PROGRAMAÇÃO ORIENTADA A OBJETOS

MODELAGEM DE CLASSES

AGENDA

1. Processo de desenvolvimento de software.
2. Unified Modeling Language (UML).
3. Dicas para um modelagem eficiente.
4. Diagramas da UML.
5. Diagrama de Classes.
6. Criação do Diagrama de Classes.

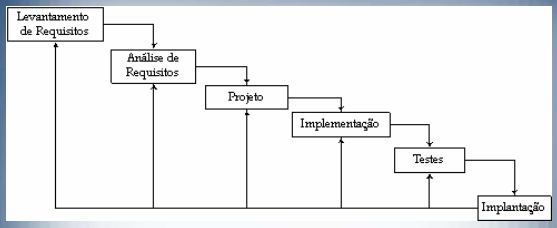
PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- **Processo em Cascata:**
  - Subdivide um projeto com base nas atividades;
  - Considerando que, para construir um software, você precisa de realizar certas atividades: análise de requisitos, projeto, codificação e teste;
  - Para um projeto de um ano ter-se-ia:
    - Uma fase de análise de dois meses.
    - Uma fase de projeto de quatro meses.
    - Uma fase de codificação de três meses.
    - Um fase de testes de três meses.

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- **Metodologia Estruturada**
  - Baseada no processo em cascata.
  - Fases:
    1. Levantamento de Requisitos
    2. Análise de Requisitos
    3. Projeto
    4. Implementação
    5. Testes
    6. Implantação

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE



PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- **Processo Iterativo:**
  - Subdivide o processo em subconjuntos de funcionalidades;
  - Para um projeto de um ano, poder-se-ia:
    - Dividi-lo em iterações de três meses cada;
    - Na primeira iteração, você pegaria um quarto dos requisitos e faria o ciclo de vida do software para este quarto: análise, projeto, código e teste;
    - No final desta iteração você teria um sistema que faria um quarto das funcionalidades do sistema;
    - Então, você faria um segunda iteração tal que, no final de seis meses, tivesse um sistema que fizesse a metade da funcionalidade do sistema.

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- **Rational Unified Process (RUP)**
  - Baseado no processo iterativo;
  - Fases:
    1. Concepção
    2. Elaboração
    3. Construção
    4. Transição

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

1. **Concepção**
   - Avaliação inicial do projeto;
   - Incorpora o estudo de viabilidade e uma parte da análise de requisitos;
2. **Elaboração**
   - Incorpora a maior parte da análise de requisitos, a análise de domínio e o projeto;
   - Identifica os casos de uso principais do projeto;

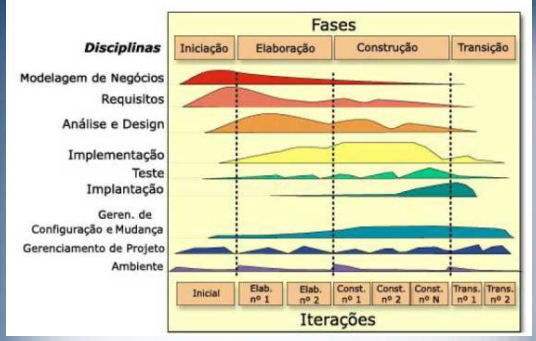
PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

3. **Construção**
   - Corresponde a programação e testes;
   - Deve deixar o sistema pronto para o lançamento;
4. **Transição**
   - Instalação;
   - Treinamento dos usuários;
   - Manutenção do sistema.

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- Nessas fases utilizamos **workflows**, que são atividades definidas para cada uma delas;
- Esses **workflows** são:
  - Requisitos
  - Análise
  - Projeto
  - Implementação
  - Teste

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE



PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- Uma determinada fase pode utilizar um ou todos os **workflows**;
- O que uma fase produz são artefatos, que podem ser:
  - Diagramas
  - Documentos
  - Conceitos
  - Modelos

UNIFIED MODELING LANGUAGE (UML)

- **Linguagem de Modelagem**
  - **Notação gráfica** para descrever projetos de software;
- Importante:
  - Não é um **processo** nem uma **metodologia**;
  - Uma **metodologia** diz como projetar um software, enquanto a linguagem de modelagem ilustra o projeto criado através da **metodologia**;

UNIFIED MODELING LANGUAGE (UML)

- É uma **linguagem de modelagem** padrão que consiste na notação para descrever cada aspecto de um projeto de software;
- Particularmente adequada para a modelagem de sistemas **orientados a objetos**;

UNIFIED MODELING LANGUAGE (UML)

- **Objetivos:**
  - Permitir a modelagem de sistemas (não apenas softwares) usando conceitos da OO;
  - Estabelecer um acoplamento explícito com artefatos conceituais e executáveis;
  - Resolver questões de escala inerentes a sistemas complexos e de missão crítica;
  - Criar uma linguagem de modelagem utilizável por humanos e máquinas.

DICAS PARA UMA MODELAGEM EFICIENTE

- **Pergunte**: O que estou tentando transmitir?
  - A resposta mostra o que precisa ser modelado;
- **Pergunte**: Para quem estou tentando transmitir a informação?
  - A resposta mostra como vai ser modelado.

DICAS PARA UMA MODELAGEM EFICIENTE

- O modelo tem que ser o mais simples possível;
- Não fique preso a linguagem de modelagem:
  - O modelo só precisa transmitir corretamente o projeto.
- A UML é simplesmente uma ferramenta para auxiliar na transição do projeto:
  - Ainda será necessária produzir código.

DIAGRAMAS DA UML

DIAGRAMA	OBJETIVO
Atividades	Comportamento procedimental e paralelo.
Classes	**Classe, características e relacionamentos.**
Comunicação	Interação entre objetos; ênfase nas ligações.
Componentes	Estrutura e conexão de componentes.
Estruturas Compostas	Decomposição de uma classe em tempo de execução.
Distribuição	Distribuição de artefatos nos nós

DIAGRAMAS DA UML

DIAGRAMA	OBJETIVO
Visão Geral da interação	Mistura de diagrama de seqüência e de atividades.
Objetos	Exemplo de configurações de instâncias.
Pacotes	Estrutura hierárquica em tempo de compilação.
Seqüência	Interação entre objetos; ênfase na seqüência.
Máquina de Estado	Como os eventos alteram um objeto no decorrer de sua vida.
Sincronismo	Interação entre objetos; ênfase no sincronismo.
Casos de Uso	Como os usuários interagem com um sistema.

DIAGRAMAS DA UML

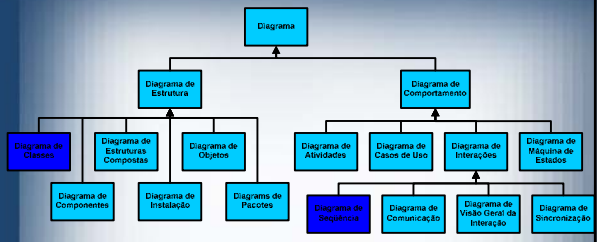


DIAGRAMA DE CLASSES

- Mostra as relações estáticas que existem entre um grupo de classes e interfaces no sistema.
- Um **diagrama de classes** consiste de:
  - Um conjunto de **nodos** representando classes e interfaces;
  - Um conjunto de **arcos** representando o relacionamento entre as classes.

DIAGRAMA DE CLASSES

- Uma **classe** é representada por uma caixa retangular com três compartimentos:
  - O **primeiro** compartimento apresenta o nome da classe;
  - O **segundo** compartimento apresenta os atributos da classe;
  - O **terceiro** compartimento apresenta os métodos da classe;

DIAGRAMA DE CLASSES

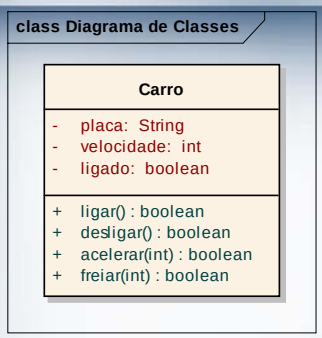


DIAGRAMA DE CLASSES

- Mostra:
  - As **propriedades** e as **operações** de uma classe;
  - As **restrições** que se aplicam a maneira como os objetos são conectados;
- Utiliza a palavra **característica** como um termo geral que cobre as **propriedades** e **operações** de uma classe.

DIAGRAMA DE CLASSES

- **Propriedades**
  - Representam as características estruturais de uma classe;
  - Correspondentes aos campos de uma classe;
  - São um conceito simples, mas elas aparecem em duas notações distintas: **atributos e associações**.
- **Observação:** associações nada mais são do que atributos de uma classe em outra classe.

DIAGRAMA DE CLASSES

- **Atributos**
  - Descreve uma propriedade como uma linha de texto dentro da caixa de classe em si;
  - Sintaxe:

```
visibilidade nome: tipo multiplicidade = valor por omissão
nome: String[1] = "Sem título"
```

  - Somente o **nome** é necessário.

DIAGRAMA DE CLASSES

- **Associações**
  - Linha cheia entre duas classes, direcionada da classe de origem para a classe de destino;
  - O nome da propriedade fica no destino final da associação junto com a multiplicidade;
  - O destino final da associação vincula à classe que é o tipo de propriedade;

DIAGRAMA DE CLASSES

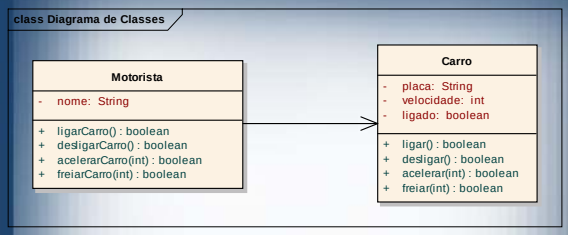


DIAGRAMA DE CLASSES

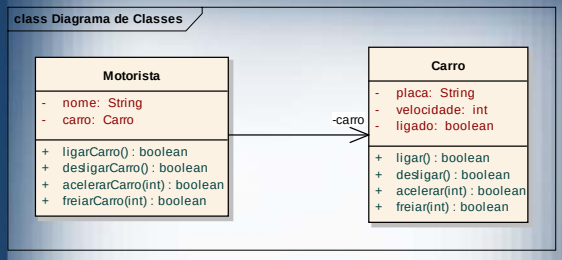


DIAGRAMA DE CLASSES

- **Propriedades (Atributos X Associações)**
  - Embora grande parte das mesmas informações apareça nas duas notações (atributo e associação), alguns itens são diferentes;
  - Em particular, as associações podem mostrar multiplicidades na duas extremidades da linha;

DIAGRAMA DE CLASSES

- **Propriedades (Atributos X Associações)**
  - Quando usar uma ou outra?
    - Atributos para coisas pequenas;
    - Associações para coisas significativas;
    - A escolha está relacionada à ênfase que se quer dar.

DIAGRAMA DE CLASSES

- **Operações**
  - São as operações que uma classe sabe realizar;
  - Correspondem aos **métodos** presentes em uma classe;
  - Normalmente você não mostra as operações que simplesmente manipulam propriedades, pois elas podem ser, na maioria das vezes, inferidas.

DIAGRAMA DE CLASSES

- **Operações**
  - Sintaxe para definição de uma operação:

```
visibilidade nome (lista de parâmetros): tipo de retorno
```

  - Sintaxe da lista de parâmetros:

```
nome: tipo [= valor por omissão]
```


RELACIONAMENTO ENTRE CLASSES

- Um **relacionamento** descreve como as classes interagem entre si;
- Um **relacionamento** é uma conexão entre dois ou mais elementos da notação;

RELACIONAMENTOS ENTRE CLASSES

- A UML reconhece quatro tipos de relacionamentos:
  - Dependência;
  - Associação (Simples, Agregação e Composição);
  - Herança (Generalização e Especialização);
  - Realização.

DEPENDÊNCIA

- Uma **dependência** entre dois elementos existe se mudanças na definição de um elemento (o fornecedor) podem causar mudanças ao outro lado (o cliente);
- Se uma classe muda a sua interface, qualquer mensagem enviada para essa classe pode não se mais válida.

DEPENDÊNCIA

- **Dependência** é o relacionamento mais simples entre objetos;
- Indica que um objeto depende da especificação de outro objeto;
- Especificação é uma maneira diferente de dizer **interface** ou **comportamento**;
- A relação de dependência é também chamada de **uso**.

DEPENDÊNCIA

- Uma classe **usa** outra classe se ela manipula objetos dessa classe;
- Uma classe A usa uma classe B se:
  - Um método de A envia uma mensagem para um objeto da classe B, ou
  - Um método de A cria, recebe ou retorna objetos da classe B.
- Sem, porém, ter a um objeto da classe B em seus atributos.

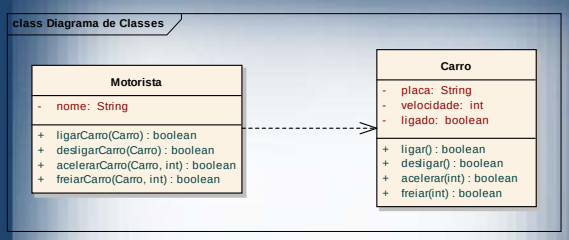
DEPENDÊNCIA

- Uma classe cliente é dependente de serviços de uma classe fornecedora;
- Não existe uma dependência estrutural interna com esse fornecedor;
- Alteração na especificação de uma classe poderá afetar a classe que a utiliza, mas não necessariamente o inverso.

DEPENDÊNCIA

- **Representação:**
  - Graficamente, uma dependência é representada por linhas tracejadas, possivelmente com setas e ocasionalmente incluindo um rótulo.

DEPENDÊNCIA



ASSOCIAÇÃO

- Uma **associação** indica que um objeto contém outro objeto;
- O relacionamento associação é também chamado de **inclusão**;

ASSOCIAÇÃO

- A relação de **inclusão** é fácil de entender porque ela é concreta;
- A inclusão significa que objetos da classe A contêm objetos da classe B;
- A **associação** permite uma classe ser parte ou usar uma outra classe;
- A associação é um relacionamento estrutural.

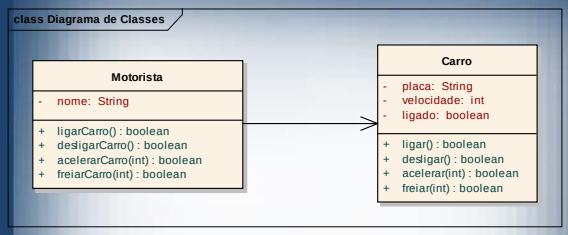
ASSOCIAÇÃO

- De certa forma, a classe que possui um objeto de outra classe, depende desta classe (poderia ter-se um relacionamento de dependência);
- Mas neste caso, a associação é um relacionamento mais forte e sendo assim, deve ser representada como tal.

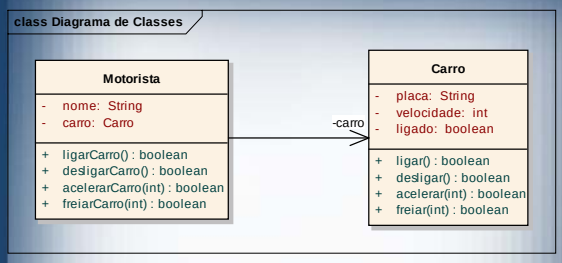
ASSOCIAÇÃO

- **Representação:**
  - Graficamente, uma associação é representada por linhas sólidas, possivelmente direcionadas, ocasionalmente incluindo rótulos, e freqüentemente, contendo outros adornos, como nomes de papéis e multiplicidades.

ASSOCIAÇÃO



ASSOCIAÇÃO



ASSOCIAÇÃO

- Existem alguns aprimoramentos que podem ser aplicados às associações:
  - nome: descreve a natureza do relacionamento;
  - papel: descreve a face que a classe próxima a uma das extremidades apresenta à classe encontrada na outra extremidade da associação;
  - multiplicidade: descreve a quantidade de objetos que a classe próxima a uma das extremidades deve ter da classe na outra extremidade da associação;
  - navegabilidade: descreve a forma como se pode navegar entre os objetos das classes.

ASSOCIAÇÃO

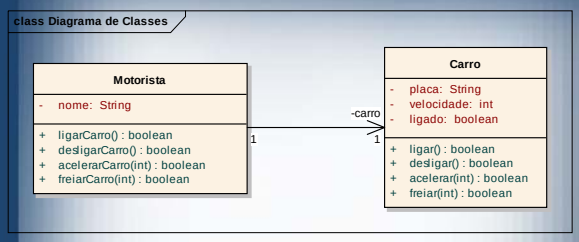


DIAGRAMA DE CLASSES

- **Multiplicidade**
  - Propriedade que é uma indicação de quantos objetos podem preencher a propriedade;
  - Multiplicidades comuns:
    - 1 (um Pedido deve ter exatamente um Cliente);
    - 0..1 (um ClienteCorporativo pode ter ou não um único RepresentanteDeVendas);
    - * (um Cliente não precisa fazer um Pedido e não existe nenhum limite superior para o número de Pedidos que um Cliente pode fazer – zero ou mais pedidos;
    - n..m (multiplicidades com um limite inferior e um limite superior).

DIAGRAMA DE CLASSES

- **Multiplicidade**
  - Nos atributos, existem termos que se referem a multiplicidade:
    - **Opcional:** significa um limite inferior igual a 0;
    - **Obrigatório:** significa um limite inferior igual a 1 ou possivelmente mais;
    - **Valor único:** significa um limite superior igual a 1;
    - **Valores múltiplos:** significa um limite superior maior que 1;
  - Para propriedades com valores múltiplos, utilizar uma forma plural para o seu nome.

DIAGRAMA DE CLASSES

- **Navegabilidade**
  - Define como podemos navegar entre os objetos;
  - Tipos:
    - Unidirecional (em um sentido);
    - Bidirecional (nos dois sentidos);
  - Navegabilidade bidirecional é pouco utilizada e deve ser evitada sempre que possível.

DIAGRAMA DE CLASSES

- **Navegabilidade**
  - Uma **associação unidirecional** implica um objeto da classe a partir da qual está originando pode chamar os métodos na classe na direção na qual a seta está apontando;
  - No Java, isto se manifesta como uma variável de instância na classe que pode chamar os métodos.

DIAGRAMA DE CLASSES

- **Navegabilidade**
  - Uma **associação bidirecional** simplesmente significa que qualquer um dos objetos na associação pode chamar os métodos na outra;
  - No Java, isto resulta em um variável de instância em cada classe com base no tipo da outra classe.

DIAGRAMA DE CLASSES

- **Navegabilidade**
  - Implementar uma **associação bidirecional** em um linguagem de programação é um pouco complicada, pois você precisa certificar-se de que as duas propriedades se mantenham sincronizadas.
  - Por isto, é aconselhável evitar associações bidirecionais.

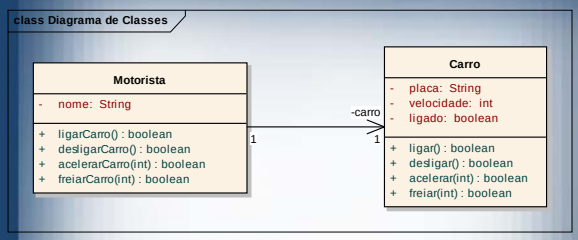
ASSOCIAÇÃO

- Pode ser de três tipos:
  - Simples
  - Agregação
  - Composição

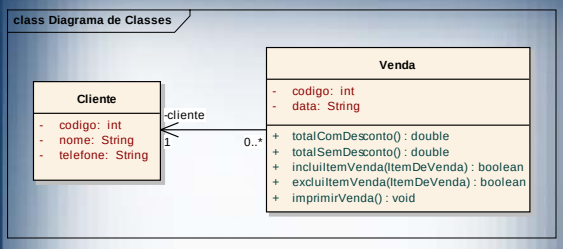
ASSOCIAÇÃO SIMPLES

- Relacionamento binário (entre duas classes);
- Não caracteriza uma relação todo-parte.

ASSOCIAÇÃO SIMPLES



ASSOCIAÇÃO SIMPLES



AGREGAÇÃO

- Utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação todo-parte;
- Indica que um objeto parte “é um atributo” do objeto todo;
- O ciclo de vida do objeto parte é é **diferente** do ciclo de vida do objeto todo.

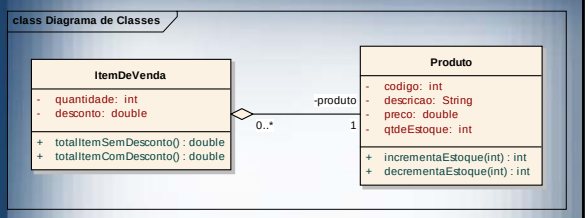
AGREGAÇÃO

- Uma associação do tipo **agregação** modela um relacionamento ***tem um*** (*ou* ***parte de um***) entre duas classes;
- Os objetos podem existir **independentes** uns dos outros;
- O objeto parte pode ser agregado a mais de um objeto todo.

AGREGAÇÃO

- **Representação**
  - Graficamente, a agregação é representada como uma associação com um losango não preenchido na classe que forma o todo.

AGREGAÇÃO



COMPOSIÇÃO

- Utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação todo-parte;
- Indica que um objeto parte "é um atributo" do objeto todo;
- O ciclo de vida do objeto parte é **limitado** ao ciclo de vida do objeto todo.

COMPOSIÇÃO

- Implica em um acoplamento muito mais forte do todo com a parte entre os participantes de modo que as partes não possam existir sem o todo;
- A **composição** é usada em agregações em que o tempo de vida da **parte** depende do tempo de vida do objeto **todo**.
- O objeto parte pode ser agregado a apenas um objeto todo.

COMPOSIÇÃO

- Representação
  - Graficamente, a composição é representada como uma associação com um losango preenchido na classe que forma o todo.

COMPOSIÇÃO

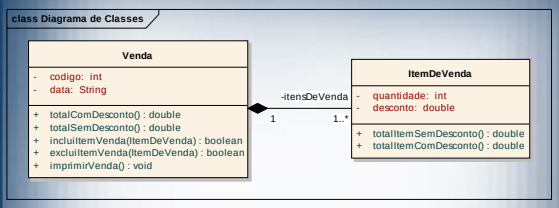
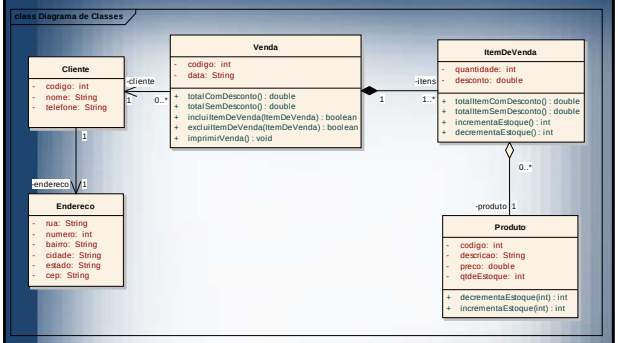


DIAGRAMA DE CLASSES



HERANÇA

- A **relação de herança** denota especialização;
- Se a classe A estender a classe B, a classe A vai herdar métodos e atributos da classe B e ter recursos a mais;
- A **herança** permite definir uma classe como uma extensão de outra.

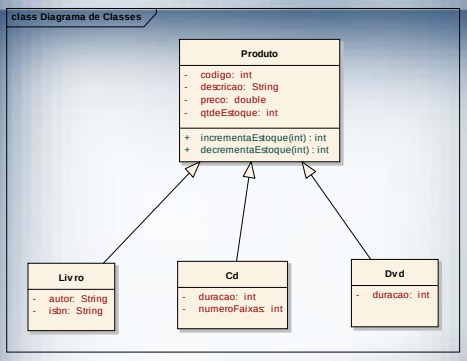
HERANÇA

- Nas principais linguagens orientadas a objetos, a subclasse herda todos os recursos da superclasse e pode sobrepor todos os métodos da superclasse;
- Um princípio importante para utilizar a herança eficientemente é a **capacidade de substituição**.
- No Java, uma classe pode herdar de no máximo uma outra classe.

HERANÇA

- **Representação**
  - Graficamente, um relacionamento de generalização é representado por uma linha sólida com uma seta em branco apontando o pai.

HERANÇA



REALIZAÇÃO

- Ocorre entre interfaces e as classes que as implementam;
- No Java, uma classe pode implementar uma ou mais interfaces.

REALIZAÇÃO

- **Representação**
  - Graficamente, um relacionamento de realização é representado por uma linha tracejada com seta branca entre uma interface e uma classe que a implemente.

REALIZAÇÃO