplo da Figura 11**

Classe \ código	.h
C1	<pre> #ifndef C1_h #define C1_h  class PropriedadesDaRelacao;  class C1 {  public:      PropriedadesDaRelacao *PropriedadesDaRelacaoAssoc; };  #endif // C1_h                     </pre>

PropriedadesDaRelacao	<pre>#ifndef PropriedadesDaRelacao_h #define PropriedadesDaRelacao_h  #include "C1.h" #include "C2.h"  class PropriedadesDaRelacao { public:     C1 myC1;     C2 myC2; };  #endif // PropriedadesDaRelacao_h</pre>
C2	<pre>#ifndef C2_h #define C2_h  class PropriedadesDaRelacao;  class C2 { public:     PropriedadesDaRelacao *PropriedadesDaRelacaoAssoc; };  #endif // C2_h</pre>

## Dicas

Ao modelar relações em UML:

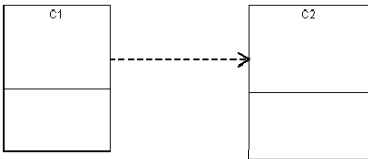
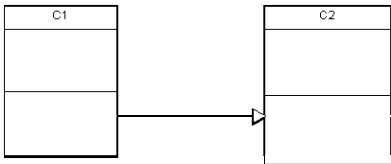
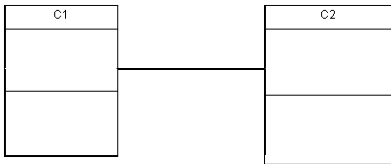
- Utilize Dependências apenas quando o relacionamento modelado não for estrutural.
- Utilize Generalização apenas quando voce conseguir ter um “ é um tipo de “ entre as classes representadas.

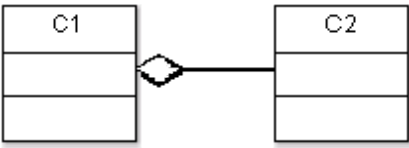
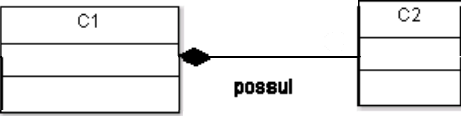
- Cuidado com a generalização cíclica. Generalização cíclica é quando um ciclo fechado é definido entre classes que se relacionam pela Generalização.
- Utilize Associação primeiramente quando houver uma relação estrutural entre os objetos.

## Resumo.

Programar de maneira orientada a objetos, consiste, entre outras coisas, definir classes e seus relacionamentos. Classes se relacionam com outras Classes com a finalidade de representar objetos e sistemas do mundo real.

Em resumo, os relacionamentos apresentados neste material são:

Relacionamento	Símbolo	Significado
<b>Dependência</b>		A classe C1 depende da classe C2 para executar alguma tarefa. Pode-se dizer que: “C1 precisa de C2”.
<b>Generalização ou Herança</b>		A classe C1 é dita filha da classe C2. A classe C1 possui toda a estrutura da classe C2. Pode-se dizer: “ C1 é um tipo de C2 “.
<b>Associação Plana</b>		As classes C1 e C2 são igualmente importantes e são associadas de maneira estrutural para representar uma idéia, conceito ou um todo.

<b>Agregação Simples</b>		<p>As classes C1 e C2 representam uma relação estrutural. C1 e C2 não são igualmente importantes.</p> <p>O diamante indica qual a classe mais importante da relação.</p>
<b>Composição</b>		<p>As classes C1 e C2 apresentam uma relação estrutural muito forte. C2 só existe por causa de C1.</p> <p>O diamante escuro indica quem é o mais importante da relação.</p>

## Exercício:

Booch et al [1] apresenta um diagrama de classe UML exemplo. A Figura 12 registra esse exemplo.

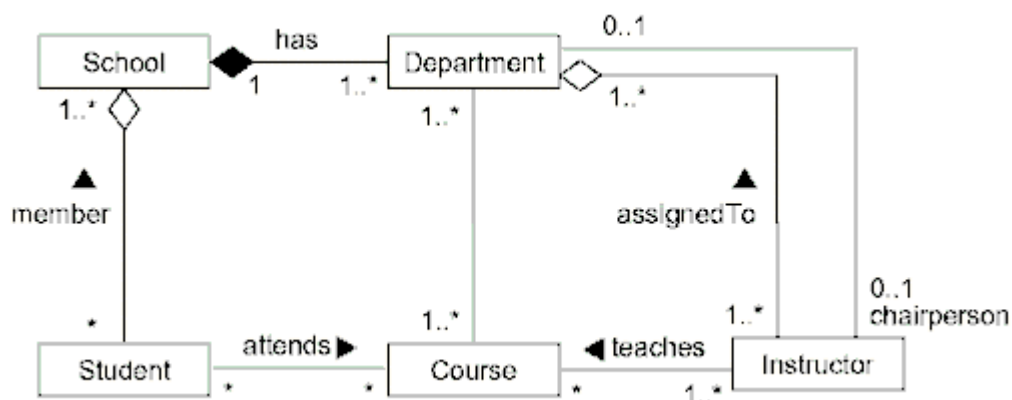


Figura 11- Diagrama de Classes (Booch et al, 1998)

Responda:

- a) Qual é a classe mais importante do Diagrama?
- b) Que tipo de relação existe entre a classe School e Student?
- c) Que tipo de relação existe entre a classe School e Department?
- d) Que tipo de relação existe entre a classe Course e Instructor?
- e) Quais as relações existentes entre a classe Instructor e Department?
- f) Como seria o código .h da classe School ?
- g) Como seria o código .h da classe Department ?
- h) Como seria o código .h da classe Course?
- i) Descreva com suas palavras, como o autor definiu a relação entre as classes.

## **Referência Bibliográfica Utilizada**

- [1] **Unified Modeling Language User Guide, The.** Grady Booch, James Rumbaugh, Ivar Jacobson. Publisher: Addison Wesley. First Edition October 20, 1998 ISBN: 0-201-57168-4, 512 pages