

DIM0436  
6. UML II

Richard Bonichon

201400807

# Outline

- 1 Diagramas de sequências
- 2 Diagramas de objetos
- 3 Diagramas de estados
- 4 Diagramas de atividades
- 5 Exercícios

1 Diagramas de sequências

2 Diagramas de objetos

3 Diagramas de estados

4 Diagramas de atividades

5 Exercícios

## Objetivo

- Modelizar um fluxo de controle.
- Ilustrar cenários típicos.

\*

• O diagrama de seqüência é um tipo de diagrama de interação:

- Diagramas de iteração descrevem como grupos de objetos colaboram entre si.
- Normalmente um diagrama de seqüência captura o comportamento de um único cenário.

Dá ênfase à ordenação temporal das mensagens trocadas entre os objetos.

- Durante a fase de **análise de requisitos**

- Durante a fase de **análise de requisitos**
  - ▶ Para refinar casos de uso

- Durante a fase de **análise de requisitos**
  - ▶ Para refinar casos de uso
  - ▶ Para achar objetos adicionais **objetos participantes**

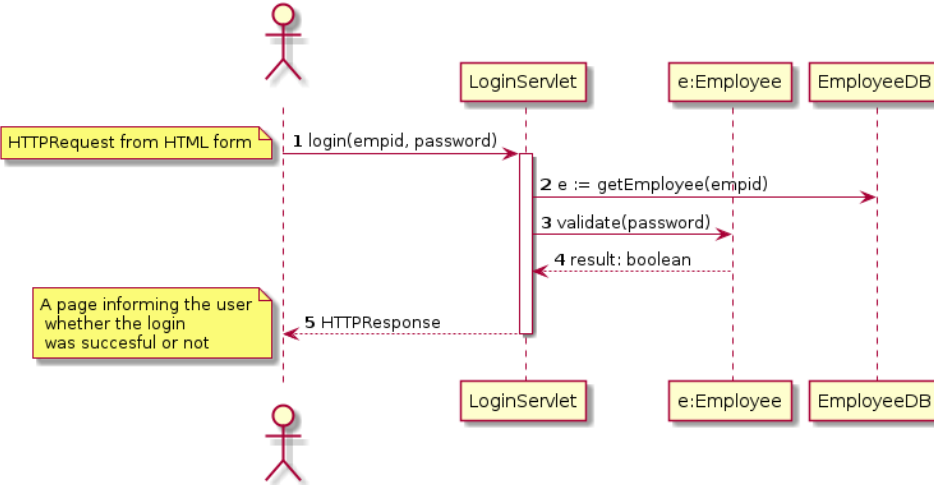
- Durante a fase de **análise de requisitos**
  - ▶ Para refinar casos de uso
  - ▶ Para achar objetos adicionais **objetos participantes**
- Durante o desenho do sistema



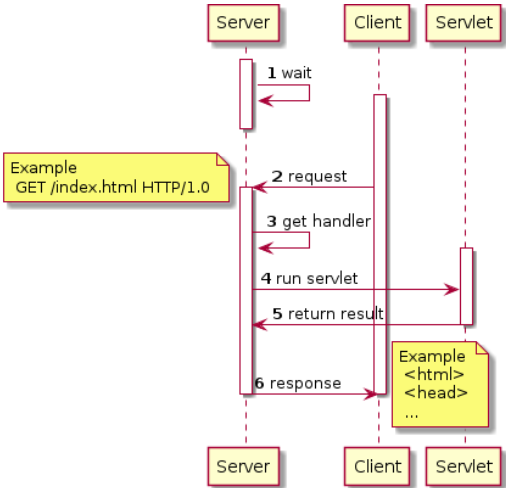
- Durante a fase de **análise de requisitos**
  - ▶ Para refinar casos de uso
  - ▶ Para achar objetos adicionais **objetos participantes**
- Durante o desenho do sistema
  - ▶ Para refinar interfaces de sub-sistemas

- Objetos em cima
- Atores anônimos
- **lifelines** (linhas de vida) são linhas saindo dos objetos
- Setas representam mensagens
  - ▶ Rótulos são nomes
  - ▶ Parâmetros entre (), o perto de **data tokens**
- **Tempo** é a dimensão vertical

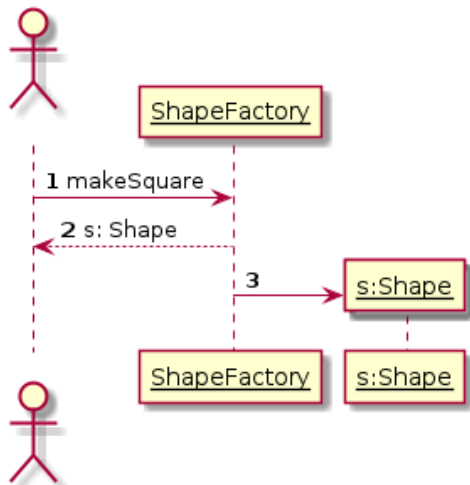
# Exemplo:



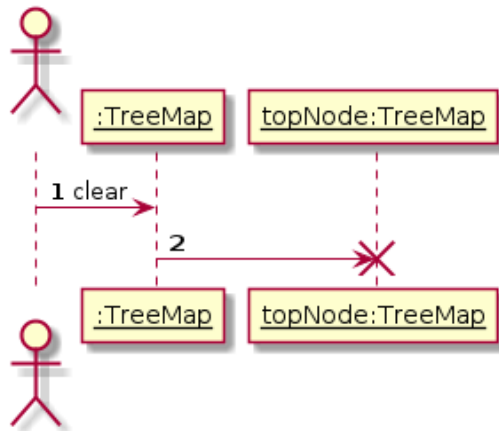
# Exemplo:

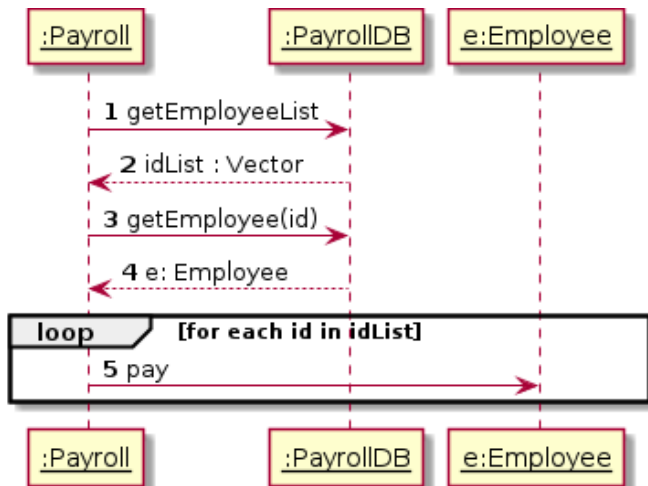


# Criação

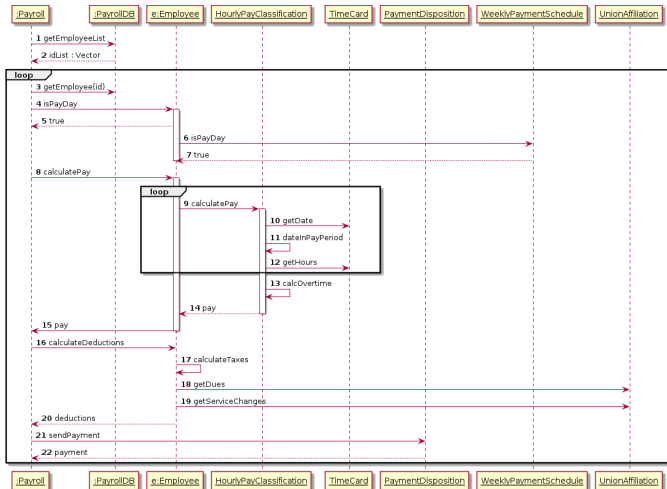


## Releasing an object to the garbage collector





# Legibilidade





# Seja minimalista

- Não desenhe diagramas de sequências grandes
  - ▶ Ninguém sabe lê-los
  - ▶ Ninguém vai lê-los
  - ▶ Perda de tempo
- Desenhe diagramas menores e essenciais
  - ▶ Não mais que uma página com muito espaço para explicações
- Não desenhe dezenas ou centenas de diagramas

# Código ou diagramas ?

```
1 public class Payroll {
2     private PayrollDB itsPayrollDB;
3     private PayrollDisposition itsDisposition;
4
5     public void doPayroll() {
6         List employeeList = itsPayrollDB.getEmployeeList();
7         for (Iterator iterator = employeeList.iterator();
8             iterator.hasNext();) {
9             String id = (String) iterator.next();
10            Employee e = itsPayrollDB.getEmployee(id);
11            if (e.isPayDay()) {
12                double pay = e.calculatePay();
13                double deductions = e.calculateDeductions();
14                itsDisposition.sendPayment(pay - deductions);
15            }
16        }
17    }
18 }
```

*When code can stand on its own, then diagrams are redundant and wasteful.*

# Vantagens

- um **bom** diagrama de sequência ainda fica mais alto-nível que o código real
- Diagramas de sequência são independentes da linguagem de implementação
- Não implementadores podem escrever diagramas de sequência
- Mais simples de fazer em time (que o código)
- Pode ver objetos/classes de uma vez só numa página.

# Protocolo simétrico Needham-Schröder

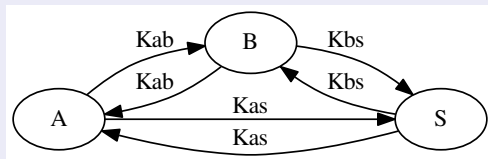
- Muitos protocolos baseados nas ideias de Needham-Schröder (1978), como Kerberos
- Protocolo tipo *Shared-key authentication*
  - ▶ Chave de sessão
  - ▶ Criptografia simétrica
- Sem infraestrutura de chave pública (PKI)

# Visão geral

## Participantes

- Dois participantes A e B desejando comunicar-se
- Um servidor de chave confiável S

## DFD



## Hipóteses

A (resp. B) tem uma comunicação simétrica segura com S usando Kas (resp. Kbs)

# Nonce e timestamp

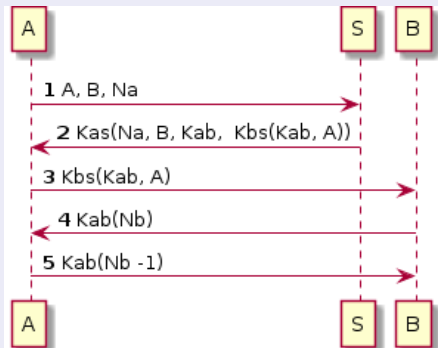
Um **nonce** (*number used once*) é um número aleatório, incluído nas mensagens. Se um *nonce* for gerado e enviado por A em uma etapa e retornado por B numa mensagem mais tarde, A sabe que a mensagem de B é novo e não é um replay de uma outra troca.

Um *nonce* não é um *timestamp*. A única suposição é que ele não foi usado numa troca antiga com alta probabilidade.

# Protocolo criptográfico: Needham-Schröder

## Perguntas sobre todo protocolo

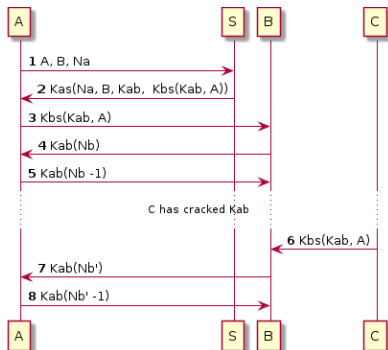
- O que o remetente quer dizer com a mensagem ?
- O que o destinatário tem direito de acreditar depois da recepção da mensagem ?



- Autenticação e sigilo assegurado?
- É possível fazer-se passar uma ou mais das partes?
- É possível interpor mensagens de uma troca anterior (ataque de repetição)?
- Que ferramentas podem implantar um atacante?
- Se alguma chave for comprometida, o que são as consequências ?



# Reuso de sessões antigas



## Problema

Message 3 ( $K_{bs}(K_{ab}, A)$ ) não é protegido por *nonce*. É impossível saber se  $K_{ab}$  é atual. Um intruder tem tempo ilimitado para descobrir uma chave antiga como se ela fosse nova.

## Ataque

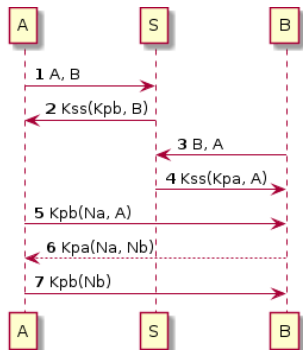
Um funcionário execute os primeiros passos do protocolo para juntar vários  $K_{bs}(K_{ab}, A)$  para todos os servidores B. Uma vez demitido, ainda pode usar os servidores.

# Protocolo assimétrico Needham-Schroeder

Alice e Bob usam um servidor confiado por ambas partes para distribuir chaves públicas.

- $K_{pa}$  e  $K_{sa}$ , partes públicas e privadas duma chave de A
- $K_{pb}$  e  $K_{sb}$  para B
- $K_{ps}$  e  $K_{ss}$  para S

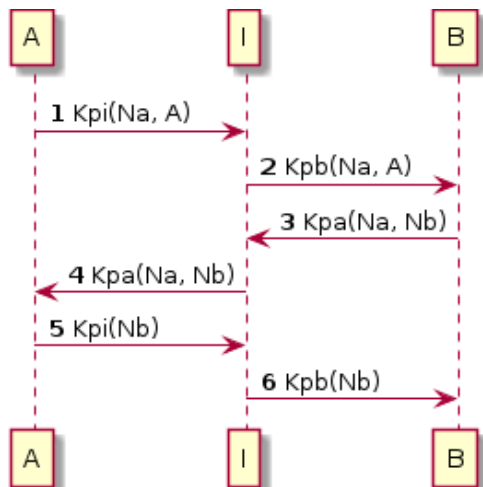
# Modelização



# Ataque Needham-Schröder

\* O protocolo é vulnerável a uma ataque *man-in-the-middle*.

\*



1 Diagramas de sequências

2 Diagramas de objetos

3 Diagramas de estados

4 Diagramas de atividades

5 Exercícios

*When they are needed, they are indispensable(...)*

Um diagrama de objetos mostra:

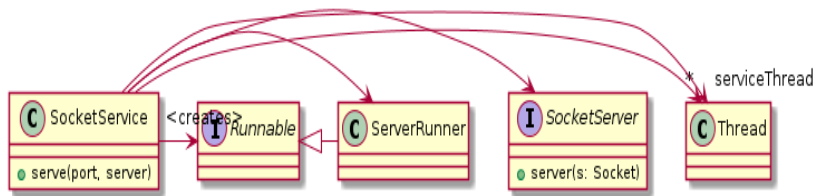
- Objetos
- Relacionamentos
- Atributos (valores) de um instante dado

E o diagrama de classes

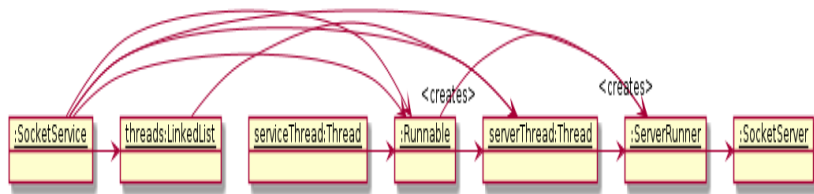
- Muito parecido
- O diagrama de objetos é mais interessante se tiver criação dinâmica na estrutura.



## Serviço de socket: classes



# Serviço de socket: Objetos



- **Snapshot** do sistema
- Usado com sistemas de estrutura dinâmica
- Perto dos diagramas de classes

- 1 Diagramas de sequências
- 2 Diagramas de objetos
- 3 Diagramas de estados**
- 4 Diagramas de atividades
- 5 Exercícios

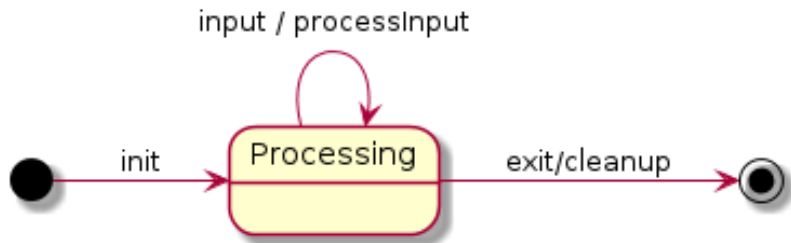
## Objetivo

- Descrição das respostas do sistema aos eventos em função do seu estado
- **Finite State Machine**

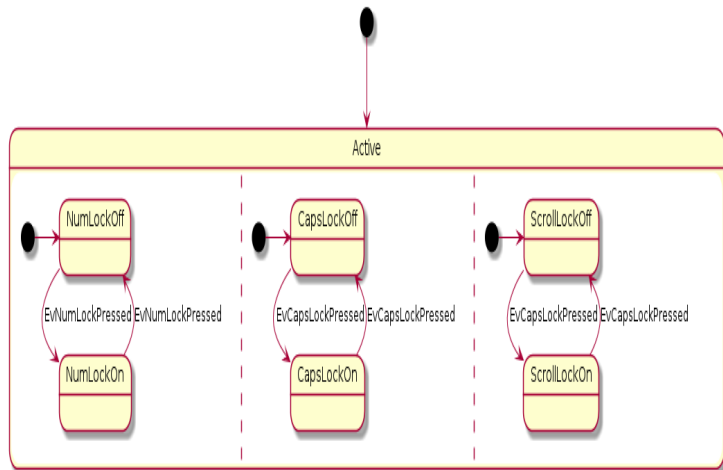
## Elementos

- Estados (retângulos)
- Transições (setas entre estados) com rótulos com o nome do evento.

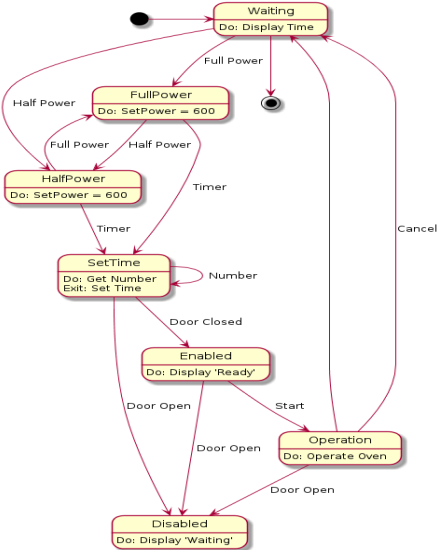
## (Pseudo)-estados iniciais e finais



# Exemplo

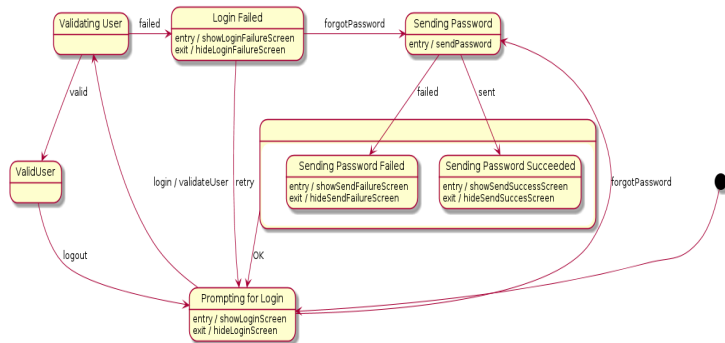


# Micro-ondas



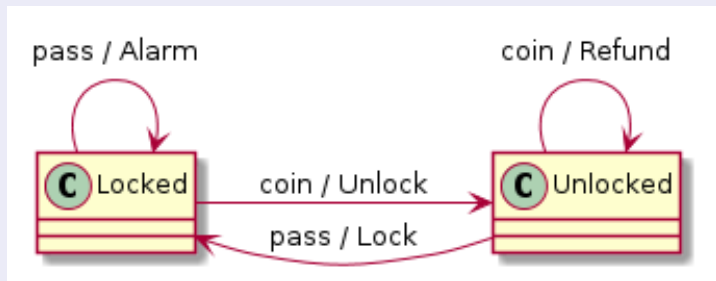


# Máquina de estado de login



# Torniquete de metro

## FSM

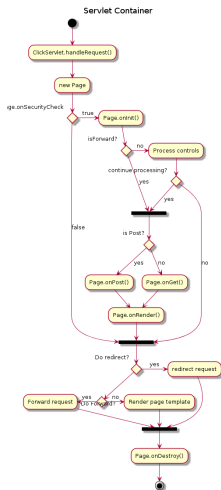


## Texto

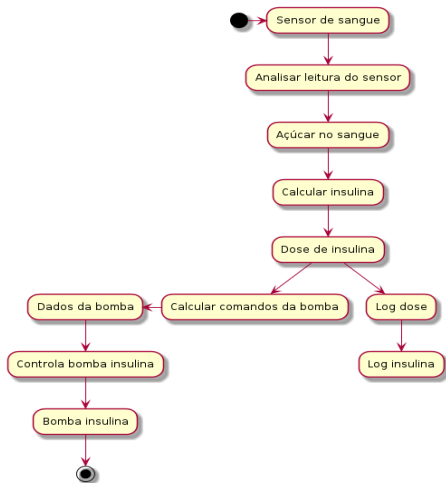
Estado corrente	Evento	Novo Estado	Ação
Locked	coin	Unlocked	Unlock
Locked	pass	Locked	Alarm
Unlocked	coin	Unlocked	Refund
Unlocked	pass	Locked	Lock

- 1 Diagramas de sequências
- 2 Diagramas de objetos
- 3 Diagramas de estados
- 4 Diagramas de atividades**
- 5 Exercícios





# Exemplo



# Modelo de bombas de insulina



- 1 Diagramas de sequências
- 2 Diagramas de objetos
- 3 Diagramas de estados
- 4 Diagramas de atividades
- 5 Exercícios

-  *Site da UML*, <http://www.uml.org>.
-  Grady Booch, James Rumbaugh, and Ivar Jacobson, *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*, Addison-Wesley Professional, 2005.
-  Martin Fowler, *UML Distilled (3rd Ed.): A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, Boston, MA, USA, 2003.
-  R.C. Martin, *UML for Java programmers*, Robert C. Martin series, Prentice Hall PTR, 2003.

## Perguntas ?



<http://dimap.ufrn.br/~richard/dim0436>



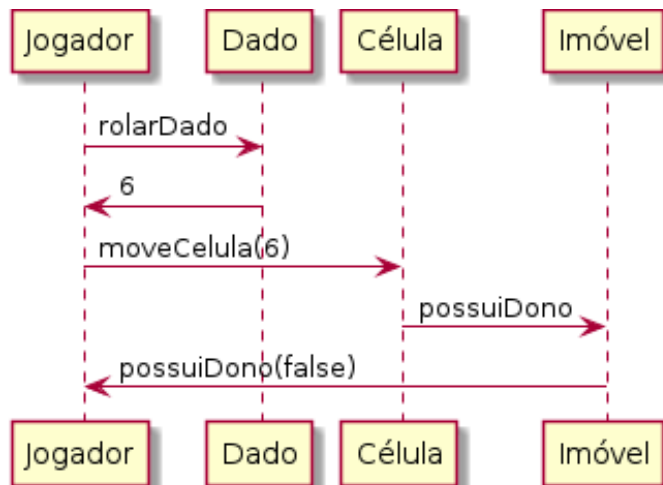
- 1 Diagramas de sequências
- 2 Diagramas de objetos
- 3 Diagramas de estados
- 4 Diagramas de atividades
- 5 Exercícios

# Monopoly

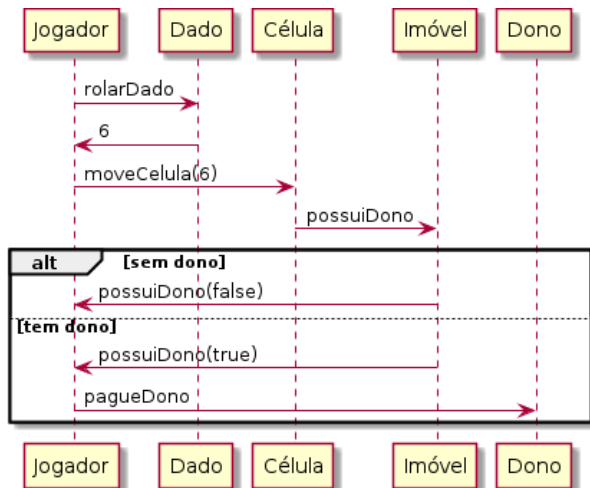
Escreva o diagrama de sequência para o extrato do Monopoly abaixo

- Um jogador rola os dados e recebe um 6. O jogador move 6 células.
- O jogador chega numa célula que é um imóvel sem dono.
- O jogador acaba sua vez.

# Solução



# Solução +



# Norma

```
class Array {  
    ...  
public:  
    // return the index-th component of the array  
    double get(int index);  
    ...  
};  
double norm(const Array& myArray) {  
    double theNorm = 0;  
    for(int index = 0; index < myArray.size() - 1; index++) {  
        theNorm = theNorm + myArray.get(index);  
    }  
    theNorm = sqrt(theNorm);  
    return theNorm;  
}
```

Desenhe:

- o diagrama de sequência da função `norm`
- o fluxograma da função

# Calendário

Desenhe um diagrama de sequência para adicionar um agendamento num calendário. usando o cenário abaixo:

- No início o usuário escolhe a adição de um novo agendamento na UI. A UI percebe a parte do calendário usada e uma janela de agendamento aparece para essa data e tempo.
- O usuário digita as informações necessárias sobre o nome, localização, horários de início e término do agendamento. A interface do usuário irá impedir que o usuário entre um compromisso que tem informações inválidas, tal como um nome vazio ou duração negativa. O calendário cadastra o novo agendamento na lista de compromissos do usuário. Qualquer lembrete selecionado pelo usuário é adicionado à lista de lembretes.
- Se o usuário já tem um compromisso nesse momento, uma mensagem de aviso é mostrada e o sistema pede ao usuário se ele quiser escolher um horário disponível ou substituir a designação anterior. Se o usuário digitar uma consulta com o mesmo nome e duração como uma reunião de grupo existente o calendário pergunta ao usuário se ele quiser juntar-se ao grupo de reunião. Caso sim, o usuário é adicionado à lista de participantes do grupo de reunião.

# Threads

Qual é o maior número de threads concorrentes possível no diagrama abaixo ?

