

MODELAÇÃO E ANÁLISE DE SISTEMAS

# Modelos de interação - diagrama de sequência

Ilídio Oliveira

v2022-10-28

# Objetivos de aprendizagem

Compreender o papel da modelação do comportamento no SDLC

Compreender as regras e as diretrizes de estilo para sequência, comunicação e diagramas de estado

Compreender a complementaridade entre os diagramas de sequência e comunicação

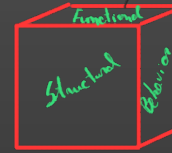
Explicar a relação entre modelos de função, estruturais e de comportamento

## Os modelos são vistas parciais



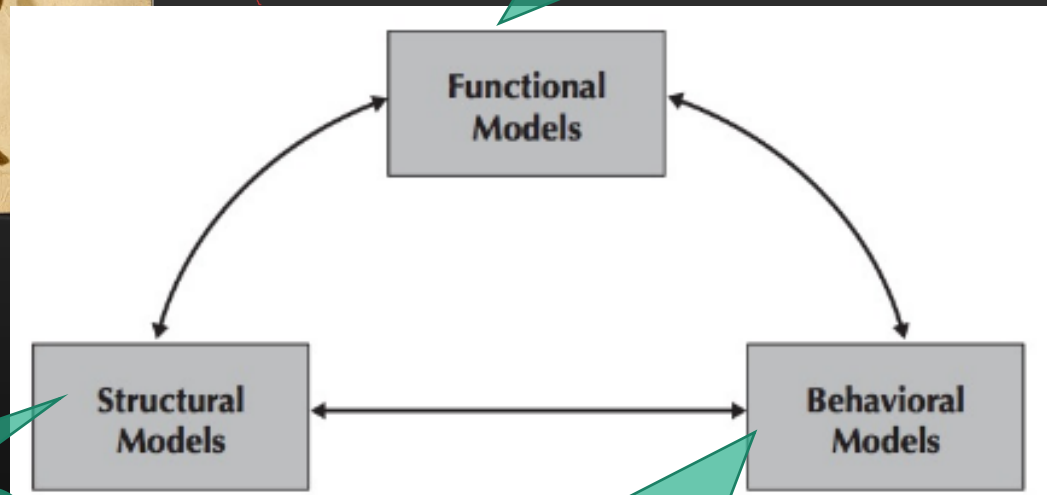
Num projeto, recorremos a  
vários modelos  
complementares

# Três categorias de modelos



Eg: Curso de utilização  
Diag. de atividades  
e  
Pedaço  
de  
caixa!

O que é que o Sistema deve fazer?  
[do ponto de vista do observador  
externo;  
caixa fechada]



Quais são as partes/"coisas"  
que constituem e o  
relacionamento estrutural  
entre elas ?

Como é que as partes  
colaboram/interagem ao  
longo do tempo? [perspetiva  
dinâmica]

# O que tratam os modelos comportamentais

Os modelos comportamentais descrevem os aspetos dinâmicos de um sistema de informação.

**Durante a análise:**

os modelos comportamentais descrevem qual é a lógica interna dos processos sem especificar como os processos vão ser implementados.

**Durante o Desenho/Implementação:**

os modelos comportamentais descrevem a interação entre módulos de software

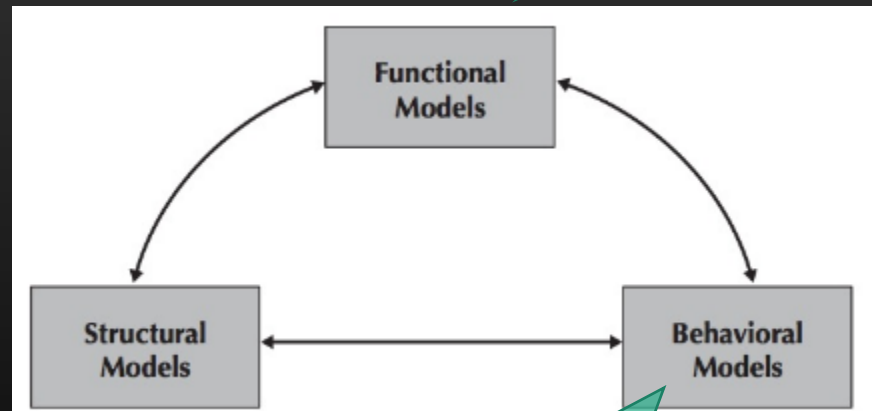
**A modelação comportamental pode ser conduzida pelos casos de utilização:**

Um dos principais objetivos é mostrar como os objetos de um domínio trabalham em conjunto para formar uma colaboração que realiza cada um dos cenários dos casos de utilização.

Modelos estruturais → os objetos e as relações

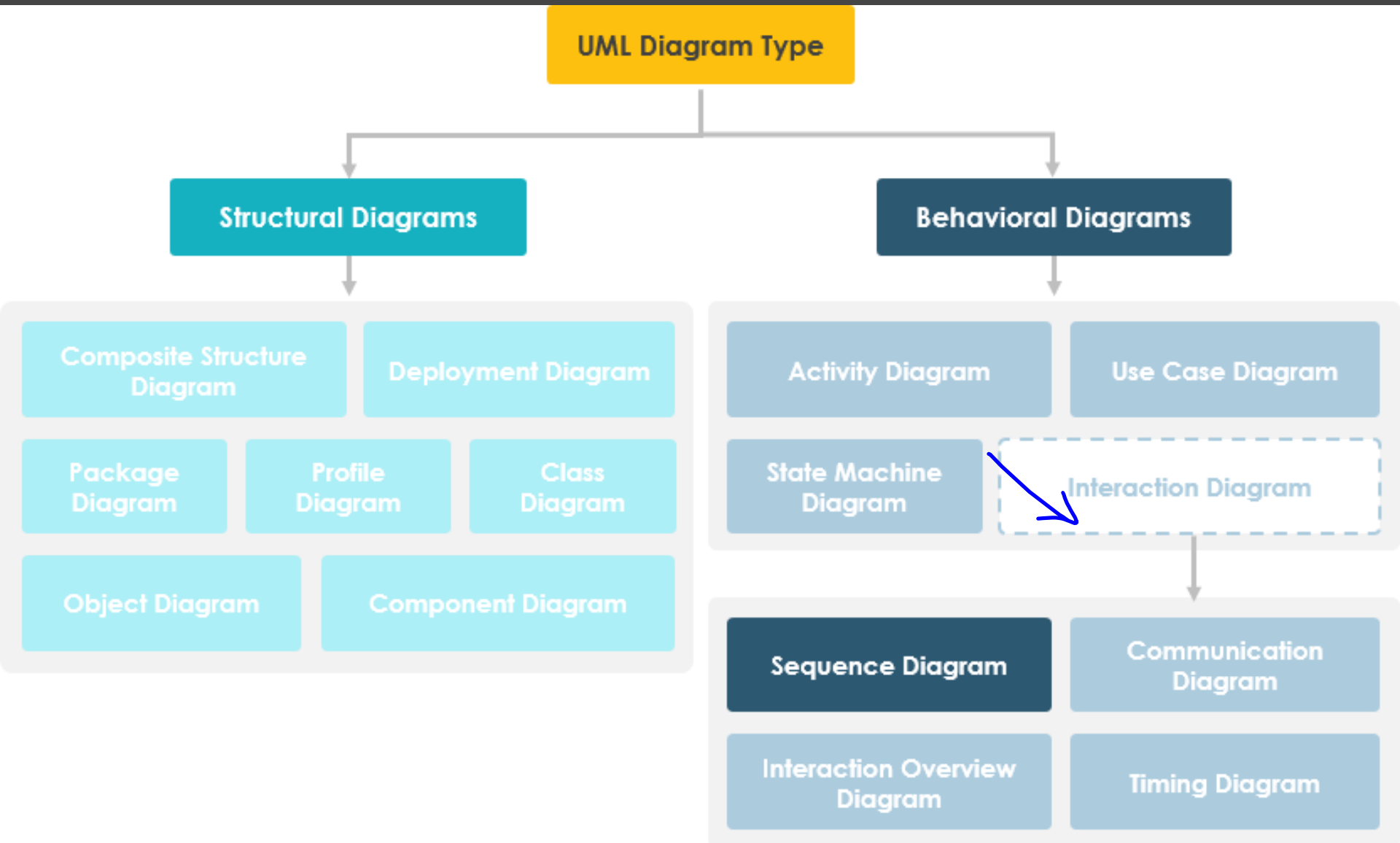
Modelos comportamentais → visão interna de um processo.

Observação do exterior  
(e.g.: casos de utilização)



Partes e relacionamentos  
estruturais entre elas (e.g.: d.  
de classes)

Explicação de como é feito  
(e.g.: interação entre objectos  
para realizar um CaU)



# Tipos de modelos comportamentais

## Representação dos detalhes de um processo

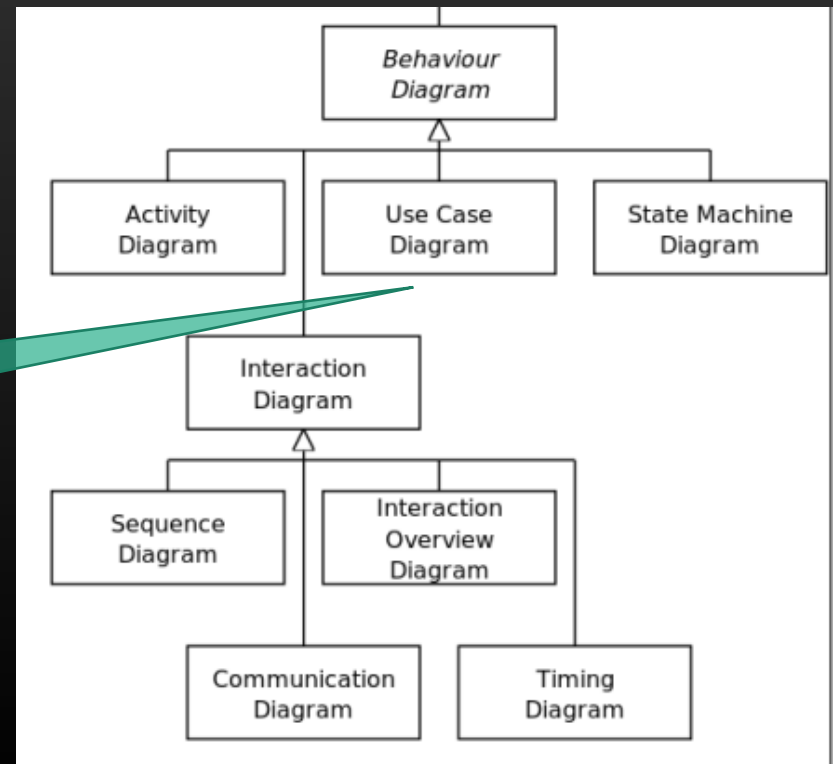
Diagramas de interação (Sequência & Comunicação)

Mostra como os objetos colaboram para fornecer a funcionalidade definida nos casos de utilização.

O d. de CaU é um diagrama do tipo comportamental na UML; é usado na Análise para construir Modelos Funcionais.

## Representações de alterações nos dados (Estado)

Máquinas de estado

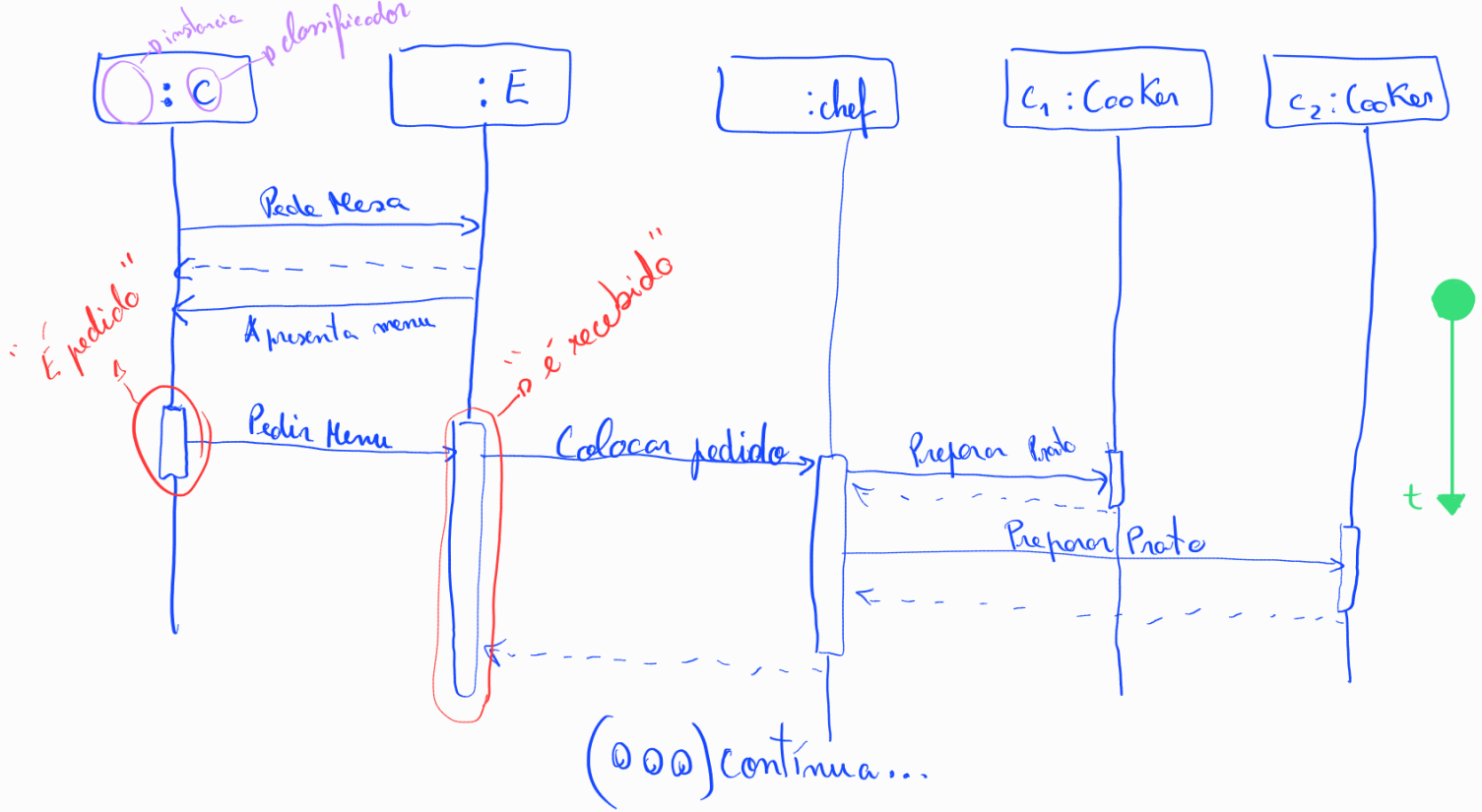




# Cenário do restaurante: envio de mensagens



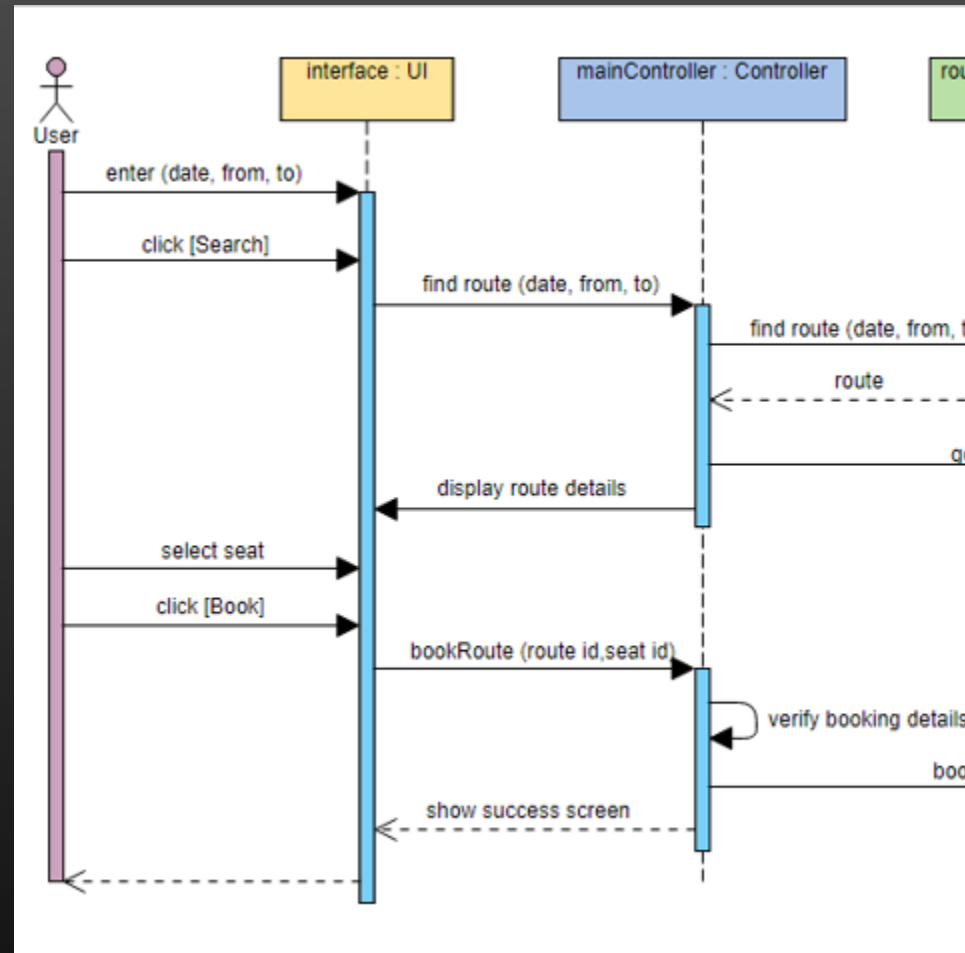




# Diagrama de sequência

Ilustrar os objetos que participam numa colaboração (por exemplo: caso de utilização) e as mensagens que passam entre eles ao longo do tempo.

Um diagrama de sequência é um modelo dinâmico que mostra a sequência explícita de mensagens que são passadas entre objetos numa interação definida.



<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

participant1 : ParticipantClass

participant2

Time

participant1:ClasseParticipante

participant2:

1: mensagem( argumentos)

O invocador da  
mensagem (solicita  
colaboração)

O recetor da  
mensagem (presta  
colaboração/serviço)

Mensagem, com  
parâmetros (assinatura da  
mensagem)

Retorno  
(opcional)

Tempo  
foi de  
cima para  
baixo!

# Foco na colaboração ao nível do objeto

## Diagramas de classe

O foco de modelação dos diagramas de classe está no nível de classe (classificador).

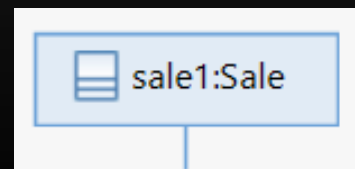
Cada tipo de objetos tem atributos que descrevem informação (estado) sobre o objeto.

## Diagramas de interação

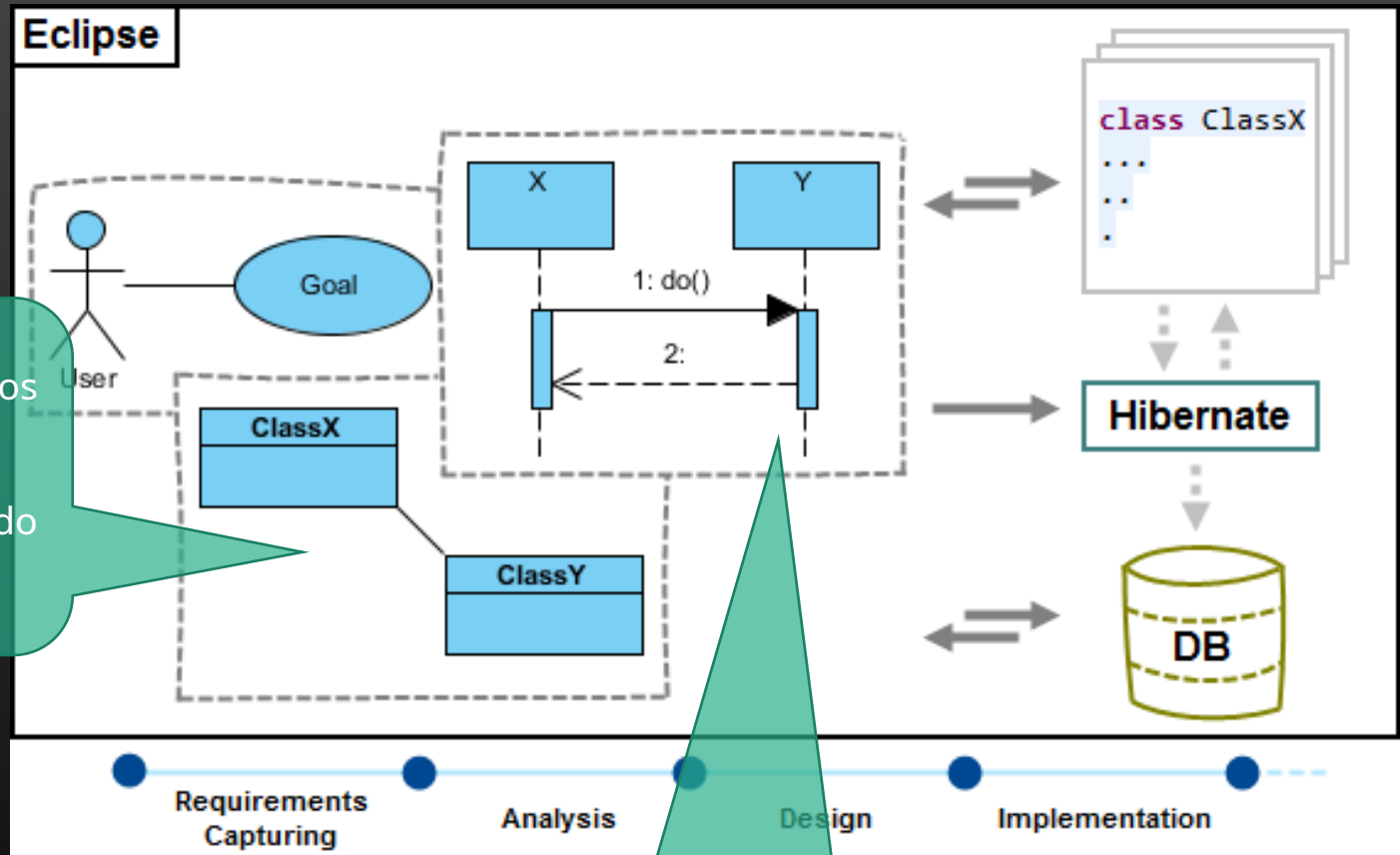
Os diagramas de interação focam-se no **nível do objeto/instância**

Cada objeto também tem comportamentos. Os comportamentos são descritos pelas operações. Uma operação é uma ação que um objeto pode realizar.

Cada objeto também pode **enviar e receber mensagens**. As mensagens são solicitações enviadas a objetos para que executem um dos seus comportamentos. Uma mensagem é uma *chamada* de um objeto para outro objeto.



# Nível de abstração: análise ou implementação?



O D. de Sequência mostra como é que os objetos (que seguem uma Classe) colaboram com outros, para realizar uma atividade. Também aqui, essa atividade pode ser no nível do "domínio" ou no nível do código.

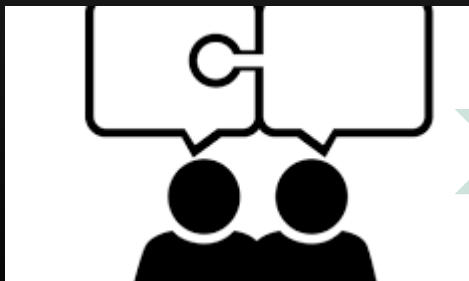
# O modelo do domínio e a implementação situam-se em planos de abstração diferentes

## Diagramas de sequência: etapa de análise

Explicar a **colaboração** observada para realizar um processo do domínio.

Os “objetos” são tipicamente entidades de alto nível:

- papéis de pessoas ou subsistemas

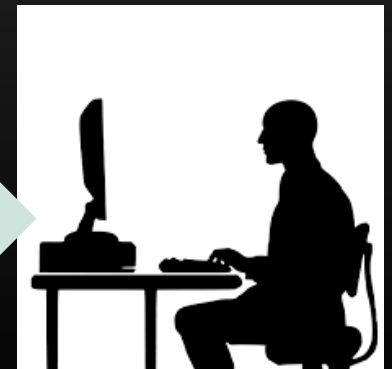


I Oliveira

## Diagrama de sequência: perspectiva de implementação

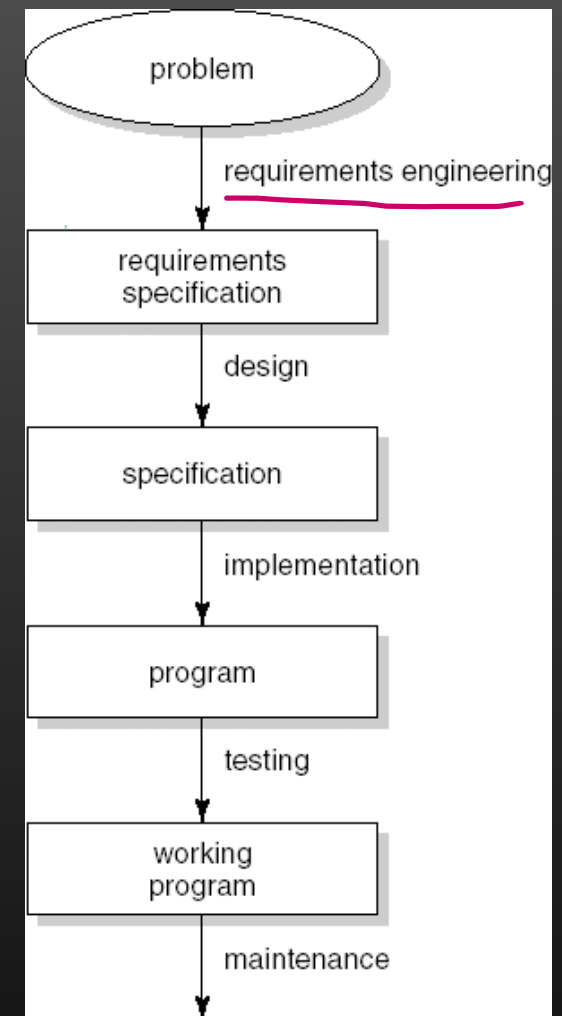
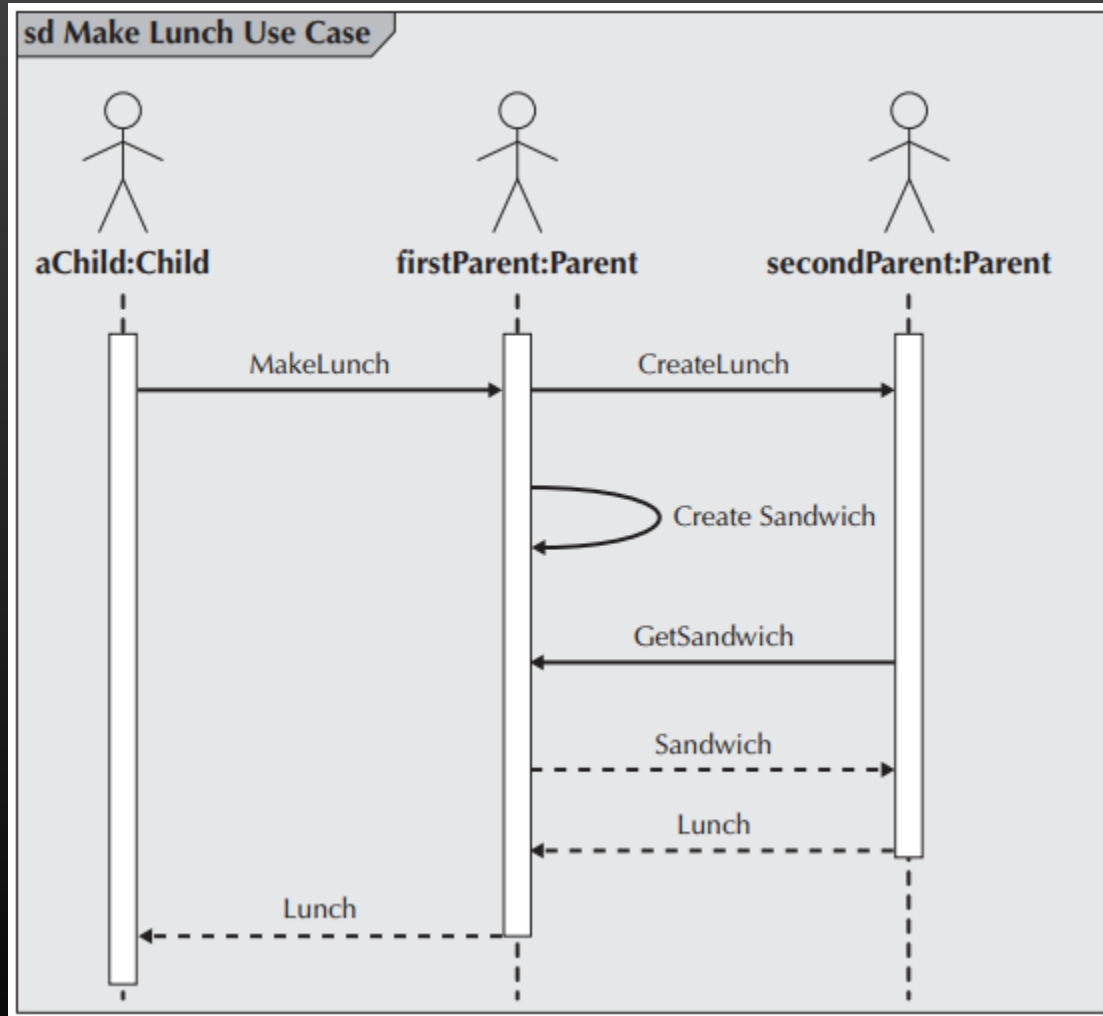
Explicar a colaboração entre objetos de código para realizar uma operação (da implementação)

Os objetos são instâncias numa linguagem OO (e.g. Java)



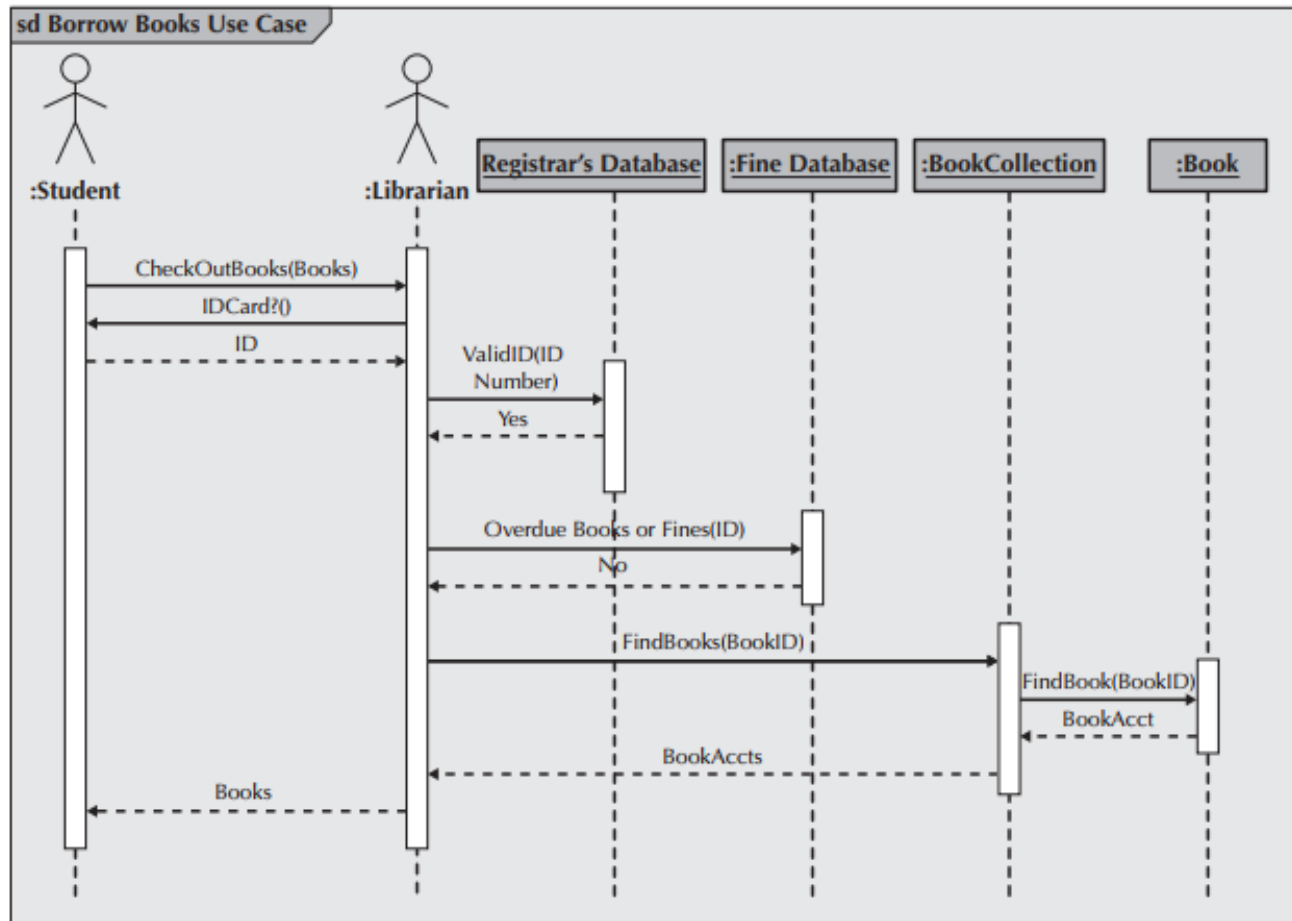
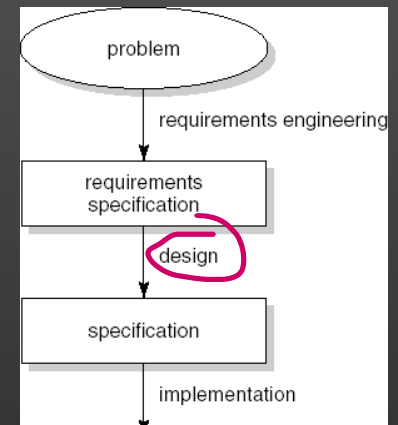
13

# Colaboração entre atores/fluxo de um CaU



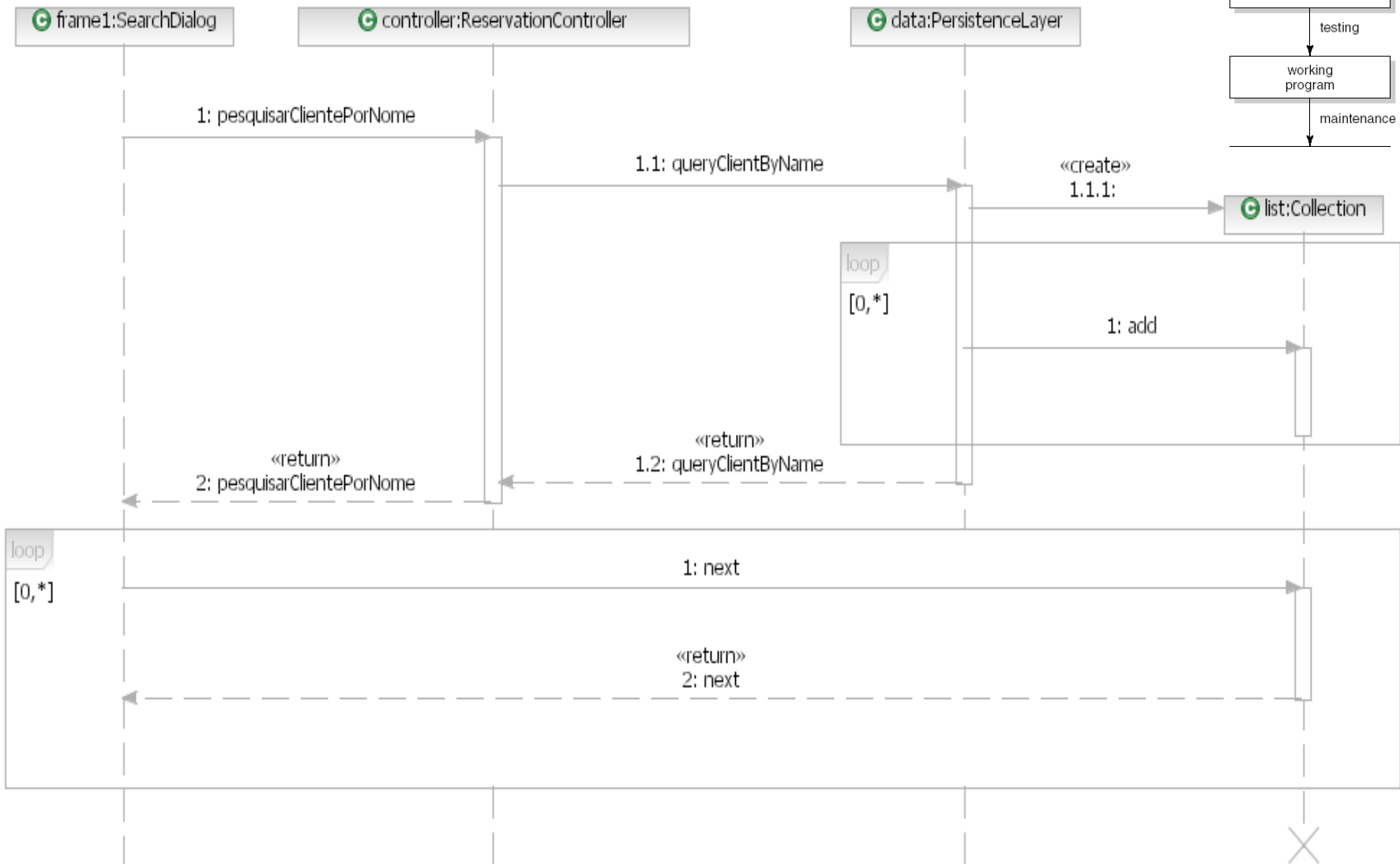
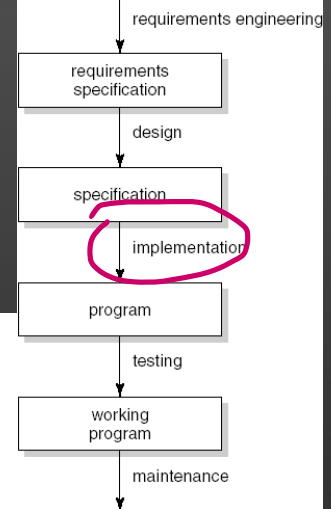


# Mostrar a realização de um caso de utilização



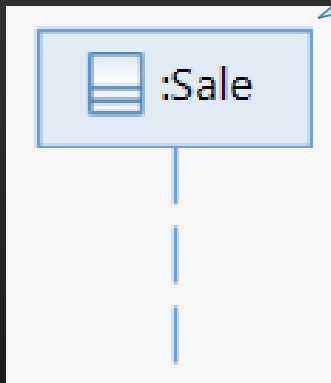
**FIGURE 6-9**  
Sequence Diagram  
of the Borrow  
Books Use Case  
for Students with  
a Valid ID and No  
Overdue Books  
or Fines

# Mostrar a colaboração entre objetos de código

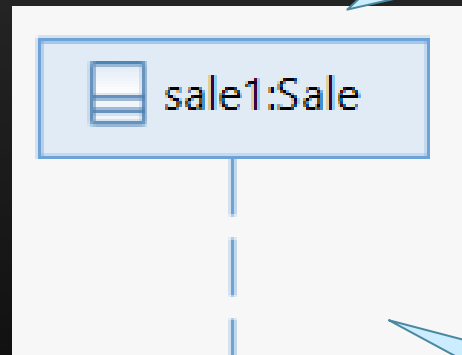


## Aspetos notacionais dos DS

Linha de vida a representar uma instância sem nome da classe Sale.

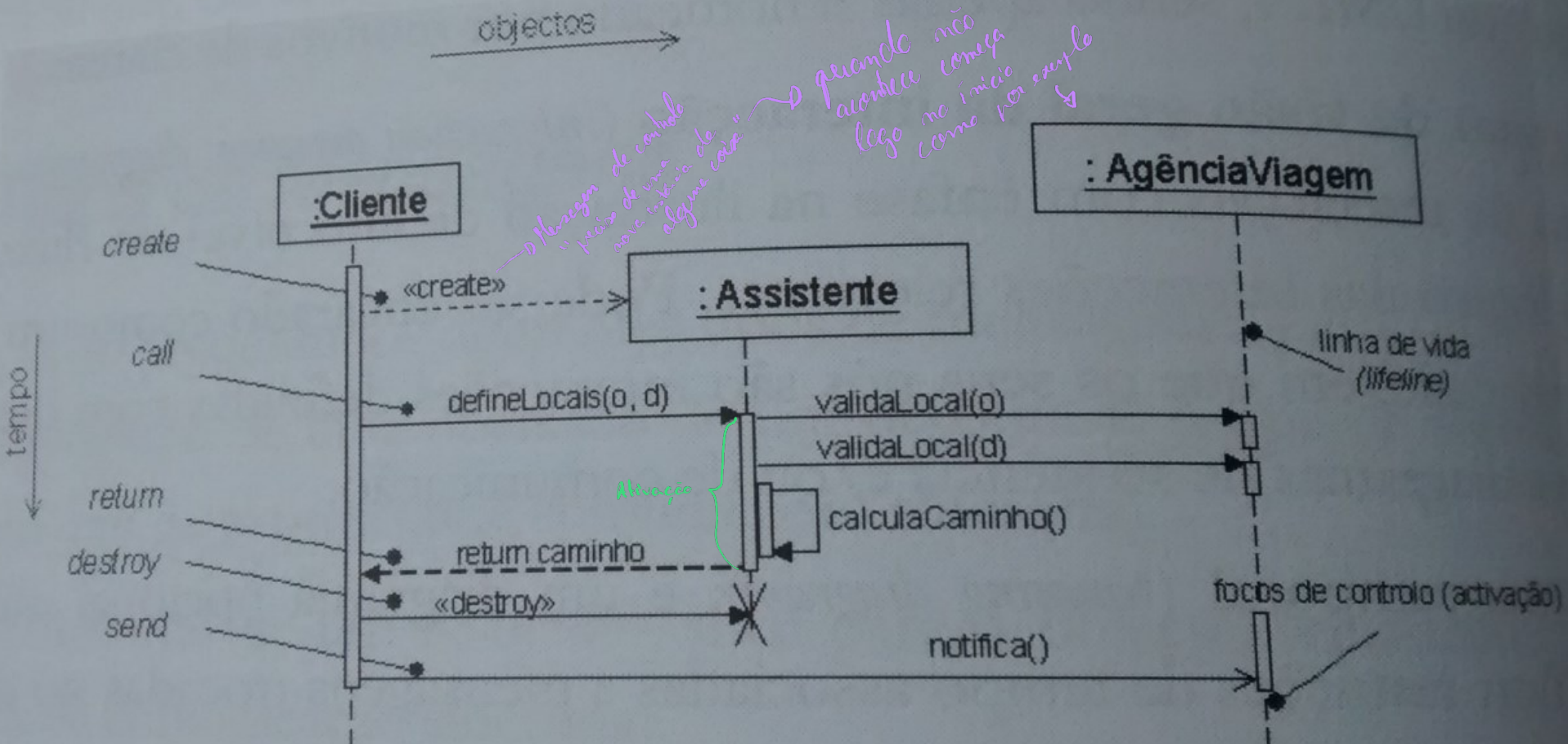



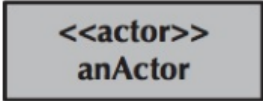
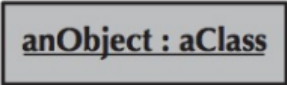

Linha de vida a representar uma instância chamada sale1 da classe Sale.



Em JAVA:  
`Sale sale1 = ... ;`

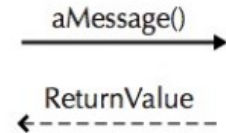
# Colaboração entre objetos por mensagens (síncronas)



Term and Definition	Symbol
<p><b>An actor:</b></p> <ul style="list-style-type: none"> <li>■ Is a person or system that derives benefit from and is external to the system.</li> <li>■ Participates in a sequence by sending and/or receiving messages.</li> <li>■ Is placed across the top of the diagram.</li> <li>■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with &lt;&lt;actor&gt;&gt; in it (alternative).</li> </ul>	 <p><b>anActor</b></p> 
<p><b>An object:</b></p> <ul style="list-style-type: none"> <li>■ Participates in a sequence by sending and/or receiving messages.</li> <li>■ Is placed across the top of the diagram.</li> </ul>	
<p><b>A lifeline:</b></p> <ul style="list-style-type: none"> <li>■ Denotes the life of an object during a sequence.</li> <li>■ Contains an X at the point at which the class no longer interacts.</li> </ul>	

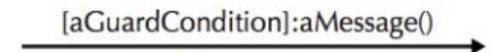
**A message:**

- Conveys information from one object to another one.
- A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.



**A guard condition:**

- Represents a test that must be met for the message to be sent.



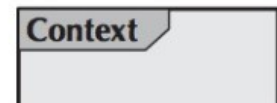
**For object destruction:**

- An X is placed at the end of an object's lifeline to show that it is going out of existence.

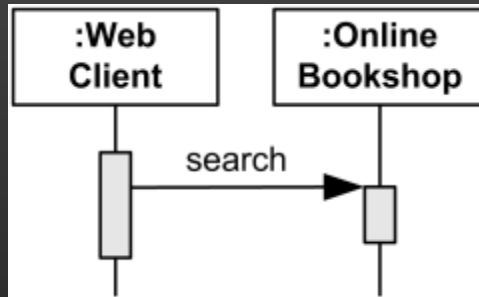


**A frame:**

- Indicates the context of the sequence diagram.



# Semântica da invocação



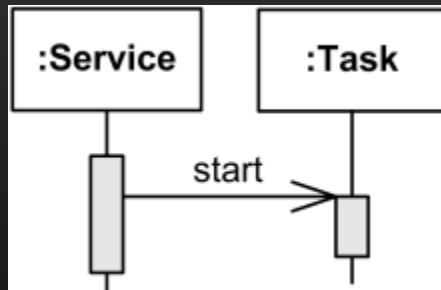
...

```
result = B.search();
```

...

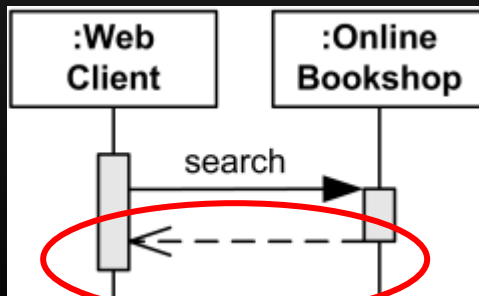
síncrona

↳ Espera pela resposta



assíncrona

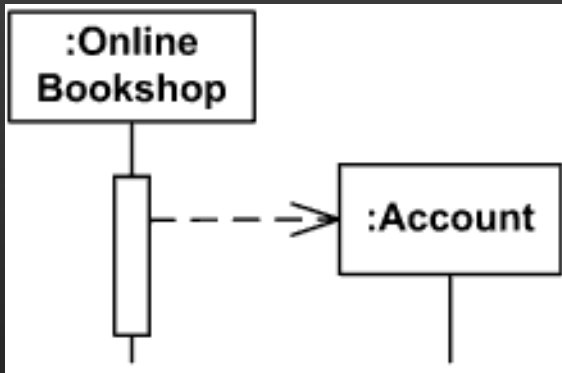
↳ Não bloqueia, não aguarda resposta



retorno



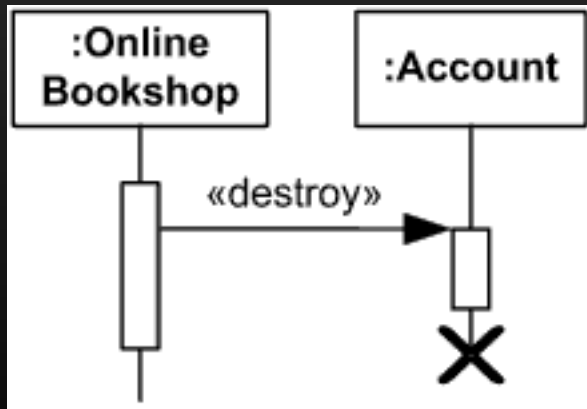
# Modelar a criação/destruição do participante



...

```
Account a= new Account()
```

...



destroy

...

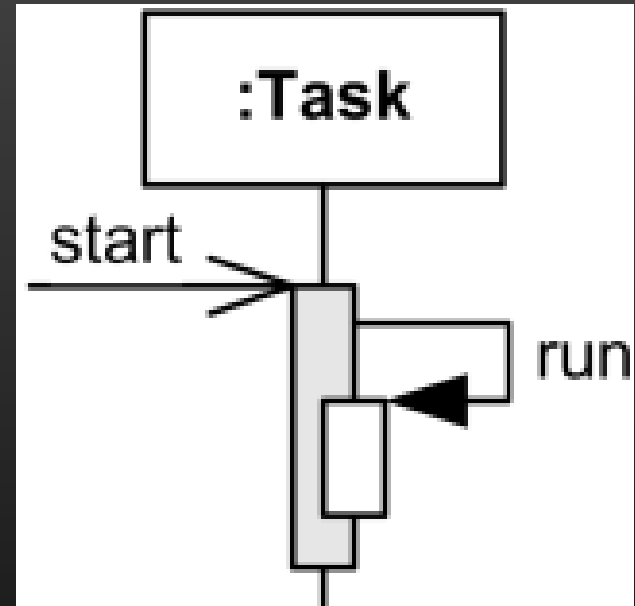
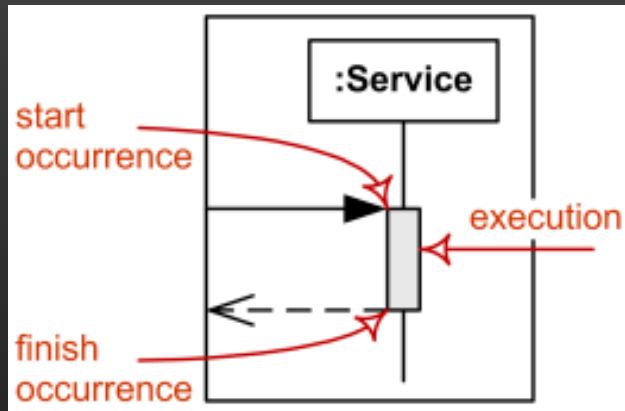
```
a=null; // in java
```

```
[a release] // objective C
```

```
free a; // C
```

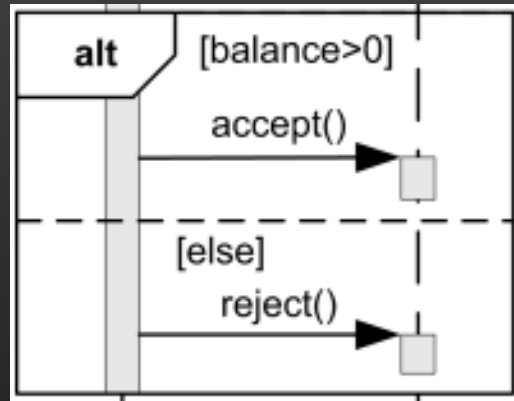
From <http://www.uml-diagrams.org/sequence-diagrams.html>

# Ativação



From <http://www.uml-diagrams.org/sequence-diagrams.html>

## Fragmentos para mostrar alternativas

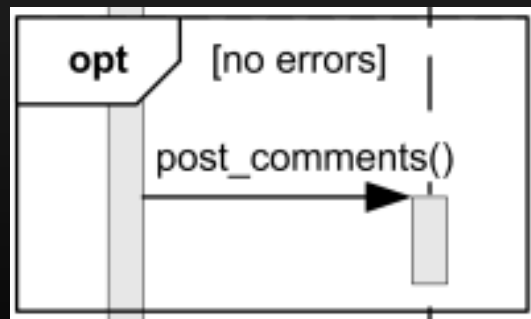


If (balance>0)

otherObj.Accept()

Else

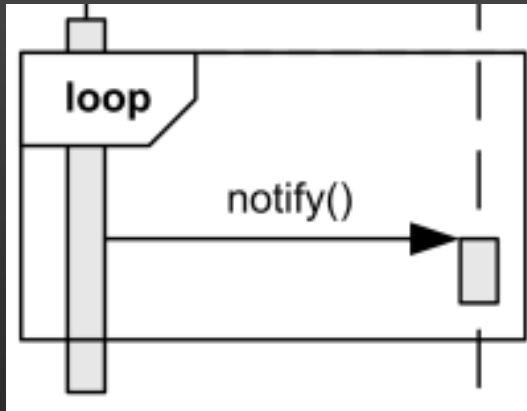
otherObj.Reject()



If ( no errors )

otherObj.Post\_comments()

## Fragmentos para mostrar loops



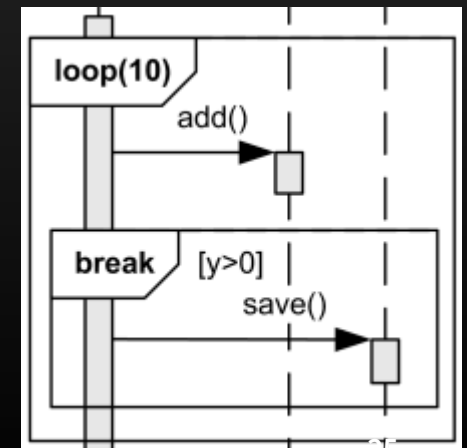
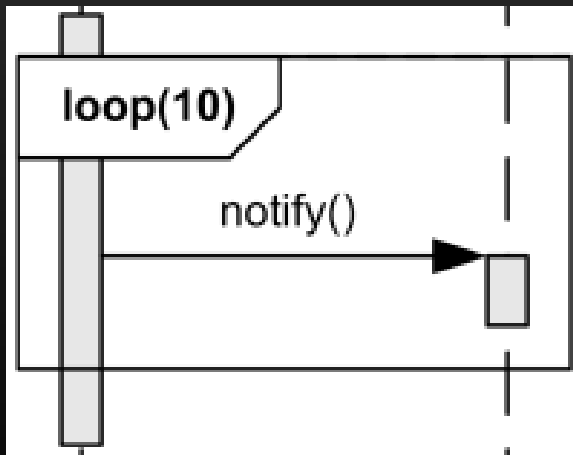
```
i=0;
While( i<10 )
{
```

```
    otherObj.Add()
```

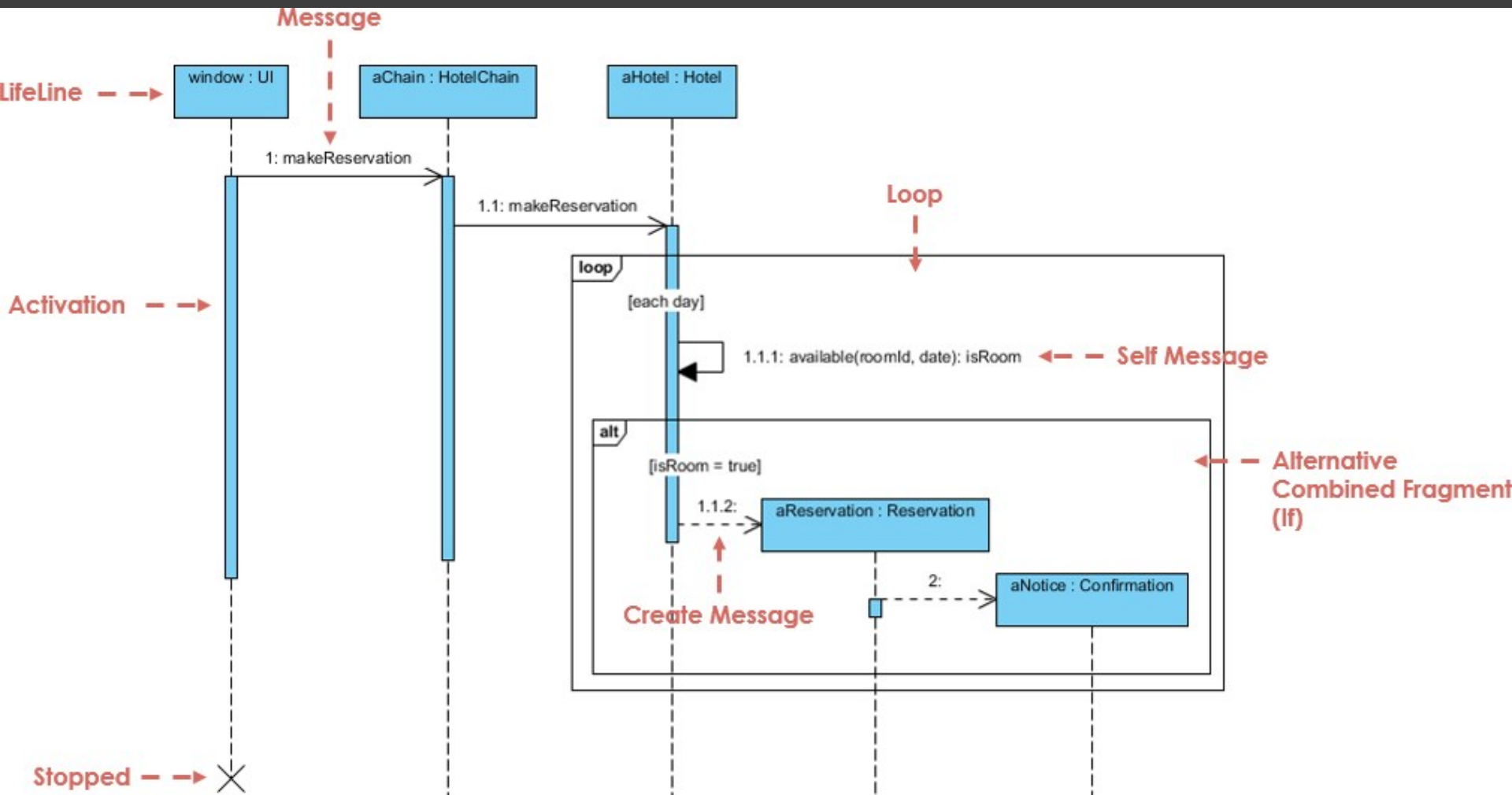
```
    If ( y>0 ) break;
```

```
    i++
```

```
}
```



# Principais elementos notacionais



# Quatro tipos de diagramas de iteração disponíveis

## D. Sequência

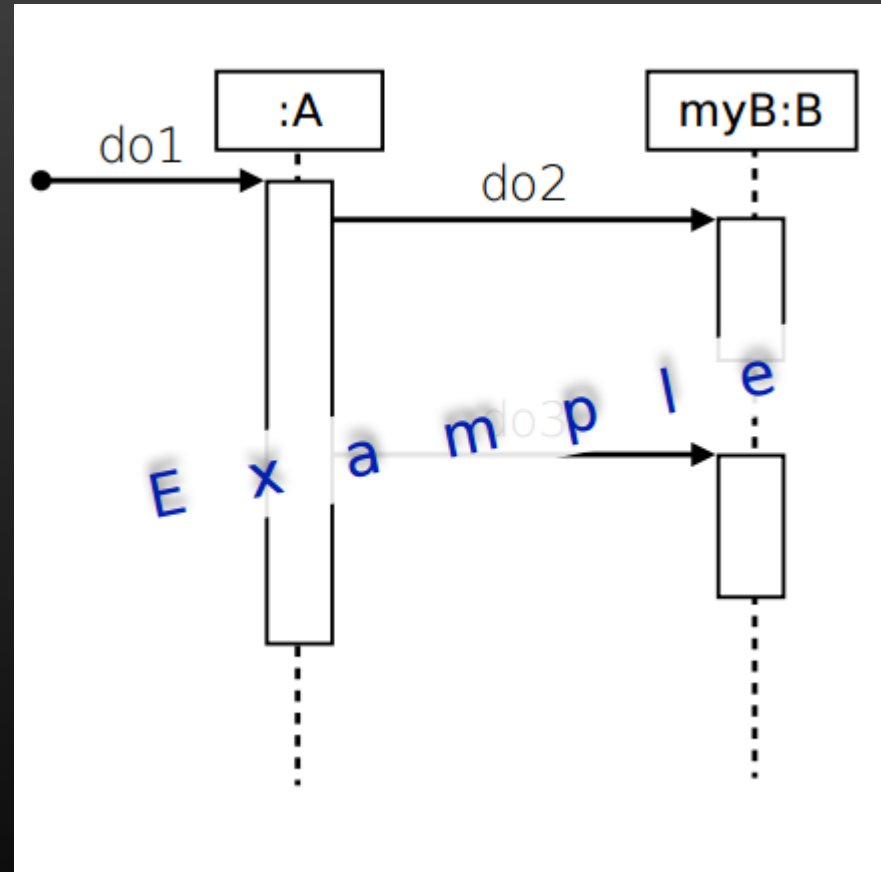
→ Formato alinhado

## D. Comunicação

Formato grafo

Diagrama temporal (*timming*)

Diagrama de visão geral da interação (*interaction overview*)



# Quatro tipos de diagramas de iteração disponíveis

## D. Sequência

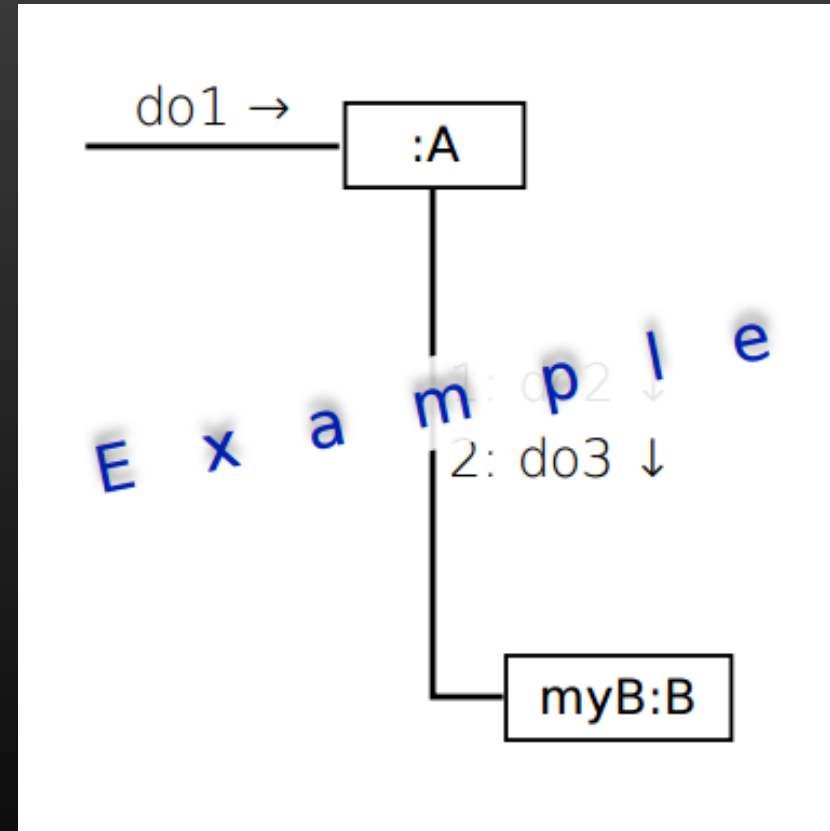
Formato alinhado

## → D. Comunicação

Formato grafo

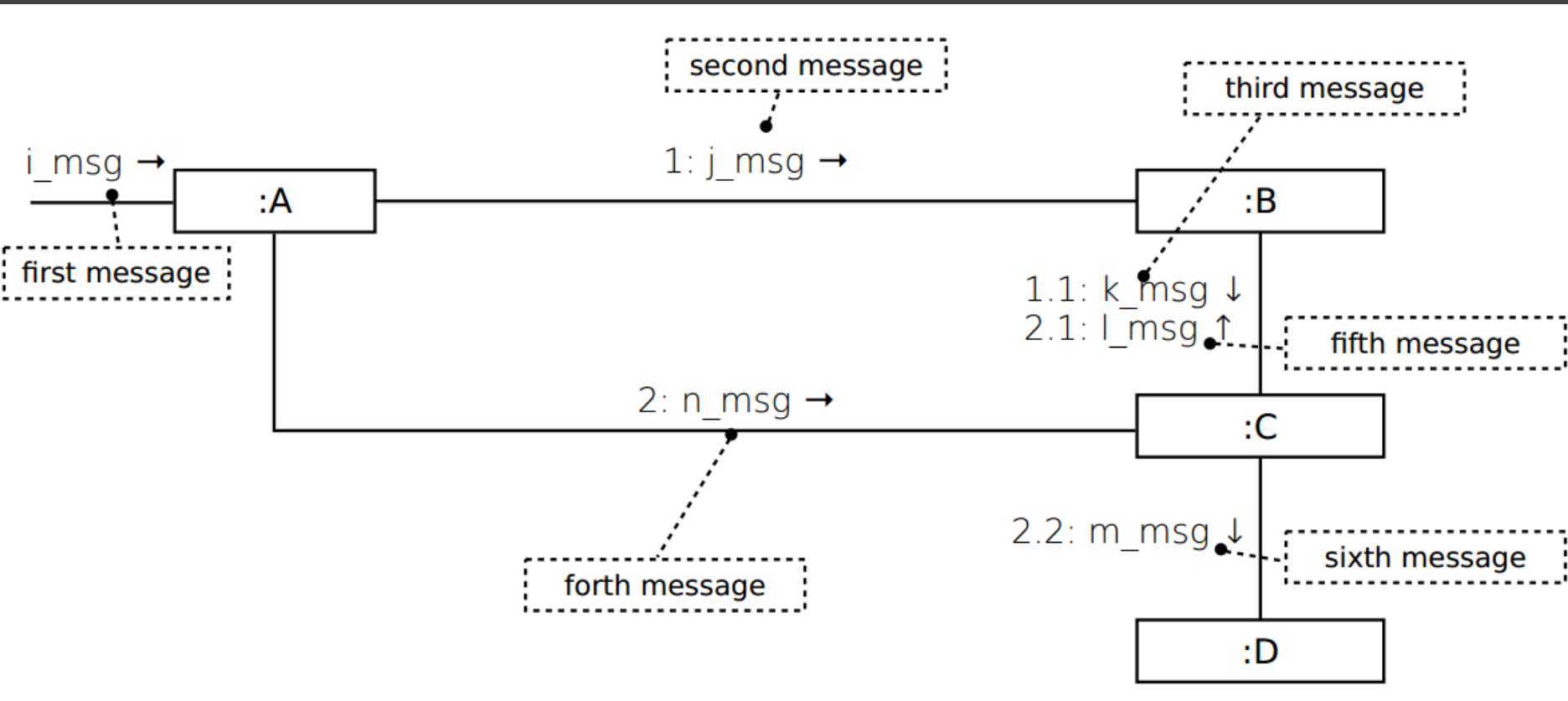
Diagrama temporal (*timming*)

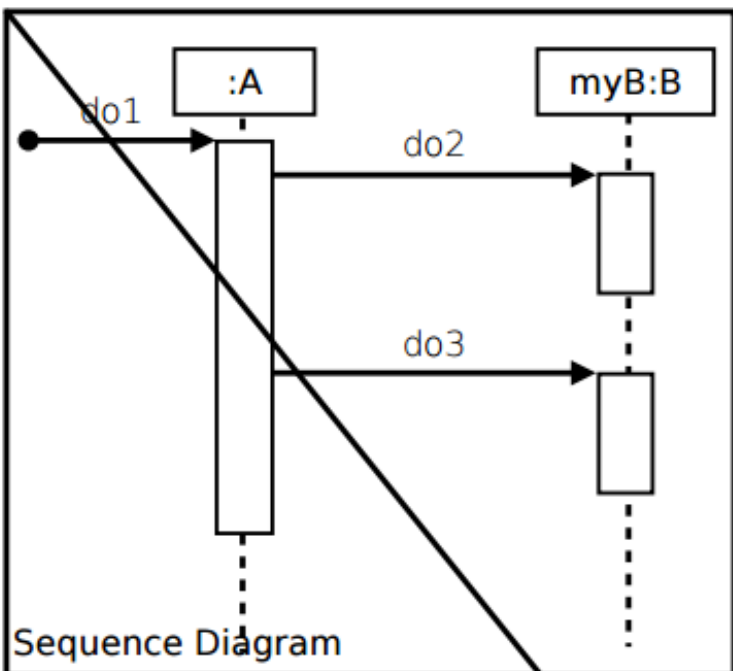
Diagrama de visão geral da interação (*interaction overview*)





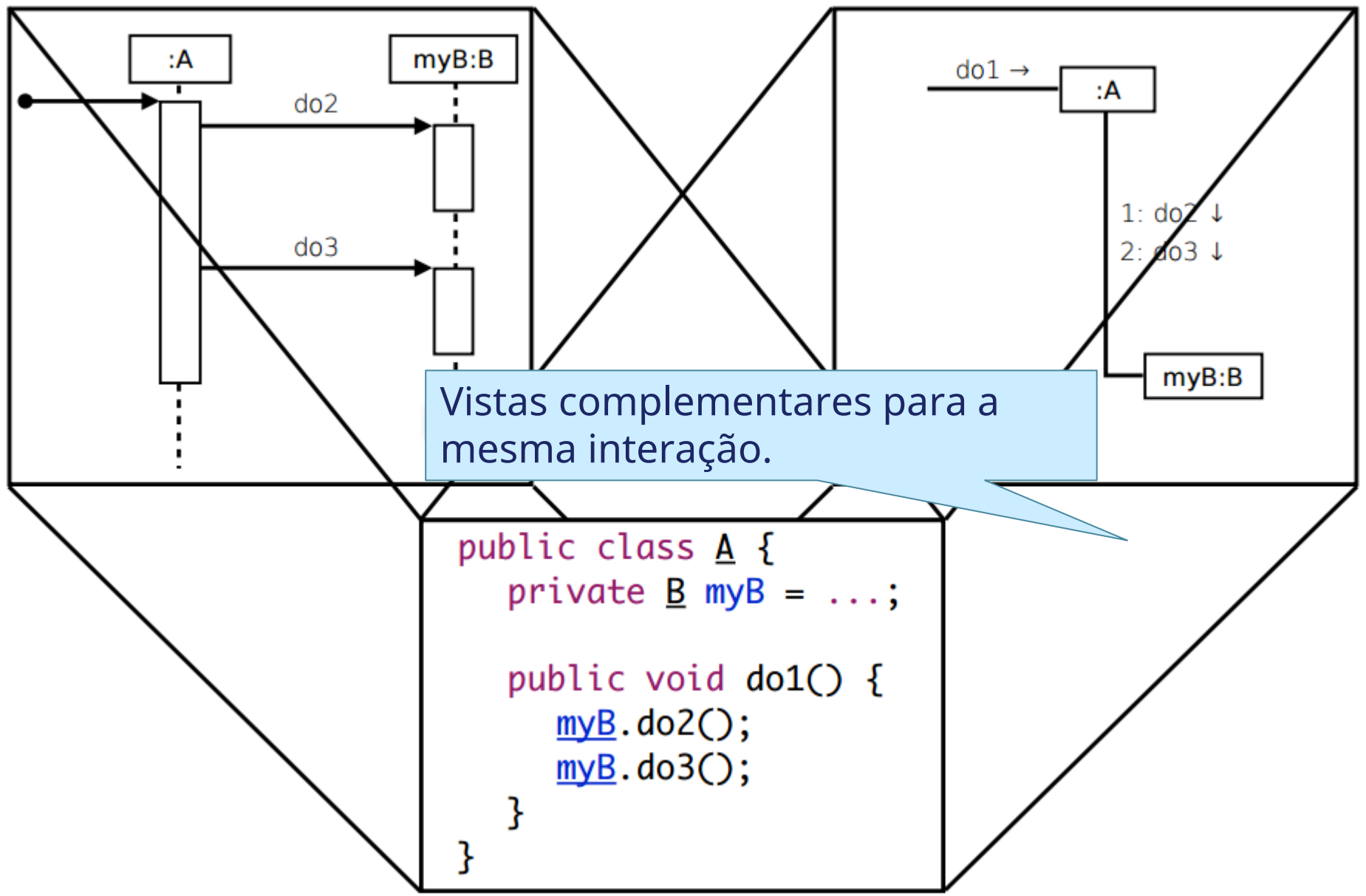
# Diagramas de comunicação



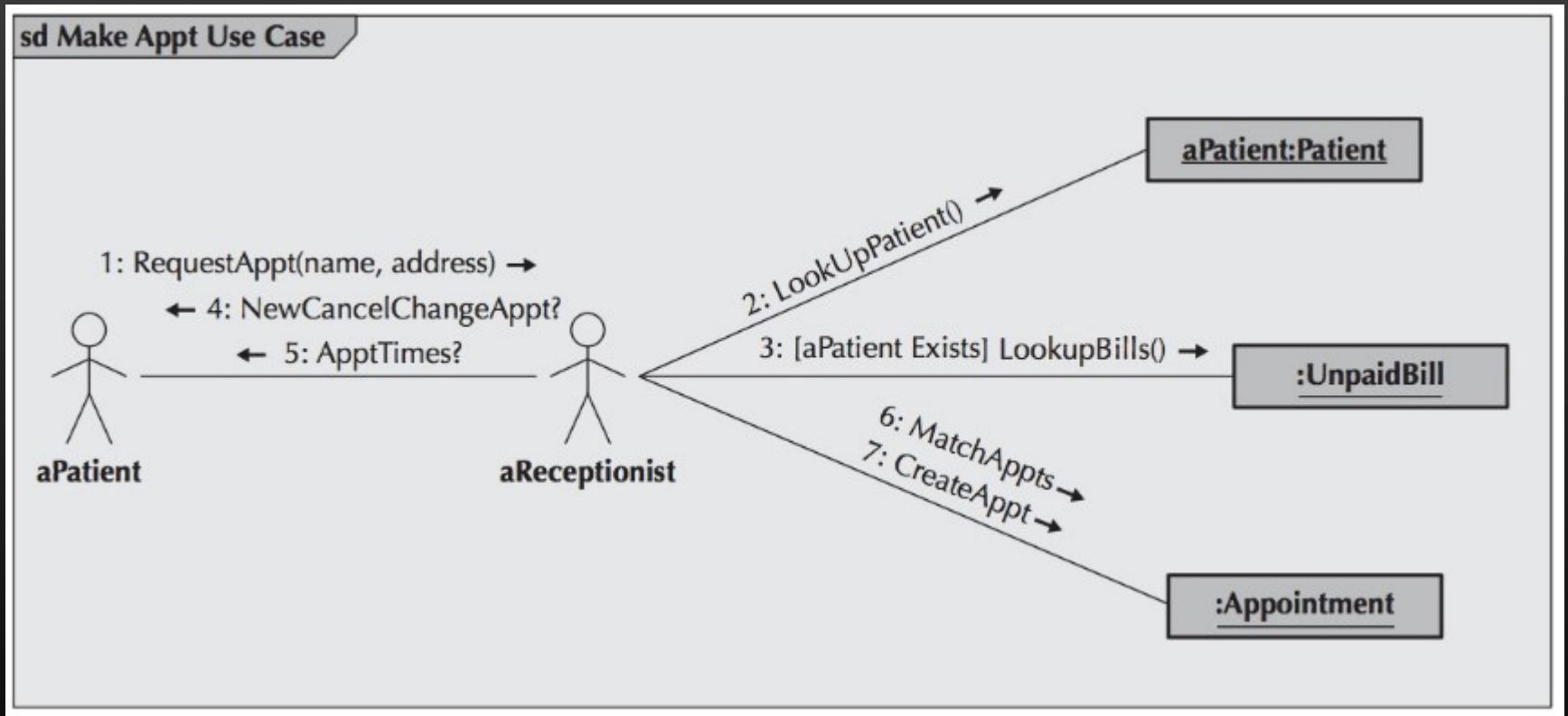


Os diagramas de podem ser usados para visualizar a colaboração entre objetos em Java.

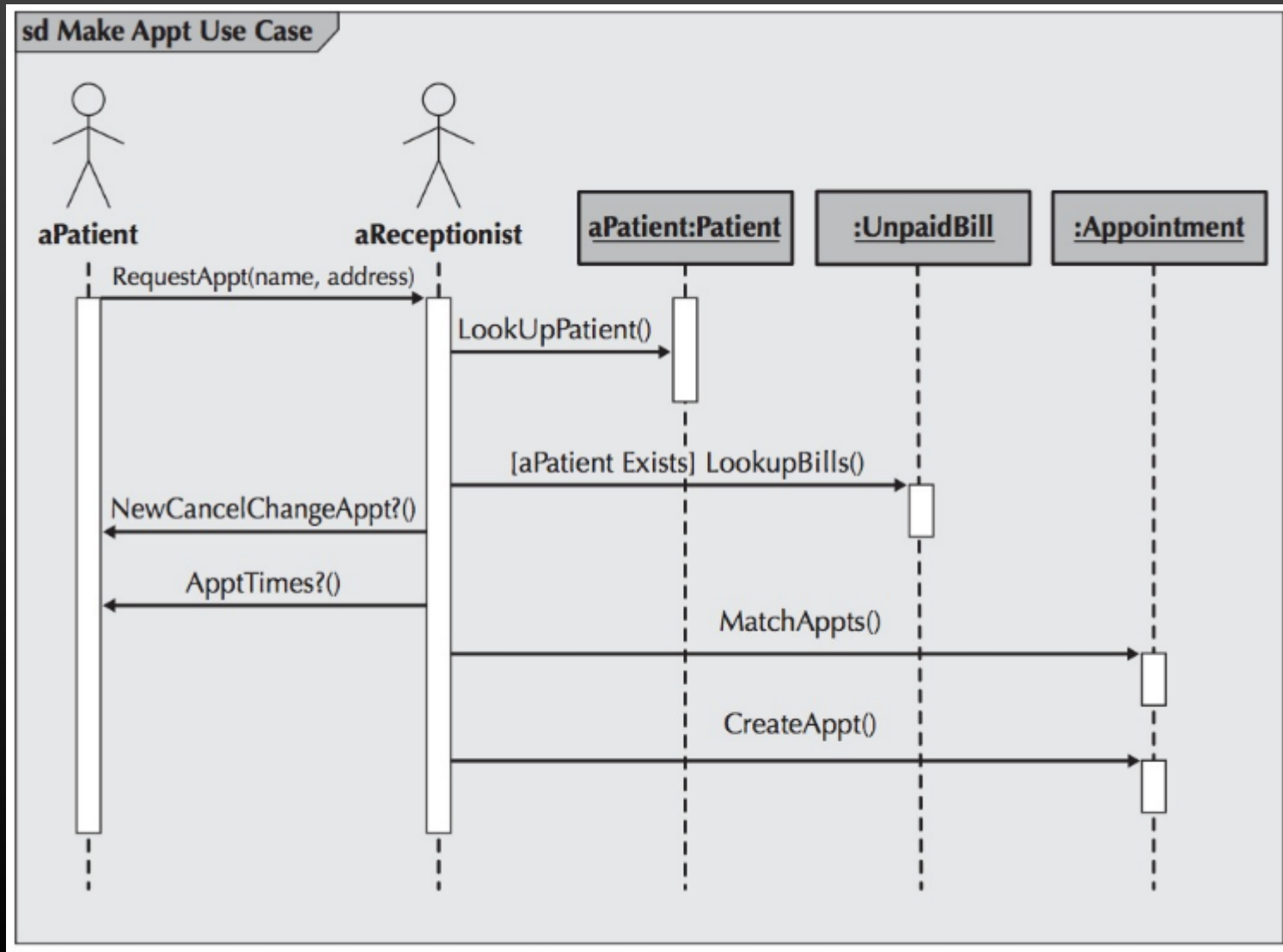
```
public class A {  
    private B myB = ...;  
  
    public void do1() {  
        myB.do2();  
        myB.do3();  
    }  
}
```



**Figure** Communication Diagram for a Scenario of the Make Patient Appt Use Case



**Figure** Sequence Diagram for a Scenario of the Make Patient Appt Use Case



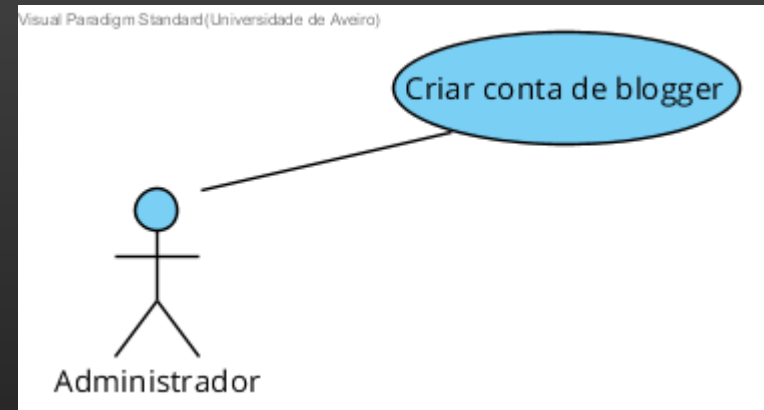
## Diagramas de sequência vs diagramas de comunicação

	Benefícios	Limitações
Diagrama de sequência	<ul style="list-style-type: none"><li>• Mostra claramente a sequência/ordem temporal das mensagens</li><li>• Possibilidades de notação alargadas</li></ul>	<ul style="list-style-type: none"><li>• Cresce para a direita à medida que se acrescentam objetos</li></ul>
Diagrama de comunicação	<ul style="list-style-type: none"><li>• Mais fácil de desenhar (objetos podem ser adicionado em qq parte)</li></ul>	<ul style="list-style-type: none"><li>• Menos expressivo (ordem temporal)</li><li>• Menos opções de notação</li><li>• Pouco suportado nas ferramentas UML</li></ul>

# Um caso de utilização é **realizado pela colaboração entre atores e sistema**

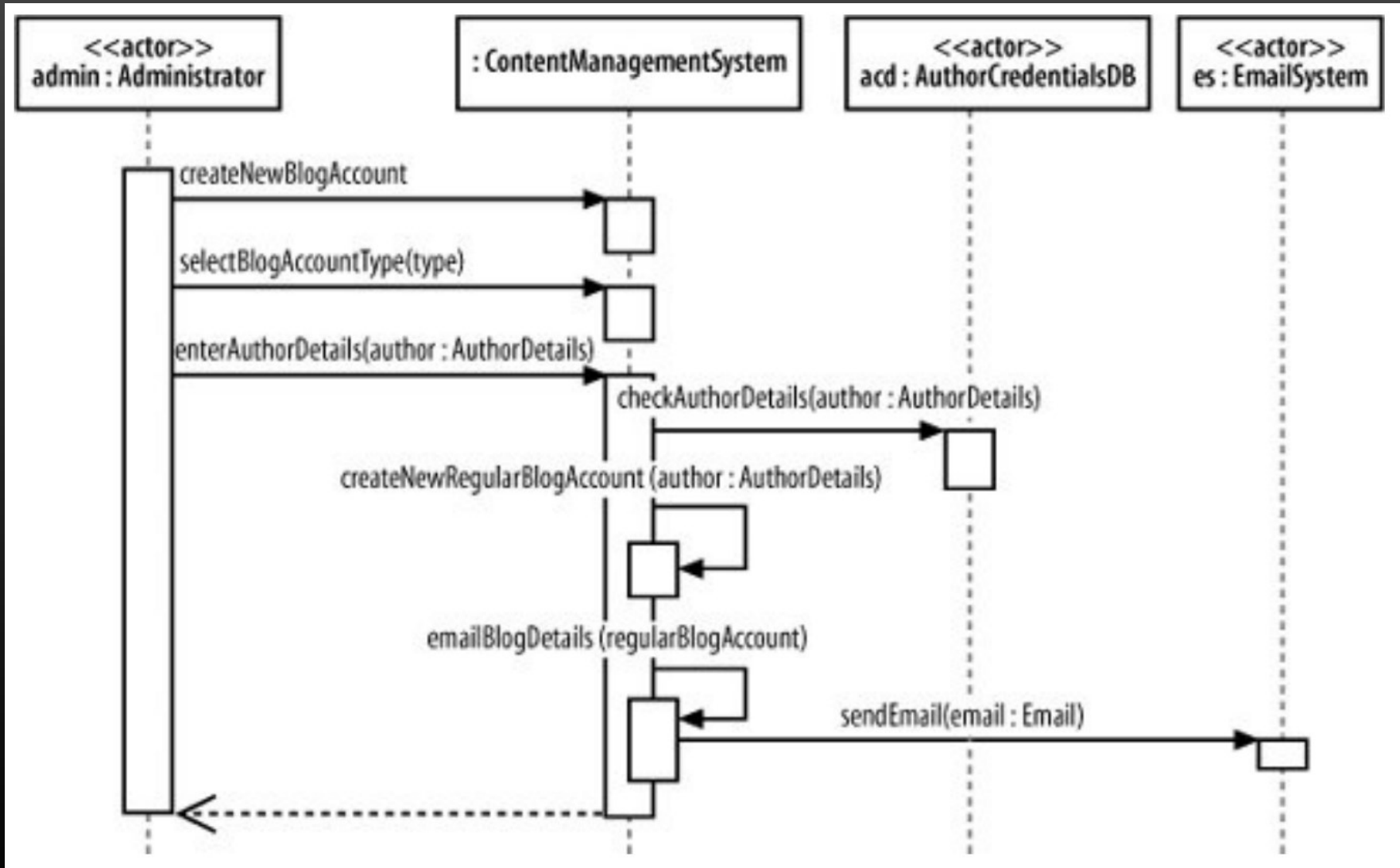
## FLUXO TÍPICO

1. O Administrador pede ao sistema (CMS) para criar uma nova conta de *blogger*
2. O Administrador escolhe o tipo Normal
3. O Administrador fornece os detalhes do autor do blog
4. O CMS verifica os detalhes (se já existe) na base de Dados de Autores
5. O CMS cria a nova conta normal.
6. Um sumário dos detalhes da nova conta são enviados por email ao autor.

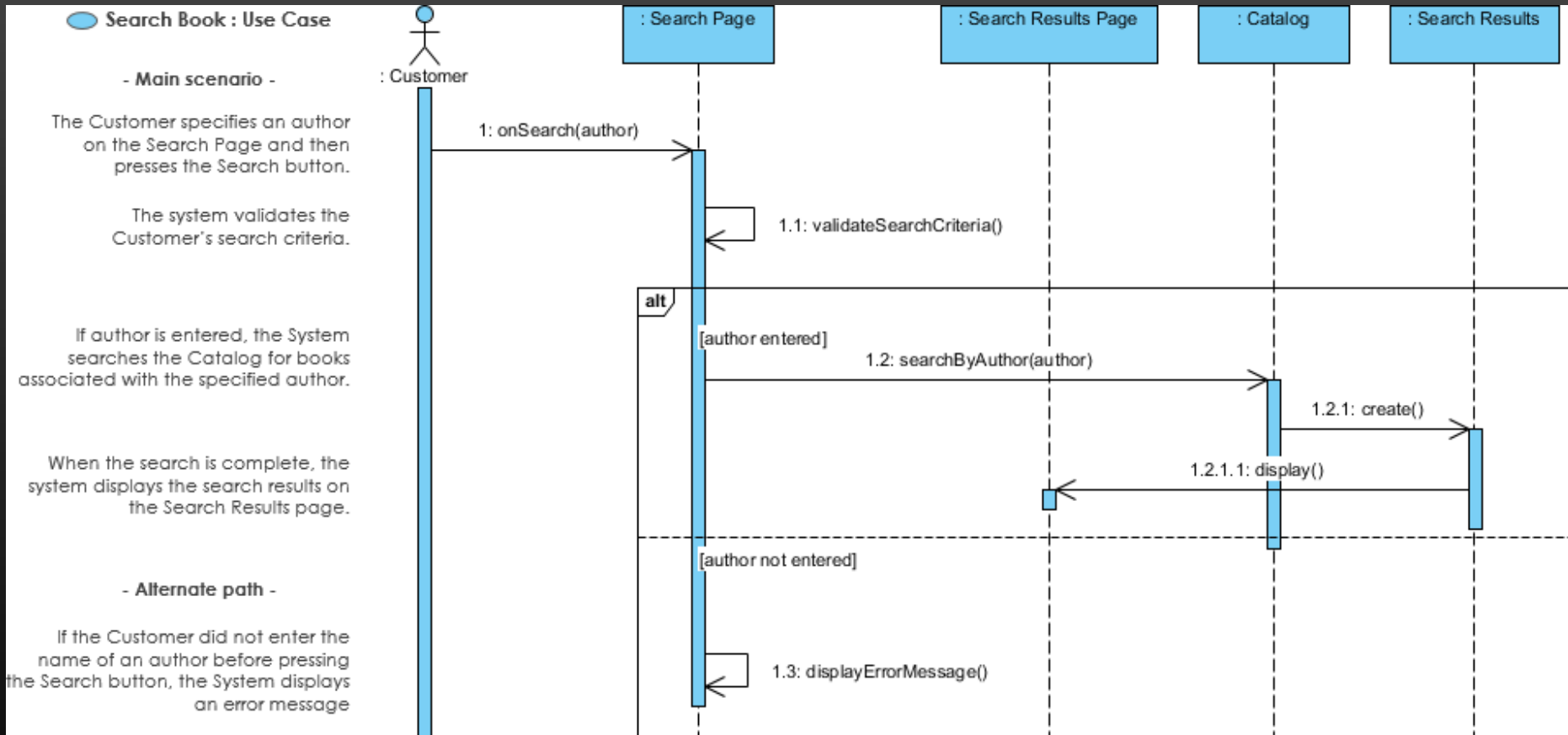




## Realização do caso de utilização como uma colaboração entre objetos



# A realização de um caso de utilização

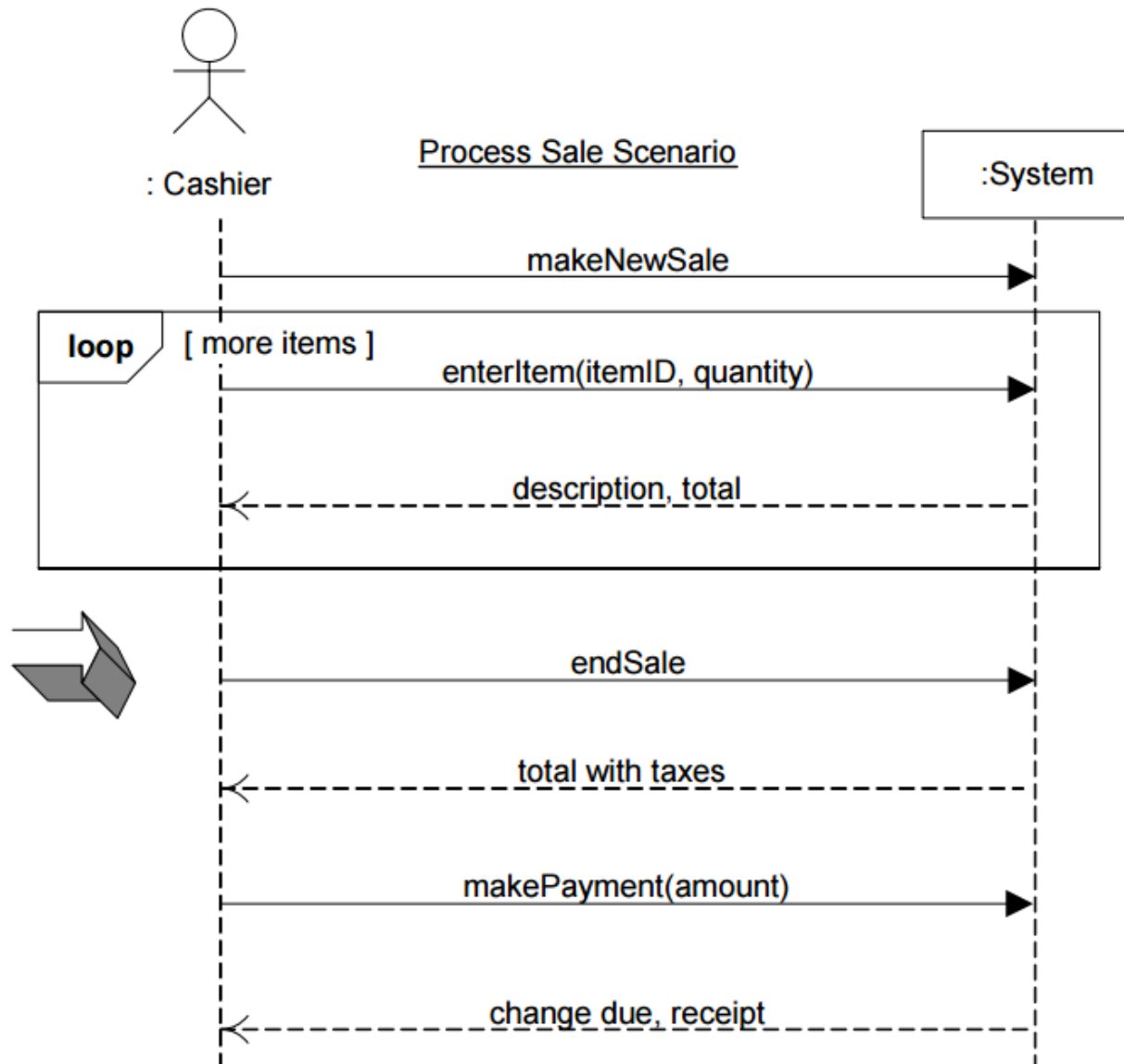


# Dos casos de utilização para o sistema (método Larman)

## Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
  2. Cashier starts a new sale.
  3. Cashier enters item identifier.
  4. System records sale line item and presents item description, price, and running total.
- Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
  6. Cashier tells Customer the total, and asks for payment.
  7. Customer pays and System handles payment.

...



# Diagramas de sequência de sistema (DSS)

Mostram, para um cenário de um CaU:

- os eventos que os atores externos geram,
- a sua ordem temporal,
- as necessidades de integração entre sistemas

Estilo para a construção

O sistema é tratado como uma caixa-fechada

- Não mostra componentes internos

Mensagens que chegam ao Sistema são modeladas como operações

- Os serviços solicitados/invocados pelos atores

system as black box

the name could be "NextGenPOS" but "System" keeps it simple

the ":" and underline imply an instance, and are explained in a later chapter on sequence diagram notation in the UML

external actor to system

### Process Sale Scenario

: Cashier

:System

makeNewSale

a UML loop **interaction frame**, with a boolean **guard** expression

loop

[ more items ]

enterItem(itemID, quantity)

description, total

endSale

total with taxes

makePayment(amount)

change due, receipt

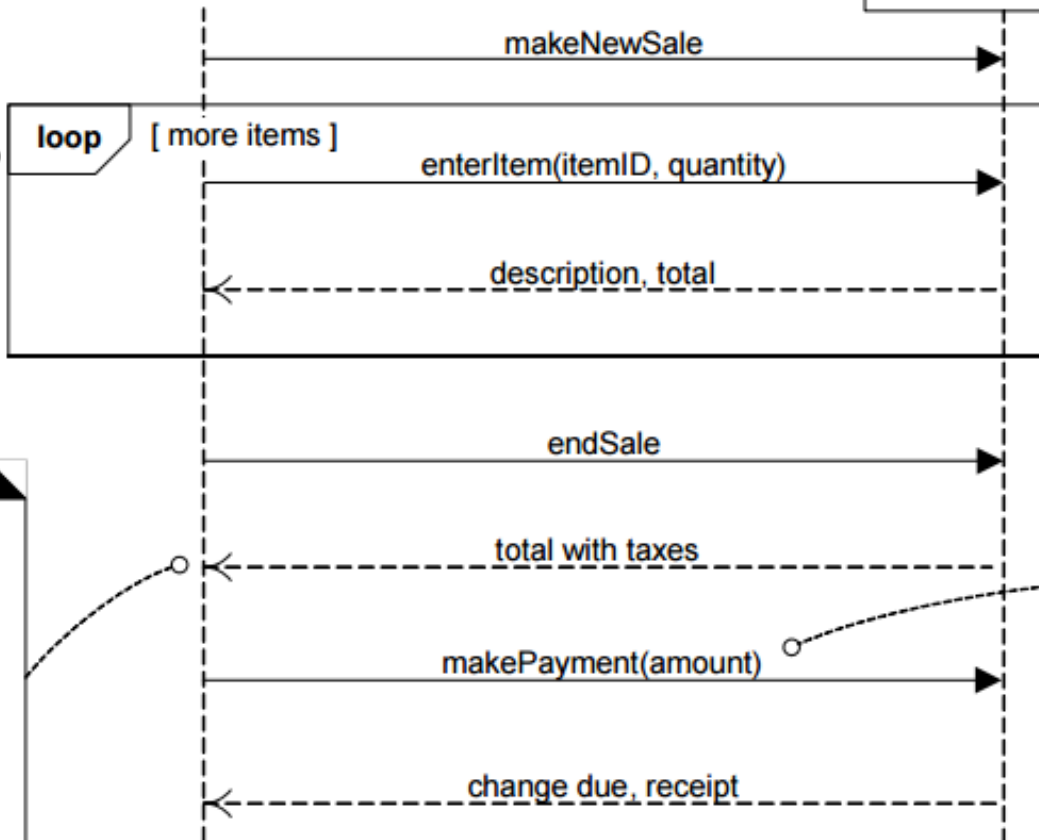
return value(s) associated with the previous message

an abstraction that ignores presentation and medium

the return line is optional if nothing is returned

a message with parameters

it is an abstraction representing the system event of entering the payment data by some mechanism



## DSS

um novo diagrama de sequência de sistema (DSS) p/ cada CaU

uma *lifeline* para o actor primário (ou actores) e o sistema.

O sistema é modelado por uma classe que o representa globalmente

Opcionalmente, pode-se incluir o texto da descrição do CaU

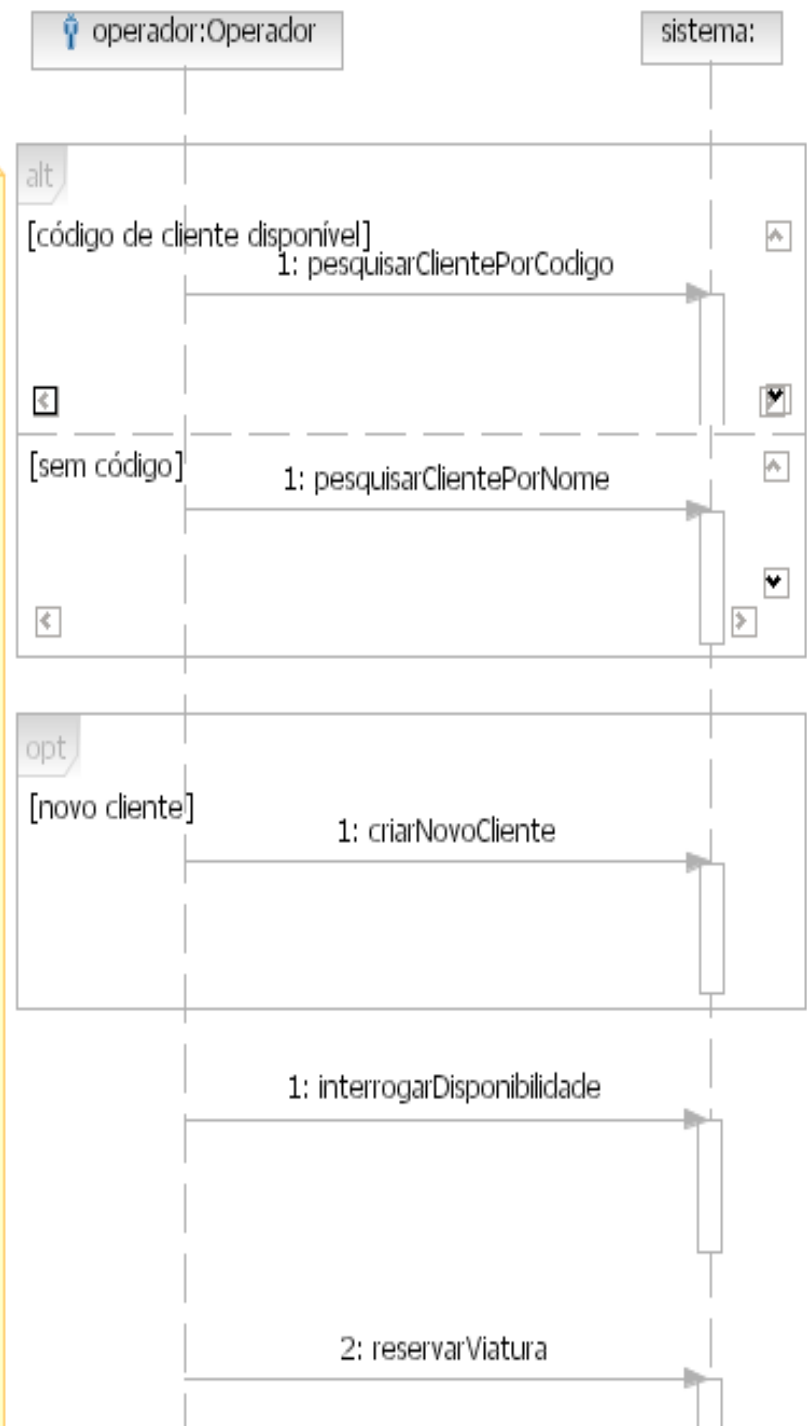
Iniciado quando um cliente telefona para o callCenter para solicitar uma reserva.

O operador pesquisa o cliente por código ou nome.

Se o cliente ainda não existe no sistema, os dados desse novo cliente são recolhidos e o cliente registrado.

Os elementos da reserva são recolhidos pelo operador, que verifica se existe disponibilidade para o período pretendido. Nesse caso, a reserva é confirmada.

O cliente é informado do código de reserva (gerado pelo sistema).



# Os DSS mostram operações de sistema

Uma **operação de sistema** (OpS) corresponde a um ponto de entrada no sistema

Encapsula um conjunto de interações subjacentes entre objetos, necessárias para realizar essa operação.

**Etapas seguintes na construção do modelo dinâmico:**

mostrar, para cada OpS, a rede de objetos que vai ser ativada.

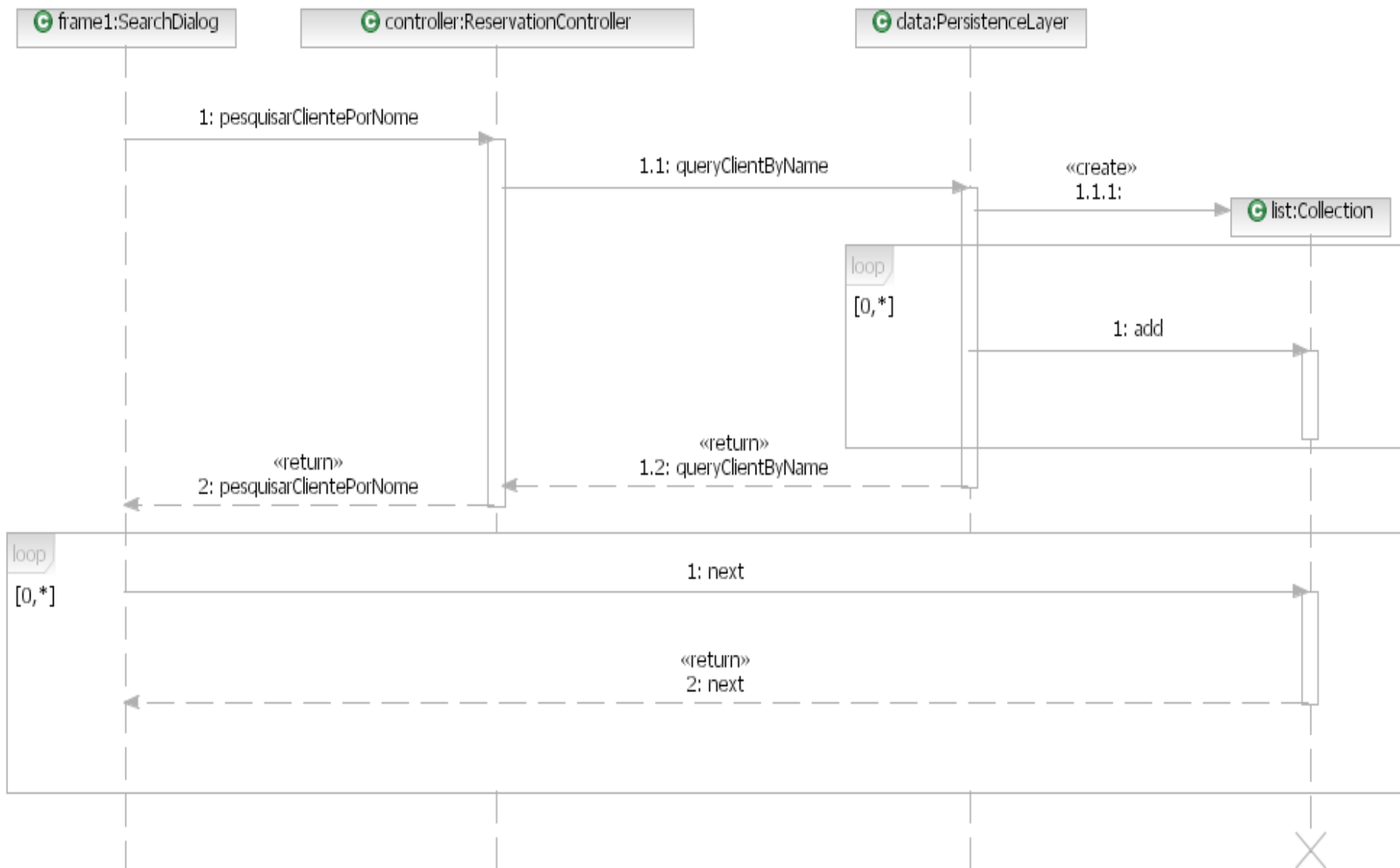
Estes objetos são instâncias de classes que devemos identificar

Como escolher as classes que participam na solução?

Essa atividade é o que designamos por desenho

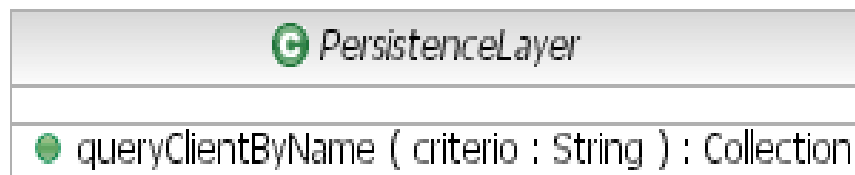
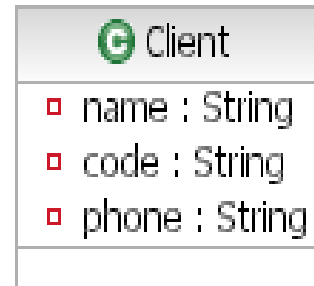
**Ponto essencial: distribuir responsabilidades pelos objetos**

segundo os princípios do desenho por objectos (*Object-Oriented Design*).



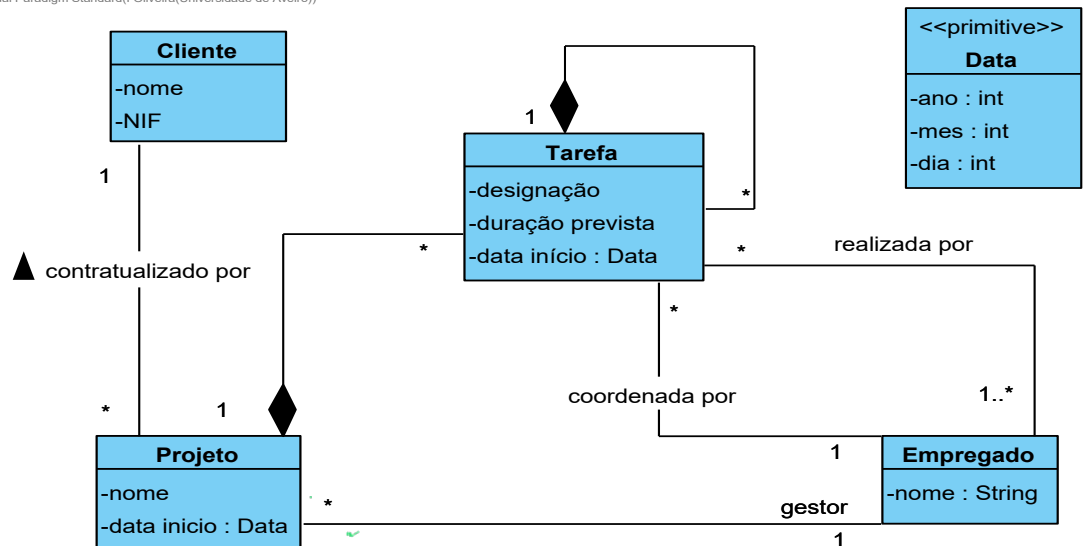


## Os diagramas de interacção ajudam a distribuir responsabilidades → encontrar os métodos das classes



# Distribuição de responsabilidades pelos objetos

Visual Paradigm Standard (I Oliveira (Universidade de Aveiro))



checkIn() / checkOut()

Começar a trabalhar numa tarefa /  
terminar

Empregado

reportarEstadoProjeto()

calcula a % de execução do projeto  
(tarefas já concluídas / previstas)

Projeto

formatarISO8601()

Representa uma data no formato  
aaaa/mm/dd

Data

atribuirGestor(Empregado  
emp1)

Define o gestor de um projeto

Empregado

contaCorrente()

Lista os movimentos lançados para  
um cliente, no presente ano fiscal.

## Referências

[DEN'15] Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems analysis and design: An object-oriented approach with UML*. John Wiley & Sons.

→ chap. 6

[LAR'12] Larman, C. (2012). *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development*. Pearson Education.

→ chap. 10, chap. 15.