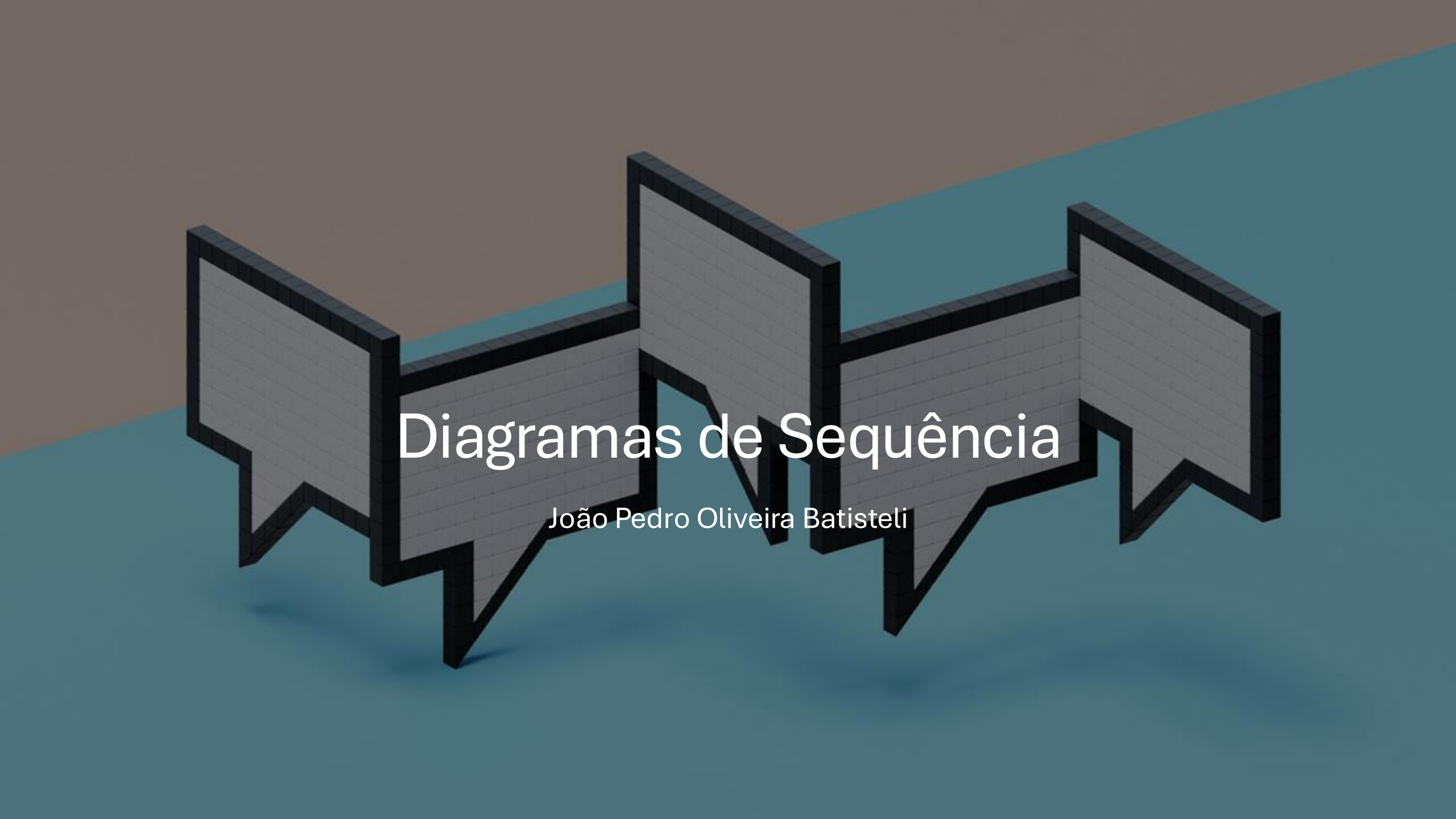


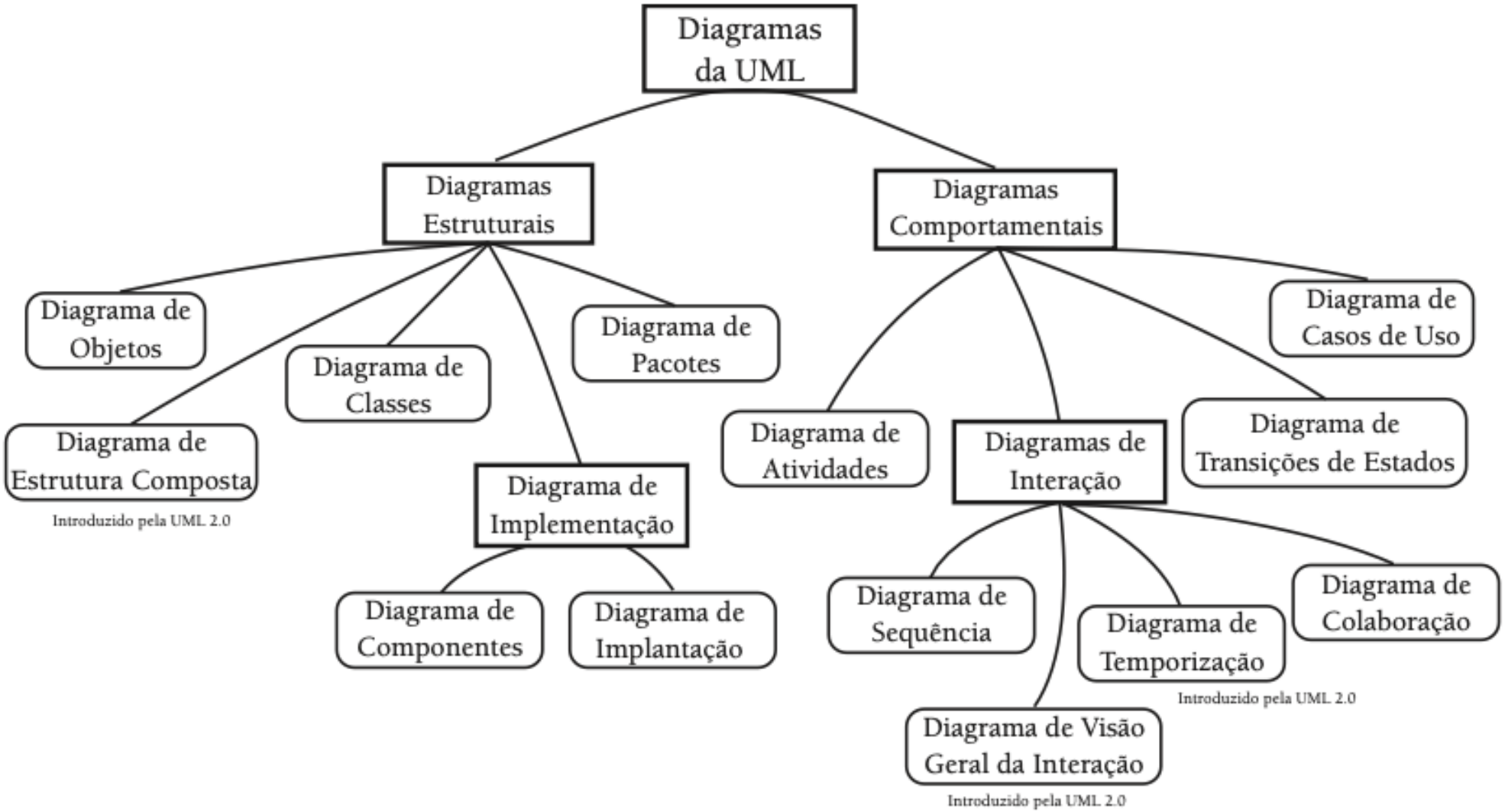
Diagramas de Sequência

João Pedro Oliveira Batisteli

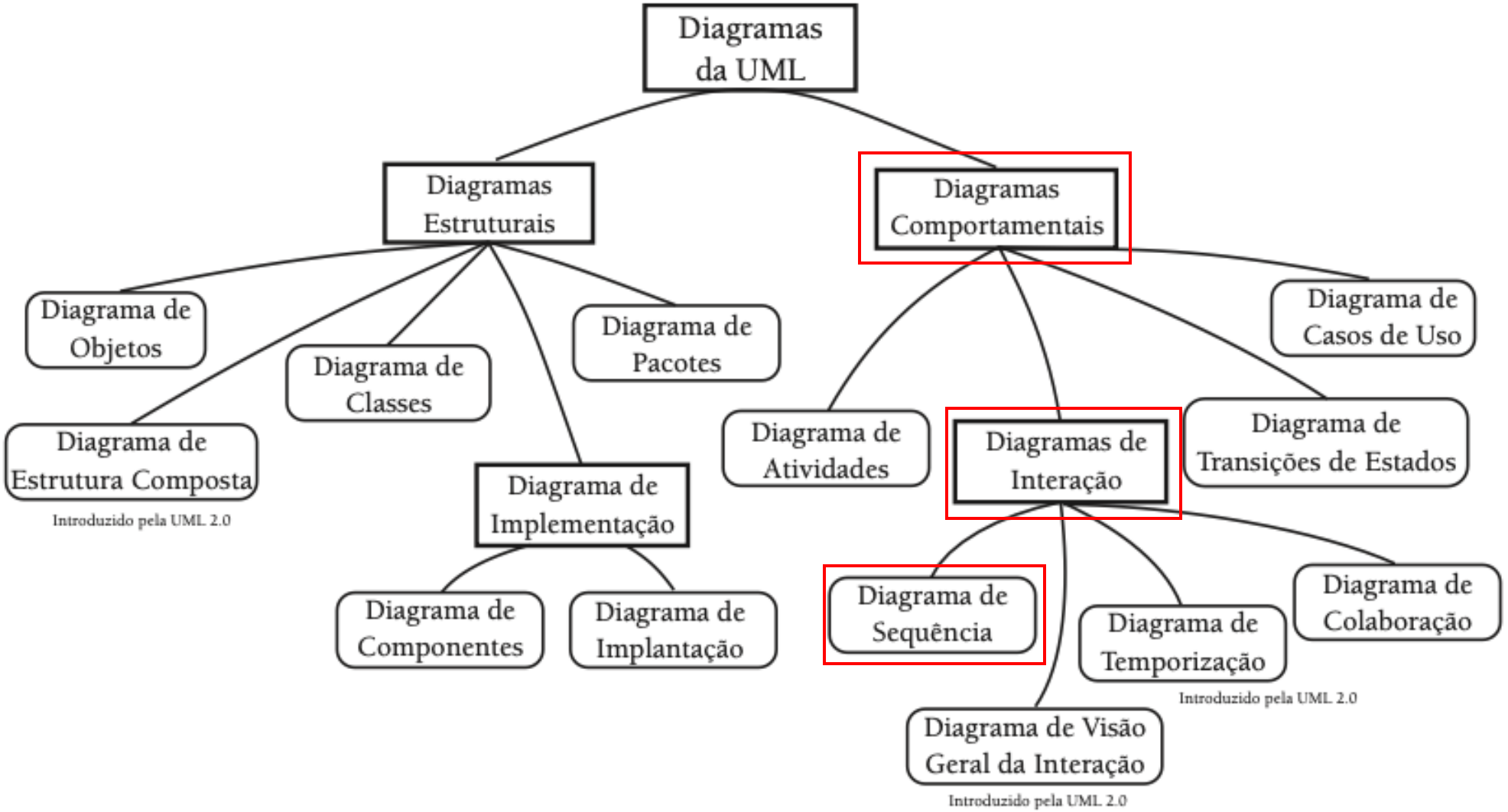
Diagrama de Sequência

- **Tipo:** Diagrama **comportamental**
- **Objetivo principal:** Representar as **interações entre objetos** ao longo do tempo
- **Ênfase:** Mostrar **a ordem temporal** em que as mensagens são trocadas
- **Permite identificar:**
  - Quais **mensagens** são enviadas entre os objetos
  - **Quem participa** da interação
  - **Em que sequência** essas mensagens ocorrem

Diagramas definidos pela UML



Diagramas definidos pela UML



Elementos básicos de um diagrama de sequência

- **Os principais elementos gráficos são:**
  - **Atores:** representam os usuários ou sistemas externos que interagem com o sistema.
  - **Objetos / Classes:** representam as entidades internas que trocam mensagens.
  - **Mensagens:** indicam a comunicação entre os objetos, podendo representar chamadas de métodos, retornos ou sinais.

Elementos básicos de um diagrama de sequência

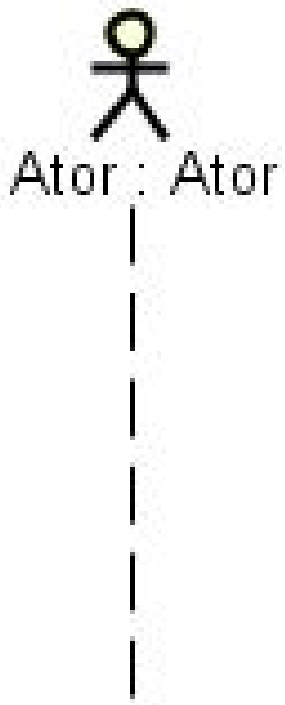
- **Linhas de vida:** mostram o tempo de existência de cada ator ou objeto durante a interação.
- **Focos de controle (ativação):** indicam o período em que um objeto está executando uma operação.
- **Criação e destruição de objetos:** representam o momento em que um objeto é instanciado ou removido durante a execução.
- **Iterações:** indicam repetições de mensagens ou comportamentos dentro do fluxo.

Elementos gráficos

1) Ator

Entidade externa que:

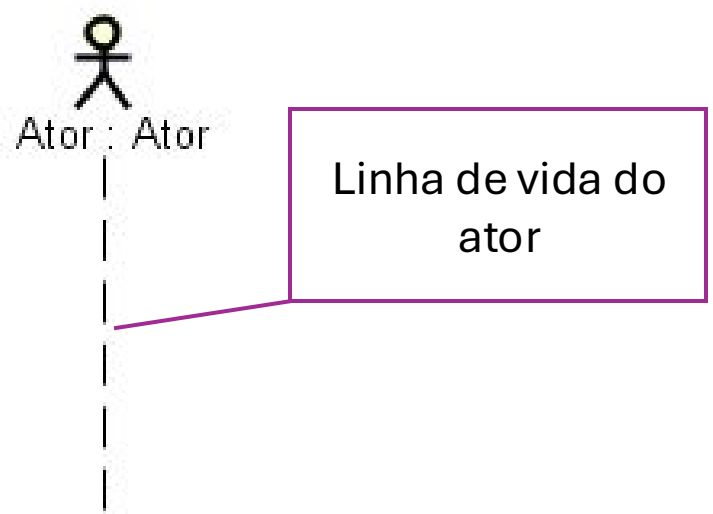
- Interage com o sistema;
- Solicita serviços;
- Tem a mesma representação do diagrama de caso de uso, porém, contendo uma linha de vida.

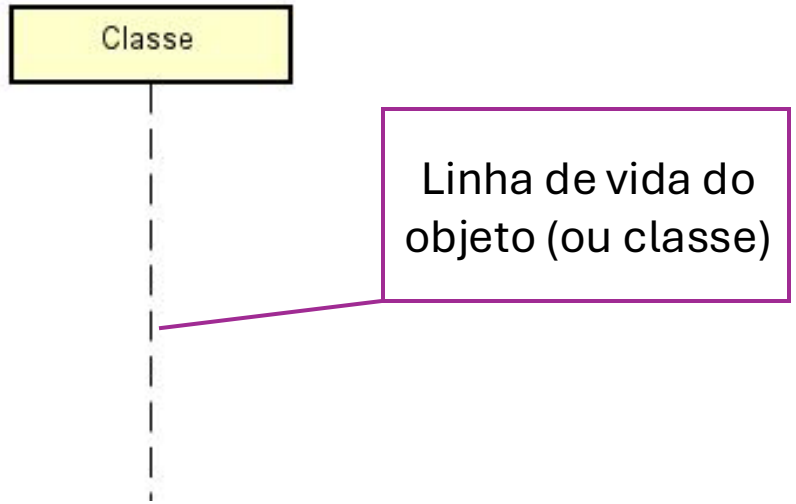


Elementos gráficos

2) Linha de Vida (lifeline)

- Linha vertical tracejada abaixo do objeto;
- Representa o tempo em que um objeto existe durante o processo;
- Das linhas de vida **partem as mensagens**.

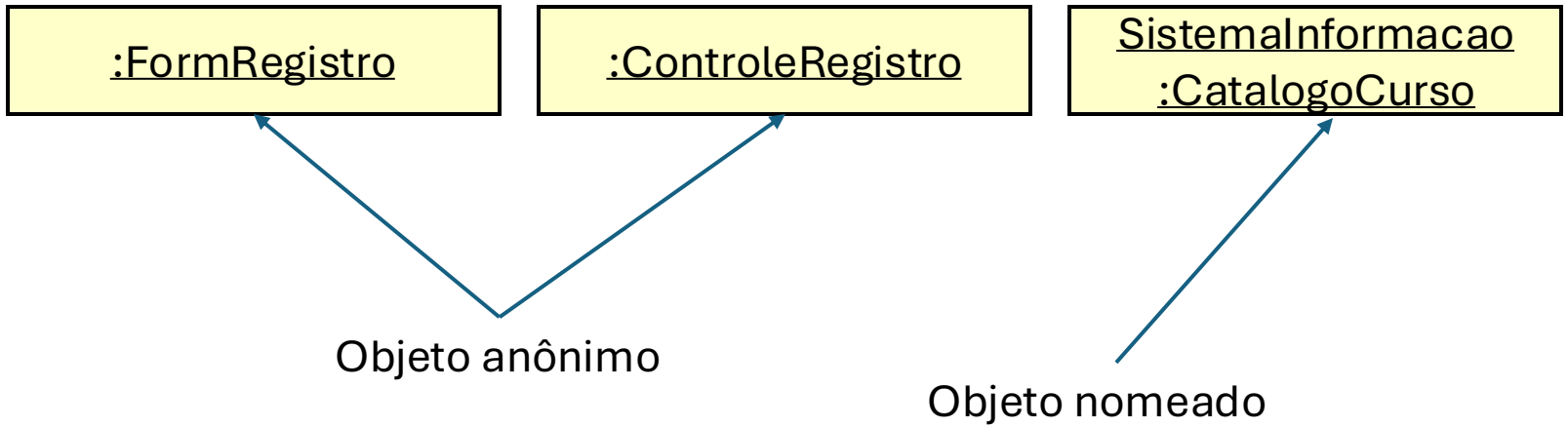




Elementos gráficos

2) Objetos

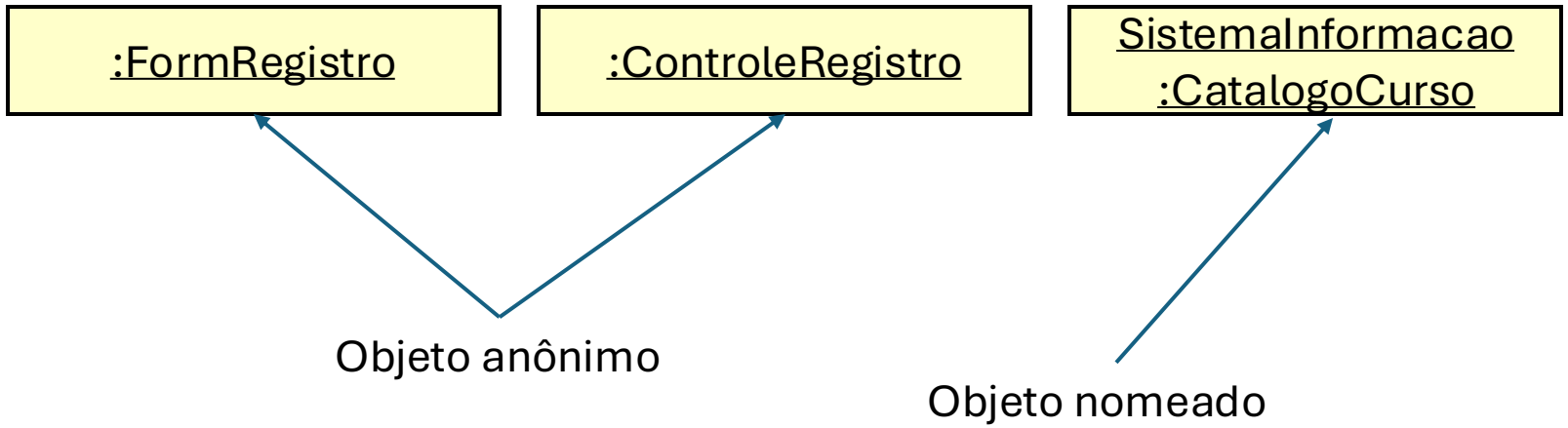
- São desenhados como retângulos com nomes sublinhados



Elementos gráficos

2) Objetos

Formato = ‘nome do objeto : tipo do objeto’



Elementos gráficos

3) Mensagem

- A **mensagem** é o **conceito central** da interação entre objetos.
- Em um **sistema orientado a objetos**, os objetos cooperam **trocando mensagens**.
- As **funcionalidades do sistema** são realizadas pelos objetos, e **a única forma de interação entre eles** é por meio dessas mensagens.
- Um **objeto envia uma mensagem** a outro quando **deseja que o receptor execute uma tarefa** específica.

Quando um objeto "precisa de ajuda" para realizar algo, ele solicita isso enviando uma mensagem.

Elementos gráficos

3) Mensagem

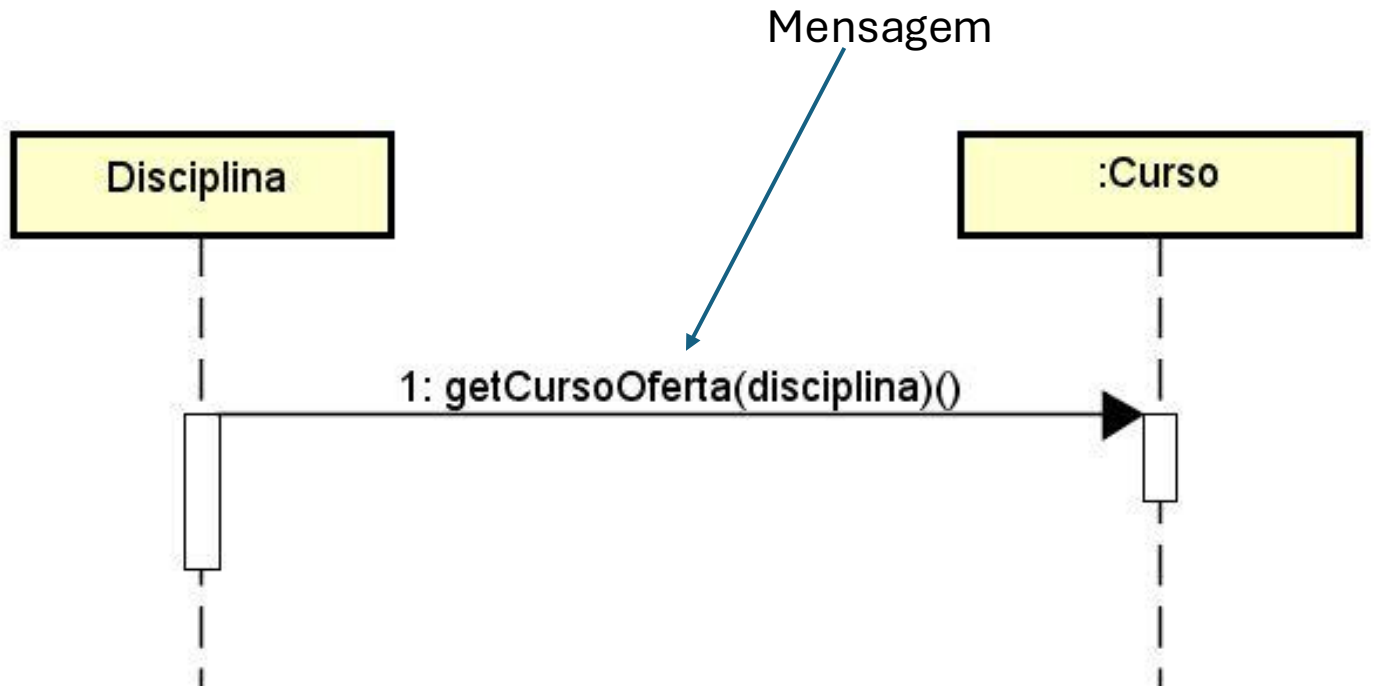
•Ao construir **diagramas de interação**, cada mensagem trocada entre objetos **implica na existência de uma operação** na classe do objeto receptor.

•Assim, uma **mensagem** representa uma **requisição** feita por um objeto **remetente** a um objeto **receptor**, pedindo que este execute **uma operação definida em sua classe**.

•A mensagem deve conter **informações suficientes** (parâmetros, dados, contexto) para que a operação possa ser **corretamente executada**.

Elementos gráficos

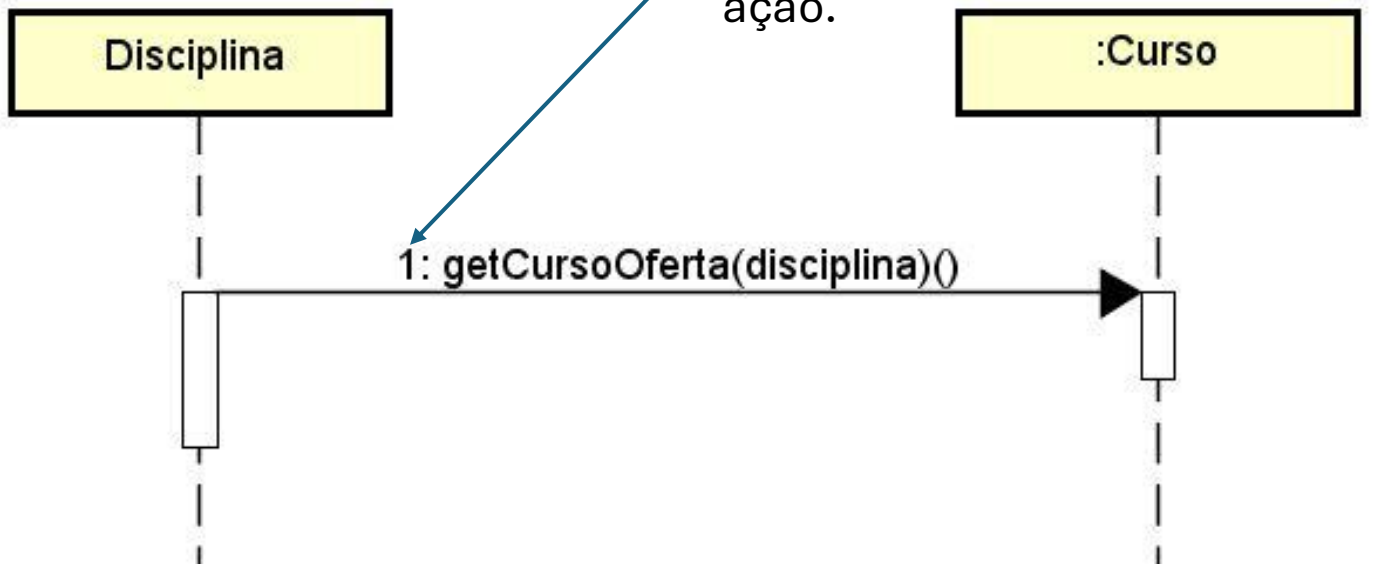
3) Mensagem



Elementos gráficos

3) Mensagem

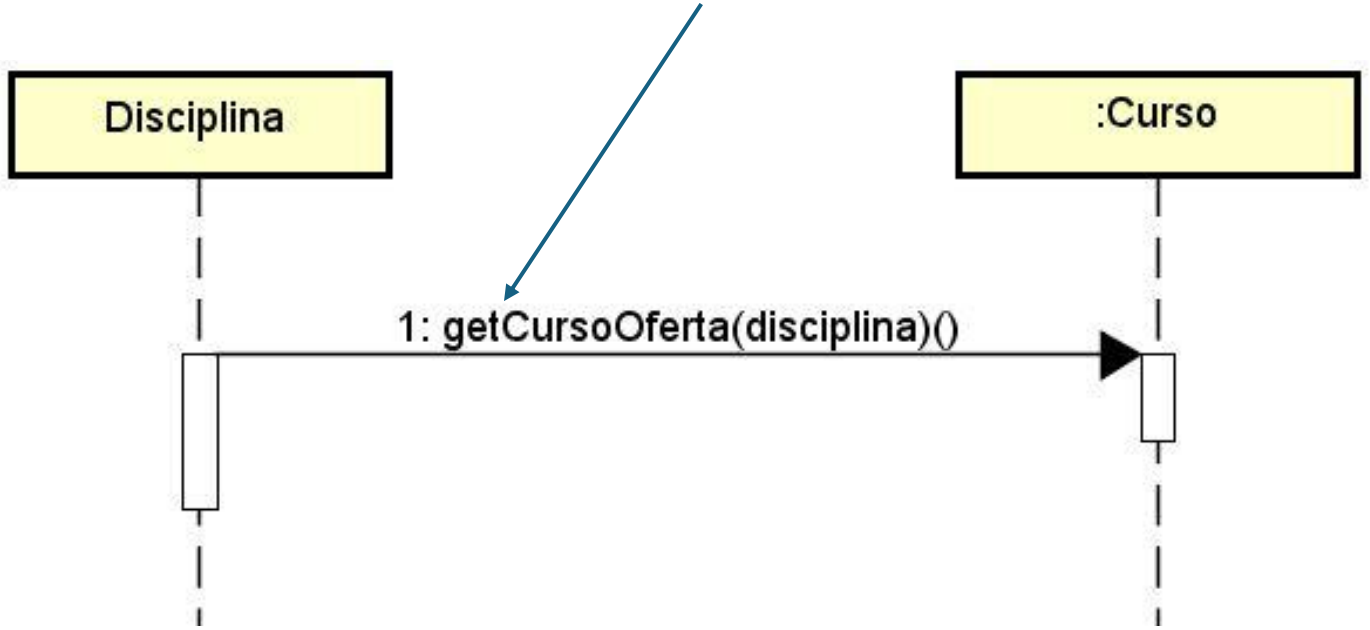
Número de ordem da mensagem: Em um diagrama de sequência, as mensagens são numeradas para mostrar a ordem em que ocorrem. Neste caso, é a primeira e única ação.



Elementos gráficos

3) Mensagem

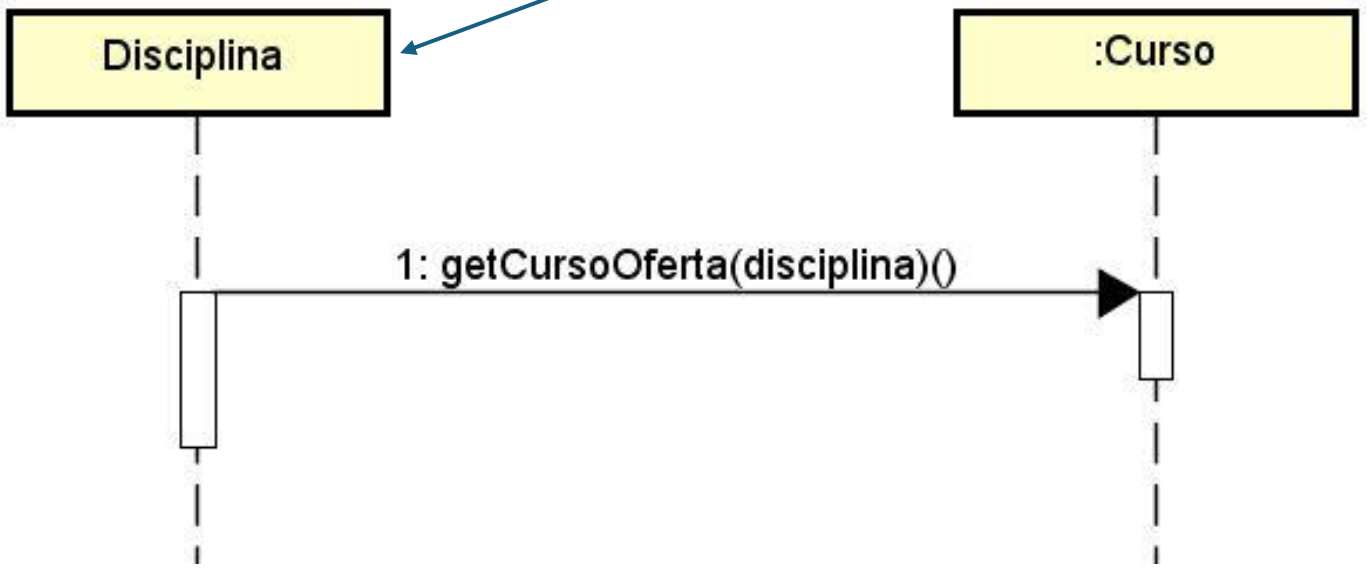
Este é o **nome da mensagem** ou o **método** que está sendo chamado.



Elementos gráficos

3) Mensagem

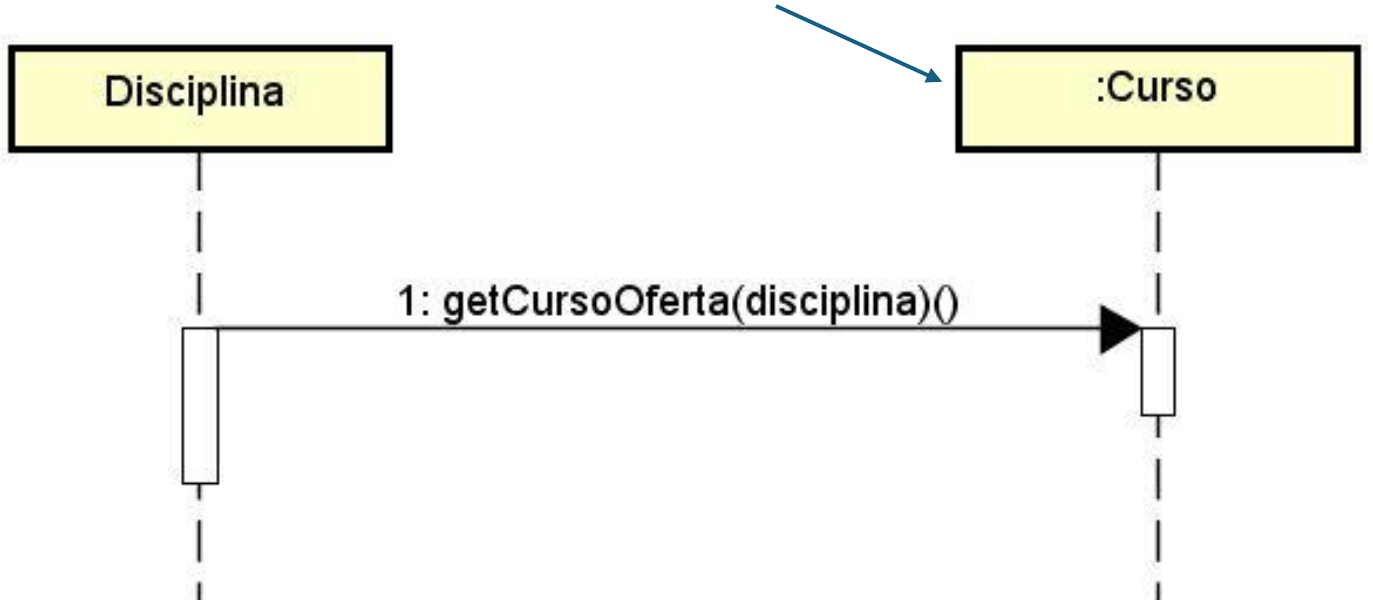
Disciplina: Este é o **objeto emissor** da mensagem. A linha tracejada que se estende para baixo a partir dele é a sua **linha de vida**, representando a sua existência no tempo durante a sequência.



Elementos gráficos

3) Mensagem

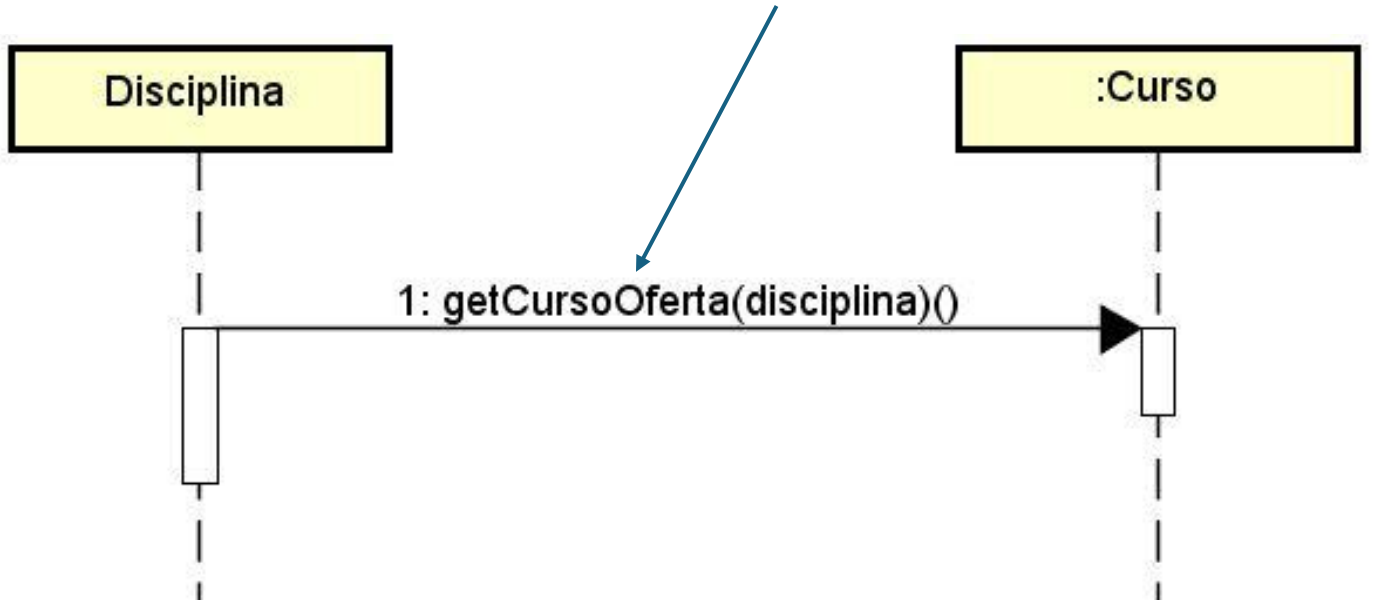
:Curso: Este é o **objeto receptor** da mensagem. O prefixo de dois pontos (:) é uma convenção comum em diagramas UML para indicar que é uma **instância** de uma classe chamada Curso.



Elementos gráficos

3) Mensagem

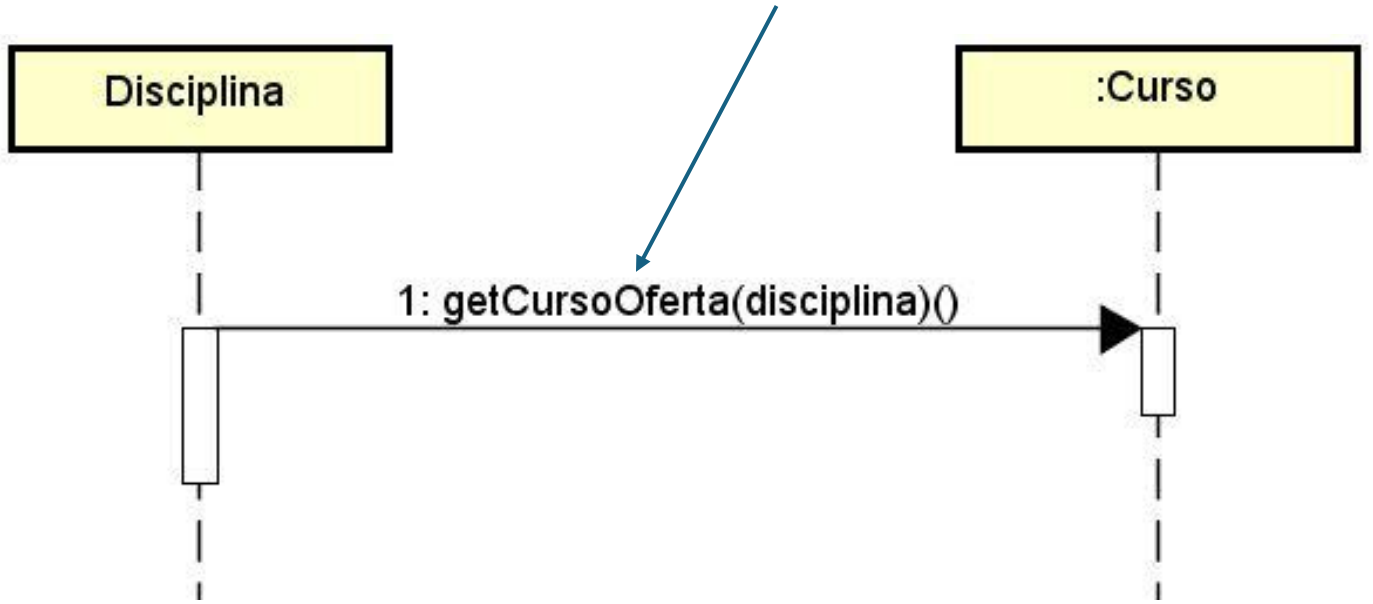
O objeto Disciplina envia uma mensagem (ou faz uma chamada de método) chamada getCursoOferta para o objeto Curso, passando a si mesmo (disciplina) como um parâmetro.



Elementos gráficos

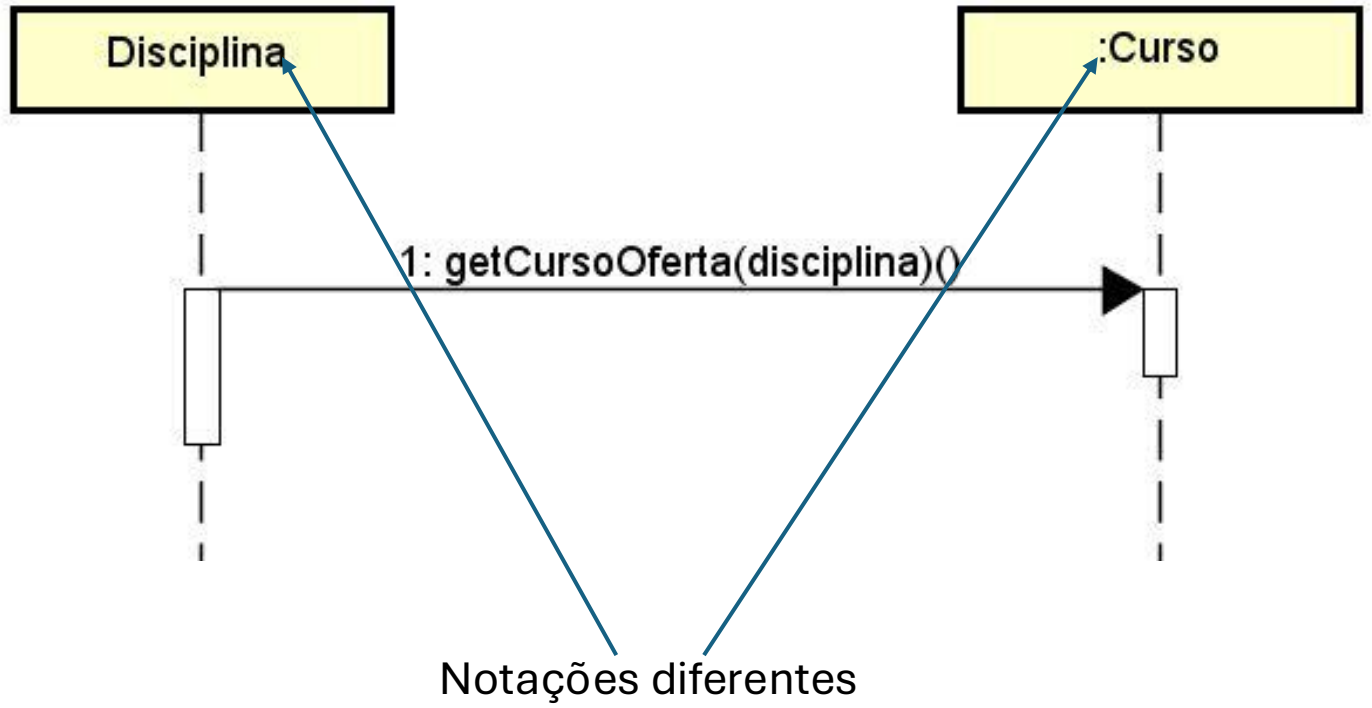
3) Mensagem

O objeto Disciplina envia uma mensagem (ou faz uma chamada de método) chamada getCursoOferta para o objeto Curso, passando a si mesmo (disciplina) como um parâmetro.



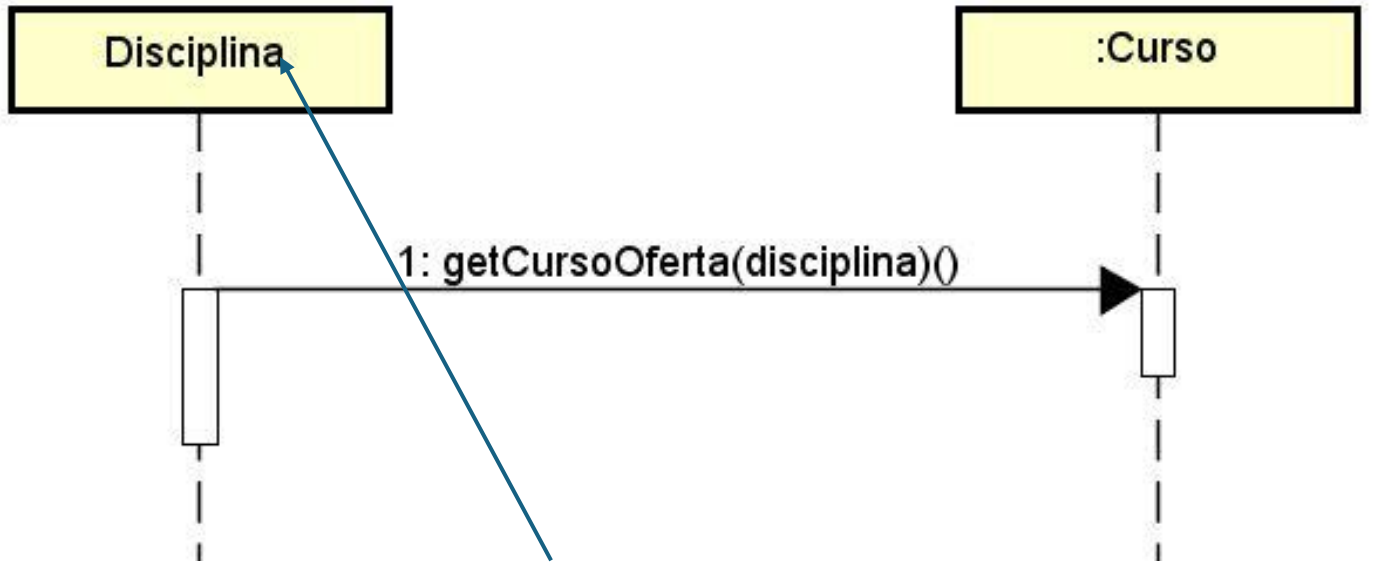
Elementos gráficos

3) Mensagem



Elementos gráficos

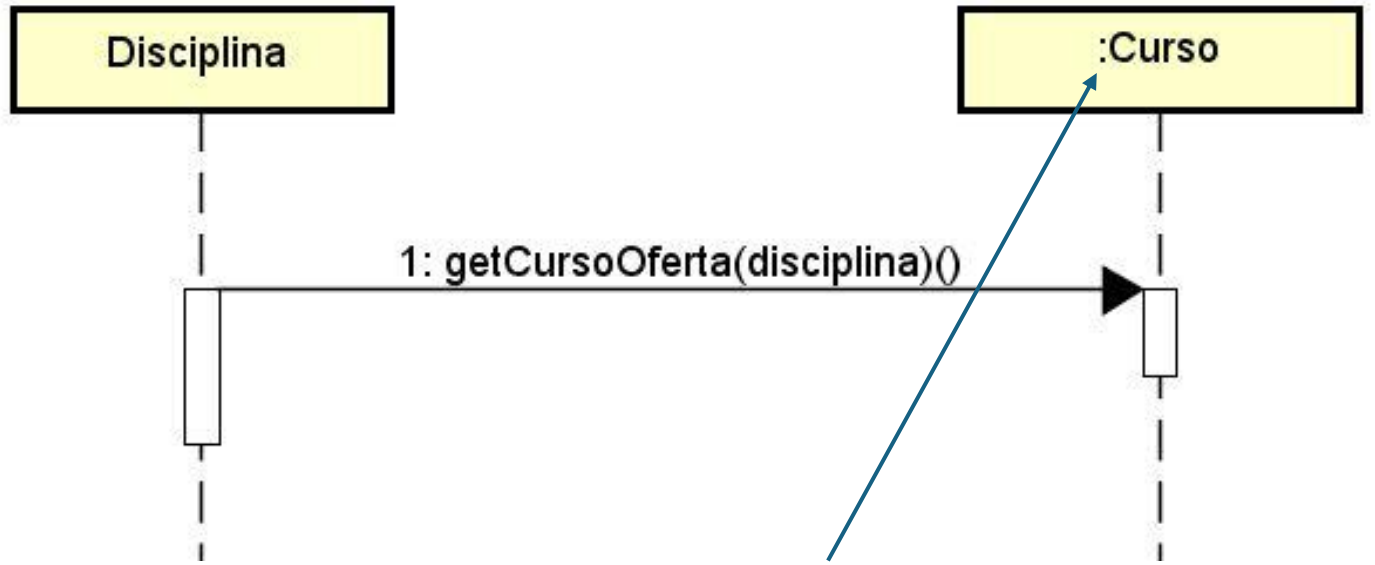
3) Mensagem



Esta notação, sem os dois pontos, é usada para mostrar uma **classe**. Ela sugere que o diagrama está mostrando uma interação que poderia ocorrer em um contexto mais genérico, onde qualquer instância da classe Disciplina pode iniciar a ação.

Elementos gráficos

3) Mensagem



Forma padrão de representar um **objeto (uma instância de uma classe)** em diagramas de sequência. O nome do objeto foi omitido, mas a presença dos dois pontos indica que estamos nos referindo a uma instância da classe Curso

Elementos gráficos

3) Mensagem

- As mensagens podem ser:
  - Simples;
  - Síncrona;
  - Assíncrona;
  - Retorno.

Elementos gráficos

Mensagem Simples

- Representa a **transferência de controle** de um objeto para outro, **sem detalhar** o tipo de comunicação envolvida.
- É usada quando o **foco está na sequência das interações**, e **não nos detalhes de execução**.
- **Não especifica** se a mensagem é **síncrona**, **assíncrona** ou **de retorno**.

*Em resumo: a mensagem simples mostra **quem interage com quem**, mas não **como** a comunicação ocorre.*

Elementos gráficos

Mensagem Síncrona

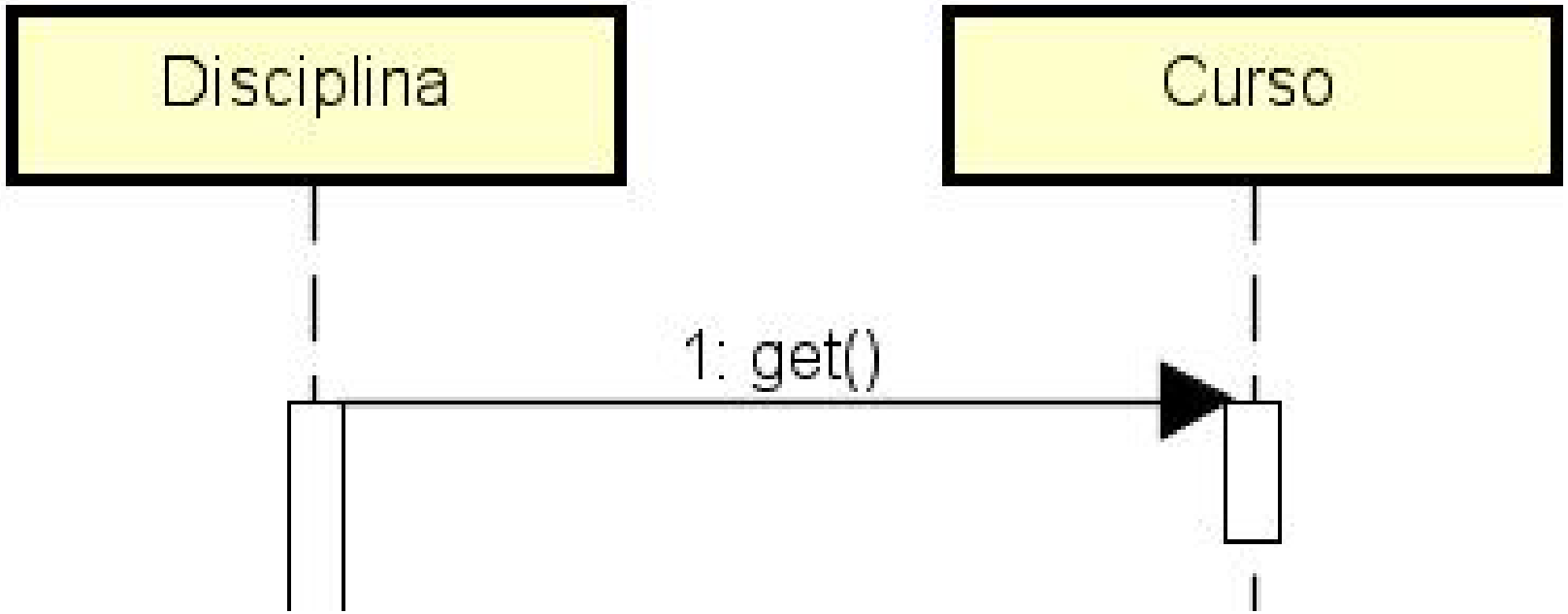
- Representa uma **comunicação com sincronismo rígido** entre os objetos envolvidos.
- O **objeto remetente** envia a mensagem e **aguarda a resposta** do **objeto receptor** antes de continuar sua execução.
- Esse tipo de mensagem indica que o **controle permanece suspenso** no objeto que chamou a operação até que o processamento da mensagem seja concluído.

Em outras palavras, o remetente "espera" o término da ação antes de seguir seu fluxo.

Ex: Chamadas de método tradicionais (por exemplo, obj.operacao())

Elementos gráficos

Mensagem Síncrona - Exemplo



Elementos gráficos

Mensagem Assíncrona

- Representa uma **comunicação sem dependência direta** entre os estados dos objetos envolvidos.
- O **objeto de origem** envia a mensagem e **continua sua execução normalmente**, **sem esperar** que o **objeto de destino** conclua o processamento.
- A **seta é aberta** (linha cheia com ponta em meia-seta), indicando **envio sem bloqueio**.

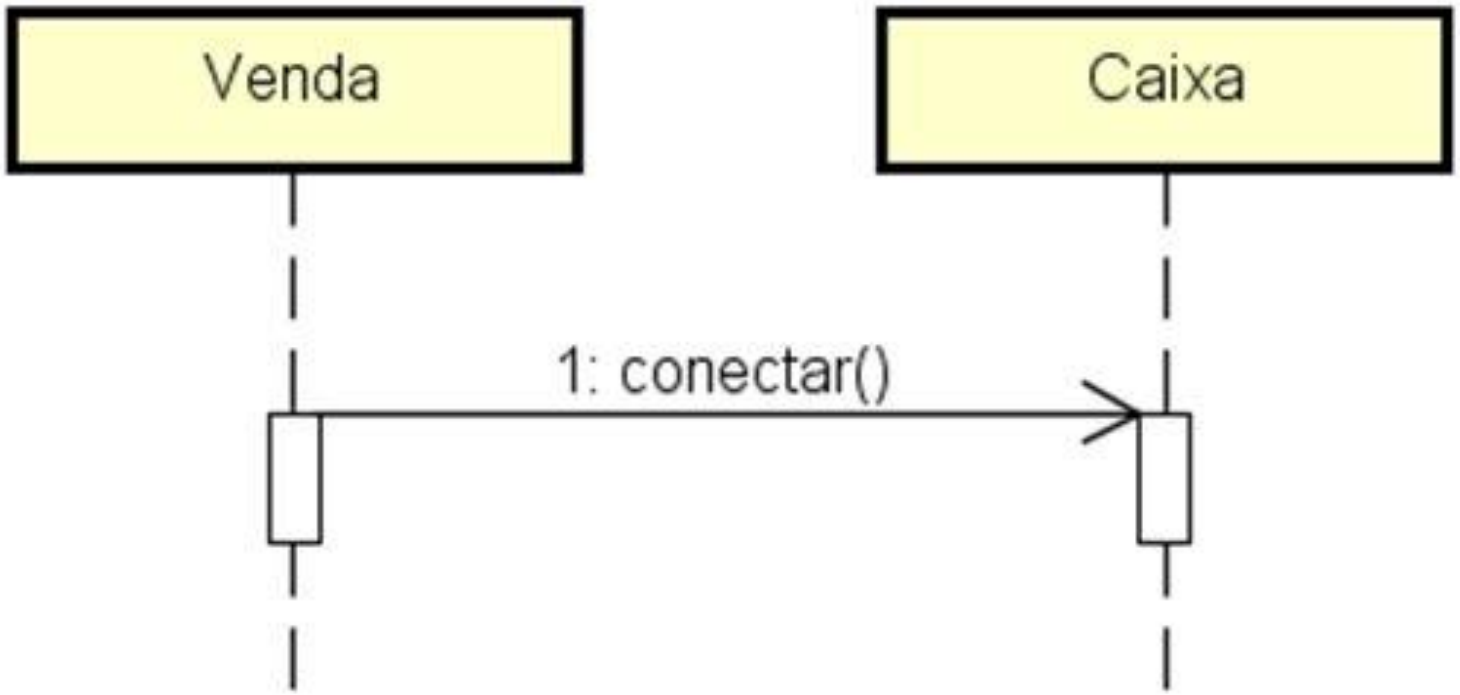
Em resumo: o remetente "dispara" a ação e segue em frente.

Elementos gráficos

- **Mensagem Assíncrona - Exemplo**
- Imagine um sistema de pedidos:
  - O objeto **Cliente** envia uma mensagem para **ServiçoDeNotificação**: enviarEmailConfirmacao()
  - Enquanto o serviço envia o e-mail, o **Cliente** continua processando outras ações, como **registrar o pagamento** ou **atualizar o carrinho**.
- **Esse é um exemplo de mensagem assíncrona**, pois o remetente não espera o término da operação antes de prosseguir.

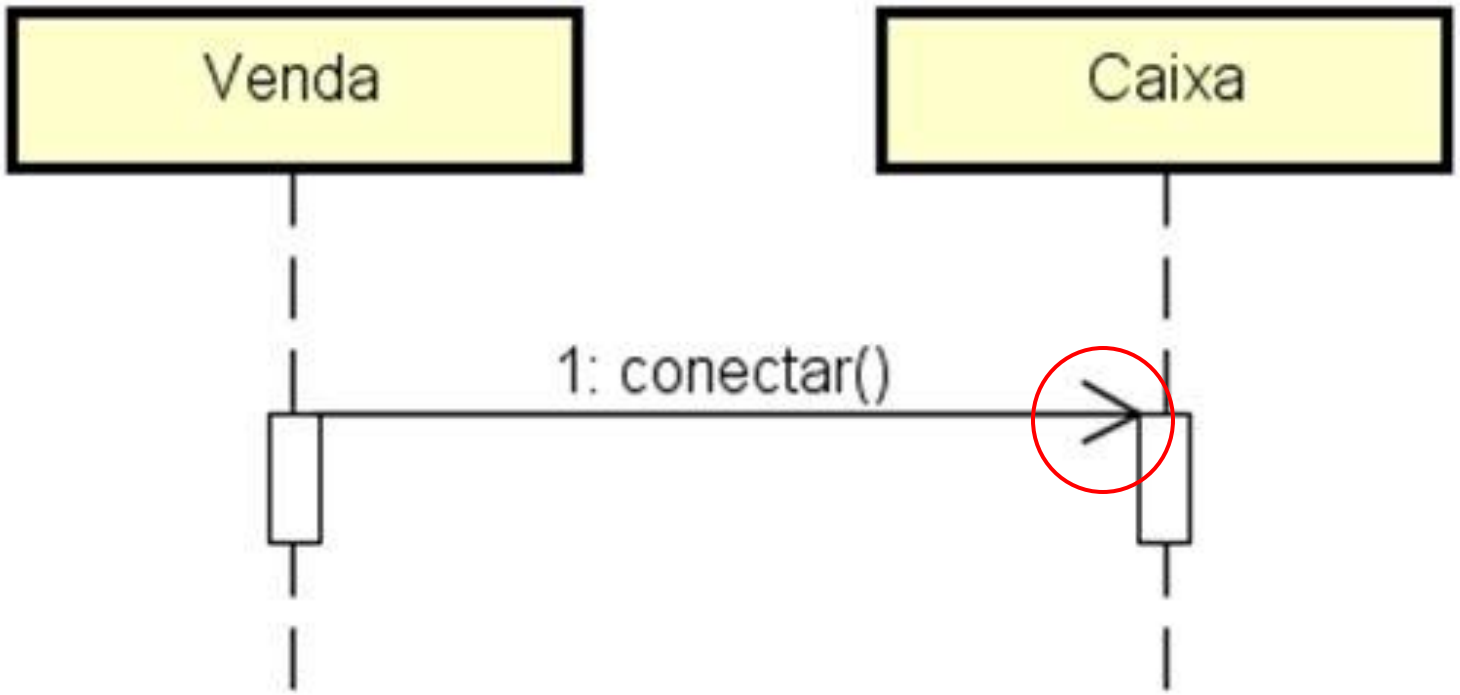
Elementos gráficos

- **Mensagem Assíncrona - Exemplo prático**



Elementos gráficos

- **Mensagem Assíncrona - Exemplo prático**



Elementos gráficos

Mensagens de Retorno

- Representam a **resposta** de um objeto **ao remetente** que fez a chamada.
- Podem retornar **valores específicos** (resultado de um método) ou apenas indicar que a **execução foi concluída com sucesso**.
- Podem ser enviadas **a outros objetos ou ao ator** que iniciou a interação.
- São **opcionais** nos diagramas de sequência, podem ser omitidas quando o retorno não é relevante para a compreensão do fluxo.
- São representadas por uma **linha tracejada com seta**, apontando de volta ao remetente.

Elementos gráficos

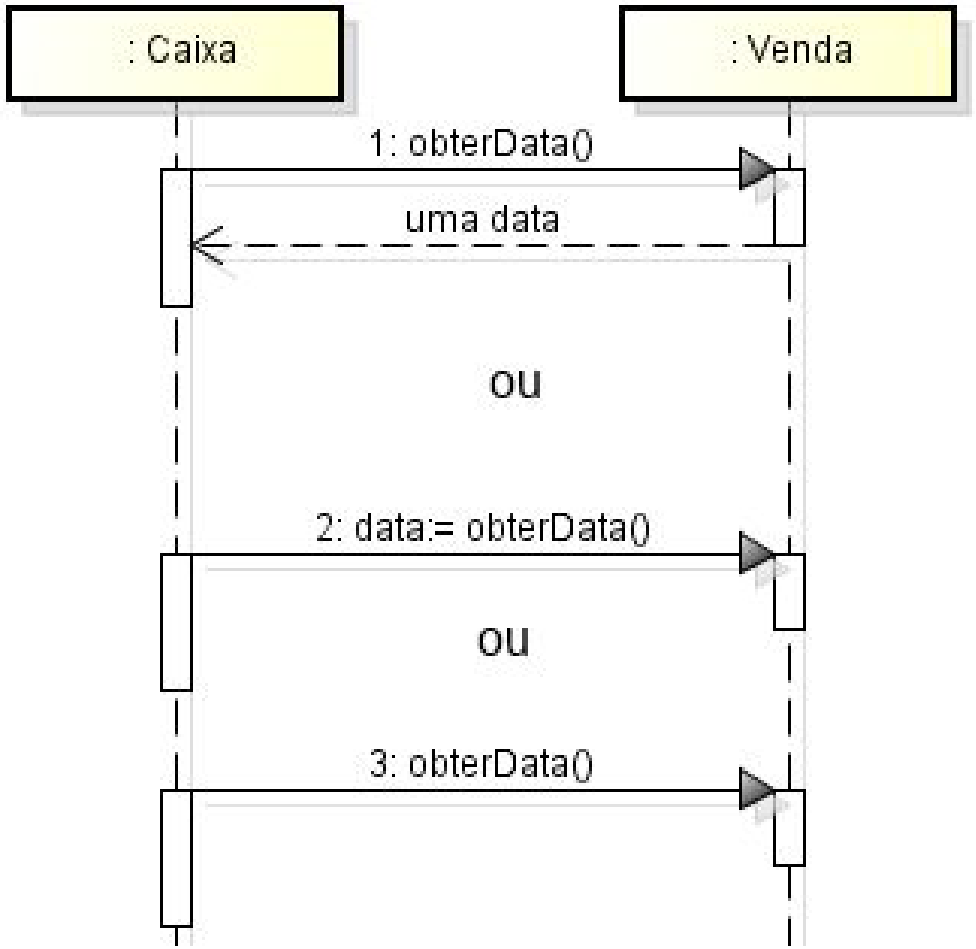
Mensagens de Retorno - Exemplo

- Um objeto **ControleDePedidos** envia a mensagem calcularTotal() para o objeto **Carrinho**
- Após o processamento, o **Carrinho** envia uma **mensagem de retorno** com o valor calculado: total = 259.90

A linha tracejada indicará o retorno da operação ao objeto que fez a chamada.

Elementos gráficos

Mensagens de Retorno



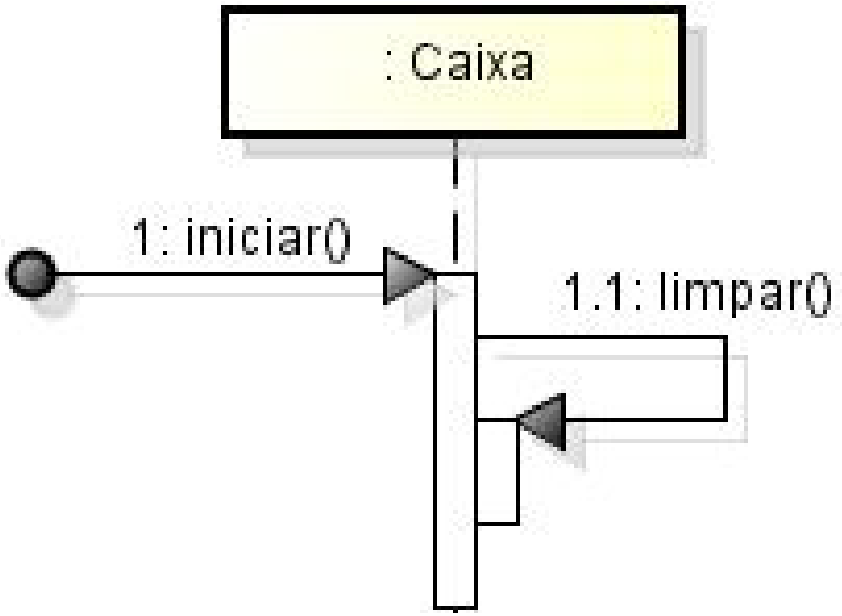
Elementos gráficos

Mensagens *Self* ou *This*

- Representam **autochamadas**, isto é, **mensagens que um objeto envia para si mesmo**.
- Indicam que o **objeto está executando uma operação interna**, chamando outro método pertencente à **mesma classe**.
- São utilizadas para **evidenciar a sequência interna de ações** dentro de um único objeto.
- No diagrama, são representadas por uma **seta em laço**, que retorna à própria linha de vida do objeto.

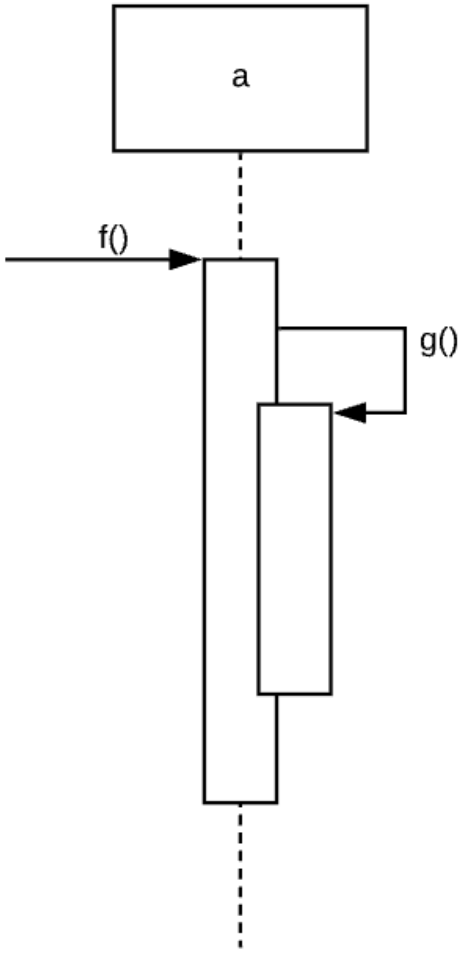
Elementos gráficos

Mensagens *Self* ou *This*



Elementos gráficos

Mensagens *Self* ou *This*




```
class A {

  void g() {
    ...
  }

  void f() {
    ...
    g();
    ...
  }

  main() {
    A a = new A();
    a.f();
  }
}
```

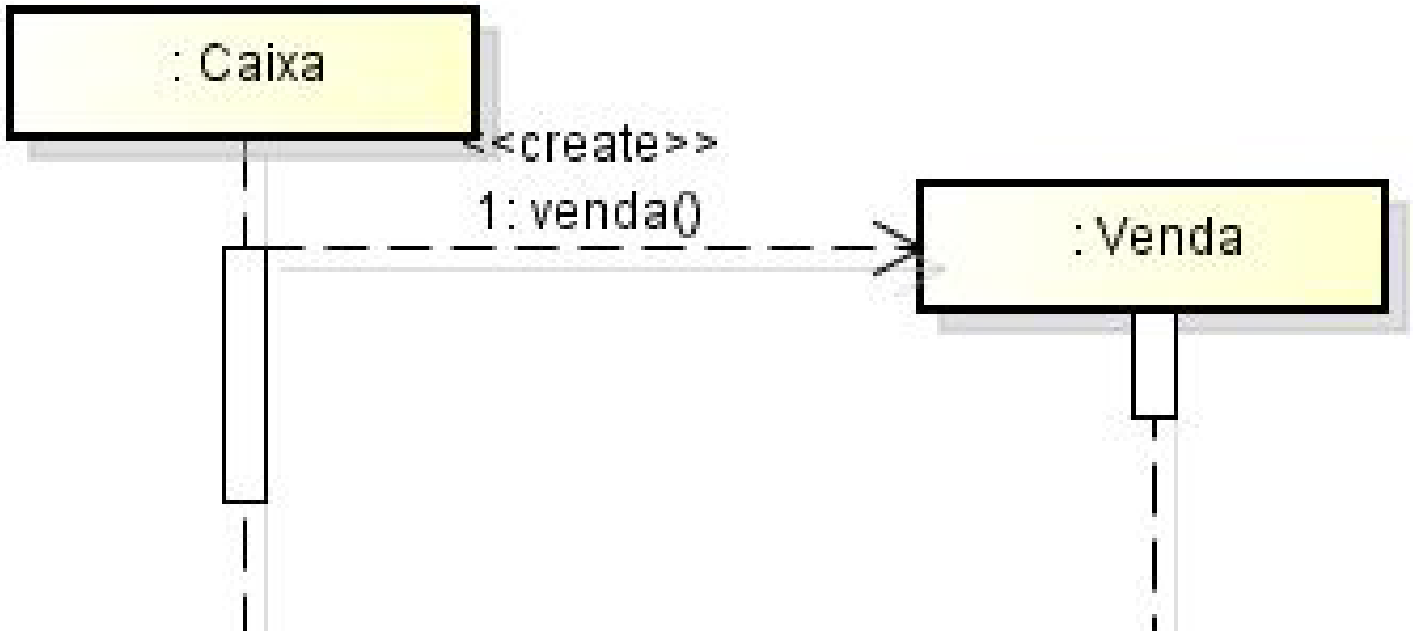
Elementos gráficos

Mensagens de Criação de Instância

- Indicam o **momento em que um novo objeto é criado** durante a interação.
- A **seta da mensagem** aponta diretamente para o **retângulo que representa o novo objeto**, mostrando que ele **passa a existir a partir daquele ponto** do diagrama.
- Essa mensagem corresponde à **chamada de um método construtor**, responsável por **instanciar o objeto**.

Elementos gráficos

Mensagens de Criação de Instância



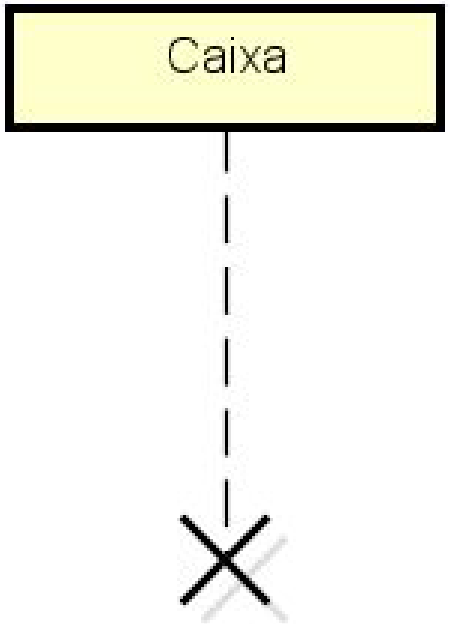
Elementos gráficos

Destruição de Objetos

- A **destruição de um objeto** é indicada por um **“X”** no final de sua **linha de vida**.
- A **mensagem de destruição** pode utilizar o estereótipo **«destroy»**.
- A destruição pode ocorrer **ao receber uma mensagem** ou **durante o retorno de uma chamada**.
- Um **objeto também pode se autodestruir**, encerrando sua própria existência no sistema.

Elementos gráficos

Destruição de Objetos



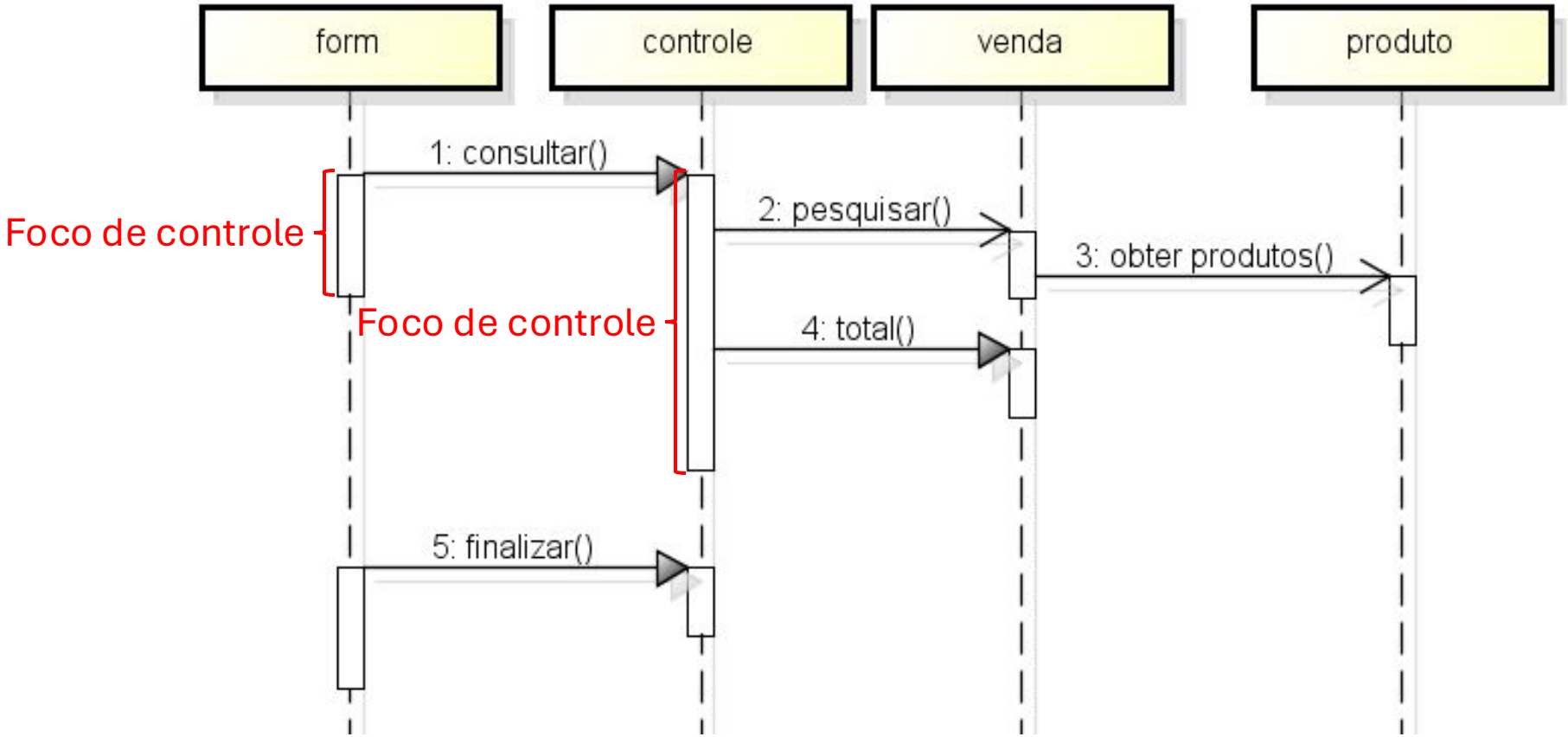
Elementos gráficos

Foco de Controle (ou Ativação)

- Representa o **período em que um objeto está ativo** durante uma interação.
- Indica os **momentos em que o objeto executa uma ou mais operações** relacionadas a um processo específico.
- Mostra visualmente **quando o objeto está processando uma mensagem** ou realizando alguma ação.

Elementos gráficos

Foco de Controle (ou Ativação)



Organização dos elementos

- Os **objetos participantes** da interação são dispostos **horizontalmente** na parte superior do diagrama.
- **Abaixo de cada objeto** existe uma **linha de vida**, que representa sua existência ao longo do tempo.
- Cada linha de vida possui um **foco de controle**, indicando **quando o objeto está executando alguma ação**.
- As **mensagens trocadas entre objetos** são representadas por **linhas horizontais rotuladas**,
  partindo da **linha de vida do remetente** e chegando à **linha de vida do receptor**.

Organização dos elementos

- A **posição vertical das mensagens** indica a **ordem temporal** em que elas são enviadas.
- A **ordem de envio** também pode ser deduzida a partir das **expressões de sequência**.
- É possível representar **criação e destruição de objetos** diretamente no diagrama.

Organização dos elementos

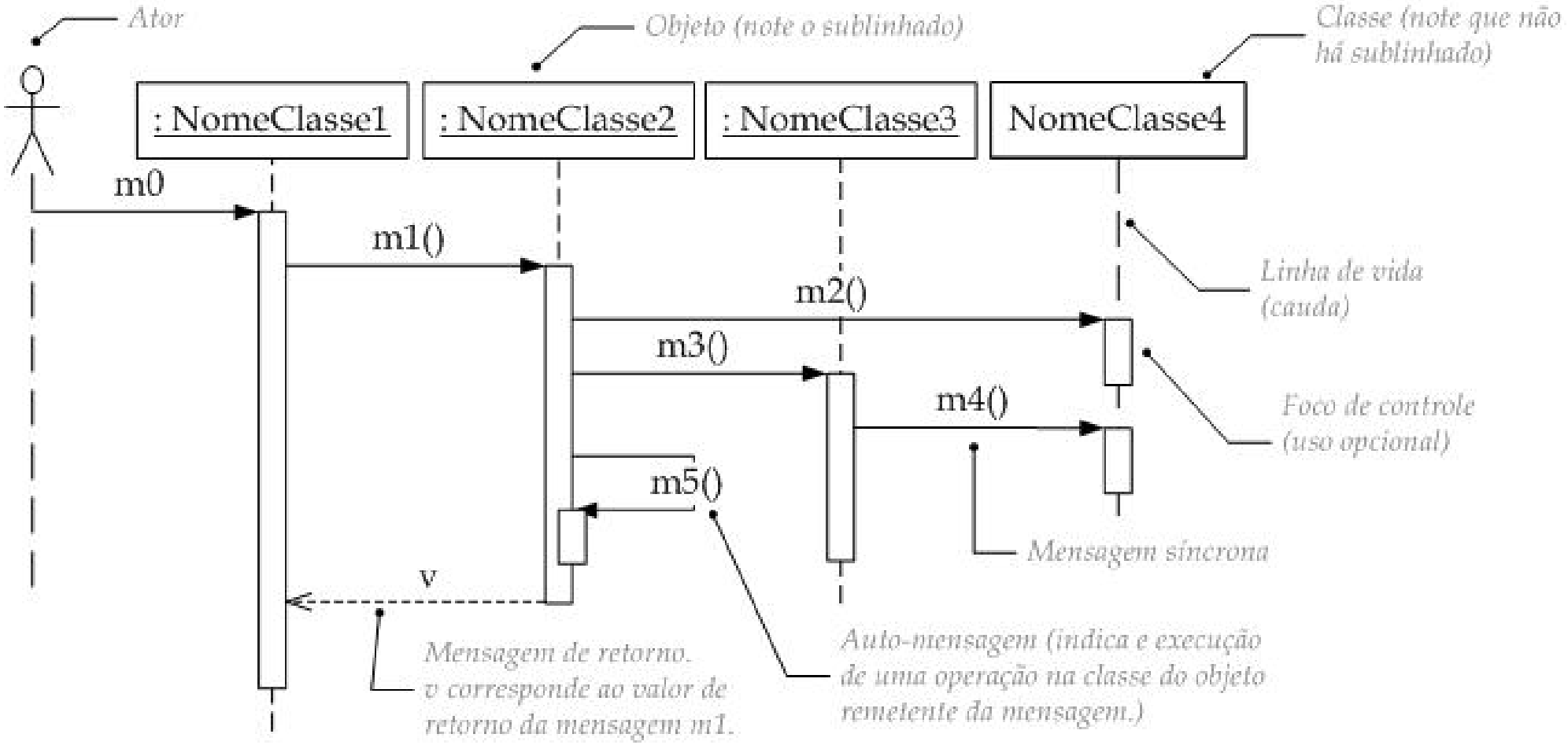
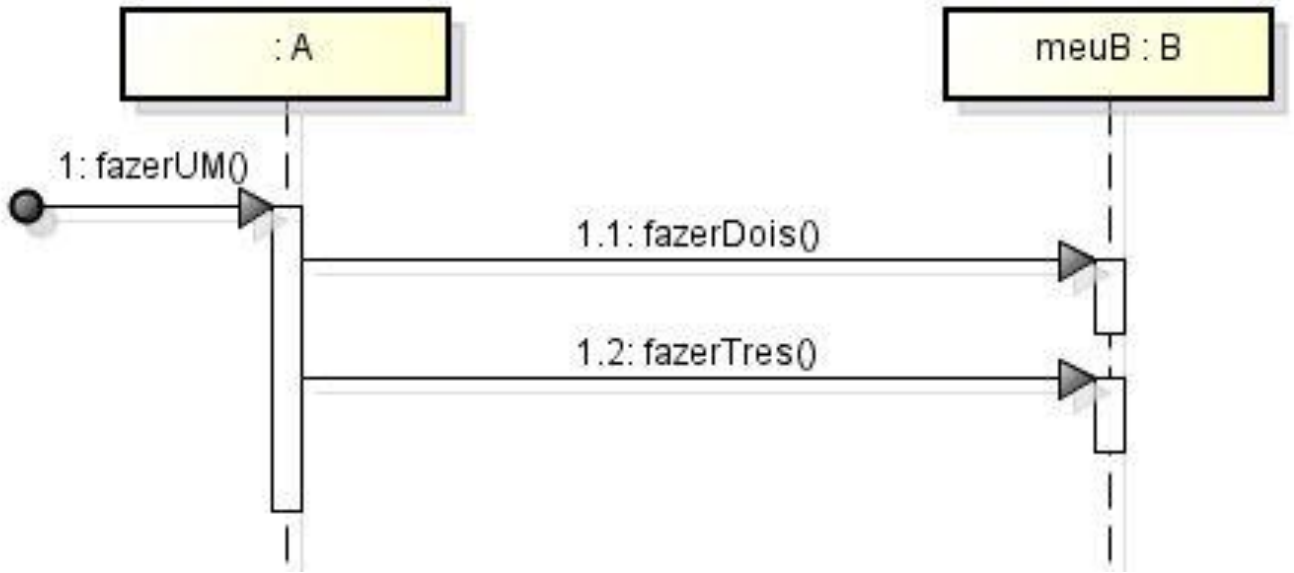


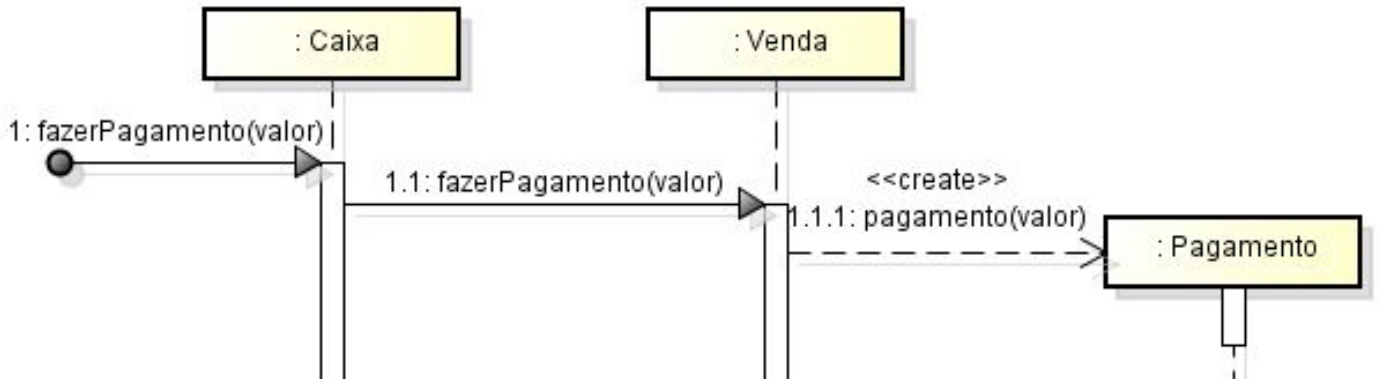
Diagrama de Sequência – Ex 1




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

    public void fazerUm() {
        meuB.fazerDois();
        meuB.fazerTres();
    }
}
```

Diagrama de Sequência – Ex 2




```java
public class Venda {
    private Pagamento pagamento;

    public void fazerPagamento(double valor) {
        pagamento = new Pagamento(valor);
    }
}

public class Pagamento {
    private double valor;

    public Pagamento(double valor) {
        this.valor = valor;
    }
}
```

Quadros (molduras) de iteração

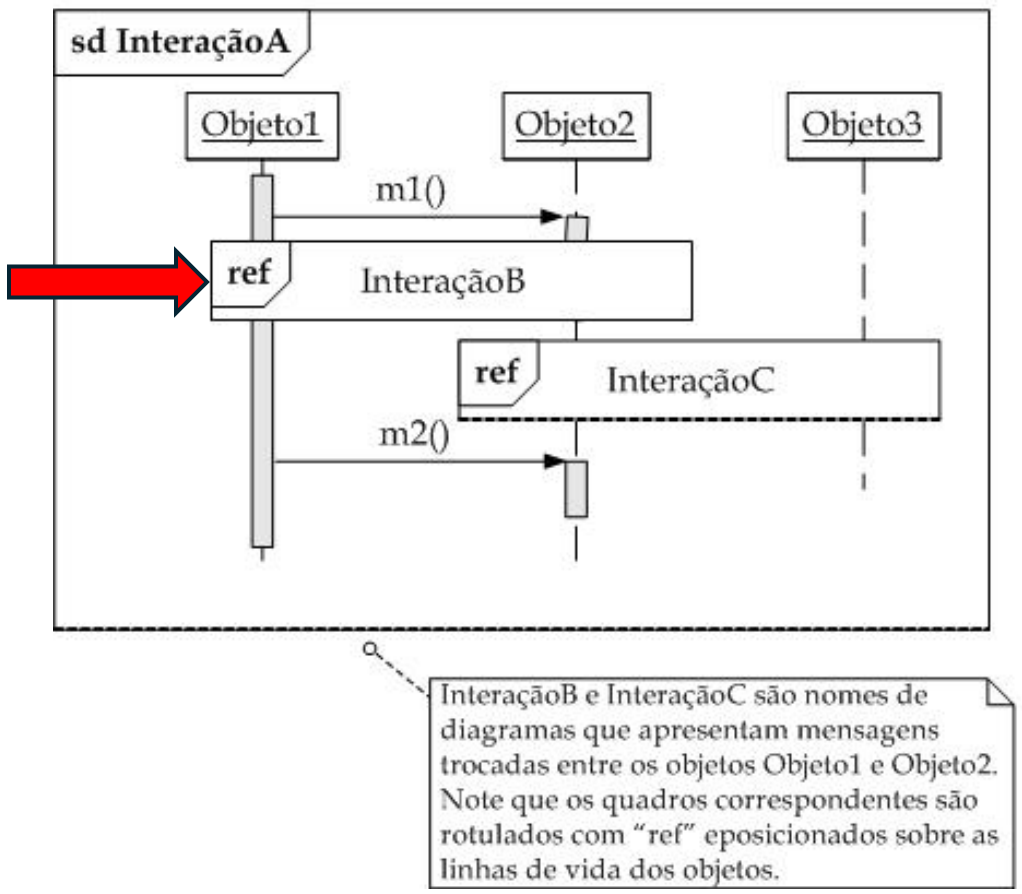
- São **elementos gráficos** utilizados para **modularizar** e **organizar** a construção de **diagramas de sequência** (ou de comunicação).
- Delimitam uma **seção do diagrama**, tornando-o mais legível e estruturado.
- **Objetivos principais:**
  - **Nomear** o diagrama que aparece dentro do quadro.
  - **Referenciar** outro diagrama definido separadamente.
  - **Definir o fluxo de controle** da interação representada.

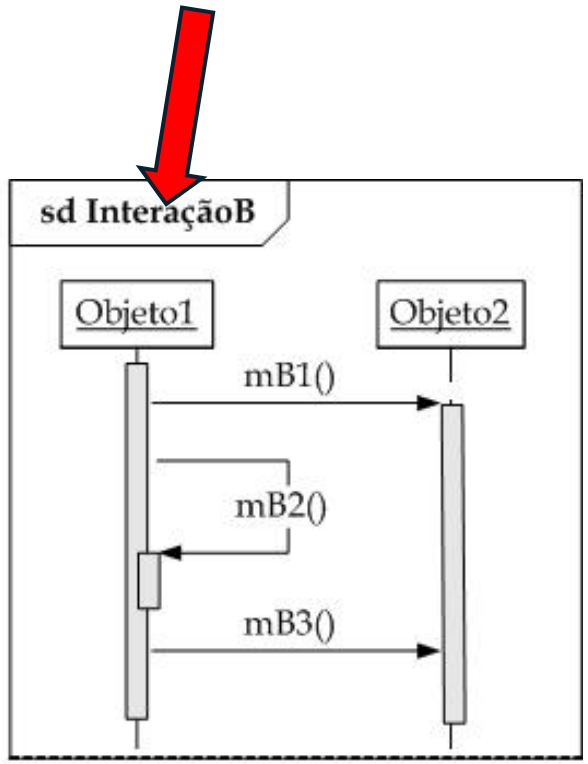
Operador ref (Referred Interaction)

- Permite que uma **interação existente seja referenciada** dentro de outro diagrama de sequência.
- É utilizado por meio do operador **ref**, inserido dentro de uma **moldura**.
- Indica que o conteúdo mostrado deve ser **substituído** (ou detalhado) por **outro diagrama** com o **mesmo nome** especificado após o operador ref.

Em outras palavras, o ref cria uma ligação entre diagramas, permitindo reutilizar interações já definidas.

Quadros (molduras) de iteração

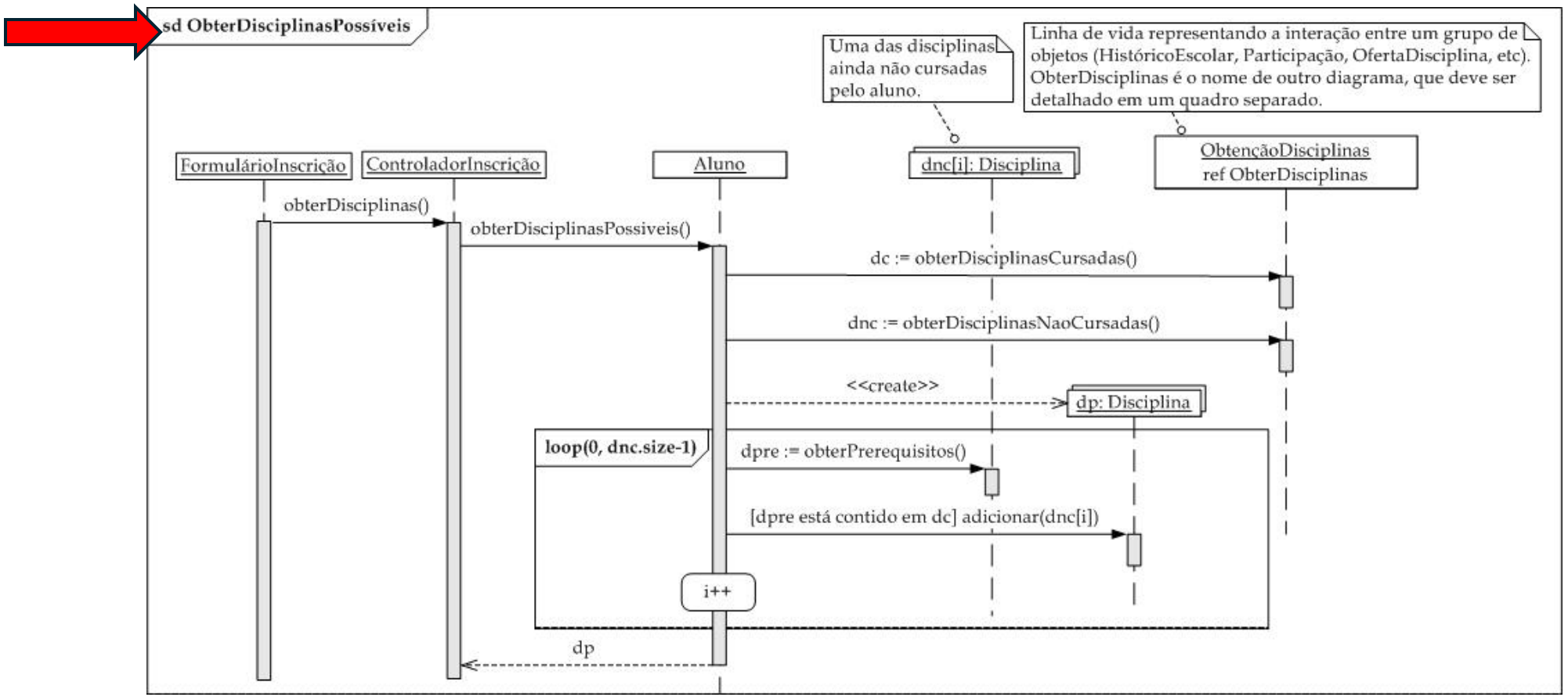




Outros Operadores

Operador moldura	Significado
alt	Abreviatura Alternatives. Fragmento alternativo que define uma escolha entre dois ou mais comportamentos
loop	Fragmento de loop enquanto a guarda for verdadeira. Pode-se usar também escrever loop(n) para indicar a iteração de n vezes.Existe discussão de que a especificação será aperfeiçoada para definir um loop FOR , exemplo, loop(1,1,10)
opt	Abreviatura de Option. Fragmento que representa uma escolha de comportamento onde esse comportamento será ou não executado, não havendo uma escolha entre mais de um comportamento possível
par	Abreviatura de Parallel. Determina que o fragmento combinado representa uma execução paralela de dois ou mais comportamento
critical	Região crítica. Identifica uma operação atômica que não pode ser interrompida por outro processo até ser totalmente conluída
Break	Quebra. Indica uma 'quebra' na execução normal do processo. É usado principalmente para modelar o tratamento de execessão

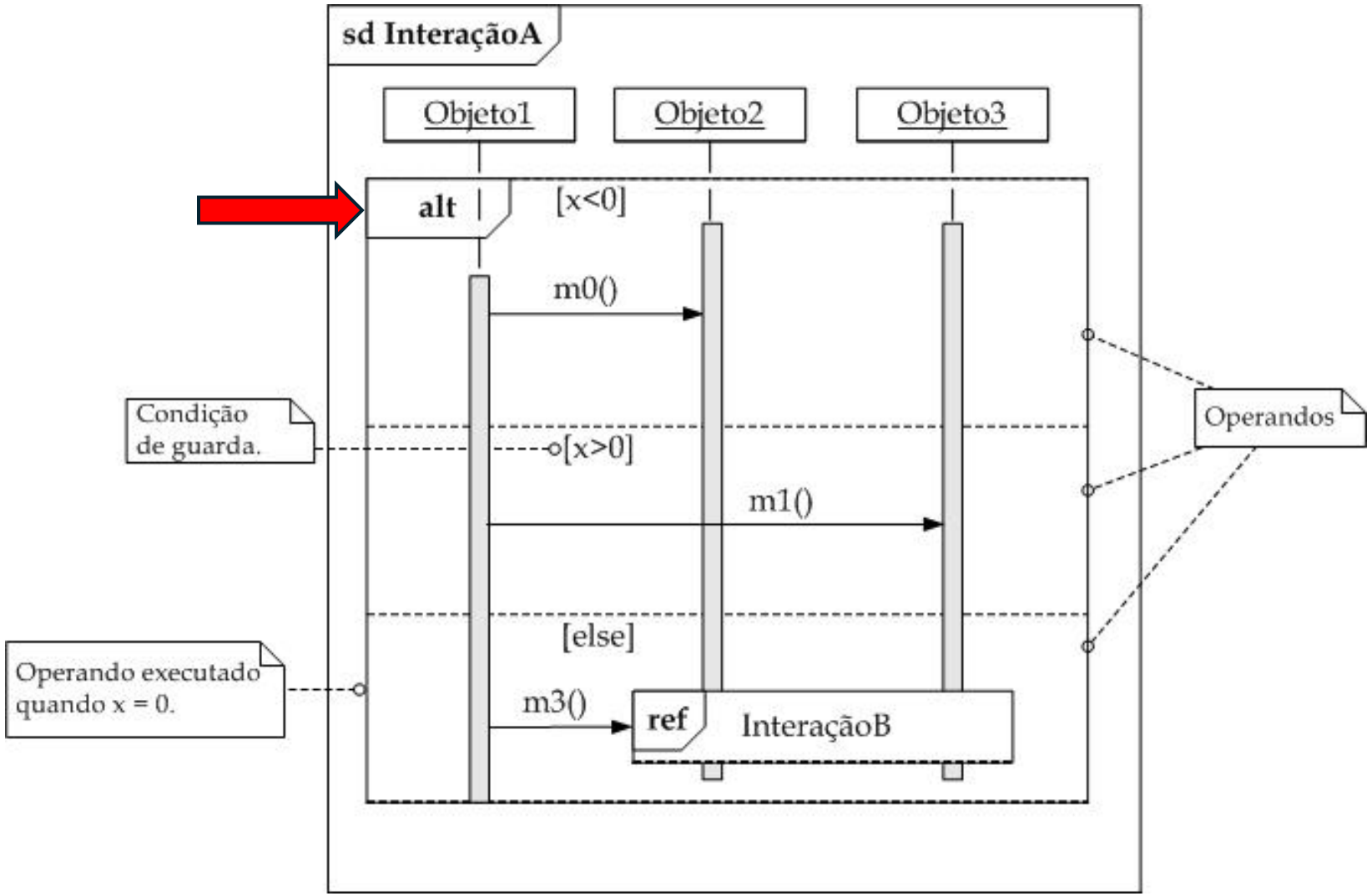
Diagramas nomeados



Modularização de Interações: Fluxo de Controle – Alternativas

- A **modularização de interações** permite **dividir o comportamento de um sistema** em partes menores e mais compreensíveis.
- O **fluxo de controle alternativo** é utilizado quando há **diferentes caminhos possíveis** de execução dentro de uma interação.
- É representado por uma **moldura com o operador alt (alternative)**.
- Cada **região interna** da moldura corresponde a **uma condição** (ou **guard**) que determina **qual caminho será seguido**.
- **Apenas um dos blocos é executado**, de acordo com a condição verdadeira no momento da interação.

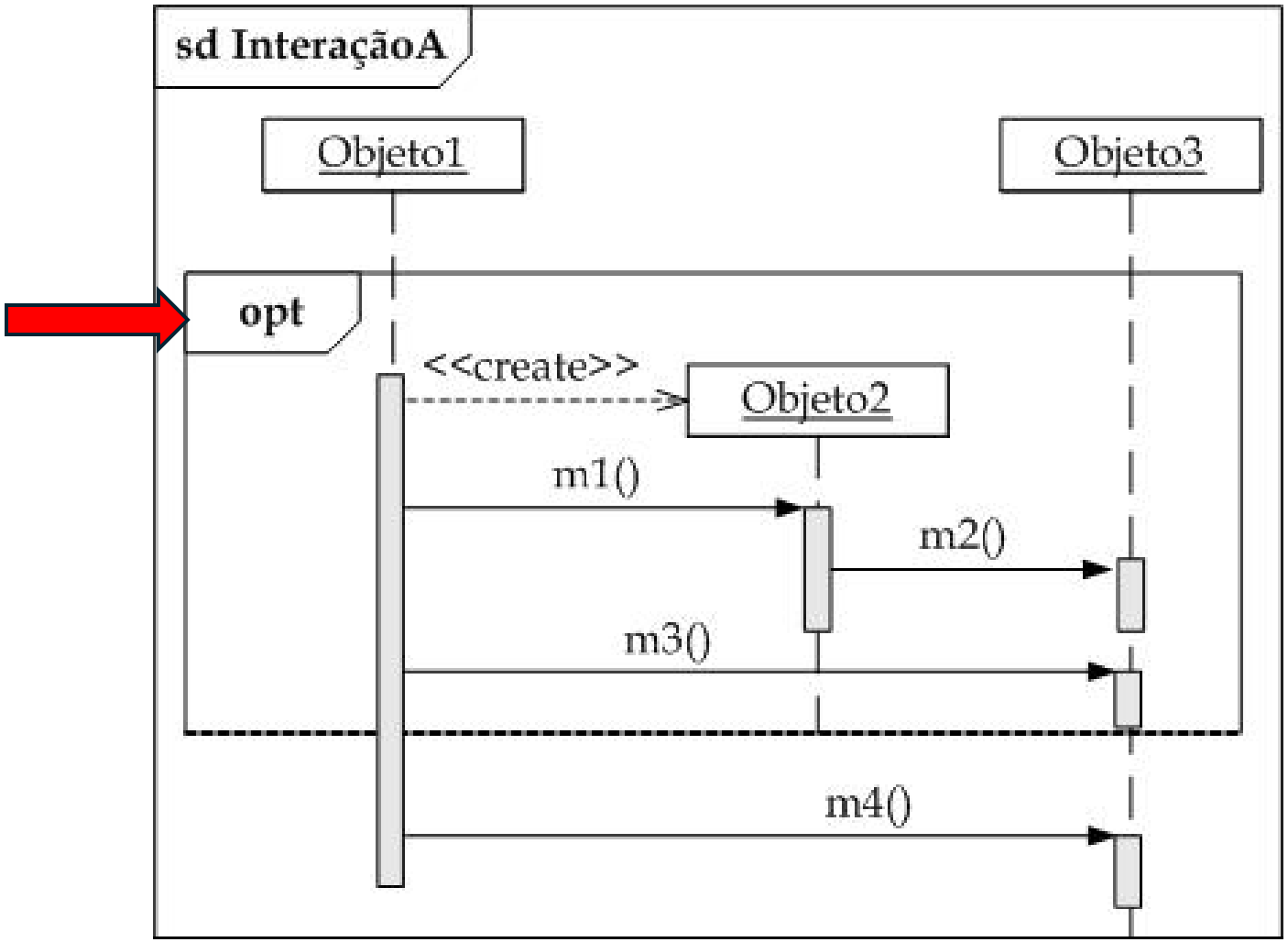
Modularização de Interações: Fluxo de Controle – Alternativas



Modularização de Interações: Fluxo de Controle – Opções

- O **fluxo de controle por opções** representa um **comportamento condicional** que **pode ou não ocorrer** durante a interação.
- É utilizado quando existe **apenas uma condição possível**, diferentemente do operador alt, que apresenta múltiplas alternativas.
- É representado por uma **moldura com o operador opt (option)**.
- Caso a **condição especificada** (guard) seja verdadeira, as **mensagens dentro da moldura são executadas**.
- Se a condição for falsa, **nenhuma ação ocorre** e o fluxo normal da interação continua.

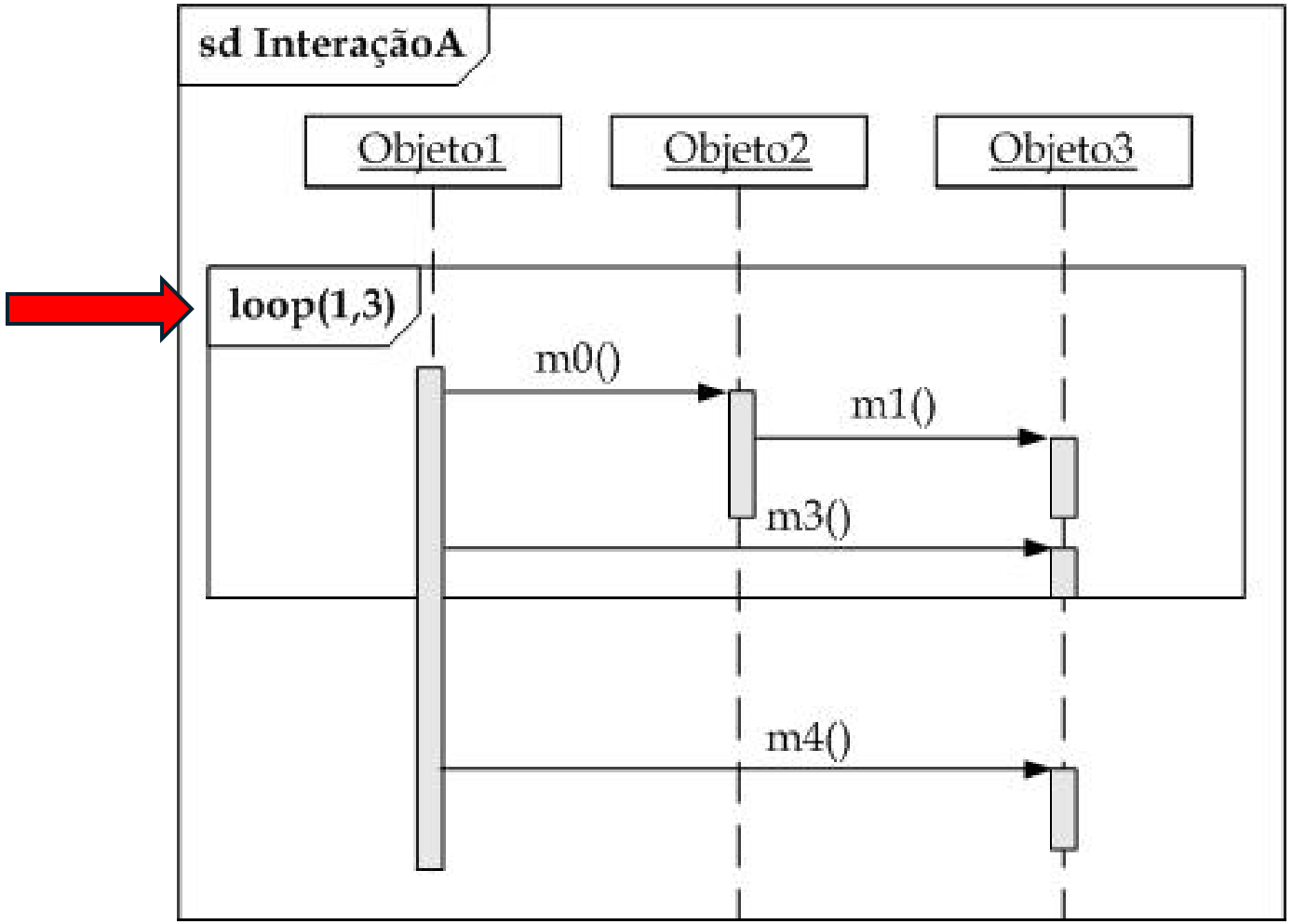
Modularização de Interações: Fluxo de Controle – Alternativas



Modularização de Interações: Fluxo de Controle – Iterações

- O **fluxo de controle por iteração** representa **repetições** de uma sequência de mensagens em um diagrama de sequência.
- É utilizado quando **um mesmo conjunto de interações ocorre várias vezes**, de forma **sequencial ou condicional**.
- É representado por uma **moldura com o operador loop**.
- A moldura pode conter uma **condição de repetição** (guard), que define **quantas vezes** ou **enquanto** a interação deve se repetir.
- O **número de repetições** pode ser especificado por um **intervalo**, uma **expressão lógica** ou uma **condição de parada**.

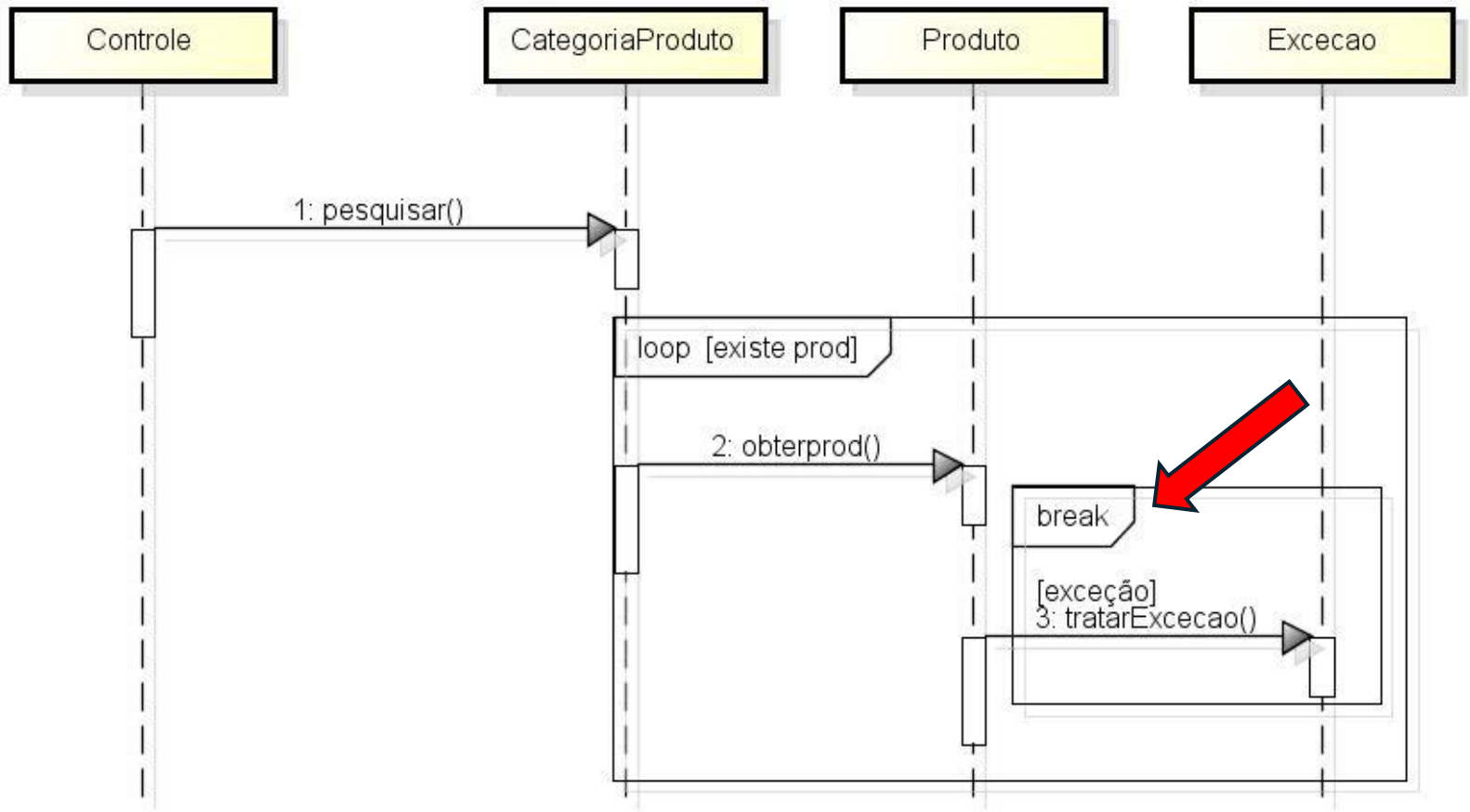
Modularização de Interações: Fluxo de Controle – Alternativas



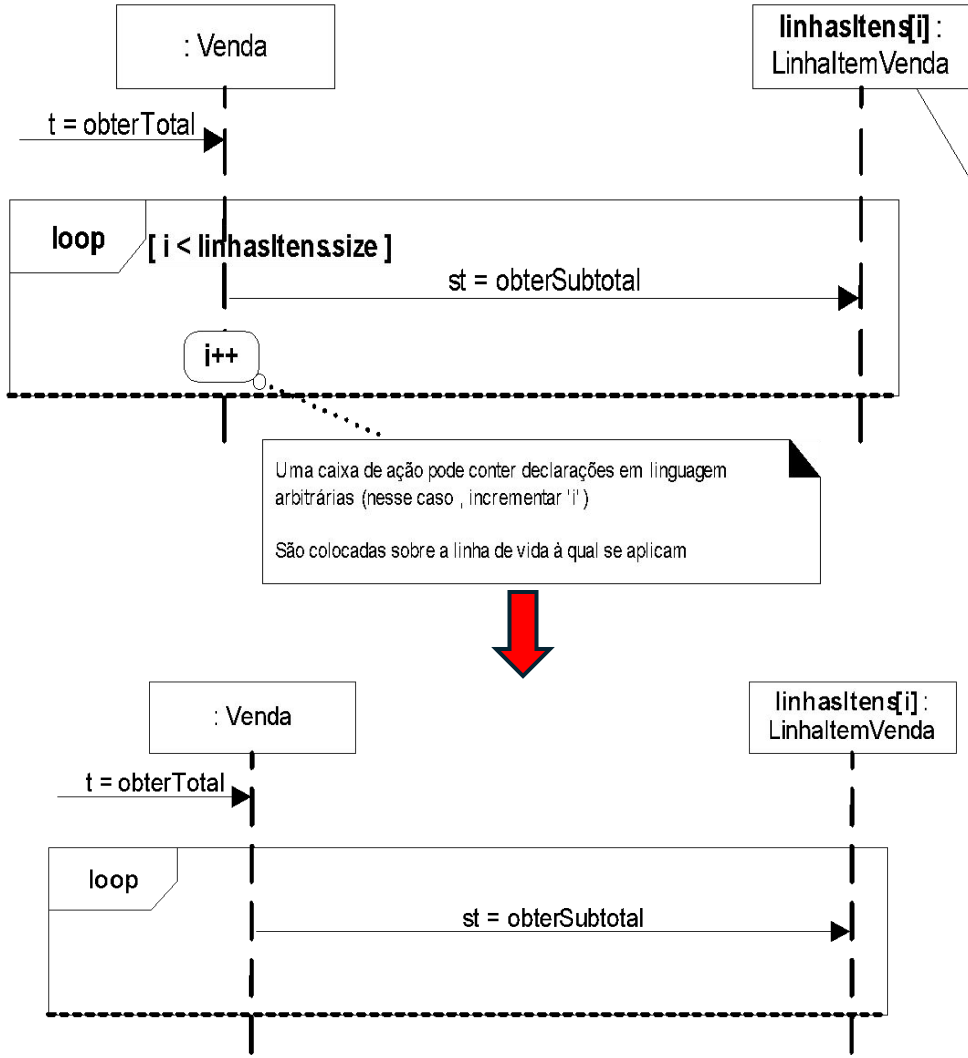
Modularização de Interações: Fluxo de Controle – Tratamento de Exceções

- O **fluxo de controle com tratamento de exceção** representa situações em que uma **interação normal é interrompida** devido a um **erro ou condição excepcional**.
- É utilizado para **modelar comportamentos alternativos** que devem ocorrer **quando uma exceção é lançada**.
- É representado por uma **moldura com o operador break**.
- O conteúdo dentro da moldura indica **o que deve ser executado** quando a condição de exceção ocorre.
- Após o tratamento, o **fluxo normal da interação é encerrado** ou redirecionado, conforme definido no diagrama.

Modularização de Interações: Fluxo de Controle – Alternativas



Modularização de Interações: Fluxo de Controle – Alternativas

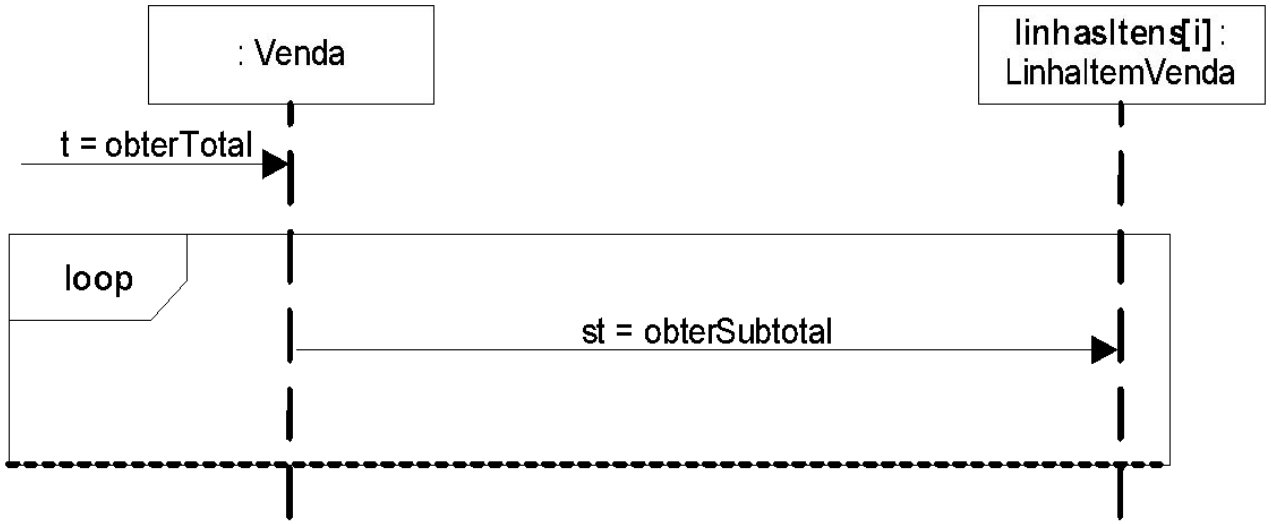



```
Public class Venda
{
    private List<LinhaItemVenda> linhasItem = new ArrayList<>();

    public Moeda obterTotal()
    {
        Moeda total = new Moeda();
        Moeda subtotal = null;

        for (LinhaItemVenda linhaItem : linhasItem)
        {
            subtotal = linhaItem.obterSubtotal();
            total.add(subtotal);
        }
        return total;
    }
}
```

Modularização de Interações: Fluxo de Controle – Alternativas



Modularização de Interações: Fluxo de Controle – Alternativas

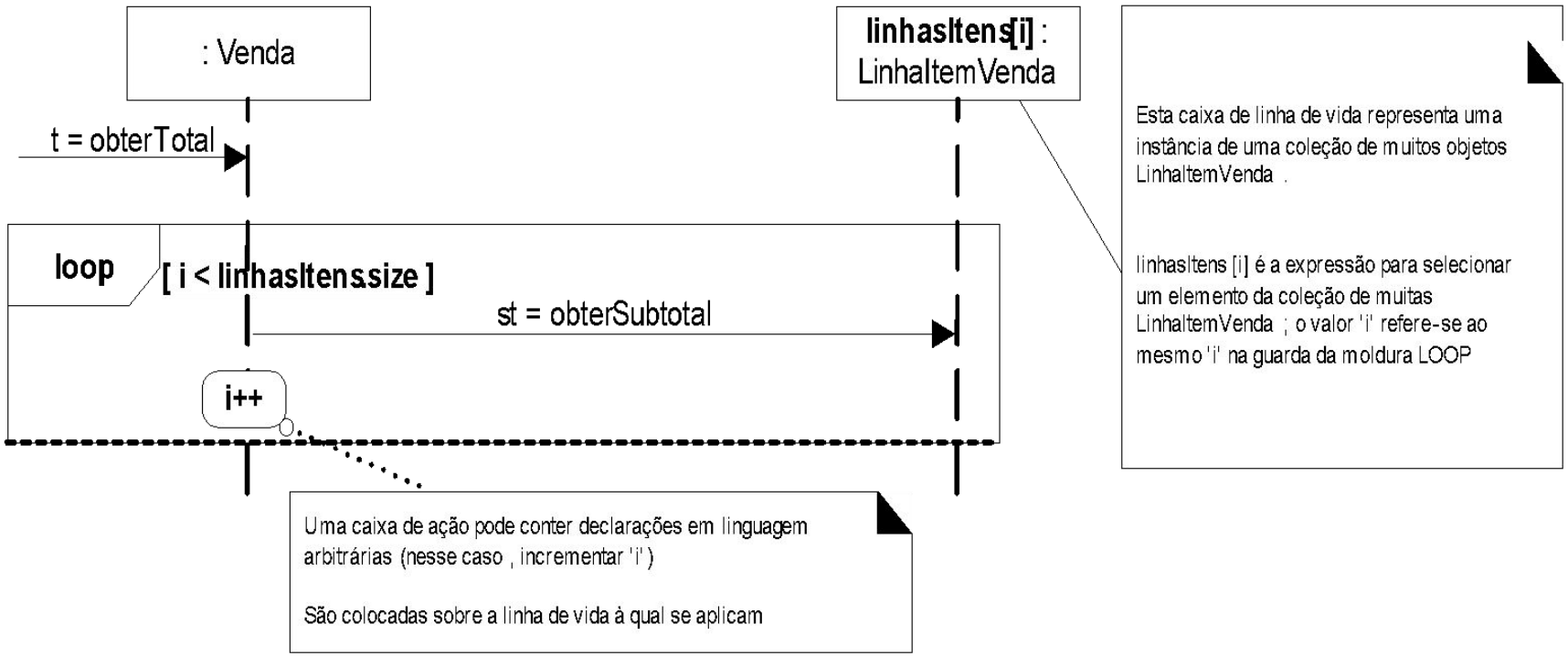


Diagrama de Sequência – Ex 3

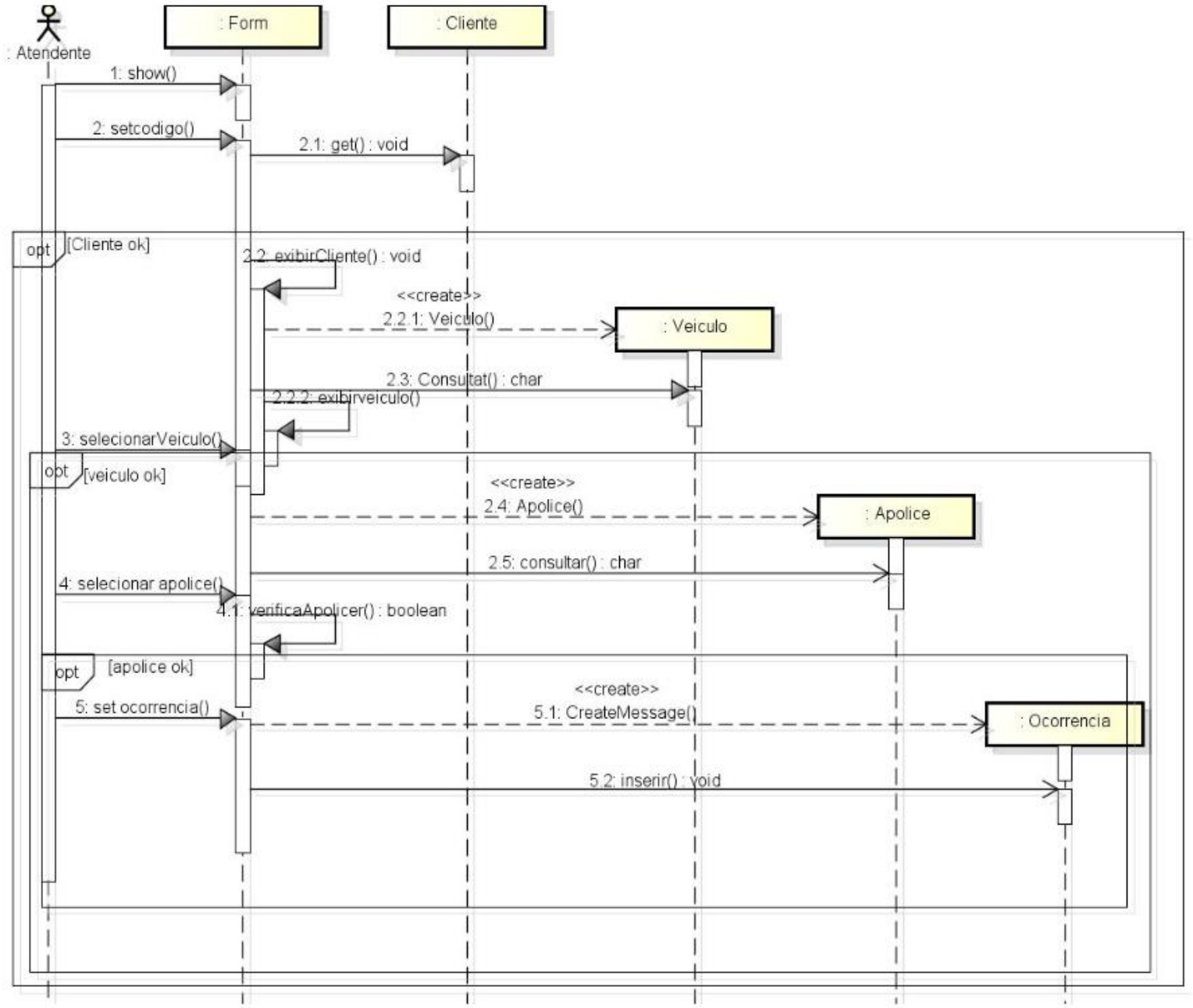


Diagrama de Sequência : Exemplo 3 - projeto detalhado

- Seja o caso de uso “Usuário Consulta Contato” do Sistema AGENDA
- Roteiro:
    1. Identificamos o caso de uso no modelo da análise (Modelo Lógico – Diagrama de Classes)

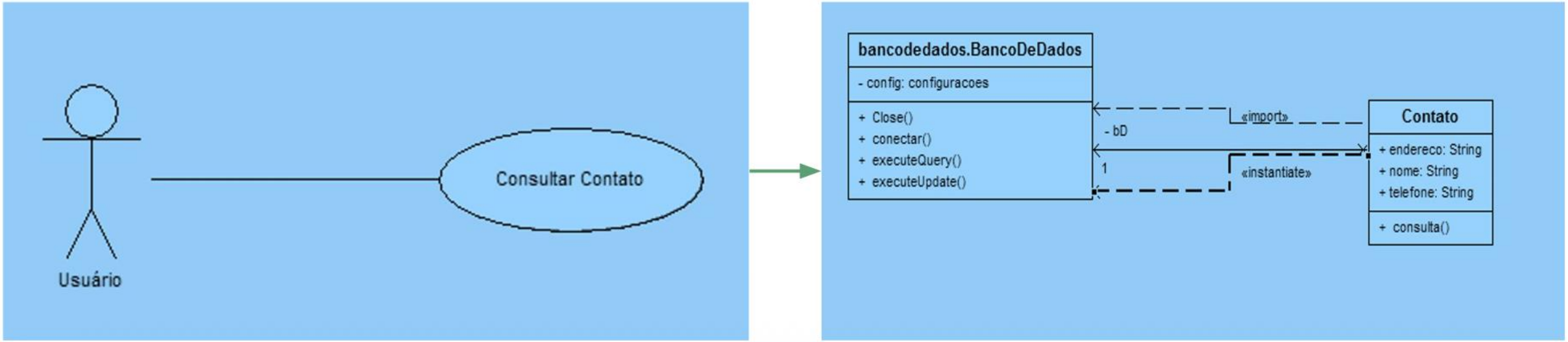


Diagrama de Sequência : Exemplo 3 - projeto detalhado

1 - Identificamos o caso de uso no modelo da análise (Modelo Lógico – Diagrama de Classes)

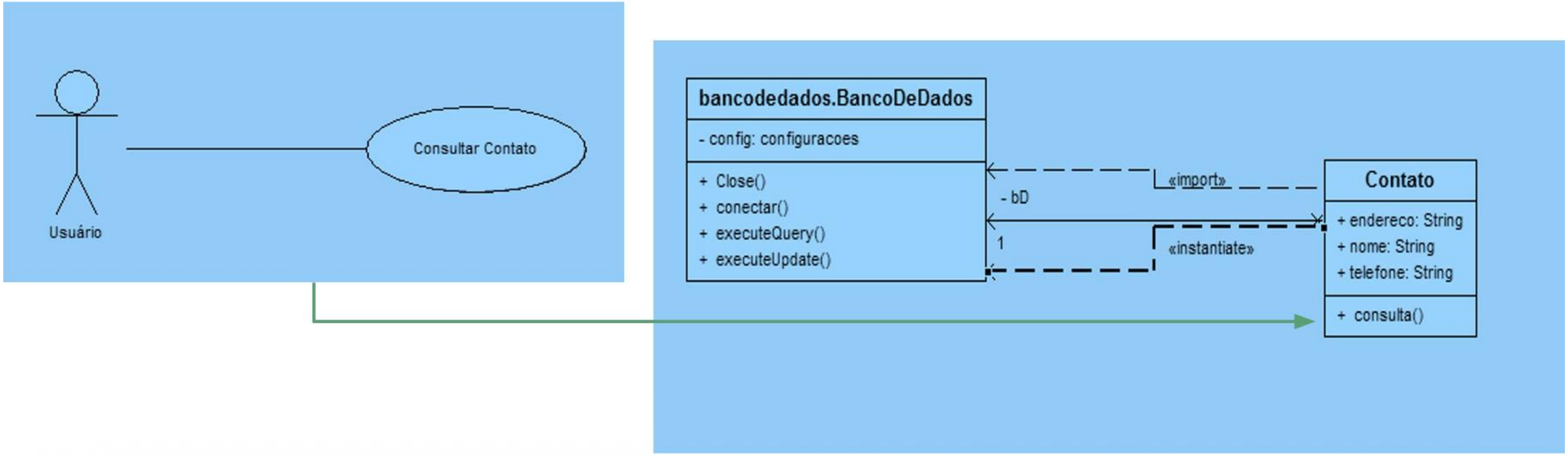


Diagrama de Sequência : Exemplo 3 - projeto detalhado

2- Criamos o Protótipo de Tela

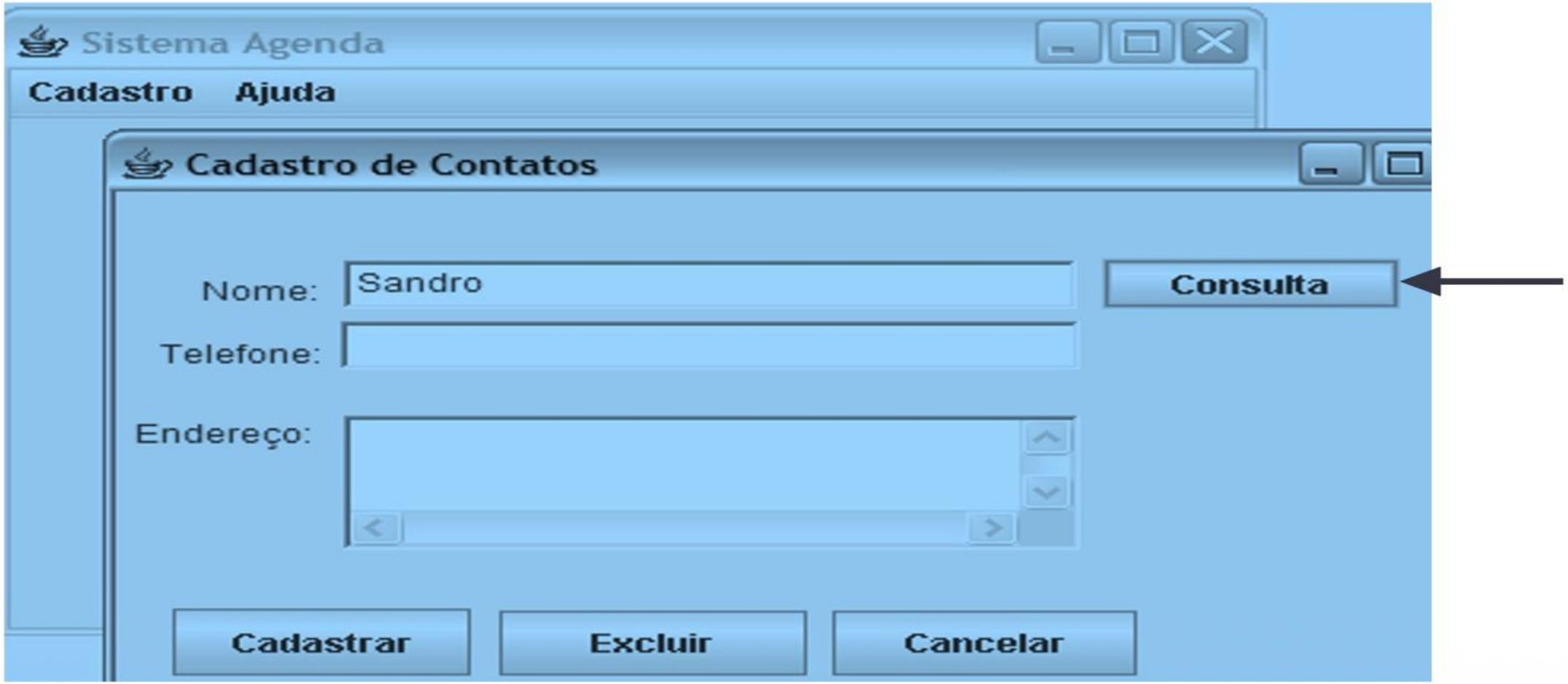


Diagrama de Sequência : Exemplo 3 - projeto detalhado

3 - Em seguida, geramos o Diagrama de sequência

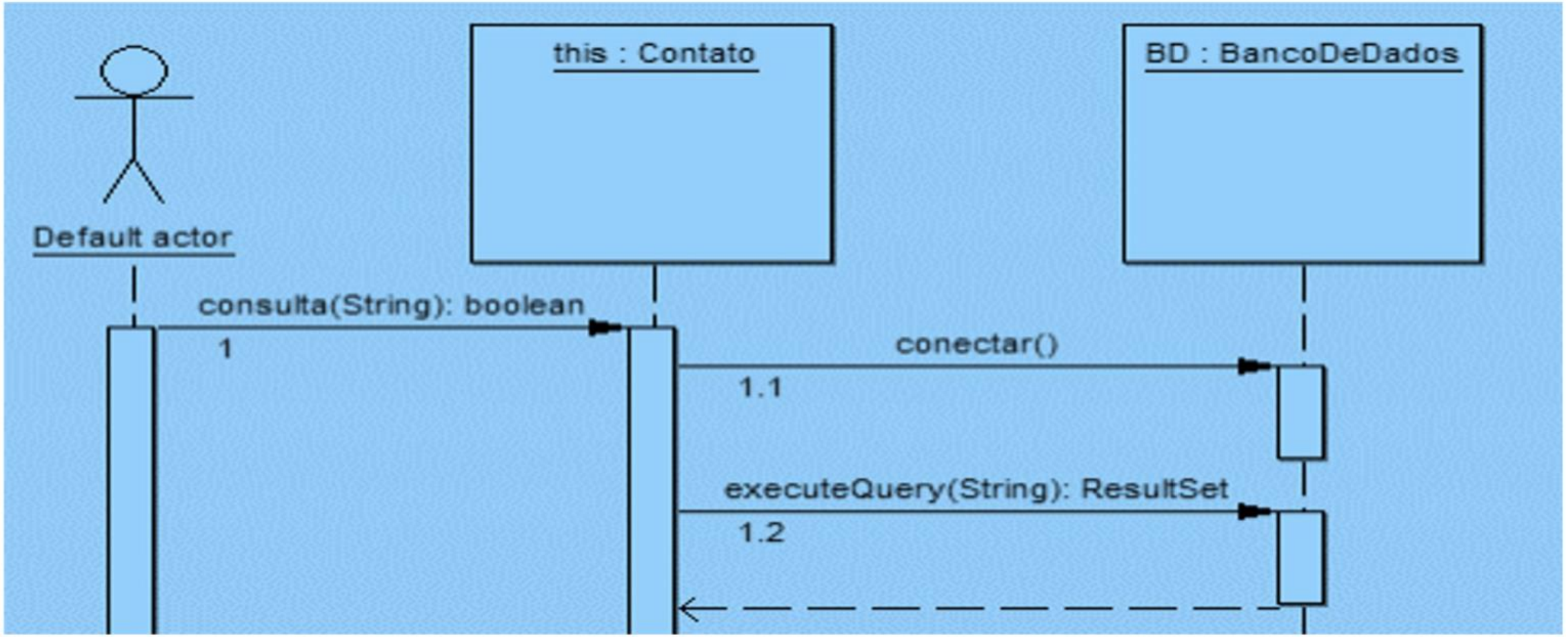


Diagrama de Sequência : Exemplo 3 - projeto detalhado

4 - Finalmente, a codificação

```
package modelo;
import bancodedados.BancoDeDados;
import java.sql.ResultSet;

public class Contato {

    public String nome="";
    public String telefone="";
    public String endereco="";
    private BancoDeDados BD = new BancoDeDados();
    private boolean retorno=false;

public boolean consulta(String N) // Método para consulta ao Banco de dados
{   try{
    ResultSet RSdados;
    String frase= "select * from contato";
    BD.conectar();
    RSdados = BD.executeQuery(frase);

    while (RSdados.next())
        {if (RSdados.getString("nome").matches(N))
           {this.telefone = RSdados.getString("telefone");
            this.endereco = RSdados.getString("endereco");
            retorno = true;}
        }
    }
    catch(Exception e){ System.out.println("Erro em  -> " + e);}
    return retorno;}
}
```

Diagrama de Sequência – Ex 3