



# Conteúdo

1. Introdução
2. Levantamento de Requisitos
3. Análise Orientada a Objetos
4. Projeto Orientado a Objetos
5. UML
6. Métodos Ágeis



# Projeto Orientado a Objetos



# Projeto Orientado a Objetos

Durante o projeto de objeto, é desenvolvida uma solução lógica baseada no paradigma orientado a objetos.

Uma aplicação orientada a objetos é construída a partir de objetos que interagem através do envio de mensagens que requerem informações ou ação.

Para um mesmo problema existem diferentes soluções de projeto e os projetistas não produzem soluções idênticas.

Geralmente são definidos:

- **Diagrama de Classes de Projeto:** define as classes de software e interfaces que serão implementadas.
- **Diagramas de Interação:** ilustram como os objetos colaboram para satisfazer os requisitos.

# Diagrama de Classes de Projeto

As classes de projeto mostram definições das classes de software e não dos conceitos do mundo real.

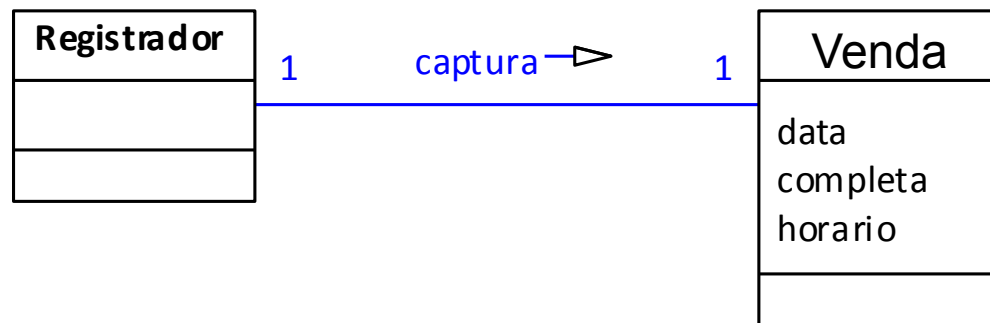
## Classe do Modelo de Domínio x Classe do Modelo de Projeto

➔ Classe do Modelo de Domínio: abstração de um conceito do mundo real tratado pelo sistema.

➔ Classe do Modelo de Projeto: componente de software.

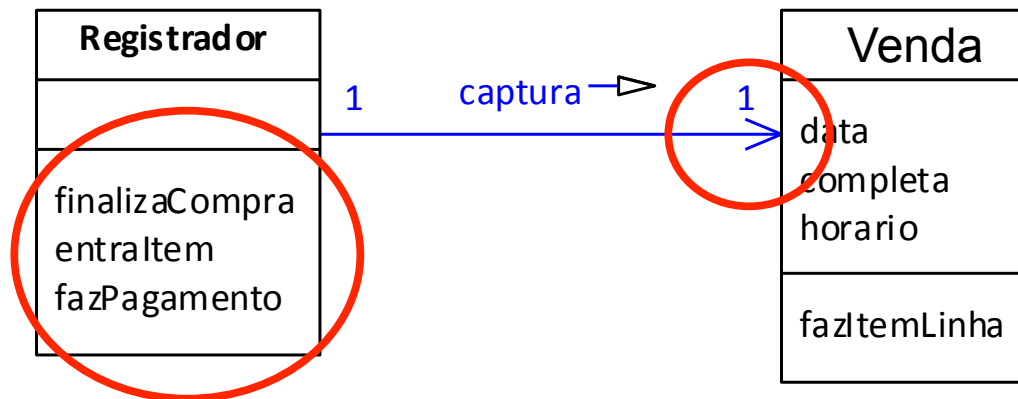
# Diagrama de Classes de Projeto

Classes do Modelo de Domínio x Classes do Modelo de Projeto



**Modelo do Domínio**

**Modelo do Projeto**



# Diagrama de Classes de Projeto

Informações incluídas nos diagramas de classes de projeto:


- Classe, Atributo, Associação, Agregação e Herança
  - Operação
  - Interface
  - Navegabilidade
  - Dependência
- aparecem no modelo conceitual

# Diagrama de Classes de Projeto


Definição do diagrama de classes de projeto:

- O diagrama é definido a partir dos diagramas de interação.
- Um diagrama preliminar (classes, atributos e relacionamentos) pode ser definido no início do projeto, a partir do diagrama de classes conceituais.

O diagrama é refinado em paralelo com os diagramas de interação.



# Diagramas de Interação (Sequência e Comunicação) da UML





# Diagrama de Interação

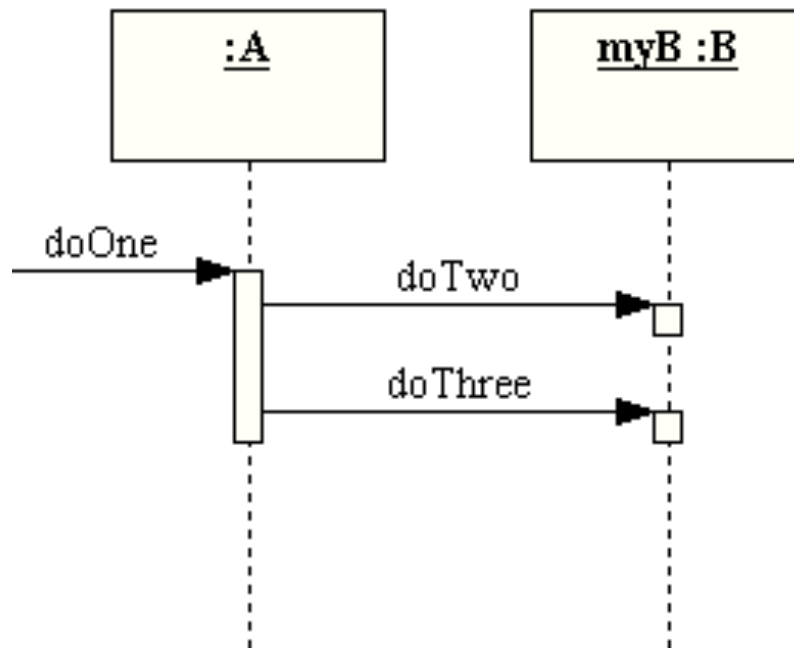
Diagrama de Interação: expressam interações através de mensagens.

➔ Diag. de Interação = Diag. de Sequência + Diag. de Comunicação  
└─ mais usado

Obs: Diagrama de Comunicação da UML 2.0 = Diagrama de Colaboração da UML 1.0

# Diagrama de Sequência

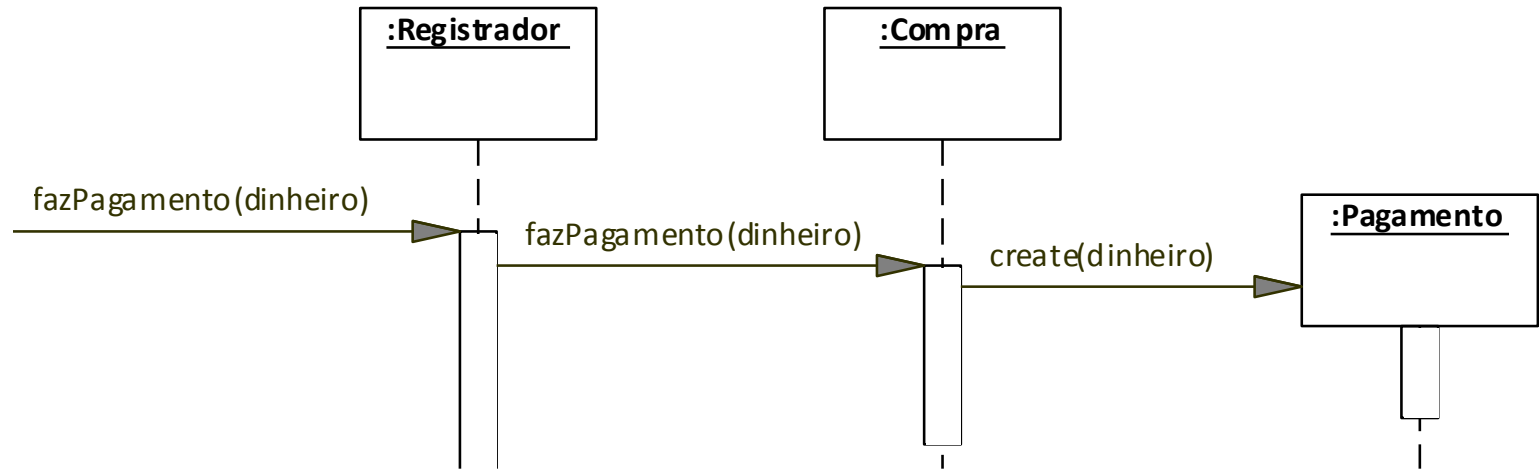
**Diagramas de Sequência:** Ilustram as interações entre os objetos em uma espécie de cerca, na qual cada novo objeto é adicionado a direita.



```
public class A
{
    private B myB = new B( );

    public void doOne( )
    {
        myB.doTwo( );
        myB.doThree( );
    }
    // ...
}
```

# Diagrama de Sequência



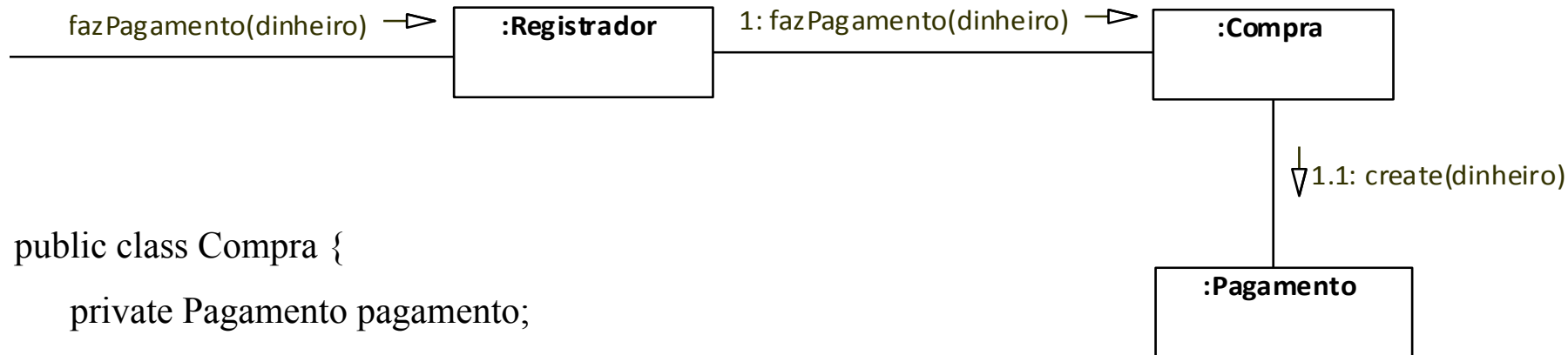
## Código relacionado com o Diagrama de Sequência

```
public class Compra {
    private Pagamento pagamento;

    public void fazPagamento (Money dinheiro){
        pagamento = new Pagamento (dinheiro);
        //...
    }
}
```

# Diagrama de Comunicação

**Diagramas de Comunicação:** Ilustram as interações entre os objetos em um grafo, no qual os objetos são colocados em qualquer lugar do diagrama.

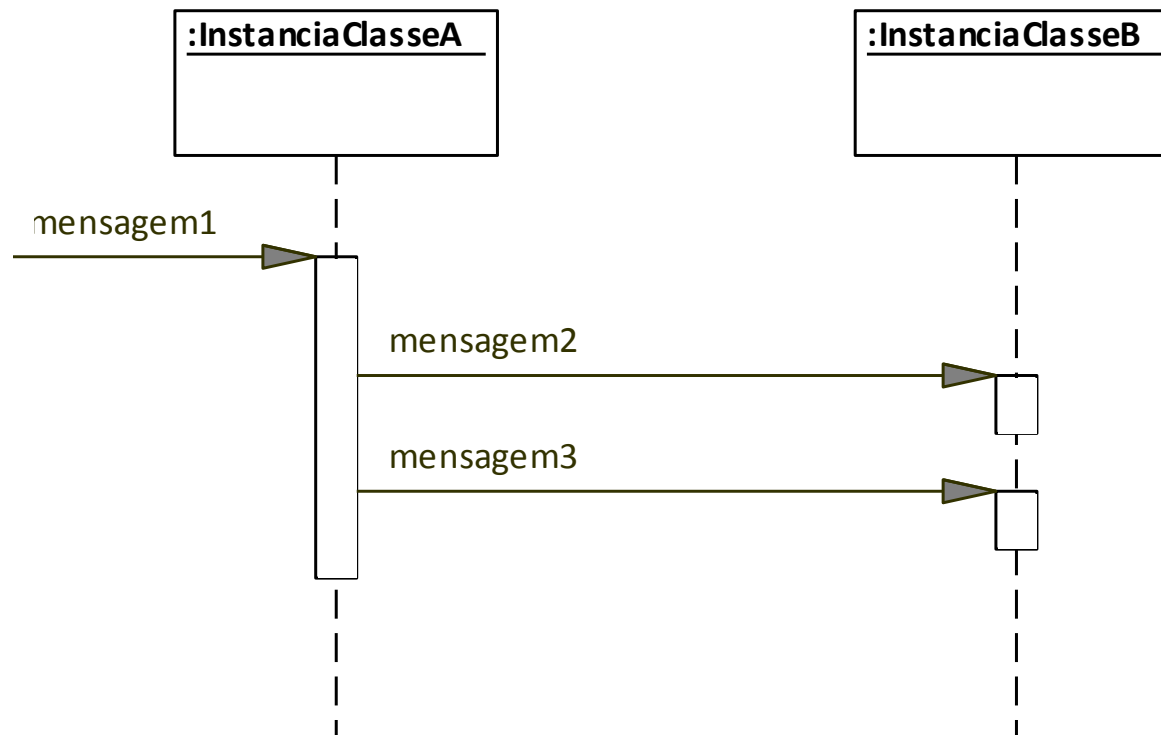


```
public class Compra {
    private Pagamento pagamento;
    public void fazPagamento (Money dinheiro){
        pagamento = new Pagamento (dinheiro);
        //...
    }
}
```

# Diagrama de Sequência

Vantagens (em relação aos diagramas de comunicação):

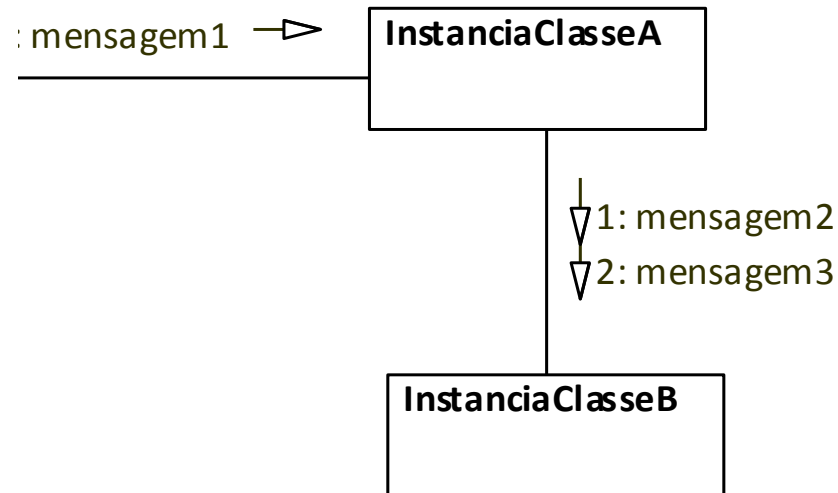
- Mostra claramente a ordem das mensagens das mensagens;
- Conjunto maior de opções de notação.



# Diagrama de Comunicação

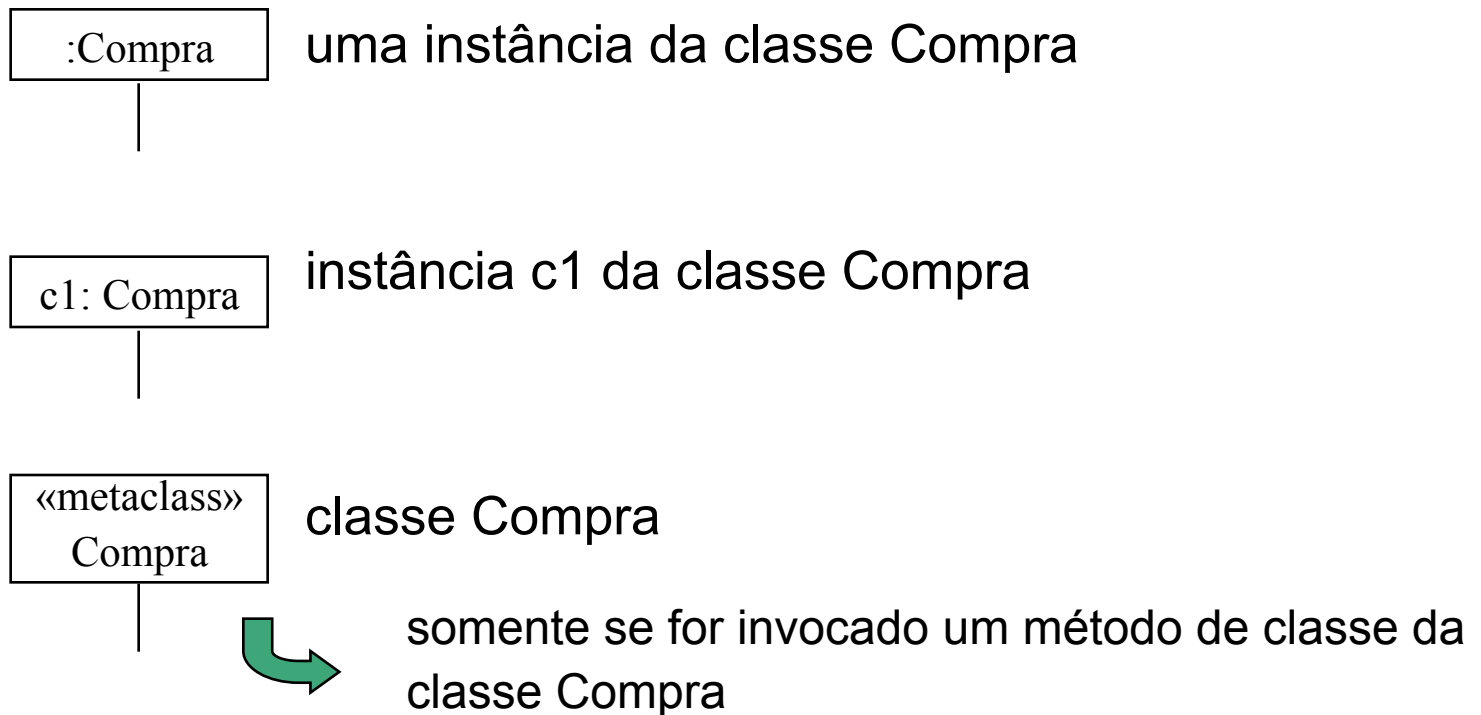
Vantagens: (em relação aos diagramas de sequência)

- Facilita a inclusão de novos objetos sem ocupar muito espaço.



# Notação dos Diagrama de Interação

→ Os diagramas de interação representam a troca de mensagens, geralmente, entre instâncias das classes.



# Notação dos Diagrama de Interação

compras:  
ListaArray<Compra>

uma instância da classe ListaArray que contém objetos do tipo Compra

compras[i]: Compra

uma instância da classe Compra selecionada da coleção compras

x : Lista

Lista pode ser uma interface



# Exercício

Defina um diagrama de sequência para o método a seguir:

```
public void submeteArtigo(String nome, String resumo, String[] nomesAutores,
{
    Artigo artigo = new Artigo(nome);
    artigo.setResumo(resumo);
    artigo.setIdentificador(identArtigo);
    identArtigo++;
    for (int i=0; i<nomesAutores.length; i++){
        String nomeAutor = (String)nomesAutores[i];
        Autor autor = new Autor (nomeAutor);
        int indice = this.autores.indexOf(autor);
        if (indice == -1)
            this.autores.add(autor);
        else
            autor = this.autores.get(indice);
        artigo.associaAutor(autor);
    }
    this.artigos.add(artigo);
}
```

# Notação dos Diagrama de Interação

→ Sintaxe para a expressão das mensagens:

retorno = mensagem (parâmetro : tipoParâmetro) : tipoRetorno

## Exemplos:

- inicializa (tamanho)
- espec = getEspecProduto (id)
- espec = getEspecProduto (id: IDitem)
- espec = getEspecProduto (id: IDitem) : EspecProduto

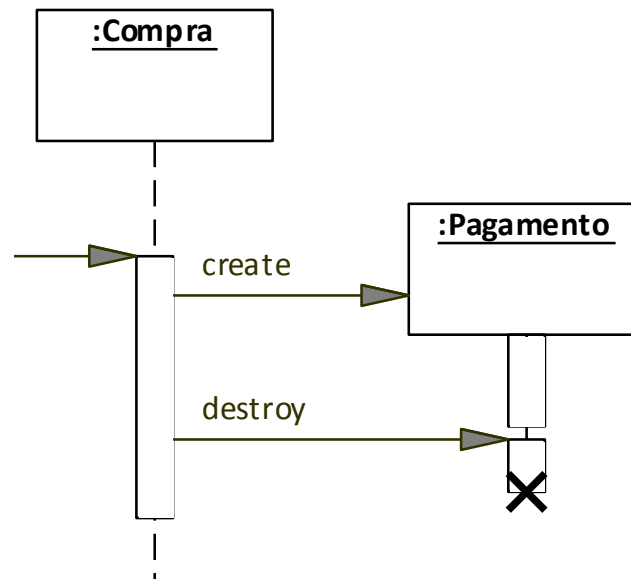


# Notação dos Diagramas de Sequência

- 1. Linha de Vida de um Objeto
- 2. Mensagem
- 3. Retorno de uma mensagem
- 4. Criação de Instâncias
- 5. Looping
- 6. Mensagem Condicional

# 1. Linha de Vida de um Objeto

A linha vertical (tracejada ou não) representa o tempo de vida de um objeto.

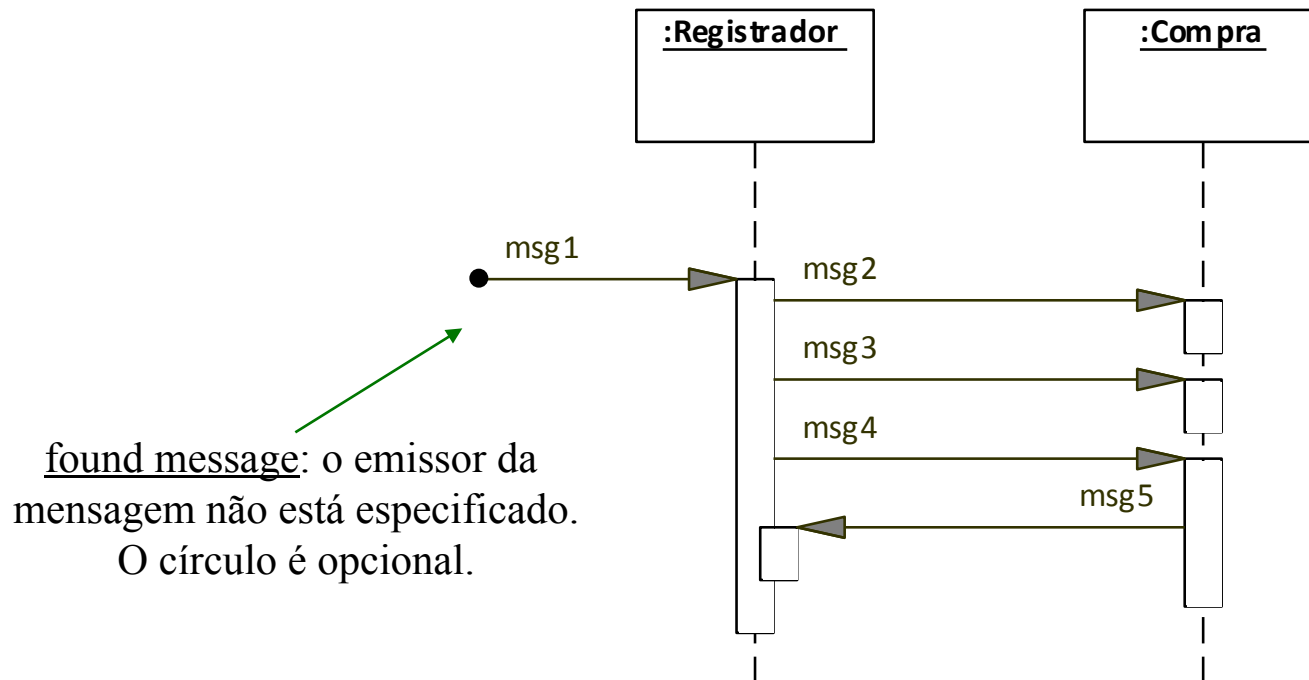


➡ A destruição de um objeto é representada pela mensagem **destroy**, um grande X e uma linha de vida menor.

## 2. Mensagem

Cada mensagem entre dois objetos é representada pela sua expressão em uma linha direcionada (com seta preenchida) entre dois objetos.

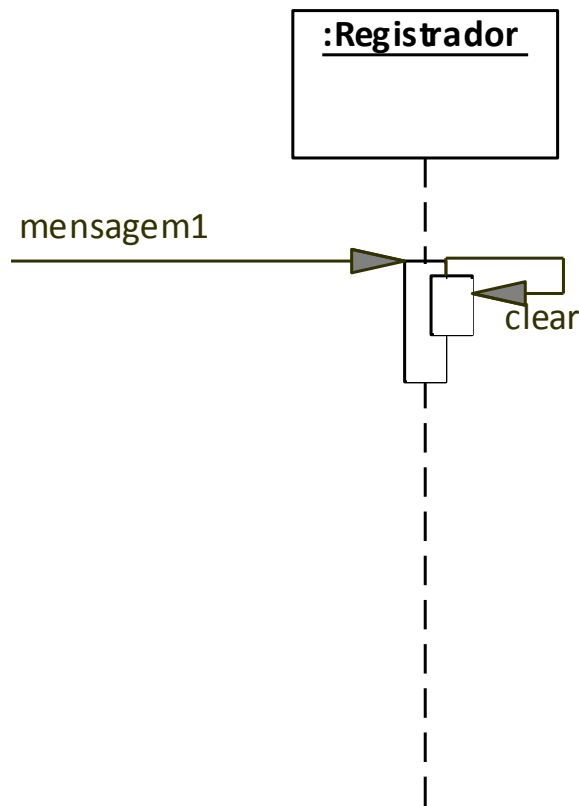
A ordem sequencial das mensagens é organizada de cima para baixo.



➡ O retângulo de especificação de execução mostra o foco de controle. É opcional.

## 2. Mensagem

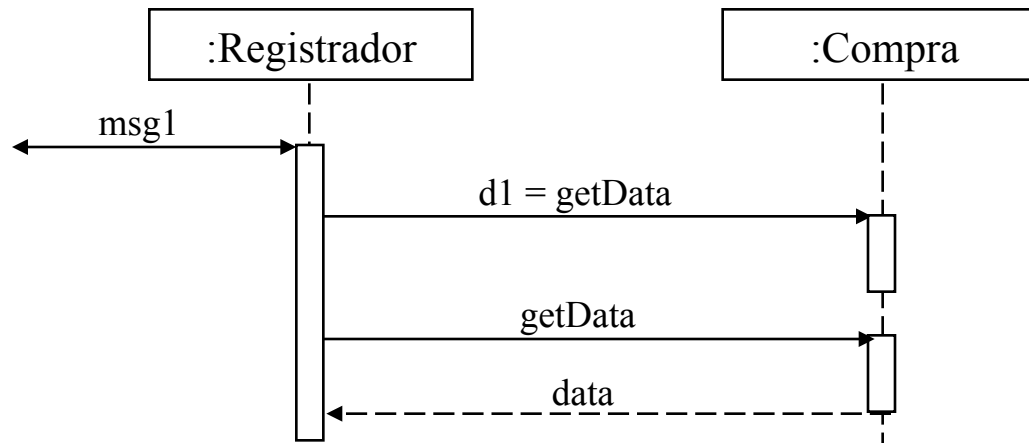
- Uma mensagem pode ser enviada de um objeto para ele mesmo.



### 3. Retorno de uma Mensagem

O retorno de uma mensagem pode ser mostrado como:

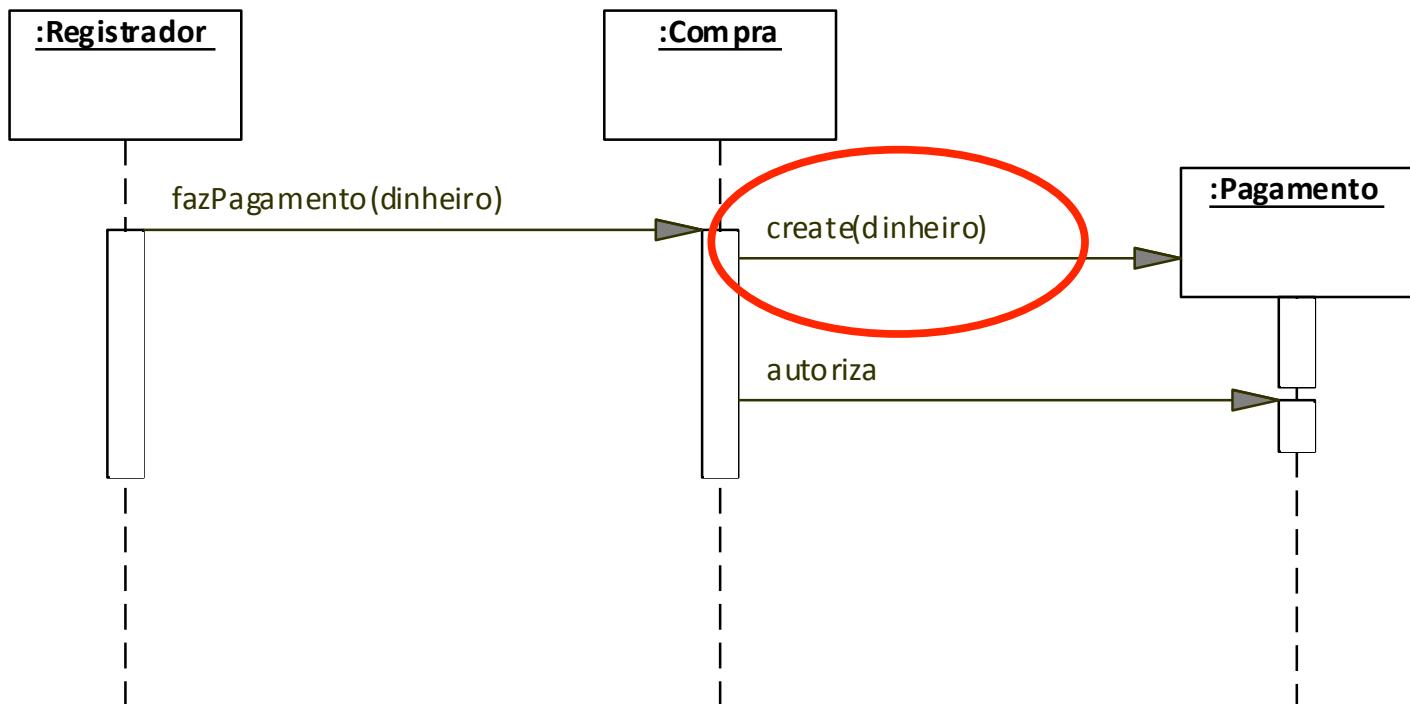
- uma linha tracejada no final do retângulo de especificação de execução
- ou pela especificação do retorno na própria mensagem.



## 4. Criação de Instâncias

Geralmente o nome **create** é usado para representar uma mensagem que cria uma instância.

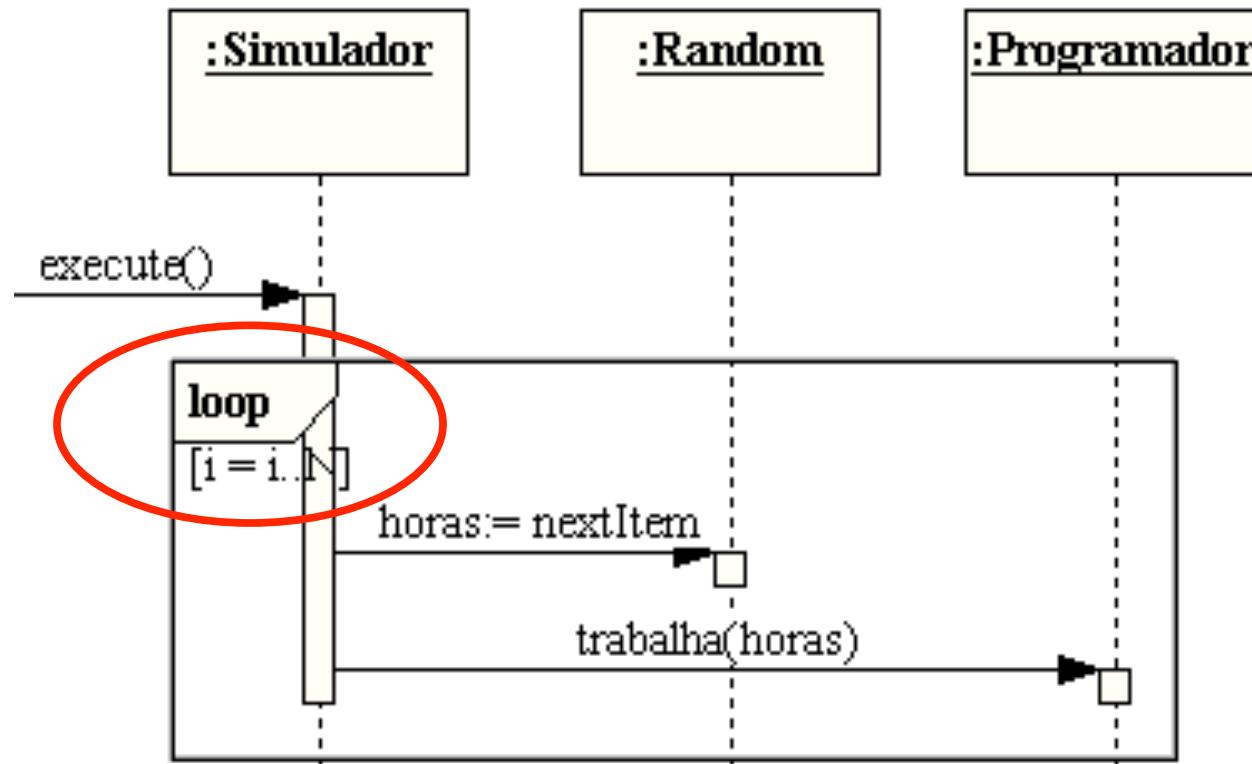
➡ O objeto criado é colocado na altura da sua criação.





## 5. Looping

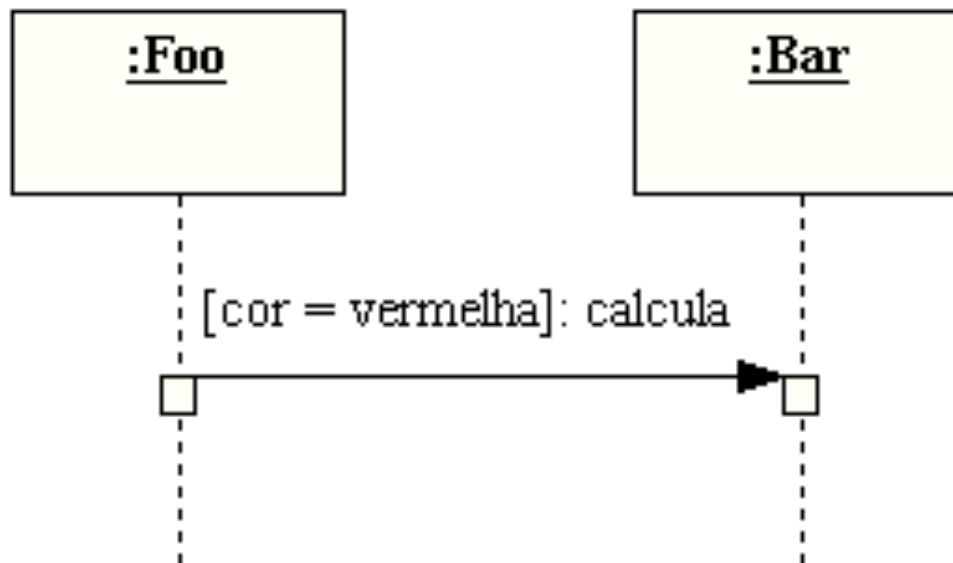
O looping é indicado com um retângulo onde aparece o label loop e uma cláusula de looping.



## 6. Mensagem Condicional

A mensagem condicional é indicada com uma cláusula junto a mensagem.

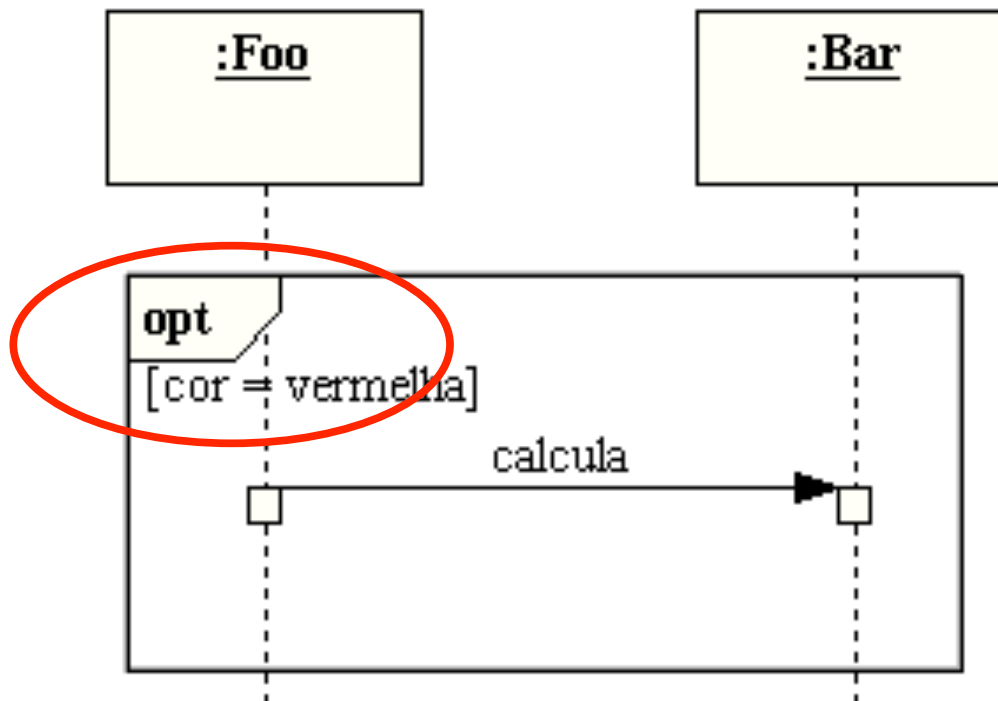
A mensagem é enviada somente se a cláusula é verdadeira.



## 6. Mensagem Condicional

Um condição sob um conjunto de mensagens é indicada com uma cláusula de looping.

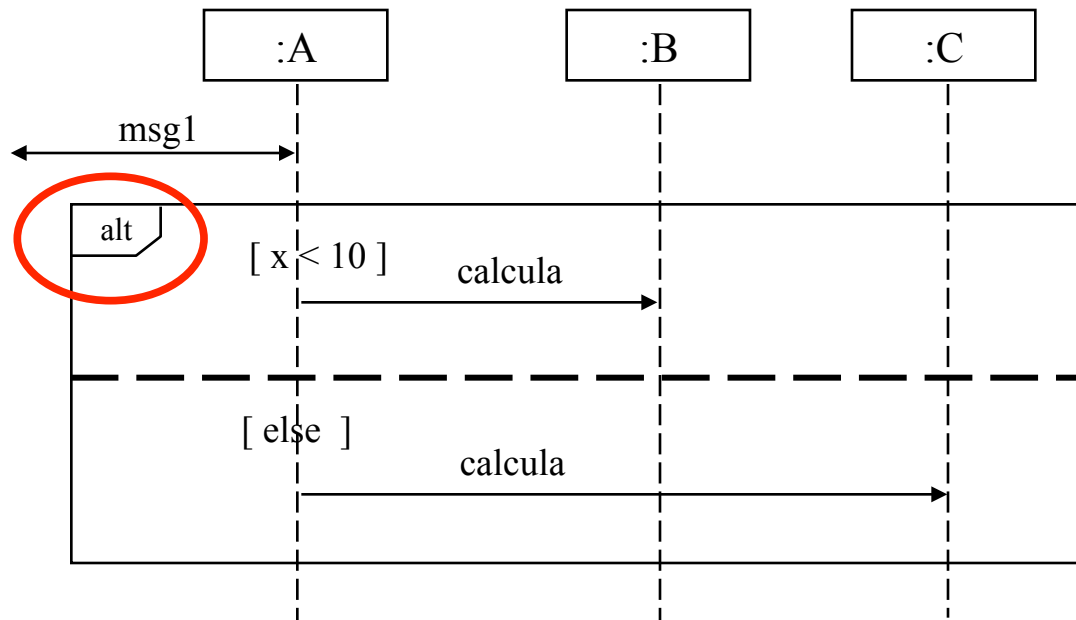
As mensagens são enviadas somente se a cláusula é verdadeira.



## 6. Mensagem Condicional

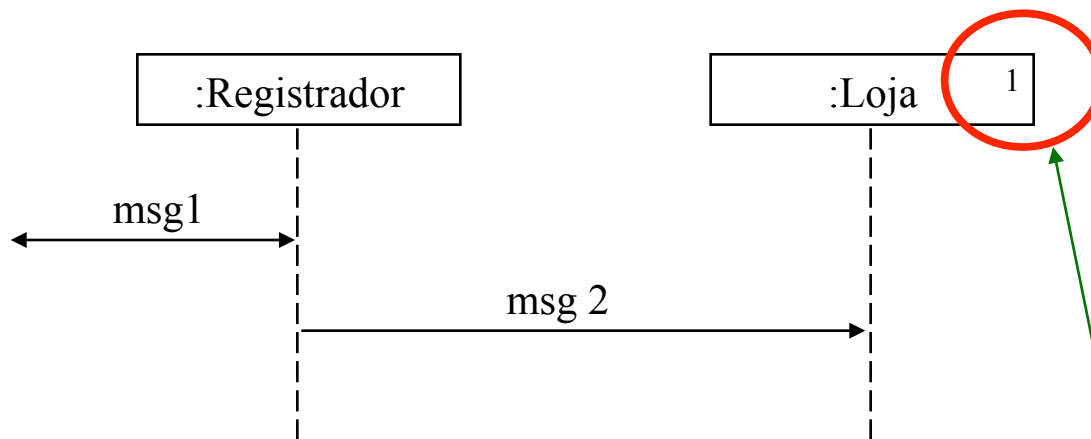
- Mensagens podem ser mutuamente exclusivas.

Mensagens condicionais mutuamente exclusivas são indicadas com um retângulo onde aparece o label alt e uma linha pontilhada separando as mensagens.



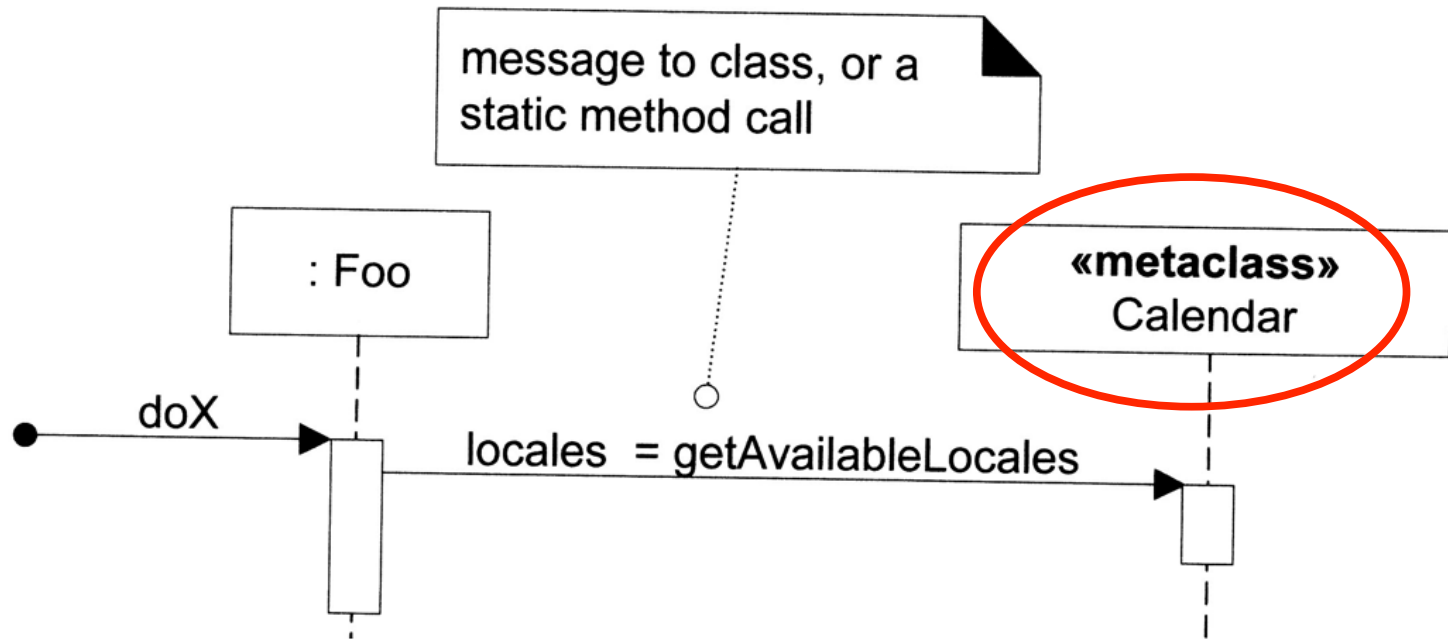
# Mensagem para um Singleton

→ Singleton: somente 1 instância de uma classe pode ser instanciada.

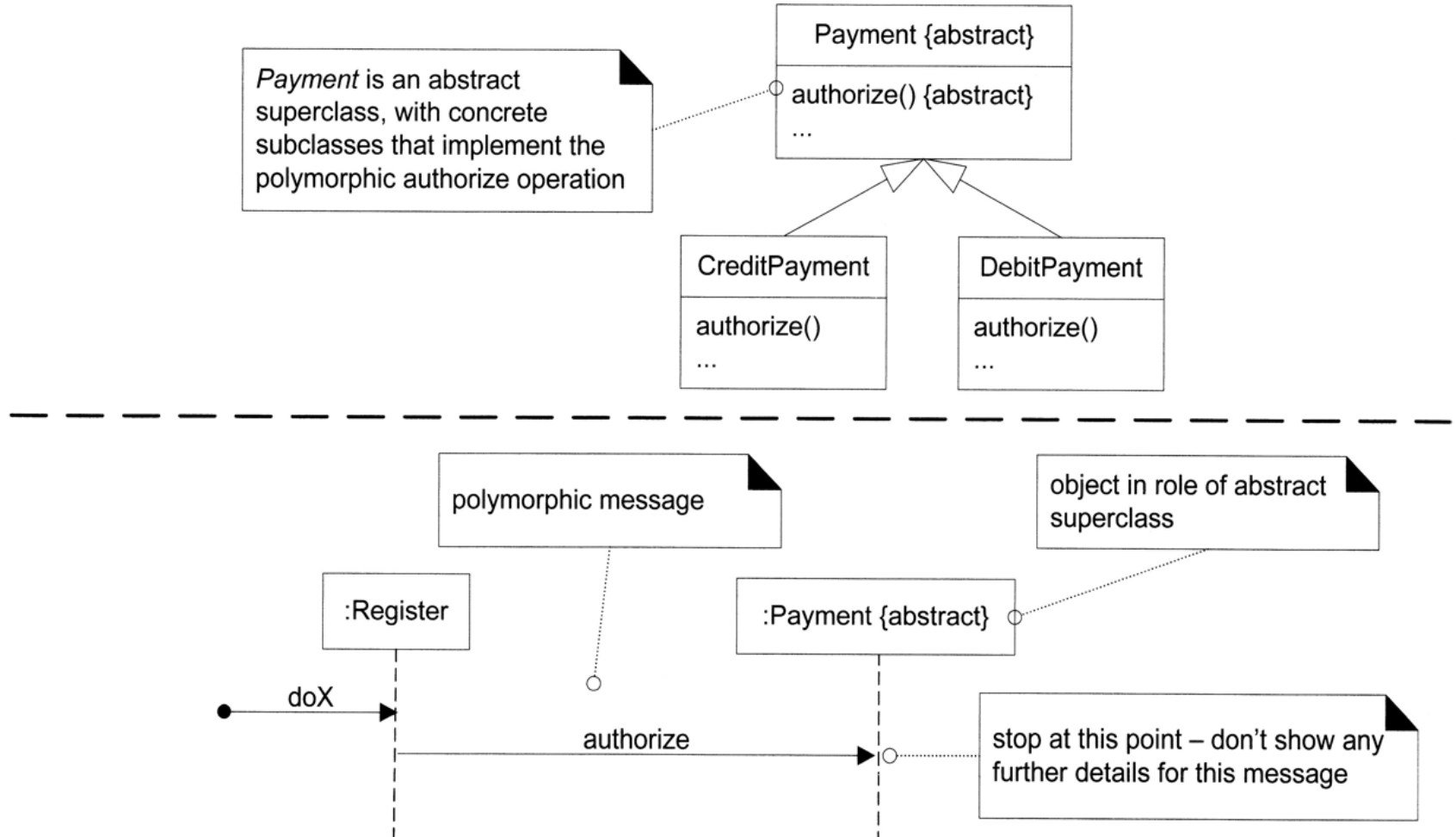


o objeto é marcado com o "1" na parte superior direita

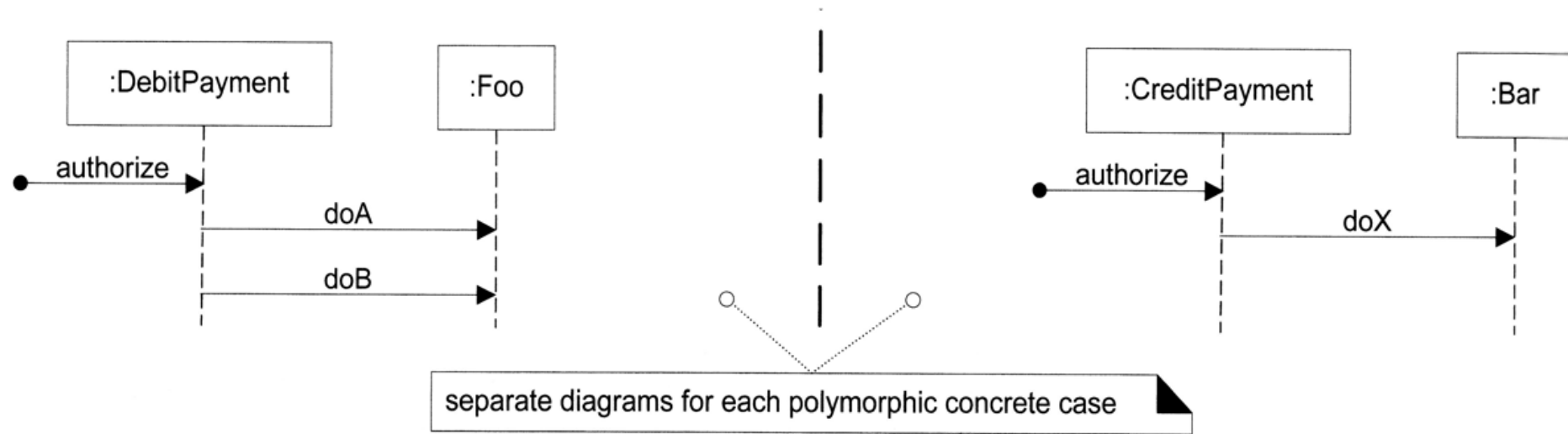
# Msgs que invocam métodos de classe



# Mensagens Polimórficas

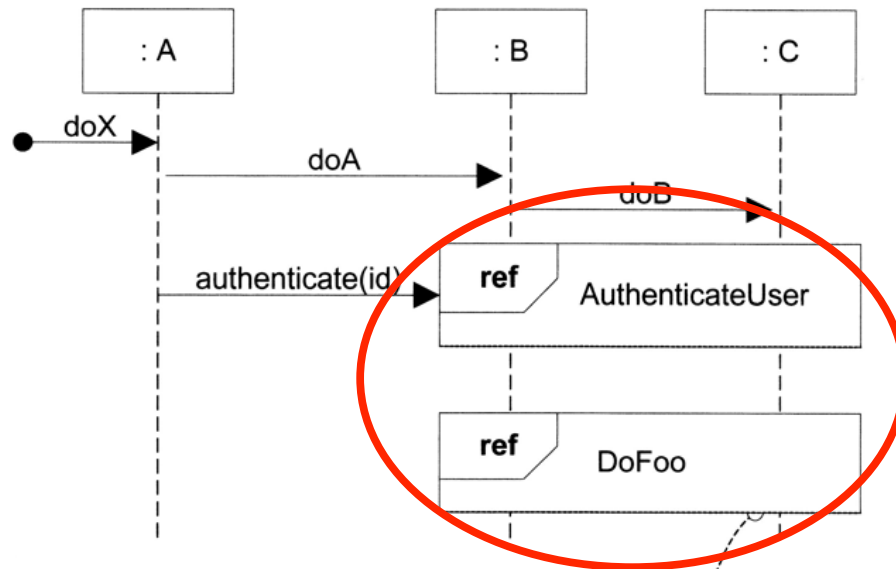


# Mensagens Polimórficas





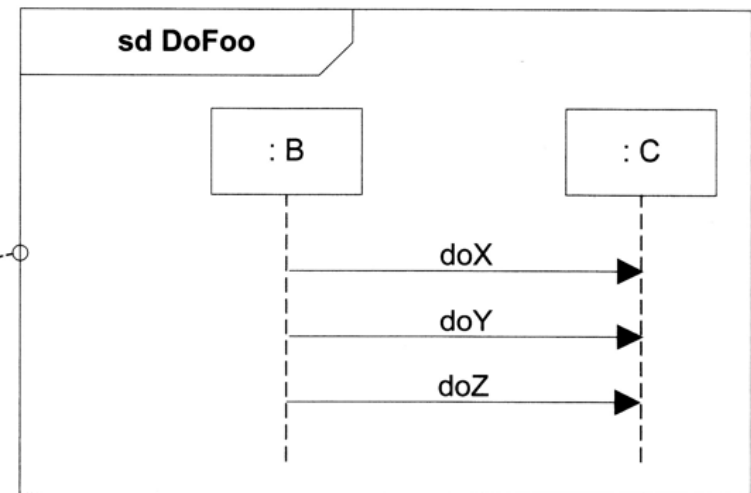
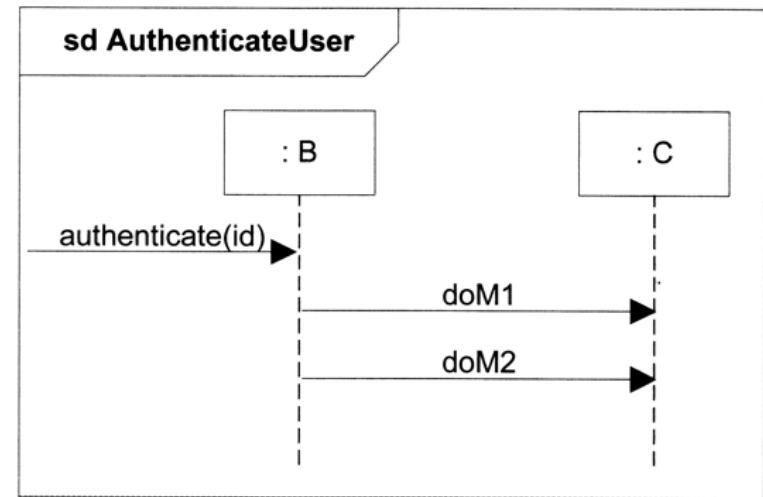
# Referência entre Diagramas



interaction occurrence

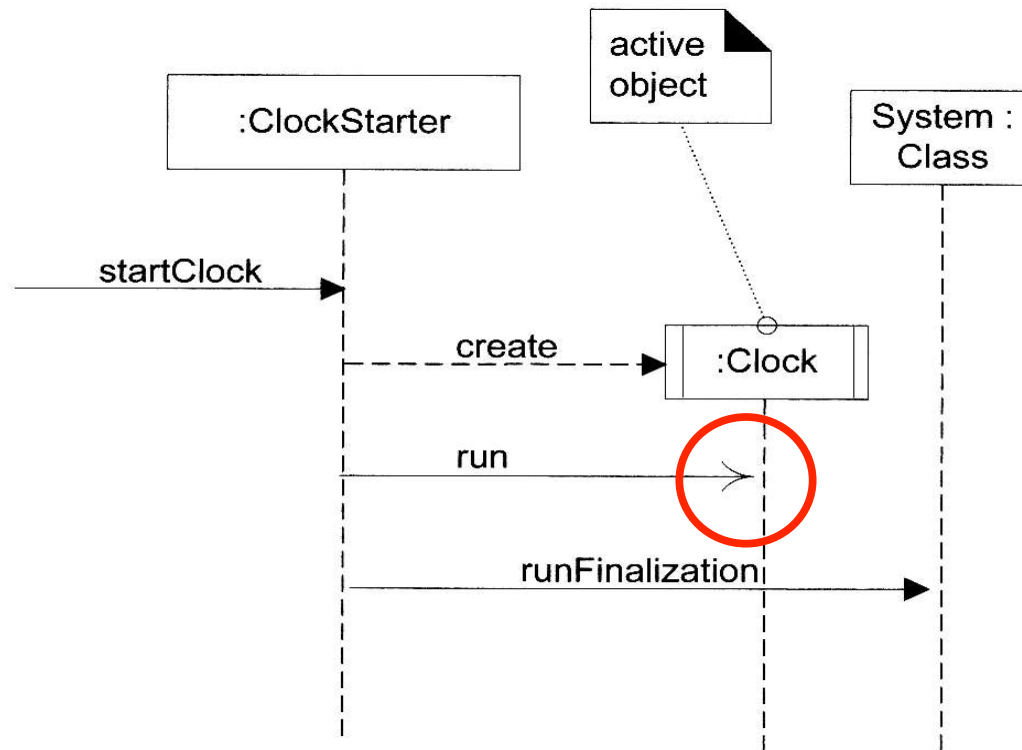
note it covers a set of lifelines


note that the sd frame it relates to has the same lifelines: B and C



## 7. Chamadas Assíncronas e Síncronas

Uma chamada de uma mensagem assíncrona não precisa esperar por uma resposta.



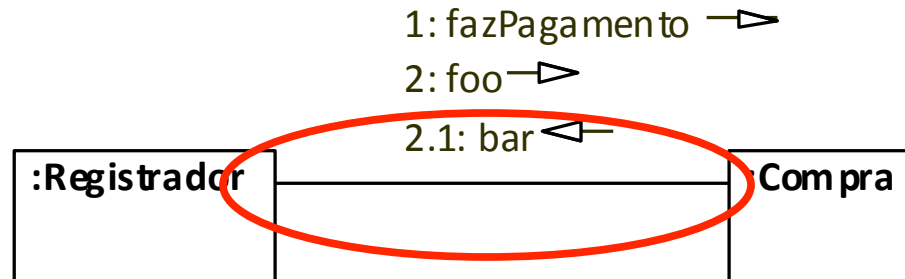


# Notação dos Diagramas de Comunicação

- 1. Ligação (Link)
- 2. Mensagem
- 3. Criação de Instâncias
- 4. Sequência dos Números das Mensagens
- 5. Mensagem Condicional
- 6. Looping

# 1. Ligação (Link)

Uma ligação é um caminho de conexão entre dois objetos. Ela indica que existe alguma forma de navegação entre os objetos.

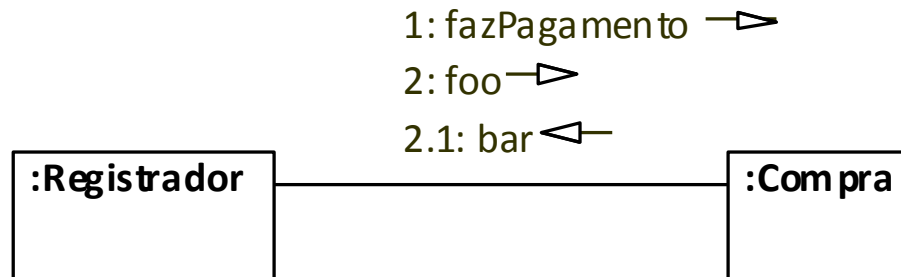


➡ Várias mensagens podem fluir em uma mesma ligação.

## 2. Mensagem

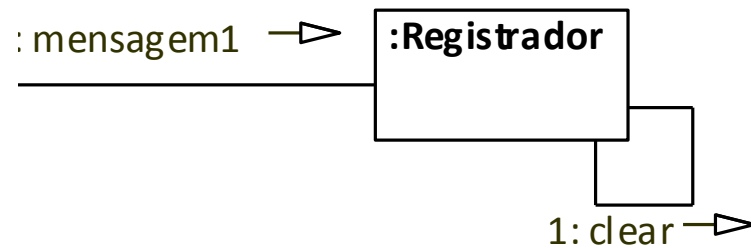
Cada mensagem entre dois objetos é representada pela sua expressão e uma pequena seta indicando a direção da mensagem.

Um número sequencial é adicionado para mostrar a ordem sequencial das mensagens na tarefa corrente.



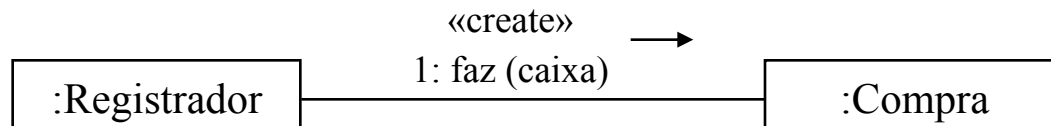
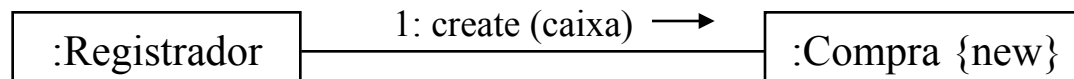
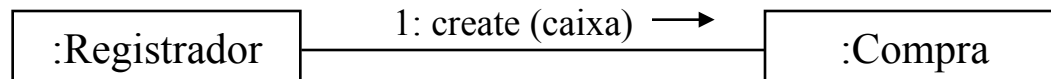
## 2. Mensagem

- Uma mensagem pode ser enviada de um objeto para ele mesmo.



### 3. Criação de Instâncias

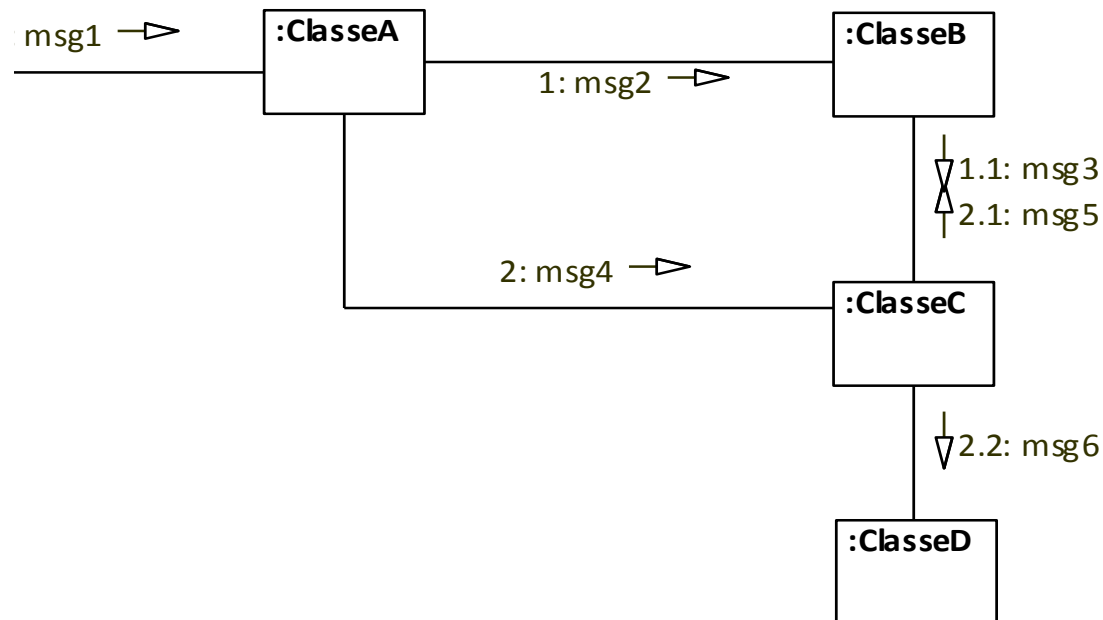
Geralmente o nome **create** é usado para representar uma mensagem que cria uma instância.



## 4. Sequência dos Números das Mensagens

A ordem das mensagens é mostrada através de números sequenciais:

- A primeira mensagem, geralmente, não é numerada.
- A ordem e aninhamento das mensagens subsequentes é mostrada com um esquema de numeração onde um número é anexado às mensagens aninhadas.

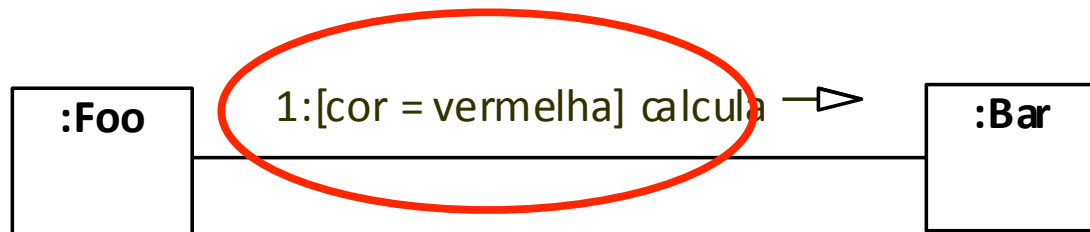




## 5. Mensagem Condicional

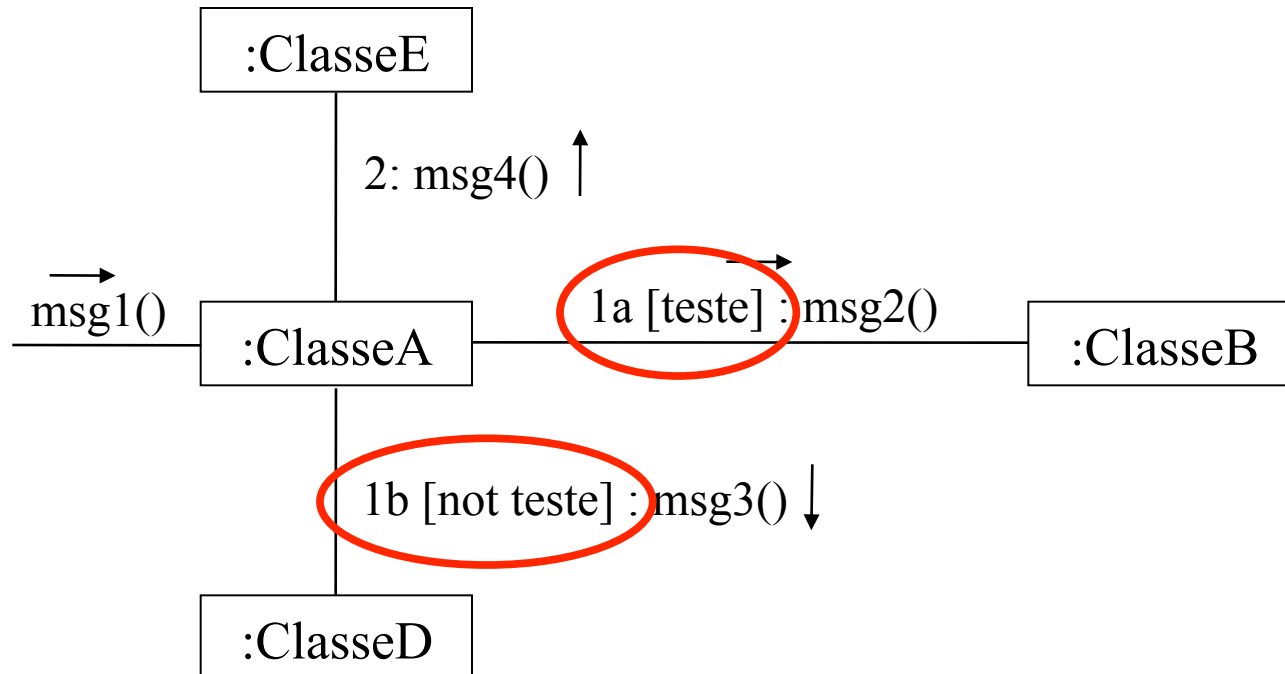
Uma mensagem condicional é mostrada pelo número seqüencial seguido de uma cláusula condicional entre colchetes.

A mensagem é enviada somente se a cláusula é verdadeira.



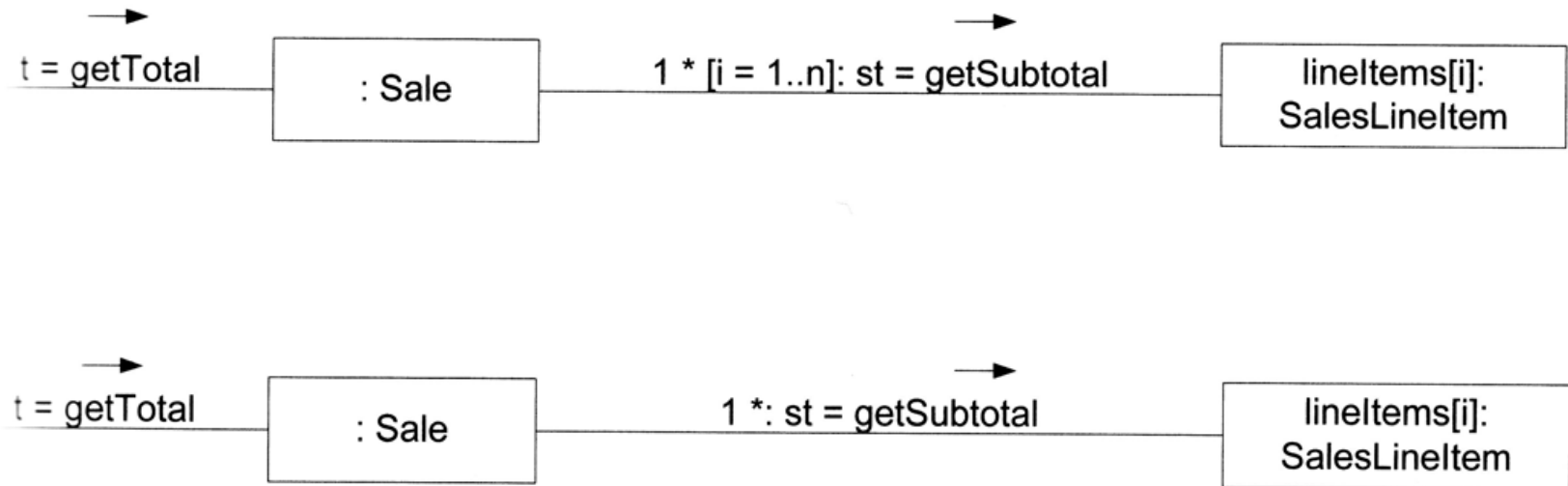
## 5. Mensagem Condicional

- Mensagens podem ser mutuamente exclusivas.

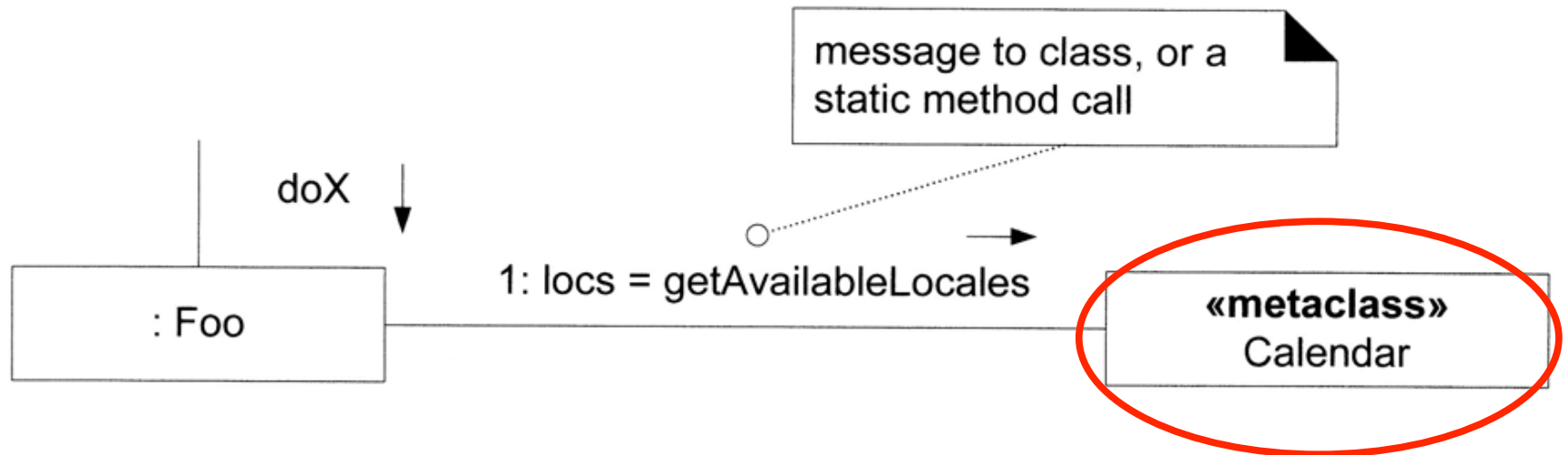




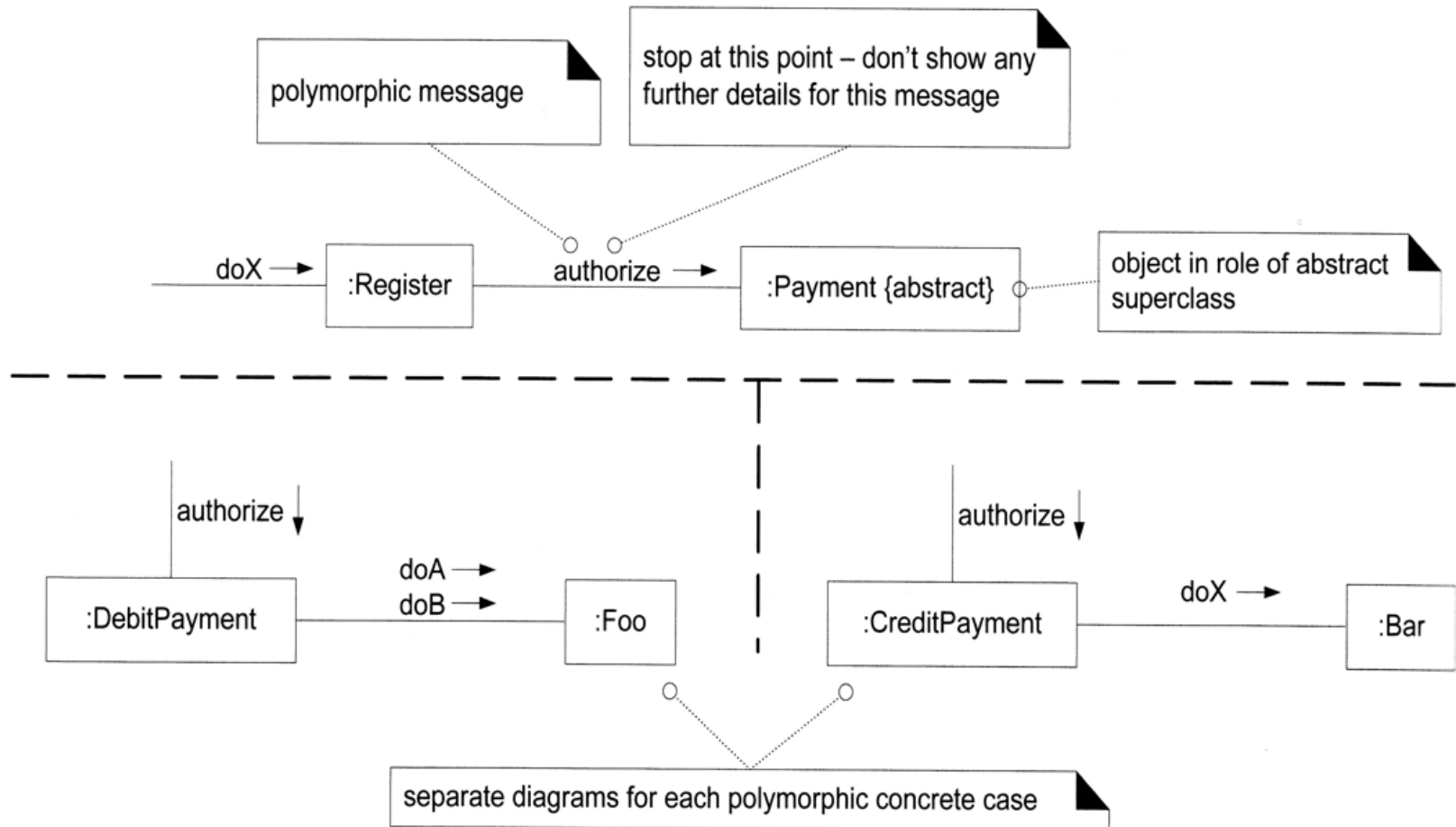
## 6. Looping sobre uma Coleção de Objetos



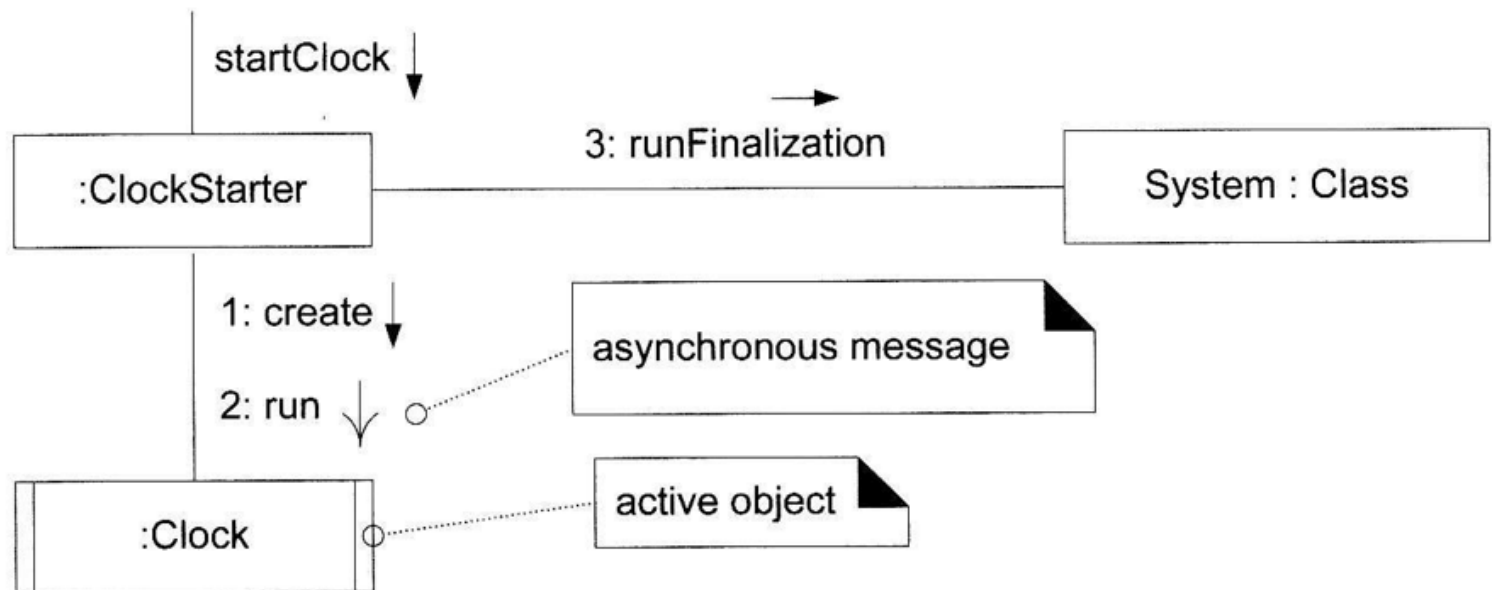
# Msgs que invocam métodos de classe



# Mensagens Polimórficas

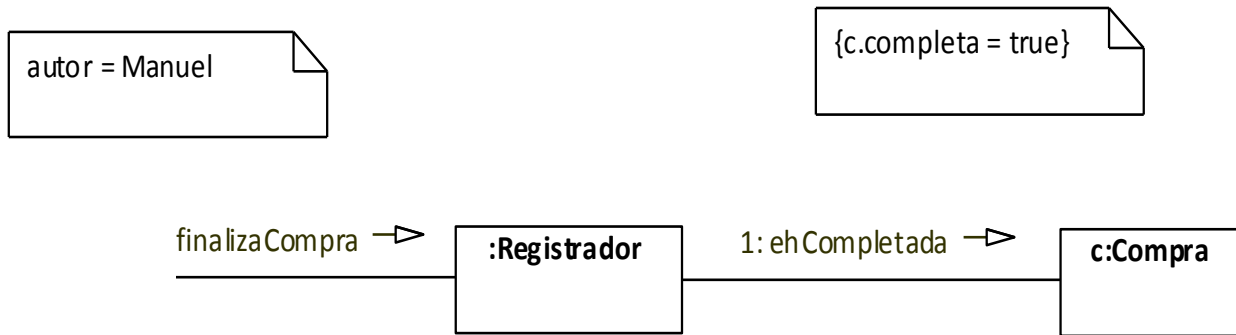


## 7. Chamadas Assíncronas e Síncronas



# Nota

**Símbolo de Nota:** é um retângulo com a “orelha virada” que pode ser utilizado em qualquer diagrama da UML. Geralmente é utilizado no diagrama de classes.



Pode conter:

- ➔ Nota ou Comentário: não tem impacto semântico. Exemplo: autor de um diagrama;
- ➔ Restrição: é uma informação semanticamente significativa anexada a um elemento do modelo. É apresentada entre chaves { };
- ➔ Método: implementação de uma operação.



# Exercícios

Definam diagramas de sequência para os seguintes métodos:

```
public class ListBoxChoice extends MultiChoice {  
    JawsList list;  
    Vector choices;  
  
    public JPanel getUI() {  
        JPanel p = new JPanel();  
        list = new JawsList(choices.size());  
        list.setMultipleMode(true);  
        p.add(list);  
        for (int i=0; i< choices.size(); i++)  
            list.add((String)choices.elementAt(i));  
        return p;  
    }  
}
```

# Exercícios

```
public class TimeSwimData extends SwimData {  
    protected Vector swimmers;
```

```
    public TimeSwimData(String filename) {  
        String s = "";  
        swimmers = new Vector();  
        InputFile f = new InputFile(filename);  
        s = f.readLine();  
        while (s != null) {  
            swimmers.addElement(new Swimmer(s));  
            s = f.readLine();  
        }  
        f.close();  
    }
```

# Exercícios

```
public class ContadorPalavras {  
  
    public TabelaHash contaPalavras (String frase){  
        TabelaHash tabela = new TabelaHash(13);  
        StringTokenizer st = new StringTokenizer (frase, " ", false);  
        while (st.hasMoreTokens()){  
            String token = st.nextToken();  
            Integer ocorrencia = (Integer) tabela.retorna(token);  
            if (ocorrencia != null){  
                int numOcorrencia = ocorrencia.intValue();  
                tabela.remove(token);  
                tabela.insere(token, new Integer(numOcorrencia+1));    }  
            else{  
                tabela.insere(token, new Integer(1));    }  
        }  
        return tabela;    }  
}
```