

# UML

**Profª: Flavia Garcia**  
**SENAC RIO**

# Análise Orientada a Objetos (AO)

## Conceitos:

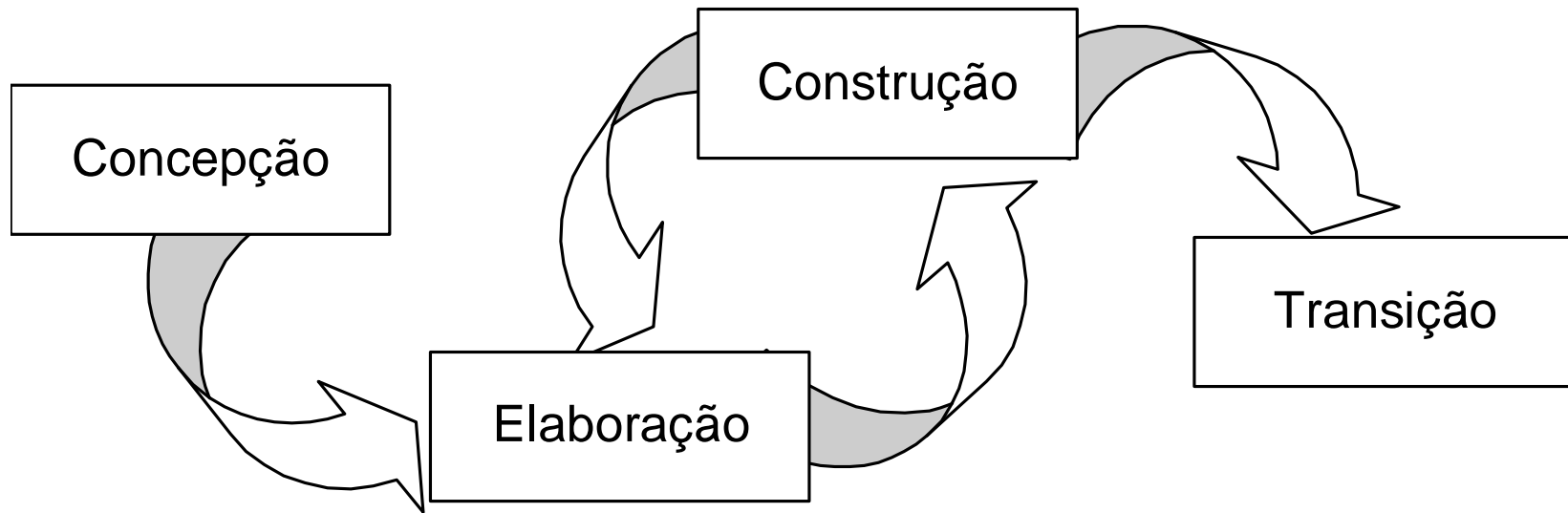
---

- **Objetos** : Em sistema de software objetos são agentes que agem interligados entre si, onde cada objeto realiza tarefas específicas e, através dessa interação, uma tarefa computacional é realizada.
- **UML** (*Unified Modeling Language* ou Linguagem de Modelagem Unificada) Linguagem padrão de diagramação, onde se visualizam os resultados da análise e do projeto.

- A Análise Orientada a Objetos (OOA) é um processo de desenvolvimento de sistemas que utiliza o conceito de objetos que interagem entre si e, através dessa interação, realizam tarefas computacionais.
- O ponto de partida para a OOA é criar um modelo descritivo contendo informações do projeto. Podemos chamar este artefato de proposta técnica, já que é ela que vai conter detalhes do problema a ser solucionado, objetivo do projeto, casos de uso, requisitos funcionais e não funcionais, as atividades do sistema e soluções propostas.
- Utilizando-se de padrões da UML, os requisitos do sistema são especificados, construindo artefatos necessários para documentação ao nível desejado.
- Durante todo o ciclo de vida do desenvolvimento (análise e programação), são gerados alguns diagramas que representam os objetos de análise, sendo eles: diagrama de caso de uso, de classe, de objeto, de componente, de implantação, de atividade, de estado, de colaboração e de sequencia.

# Ciclo de Vida

---



- A fase de *concepção* incorpora o estudo de viabilidade e uma parte da análise de requisitos.
- A fase de *elaboração* incorpora a maior parte da análise de requisitos, a análise de domínio e o projeto.
- A fase de *construção* corresponde à programação e testes.
- A fase de *transição* consiste na instalação e manutenção do sistema.

- Os casos de uso especificam o comportamento do sistema em partes, ou seja, o conjunto de cenários onde, cada cenário possui uma sequência de passos que descreve o comportamento e interação entre os atores e o sistema.
- A análise envolve o detalhamento das entidades através de diagrama de classes que é uma representação da estrutura e relação das classes que servem de modelo para os objetos.
- Cada regra é transformada em um elemento e este poderá ser reutilizado em vários cenários e casos de uso atendendo assim, umas das características da orientação a objetos que é a reusabilidade.
- Todos os elementos/objetos são associados entre si facilitando a rastreabilidade e com isto, quando um elemento é alterado, é fácil identificar todos os casos de usos e cenários que serão impactados e deverão ser considerados no projeto

## Diagramas UML



# Projeto Orientado a Objetos (POO)



## **Projeto Orientado a Objetos**

é um conjunto de atividades que têm como objetivo a criação de um modelo orientado a objetos de um sistema de software de acordo com os requisitos especificados.

### **Modelo de Objetos**

O projeto deve levar a uma implementação fácil de entender e construir, engloba a arquitetura e a implementação, sendo a arquitetura a primeira saída e a articulação das "coisas" de interesses dos sistemas e seus relacionamentos, e a implementação a segunda saída do projeto.

Muitos métodos contemporâneos veem o projeto como uma fase intermediária entre arquitetura e implementação, em vez de ver a arquitetura e o código como sendo produtos do projeto.

*Não se pode separar projeto de arquitetura e implementação porque o projeto é a atividade que dá estrutura a solução e a arquitetura está sobre a estrutura bem como boa parte do código.*

## Características de Projeto Orientado a objetos

Projeto orientado a objetos é uma estratégia de projeto em que os projetistas pensam em termos de *coisas*, *em vez de funções*.

A funcionalidade do sistema é expressa em termos de serviços oferecidos pelos objetos.

Objetos são abstrações do mundo real ou entidades do sistema que se auto gerenciam.

Objetos são independentes e encapsulam representações de informação e estado.

Áreas de dados compartilhado são eliminadas. Objetos se comunicam por passagem de mensagem.

*Projeto OO se dedica a desenvolver um modelo orientado a objeto de um sistema de software para implementar os requisitos. Os objetos em um projeto OO estão relacionados à **solução do problema que está sendo resolvido**.*

## **Vantagens do Projeto OO**

Facilidade de manutenção. Objetos podem ser entendidos como entidades independentes.

Os objetos são componentes potencialmente reutilizáveis.

Para vários sistemas, existe um nítido mapeamento entre as entidades do mundo real para objetos no sistema.

# Modelagem

# Modelagem

---

- A modelagem é uma técnica de engenharia aprovada e bem-aceita.
- Um modelo é uma simplificação da realidade
- Construimos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade.
- Construimos modelos para compreender melhor o sistema que estamos desenvolvendo.
- Ataque um problema difícil, dividindo-o em vários problemas menores que você pode solucionar.
- quanto mais complexo for o sistema, maior será a probabilidade de ocorrência de erros ou de construção de itens errados.
- Portanto, ainda que considere não ser preciso fazer a modelagem hoje, à medida que o seu sistema evoluir, você se arrependerá dessa decisão, quando for tarde demais.

## Pontos Principais

---

- Ênfase
- Complexidade
- Essência do procedimento
- Formal ou não formal
- Evolução
- Escolha do modelo correto

# Orientação a Objetos (00)

## Histórico

---

- A orientação a objetos (OO) foi concebida na década de 70.
- Origem na linguagem SIMULA-67 (década de 60 - Noruega), que já implementava alguns conceitos da OO.
- SIMULA-68 foi a primeira linguagem a suportar os conceitos da OO.
- Smaltalk, criada pela Xerox, popularizou e incentivou a OO.
- Outras linguagens OO: C++, Object Pascal (Delphi), C#, Java ...
- Java, de fato, popularizou a Orientação a Objetos.



## Orientação a Objetos

---

### Técnicas de programação tradicionais

As técnicas de programação tradicionais, como por exemplo a “decomposição funcional”, leva o desenvolvedor a decompor o sistema em partes menores (funções), criando um emaranhado de inúmeras funções que chamam umas às outras.

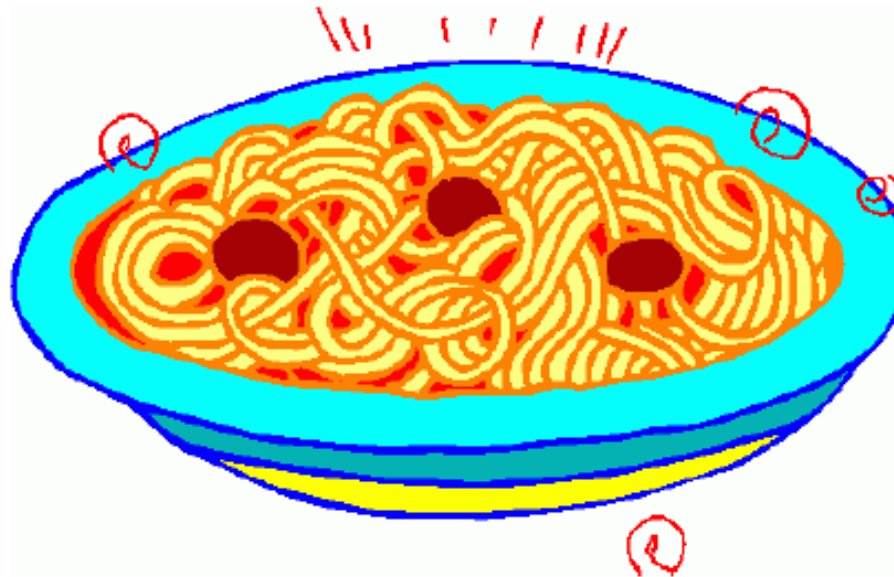
Geralmente não há separação de conceitos e responsabilidades, causando dependências enormes no sistema, dificultando futuras manutenções no código do programa.

Não existe muito reaproveitamento de código, ao contrário, muitas vezes se tem muito código duplicado.

## Orientação a Objetos

### Técnicas de programação tradicionais

É a famosa programação espaguete...



e pode causar uma grande indigestão.



## Orientação a Objetos

---

O paradigma da Orientação a Objetos, ou Programação Orientada a Objetos (POO ou OOP), eleva a programação e o desenvolvimento de sistemas para um novo patamar.

A OO é um mecanismo moderno que ajuda a definir a estrutura de programas baseada nos conceitos do mundo real, sejam eles reais ou abstratos.

A OO permite criar programas componentizados, separando as partes do sistema por responsabilidades e fazendo com que essas partes se comuniquem entre si, por meio de mensagens.

Essas partes do sistemas são chamadas de **OBJETOS**.

## Orientação a Objetos

---

A OO é mais intuitiva e fácil de aprender do que as técnicas tradicionais, pois foca o problema em conceitos do mundo real.

Dentre as vantagens que a OO proporciona, podemos destacar:

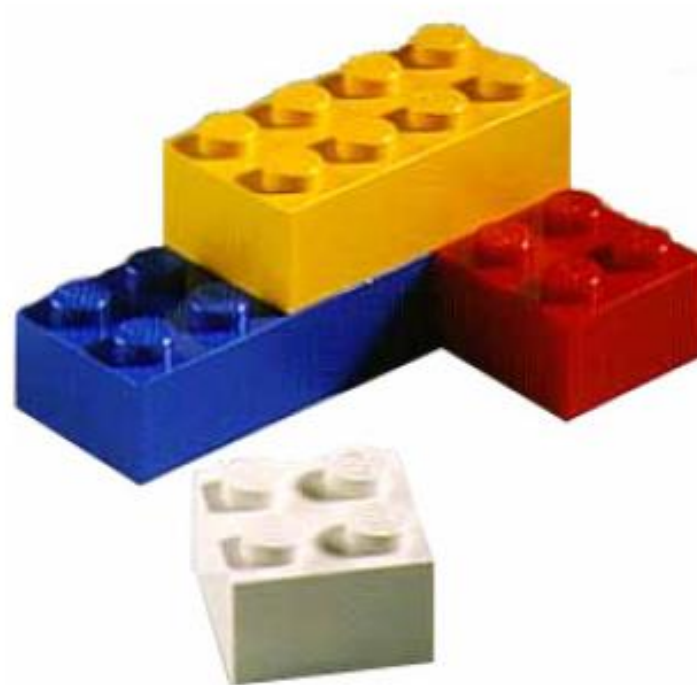
- aumento de produtividade
- reuso de código
- redução das linhas de código programadas
- separação de responsabilidades
- componentização
- maior flexibilidade do sistema
- escalabilidade
- facilidade na manutenção, dentre outras vantagens.

## Orientação a Objetos

---

A OO introduz e enfatiza os seguintes conceitos:

- Objeto
- Mensagem
- Classe
- Abstração
- Encapsulamento
- Herança
- Polimorfismo



# Objeto

---

Objetos são a chave para se compreender a tecnologia orientada a objetos. Você olha ao seu redor e tudo o que vê são objetos: carro, mesa, janela, livro, pessoa, etc.

Os objetos do mundo real têm duas características em comum: **ESTADO** e **COMPORTAMENTO**.

## Estado

O estado de um objeto revela seus dados importantes. Por exemplo, uma pessoa tem: idade, peso, altura, cor de cabelo, cor da pele.

## Comportamento

O comportamento são as ações que aquele objeto pode exercer ou executar. Por exemplo, uma pessoa pode: andar, falar, ouvir, pular.

## Objeto

---

Esses objetos podem ser tanto objetos concretos (carro, livro, nota fiscal), quanto conceitos abstratos (conta corrente, venda, pessoa jurídica).

Na Orientação a Objetos, os objetos do mundo real são modelados e representados no mundo computacional, ou seja, dentro do sistema, por meio de objetos de software.

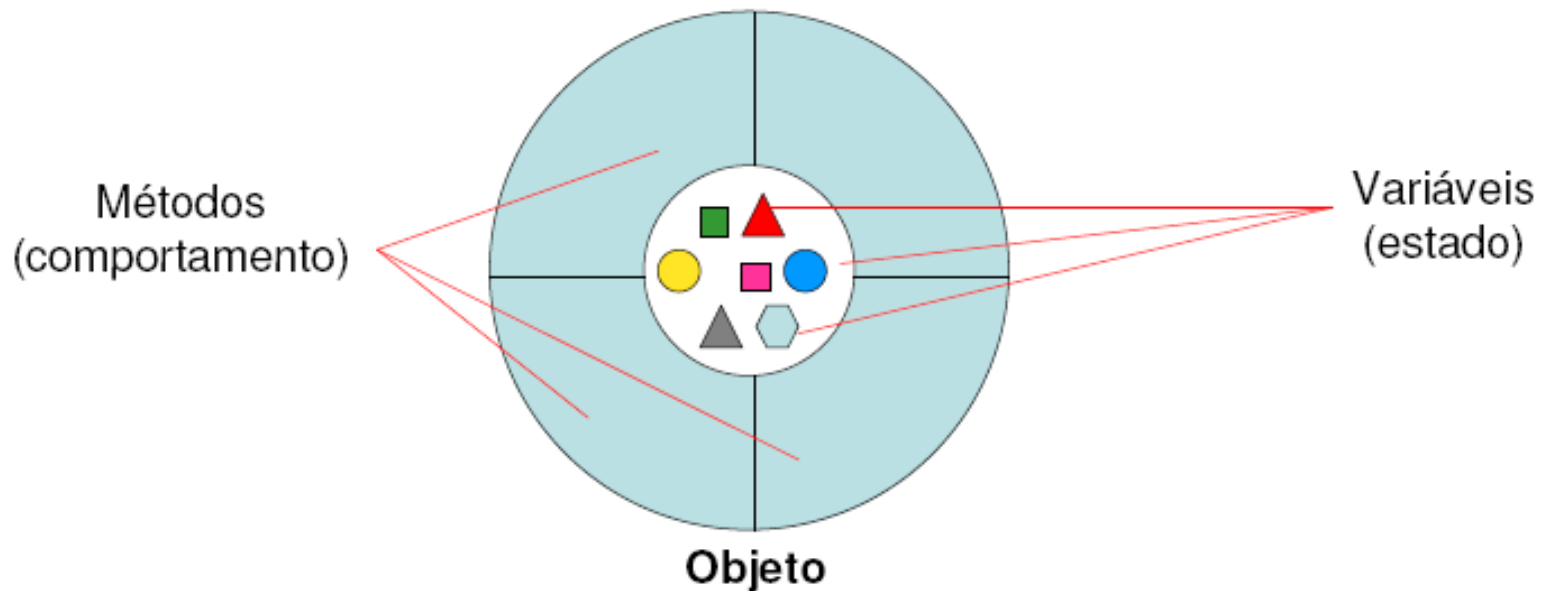
Cada objeto deve ser conhecido, bem definido e ter seu limite e um significado dentro do sistema.

Os objetos de software, assim como os objetos do mundo real, também possuem estado e comportamento.

## Objeto

Um objeto de software mantém seu estado em uma ou mais de suas variáveis. Ele implementa seu comportamento através de seus métodos. Método é o mesmo que função ou procedimento.

Por definição: Um objeto é um pedaço de software que possui variáveis (estado) e métodos (comportamento).

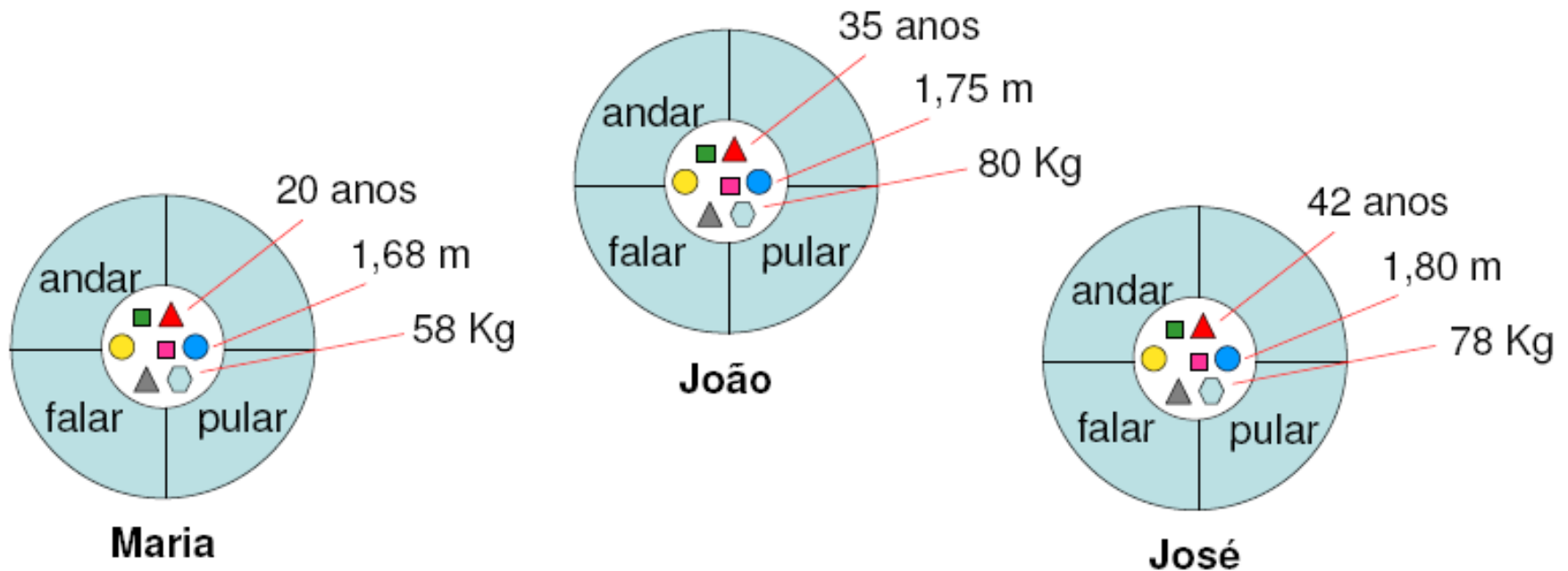




## Objeto

Um sistema pode conter um ou inúmeros objetos ativos. Cada objeto ativo no sistema em particular também é chamado de instância. As diferentes instâncias possuem seu próprio estado.

O exemplo abaixo mostra várias intâncias de pessoas.

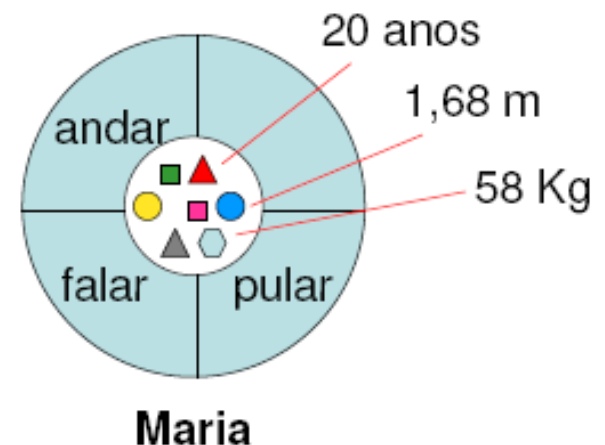


## Objeto

Cada instância de pessoa possui um estado diferente em particular, como visto na última figura.

Porém, cada instância, além do estado, também possui seus métodos (comportamento) que operam sobre o próprio estado. Em outras palavras, para pular, cada pessoa vai fazer uma determinada força dependendo da sua idade, altura e peso, por exemplo.

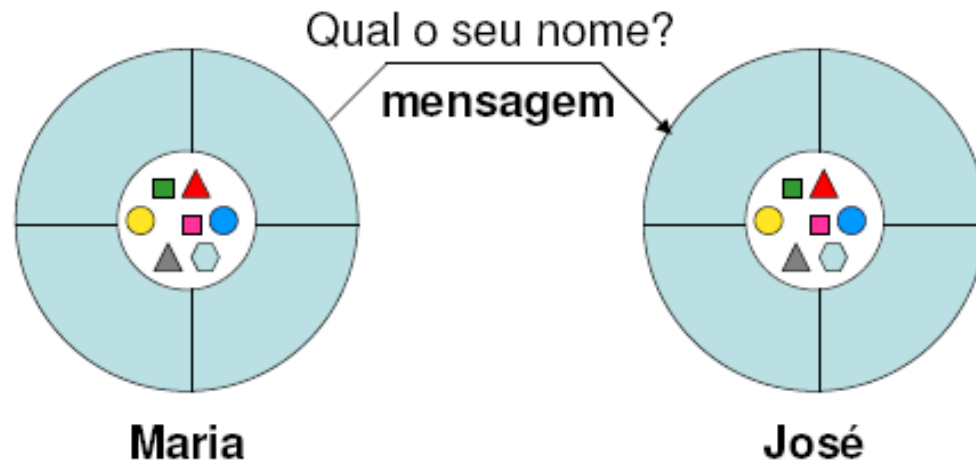
A idéia é que cada objeto seja responsável por seus dados (estado) e seja capaz de realizar as próprias operações que lhe foram atribuídas (comportamento).



## Mensagem

Um objeto por si só não significa muito em um sistema. Para ter algum sentido e valor esses objetos precisam interagir e comunicar-se entre si.

Os objetos se comunicam por meio de mensagens.



Quando um objeto **A** quer se comunicar com um objeto **B** é enviada uma mensagem de **A** para **B**.

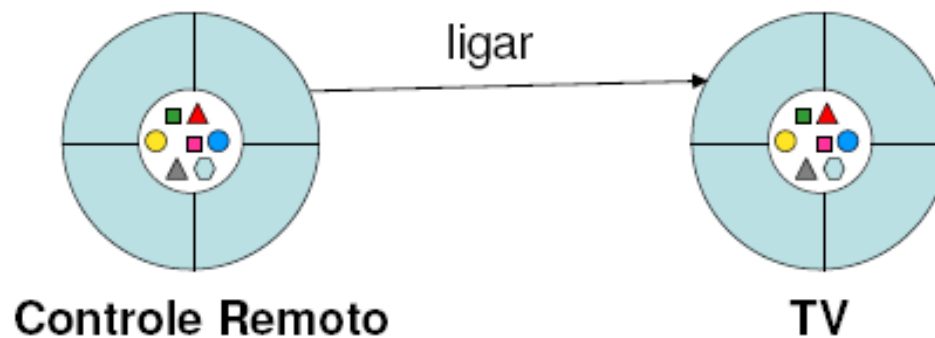
# Mensagem

Enviar uma mensagem significa executar um método.

Então, se **A** envia uma mensagem para **B**, podemos entender como o objeto **A** executando um método do objeto **B**.

As mensagens são compostas por três partes:

- Objeto a quem a mensagem é endereçada
- Nome do método a ser chamado
- Parâmetros que o método recebe



## Classe

---

No mundo real freqüentemente percebemos vários objetos de um mesmo tipo. Por exemplo: seu carro é um dos muitos carros existentes no mundo.

Usando a terminologia OO, dizemos que um carro em particular é uma instância da classe de objetos conhecida como carros.

Os carros, em geral, possuem estado (cor, potência do motor, combustível) e comportamento (ligar, acelerar, frear, mudar marcha) em comum.

O estado de cada carro é independente e pode ser diferente do estado dos outros carros. Cada carro pode ter uma cor diferentes, por exemplo.

A partir dessas semelhanças, os fabricantes de veículos tiram vantagem disso para padronizar a construção de carros de um mesmo tipo, definindo um modelo único com características iguais para todos os carros a serem produzidos.

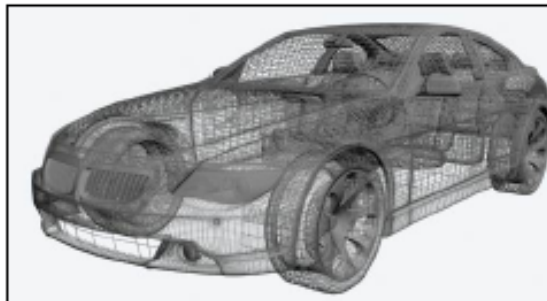
# Classe

---

Na Orientação a Objetos também é possível ter vários objetos do mesmo tipo, que compartilham características em comum.

Tirando vantagem dessa semelhança entre alguns objetos, também é possível criar modelos para esses objetos. Esse modelo é chamado de **CLASSE**. As classes são tipos que podem ser criados.

Por definição: Uma classe é um modelo (protótipo) que define as variáveis (estado) e os métodos (comportamento) comuns a todos os objetos do mesmo tipo.



**Classe**



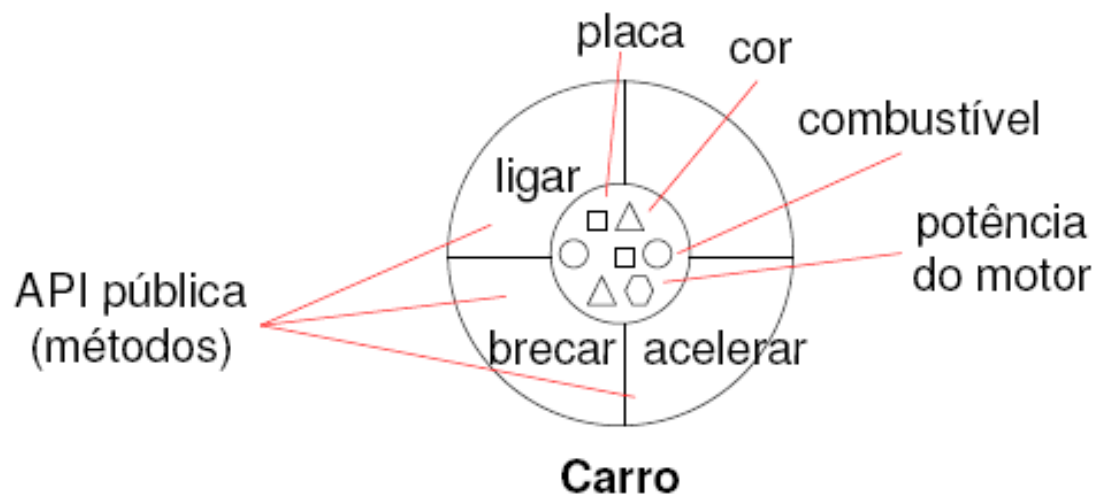
**Objeto**

## Classe

Na classe são definidas as variáveis e implementados os métodos.

Os objetos são criados a partir de suas classes.

A cada objeto criado o sistema aloca memória para o novo objeto e suas variáveis.



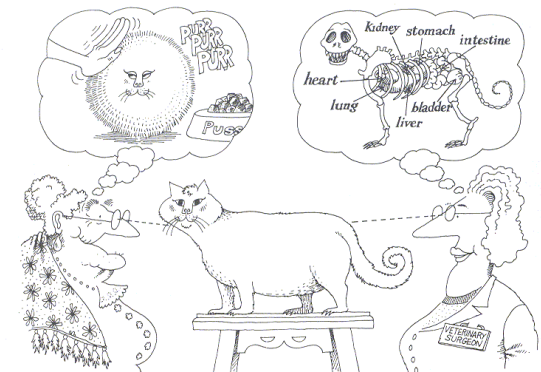
Comumente fazem confusão entre classes e objetos. Lembre-se que classe define as características comuns e os objetos são instâncias dessas classes, com estado próprio.



# Abstração

Abstração é a habilidade e a capacidade de se modelar conceitos, entidades, elementos, problemas e características do mundo real, de um domínio do problema em questão, levando-se em conta apenas os detalhes importantes para a resolução do problema e desprezando coisas que não têm importância no contexto.

Se pensarmos no conceito de “conta corrente” bancária e abstraírmos este conceito, podemos identificar detalhes comuns, como o número da conta, número da agência e saldo; e operações como débito em conta, depósito e extrato da conta. Basicamente essas são as características de conta corrente para todos os bancos, apesar de um ou outro banco ter uma política de descontos de taxas etc.



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.



# Encapsulamento

---

Na OO, **encapsulamento** é o mecanismo utilizado para disponibilizar métodos que operam sobre os dados e que protegem o acesso direto indevido aos atributos de uma instância fora da classe onde estes foram declarados.

Esta proteção consiste em se usar modificadores de acesso mais restritivos sobre os atributos definidos na classe e fornecendo métodos que alteram os valores destes atributos de alguma forma.

O encapsulamento ajuda a prevenir o problema de interferência externa indevida sobre os dados de um objeto, como objetos que possam alterar os dados de outros objetos indevidamente.

## Encapsulamento

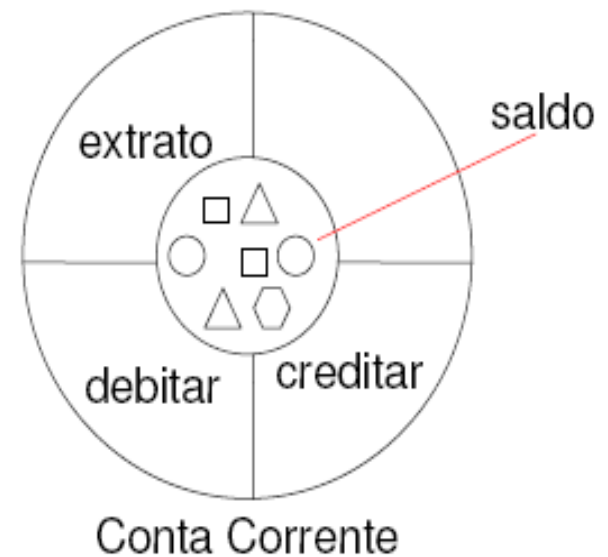
Um exemplo deste problema pode ser o saldo da conta bancária.

O saldo certamente não pode ser alterado ou manipulado diretamente, mas sim através de métodos adequados para isso, como métodos que fazem lançamentos de débitos e créditos.

A alteração direta do saldo causaria um problema de cálculos e inconsistência de dados.

Justamente por isso devemos criar classes bem encapsuladas, que fornencem métodos adequados para operar sobre os dados dos objetos daquela classe.

O uso de encapsulamento também evita que um programa torne-se tão interdependente que uma pequena mudança tenha grandes efeitos colaterais.



# Herança

---

Herança é um mecanismo da OO que permite criar novas classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser estendida.

Este mecanismo é muito interessante pois promove um grande reuso e reaproveitamento de código existente.

Com a herança é possível criar classes derivadas (subclasses) a partir de classes bases (superclasses). As subclasses são mais especializadas do que as suas superclasses, mais genéricas.

As subclasses herdam todas as características de suas superclasses, como suas variáveis e métodos.

## Herança

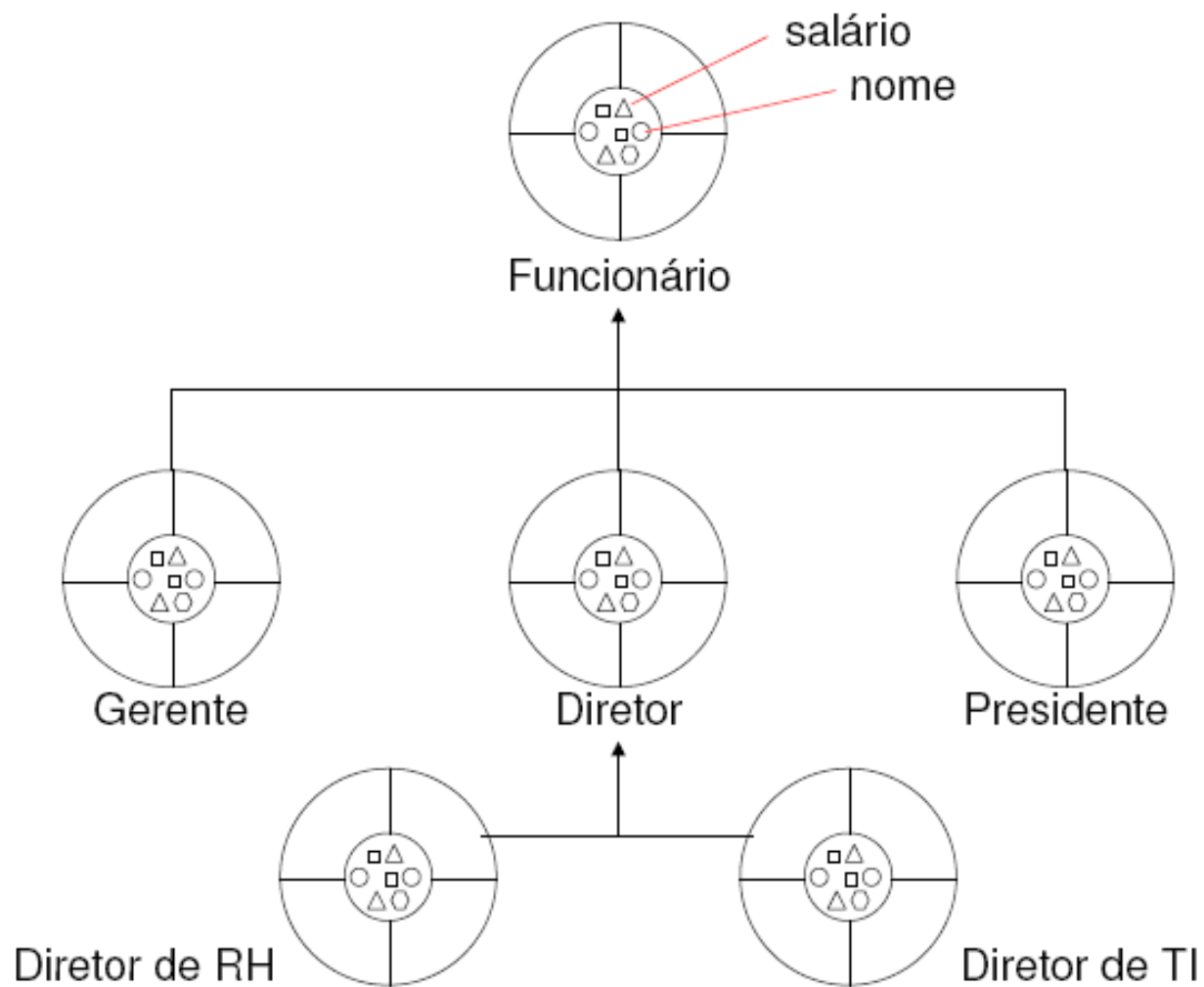
---

Imagine que dentro de uma organização empresarial, o sistema de RH tenha que trabalhar com os diferentes níveis hierárquicos da empresa, desde o funcionário de baixo escalão até o seu presidente.

Todos são funcionários da empresa, porém cada um com um cargo diferente. Mesmo a secretária, o pessoal da limpeza, o diretor e o presidente possuem um número de identificação, além de salário e outras características em comum.

Essas características em comum podem ser reunidas em um tipo de classe em comum, e cada nível da hierarquia ser tratado como um novo tipo, mas aproveitando-se dos tipos já criados, através da herança.

# Herança



## Herança

---

Os subtipos, além de herdarem todas as características de seus supertipos, também podem adicionar mais características, seja na forma de variáveis e/ou métodos adicionais, bem como reescrever métodos já existentes na superclasse (polimorfismo).

A herança permite vários níveis na hierarquia de classes, podendo criar tantos subtipos quanto necessário, até se chegar no nível de especialização desejado.

Podemos tratar subtipos como se fossem seus supertipos, por exemplo o sistema de RH pode tratar uma instância de Presidente como se fosse um objeto do tipo Funcionário, em determinada funcionalidade.

Porém não é possível tratar um supertipo como se fosse um subtipo, a não ser que o objeto em questão seja realmente do subtipo desejado e a linguagem suporte este tipo de tratamento, seja por meio de conversão de tipos ou outro mecanismo.

## Herança

---

Algumas linguagens de programação permitem herança múltipla, ou seja, uma classe pode estender características de várias classes ao mesmo tempo. É o caso do C++.

Outras linguagens não permitem herança múltipla, por se tratar de algo perigo se não usada corretamente. É o caso do Java.

Na Orientação a Objetos as palavras classe base, supertipo, superclasse, classe pai e classe mãe são sinônimos, bem como as palavras classe derivada, subtipo, subclasse e classe filha também são sinônimos.



## Polimorfismo

---

Formalmente polimorfismo quer dizer “várias formas”.

No caso da OO, polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem, dependendo do seu tipo de criação.

O poliformismo é alcançado com auxílio do uso de herança nas classes e a reescrita (*overriding*) de métodos das superclasses nas suas subclasses.

Duas subclasses de uma mesma classe podem ter implementações completamente diferentes de um mesmo método, o que leva os objetos a se comportarem de forma diferente, dependendo do seu tipo (classe).



## Polimorfismo

---

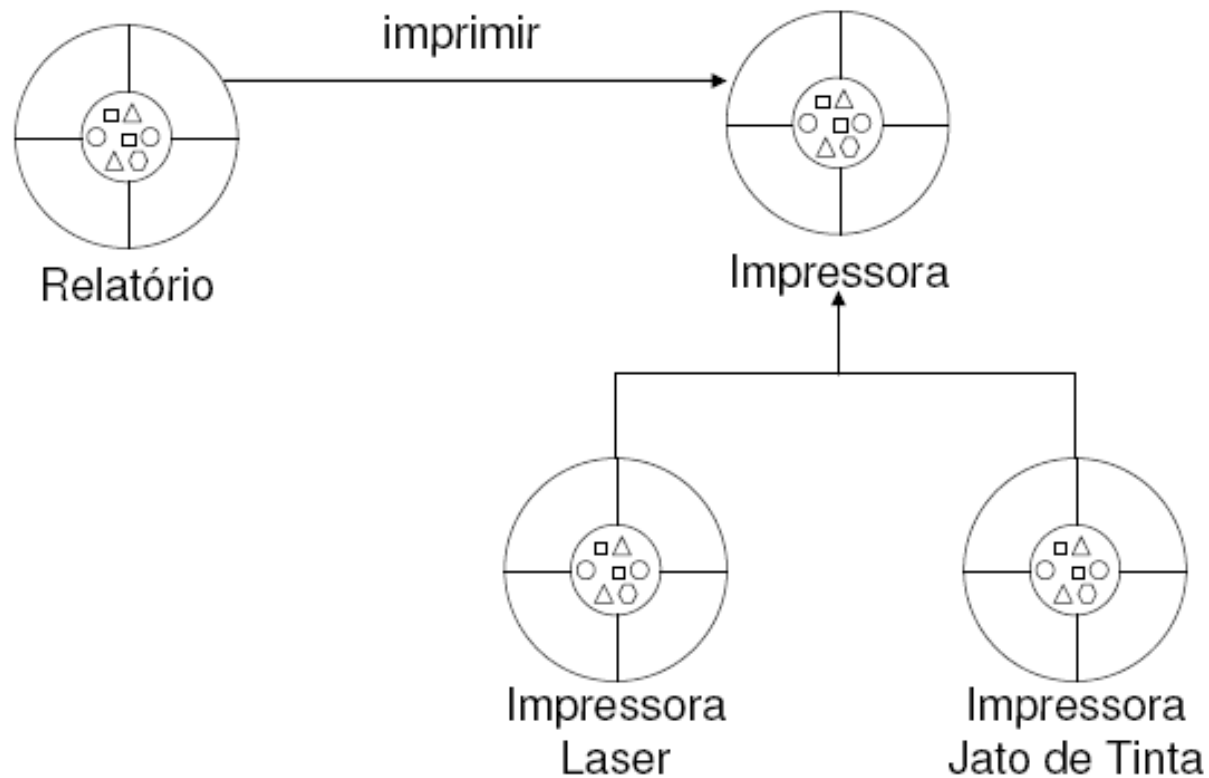
Exemplificando:

Podemos imaginar um programa que faça a impressão de um relatório, por meio de uma classe chamada Impressora, que é uma interface de acesso às funcionalidades da impressora usada, por meio de um driver fornecido pelo fabricante.

Uma impressora a laser tem um mecanismo de impressão totalmente diferente de uma impressora a jato de tinta, mas isso não importa para o programa.

Ele manda uma simples mensagem de imprimir para a impressora, e o modo como a impressora imprime no papel varia de acordo com o tipo de impressora usada, ou seja, a impressão se dá de formas diferentes para a mesma mensagem de imprimir.

# Polimorfismo



# Polimorfismo

---

Algumas linguagens promovem o polimorfismo principalmente através do uso de classes abstratas e interfaces, como é o caso da tecnologia Java.

**Classes abstratas** são classes que não podem gerar instâncias de objetos e que possuem um ou mais métodos sem implementação, deixando para suas subclasses a tarefa de implementar estes métodos abstratos.

**Interfaces** são um tipo de contrato que algumas classes têm de seguir, ou seja, as interfaces apenas definem métodos abstratos que as classes que implementam esta interface têm de implementar.

## Agregação

---

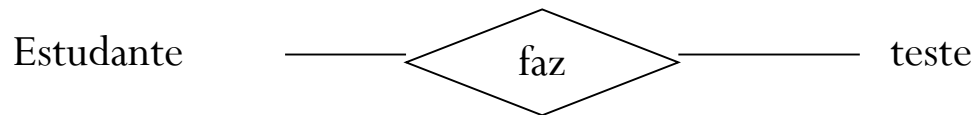
É um mecanismo que permite a construção de uma classe agregada a partir de outras classes componentes. Ex. Casa

## Associação

---

Associação – é usada para agrupar certos objetos que ocorrem em algum ponto no tempo ou sob circunstâncias similares. Na AOO , a associação é modelada através de uma conexão de ocorrências. Uma conexão de ocorrência é um relacionamento que um objeto precisa ter com outro(s) objeto (s), para cumprir suas responsabilidades.

Ex:



## Conclusão

---

O paradigma da Orientação a Objetos traz um ganho significativo na qualidade da produção de software, porém grandes benefícios são alcançados quando as técnicas de programação OO são colocadas em prática com o uso de uma tecnologia que nos permita usar todas as características da OO; além de agregar à programação o uso de boas práticas de programação e padrões de projeto (*design patterns*).

Esse é um dos motivos do sucesso da tecnologia Java, que suporta a OO completamente e também fornece mecanismos para se usar os *design patterns* conhecidos.

Além do conhecimento da Orientação a Objetos, o conhecimento da UML (*Unified Modelling Language*) ajuda muito no desenho e planejamento de sistemas na sua concepção.

# UML

## Unified Modeling Language

## Unified Modeling Language (UML)

---

UML é uma linguagem de modelagem de sistemas, usada para:

- especificar,
- modelar,
- visualizar
- e documentar

os modelos e artefatos de sistemas OO e não-OO, baseando-se em diagramas. A UML pode ser usada com todos os processos durante o ciclo de desenvolvimento do projeto (análise de requisitos, análise de sistema, design, programação e testes).

A UML foi criada por Grady Booch, Jim Rumbaugh e Ivar Jacobson, em 1997, e hoje é mantida pela OMG (Object Management Group).

## Fases do Desenvolvimento

---

**Análise de Requisitos:** Fase que captura as intenções e necessidades dos usuários do sistema, através das funções desejadas no sistema, chamadas de Casos de Uso.

**Análise:** Onde se cria as primeiras abstrações e mecanismos presentes no domínio do problema.

**Design (Projeto):** O resultado da análise é expandido em soluções técnicas. As classes do domínio do problema são mescladas com classes de infra-estrutura. É o detalhamento para a fase de programação.

**Programação:** Os modelos criados são convertidos em códigos de linguagem.

**Testes:** Testes unitários, testes de integração e testes de aceitação.



## Por que usar UML?

---

Desenvolver o modelo de uma aplicação antes de construí-la, é tão essencial quanto ter uma planta para a construção de uma casa.

Bons modelos são essenciais para a comunicação entre os times de projetos e para assegurar a beleza arquitetural.

Com o aumento da complexidade dos sistemas, é importância conhecer boas técnicas de modelagem.

Ter um rigoroso padrão de linguagem de modelagem é um fator essencial para o sucesso de um projeto.

Como a UML se tornou uma notação padrão da indústria de arquitetura de software, ela é assunto abordado em muitos livros, seminários e sites.

## Notação da UML

---

A UML é dividida em algumas partes, como segue:

**Visões:** Mostram os diferentes aspectos do sistema, dando enfoque a ângulos e níveis de abstrações diferentes, construindo uma visão completa do sistema a ser construído.

**Modelos de Elementos:** São os conceitos utilizados nos diagramas. Representam definições comuns da OO.

**Mecanismos Gerais:** Provém comentários suplementares, informações ou semântica sobre os elementos dos modelos.

**Diagramas:** São gráficos que descrevem o conteúdo em uma visão. A UML possui vários tipos de diagramas que, combinados, formam todas as visões do sistema.

## Visões

---

Os sistemas são, geralmente, compostos por diferentes níveis de visões.

Cada visão é descrita por um número de diagramas que contém informações que dão ênfase aos aspectos particulares do sistema.

Os diagramas podem fazer parte de mais de uma visão do sistema.

As visões do sistema são:

- Visão de Casos de Uso
- Visão de Componentes
- Visão Lógica
- Visão de Organização
- Visão de Concorrência

## Visões

---

- **Visão de Casos de Uso:** Descreve as funcionalidades do sistema desempenhada pelos atores externos. É a visão central, base para as outras visões do sistema. Diagramas de Casos de Uso e eventualmente Atividades.
- **Visão de Componentes:** Descreve a implementação dos módulos e suas dependências. Consiste nos componentes dos diagramas.
- **Visão Lógica:** Descreve como as funcionalidades do sistema serão implementadas. Especifica a estrutura estática e dinâmica. Diagramas de Classe e Objetos e Diagramas de Estado, Seqüência, Colaboração e Atividades.
- **Visão de Organização:** Mostra a organização física do sistema, os computadores, periféricos etc e como eles se conectam entre si. Diagrama de Execução.
- **Visão de Concorrência:** Trata da divisão do sistema em processos e processadores. Diagramas de Estado, Seqüência, Colaboração e Atividade.

## Modelos de Elementos da UML

---

Os elementos da UML são blocos de construção para os modelos dos diagramas e são essenciais para o entendimento da UML.

Cada elemento tem um propósito diferente, diferentes regras e notações.

Alguns elementos podem ser usados em diferentes diagramas.

Existem um grande conjunto de elementos disponíveis na especificação.

Vamos ver alguns destes elementos especificados pela UML.

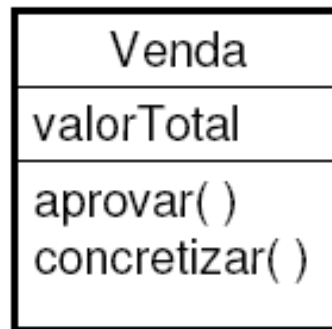
## Modelos de Elementos da UML

---

### Classe

É a descrição de conjunto de objetos que compartilham os mesmos atributos e relacionamentos (estado), operações e semântica (comportamento).

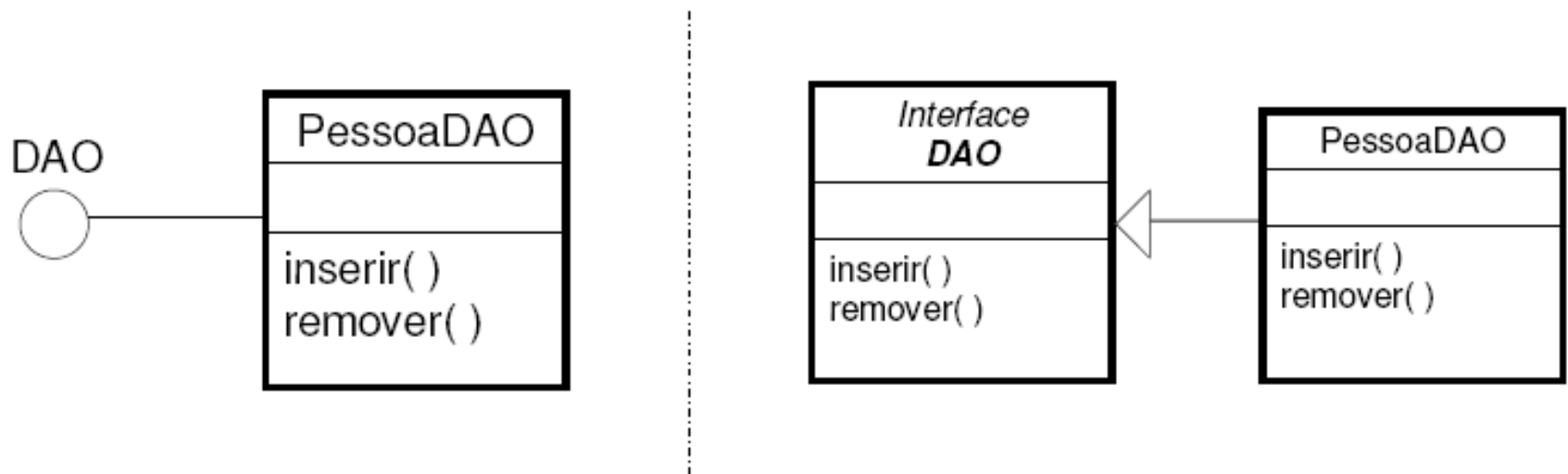
As classes podem implementar uma ou mais interfaces e, graficamente, são representadas por retângulos, com três divisões: Nome da Classe, Conjunto de Atributos e Conjunto de Métodos.



# Modelos de Elementos da UML

## Interface

É um elemento que define uma coleção de operações que especificam serviços de uma classe ou componente. Uma interface representa todo o comportamento externamente visível do elemento. Pode representar todo o comportamento, ou apenas parte dele. A interface define um conjunto de especificações de operações, mas nunca de um conjunto de implementações de operações. É representada graficamente por um círculo e o respectivo nome. Uma interface raramente aparece sozinha.



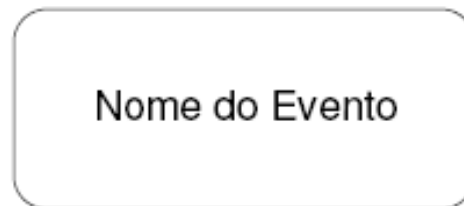
## Modelos de Elementos da UML

---

### Estado

Todo os objetos possuem um estado, que é o resultado das atividades executadas por ele.

O estado é representado por um retângulo com os cantos arredondados e o nome do evento dentro do desenho.

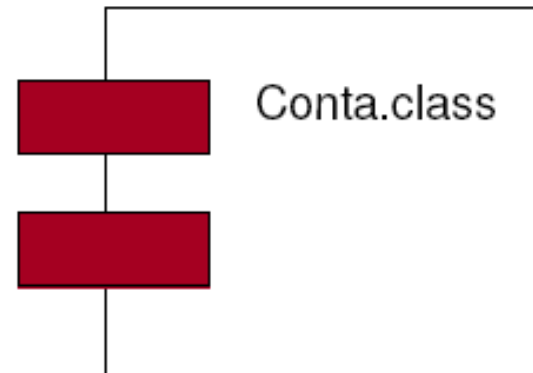
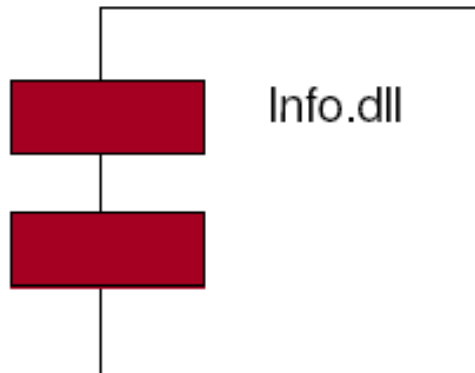




## Modelos de Elementos da UML

### Componente

É a parte modular de um sistema, cujo comportamento é definido pelas suas interfaces. O trabalho interno dos componentes deve ser invisível, e o seu uso ser independente de plataforma. Geralmente códigos-fonte, DLLs, Java Beans e outros artefatos são considerados componentes. Graficamente é representado por um retângulo com duas abas na lateral esquerda.

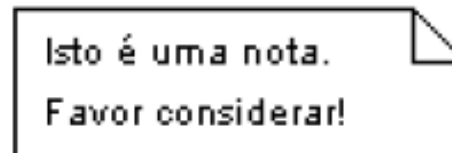


## Modelos de Elementos da UML

---

### Nota

A nota é apenas um símbolo para representar restrições e comentários anexados a um elemento. Geralmente usa-se a nota para aprimorar os diagramas. Gráficamente é representada por um retângulo com um dos cantos com uma dobra na página.



# Modelos de Elementos da UML - Relacionamentos

---

## Relacionamentos

Os elementos dos modelos UML estão ligados uns aos outros, especificando o que cada elemento significa ao outro e qual o grau de ligação deles, ou seja, qual a relação lógica entre os elementos.

A estas ligações, damos o nome de relacionamento.

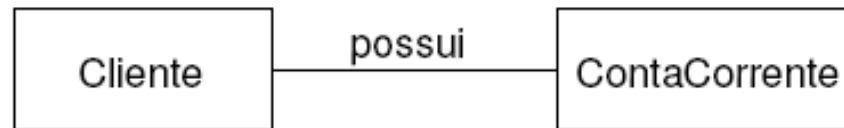
Existem diferentes tipos e graus de relacionamentos. São eles:

- Associação
- Generalização
- Dependência
- Refinamento

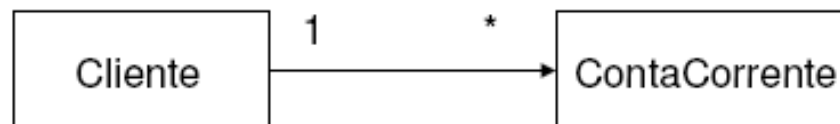
# Modelos de Elementos da UML - Relacionamentos

## Associações

A associação representa uma ligação entre dois elementos. Geralmente são expressas como uma linha sólida, de um elemento ao outro, e com um verbo (ou substantivo) que qualifique a associação.



As associações ainda podem expressar a cardinalidade e a navegação (sentido) da associação. A cardinalidade (ou multiplicidade) indica quantos elementos são possíveis de cada lado da associação e pode ser expressada como um número ou um intervalo.

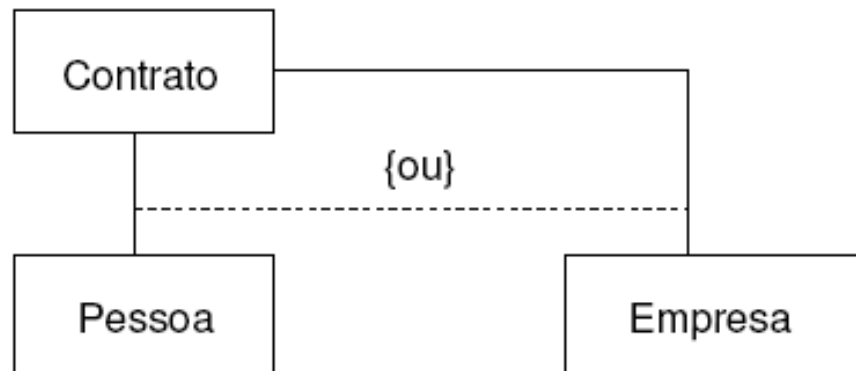


## Modelos de Elementos da UML - Relacionamentos

**Associação Recursiva:** Acontece quando um elemento se conecta a ele mesmo, e associação tem alguma semântica no modelo.

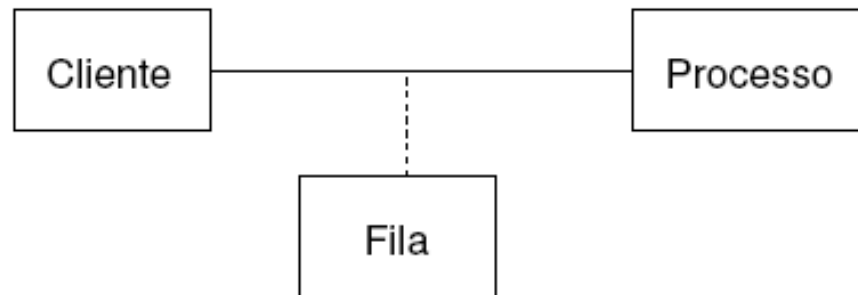


**Associação Exclusiva:** Quando algumas combinações de associações não são possíveis no domínio do problema. É uma restrição entre duas ou mais associações.

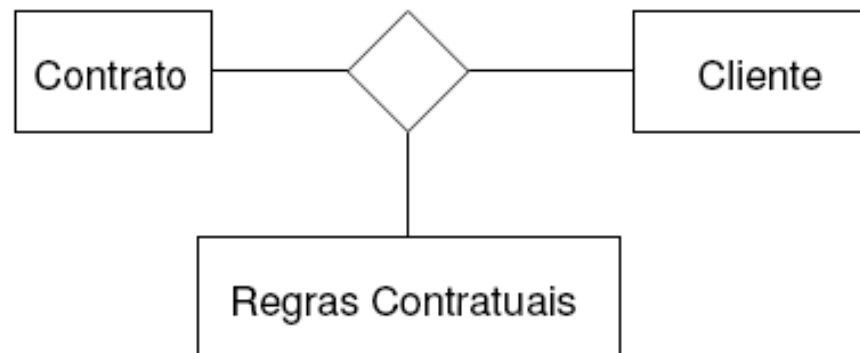


## Modelos de Elementos da UML - Relacionamentos

**Associação de Classe:** Uma classe pode ser associada a uma associação. Serve para adicionar informações extras à associação existente.

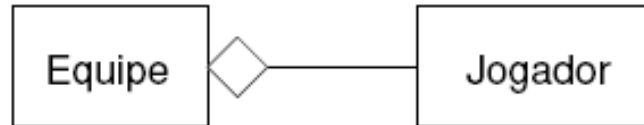


**Associação Ternária:** Usada quando mais de duas classes podem se associar entre si.

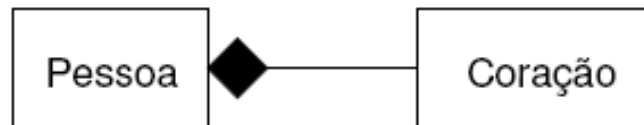


## Modelos de Elementos da UML - Relacionamentos

**Agregação:** Este é um caso particular de associação. Indica que um elemento é parte ou está contida em outra classe. Representa uma relação do tipo parte/todo.



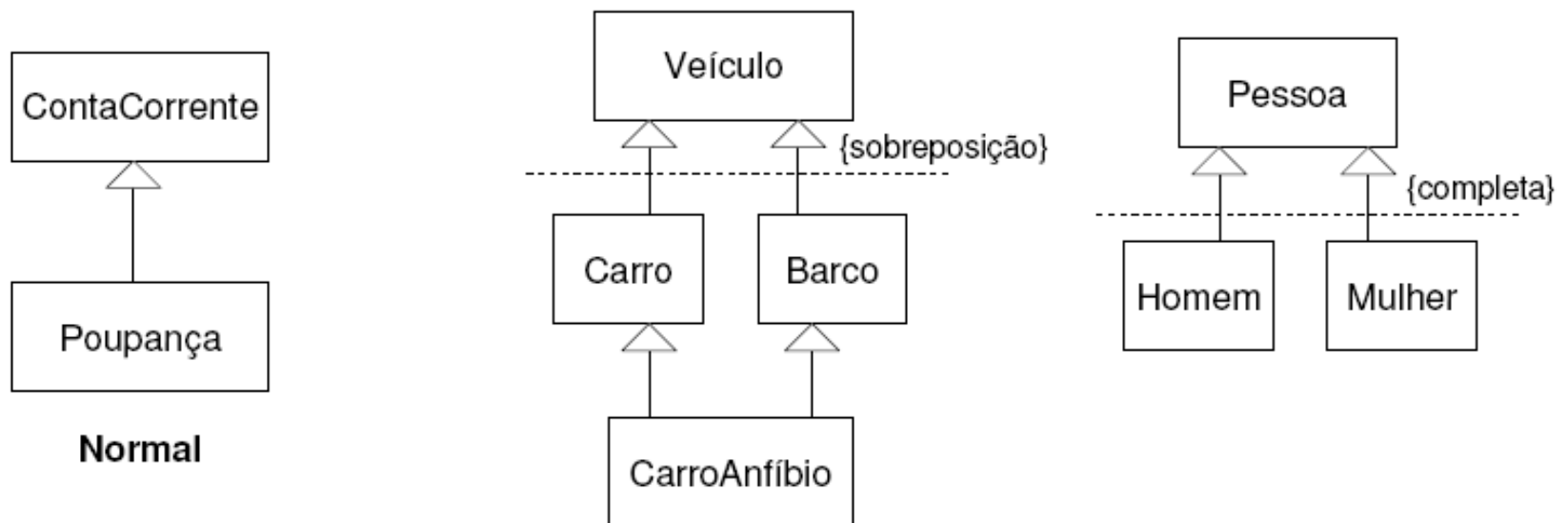
**Agregação de Composição:** É uma relação onde um elemento está contido em outro, ou seja a vida de um depende do outro, e o seus tempos de vida são os mesmos.



## Modelos de Elementos da UML - Relacionamentos

### Generalizações

A generalização é um relacionamento entre um elemento mais geral e um mais específico. O elemento mais específico possui todas as características do seu elemento mais geral, como as propriedades e seu comportamento, além de poder adicionar mais características a ele mesmo. As generalizações podem ser normal e restrita. As restritas se dividem em sobreposição, disjuntiva, completa e incompleta.

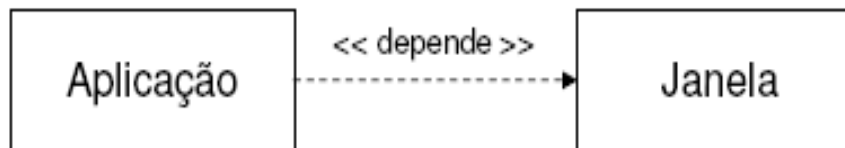




# Modelos de Elementos da UML - Relacionamentos

## Dependência

A dependência é uma conexão semântica entre dois elementos, um independente e outro dependente. Qualquer alteração no elemento independente pode afetar o elemento dependente. Em classes, a dependência indica que o elemento apenas instancia e/ou usa o elemento independente, sem manter uma relação duradoura com o elemento.



## Diagramas da UML

---

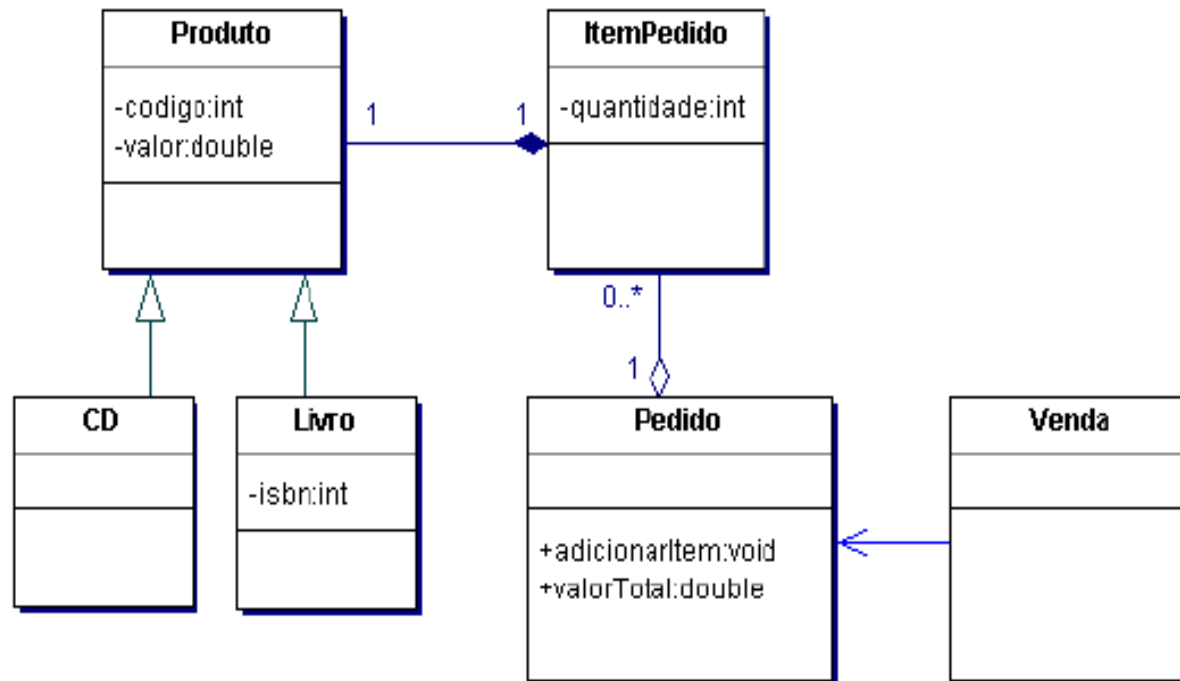
Com a UML é possível modelar os projetos de sistemas, baseados em diagramas. Os diagramas são divididos, basicamente, em:

- **Diagramas Estruturais:** diagrama de classes, diagrama de objetos, diagrama de componentes e diagrama de disponibilização.
- **Diagramas de Comportamento:** diagrama de casos de uso, diagrama de seqüência, diagrama de atividades, diagrama de colaboração e diagrama de estados.
- **Diagramas de Gerenciamento do Modelo:** pacotes, subsistemas e modelos.

# Diagrama de Classes

## Diagrama de Classes

O Diagrama de Classes mostra a estrutura estática do modelo da aplicação. Este diagrama exhibe as classes do sistema e o grau do relacionamentos entre elas.



O diagrama de classes representa a estrutura do sistema, recorrendo ao conceito de classe e suas relações. O modelo de classes resulta de um processo de abstração onde são identificados os objetos relevantes do sistema em estudo.

Um objeto é uma ocorrência que tem interesse para o sistema em estudo e que se pretende descrever no seu ambiente, contendo identidade e comportamento.

O comportamento de um objeto define o modo como ele age e reage a estímulos externos e a identidade de um objeto é um atributo que o distingue de todos os demais, sendo preservada quando o seu estado muda. Um objeto não é mais do que uma instância da classe.

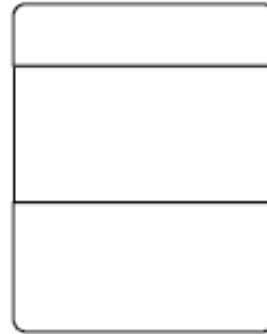
Os objetos de modelação contemplados por este diagrama são:

Classe: é a representação de um conjunto de objetos que partilham os mesmos atributos e comportamentos;

Relação: representa a ligação entre classes.

A simbologia usada para representar estes dois conceitos é:

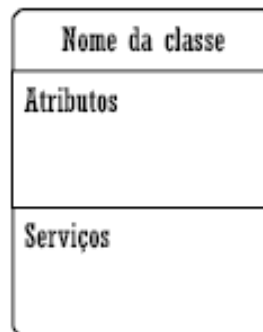
**Classe**



**Relação**



Cada classe é descrita através do seu nome, identificação de todos os seus atributos e identificação de todas as operações que traduzem o seu comportamento. O símbolo utilizado para representar a classe, contempla três áreas distintas, uma área para se identificar o nome da classe, outra para listar os atributos e, finalmente, a terceira, para listar as operações ou métodos, também designados, segundo alguns autores, por serviços.



## **Atributo**

Um atributo representa uma propriedade que todos os objetos da classe têm (por exemplo, todos os cachorros tem pelo , orelhas , altura , etc.

Mas cada objeto terá valores particulares para seus atributos (alguns cachorros são mais baixos , outros são maiores, etc.).

Uma classe pode ter qualquer número de atributos. Na UML, o nome de um atributo é um texto contendo letras e dígitos e algumas marcas de pontuação. UML sugere de capitalizar todas as primeiras letras de cada palavra no nome menos a primeira palavra (ex.: "nome", "nomeCachorro"). Num modelo, os atributos devem ser de um tipo simples (inteiro, texto, talvez data), não podem conter outros objetos.

## **Métodos**

Métodos são ações que implementam uma operação. Uma classe pode ter qualquer número de métodos e dois métodos em duas classes podem ter o mesmo nome.

Os diagrama se classes ilustram atributos e operações de uma classe e as restrições como que os objetos podem ser conectados; descrevem também os tipos de objetos no sistema e os relacionamentos entre estes objetos que podem ser : **associações e abstrações**.

Para poder representar a visibilidade dos atributos e operações em uma classe utiliza-se as seguintes marcas e significados:

- + público - visível em qualquer classe**
- # protegido - qualquer descendente pode usar**
- privado - visível somente dentro da classe**



## **Regras de Construção**

### **Como utilizar os objetos**

O diagrama de classes mostra como cada classe se relaciona com as outras, tendo como objetivo, a satisfação dos requisitos funcionais definidos para o sistema em estudo. Neste diagrama, uma classe só pode aparecer uma vez.

### **Como atribuir nomes aos objetos**

Qualquer classe e relação devem ter um nome elucidativo e claro para que o diagrama seja facilmente entendido. As classes devem ser identificadas por um nome comum, como, por exemplo, Encomenda, Produto, Cliente, etc. Os nomes das relações devem ser traduzidas através de um verbo, como, por exemplo, efetua, contém, refere, etc. Qualquer um destes nomes deve fazer parte do vocabulário do domínio do problema em estudo.

## Relacionamento entre classes

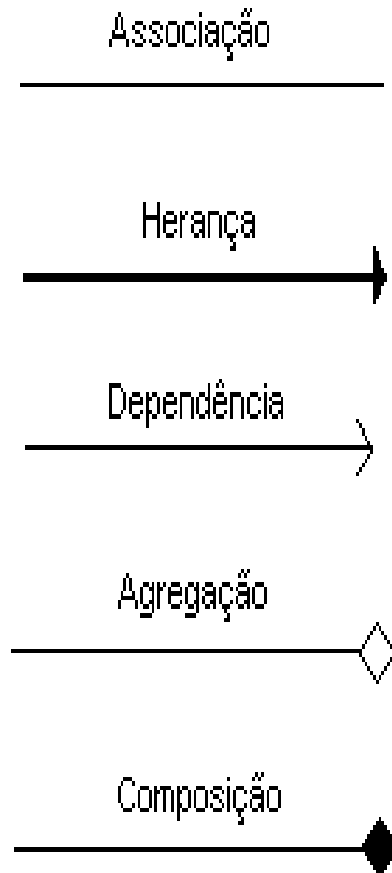
Os objetos possuem relações entre eles: um professor *ministra* uma disciplina *para* alunos *numa* sala, um cliente *faz* uma reserva *de* alguns lugares *para* uma data, etc. Essas relações são representadas também no diagrama de classe.

A UML reconhece três tipos mais importantes de relações: dependência, associação e generalização (ou herança).

Geralmente as classes não estão sós e se relacionam entre si. O relacionamento e a comunicação entre as classes definem responsabilidades, temos 3 tipos :

1. **Associações : Agregação e composição**
2. **Generalização (herança)**
3. **Dependências**

As representações usam a seguinte notação:



- **Associação:** São relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes. Podemos ter associação uniária, 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. [Nicolas Anquetil]

- **Dependência** - 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.

• **Generalização** (herança: simples ou composta) - Relacionamento entre um elemento mais geral e um mais específico. Onde o elemento mais específico herda as propriedades e métodos do elemento mais geral. A relação de generalização também é conhecida como herança no modelo a objetos. Como a relação de dependência, ela existe só entre as classes. Um objeto particular não é um caso geral de um outro objeto, só conceitos (classes no modelo a objetos) são generalização de outros conceitos.

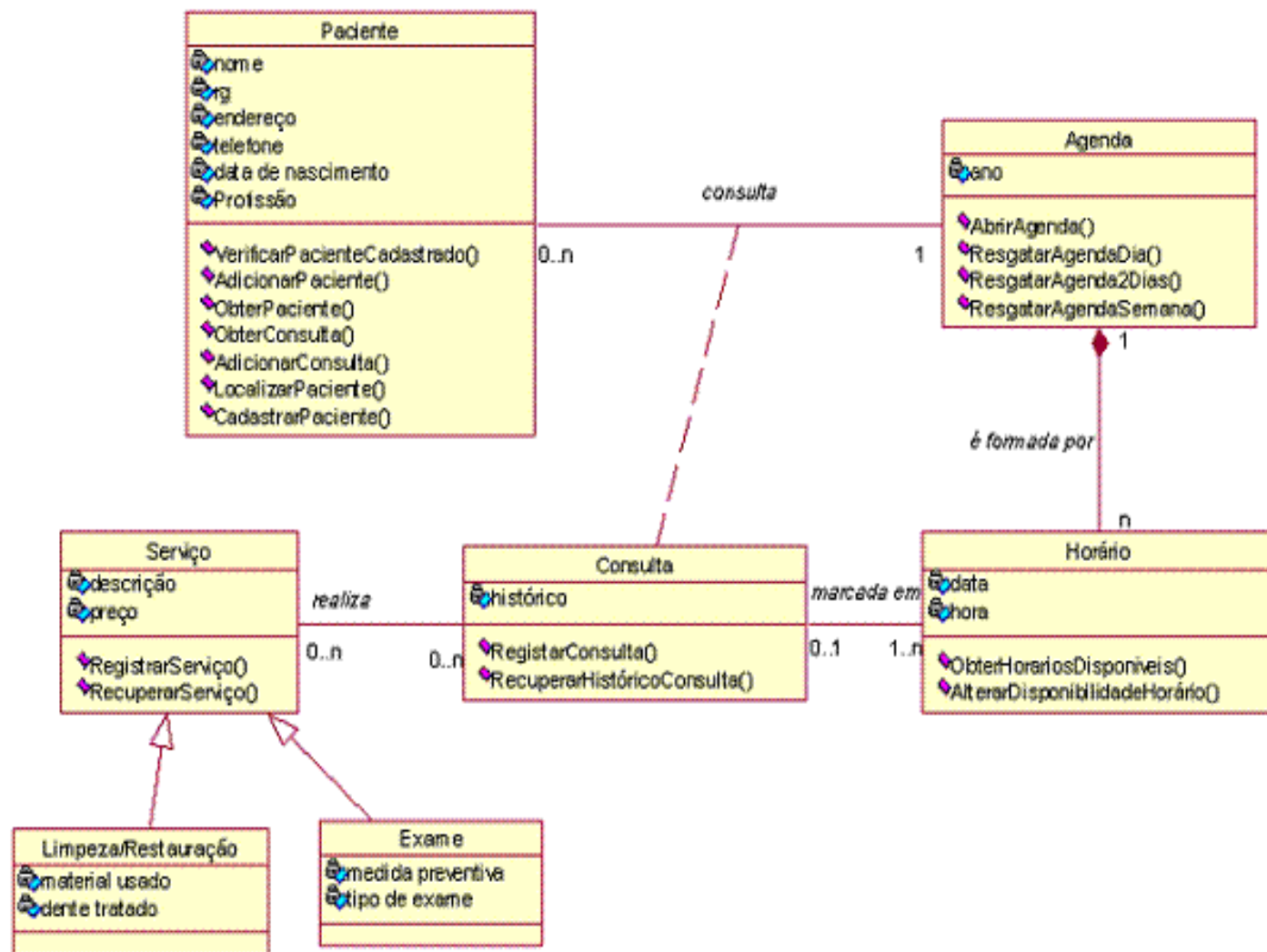
• **Agregação Regular** - tipo de associação ( é parte de , todo/parte) onde o objeto parte é um atributo do todo ; onde os objetos partes somente são criados se o todo ao qual estão agregados seja criado. Pedidos é composto por itens de pedidos.

• **Composição** - Relacionamento entre um elemento ( o todo) e outros elementos (as partes) onde as parte só podem pertencer ao todo e são criadas e destruídas com ele.

O diagrama de classes lista todos os conceitos do domínio que serão implementados no sistema e as relações entre os conceitos. Ele é muito importante, pois define a estrutura do sistema a desenvolver.

O diagrama de classes não surge do nada ele é consequência do prévio levantamento de requisitos, definição de casos de usos e classes. Como exemplo, vamos supor que você tivesse que desenvolver um sistema para automatizar um consultório dentário. As etapas básicas envolvidas seriam:

- Levantamento e análise de requisitos do sistema a ser desenvolvido. Entrevista com o dentista(s) e com as pessoas que trabalham no consultório
- Definição dos objetos do sistema: Paciente, agenda, dentista, serviço, contrato, consulta, pagamento, etc..
- Definição dos atores do sistema: paciente, dentista, secretária
- Definição e detalhamento dos casos de uso: marcar consulta confirmar consulta, cadastrar paciente, cadastrar serviços, etc.
- Definição das classes: paciente, dentista, exame, agenda, serviço
- Definir os atributos e métodos das classes:



## Exercícios

---

### Encontrando Classe, Atributos e Métodos

#### **Cenário 1 : Boneco em Movimento**

Professora Filomena decidiu criar uma classe que permita mover um boneco na tela. Esse boneco deve ter nome, posição da coordenada X, posição da coordenada Y e direção atual (cima, baixo, direita, esquerda).

- Identifique as classes, atributos e métodos desse cenário.

#### **Cenário 2 : Horário De Remédios**

As informações a seguir se referem a uma aplicação de controle pessoal de horário de remédios, existente no Palm de Maurício.

Para cada remédio cadastra-se: o nome de quem vai tomar o remédio, a data de início, a quantidade de dias que foi prescrita pelo médico, a quantidade de vezes ao dia, a dosagem e o nome do remédio.

Ao cadastrar o remédio, a aplicação sugere todos os horários possíveis para tomar o remédio. O usuário escolhe o melhor horário e a aplicação avisa até quando o remédio deve ser tomado e prepara uma planilha de horários.

O usuário, no início do dia, seleciona a opção de planilha de horários de remédios do dia.

No caso de Maurício atrasar o horário de tomar o remédio num determinado dia, a planilha reorganiza os horários daquele dia.

- Identifique as classes, atributos e métodos desse cenário.

## Exercícios

### Diagrama de Classes

#### Cenário 1 : Classificados na Web

Dalila está oferecendo aos amigos da escola, do bairro e do clube, o serviço de classificados pela Web.

Quem anuncia paga R\$ 2,0 pelo anúncio simples de 20 palavras ou R\$ 5,0 pelo anúncio destaque de página, que pode conter até 50 palavras e uma imagem. O anúncio fica disponível por quinze dias. Não entram na contagem: o valor do produto, o texto título, o nome do contato, até dois telefones de contato e uma observação sobre os telefones de até cinco palavras (por exemplo: de 18h às 20h).

Quem se toma assinante do classificado virtual recebe diariamente em seu e-mail o resumo de ofertas do site. O cliente pode cadastrar seções de interesse, a fim de receber os novos anúncios pelo e-mail, sem precisar visitar o site.

O anúncio segue o formato desse exemplo:

**Seção Computador**

<b>Pentium 4 3.0 Ghz</b>	<b>R\$ 1.300,00</b>
Computador Pentium 4.0 3.0 Ghz, 256 Mb Memória, HD 40 Gb, gravador de CD, monitor Samsung 17".	
Tratar Marta	2222-1111 / 2111-2222 à noite
inserido em: 01/08/2005	

- *Desenhe o Diagrama de classes*



## Exercícios

---

### Diagrama de Classes

#### Cenário 2 : Cursos de Aperfeiçoamento

A empresa APRF oferece cursos diversos de aperfeiçoamento profissional.

O dono precisa controlar os cursos oferecidos, a data de início e término, o horário de início e término, o nome e o telefone celular do professor e o valor da sua hora/aula.

Para a secretaria é preciso ter o controle da matrícula e do cadastro de alunos. Para matrícula cadastram-se: data da matrícula, valor pago, aluno e a turma. O cadastro de aluno compreende: nome, número da carteira de identidade, CPF, data de nascimento, endereço completo e telefones de contato.

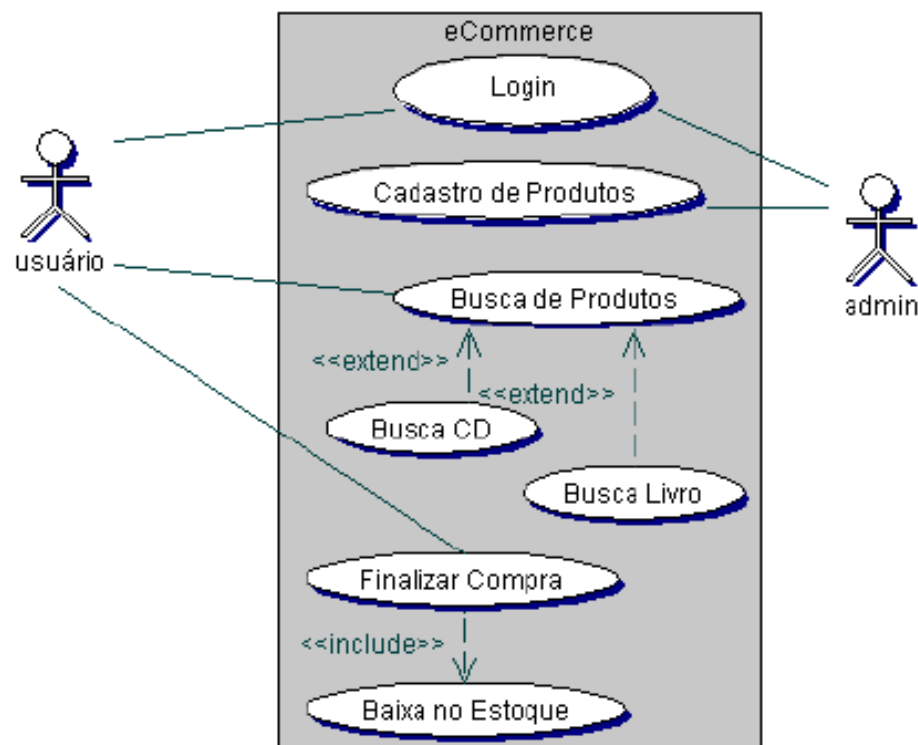
Para cada curso, deve-se controlar: a carga horária, o conteúdo programático e o valor do curso.

- *Desenhe o Diagrama de classes*

# Diagrama de Caso de Uso

## Diagrama de Casos de Uso

O Diagrama de Casos de Uso serve para visualizar os relacionamentos entre os atores e os casos de uso do sistema (cenários), numa visão geral. Serve para levantar os requisitos funcionais do sistema.



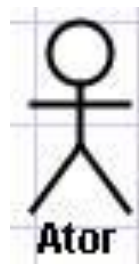
O diagrama de casos de uso é um diagrama da UML cujo objetivo é representar um *requisito* do sistema que será automatizado. Considere como requisito uma necessidade do sistema.

Simbologia de um caso de uso (requisito que será automatizado):

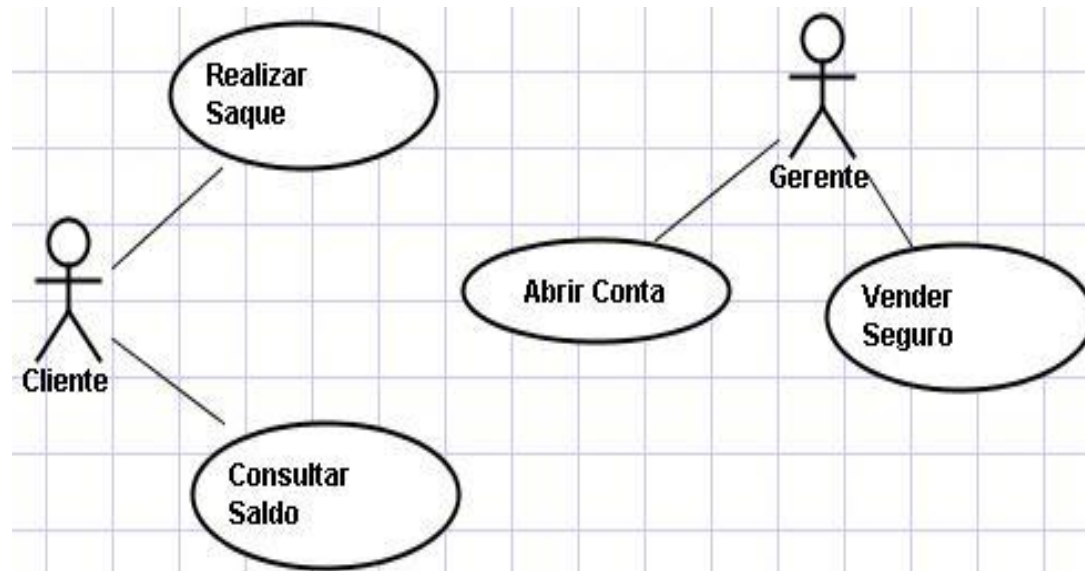


Usamos atores para representar as entidades que interagem com o sistema. Podem ser usuários, máquinas, sensores, etc... Um ator representa um papel no sistema, mas um papel pode ser representando por vários atores.

Simbologia de um ator:



Exemplo de um diagrama de casos de uso (*sistema bancário*):

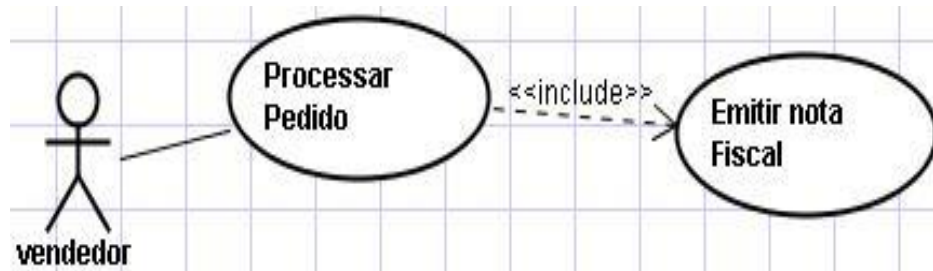


O ator cliente executará os casos de uso “realizar saque” e “consultar saldo”, enquanto o gerente poderá interagir com os casos de uso “abrir conta” e “vender seguro”.

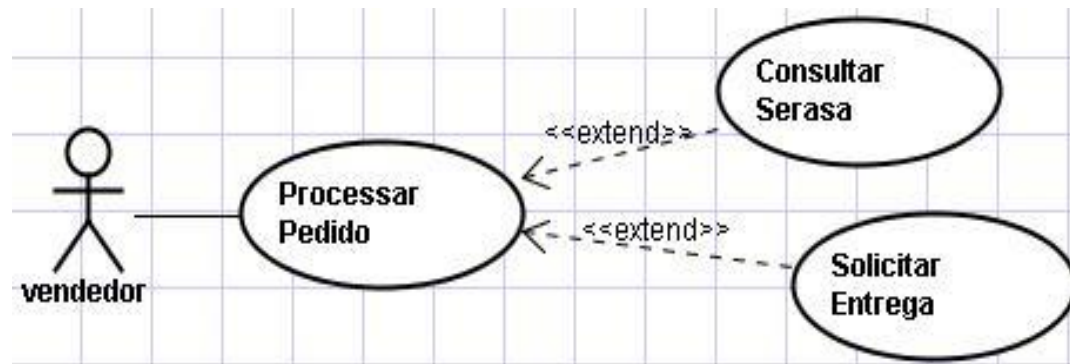
## Relacionamentos entre casos de uso

Os casos de usos podem se relacionar de duas formas:

*include*: Quando um caso de uso “A” inclui (include) outro caso de uso “B”. Isto implica que ao executar o caso de uso “A” executa-se também o caso de uso “B”.

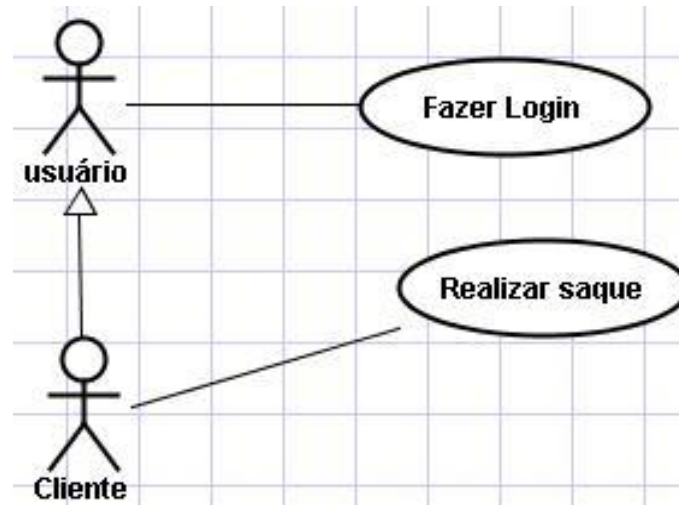


*Extends*: Quando um caso de uso “A” tem um relacionamento do tipo extends com outro caso de uso “B”. Implica que ao executar o caso de uso “A” não necessariamente “B” será executado.



## Relacionamento entre Atores

O ator pode herdar as funcionalidades (casos de uso) de outro ator.



## Objetivo

O Diagrama de *Casos de Uso* tem o objetivo de auxiliar a comunicação entre os analistas e o cliente.

Um diagrama de Caso de Uso descreve um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário.

O cliente deve ver no diagrama de Casos de Uso as principais funcionalidades de seu sistema.

## Notação

- ✓ O diagrama de Caso de Uso é representado por:
- ✓ atores;
- ✓ casos de uso;
- ✓ relacionamentos entre estes elementos.



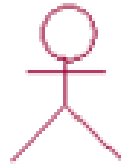
Estes relacionamentos podem ser:

- ✓ associações entre atores e casos de uso;
- ✓ generalizações entre os atores;
- ✓ generalizações, *extends* e *includes* entre os casos de uso.
- ✓ *casos de uso* podem opcionalmente estar envolvidos por um retângulo que representa os limites do sistema.

*casos de uso* podem opcionalmente estar envolvidos por um retângulo que representa os limites do sistema.

## Em maiores detalhes:

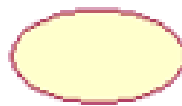
- **Atores**



Ator

Um ator é representado por um boneco e um rótulo com o nome do ator. Um ator é um usuário do sistema, que pode ser um usuário humano ou um outro sistema computacional.

- **Caso de uso**

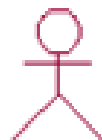


Caso de Uso

Um *caso de uso* é representado por uma elipse e um rótulo com o nome do *caso de uso*. Um *caso de uso* define uma grande função do sistema. A implicação é que uma função pode ser estruturada em outras funções e, portanto, um *caso de uso* pode ser estruturado.

- **Relacionamentos**

- Ajudam a descrever *casos de uso*
- Entre um ator e um *caso de uso*
  - Associação



Ator



Caso de Uso

Define uma funcionalidade do sistema do ponto de vista do usuário.

## Exercícios

---

Qual é a notação da UML para um caso *de uso*?

Qual é a notação da UML para um ator?

Qual a notação utilizada na UML para o relacionamento de generalização?

Qual o objetivo dos diagramas de casos de uso?

### **Cenário 1 :**

Rafaela possui vários temas de festas infantis para aluguel. Ela precisa controlar os aluguéis e para isso quer uma aplicação que permita cadastrar: o nome e o telefone do cliente, o endereço completo da festa, o tema escolhido, a data da festa, a hora de início e término da festa. Além disso, para alguns clientes antigos, Rafaela oferece descontos. Sendo assim, é preciso saber o valor realmente cobrado num determinado aluguel. Para cada tema, é preciso controlar: a lista de itens que compõem o tema (ex: castelo, boneca da Cinderela, bruxa etc.), o valor do aluguel e a cor da toalha da mesa que deve ser usada com o tema.

## **Cenário 2 :**

Construa um modelo de casos de uso para a seguinte situação fictícia: "Estamos criando um serviço de entregas. Nossos clientes podem nos requisitar a entrega de volumes. Alguns volumes são considerados de maior valor por nossos clientes, e, portanto, eles querem ter tais volumes segurados durante o transporte. Contratamos uma companhia de seguro para segurar volumes de valor".

## **Cenário 3 :**

Bruna resolveu desenvolver uma aplicação para controlar as ligações telefônicas de sua casa, a fim de checar se o valor que paga mensalmente está correto. Assim, sempre que desejar, poderá listar as ligações efetuadas em um determinado período, contabilizando o valor a pagar.

Para que isso seja possível, toda ligação será feita pelo computador. A cada solicitação de ligação, a aplicação deverá registrar: a data da ligação, a hora da ligação, quantidade de minutos gastos (que deve ser registrado no momento que a ligação for encerrada), o número de pulsos (que deve ser calculado pela aplicação) e o telefone para onde se discou.

A aplicação permitirá o controle de uma agenda de telefones, com número do telefone e nome da pessoa de contato. O usuário poderá escolher, no momento da ligação, se deseja um dos registros da agenda ou se digitará diretamente o número do telefone, A forma de cálculo dos pulsos considera os seguintes critérios:

- A ligação ao ser completada já conta um pulso. A partir daí, a cada quatro minutos de conversação concluída, cobram-se mais um pulso.
- Cada pulso custa R\$ 0,08 para ligações locais.

Exemplo:

Ligação de 2m - 1 pulso   Ligação de 4m30s - 2 pulsos   Ligação de 8m - 3 pulsos

- Os finais de semana possuem uma promoção. Cada ligação contabiliza somente um pulso, independente do número de minutos de conversação.

## Cenário 4 :

A loja CdcomCarinho trabalha com a venda, à vista e parcelada, de CD's de todos os gêneros musicais. Ela oferece a seus clientes, do estado do Rio de Janeiro, um serviço de “delivery”, permitindo que eles recebam, em casa, produtos requisitados pelo telefone.

Seus clientes estão acostumados a uma abordagem diferencial, ou seja, a loja costuma mandar mala direta quando chega algum produto cujo gênero se encaixe com o perfil daquele cliente. Há, também, ofertas promovidas durante datas especiais, por exemplo, no aniversário dos clientes, no dia dos namorados, etc. Clientes que já compraram mais de 20 CD's na loja são classificados como “Clientes Prata” e recebem descontos de 10%. Clientes, com mais de 50 compras, são denominados “Clientes Ouro”, com descontos de 25%.

O Gerente da loja precisa de uma análise periódica de qual Gênero de CD está vendendo mais para planejar os próximos pedidos aos fornecedores. E deve saber, também, qual a região do Estado do Rio que mais compra, para definir o foco da equipe de Marketing.

Os vendedores (por telefone ou na loja) recebem salário além da comissão sobre as suas vendas.

A CdcomCarinho deseja informatizar seu controle de vendas e de entregas. E, pretende, também, ampliar seu negócio através de vendas pela Internet.

### Sua função:

Imagine que você está fazendo o Levantamento de Requisitos para um Sistema que automatizará as funcionalidades do negócio descrito anteriormente. Descreva os **atores** que deverão interagir com o sistema e os **casos de uso** que especificam as funcionalidades do sistema.

Descreva o **fluxo de eventos** de, pelo menos, **dois** casos de uso.

Finalmente, esboce os **Diagramas de Caso de Uso focados** nestes casos de uso descritos e o **Diagrama de Caso de Uso de contexto**.

# Diagrama de Atividades

# Diagramas de Atividades

Uma atividade é o estado de estar fazendo algo: tanto um processo de mundo real, tal como datilografar uma carta, ou a execução de uma rotina de *software*, tal como um método em uma classe.

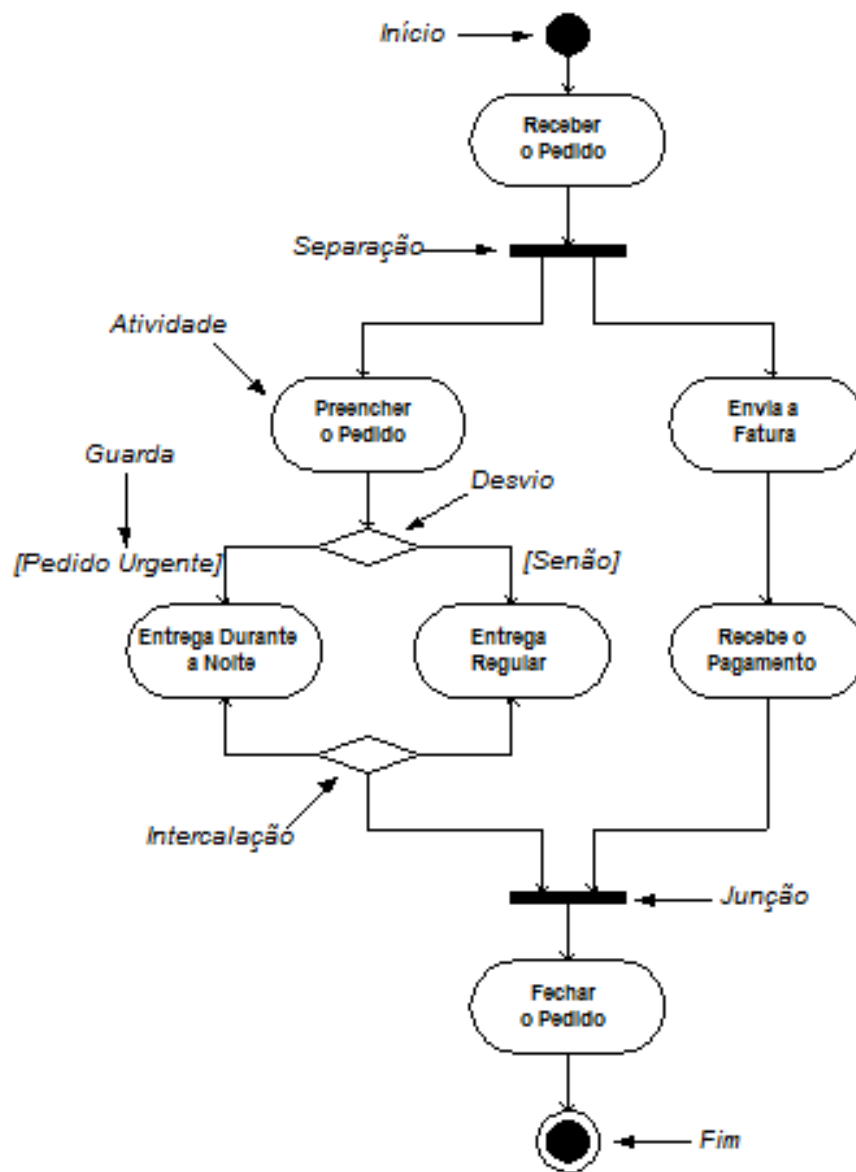
O diagrama de atividades descreve a sequência de atividades, com suporte para comportamento condicional e paralelo. Um diagrama de atividades é uma variante de um diagrama de estados no qual a maioria, se não todos, dos estados é estado de atividade. Portanto, muito da terminologia segue a mesma terminologia de estados.

## Decompondo uma Atividade

Uma atividade pode ser dividida em subatividades. Isso funciona mais ou menos como superestados e subestados em um diagrama de estados. Você pode somente mostrar o superestado no diagrama-pai, ou pode mostrar o superestado no seu comportamento externo dentro dele.

As vantagens dos estados de fim e início explícitos são que uma atividade de entrega pode ser usada em outros contextos, e o diagrama-pai é desacoplado dos conteúdos do diagrama subsidiário.

Na figura 1 o símbolo central é o estado de atividade, ou simplesmente atividade.





# Quando Utilizar Diagramas de Atividades

Os diagramas de atividades têm qualidades e fraquezas definidas, por isso a melhor maneira de usá-los é em combinação com outras técnicas.

A maior qualidade dos diagramas de atividades está no fato de que eles suportam e encorajam comportamento paralelo.

A maior desvantagem destes diagramas é que eles não deixam muito claras as ligações entre ações e objetos.

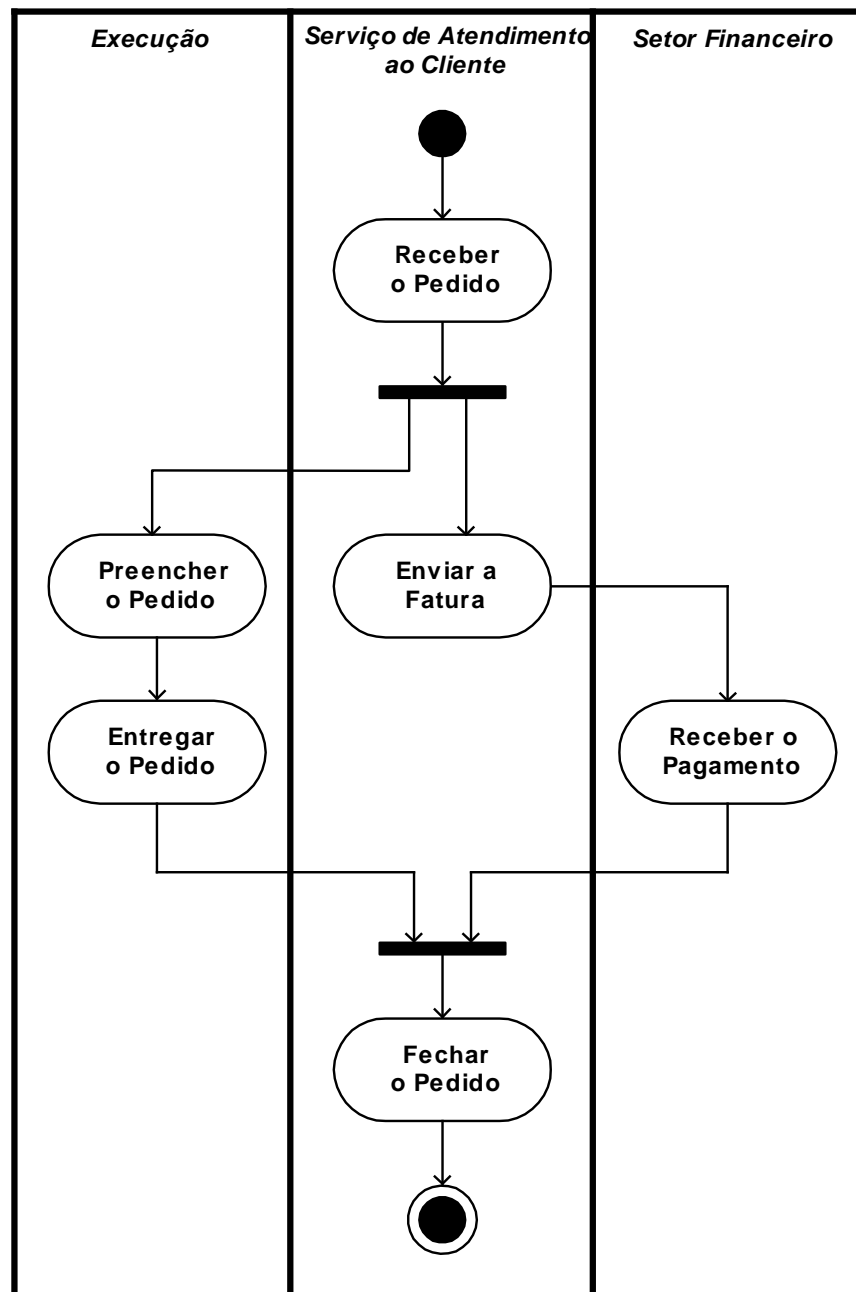
Você pode definir uma ligação para um objeto rotulando uma atividade com um nome de objeto ou usando raias que dividem um diagrama de atividades em base em responsabilidades, mas isso não tem a clareza simples de diagramas de interação. Por esta razão, algumas pessoas sentem que diagramas de atividades não são orientados a objetos e, portanto, são maus.

Devemos utilizar diagramas de atividades nas seguintes situações:

- Analisando um caso de uso.
- Compreendendo workflow
- Descrevendo um algoritmo seqüencial complicado
- Lidando com aplicações de processamento paralelo.

Não devemos usar diagramas de atividades nas seguintes situações:

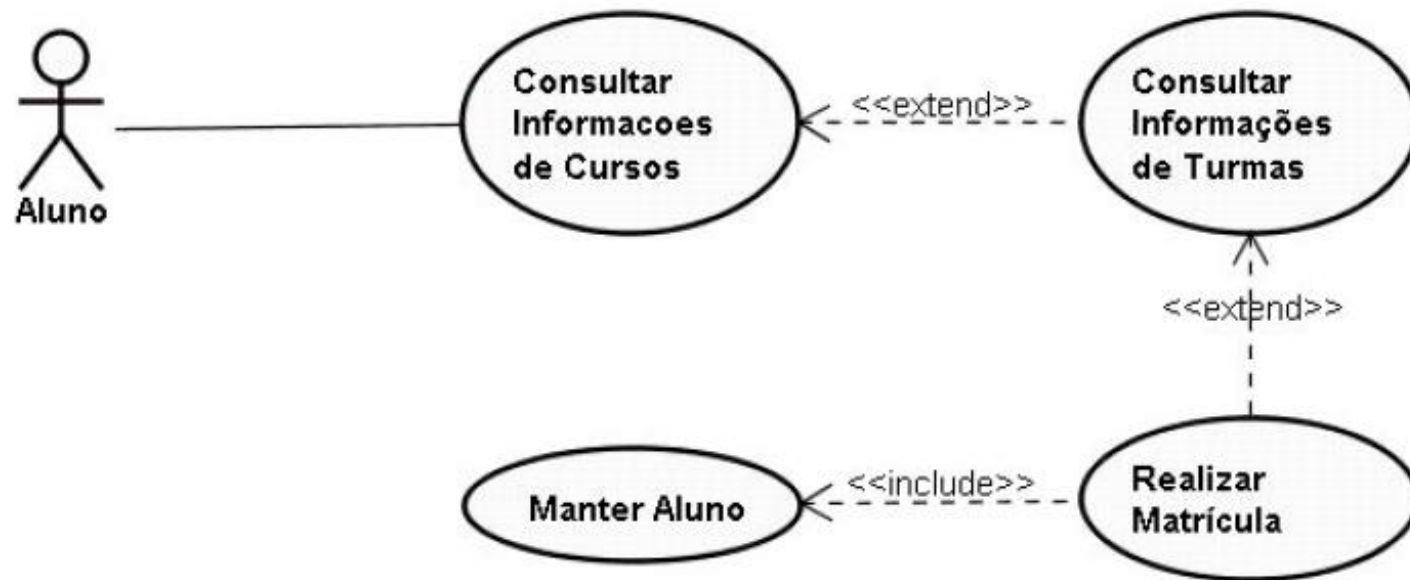
- Tentando ver como os objetos colaboram.
- Tentando ver como um objeto se comporta durante o seu ciclo de vida.
- Representando lógica condicional existente.



**Fig 5: Raias**

## Exercícios

Analise o Diagrama de Casos de Uso abaixo, referente a um módulo de matrícula e construa um Diagrama de Atividades para demonstrar modelagem dos processos do negócio.



2. Leia, interprete a descrição do caso de uso abaixo e complemente a sua especificação através de um Diagrama de Atividades.

**Projeto:** Controle de Cursos

**Nome:** Manter Aluno

**Descrição:** Este caso de uso permite a inclusão, exclusão, alteração e consulta de alunos, pela atendente

**Ator Principal:** Aluno

**Ator Secundário:** Atendente

**Pré-condição:** A atendente deverá estar devidamente identificada pelo sistema

### **Fluxo Principal:**

1. A Atendente informa o código do aluno [A1]
2. A Atendente solicita a busca
3. O sistema pesquisa os dados do aluno
4. O sistema exibe os dados do aluno [A2]
5. A Atendente edita os dados do aluno [A3]
6. A Atendente solicita a gravação dos dados
7. O sistema valida os dados informados
8. O sistema grava os dados do aluno [A4]
9. Fim do caso de uso

## **Fluxos Alternativos:**

### **A1. Novo Aluno**

1. A Atendente solicita a inclusão de um novo aluno
2. O sistema solicita os dados do novo aluno
3. A Atendente informa os dados do aluno
4. Vai para o passo 6 do fluxo principal

### **A2. Aluno não encontrado**

1. O sistema informa a situação à atendente
2. Vai para o passo 1 do Fluxo Principal

### **A3. Exclusão de Aluno**

1. Atendente solicita exclusão do aluno
2. O sistema solicita confirmação da exclusão
3. [se confirmação positiva] Sistema exclui aluno
4. Vai para o passo 9 do fluxo principal

### **A4. Dados inválidos**

1. Se algum dado do aluno estiver em desacordo com as regras de validações e restrições, o sistema informa situação à Atendente
2. Vai para o passo 5 do fluxo principal

**Pós-condições:** Os dados são incluídos, alterados ou excluídos conforme solicitação do aluno

### **Restrições e Validações:**

1. Nenhum campo poderá ser deixado em branco
2. O campo CPF deverá ser preenchido somente com números
3. O ano de nascimento deverá ser informado com 4 dígitos

# Diagrama de Sequencia

# Diagrama de Sequência

Apresenta a colaboração dinâmica entre os vários objetos de um sistema;

Através deste diagrama é possível perceber a sequência de mensagens enviadas entre os objetos;

Mostra o que ocorre em pontos específicos da execução do sistema;

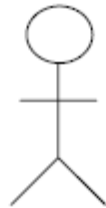
Apresenta as interações entre atores e sistema.



- Notação:



Objeto



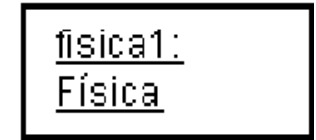
Ator



Mensagens



Tempo



- Interação entre os objetos
- Determina a seqüência de eventos que ocorrem em um determinado processo
  - Quais condições devem ser satisfeitas ...
  - Quais métodos devem ser disparados ...
  - E em qual ordem ...
- Baseia-se no Diagrama de Casos de Uso
  - 1 Caso de Uso  $\rightarrow$  N Diagramas de Seqüência
- Baseia-se, também, no Diagrama de Classes
  - Fornecem as classes e os métodos associados

- Exemplo – Especificação do Caso de Uso Efetuar Login:

Cenário Normal

- O sistema apresenta a tela de login
- O usuário digita seu username
- O usuário digita sua senha
- O sistema verifica as informações de login
- O sistema apresenta a tela inicial da aplicação

Cenário de Exceção - A partir do passo 4

- O sistema verifica que a senha e/ou username estão incorretos
- O sistema mostra mensagem de erro
- O usuário re-digita o username e a senha

- **Componentes - ATORES**

Exatamente os mesmos dos Casos de Uso

Interagem → Solicitam serviços → Eventos → Processos

Não são obrigatórios no Diagrama de Seqüência

- **Componentes - OBJETOS**

Representam as instâncias das classes

Retângulos contendo um texto

Primeira parte, em minúsculo, o nome do objeto

Segunda parte, em letras iniciais maiúsculas, o nome da classe

Informações separadas por dois pontos (:)

- **Linha de vida**

Linha vertical tracejada

Representa o tempo que um objeto existiu durante um processo

Linhas finas verticais tracejadas

Iniciam no retângulo que representa o objeto

Interrompida por um “X” quando o objeto é destruído

- Exemplo – Especificação do Caso de Uso Efetuar Login:

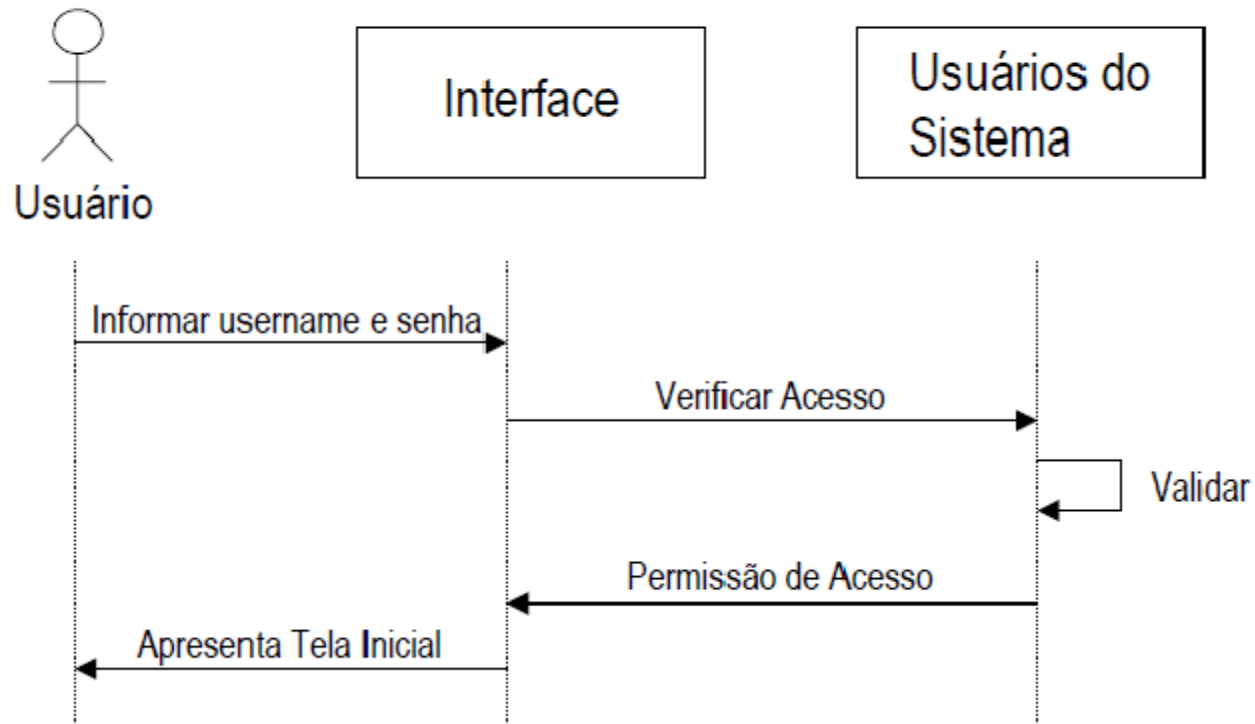
Cenário Normal

- O sistema apresenta a tela de login
- O usuário digita seu username
- O usuário digita sua senha
- O sistema verifica as informações de login
- O sistema apresenta a tela inicial da aplicação

Cenário de Exceção - A partir do passo 4

- O sistema verifica que a senha e/ou username estão incorretos
- O sistema mostra mensagem de erro
- O usuário re-digita o username e a senha

- Exemplo – Cenário Normal do Caso de Uso Efetuar Login:



## Exercícios

**Desenvolva o diagrama de sequencia para um sistema que visa atender a uma Vídeo Locadora**

# Ferramenta CASE



O termo CASE (Computer-aided software engineering) significa engenharia de software auxiliada por computador.

Antigamente havia a necessidade de visualizar o software como um todo mas não havia esta possibilidade.

Com a crescente demanda de integração, pela unificação de planejamento administrativo, análise e projetos, surgiram assim as ferramentas CASE.

Uma ferramenta CASE é um software que auxilia no ciclo de desenvolvimento de um sistema, desde a fase de análise, fase de testes, desenvolvimento e apoiam na manutenção de metodologias. Armazenam as informações de uma forma própria, como textos, imagens, gráficos, possibilitando a integração com o usuário.

As ferramentas CASE dividem-se em 3 tipos:

- Integrated System CASE (I-CASE), que visam desde a análise até a geração do código.
- Case's de automação de uma fase ou mais do desenvolvimento. São ferramentas que se prendem a uma etapa do projeto, como por exemplo Ferramentas para modelagem de dados., Ferramenta de testes.
- Ferramentas que seguem uma metodologia específica, como por exemplo ferramentas para Scrum, XP.

Algumas vantagens que podemos ver no uso de ferramentas CASE são:

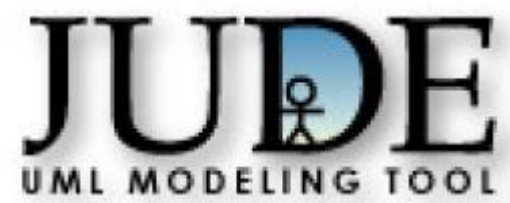
- Maior qualidade nos produtos finais;
- Produtividade;
- Eliminação de retrabalho;
- Mais tempo para a tomada de decisão;
- Flexibilidade para mudanças;
- Melhor documentação;
- Manutenção mais fácil e ágil;

As ferramentas CASE nos auxiliam em todas as fases do projeto, portanto seu uso é de extrema importância, considerando suas vantagens não vemos outra saída a não ser um projeto bem desenvolvido e documentado.

A aceitação das ferramentas CASE ocorreu com diagramas como o DFD (**Diagrama de Fluxo de Dados**)/DER (**Diagrama de Entidade e Relacionamento**), que só foram utilizados quando surgiram as primeiras ferramentas para auxiliar na tarefa de diagramação.

Podemos dizer que as ferramentas CASE evoluíram de uma forma aberta, de maneira que as mesmas cada vez mais se integram com outras ferramentas de fabricantes distintos.

A ferramenta CASE que escolhi para o artigo foi o **Enterprise Architect**, ela segue a linha de notação para a UML.



**Introdução**

**Download e Instalação**

**Apresentação da Ferramenta**

**Salvando o Projeto**

**Criando um Diagrama**

**Diagrama de Classe**

**Diagrama de Casos de Uso**

**Diagrama de Seqüência**

**Facilidades da Ferramenta**

**Geração de Código**

## Download e Instalação

Para o funcionamento integral da nova versão do JUDE é indicada a instalação do J2SE, que pode ser encontrada no site da sun ( <http://java.sun.com/j2se/> ).

Para a instalação do JUDE Community (versão gratuita) proceda da seguinte forma:

- Acesse o site (<http://jude.change-vision.com/jude-web/download/index.html>) e faça o cadastro;
- Faça o download da versão mais atual do JUDE Community (gratuita);
- Instale o arquivo executável.

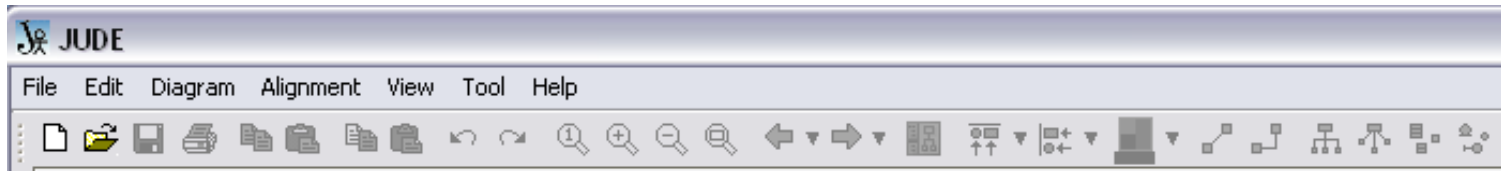
Note que a versão community não requer chave, é gratuita, existem outras versões que são pagas.

Caso haja algum problema na abertura do Jude, confirme se o J2SE (ou JRE) está devidamente instalado com a versão 1.4.1\_02 ou superior.

## Apresentação da Ferramenta

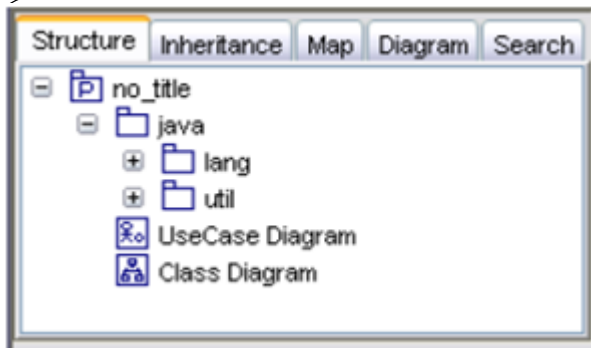
Para abrir o Jude clique no ícone da área de trabalho, ou pela barra de ferramentas em Programas, JUDE Community, clique em JUDE Community.


Assim que entramos no JUDE vemos a barra de menu e a barra de botões (ou barra de ferramentas).



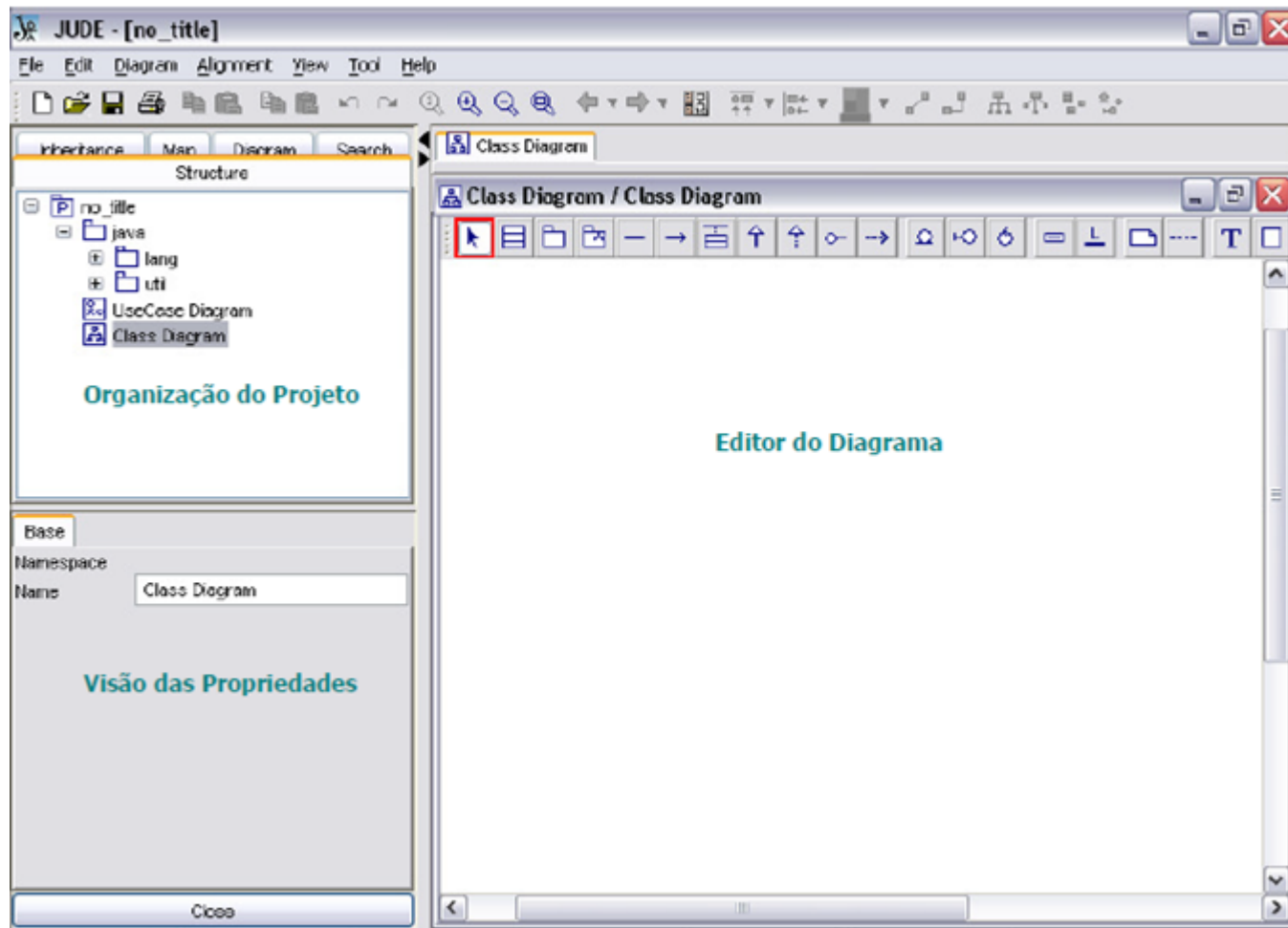
Para criar um novo projeto clique em File e posteriormente em New, ou clique na barra de ferramentas.

Uma barra vertical à esquerda é criada exibindo a visão do projeto, é a área de organização do projeto, repare que automaticamente são criados os diagramas de Casos de Uso e de Classes.



Vamos abrir o diagrama de classes clicando duas vezes (duplo clique) em  "Class Diagram" na árvore de diagramas do projeto.

A seguir a organização da ferramenta.





## **O Jude está organizado em três partes:**

- Organização do Projeto;
- Visão das Propriedades; e
- Editor do Diagrama

A Organização do Projeto é uma área que possui várias abas onde temos visões diferentes do projeto, são elas: Support Structure Tree (árvore de estrutura do projeto), Inheritance Tree (exibe as heranças identificadas), MapView (exibe todo o editor de diagrama), DiagramList (mostra a lista de diagramas do projeto), Search & Replace (para localização de modelos e substituição de nomes).

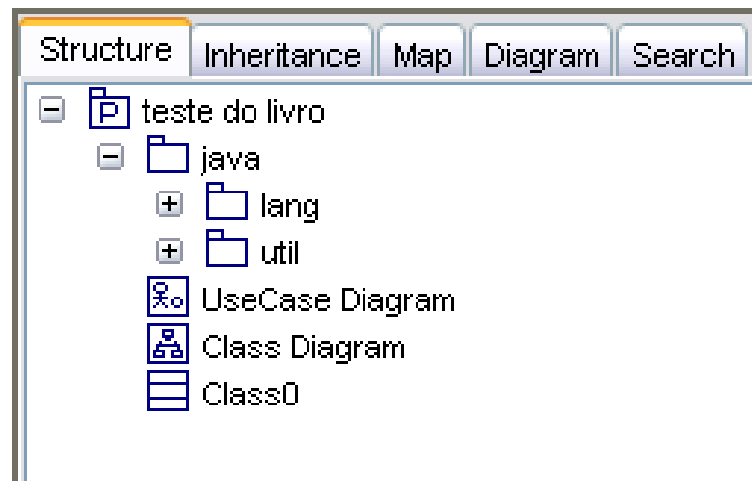
A Visão das Propriedades é a área onde podemos alterar as propriedades dos elementos do diagrama, basta selecionar um item que será exibido suas propriedades. Por exemplo, com o diagrama de classes aberto e a seleção de uma classe, são exibidas todas as propriedades da classe, como: seu nome, a visibilidade, atributos e operações e etc.

O Editor do Diagrama é a área onde são exibidos os diagramas, clicando duas vezes no diagrama, que é exibido na lista de diagramas, o diagrama será carregado nesta área exibindo todos os seus elementos.

## Salvando o Projeto

Para salvar o projeto faça, no menu, “File”, “Save” (ou “Save As”) e informe o nome do arquivo e o diretório em que o projeto deverá ficar armazenado.

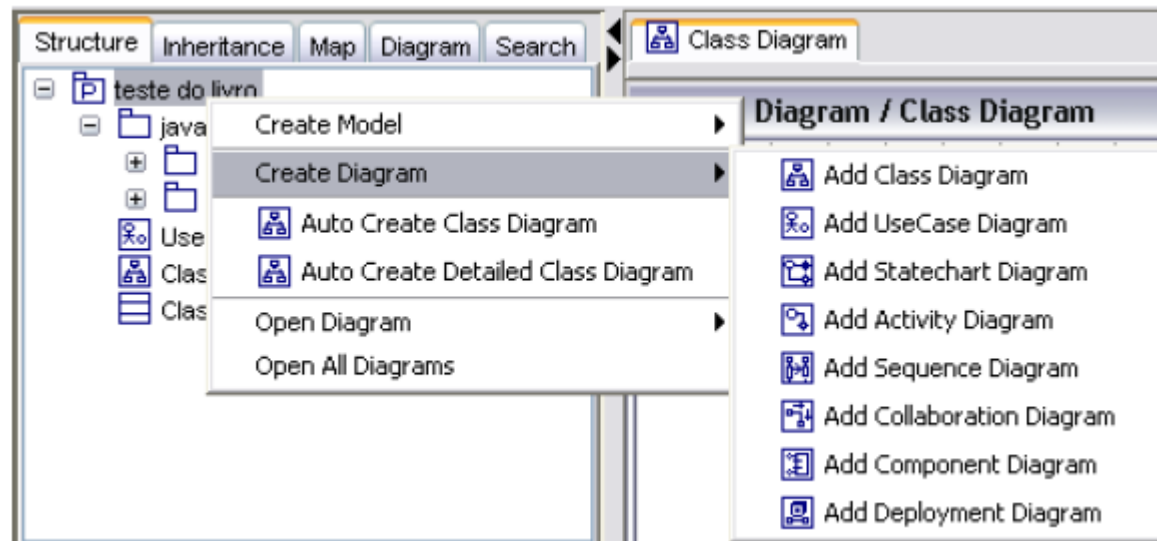
Perceba que o nome do projeto informado é atribuído ao pacote raiz na área de organização (figura do pacote com a letra P) do projeto, no exemplo abaixo o nome do projeto informado foi ‘teste do livro’.



## **Criando um novo diagrama**

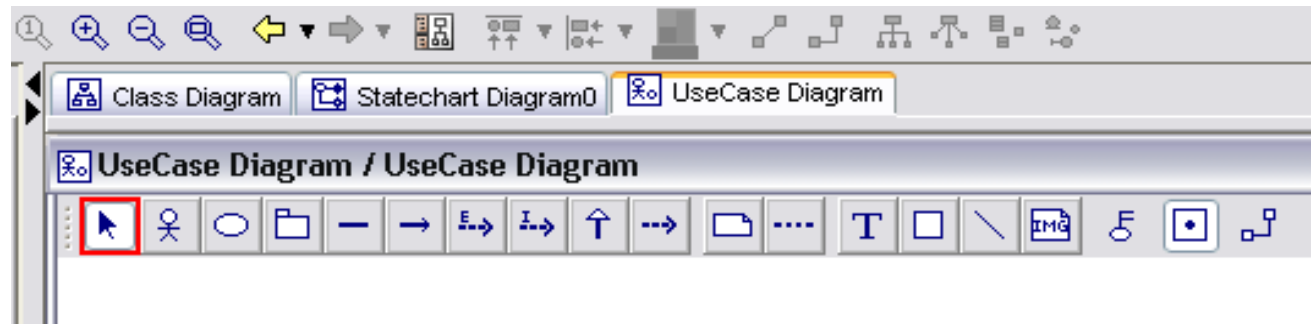
Para criar um novo diagrama, dependendo do diagrama que se quer criar, deve selecionar um pacote, subsistema ou modelo, clicar com o botão direito e selecionar o item “Create Diagram” do menu suspenso, para então selecionar o diagrama a ser criado. As opções de diagramas a serem criados são:

- “Add Class Diagram” – Diagrama de Classes
- “Add UseCase Diagram” – Diagrama de Casos de Uso
- “Add StateChart Diagram” – Diagrama de Estado
- “Add Activity Diagram” – Diagrama de Atividades
- “Add Sequence Diagram” – Diagrama de Seqüência
- “Add Collaboration Diagram” – Diagrama de Colaboração
- “Add Component Diagram” – Diagrama de Componente
- “Add Deployment Diagram” – Diagrama de Implantação



O diagrama também pode ser criado através do menu, para isso selecione o local na árvore da estrutura do projeto e no menu clique em “Diagram” e selecione o diagrama a ser criado (idem à lista anterior).

Note que na abertura de um novo diagrama a área do editor de diagramas exibe em abas os diagramas abertos. Veja na figura abaixo os diagramas de classe, de estados e de casos de uso que estão abertos ao mesmo tempo.



A qualquer momento pode ser feita à alteração do nome do diagrama, basta selecionar o diagrama na 'Organização do Projeto' e abaixo na 'Visão das Propriedades' é exibido o nome do diagrama, clique na caixa de texto e digite o nome desejado.

A seguir a hierarquia de criação de diagramas:

Modelo Selecionado	Diagramas	Modelos
project		package model subsystem
package	class diagram usecase diagram state chart diagram activity diagram sequence diagram collaboration diagram component diagram deployment diagram	package subsystem class interface actor usecase
model	class diagram usecase diagram	model package

	state chart diagram activity diagram sequence diagram collaboration diagram component diagram deployment diagram	subsystem class interface actor usecase
subsystem	class diagram usecase diagram state chart diagram activity diagram sequence diagram collaboration diagram component diagram deployment diagram	subsystem package model class interface actor usecase
class interface actor	state chart diagram activity diagram	attribute method
usecase	state chart diagram activity diagram sequence diagram collaboration diagram	
method	state chart diagram activity diagram sequence diagram collaboration diagram	



## Diagrama de Classe

O primeiro passo é criar o diagrama, já vimos anteriormente como fazer isso. Para trabalhar com o diagrama ele deve estar aberto na área 'Editor de Diagrama', para isso dê um duplo clique no diagrama na área de 'Organização do Projeto'.

Apresentando a barra de ferramentas do diagrama de classes:



- Seta de seleção (ponteiro)
- Classe
- Pacote
- Subsistema
- Associação
- Associação com navegação
- Classe associativa (novo)
- Generalização
- Realização
- Interface
- Dependência
- Classe de Entidade
- Classe de Fronteira
- Classe de Controle
- Objeto
- Link
- Nota
- Link da Nota
- Texto
- Retângulo
- Linha
- Imagem
- Modo de Segurança

Para inserir uma classe basta clicar no botão da classe , clicar na área do diagrama e digitar o nome da classe. Quando quiser ver as propriedades, ou ajeitar a classe em outro lugar do diagrama use o ponteiro  para selecionar e arrastar a classe ou outros objetos do diagrama. Quando um objeto é selecionado suas características ficam expostas na área de Visão das Propriedades.

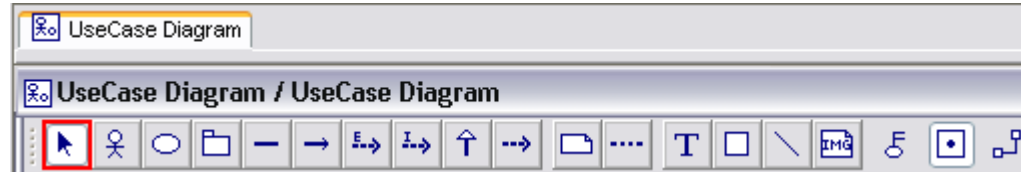
Para a criação dos atributos e operações selecione a classe e navegue na área de 'Visão das Propriedades' nas abas "Attribute" e "Operation" clique em "Add" e dê um duplo clique no nome do atributo ou da operação e informe o nome.

A associação, associação com navegação, generalização, realização ou dependência são criados de uma classe para outra. Para criação, selecione uma dessas associações e depois, na área do 'Editor do Diagrama' clique em uma classe origem e arraste até a classe destino.



# Diagrama de Casos de Uso

Com a abertura do diagrama é exibida a barra de ferramentas do Diagrama de Casos de Uso:



Os elementos do diagrama são descritos a seguir:

- Seta de seleção de itens;
- Ator (É quem executa a(s) funcionalidade(s) a(s) qual(is) está associado);
- Caso de uso (Deve descrever a funcionalidade no caso de modelagem de sistema);
- Pacote (É um agrupamento lógico de itens do sistema);
- Associação (Amarra um ator a um caso de uso, representa a interação, o ator inicializa o caso de uso);
- Associação unidirecional (É utilizado quando se quer representar a participação de um usuário em um caso de uso, o ator não inicia o caso de uso, apenas participa do evento);
- Associação de extensão (Extend - Usado entre casos de uso, para representar uma execução ocasional, no qual um caso de uso para seu serviço para usar outro caso de uso);
- Associação de inclusão (Include - Representa uma execução obrigatória entre casos de uso.

O caso de uso A inclui o caso de uso B quando sempre usa seus serviços);

- Generalização/Especialização (É a herança que pode ocorrer entre casos de uso e atores)


Para a representação da hierarquia de herança (também chamado de árvore de herança) clique no menu [Edit] | [Generalization Style] | [Shared].

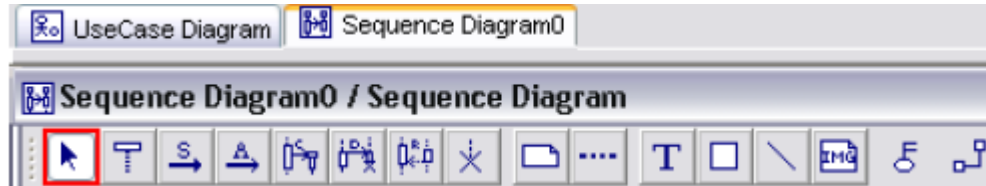
- Dependência

Representação de itens (casos de uso, atores ou pacotes) dependentes e independentes.

- Nota
- Link de nota
- Texto
- Quadrado
- Linha
- Imagem

## Diagrama de Seqüência










Para criar o diagrama de seqüência use o menu [Diagram] | [Sequence Diagram] , , quando o diagrama é criado ao lado no 'Editor do Diagrama' é aberto o diagrama de seqüência em branco com a barra de ferramentas:




- Seta de seleção de itens
- Objeto
- Mensagem
- Mensagem assíncrona
- Mensagem de criação de objeto
- Mensagem de destruição do objeto
- Mensagem de retorno
- Terminação
- Nota
- Link da nota
- Texto
- Quadrado
- Linha
- Imagem

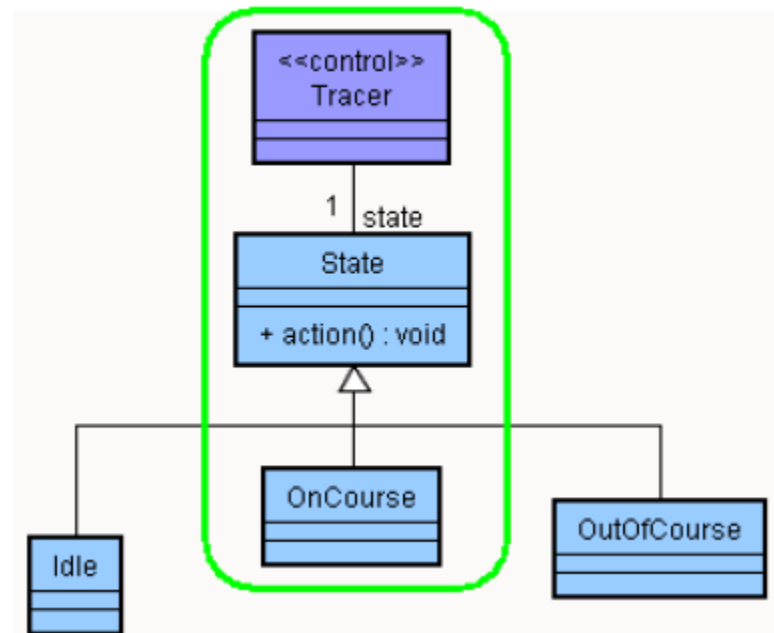
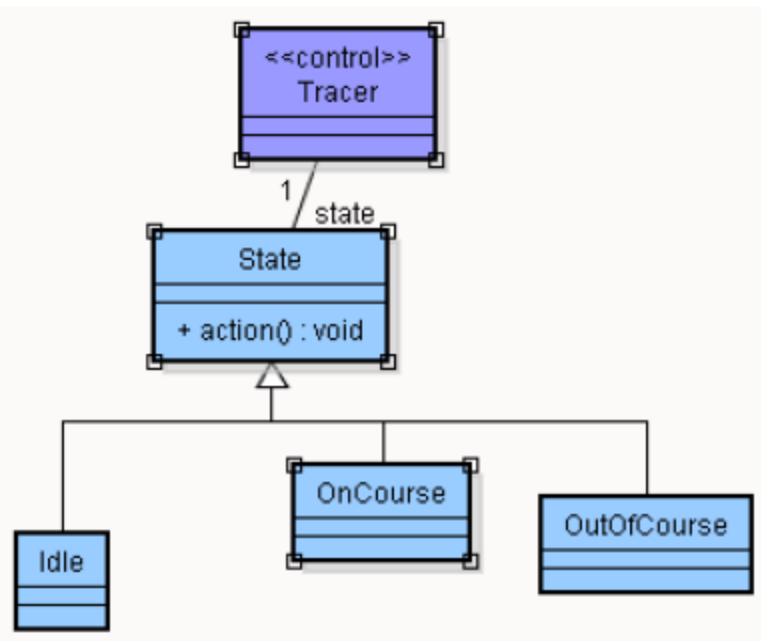
## Facilidades da Ferramenta

Alinhamento dos itens do diagrama através dos botões de alinhamento na barra de ferramentas.


-  Alinhamento no Topo (Align Top)
-  Alinhamento Horizontal (Align Horizontal Center)
-  Alinhamento Abaixo (Align Bottom)
-  Alinhar a Esquerda (Align Left)
-  Alinhar Vertical Centralizado (Align Vertical Center)
-  Alinhar Vertical à Direita (Align Right)
-  (Align Horizontal Even)
-  (Align Vertical Even)
-  (Align Size)

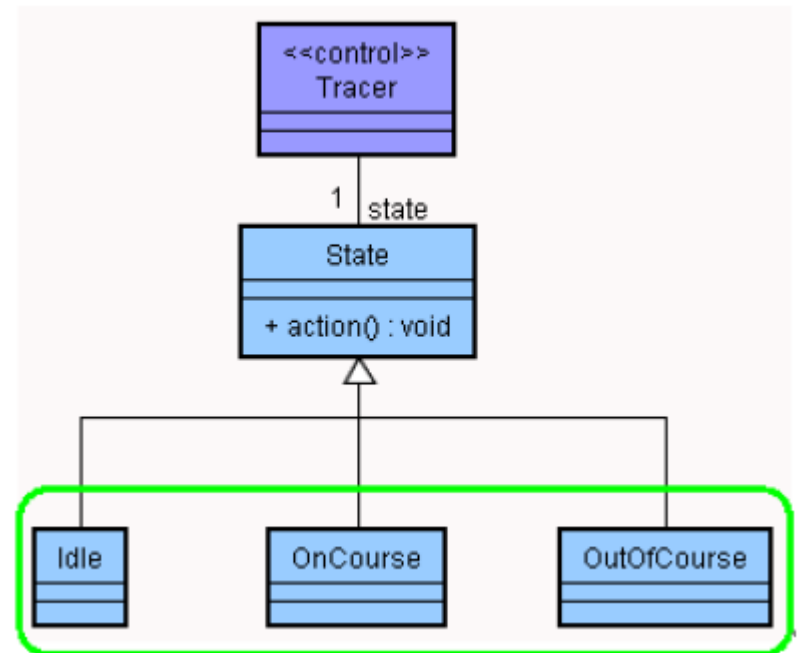
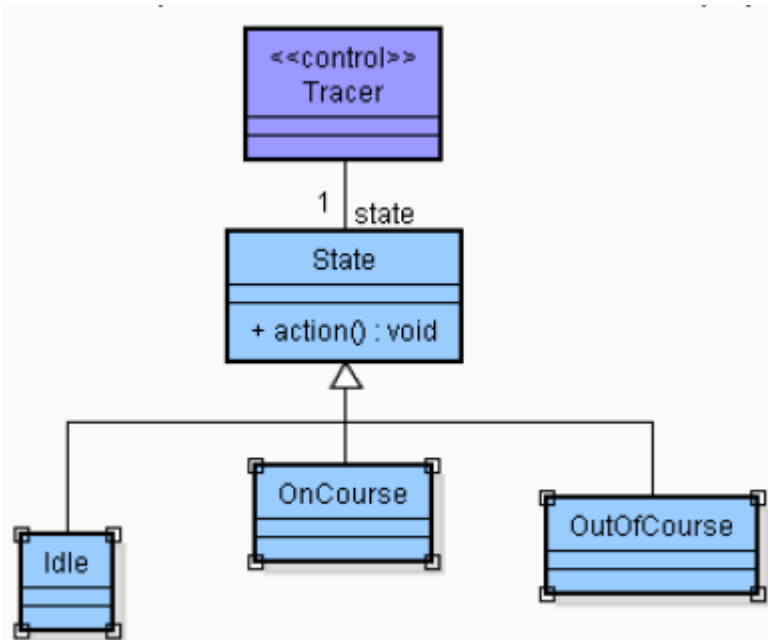
Exemplo para alinhamento vertical.

1. Selecione os itens a serem alinhados (para selecionar vários itens aperte o Shift + Clique ou use o botão esquerdo arrastando sobre os elementos do diagrama).
2. Clique no botão de alinhar  ( ) vertical da barra de ferramentas.



## Exemplo para alinhamento horizontal.

1. Selecione os itens a serem alinhados (para selecionar vários itens aperte o Shift + Clique ou use o botão esquerdo arrastando sobre os elementos do diagrama).
2. Clique no botão de alinhamento horizontal  da barra de ferramentas.



## Geração de Código

O JUDE gera código na linguagem Java, este código gerado trata-se apenas da definição da classe e de seus atributos e métodos, para isso o Diagrama de Classes deve estar pronto, as classes bem definidas, assim como o tipo e tamanho de todos os atributos, e nos métodos os parâmetros definidos.


Veja o exemplo abaixo do desenho da classe e o código gerado utilizando a ferramenta.



```
package Default Package;
import java.util.Currency;
public class Livro {
    private String titulo;
    private int QtdPáginas;
    private Currency precoBase;
    private Currency precoPromocional;
    private String capa;
    private int ISBN;

    public void livro(String tit, Currency precoBase,
Currency precoPromocional, int ISBN) {
        }
    }
}
```

Para geração do código deve ser feito:

Na opção do menu 'Tool' clique na sub-opção 'Export Java' então informe o arquivo que servirá de base para a criação do código Java, de onde serão extraídas as classes para a geração do código. Após a seleção do arquivo é exibida uma janela de seleção de classes, a barra mais à esquerda é uma lista com a hierarquia de pacotes, abra o pacote onde se localizam as classes que serão exportadas. Assim que selecionado um pacote suas classes são listadas na lista (candidate list – coluna central) nesta coluna, selecione as classes desejadas clicando na classe e depois no botão . Então o nome da classe aparece na lista de itens selecionados (selected list – coluna da direita).

Os outros botões são bem intuitivos:



- passa para a lista de itens selecionados todas as classes do pacote.



- desfaz a seleção de uma classe (a que estiver selecionada) na lista de itens selecionados.



- desfaz toda a seleção de classes da lista de itens selecionados.

Depois de todas as classes selecionadas clique no botão 'Approve' então deve aparecer a mensagem confirmando a exportação (Successfully exported).

Os arquivos exportados podem ser encontrados na pasta criada no mesmo diretório do arquivo jude. É criado um diretório com o nome do pacote e dentro deste todos os arquivos com extensão java.

A exportação do JUDE não se limita somente à linguagem de programação é possível exportar os diagramas para imagem ou para html.