Orientação a Objetos

CCUEC/Unicamp
março/99

Objetivo

- Introduzir os conceitos básicos de Orientação a Objetos
- Familiarizar-se com a nomenclatura e notações da nova tecnologia

Roteiro

Conceitos Básicos

- Objetos
- Mensagens
- Métodos
- Classes
- Relacionamentos
- Classes abstratas
- Polimorfismo
- Metaclasses

Roteiro

Conceitos Avançados

- Padrões de Projeto (Design Patterns)
- Frameworks
- Componentes e wrappers
- RMI

Roteiro

Metodologia de Desenvolvimento

- Visão Geral
- Método Integrado
- Reutilização de Projeto e Software

Orientação a Objetos

Origens

- Linguagens de Programação - Simula, Smalltalk, Flavours, Objective C, C++,...
- Inteligencia Artificial - frames
- Banco de Dados - pesquisa em modelos de dados semânticos

Paradigma

"Paradigma é um conjunto de regras que estabelecem fronteiras e descreve como resolver os problemas dentro destas fronteiras.
Os paradigmas influenciam nossa percepção; ajudam-nos a organizar e a coordenar a maneira como olhamos para o mundo..."

Reengenharia - Reestruturando a Empresa
Daniel Morris e Joel Brandon

Orientação a Objetos

O termo orientação a objetos significa organizar o mundo real como uma coleção de objetos que incorporam
estrutura de dados* e *um conjunto de operações
que manipulam estes dados

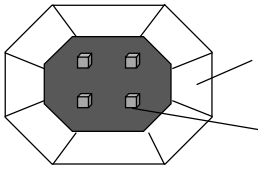
Objetos - Exemplos

Coisas tangíveis → o livro “Violetas na Janela”

Incidente (evento/ocorrência) → a Copa das Confederações

Interação (transação/contrato) → o débito de R$100,00 na conta “x” do dia 11/8/1999

Objetos - Metáfora



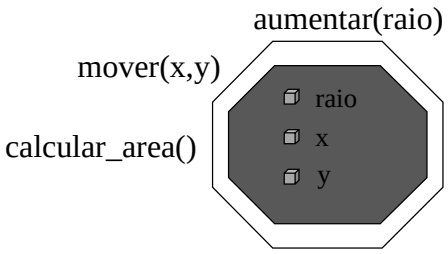
operações/métodos/interface (*público*)

dados/propriedades/atributos (*privado*)

A estrutura de dados armazena o **estado** de um objeto (valores dos atributos)

As operações definem o **comportamento** do objeto, que é a forma como um objeto age e reage em termos de mudanças de estado e passagem de mensagens

Objetos - Exemplo



x,y: coordenadas do centro do círculo

Um Círculo

Encapsulamento (“data hiding”)

Encapsulamento é definido como uma técnica para minimizar interdependencias entre “módulos” através da definição de interfaces externas.

fenômeno da “caixa preta”

Encapsulamento - Exemplo I

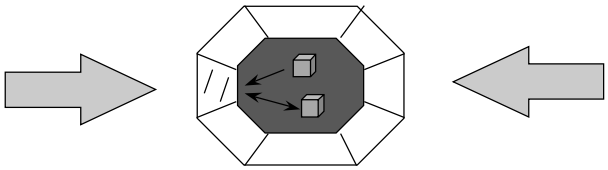
Considere o seguinte trecho de código C:

```
double  d  = 0;
        d += 2.5;
```

- Como um *double* é representado internamente?
- O que acontece quando você usa o operador +=?
- Você alguma vez se importou com isto?
- Você deveria se preocupar?

Objetos - Encapsulamento

A interface (pública) de um objeto declara todas as operações permitidas (métodos)



Todo o acesso aos dados do objeto é feito através da chamada a uma operação (método) da sua interface

Mudanças na implementação de um objeto, que preservem a sua interface externa, não afetam o resto do sistema.

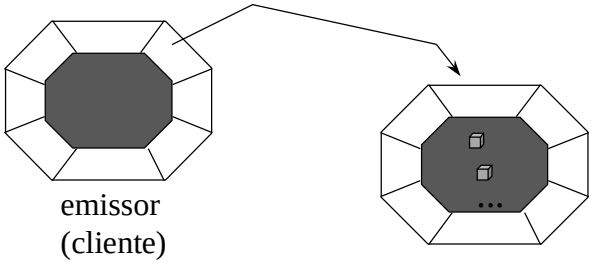
Encapsulamento - Benefícios

Segurança
protege os atributos dos objetos de terem seus valores corrompidos por outros objetos

Independência
"escondendo" seus atributos, um objeto protege outros objetos de complicações de dependência de sua estrutura interna

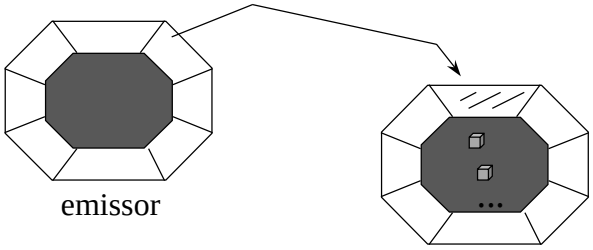
Mensagens

Objetos interagem e comunicam-se através de ***mensagens...***



Métodos

...as mensagens identificam os ***métodos*** a serem executados no objeto receptor



Mensagens e Métodos

Para invocar um método de um objeto, deve-se enviar uma mensagem para este objeto.

Para enviar uma mensagem deve-se:

- identificar o ***objeto*** que receberá a mensagem
- identificar o ***método*** que o objeto deve executar
- passar os ***argumentos*** requeridos pelo método

Métodos

O que um determinado método pode fazer com os valores dos atributos do objeto ?

Tipos de métodos → ciclo de vida
→ recuperadores (acesso)
→ cálculos

Objetos - Resumo

Um objeto possui:

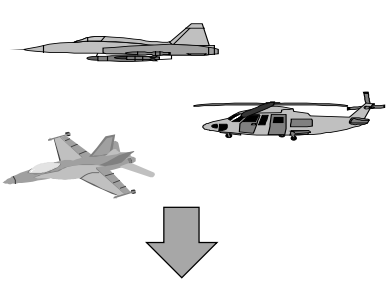
- **um estado**
  (definido pelo conjunto de valores dos seus atributos em determinado instante)

- **um comportamento**
  (definido pelo conjunto de métodos definido na sua interface)

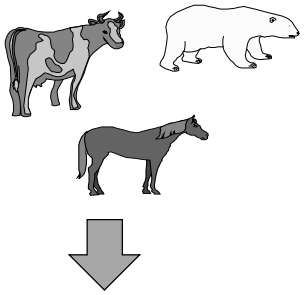
- **uma identidade única (!)**

Abstração

Focalizar o essencial, ignorar propriedades acidentais



Aeronave



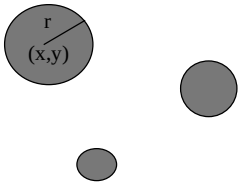
Mamífero

A abstração deve ser sempre feita com algum objetivo, porque este determina o que é e o que não é importante.

Classes

Uma **classe** descreve um conjunto de objetos com:

- propriedades semelhantes
- comportamentos semelhantes
- relacionamentos comuns com outros objetos



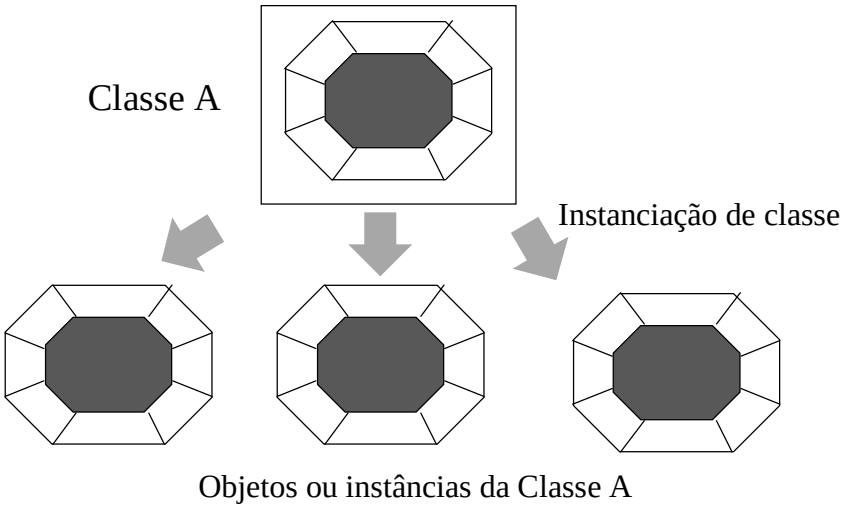
Objetos/instâncias

classificação

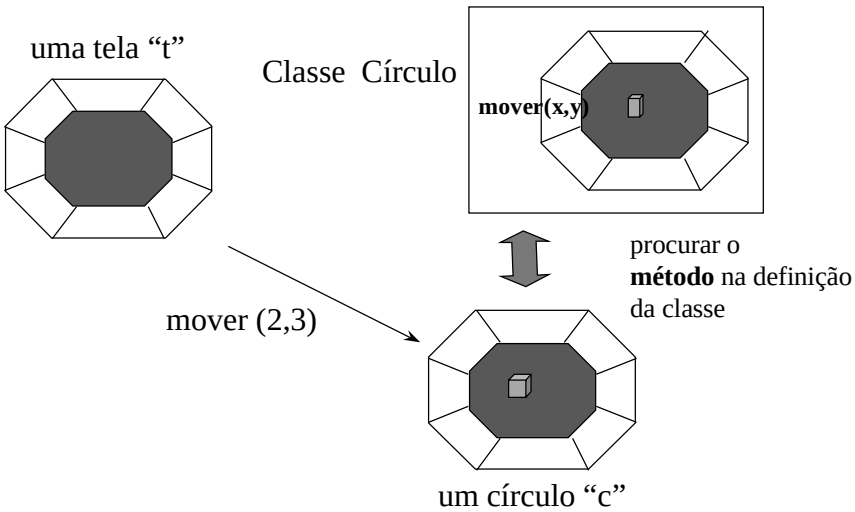
instanciação

Classe Circulo
Atributos
raio
x
y
Métodos
mover
mostrar
aumentar...

Classe, uma Fábrica de Objetos

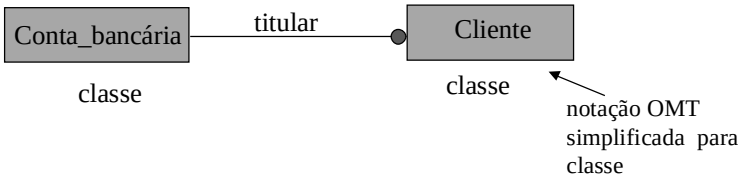


Comunicação entre Objetos



Relacionamentos

- Um relacionamento modela uma conexão física ou conceitual entre objetos
- Relacionamentos são bidirecionais

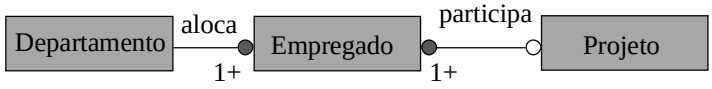


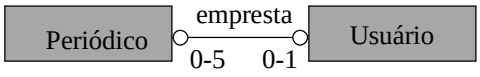
Objetos da classe **Conta_bancária** estão associados a objetos da classe **Cliente**

Relacionamentos - Multiplicidade

Especifica quantas instâncias de uma classe podem, ao mesmo tempo, ser relacionadas com uma única instância de uma classe associada

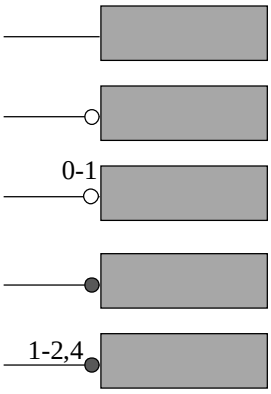
Exemplos



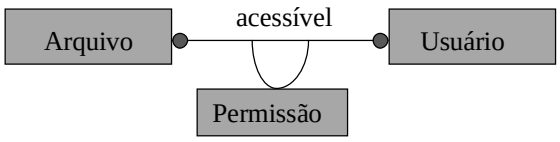


Relacionamentos - Multiplicidade

Resumo



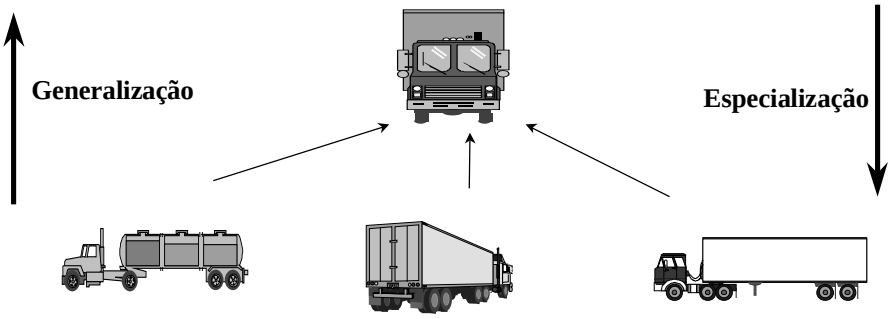
Relacionamentos - Atributos



Às vezes, atributos estão mais ligados com relacionamentos (ou associações) entre classes, do que com qualquer uma das classes envolvidas

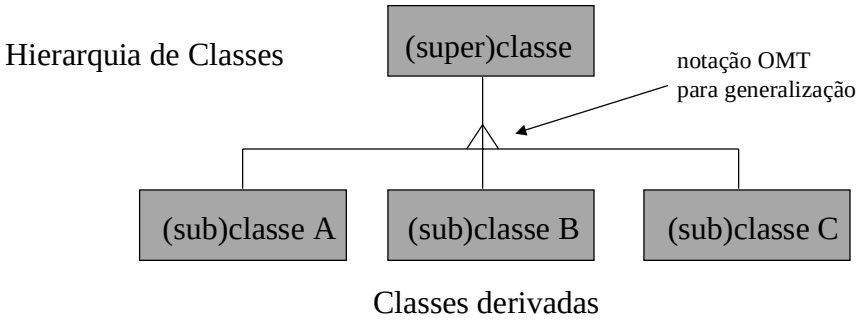
Generalização/Especialização

Generalização é a abstração que permite **compartilhar** semelhanças, preservando diferenças



Generalização entre Classes

Ao relacionamento entre uma classe e uma ou mais **versões refinadas** dessa classe, denominamos de **generalização**



Herança

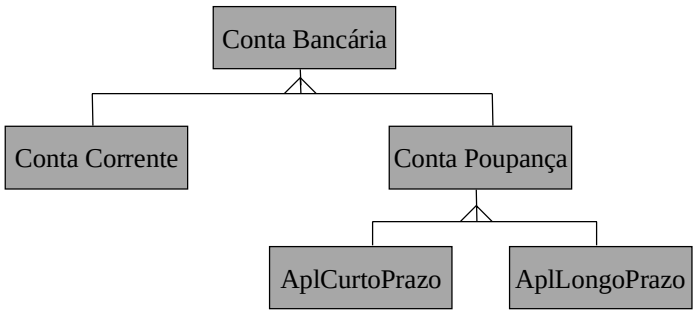
Herança é um mecanismo para derivar novas classes a partir de classes existentes através de um processo de refinamento.

Uma classe derivada **herda** a estrutura de dados e métodos de sua classe "base", mas pode seletivamente:

- **adicionar novos métodos**
- **estender a estrutura de dados**
- **redefinir a implementação de métodos já existentes**

Uma classe "base" proporciona a funcionalidade que é comum a todas as suas classes derivadas, enquanto que uma classe derivada proporciona a funcionalidade adicional que especializa seu comportamento.

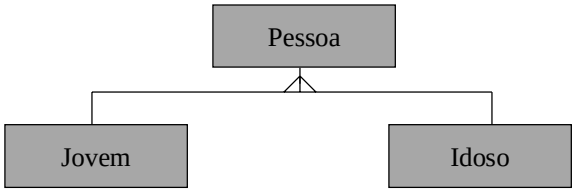
Herança - Exemplo



Hierarquia de Classes de Contas Bancárias

Herança de Comportamento

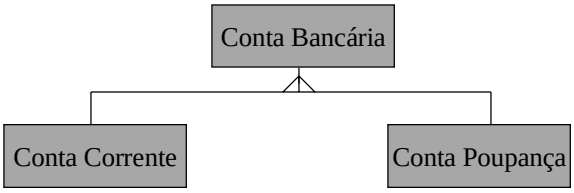
Uma classe S é um subtipo de T se e somente se S proporciona pelo menos o comportamento de T



Restrição
(atributos e métodos iguais - valores de atributos caracterizam novos objetos)

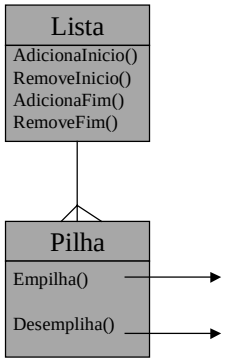
Herança de Comportamento

Uma classe S é um subtipo de T se e somente se S proporciona pelo menos o comportamento de T



Substituição
(métodos iguais - não é preciso redefinir métodos já definidos na super classe)

Herança de Implementação



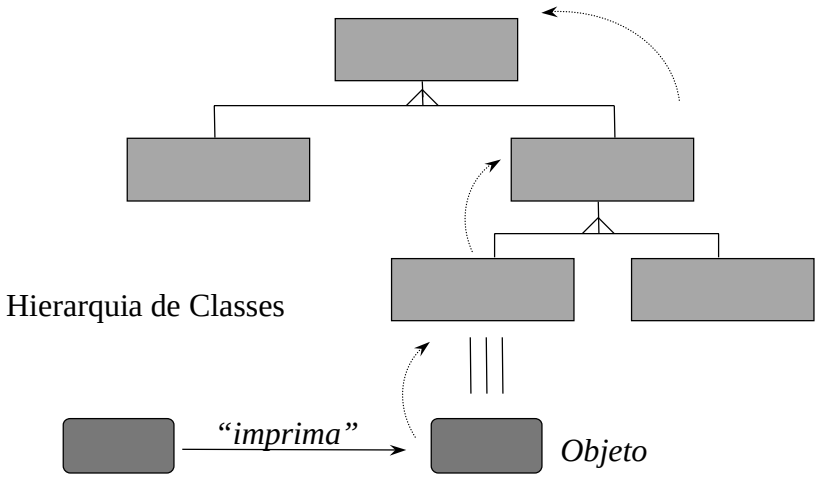
Herança utilizada como uma técnica para implementar uma classe similar a outras já existentes

usa AdicionaFim()

usa RemoveFim()

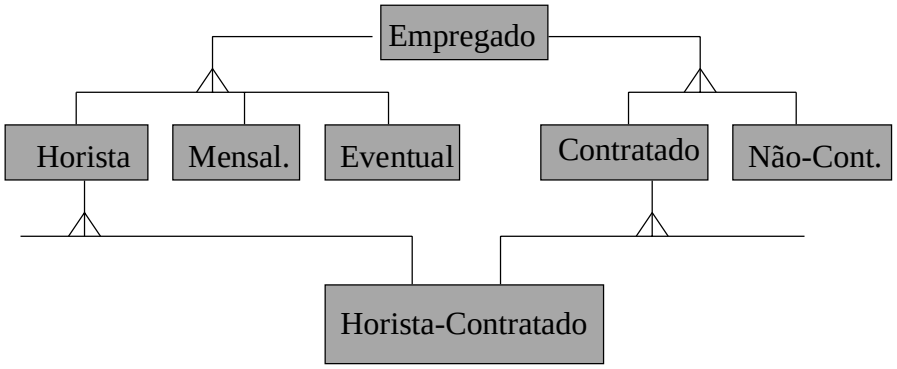
Herança por inclusão
(estruturas iguais - comportamentos diferentes)

Métodos em uma Hierarquia



Herança Múltipla

Permite que uma classe tenha mais que uma superclasse associada e que herde características de todas elas



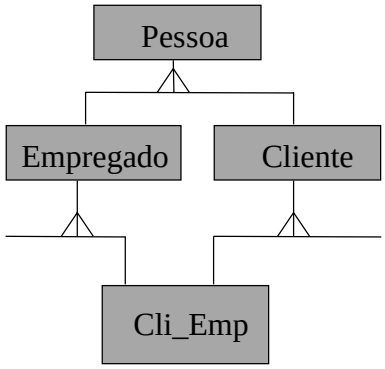
Herança Múltipla em O2

```
Class Pessoa
  type tuple(nome:string, end:Ender)

Class Empregado inherits Pessoa
  type tuple(sal:Money, filho: set(Pessoa) )

Class Cliente inherits Pessoa
  type(credito:Money, status:Boolean)

Classe Cli_Emp inherits Cliente, Empregado
  type(desconto:Money)
```



Herança Múltipla

Vantagens

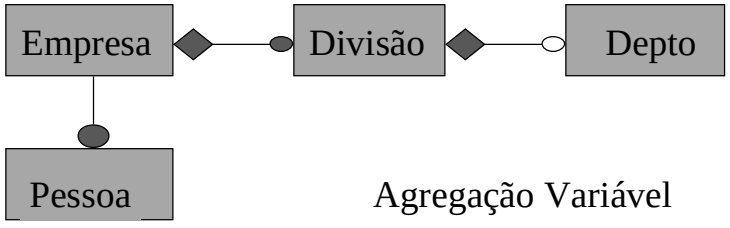
- possibilidade de combinação de informações de diversas fontes
- maior capacidade na especificação de classes
- aumento da possibilidade de reuso

Desvantagens:

- perda da simplicidade conceitual e de implementação

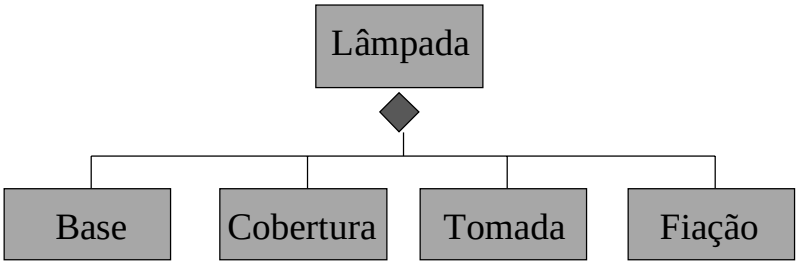
Agregação

Se duas classes estão fortemente relacionadas como parte-todo, então temos uma **agregação**.
Se duas classes são usualmente consideradas como independentes, então é uma associação, mesmo que eventualmente possam ser parte-todo.



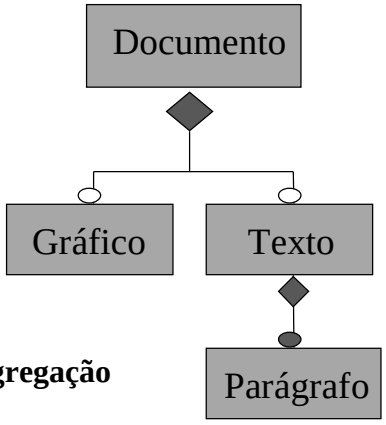
Agregação Variável

Agregação



Agregação Fixa

Agregação



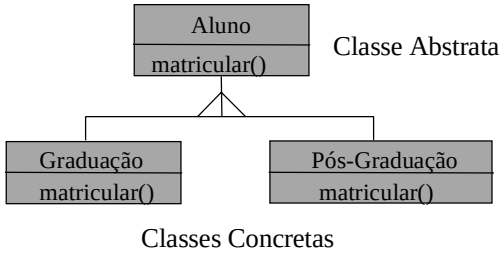
Hierarquia de Agregação

Agregação

Uma **agregação** implica nas seguintes **restrições**:

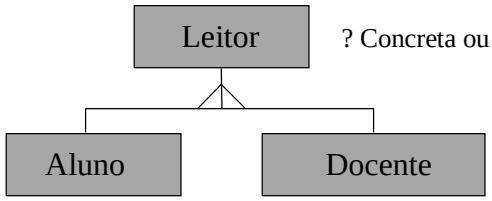
- o tempo de vida dos objetos agregados está ligado ao agregador (os agregados só podem existir se o agregador existe; se o agregador é removido, os agregados também o são)
- os objetos agregados não podem ser compartilhados
- clientes devem acessar os objetos agregados via agregador

Classes Abstratas - Exemplo I



Uma operação abstrata só determina a existência de um comportamento **não definindo uma implementação**

Classes Abstratas - Exemplo II



? Concreta ou Abstrata ?

Classes Concretas

Regra do Negócio: Podem fazer empréstimos na nossa biblioteca, alunos, docentes, funcionários e externos

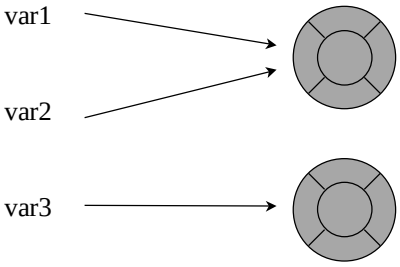
Classes Abstratas

Uma classe abstrata é uma classe que:

- provê organização
- não possui instâncias
- possui uma ou mais operações abstratas
- possui subclasses que implementam estas operações

Variáveis

Variáveis contém referências a objetos:



A variável não é um objeto mas, apenas um apelido para ele.

Tipagem Convencional

```
integer a;

Boolean b;

Float c;

Array v[3];

.......
```

Tipagem

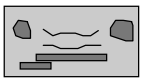
• Em algumas linguagens uma variável pode possuir um tipo

• O tipo determina quais “tipos” de objetos uma variável pode referenciar

• Em muitas linguagens o conceito de **tipo e classe** se misturam

Polimorfismo

texto
imagens
tuplas

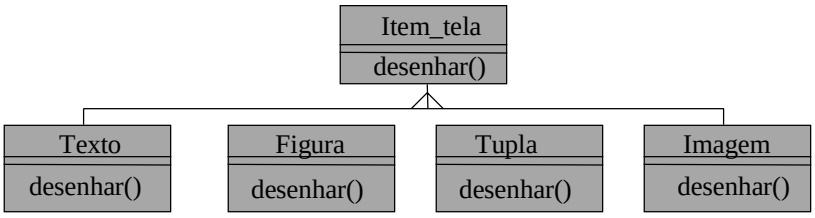


tela multimídia

linguagem tradicional

```
begin case of type(x)
       texto:   desenhar_texto(x)
       imagem:  desenhar_imagem(x)
       tupla:   desenhar_tupla(x)
       ....
       end
end
```

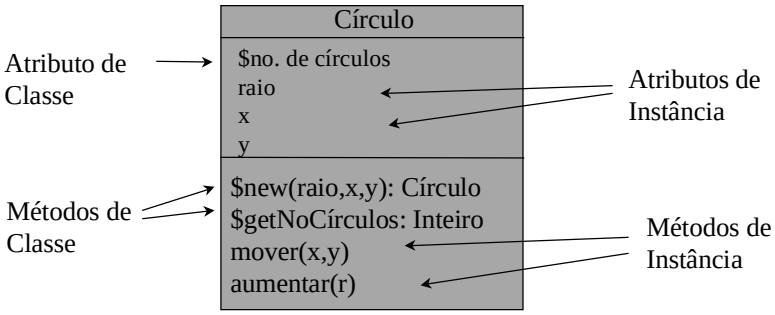
Polimorfismo



. **redefinição** da operação de desenhar (“overriding”)
. **mesmo nome** (desenhar) para 3 funções (“overloading”)

For x in X do x .desenhar(); // “late binding”

Atributos e Métodos de Classe



Notação OMT completa para classe

Atributos e Métodos de Classe

Métodos de classe versus métodos de instância

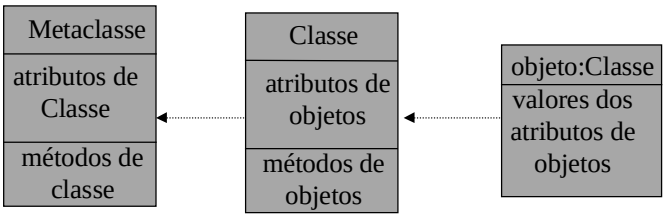
• métodos de classe somente podem ser invocados em uma classe
• métodos de instância somente podem ser invocados em uma instância

Atributos de classe versus atributos de instância

• há apenas uma cópia de um atributo de classe para todos os objetos em uma classe
• todo objeto mantém sua própria cópia de um atributo de instância

Metaclasses

Metaclasse é uma classe que descreve outra classe, isto é, ela é uma classe cujas instâncias são classes



C++ e Java dão apoio para as noções de atributos e métodos de classe através de uma “fusão” do conceito de metaclasse com o de classe

Conceitos Avançados

Reutilização de Projeto

- Design Patterns - padrões de projeto
- Frameworks orientados a objetos

Design Patterns

Padrões de projeto são padrões de organização de hierarquias de classes, protocolos e distribuição de responsabilidades entre classes, que caracterizam construções elementares de projeto orientado a objetos.

Um padrão de projeto é um estrutura que aparece repetidamente nos projetos orientados a objetos para resolver um determinado problema de forma flexível e adaptável dinamicamente.

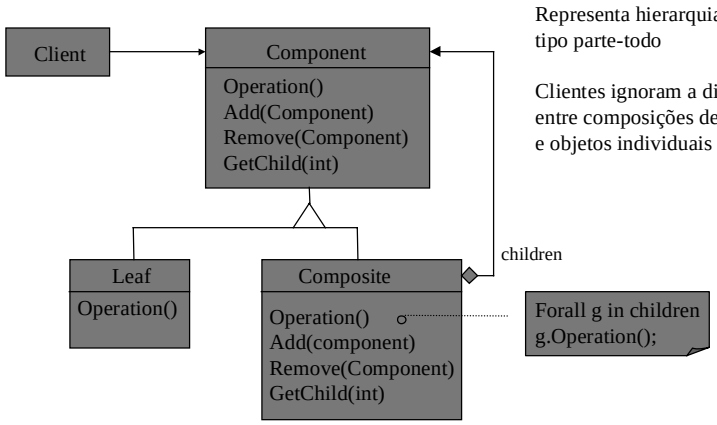
Design Patterns

Descrição de um design pattern:

- objetivo
- motivação
- aplicabilidade
- estrutura
- participantes
- colaborações
- conseqüências
- implementação
- exemplo de codificação

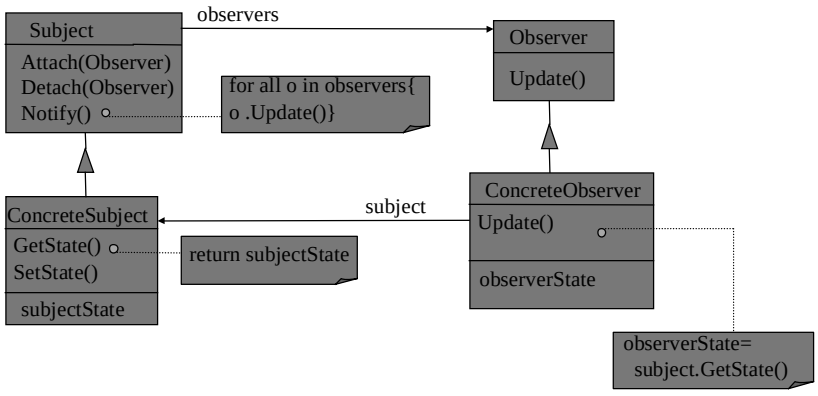
Design Patterns - Exemplo I

Composite



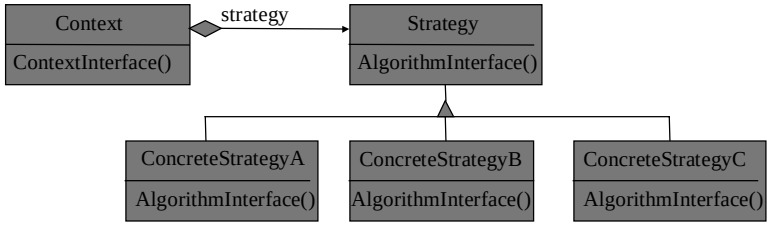
Design Patterns - Exemplo II

Observer: define uma dependência de um para muitos entre objetos de tal forma que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente



Design Patterns - Exemplo III

Strategy: define uma família de algoritmos, encapsula um a um, e os torna intercambiáveis. A classe Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam

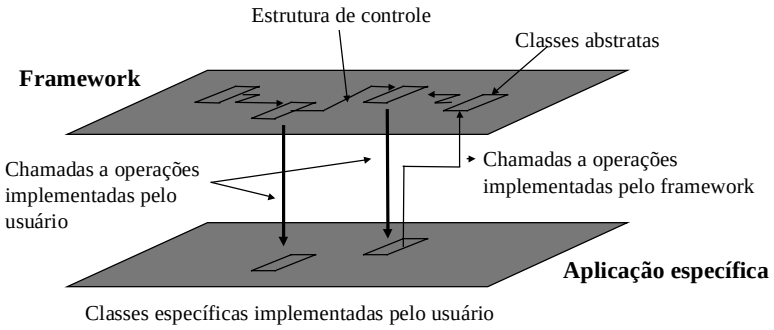


Frameworks

Classes abstratas funcionam como um molde para as suas subclasses.
Da mesma forma, um projeto constituído por classes abstratas funciona como um molde para aplicações.
Um projeto constituído por classes abstratas é denominado **framework de aplicações orientado a objetos.**

Um framework pode ser considerado como uma infra-estrutura de classes que provêem o comportamento necessário para implementar aplicações dentro de um domínio através dos mecanismos de especialização e composição de objetos, típicos das linguagens orientadas a objetos

Frameworks - Visão Conceitual



Visão conceitual da estrutura de um framework

Frameworks - Exemplo

MVC - Model/View/Controller

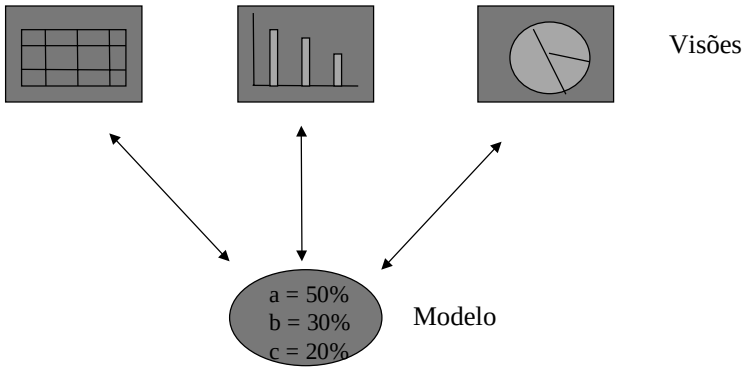
Modelo: contém a lógica do negócio (classes de sistema como gerenciador de dados, gerenciador de comunicações e expedidor que oculta a complexidade de bancos de dados, de interface, middleware, etc...

Visão: contém a apresentação visual baseada no modelo do usuário; a interface é dirigida a eventos (processamento de eventos e formatação de dados)

Controlador: contém o mecanismo de resposta de entradas (manipula a interface do usuário e traduz eventos (Visão) em mensagens (para o Modelo))

Frameworks - Exemplo

MVC - Model/View/Controller



Componentes

Reutilização de software

Um componente é um pedaço de código encapsulado e acessível apenas a partir de sua interface

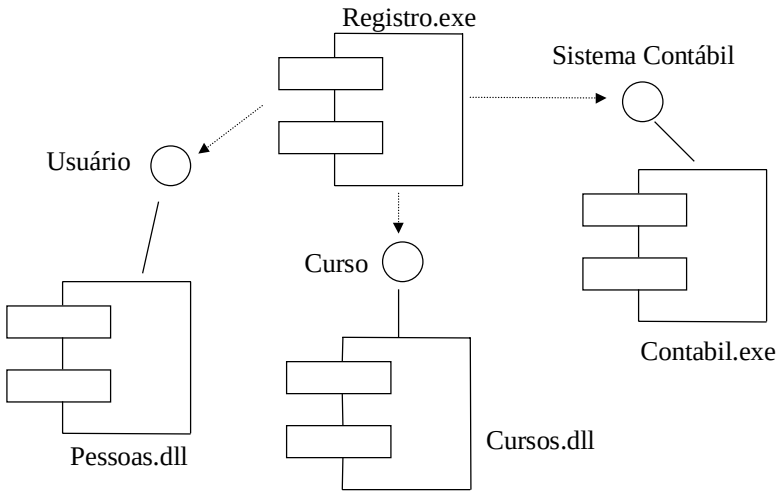
Um componente possui:

• seu **comportamento externo** (o que ele faz) que é definido na sua especificação
• suas **operações internas** (como ele faz o que se propõe a fazer) que está escondido do mundo externo e pode ser entendido apenas se examinarmos seu código fonte
• seu **executável** (runtime binário .exe .dll)

Componentes

Componentes de negócio são essencialmente **objetos.**
Cada componente implementa a **lógica de negócio** e as propriedades relativas a uma entidade do mundo real.
O que os distingue dos objetos tradicionais é a capacidade de ser utilizados por aplicações produzidas em **diferentes linguagens e tecnologias**, rodando sobre diferentes sistemas operacionais.
A tecnologia de componentes altera radicalmente a forma como os sistemas de informação são desenvolvidos.
Os componentes podem ser considerados como blocos básicos de construção.
Para criar um novo sistema, os desenvolvedores apenas combinam componentes.

Diagrama de Componentes (UML)



Wrappers - legacy systems

Um **wrapper** é um componente que fornece serviços implementados por aplicações *legacy*.

Um **wrapper** pode ser utilizado para eliminar as dependências entre os sistemas atuais e fornecer a funcionalidade das aplicações *legacy* para novas soluções baseadas em componentes.

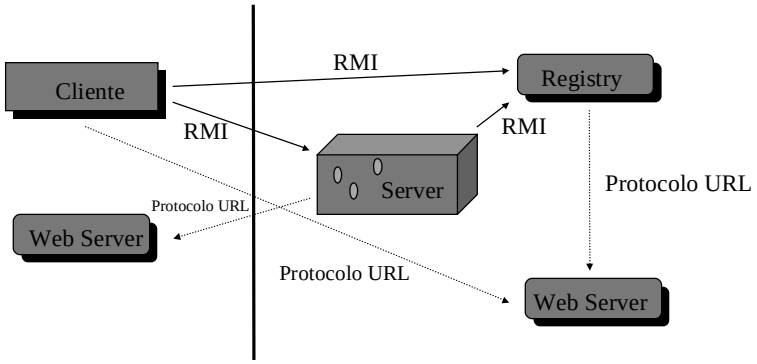
RMI (Remote Method Invocation)

O sistema RMI permite que um objeto rodando em uma máquina virtual Java (JVM) chame métodos de um objeto que esteja rodando em outra JVM.

RMI permite a comunicação remota entre programas escritos em Java.

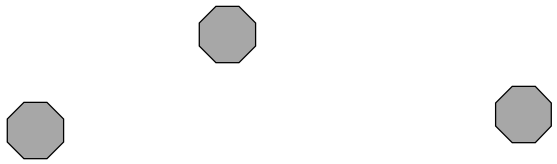
RMI provê um mecanismo através do qual o servidor e o cliente se comunicam e passam informações.
Estas aplicações são chamadas de aplicações de **objetos distribuídos**

RMI

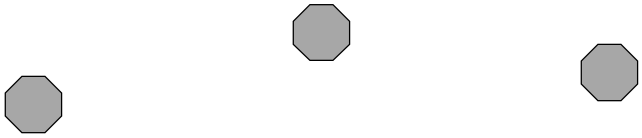


Metodologia OO

Metodologia OO

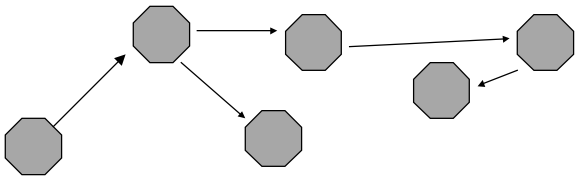


O que faz com que uma metodologia de desenvolvimento de sistemas seja considerada Orientada a Objetos?



Metodologia OO

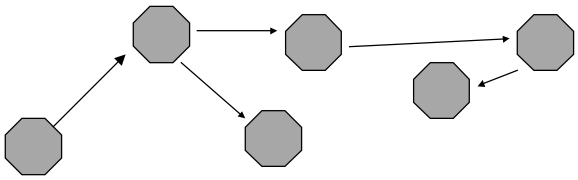
Uma metodologia de desenvolvimento de sistemas é considerada Orientada a Objetos se ela orienta a construção de sistemas a partir do entendimento do mundo real como um conjunto de objetos que comunicam-se entre si de forma coordenada



Metodologia OO

Quais são as principais atividades ?

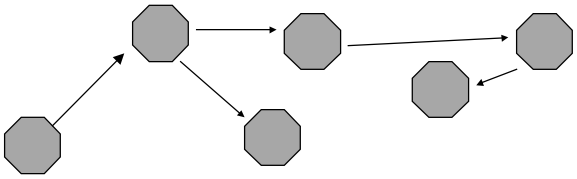
- Entender quais são os **objetos** envolvidos no domínio do problema
- Entender como se **comunicam** no mundo real
- **Projetar** a forma como devem ser **implementados**



Metodologia OO

Quais as principais técnicas utilizadas?

- Entendimento do mundo real - Revisão de processos, Use cases
- Objetos e seus relacionamentos - Modelo de Objetos, CRC, DTE, DI
- Projeto - Padrões de projeto, frameworks, arquiteturas
- Implementação - ambientes de desenvolvimento, middleware, banco de dados



Métodos de Desenvolvimento OO

Booch - Object-Oriented Design with Applications

Wirfs-Brock - Designing Object-Oriented Software (**CRC**)

Rumbaugh - Object-Oriented Modeling and Design (**OMT**)

Coad-Yourdon - Object-Oriented Analysis

Jacobson - OO Software Engineering - A **Use Case** Driven Approach

Shlaer-Mellor - Object **Lifecycles**-Modeling the World in States

Coleman et al: Fusion - OO Development: The **Fusion** Method

UML - Unified Modeling Language

- Inception - Definição do problema, atores, use cases
- Elaboration - modelo de objetos, arquitetura, plano de implementação (interações)
- Construction - implementação de cada interação
- Transition - entrega da aplicação para o usuário

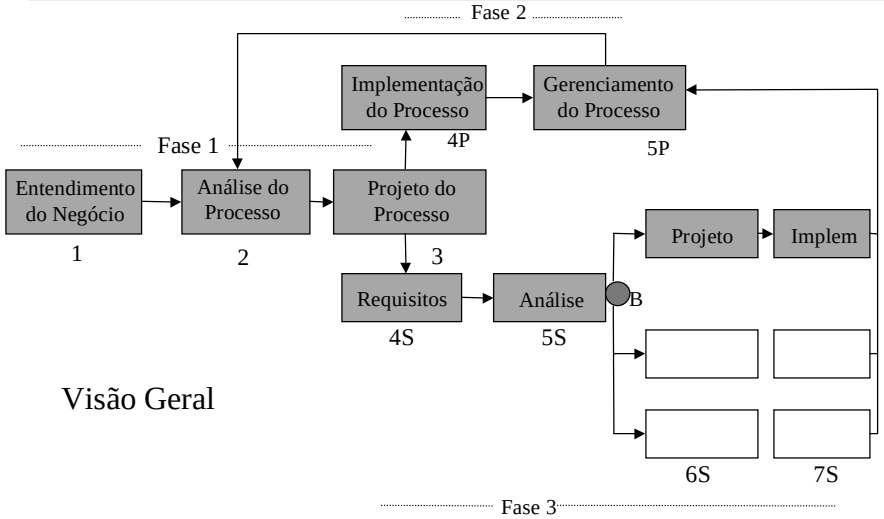
Métodos de Desenvolvimento OO

Estratégia

Escolher um Método como sendo o método principal para ser seguido e ater-se à sua notação para todo o ciclo de vida.

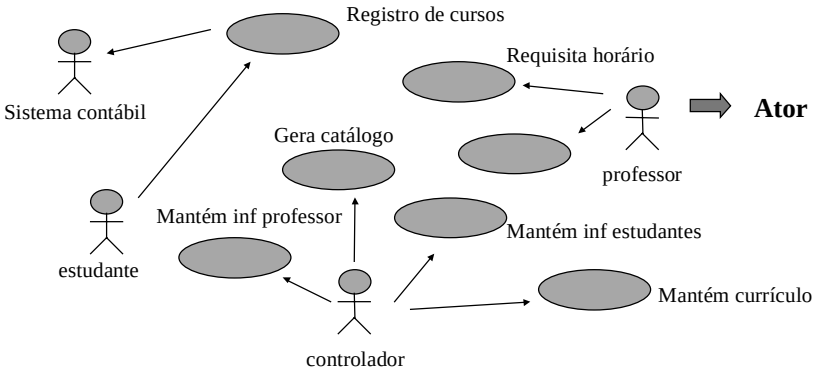
Usar técnicas de outros métodos para dar suporte aos esforços de modelagem e desenvolvimento.

Método Integrado



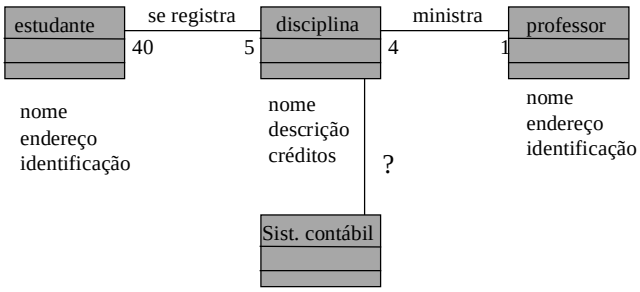
Visão Geral

Análise - Use Case



Registro de cursos: o use case é iniciado pelo estudante. Pelo use case o estudante pode criar, rever, modificar e remover uma disciplina para um dado semestre. Todas as informações contábeis são enviadas para o sistema contábil (4 cenários - criar, rever, remover, adicionar)

Análise - Modelo de Classes



Modelo de classes extraído a partir da análise dos use cases

Análise - CRC

Classe/responsabilidades/colaboradores

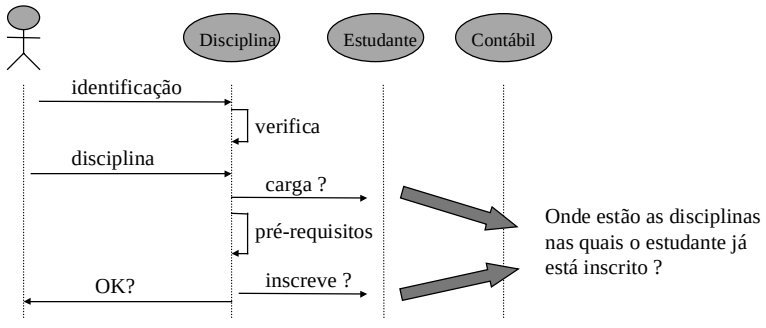
Classe: Disciplina	
Responsabilidades	**Colaboradores**
Saber se ela está lotada Saber os alunos matriculados Saber os professores responsáveis Sabe os pré-requisitos	Estudante Professor

"Feedback" para validar os métodos da classe

Utilizar os *use cases*.
As responsabilidades são "atividades" que devem ser cumpridas pelos métodos da classe em questão, ou, são "ações" que o objeto deve saber executar.

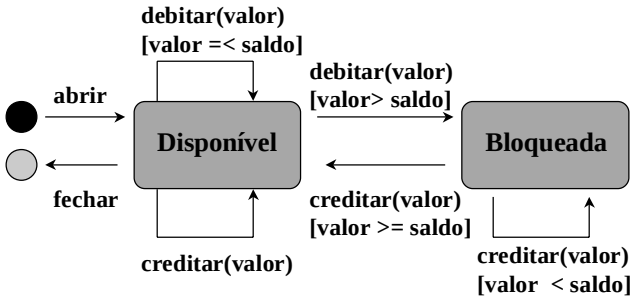
Análise - Diagrama de Interação (DI)

Maneira formal de representar os *use cases*; agora, como já temos o modelo de classes de objetos, já podemos "traduzir" os *use cases* para um diagrama de interação entre classes (utilizar também os CRCs)

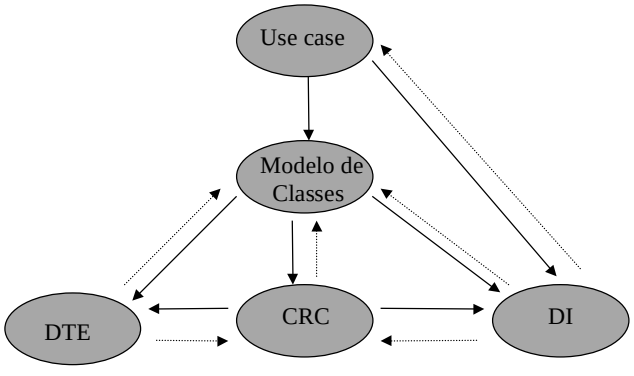


Análise - Diagrama Transição de Estado

Preocupa-se com o comportamento dos objetos no que se refere à mudança de estado desses objetos. É uma incursão novamente para dentro da classe. O DTE de cada classe deve estar compatível com os diagramas de interação. As mensagens que chegam até os objetos devem aparecer no diagrama de transição.

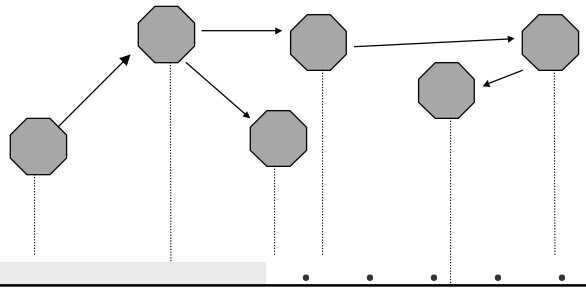


Relacionamento entre as técnicas de Análise



Análise

No final da análise, já sabemos quais os objetos do “negócio” envolvidos e como eles devem interagir de forma geral, através do modelo de classes e do diagrama de interação



Projeto

Será que as classes de negócio são suficientes para a implementação do sistema ?

- O modelo de objetos pode ser otimizado ?
- Como o usuário vai interagir com o sistema ?
- Quem irá controlar o fluxo de mensagens ?
- Quem vai interagir com o banco de dados ?

Projeto

Tanto o domínio do problema, como o domínio da solução devem ser entendidos como um conjunto de classes de objetos

- **Classes de interface** - interação com o usuário
- **Classes de controle** - seqüência das mensagens
- **Classes de banco de dados** - persistência dos dados
- **Classes de comunicação** - mensagens entre objetos

Projeto

Principais atividades de projeto

Projeto de Objetos

- Otimização e refinamento do modelo de classes da análise
- Conversão para o modelo relacional de banco de dados

Projeto de Interface

Define a forma de interação dos usuários com a aplicação sendo construída

Projeto de Sistema

Produz uma arquitetura de aplicação a qual inclui decisões sobre a organização do sistema e a alocação de módulos em componentes de hardware e software

Projeto

Tecnologias de apoio

Projeto de Objetos

- Design Patterns (padrões de projeto)
- Conversão para o modelo relacional
- Recomendações de projeto (Método Integrado)

Projeto de Interface

Utilização segundo um guia de estilo que oriente a utilização correta de recursos gráficos (GUIs), formulários, frames e outros, considerando o ambiente de implementação (por exemplo, Web)

Projeto de Sistema

Arquiteturas de aplicação , frameworks, componentes, packages