Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2024/2025

Desenho Centrado em Objectos

Desenho de Classes
— preparado para a mudança —

Desenho

representar uma solução de desenho OO

Desenho

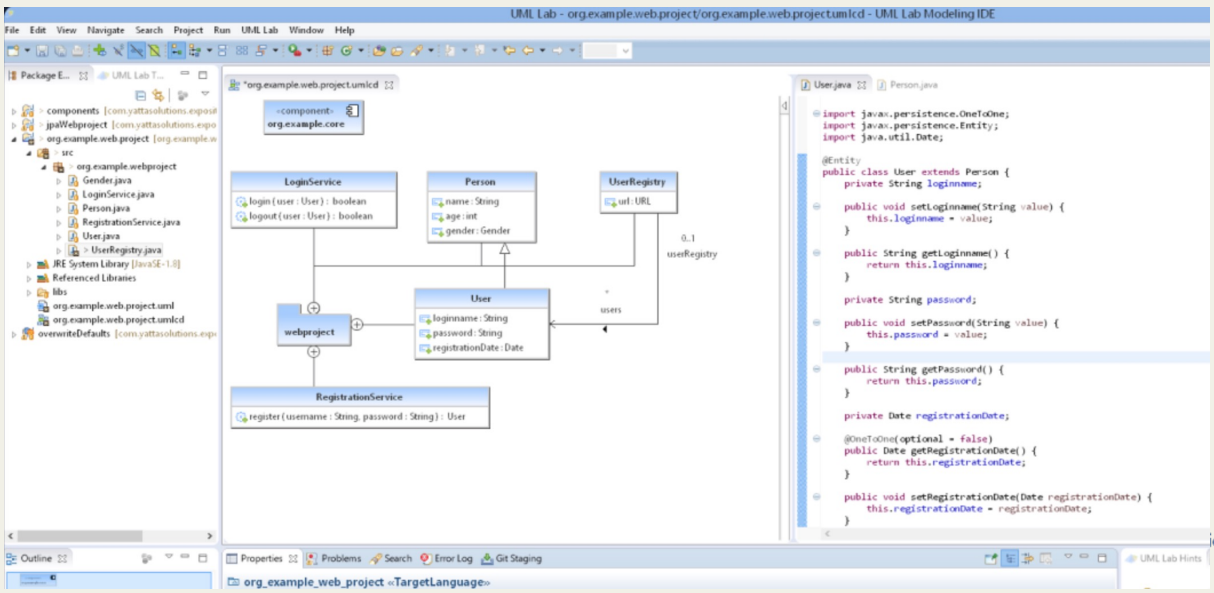
- O **processo de desenho** de um sistema de software envolve decidir sobre
  - a estrutura do código que tem de ser escrito e
  - como o sistema vai funcionar
- Ou seja, é um processo em que se procura definir uma **solução de desenho** para o sistema

- As linguagens de programação não são suficientemente abstratas para permitir descrever estas decisões
- Pelo que é preciso outro meio de representação, idealmente que permita **representações visuais** com imagens

Ciências ULisboa | Informática

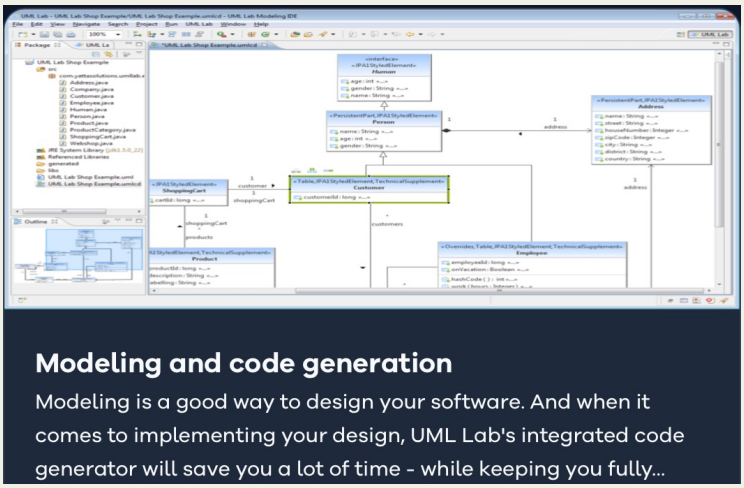
UML: Unified Modelling Language

- Notação que permite representar, através de vários tipos de **diagramas**, diferentes tipos de decisões de desenho
  - abstraindo de detalhes que não são relevantes
  - nomeadamente a linguagem usada na implementação



UML: Unified Modelling Language

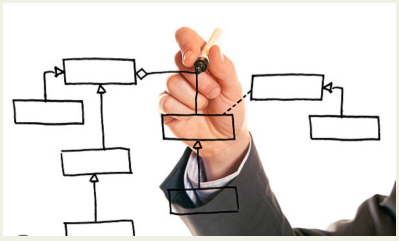
- É uma notação bem estabelecida e *standard*
- Existem muitas ferramentas que suportam a edição de diagramas UML, que geram diagramas a partir de código, que geram código a partir de diagramas UML,….



https://www.uml-lab.com

UML: Unified Modelling Language

- A maior parte dos engenheiros de software
  - sabem interpretar diagramas UML
  - usam a notação para fazer esboços, comunicação e documentação



- Em DCO vamos usar a notação UML essencialmente para comunicação
  - para fazer esboços que nos ajudam a discutir diferentes decisões
  - para fazer descrições que vão guiar os implementadores
  - vamos sempre omitir os detalhes que não sejam relevantes no contexto, tendo em conta o propósito em vista

Classes (em revisão)

Uma **classe** em Java (e em outras linguagens OO)

- Define um **tipo**
- Tem **membros**:
  - atributos,
  - construtores, métodos
  - classes internas
- Membros podem ser de classe (***static***) ou de instância

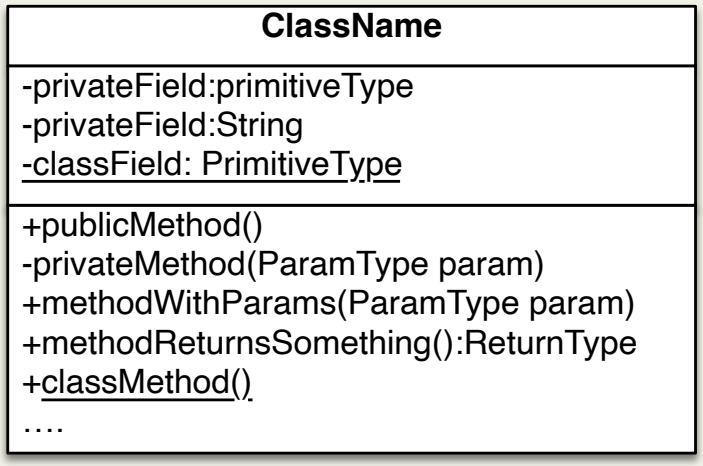
Os **objetos** de uma classe

- têm **estado** (dados) e **comportamento** (ações)
- interagem com outros objetos enviando-lhes mensagens
  - para executar uma ação (ex., *translate*)
  - pedir uma informação (ex., *getX*)

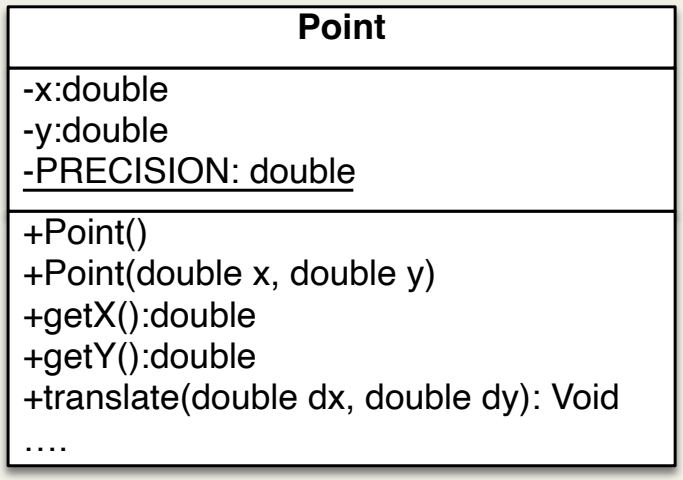
Exemplo: Pontos num espaço bidimensional


```
public class Point  {
    //precisao
    private static final double PRECISION = 0.00001;

    // coordenadas do ponto (abcissa e ordenada)
    private double x;
    private double y;

    public Point (){..

    public Point (double x, double y){
        this.x = x;
        this.y = y;
    }

    public double getX (){..

    public double getY (){..

    public void translate (double dx, double dy){
        x = x + dx;
        y = y + dy;
    }
```

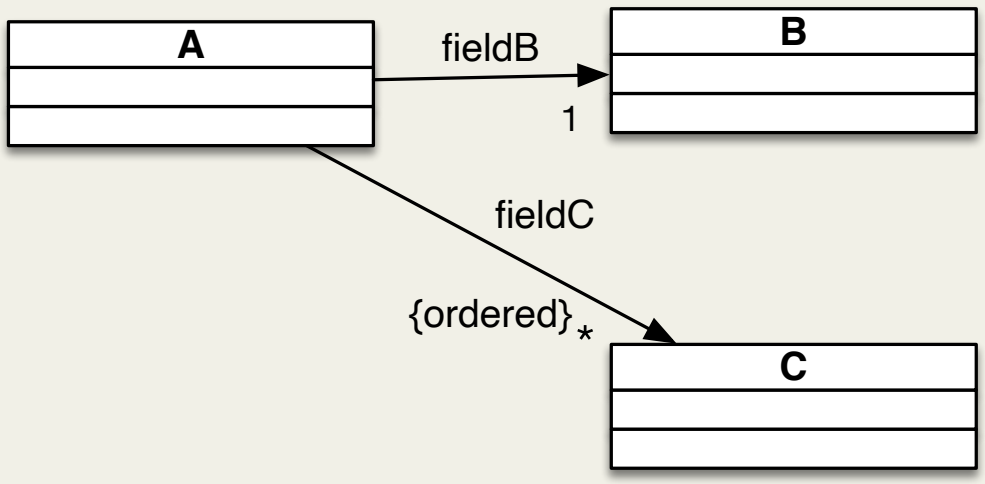
Classes: representação em UML



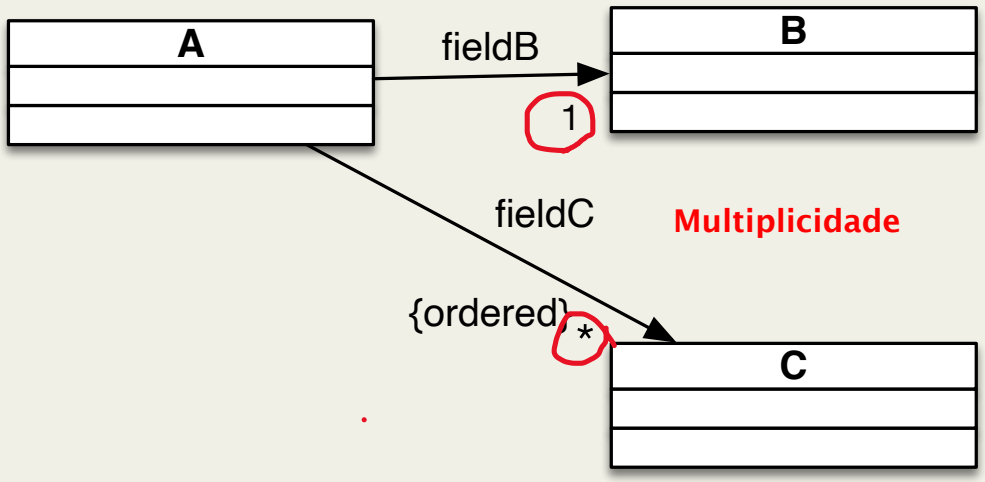
Exemplo



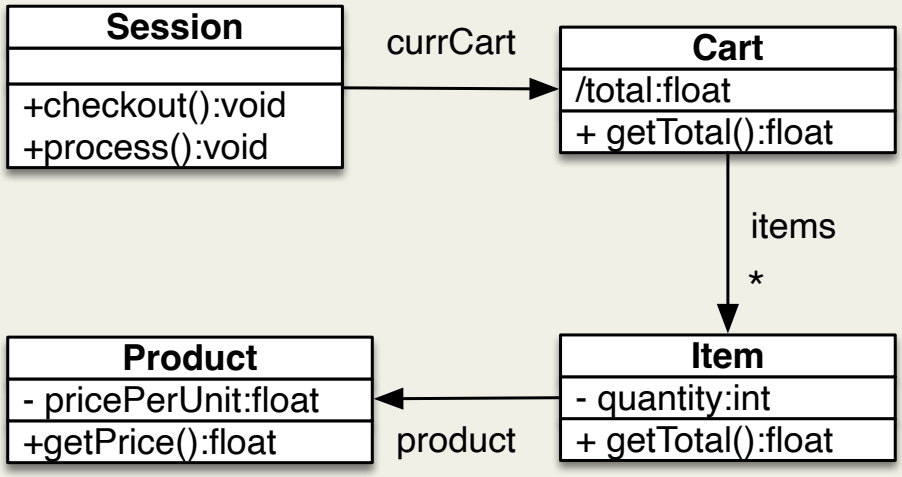
Classes: representação em UML



Classes: representação em UML



Exemplo



Interfaces (em revisão)

Um **interface**

- Define um tipo
- Pode ser definido como subtipo de outros tipos também definidos por interfaces — usa-se o **extends**
- Desta forma podemos ter tipos que têm múltiplos supertipos

Uma classe pode declarar implementar um ou mais interfaces — usa-se o **implements**

Exemplo: Pontos

```
public interface IPoint{

    double getX();

    double getY();

    void translate(double dx, double dy);

    void move(double x, double y);

    double distance(IPoint p);

    boolean colinear(IPoint p1, IPoint p2);

    IPoint copy();
}
```


Exemplo: Polinómios a uma variável

$3x^2 - 5x + 4$.

```
public interface Poly extends Iterable<Integer>, Comparable<Poly> {

    public int degree();

    public int coeff(int d);

    public Poly add(Poly other);

    public Poly sub(Poly other);

    public Poly mul(Poly other);

    public Poly minus();
}
```

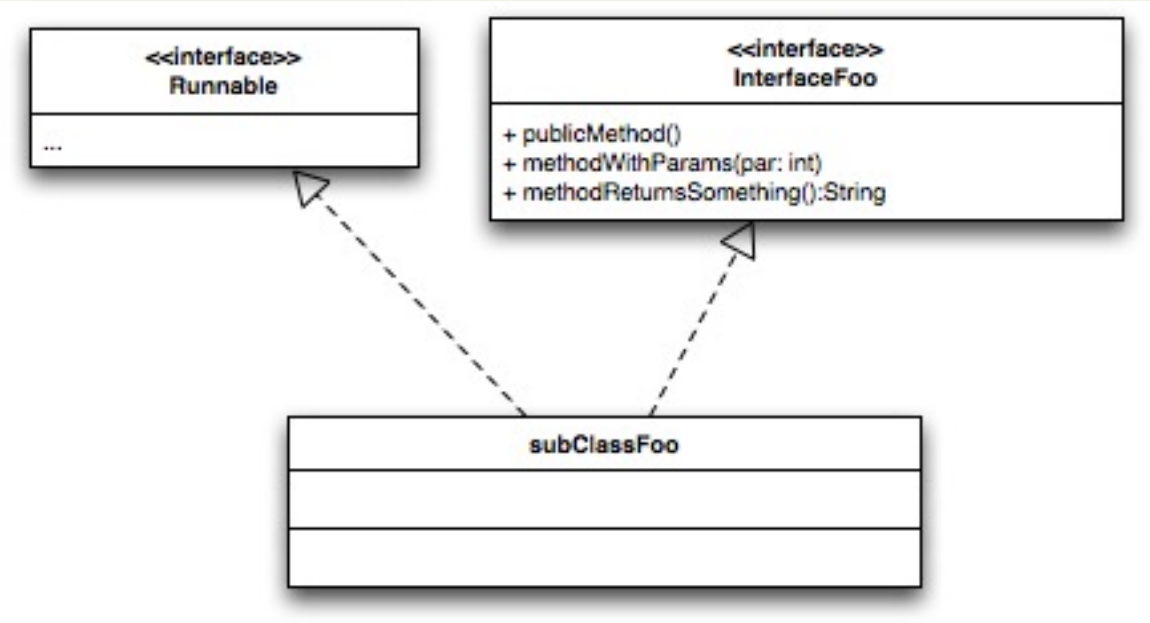
Interfaces

- Alguns interfaces servem para a classe anunciar que sabe fazer qualquer coisa
  - **Comparable<T>**
  - **Cloneable** (*marker interface*)
  - **Serializable** (*marker interface*)
  - **Runnable**
- O **Iterable<T>** tem um papel especial por causa do **for each**

Interfaces

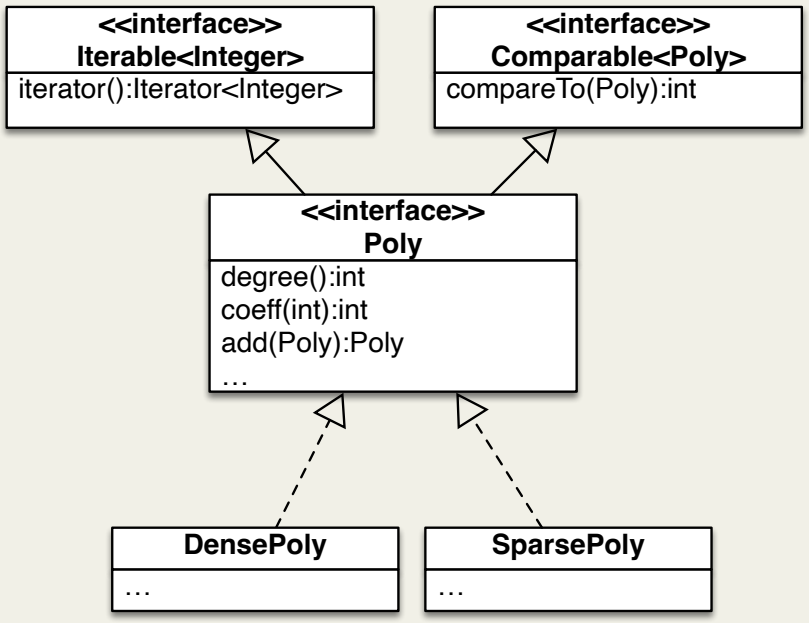
- Outros interfaces muito usados são os definidos na *framework* das **Collections**
  - **Collection<T>**
  - **Set<T>**
  - **SortedSet<T>**
  - **List<T>**
  - **Map<K,T>**
  - **…**
- Também populares são os interfaces funcionais
  - um único método abstrato, anotado **@FunctionalInterface**
  - **Function<T,R>** com **<R> apply(T param)**
  - **Predicate<T>** com **boolean test(T t)**

Interfaces: representação em UML



Usa-se o triângulo não preenchido para representar as relações de subtipo
A linha é a tracejado no **implements** e sólida no **extends**

Exemplo



Estrutura estática & Comportamento dinâmico

- No desenho de uma solução OO tomam-se decisões relativamente
  - à **estrutura estática** das unidades de implementação
    - que interfaces, classes, ... fazem parte da solução
    - que membros tem cada interface, classe, ... e como se relacionam estaticamente
  - ao **comportamento dinâmico** dos diferentes tipos de objetos que fazem parte da solução

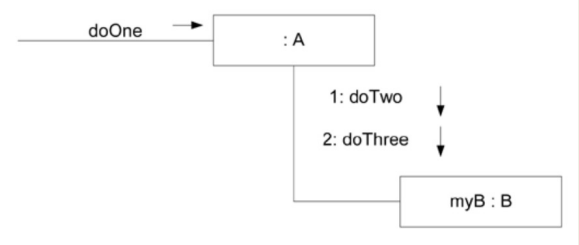
- Os **diagramas de classes** (como os exemplos anteriores) permitem representar a estrutura estática.
- Os **diagramas de interação** permitem representar as interações entre objetos realizadas através do envio de mensagens

Interações entre Objetos: representação em UML

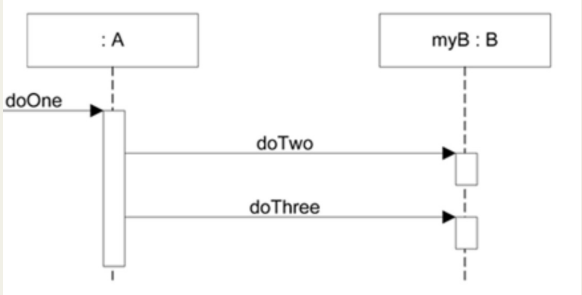
O UML tem dois tipos de diagramas de interação

- focam-se ambos na **interação** entre os objetos que é necessária para realizar uma **determinada tarefa** (*doOne* no exemplo)

Diagramas de Comunicação

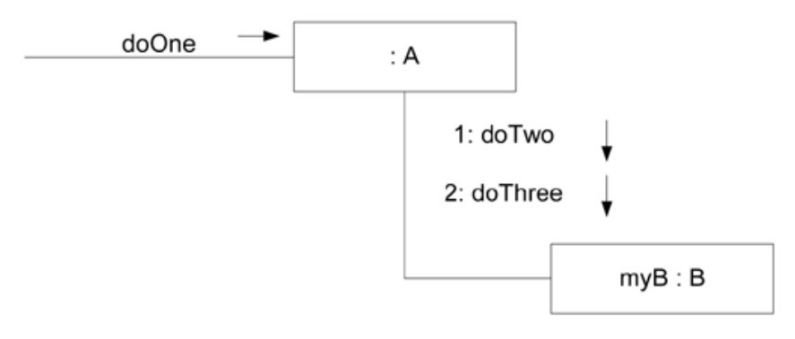


Diagramas de Sequência



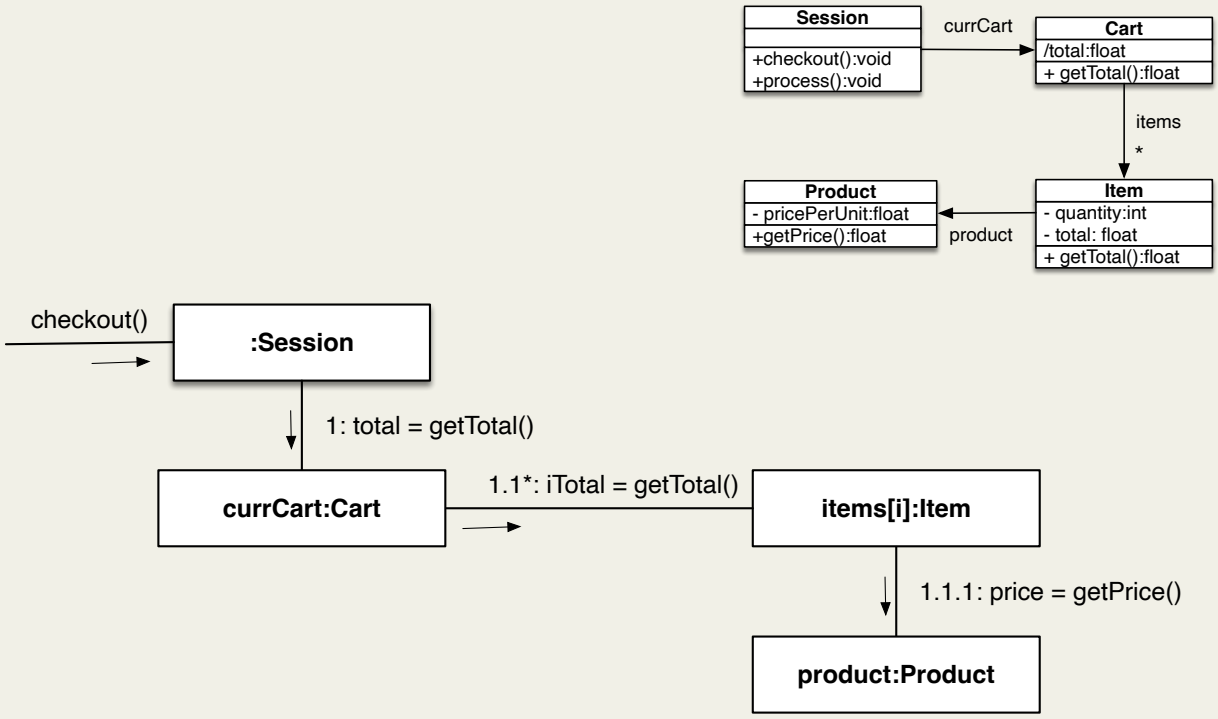
Interações entre Objetos: representação em UML

Diagramas de Comunicação



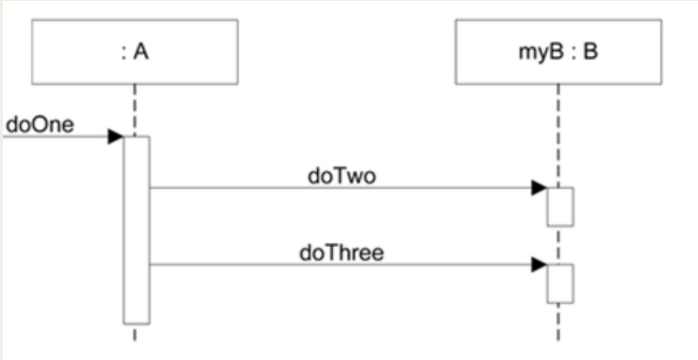
- A ordem das interações é descrita através de um sistema de numeração hierárquico
- As mensagens que um objeto envia para responder a ***s : msg*** são etiquetadas com ***s.1***, ***s.2***, ***s.3*** ....
- Etiquetas da forma ***s.k**** usadas para representar o envio iterado da mensagem

Exemplo



Interações entre Objetos: representação em UML

Diagramas de Sequência



- Mais fácil de exprimir a ordem das interações (de cima para baixo)
- Mas mais difíceis de ler e desenhar quando há muitos objetos envolvidos