



# INTRODUÇÃO A DESENHO ORIENTADO POR OBJETOS COM UML



Nuno Flores, João Pascoal Faria  
MIEIC, LPOO, 2016/17

## Índice

2

- ☐ Introdução a desenho de software
- ☐ Introdução a diagramas de classes
- ☐ Introdução a diagramas de estados
- ☐ Introdução a diagramas de sequência
- ☐ \*Referência sobre diagramas de classes
- ☐ \*Referência sobre diagramas de estados
- ☐ \*Referência sobre diagramas de sequência

## Níveis de desenho

- Desenho de alto nível ou de arquitetura
  - ▣ Partir o sistema em **componentes** (bibliotecas, etc.) que podem ser desenvolvidos em paralelo e subsequentemente integrados, bem como identificar oportunidades de reutilização
  - ▣ Normalmente intercalado com atividades de identificação de requisitos
- ⇒ □ Desenho detalhado, tipicamente, desenho orientado por objetos
  - ▣ Partir cada componente em **classes**
  - ▣ Normalmente intercalado com atividades de codificação e teste
- Desenho de algoritmos e estruturas de dados
  - ▣ Normalmente realizado dentro de cada classe

# Desenho orientado por objetos (OOD)

5

- Conceção de soluções de software orientadas por objetos
  - Visão do 'mundo' como um conjunto de objetos com identidade, estado e comportamento, interagindo entre si através de trocas de mensagens
- Requer notações (**UML**) e boas práticas.
- Pode ser abordada por 4 vistas, suportadas por diversos diagramas UML

	Estrutura estática	Comportamento dinâmico
Externa	Diagramas de classes (só API pública), pacotes e objetos	Interação entre objetos (diagramas de sequência, etc.)
Interna	Diagramas de classes (todos os elementos públicos e privados)	Ciclos de vida de objetos (diagramas de estados)

6

## Introdução a diagramas de classes

# Diagramas de classes

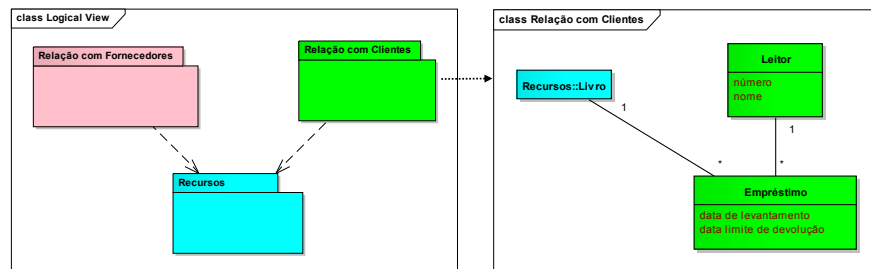
7

- Mostram classes, respectivos atributos e operações, e relações entre classes
- Podem ser usados com diversas finalidades:
  - Descrever o vocabulário num domínio (classe = conceito)
  - Projetar uma bases de dados (classe = entidade informacional)
- ⇒ □ Descrever a estrutura de uma solução orientada por objetos, a construir ou existente (classe UML = classe de implementação)

# Diagramas de pacotes

8

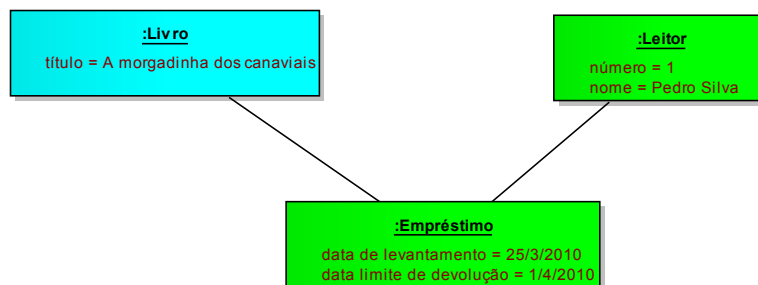
- Mostram pacotes (*packages*) e suas dependências
- Pacotes são um mecanismo de agrupamento genérico
- Interessa-nos aqui para agrupar classes
- Sugestão: Usar uma cor diferente para cada pacote



# Diagramas de objetos


9

- Mostram objetos (instâncias de classes), ligações (instâncias de associações) e valores de atributos
- Permitem ilustrar cenários/configurações particulares



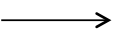



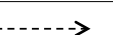

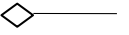

# Resumo de notação - Nós

10

Notação	Descrição
<div>ClassName</div>	Classe
<div>«interface» InterfaceName</div>  InterfaceName	Interface
<div>ClassName</div> <div>Param</div>	Classe parametrizada (genérica)
<div></div> <div>PackageName</div>	Pacote (package)
<div><u>instanceName: ClassName</u></div>	Instância de classe (objeto)

## Resumo de notação - Relações

11

Notação	Descrição
	Associação navegável (referência para objeto(s))
	Generalização (extends)
	Concretização (implements) Notação alternativa: 
	Dependência
	Agregação
	Composição
	Nesting (classe definida dentro doutra)

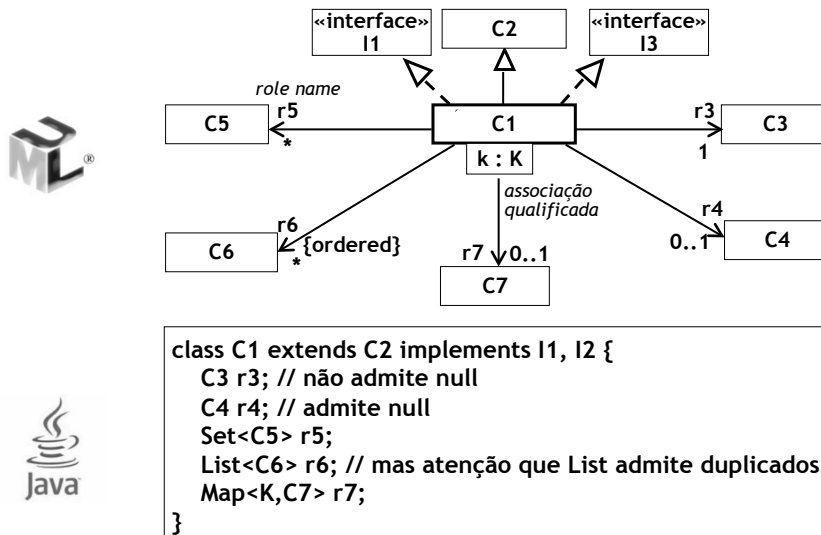
## Resumo de notação - Outros

12

Categoria	Notação	Descrição
Modificadores	<i>itálico</i>	classes e operações abstratas (abstract)
	<u>sublinhado</u>	atributos e operações estáticos (static)
	/nome	prefixo para atributo derivado (calculado)
Visibilidade	+ - # ~	public, private, protected, package
Multiplicidade	*, 0..*, 1, 0..1, 1..*	0 ou mais, 0 ou mais, 1, 0 ou 1, 1 ou mais
Restrições	{restrição}	Restrição sobre elemento do modelo (classe, atributo, extremo de associação, etc.)

## Mapeamento para Java

13



14

## Introdução a diagramas de estados

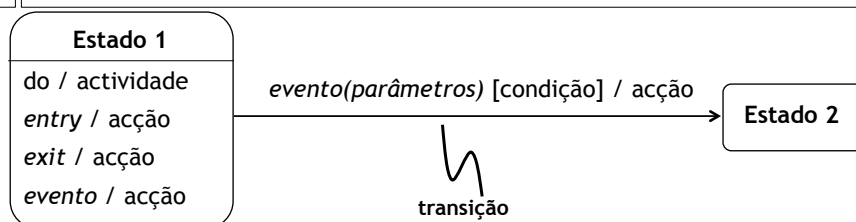
# Diagramas de estados

15

- Permitem modelar o **ciclo de vida** de objetos ou sistemas, vistos como **máquinas de estados**:
  - ▣ **Estados** possíveis (duradouros, finitos) de um objeto ou sistema
  - ▣ **Transições** (instantâneas) entre estados
  - ▣ **Eventos** que desencadeiam as transições
  - ▣ (Opcional) **Ações** (instantâneas) realizadas pelo objeto ou sistema em resposta à ocorrência de um evento
  - ▣ (Opcional) **Atividades** (duradoiras) realizadas pelo objeto ou sistema durante a permanência num estado

## Notação básica

16



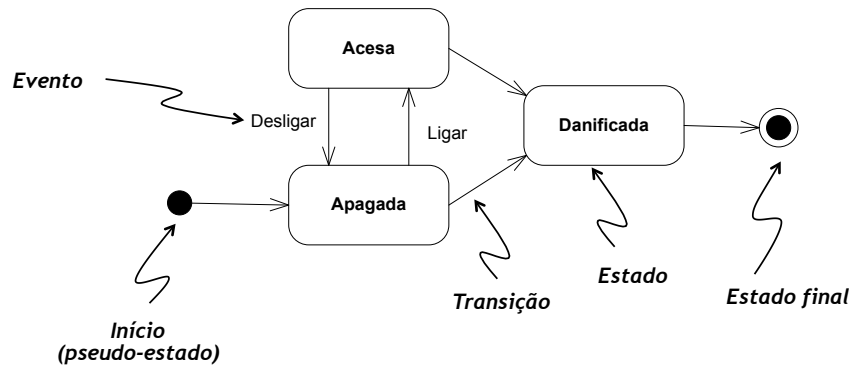
- Sequência de mudança de estado:
  - Ocorre o evento associado à transição e a condição de guarda é verdadeira
  - É interrompida a atividade associada ao estado de origem, se não tinha já terminado
  - É executada a ação à saída do estado de origem
  - É executada a ação associada à transição
  - É executada a ação à entrada do estado de destino
  - É iniciada a atividade associada ao estado de destino



## Exemplo básico

17

- Diagrama de estados de uma lâmpada



## Diferenças para autómatos finitos

18

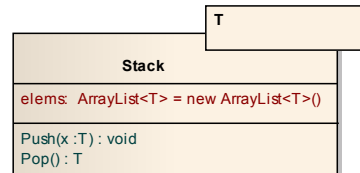
- Variáveis de estado além do diagrama
  - Variáveis de estado são as variáveis de instância do objeto
  - Diagrama só mostra um número finito de estados de alto nível, em que o comportamento do objeto/sistema é significativamente distinto
    - Por exemplo, porque o conjunto de operações disponíveis é diferente
- Estados compostos
  - Composição "ou"
  - Subestados sequenciais
  - Evita explosão combinatória de transições
- Regiões ortogonais
  - Composição "e"
  - Subestados concorrentes
  - Evita explosão combinatória de estados

## Exemplo com variáveis de estado adicionais: Stack (1/2)

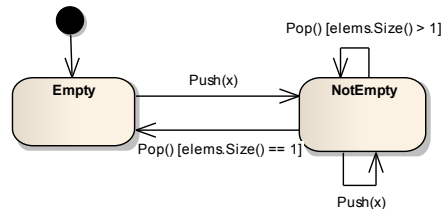
19

- O estado completo da Stack é dado pela valor das variáveis de instância
  - ▣ “elems” neste caso
  - ▣ pode ter infinidade de valores
- No diagrama de estados, apenas se distinguem estados com comportamento significativamente diferente
- Formalmente, é máquina de estados estendida
  - ▣ Máquina de estados finita + variáveis de estado adicionais (elems)

class Logical View



stm StateMachine

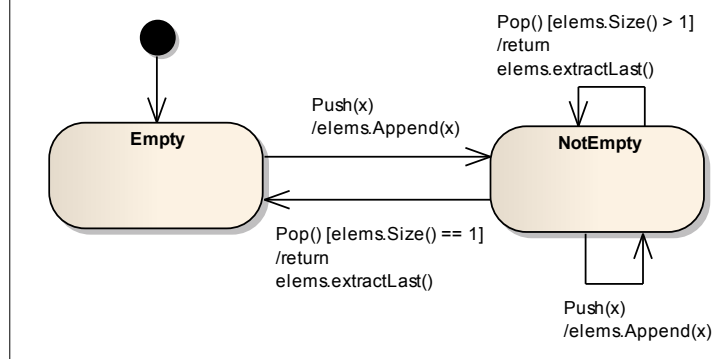


## Exemplo com variáveis de estado adicionais: Stack (2/2)

20

- Agora completo com acções (opcional, por ser pouco escalável)

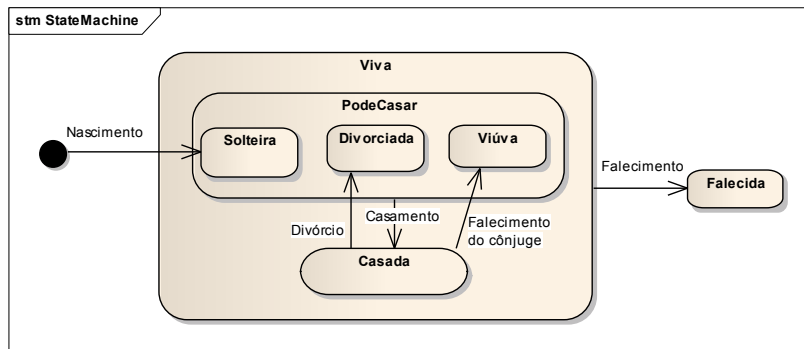
stm StateMachine



## Exemplo com estados compostos

21

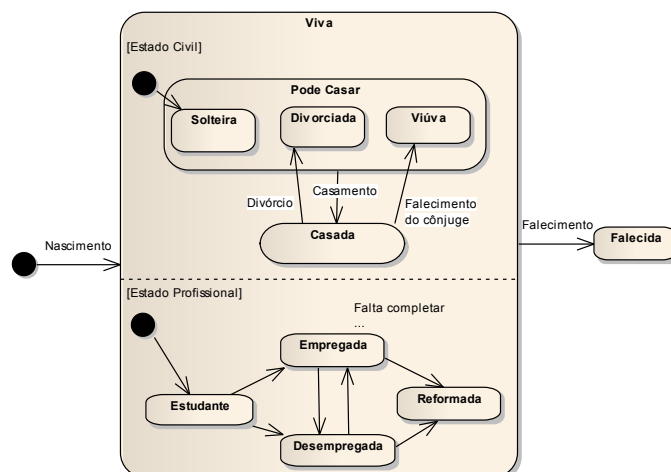
- Estado civil de Pessoa
- Estados compostos evitam explosão combinatória de transições



## Exemplo com regiões ortogonais

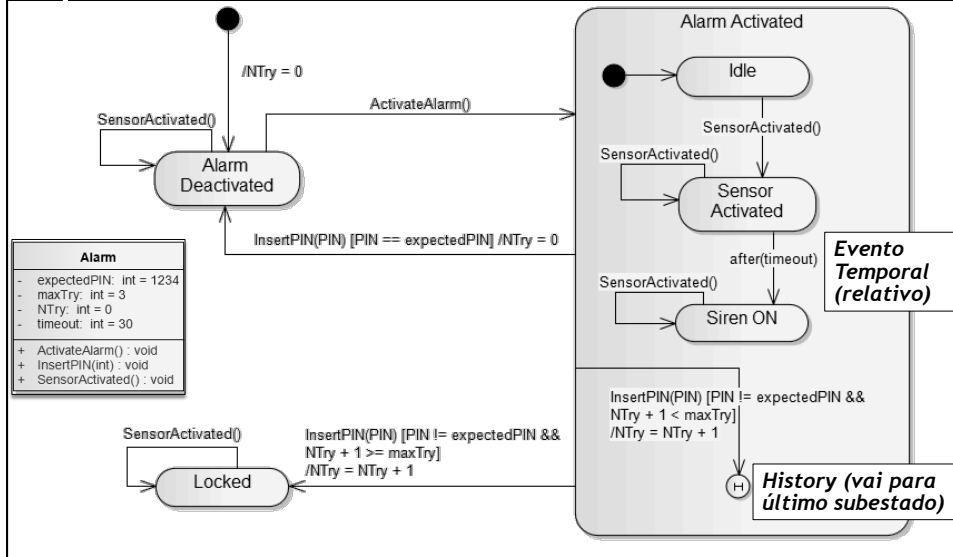
22

- Estado civil e estado profissional de Pessoa são ortogonais
- Evita explosão combinatória de estados (produto cartesiano)



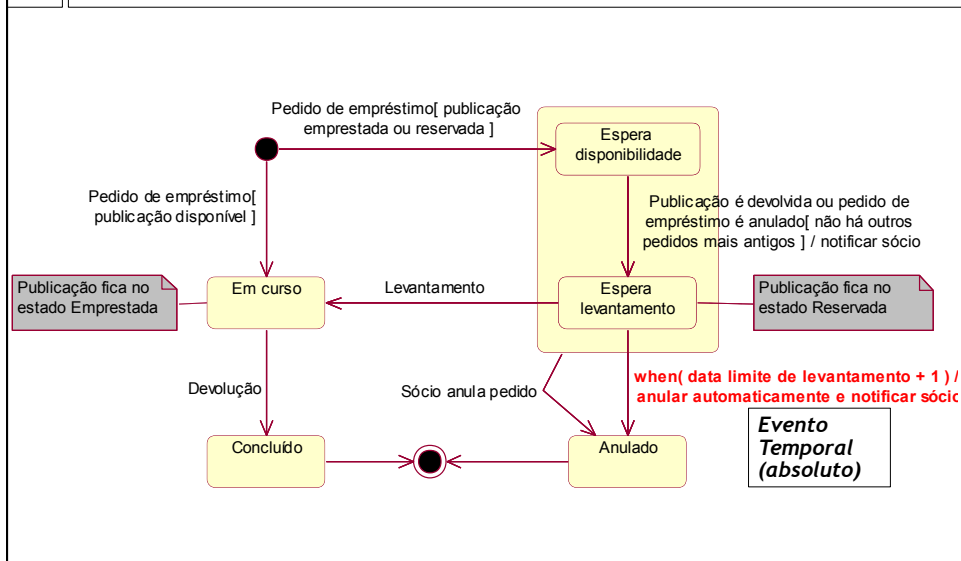
## Exemplo com eventos temporais relativos: Sistema de alarme doméstico

23



## Exemplo com eventos temporais absolutos: empréstimo de livros

24



## Outros exemplos de aplicação

25

- Interface para utilizador
  - ▣ Mapa de navegação da interface para o utilizador
  - ▣ Componentes interativos
- Sistemas controlados por computador
  - ▣ Semáforos (com eventos temporais)
  - ▣ Elevador (sistema híbrido)
- Processos/objetos de negócio
  - ▣ Requisição, Compra
- *Parsers*
  - ▣ Tirar comentários

26

## Introdução a diagramas de sequência

# Diagramas de sequência

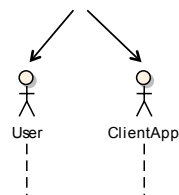
27

- Mostram uma sequência de interação (troca de mensagens) entre intervenientes num dado contexto
  - ▣ Apresentam uma visão dinâmica ou de comportamento que complementa a visão estática ou de estrutura
  - ▣ Fazem a ligação entre funcionalidade e estrutura
  - ▣ Servem de base para especificação de testes
- Interação
  - ▣ Interação entre objetos/componentes no programa/sistema
  - ▣ Interação com o utilizador e/ou com outros programas/sistemas
- Contexto
  - ▣ Caso de utilização; Cenário de utilização (variante de caso de util.)
  - ▣ Operação de uma classe; Mecanismo; Cenário de teste

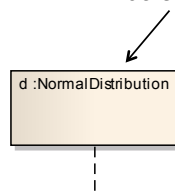
## Intervenientes (*lifelines*)

28

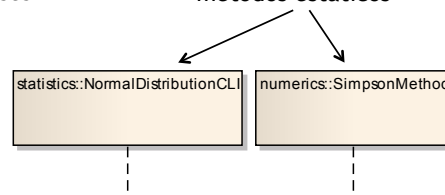
Actores - Utilizadores ou sistemas externos



Objetos - Instâncias de classes



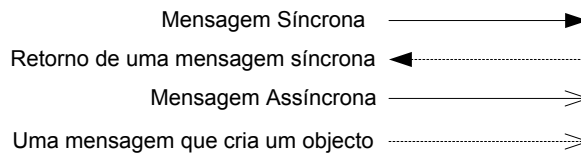
Classes - Para invocar métodos estáticos



## Mensagens - Tipos

29

- Comunicação entre dois intervenientes
  - Pode veicular informação, desencadear ações e mudanças de estado
- Tipos :
  - *Call*: Chamada de operação, normalmente síncrona (emissor bloqueia)
  - *Reply*: Retorno de chamada
  - *Send*: Envio de sinal, normalmente assíncrono (emissor não bloqueia)
  - *Create*: Criação de objeto (invocando construtor), normalmente síncrona
  - *Destroy*: Destruição de objeto (e.g., delete em C++)



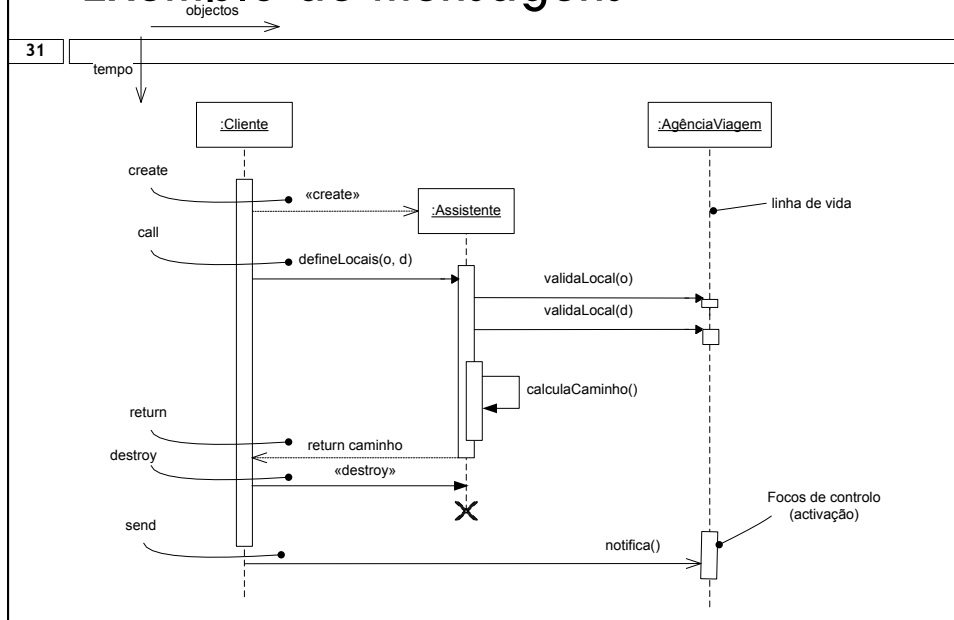
## Mensagens - Texto

30

- Texto da mensagem:

```
[attribute=] signal-or-operation-name [(arguments)] [:ret-val]
arguments ::= argument [, arguments]
argument ::= [parameter-name= ] argument-value | -
attribute ::= out-parameter-name [:argument-value] | -
```
- Exemplos
  - `cancela()`
    - Envio de uma operação sem valores em argumentos e sem retorno.
  - `cancelaProposta(data=31/12/2006, -): void`
    - Envio de uma operação com o primeiro argumento com valor "31/12/2006", o segundo argumento sem valor e com retorno do tipo "void".

## Exemplo de mensagens



## Fragmentos Combinados

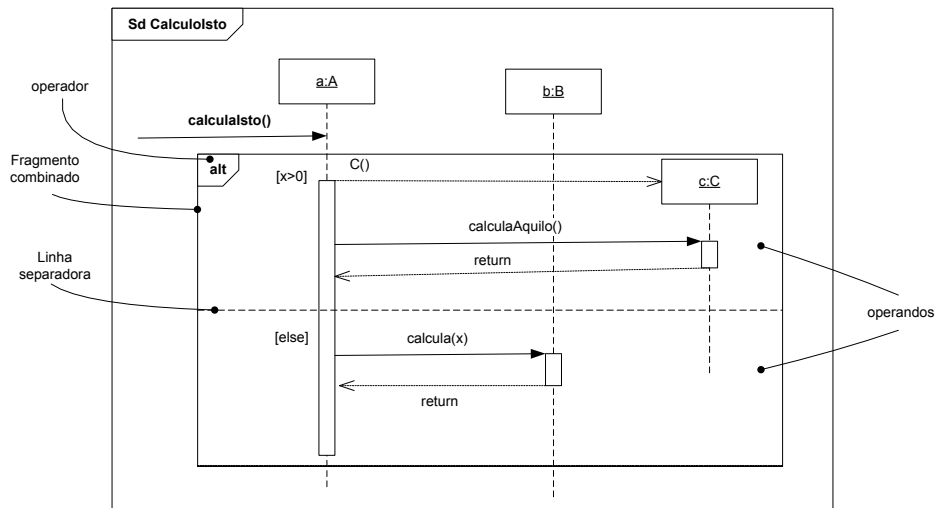
32

- Agregam um ou mais pedaços de interacção (operandos)
- Tipos (operadores) mais importantes:
  - ▣ **seq (sequência)** - Sequência de operações ordenadas fracamente (por linha de vida)
  - ▣ **strict (estrita)** - Sequência de operações ordenadas estritamente .
  - ▣ **alt (alternativa)** - Comportamentos alternativos segundo condições
  - ▣ **opt (opção)** - Execução segundo condição de guarda
  - ▣ **par (paralelo)** - Execução em paralelo de conjunto de operações
  - ▣ **loop (iteração)** - Ciclos de execução
  - ▣ **critical, neg, assert, consider, ignore** – Mais raros (ver especificação UML)



## Exemplo de fragmento combinado

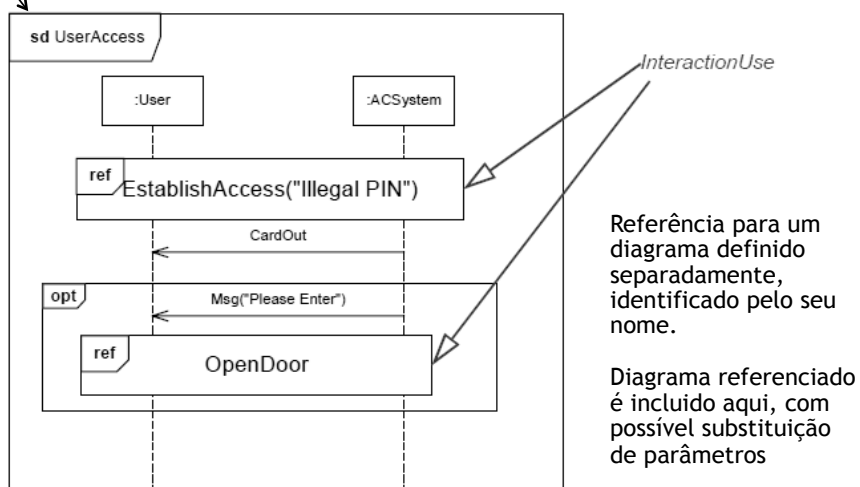
33



## Reutilização de interações

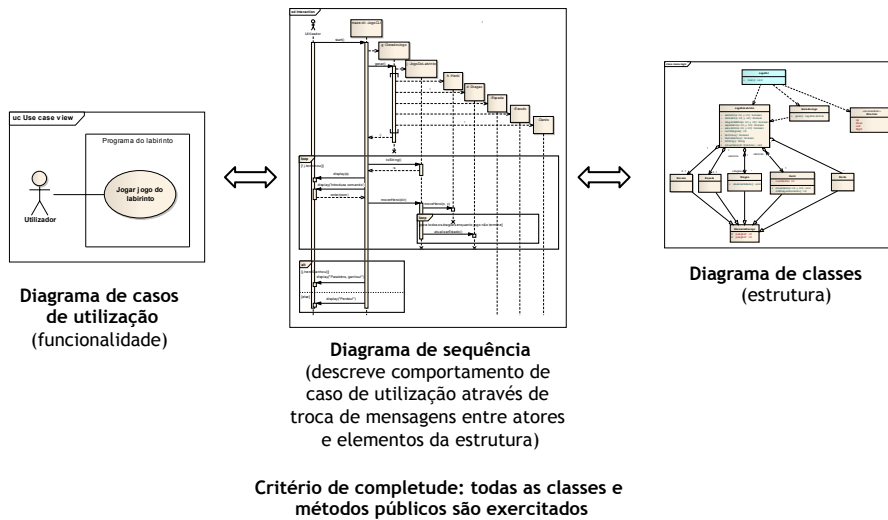
34

Moldura (frame)



## Diagramas de sequência fazem a ligação entre funcionalidade e estrutura

35



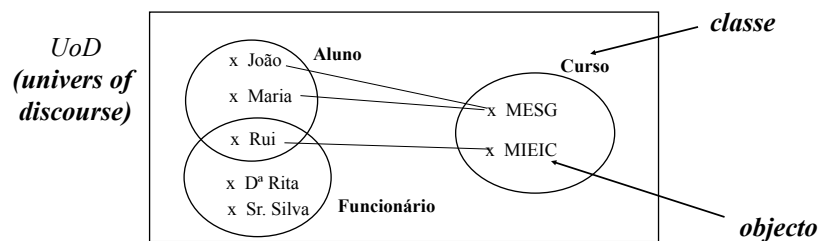
36

\*Referência sobre diagramas de classes

# Objetos versus classes

37

- Objeto = indivíduo = instância de classe
- Classe = tipo de objeto
- ⇒ □ Definição em intenção: por propriedades comuns às instâncias
  - Definição em extensão: por enumeração das instâncias



## Classes

Aluno

Curso

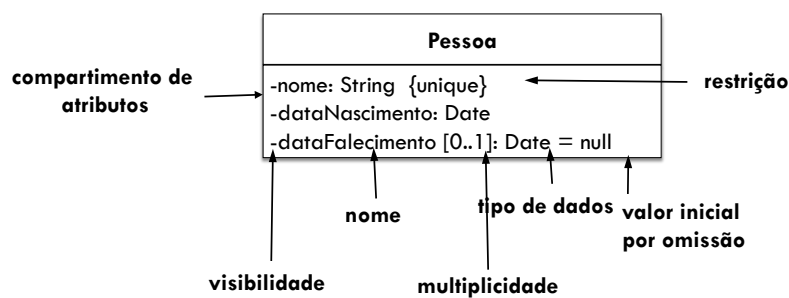
38

- Interessa-nos identificar as classes e não os indivíduos
  - João, MIEIC — Aluno, Curso
- Usualmente aparecem como substantivos comuns
  - Um aluno só se pode inscrever num curso
- Classes podem representar, e.g.,
  - **Coisas concretas:** Pessoa, Turma, Carro, Prédio, Fatura, Livro
  - **Papéis que coisas concretas assumem:** Aluno, Professor
  - **Eventos:** Aula, Acidente
  - **Especificações:** Ficha de Produto, Plano de Estudos de Curso
  - **Tipos de dados:** Data, Árvore Binária de Pesquisa
- Convenção de nomes: singular, capitalizado

## Atributos (de instância)

39

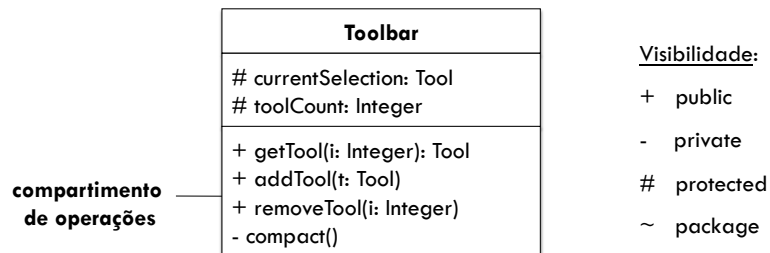
- ❑ Servem para descrever e representar o estado dos objectos
- ❑ Objectos da mesma classe têm os mesmos atributos
- ❑ Os nomes dos tipos de dados não estão pré-definidos em UML, podendo-se usar os da linguagem alvo



## Operações (de instância)

40

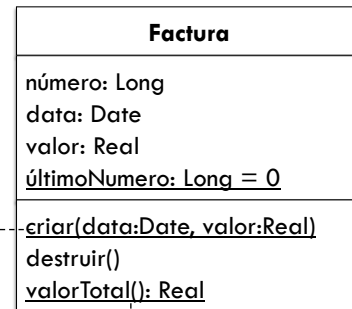
- ❑ Servem para definir o comportamento dos objetos
- ❑ Objetos da mesma classe têm as mesmas operações
- ❑ Estado interno do objeto deve ser escondido e manipulado por intermédio de operações
  - ❑ permite alterar representação do estado sem afetar clientes
  - ❑ permite validar alterações de estado e garantir integridade do objeto



## Atributos e operações estáticos (≠ de instância)

41

- Atributo estático: com um único valor, definido ao nível da classe e não das instâncias
- Operação estática: invocada ao nível da classe e não das instâncias
- Notação : sublinhado
- Correspondem a membros static em C++, Java e C#

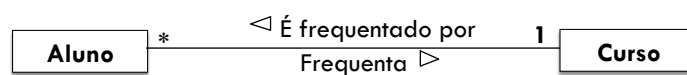


retorna a soma dos valores de todas as facturas

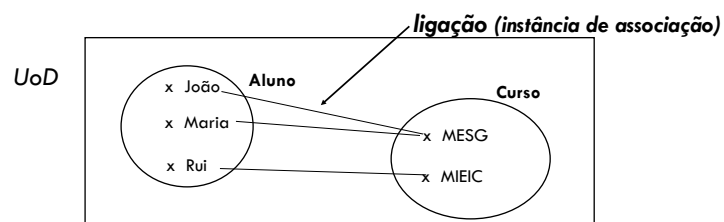
cria nova factura com a data e valor especificados, e um n° sequencial atribuído automaticamente com base em ultimoNumero

## Associações binárias

42



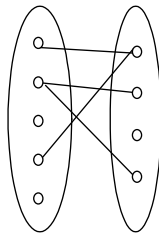
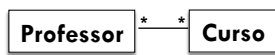
- Uma associação binária define um tipo de ligação que se pode estabelecer entre objetos de 2 classes
- Pode ter nome (ao centro) e papéis e multiplicidade dos objetos participantes (em cada extremo)



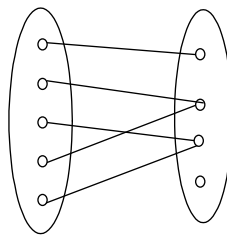
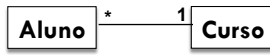
## Associações binárias: multiplicidade

43

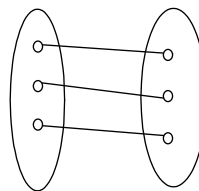
### Muitos-para-Muitos



### Muitos-para-1



### 1-para-1



#### Notação:

1 - exactamente um  
 0..1 - zero ou um (zero a 1)  
 \* - zero ou mais

0..\* - zero ou mais  
 1..\* - um ou mais  
 1, 3..5 - um ou três a 5

## Navegabilidade das associações

44

- Sentido de navegação importante para definir como é implementada a associação, i.e., que objetos guardam referência para quais



```
class C1 {
    C2 c2;
}
```

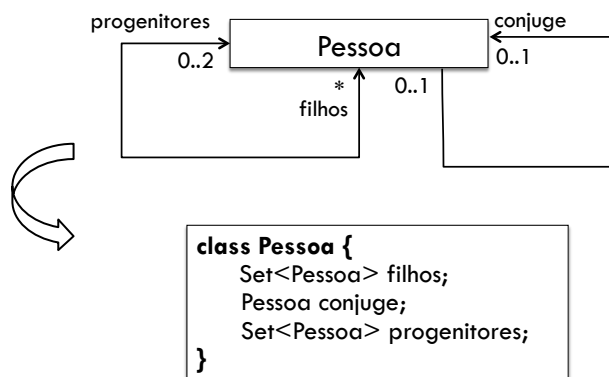
```
class C2 {
    Set<C3> colC3;
}
```

```
class C3 {
    Set<C2> colC2;
}
```

## Associação reflexiva

45

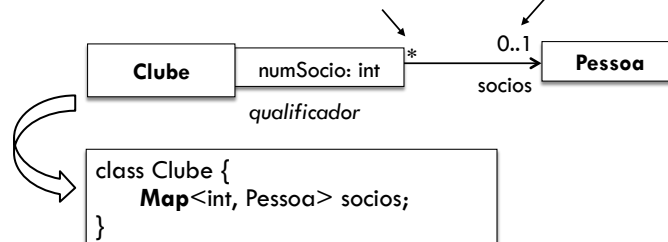
- Pode-se associar uma classe com ela própria, normalmente em papéis diferentes



## Associação qualificada

46

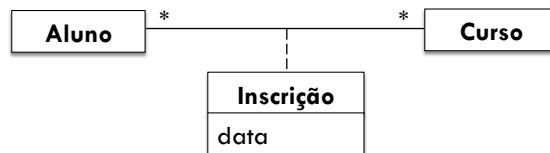
*uma Pessoa pode ser sócia de vários clubes*      *para cada par Clube + n° de sócio*



- Qualificador: lista de um ou mais atributos de uma associação utilizados para navegar de A para B
- "Chave de acesso" a B (acesso a um objeto ou conjunto de objetos) a partir de um objeto de A

## Classe-associação

47

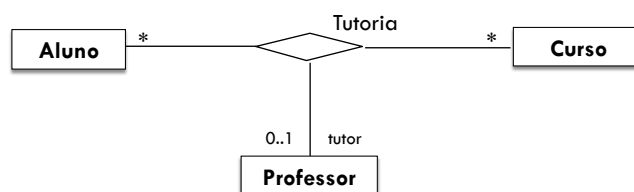


- Reúne as propriedades de associação e classe
- Permite acrescentar atributos a associações

## Associação n-ária

48

- Associação entre 3 ou mais classes



Cada instância da associação é um terno (P, A, C) indicando que o professor P é tutor do aluno A no curso C.

- A cada par (aluno, curso) pode corresponder 0 ou 1 tutor.
- A cada par (aluno, tutor) podem corresponder vários cursos.
- A cada par (tutor, curso) podem corresponder vários alunos.



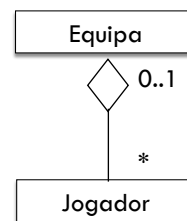
# Agregação

49

- Forma especial de associação com o significado contém / faz parte de

- Uma equipa contém 0 ou mais jogadores

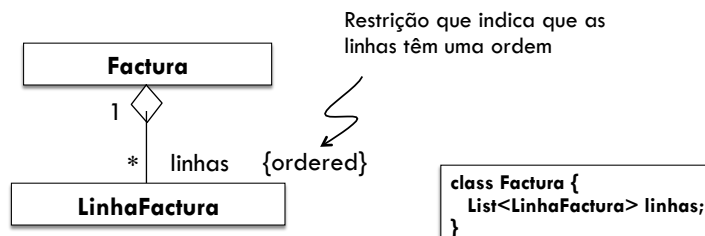
- Um jogador faz parte de uma equipa, mas também pode estar desempregado



# Composição

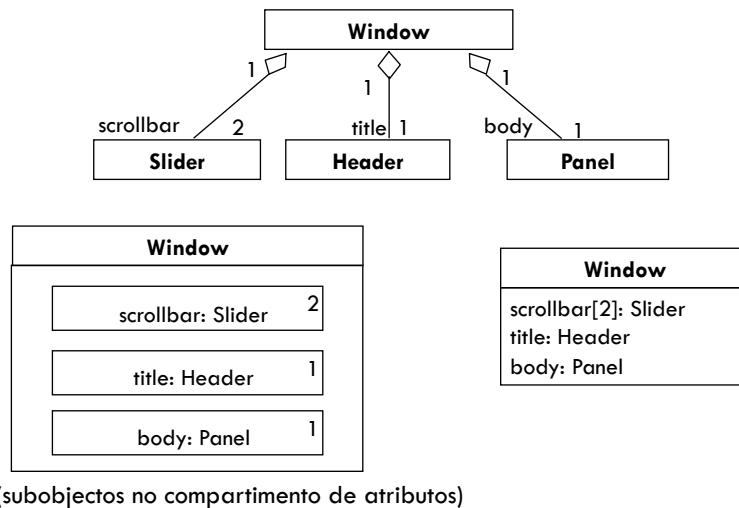
50

- Forma mais forte de agregação
  - existe um forte grau de pertença das partes ao todo
  - cada parte só pode fazer parte de um todo
  - a eliminação do todo propaga-se para as partes, em cascata
- Em C++: membro-objeto (em vez de apontador p/ objeto)



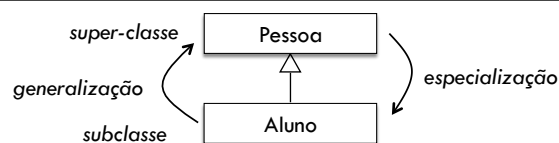
# Composição: notações alternativas

51

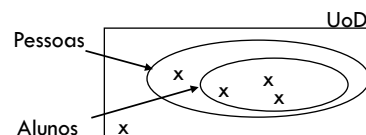


# Generalização

52

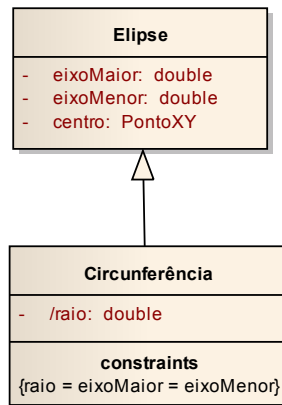


- Relação semântica “is a”: um aluno é uma pessoa
- Relação de inclusão ( $\subset$ ) nas extensões das classes
- Relação de herança nas propriedades: a subclasse herda as propriedades (atributos, operações e relações) da super-classe, podendo acrescentar outras

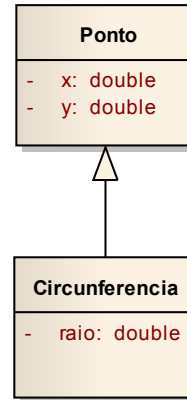


## Exemplos de generalização

53



Conceptualmente correcto, mas herança é pouco conveniente

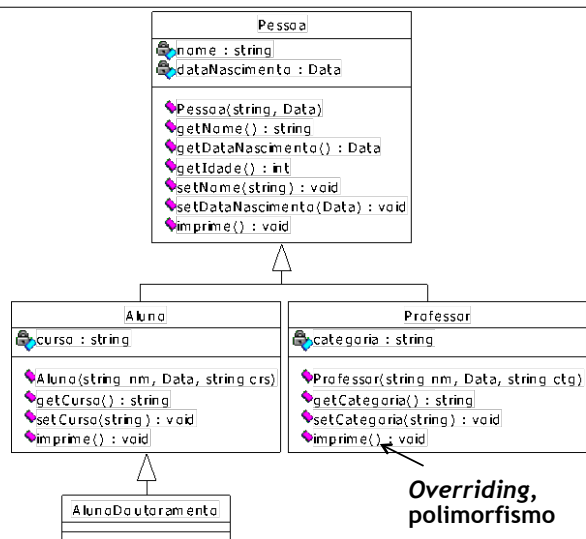


Conceptualmente incorrecto, ainda que herança seja conveniente

## Hierarquias de classes

54

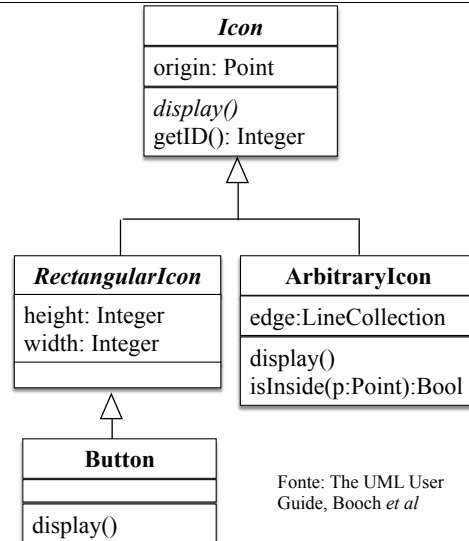
- Hierarquia de classes relacionadas por herança
  - em cada classe colocam-se as propriedades comuns a todas as subclasses
  - evita-se redundância, promove-se reutilização!



## Classes e operações abstratas (≠concretas)

55

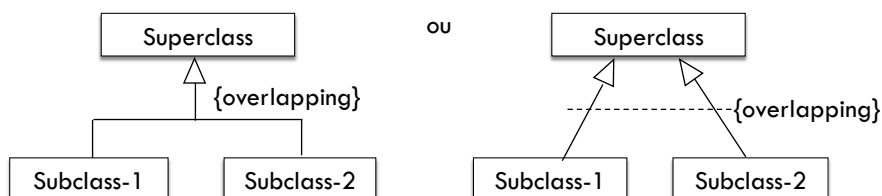
- Classe abstrata: não pode ter instâncias diretas
  - pode ter instâncias indiretas pelas subclasses concretas
- Operação abstrata: implementação a definir nas subclasses
  - a classe respectiva tem de ser abstrata
  - função virtual pura em C++
- Notação : nome *itálico* ou marca {abstract}



## \*Subclasses sobrepostas (≠disjuntas)

56

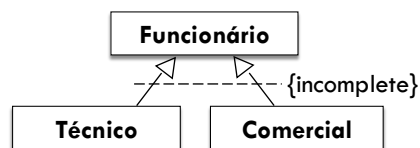
- Caso em que um objecto da superclasse pode pertencer simultaneamente a mais que uma subclasse
- Indicado por restrição {overlapping}
- Geralmente não suportado por ling. de prog. OO
  - Pode ser simulado por agregação de papéis
- O contrário é {disjoint} (situação por omissão?)



## \*Subclasses incompletas (≠completas)

57

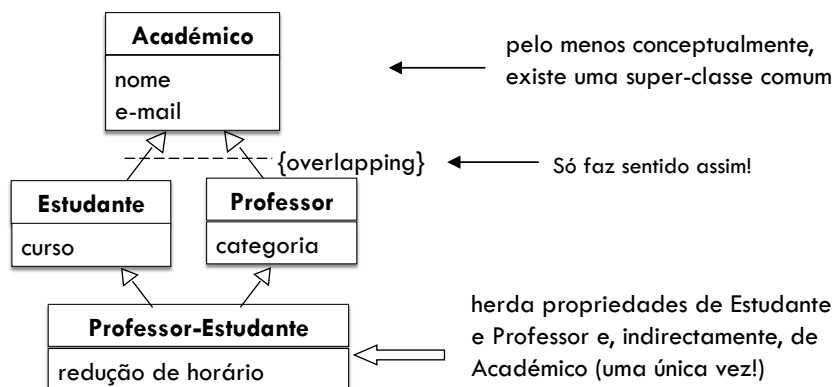
- Caso em que um objecto da superclasse pode não pertencer a nenhuma das subclasses
- Indicado por restrição {incomplete}
- O contrário é {complete} (situação por omissão?)
  - Pode-se forçar marcando superclasse como abstracta



## Herança múltipla (≠simples)

58

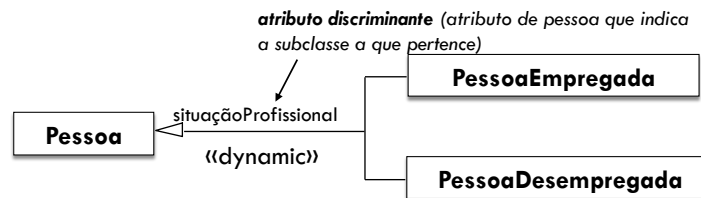
- Classe com múltiplas super-classes
- Suportada em C++ mas não em Java nem C#



## \*Classificação dinâmica (≠estática)

59

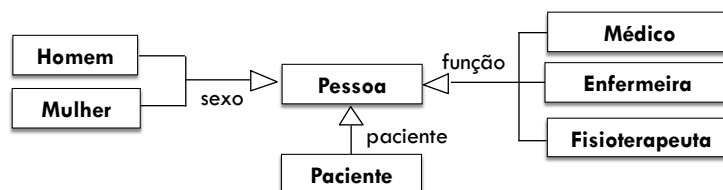
- Refere-se à possibilidade de objectos mudarem de classe dinamicamente
- Suportado em Smalltalk (mas não Java, C#, ou C++)
  - ▣ Pode ser simulado pelo padrão STATE
- Indicado em UML por estereótipo «dynamic»



## \*Classificação múltipla (≠simples)

60

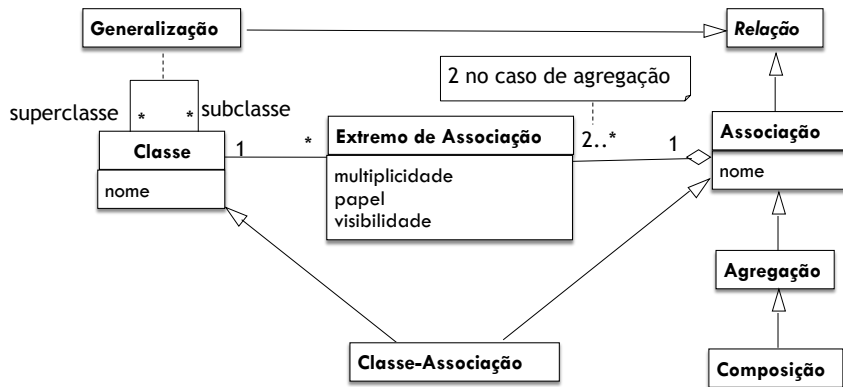
- Caso em que um objecto pode pertencer num dado momento a várias classes, sem que exista uma subclasse que represente a intersecção dessas classes (com herança múltipla)
- Não suportado pela generalidade das ling.s de prog. OO
  - ▣ pode ser simulada por agregação de papéis



*exemplo de combinação legal: {Mulher, Paciente, Enfermeira}*

## Exemplo: meta-modelo parcial

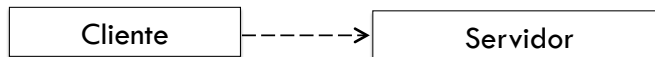
61



## Relação de dependência

62

- ❑ Relação de uso entre dois elementos (classes, componentes, etc.), em que uma mudança na especificação do elemento usado pode afectar o elemento utilizador
- ❑ Exemplo típico: classe-1 que depende de outra classe-2 porque usa operações ou definições da classe-2
- ❑ Úteis para gestão de dependências



## Relação de concretização (*realization*)

63

- Relação entre um elemento mais abstracto (que especifica uma interface ou um "contracto") e um elemento mais concreto (que o implementa esse contracto)
- Difere da generalização porque há apenas herança de interface e não herança de implementação



## Interface (≠ classe de implementação)

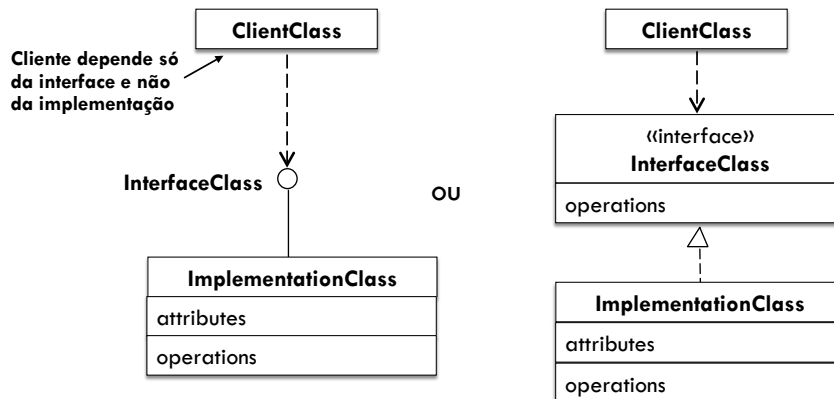
64

- Uma interface especifica um conjunto de operações (com sintaxe e semântica) que podem ser implementadas por (uma ou mais) classes ou componentes
  - Semelhante a classe completamente abstrata, só com operações abstratas, sem atributos nem associações
- Permite separação total de interface e implementação
- Uma classe pode implementar muitas interfaces
- Interfaces são mais importantes em linguagens como Java, C# e VB.NET que têm herança simples



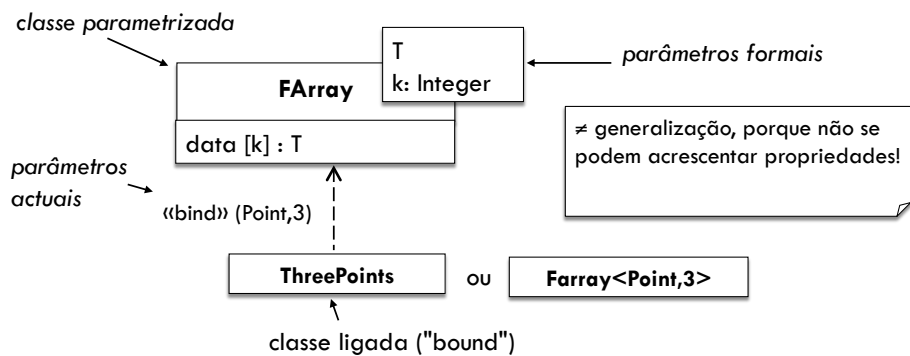
## Interface: notações alternativas

65



## Classes parametrizadas

66



C++

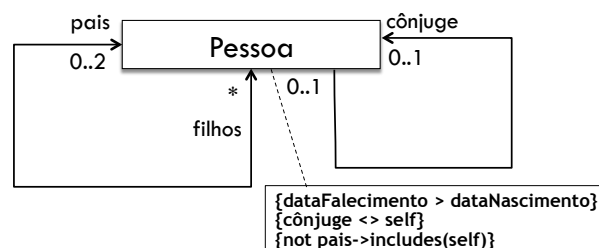
```
template <class T, int k>
class FArray { public: T[k] data; };
typedef FArray<Point,3> ThreePoints;
```

Java: class genérica

## Restrições

67

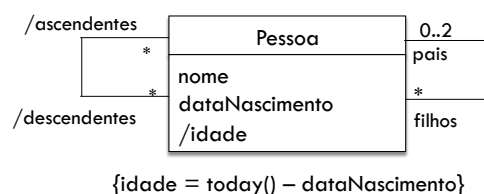
- Especificam condições que têm de se verificar em qualquer estado do sistema
- Indicadas por expressão ou texto entre chavetas junto ao(s) elemento(s) a que diz respeito
- Formalizáveis em OCL (*Object Constraint Language*)



## Elementos derivados

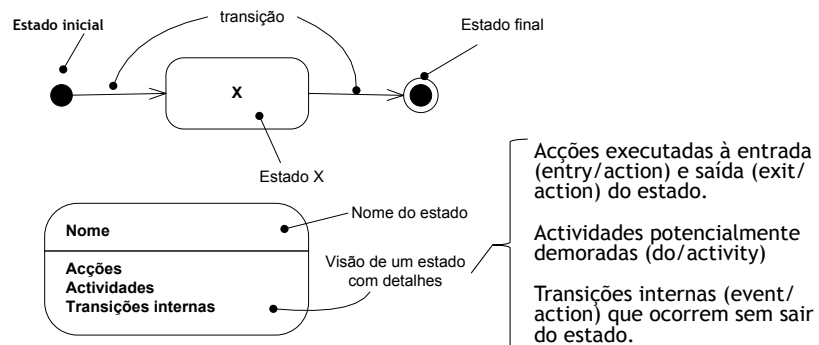
68

- Elementos (atributos ou extremos de associações) calculados em função doutros
- Indicados por “/” antes do nome
- Podem ter restrição associada com fórmula de cálculo



## Estado

- Situação registada por um objecto durante o seu ciclo de vida, durante a qual uma condição é verificada e vai executando alguma actividade, ou simplesmente espera que determinado evento ocorra.



## Transição

71

- Relação entre dois estados (origem e destino) que especifica que o objecto pode mudar de um para o outro
- Propriedades da transição: evento [condição] / acção
  - Evento ou gatilho
  - Condição de guarda
  - Acção ou efeito
- Quando o evento ocorre, se a condição se verificar, o objecto sai do estado de origem, executa a acção e entra no estado de destino
- Todas as três partes são opcionais.
  - Transição automática: não tem evento; ocorre quando termina a actividade do estado de origem.

## Evento

72

- Ocorrência de um estímulo que pode desencadear uma transição de estado ou uma acção
- Tipos de eventos
  - Sinal (*signal*)
    - Evento simbólico (com nome), modelado por objeto que é enviado assincronamente por um objeto e recebido por outro (exemplo: exceção)
  - Chamada (*call*)
    - Chamada de uma operação, tipicamente de modo síncrono.
  - Temporal (*time*)
    - Relativo: **after(t)** – decorrido o tempo t desde a entrada no estado de origem
    - Absoluto: **when(t)** – ocorre que se atinge o instante t
  - Mudança de estado
    - **when(condição)** –condição no estado interno do objeto torna-se verdadeira

# Ação

73

- Computação atômica
- Execução num período de tempo instantâneo e não interrompível
- Exemplos de ações
  - Invocação de métodos
  - Criação ou destruição de objetos
  - Envio de sinal para outro objeto (*send*)
- Podem-se associar a transições e estados
- Num estado
  - Ações de entrada: *entry*
  - Ações de saída : *exit*
  - Outras ações internas: *evento / ação*.

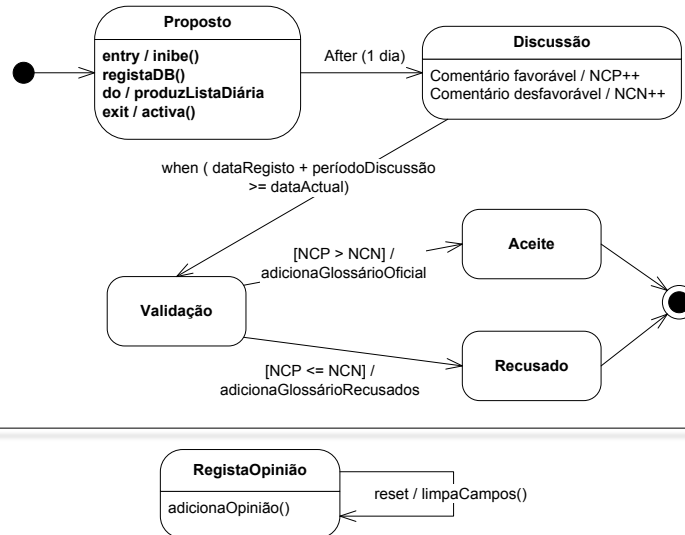
# Atividade

74

- Computação não atômica
- Interrompível por outros eventos
- Utilizadas principalmente em diagramas de atividades
- Mas também podem ser referidas na especificação de um estado
  - Através do prefixo “do”.
  - Sequência de ações : “do / *oper1()*; *oper2()*;...”

## Exemplo

75



## Estados compostos

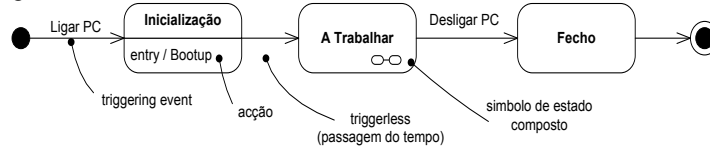
76

- Estado composto por sub-estados.
- Pode conter
  - Sub-estados concorrentes (em regiões ortogonais)
  - Sub-estados sequenciais.

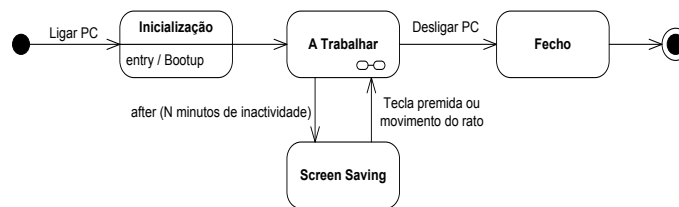
## Exemplo: Estados compostos

77

### □ Diagrama de Estados de um PC

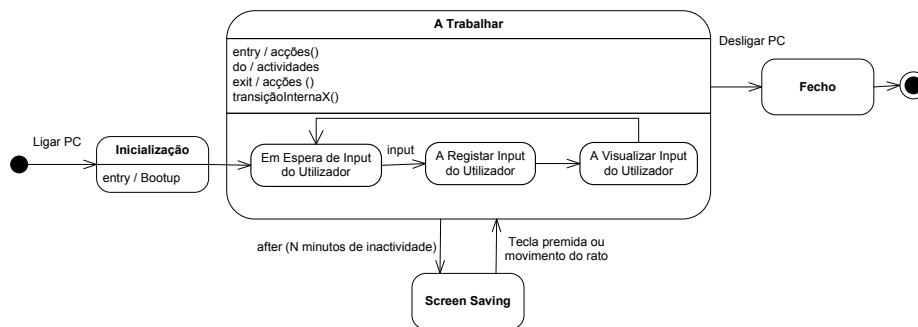


### □ Variante 1



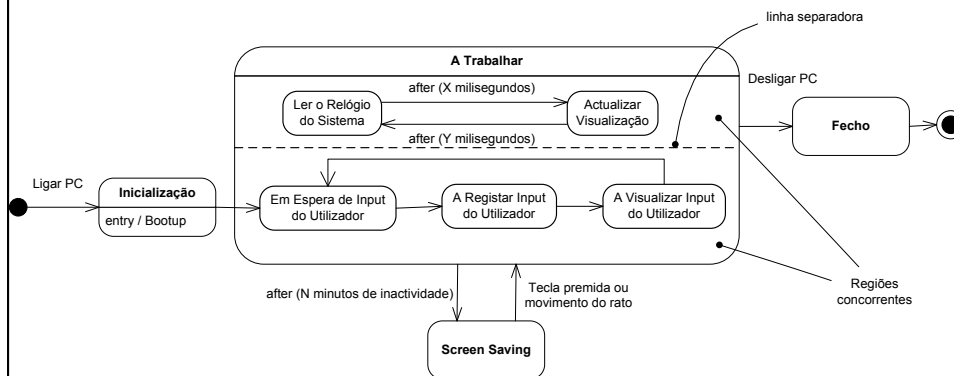
## Exemplo: Subestados sequenciais

78



## Exemplo: Subestados concorrentes

79



## Outros conceitos

80

- Especialização/generalização de estados
- *Protocol state machines* – mais abstractas, em vez de acções e condições de guarda têm pré e pós-condições
- Eventos diferidos: event/defer
- Pseudo estados (além de estado inicial e final)
  - deep history, shallow history, barra de difusão (fork), barra de junção (join), ponto de junção (junction), ponto de decisão e ponto de entrada (entry point)
- Portos - definição de tipos de serviço suportados por um estado



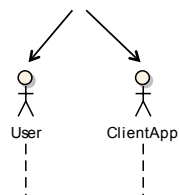
81

\*Referência sobre diagramas de sequência

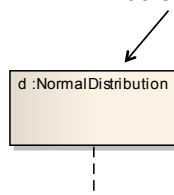
## Intervenientes (*lifelines*)

82

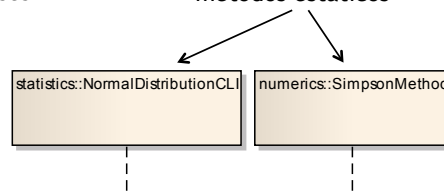
Actores - Utilizadores ou sistemas externos



Objetos - Instâncias de classes



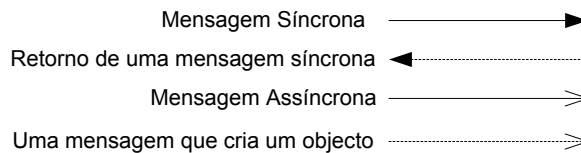
Classes - Para invocar métodos estáticos



## Mensagens - Tipos

83

- Comunicação entre dois intervenientes
  - Pode veicular informação, desencadear ações e mudanças de estado
- Tipos :
  - *Call*: Chamada de operação, normalmente síncrona (emissor bloqueia)
  - *Reply*: Retorno de chamada
  - *Send*: Envio de sinal, normalmente assíncrono (emissor não bloqueia)
  - *Create*: Criação de objeto (invocando construtor), normalmente síncrona
  - *Destroy*: Destruição de objeto (e.g., delete em C++)



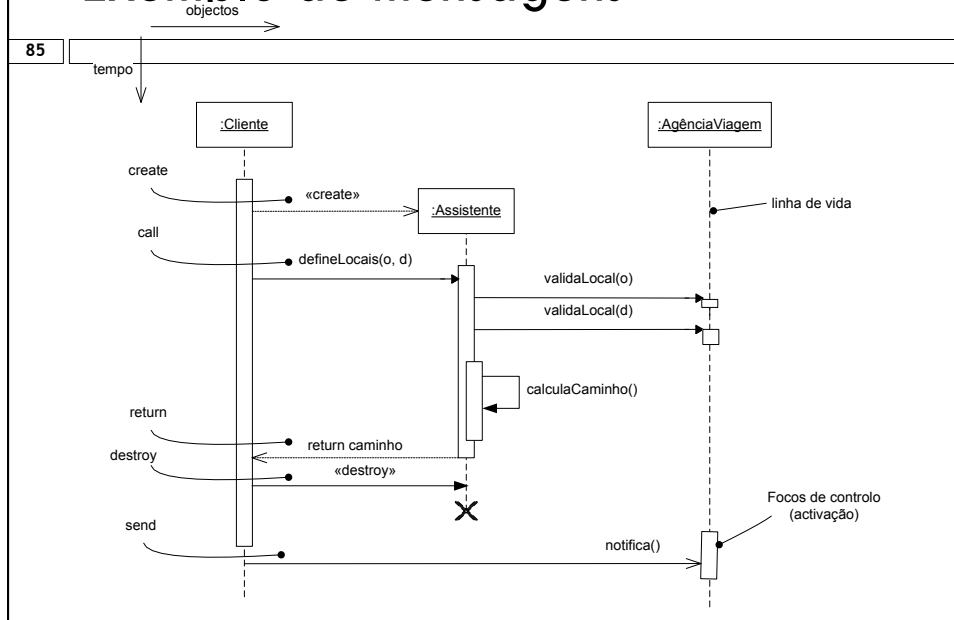
## Mensagens - Texto

84

- Texto da mensagem:

```
[attribute=] signal-or-operation-name [(arguments)] [:ret-val]
arguments ::= argument [, arguments]
argument ::= [parameter-name= ] argument-value | -
attribute ::= out-parameter-name [:argument-value] | -
```
- Exemplos
  - `cancela()`
    - Envio de uma operação sem valores em argumentos e sem retorno.
  - `cancelaProposta(data=31/12/2006, -): void`
    - Envio de uma operação com o primeiro argumento com valor "31/12/2006", o segundo argumento sem valor e com retorno do tipo "void".

## Exemplo de mensagens



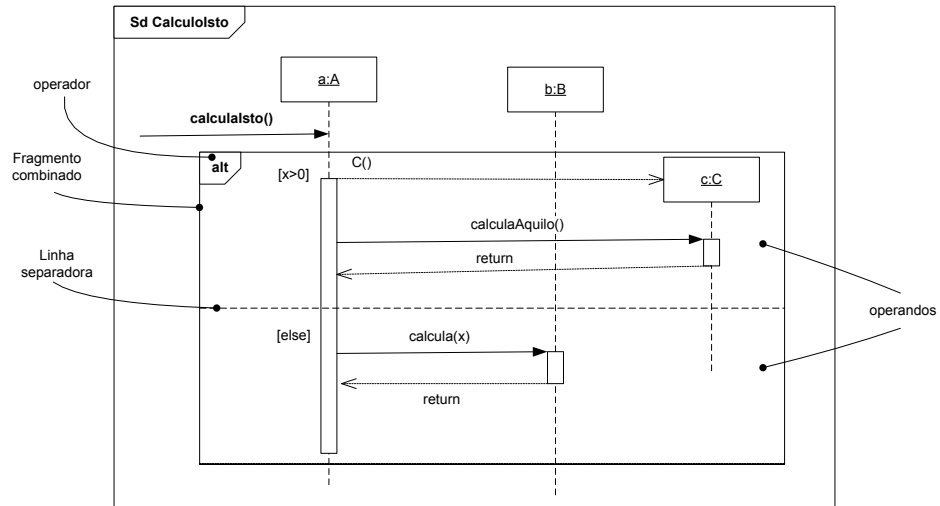
## Fragmentos Combinados

86

- Agregam um ou mais pedaços de interacção (operandos)
- Tipos (operadores) mais importantes:
  - ▣ **seq (sequência)** - Sequência de operações ordenadas fracamente (por linha de vida)
  - ▣ **strict (estrita)** - Sequência de operações ordenadas estritamente .
  - ▣ **alt (alternativa)** - Comportamentos alternativos segundo condições
  - ▣ **opt (opção)** - Execução segundo condição de guarda
  - ▣ **par (paralelo)** - Execução em paralelo de conjunto de operações
  - ▣ **loop (iteração)** - Ciclos de execução
  - ▣ **critical, neg, assert, consider, ignore** – Mais raros (ver especificação UML)

## Exemplo de fragmento combinado

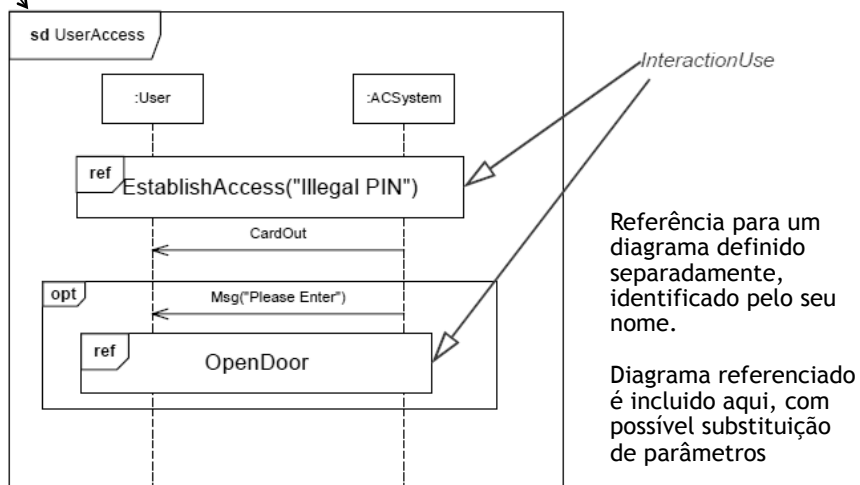
87



## Reutilização de interações

88

Moldura (frame)



# Proposta de metodologia (1)

89

- 1. Modelar principais funcionalidades do sistema
  - Diagrama de casos de utilização.
- 2. Modelar estrutura do sistema
  - a) Diagrama(s) de pacotes: Estrutura de módulos do programa
  - b) Diagrama(s) de classes: Estrutura de classes do programa
    - Ver aula anterior sobre digramas de classes!
- 3. Gerar automaticamente esqueleto do programa (*MDD - Model-Driven Development*)
  - A partir dos diagramas de classes, e.g., com Enterprise Architect (*source code engineering*)
  - Código gerado compila, mas corpo dos métodos é vazio

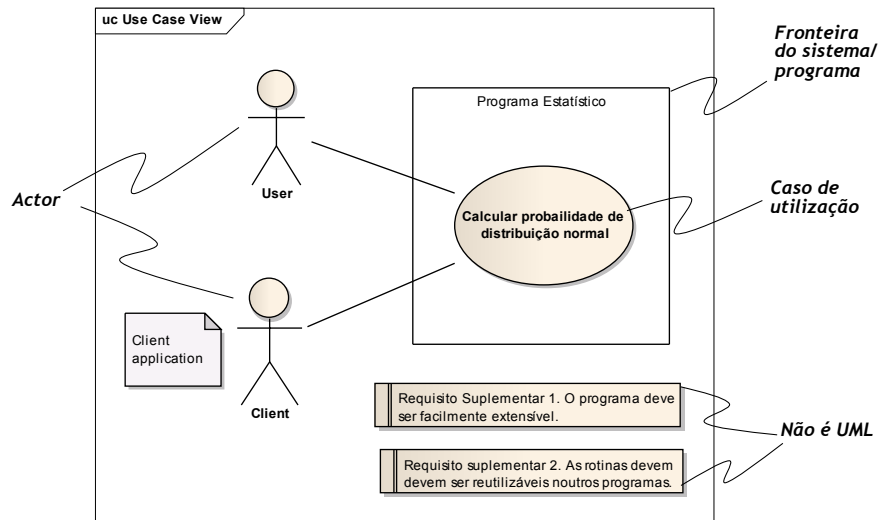
# Proposta de metodologia (2)

90

- 4. Modelar cenários de utilização e teste (diag. de sequência)
  - Interações internas entre objetos no programa
  - Interação externas com o utilizador e/ou aplicações cliente
  - Critério de completude: todas as classes e métodos são exercitados
- 5. Gerar testes unitários (JUnit) (*MBT - Model-Based Testing*)
  - A partir dos diagramas de sequência (cada diagrama dá um método)
  - Plug-in para Enterprise Architect (não instalado na FEUP)
- 6. Implementar os métodos
  - Executar os testes e ver testes a falhar (*TDD – Test-Driven Development*)
  - Escrever o corpo dos métodos
  - Executar os testes e ver testes a passar
- 7. Iterar

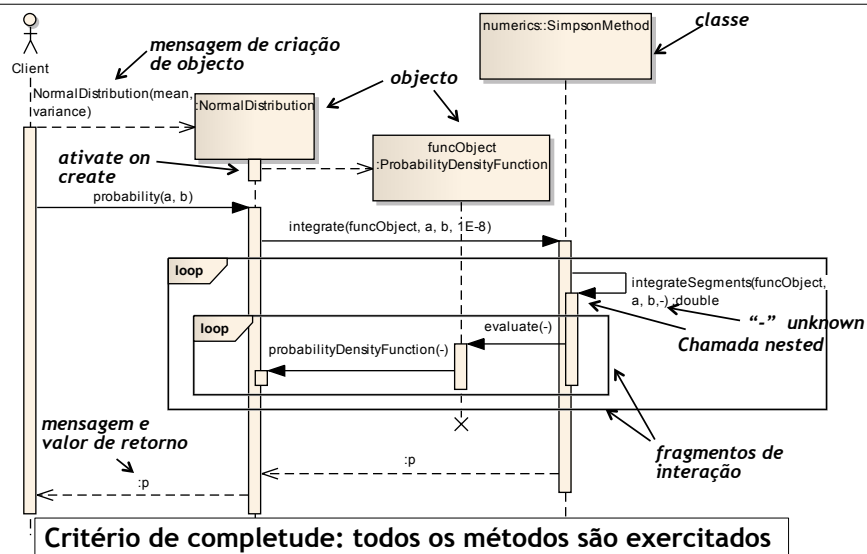
## Passo 1 – Diagrama de Casos de Utilização

91



## Passo 4 – Diagrama de sequência - interações internas

92



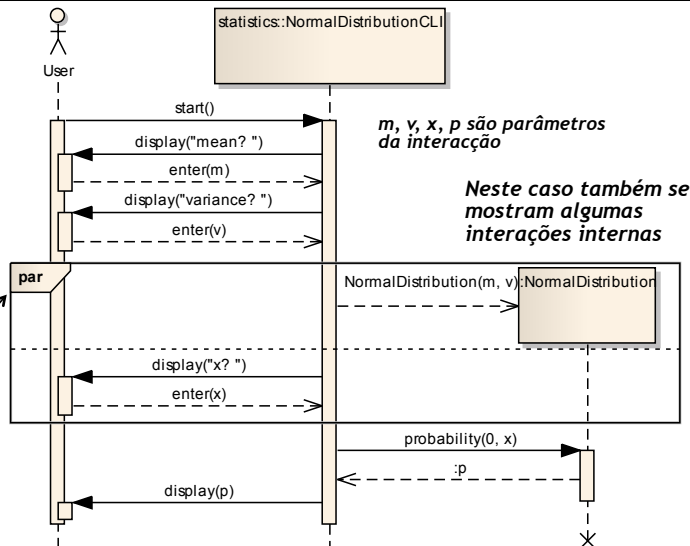
## Passo 4 – Diagrama de sequência - interação com utilizador (CLI)

93

Não há uma forma standard de modelar interação com o utilizador em UML.

Neste caso são usadas *keywords* entendidas pelo *plug-in* de geração de testes.

*Paralelo: os dois operandos (separados por traço interrompido) podem executar em paralelo (ou por qualquer ordem)*



## Referências e informação adicional

94

- [www.uml.org](http://www.uml.org) - especificações
- [www.agilemodeling.org](http://www.agilemodeling.org) – tutoriais, etc.