



# DIAGRAMA DE SÊQUENCIA

Programação Orientada a Objetos (POO)



# TÓPICOS ASSOCIADOS

- O que é o Diagrama de Sequência?
- Vantagens e desafios do uso do diagrama de sequência.
- Elementos de um Diagrama de Sequência;
- Exemplo de aplicação.
- Código do exemplo.

# O DIAGRAMA DE SEQUENCIA

## O que é?

O diagrama de sequência faz parte da família dos diagramas de comportamento padronizados pela UML. Permite visualizar como as ações se desenrolam durante a execução de um determinado processo ou cenário.

## Para que serve?

O diagrama de sequência mostra as mensagens trocadas entre os participantes, ajudando a entender o fluxo de comunicação, a identificar problemas de design e a documentar casos de uso.

# Vantagens do uso de Diagramas de Sequência

## Visualização Clara

Representam o fluxo de interações de forma visual, facilitando a compreensão e análise do processo.

## Comunicação Eficaz

Permitem a comunicação clara e concisa do processo, mesmo para pessoas sem conhecimento técnico específico.

## Identificação de Erros Rapidamente

Ajudam a identificar problemas e erros potenciais no processo, antes da implementação ou execução do processo.



# Limitações e Desafios dos Diagramas de Sequência

## Complexidade

Podem se tornar complexos e difíceis de entender, principalmente em sistemas com muitos objetos e interações.

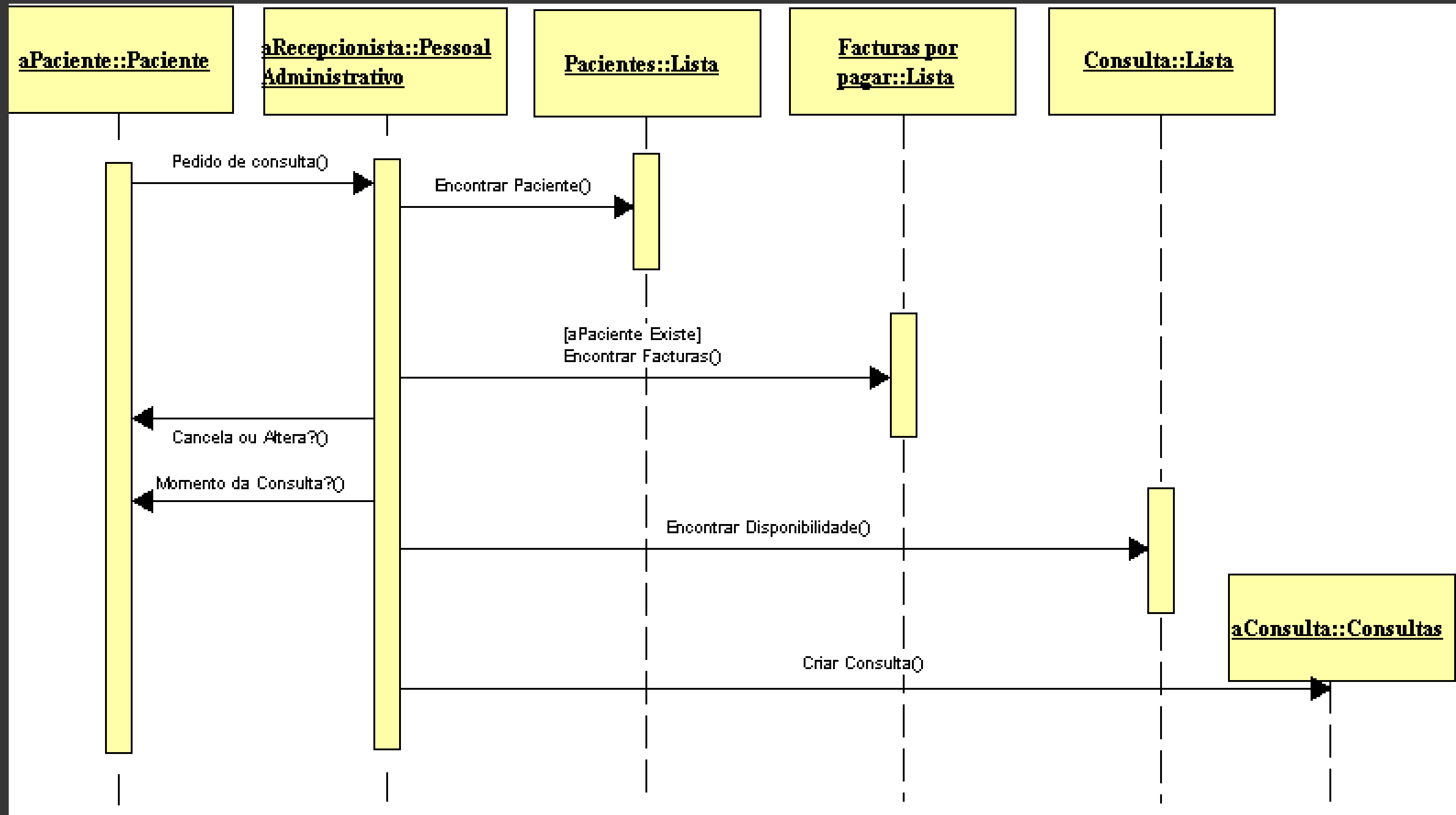
## Tempo de Criação

Criar diagramas de sequencia detalhados pode ser demorados, especialmente para processos complexos.

## Nível de Detalhes

É preciso encontrar o equilíbrio entre o nível de detalhes necessários e a clareza do diagrama







# ELEMENTOS DE UM DIAGRAMA DE SEQUENCIA






## ✦ Atores

- Representam entidades externas ao sistema que interagem com ele.
- Como usuários, sistemas externos ou dispositivos.
- Um ator é representado como um boneco-palito ou um ícone que simboliza um agente externo.
- Um cliente usando um sistema de reservas online.

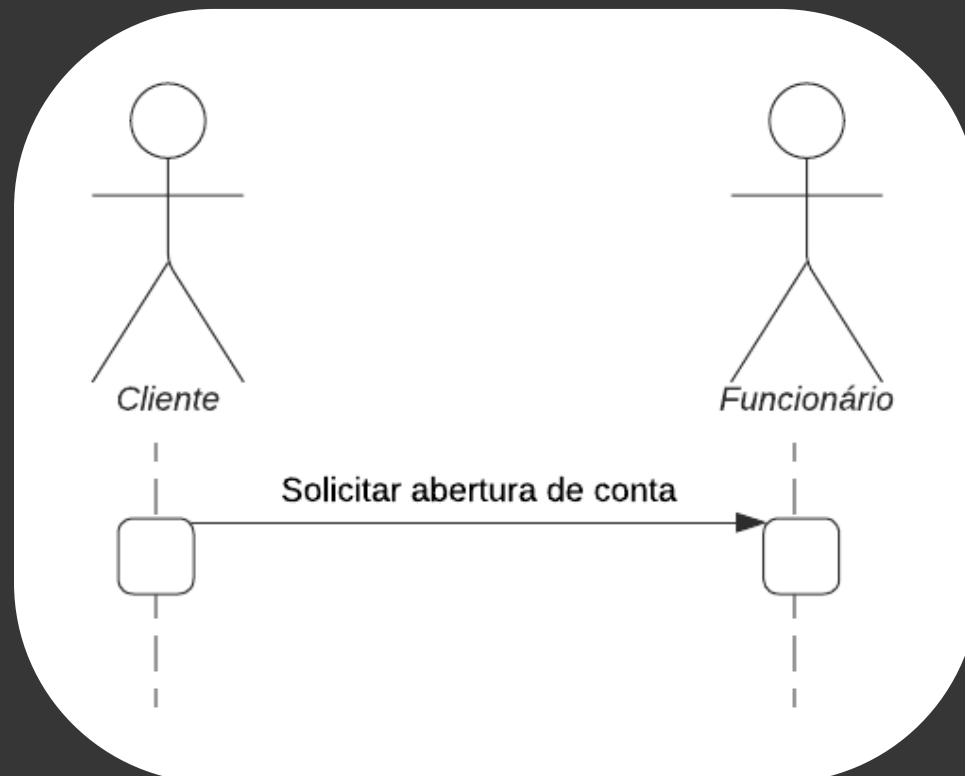
## ✦ Lifelines

- Representam a linha de vida de um objeto ou ator no sistema durante uma interação.
  - Indicam o tempo de existência e atividade do participante.
  - Uma lifeline é uma linha vertical que começa em um retângulo (representando o participante).
- 

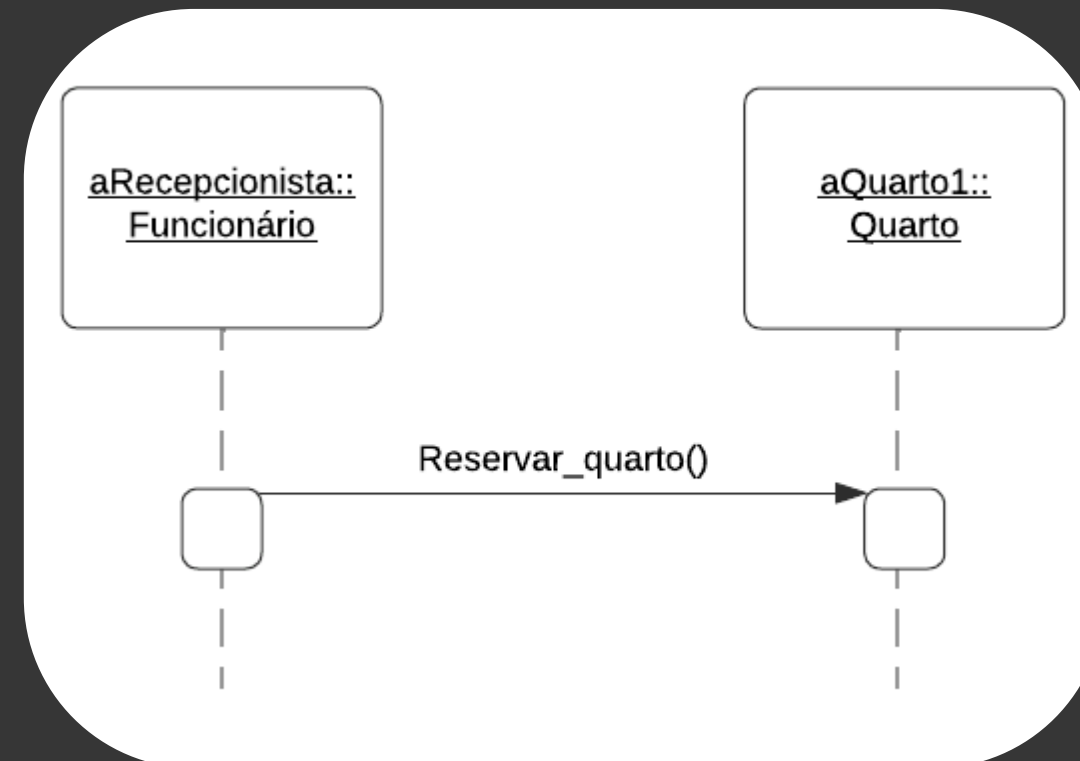


# MENSAGENS OU ESTÍMULOS

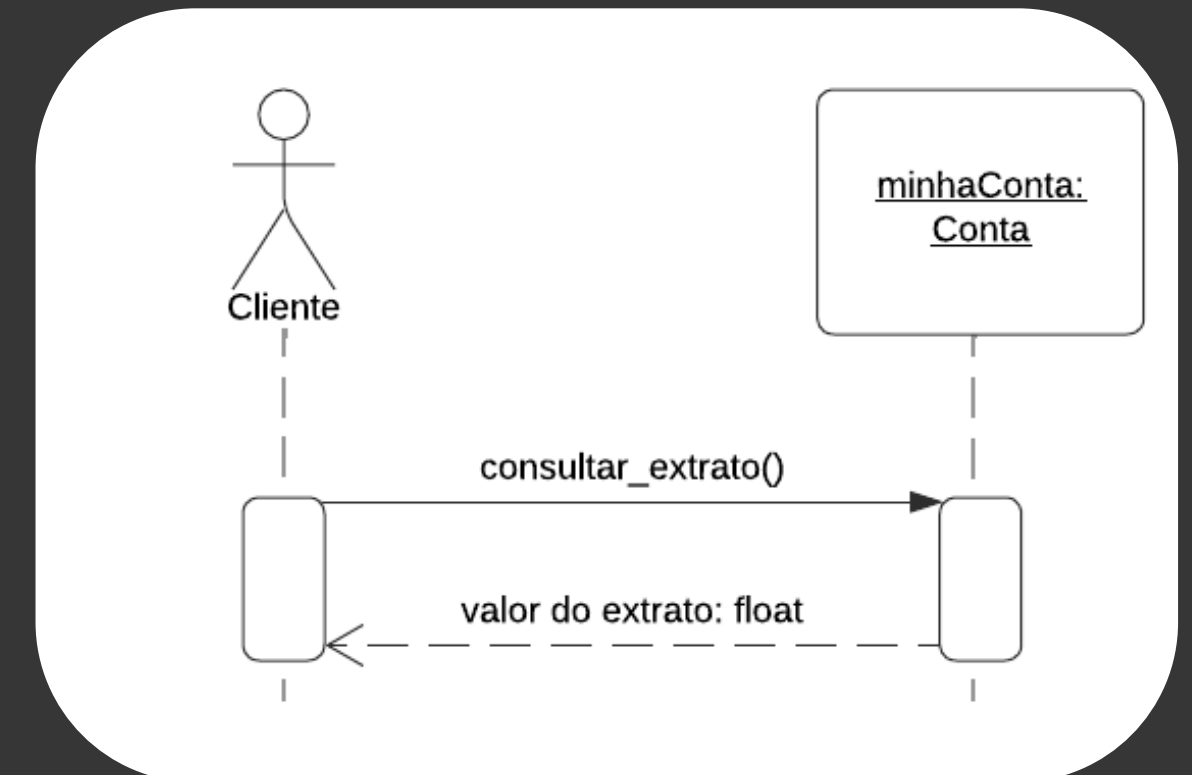
## Mensagens entre Atores



## Mensagens entre Lifelines

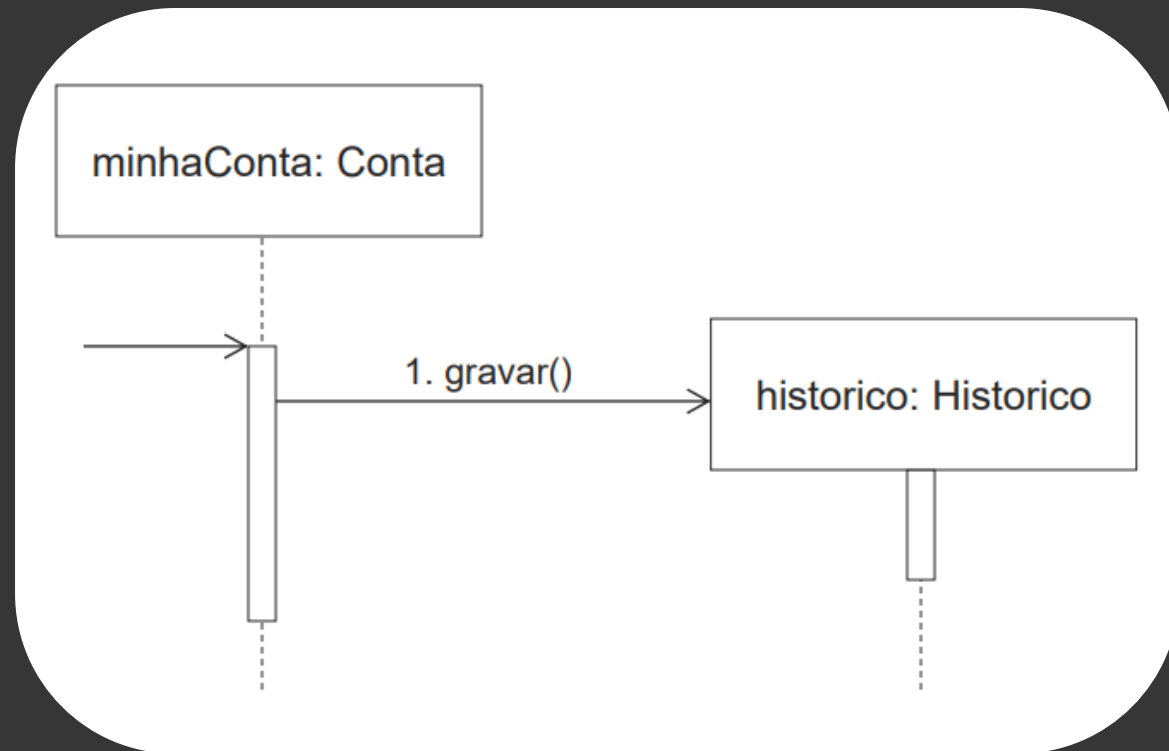


## Mensagens de Retorno

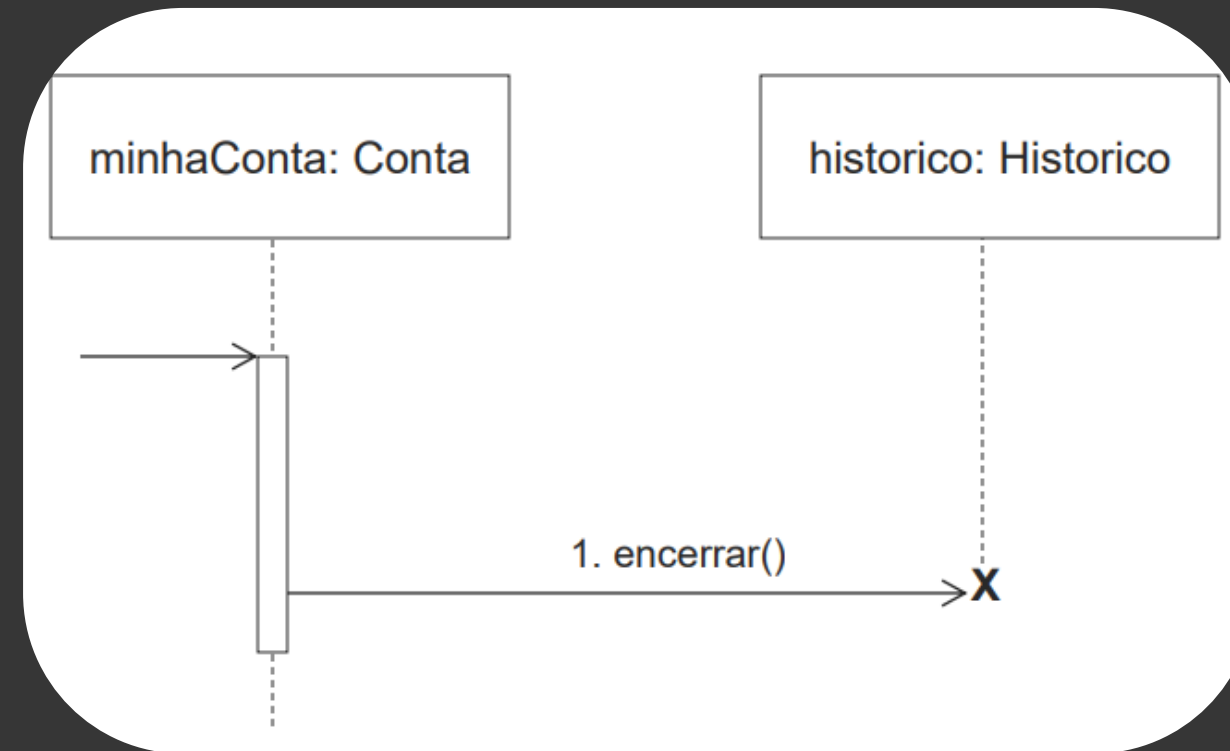


# MENSAGENS OU ESTÍMULOS

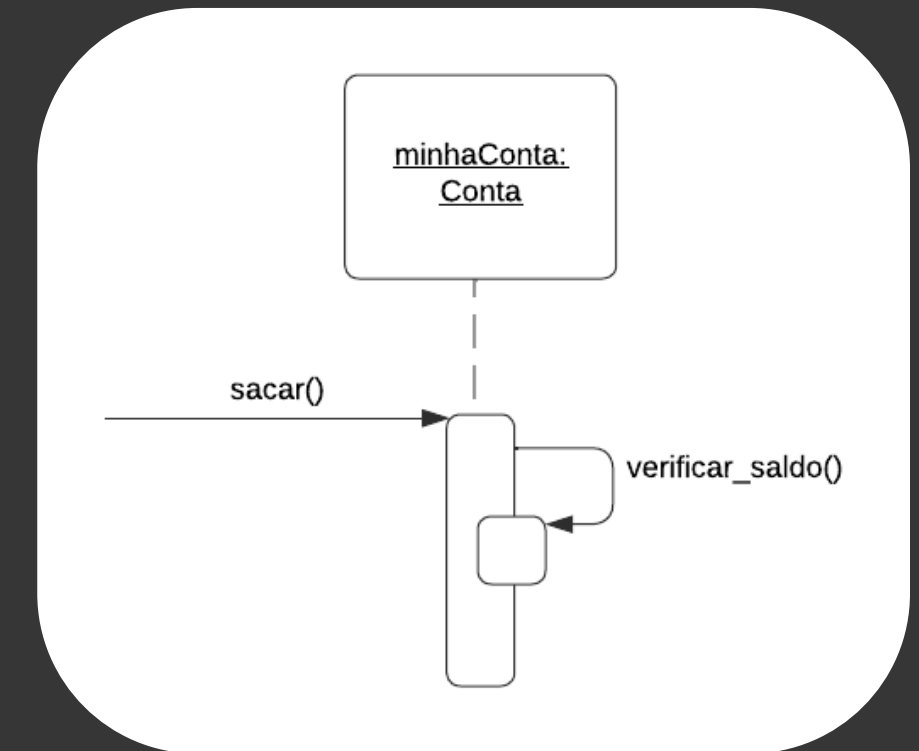
## Mensagens Construtoras



## Mensagens Destrutoras

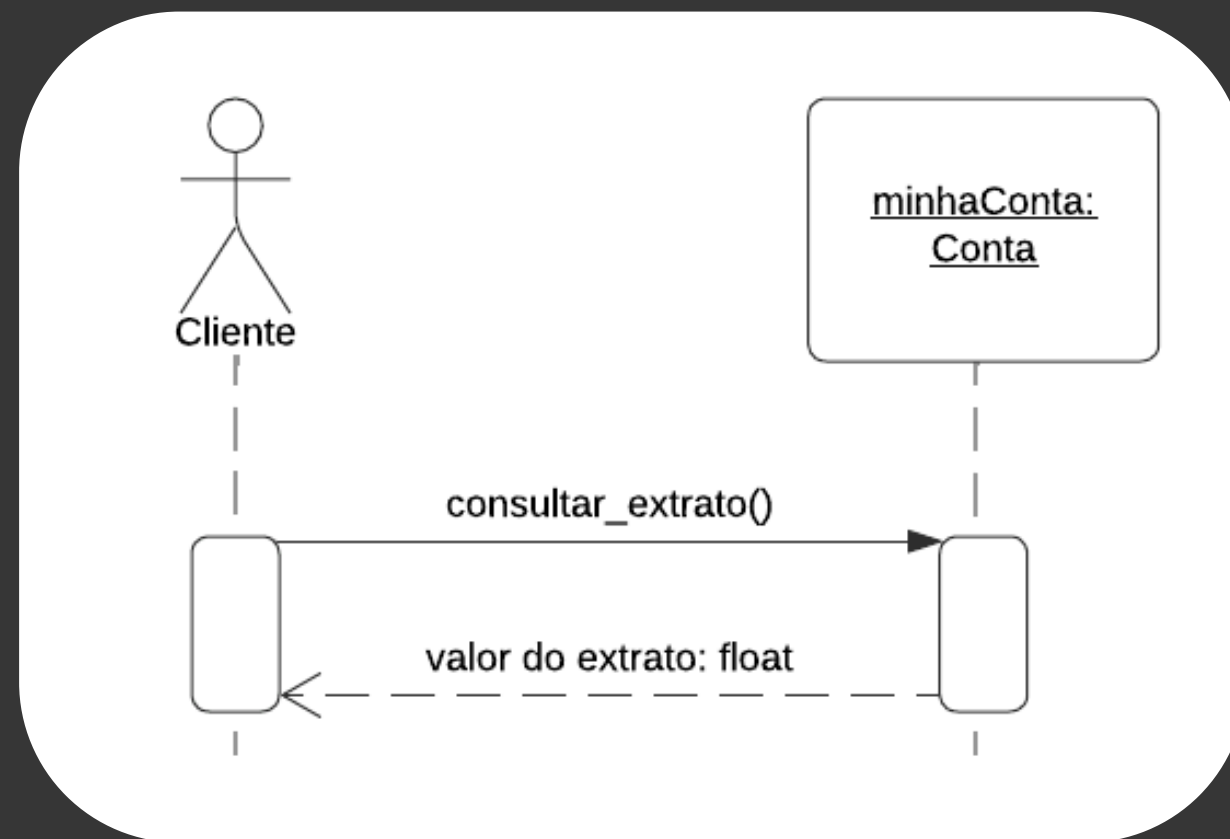


## Autochamadas

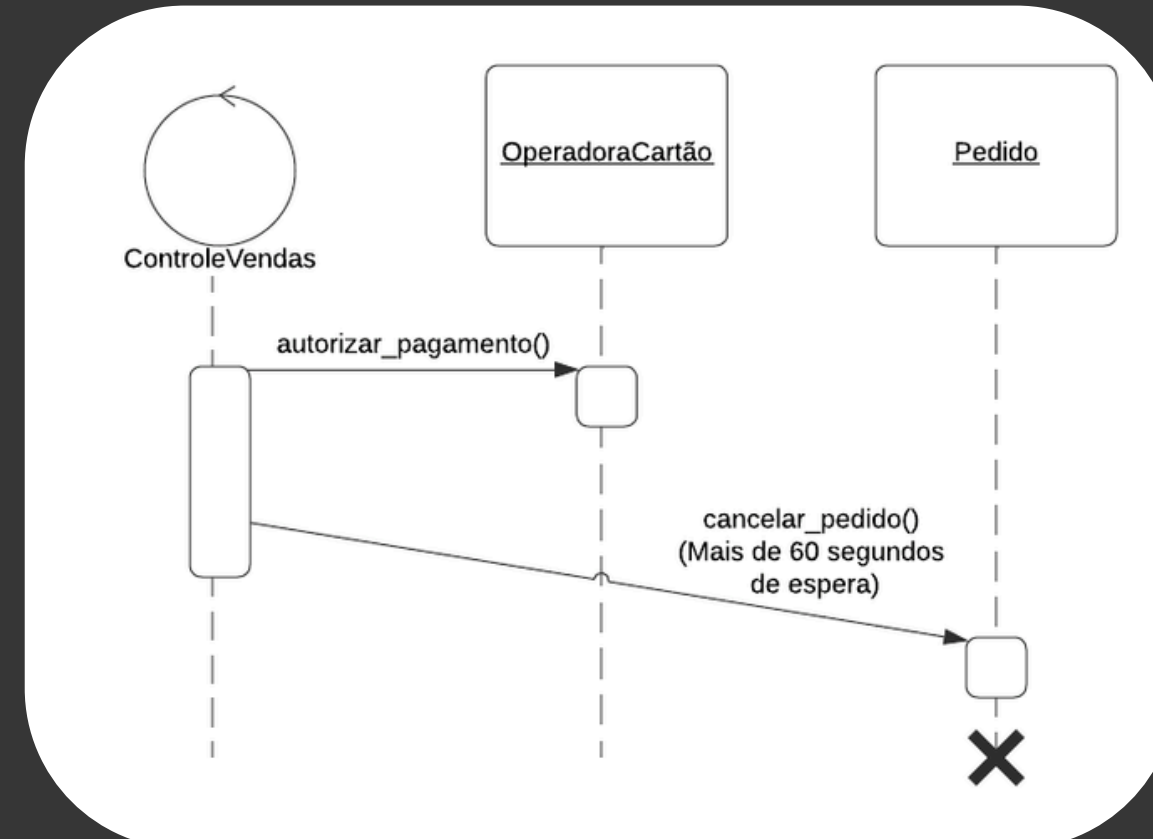


# MENSAGENS OU ESTÍMULOS

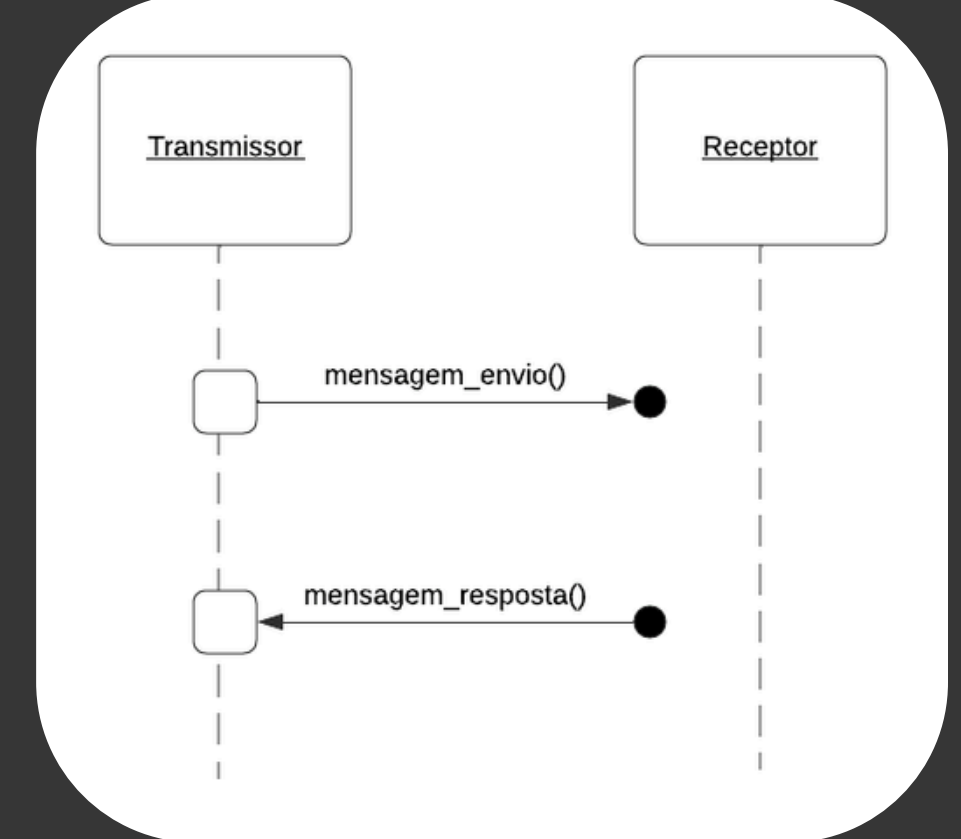
## Mensagens Assíncronas



## Restrição de Duração



## Mensagens Perdidas e Mensagens Encontradas



## PORTAS

---

Uma porta representa um ponto de comunicação, uma interface entre o ambiente externo e as partes internas de uma classe .

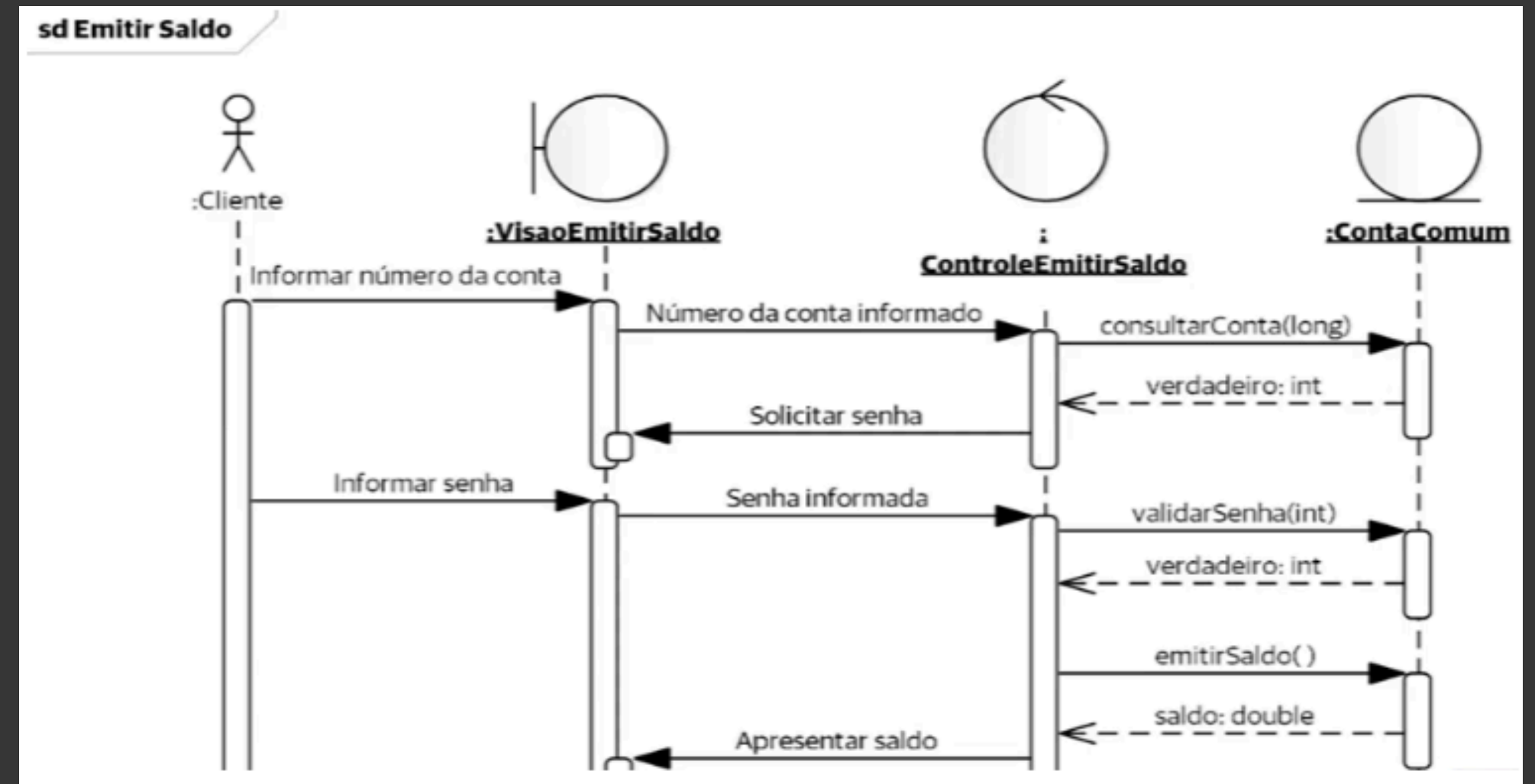
## PORTÕES

---

Um portão é uma interface entre fragmentos, um ponto de conexão para relacionar uma mensagem fora de um uso.

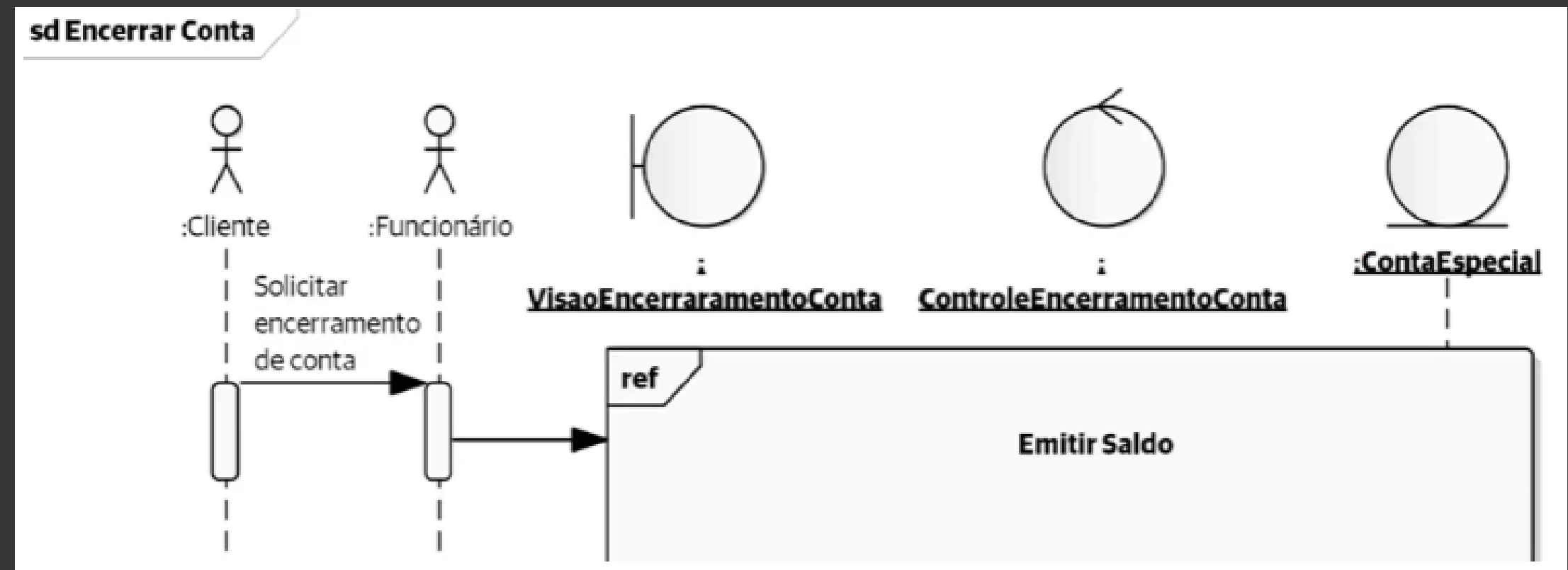
# FRAGMENTOS DE INTERAÇÃO

- Um fragmento de interação é parte de uma interação
- Cada fragmento de interação também é considerado uma interação
- Um fragmento de interação é representado como um retângulo que envolve toda a interação.
- Um diagrama de sequência completo pode ser considerado um fragmento de interação se for representado em outro diagrama.



# USOS DE INTERAÇÃO

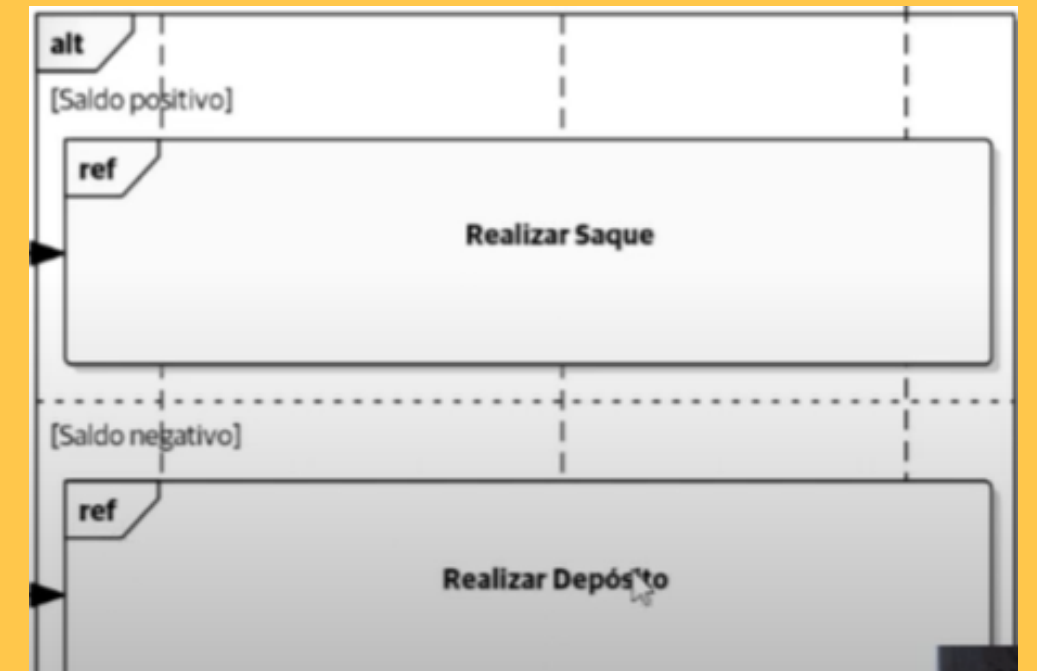
A principal vantagem é a possibilidade de poder referenciar os fragmentos utilizando Ref(abreviatura de Referred/Referido) seguido do nome do Diagrama.



# FRAGMENTOS COMBINADOS E OPERADORES DE INTERAÇÃO

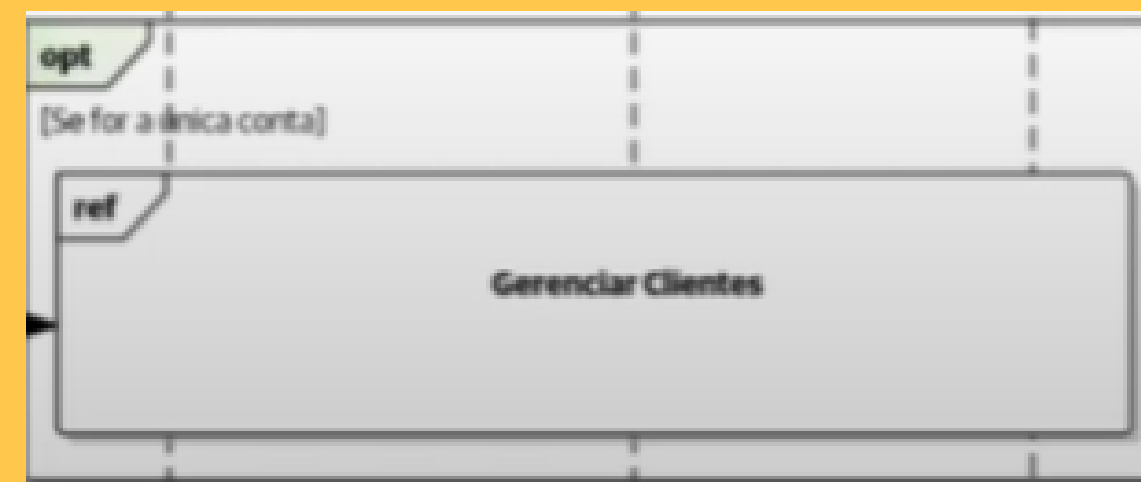
## Alt

Define que o fragmento combinado representa uma escolha entre dois ou mais comportamentos.



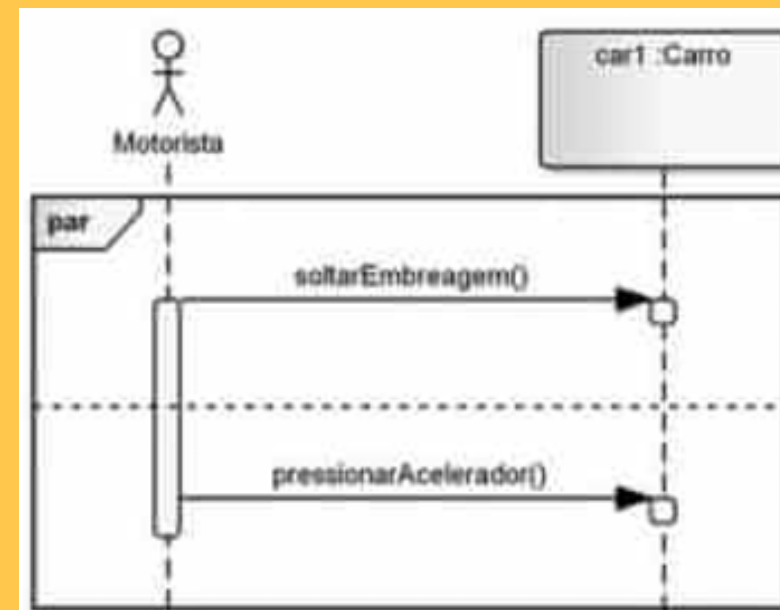
## Opt

Determina que o fragmento combinado representa uma escolha de comportamento em que esse comportamento será ou não executado.



## Par

Determina que o fragmento combinado representa uma execução paralela de dois ou mais comportamentos.



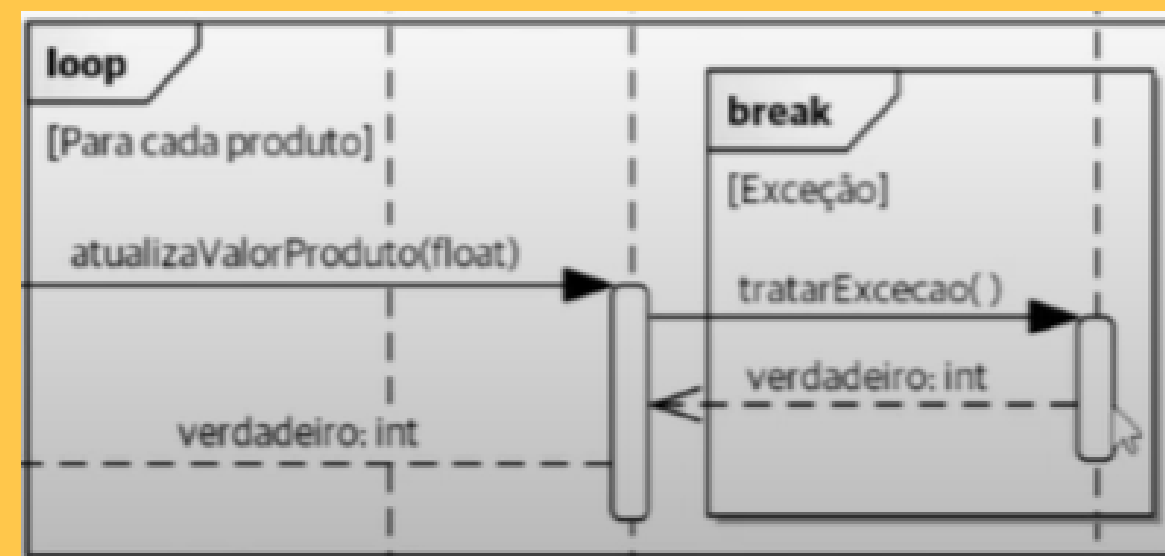
## Loop

Determina que o fragmento combinado representa um laço que poderá ser repetido diversas vezes.



## Break

Indica uma “quebra” na execução normal do processo.



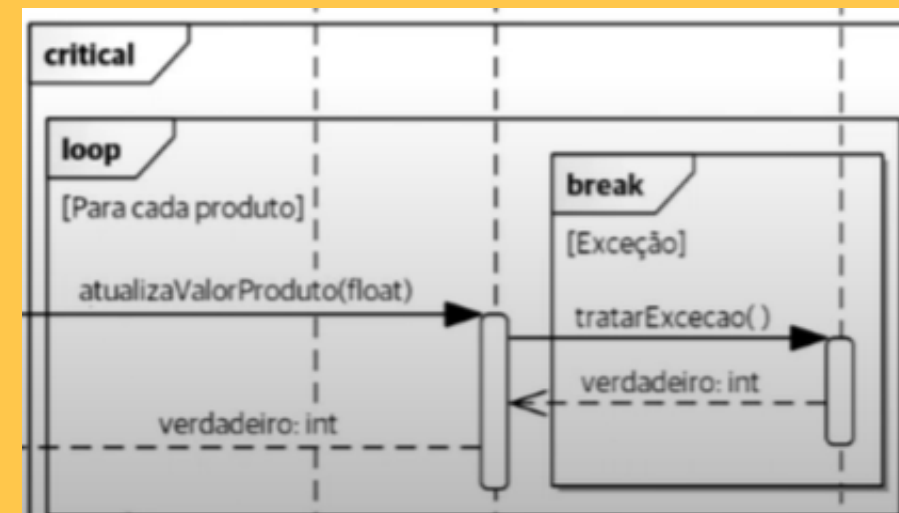
# FRAGMENTOS COMBINADOS E OPERADORES DE INTERAÇÃO



# FRAGMENTOS COMBINADOS E OPERADORES DE INTERAÇÃO

## Critical Region

Identifica uma operação atômica que não pode ser interrompida por outro processo até ser totalmente concluída.



## Seq

Identifica uma situação na qual as ocorrências de evento devem atender a certas propriedades.

## Strict

Garante que todas as mensagens no fragmento combinado sejam ordenadas do início ao fim.

## Ignore

Determina que as mensagens contidas no fragmento devem ser ignoradas.

## Consider

Determina que as mensagens devem obrigatoriamente ser consideradas e que todas as outras não contidas no fragmento devem ser automaticamente desconsideradas.

## Neg

Representa eventos considerados inválidos, que não devem ocorrer.

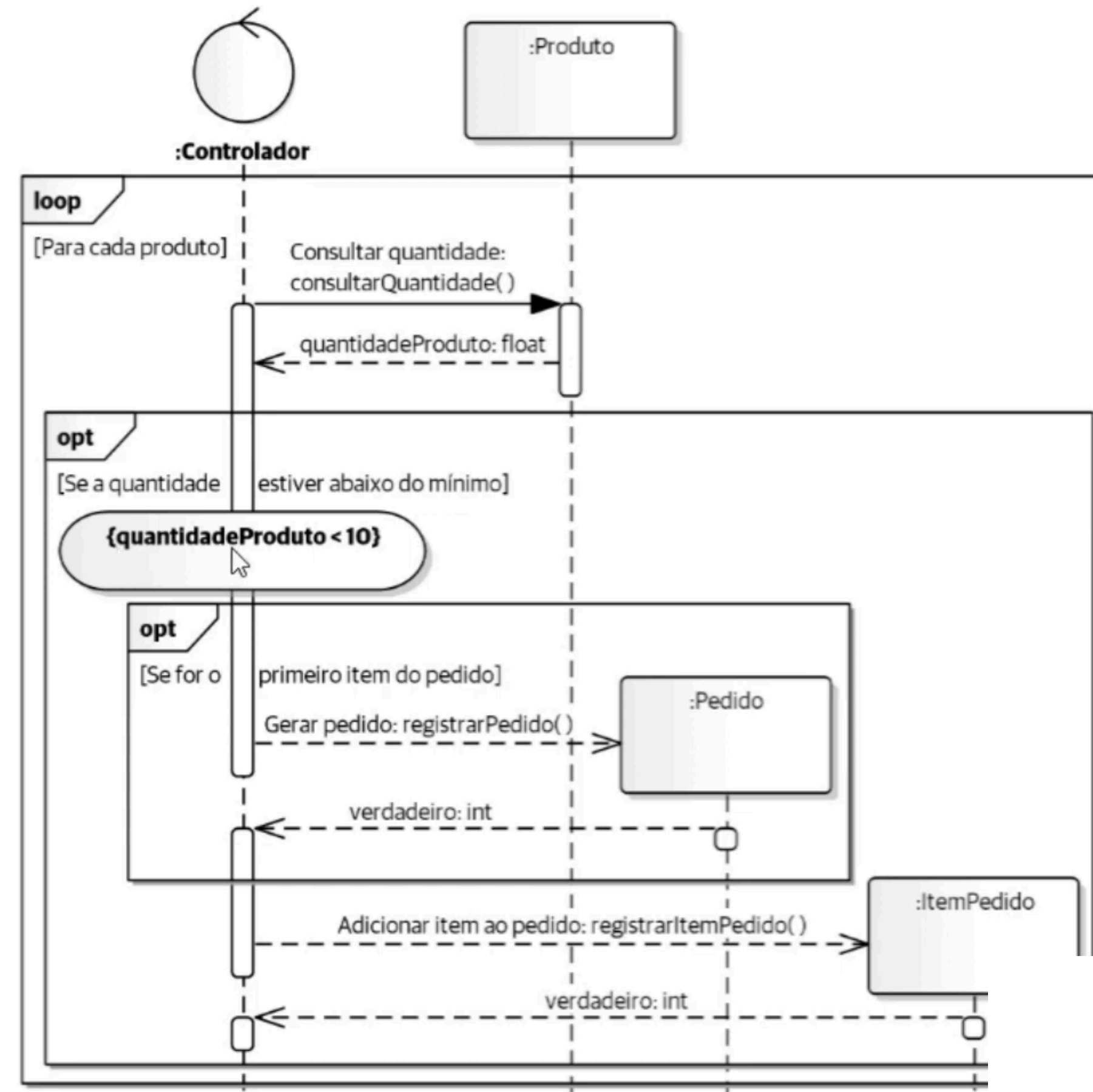
## Assertion

Esse operador de interação é o oposto do anterior, representando eventos considerados válidos

# FRAGMENTOS COMBINADOS E OPERADORES DE INTERAÇÃO

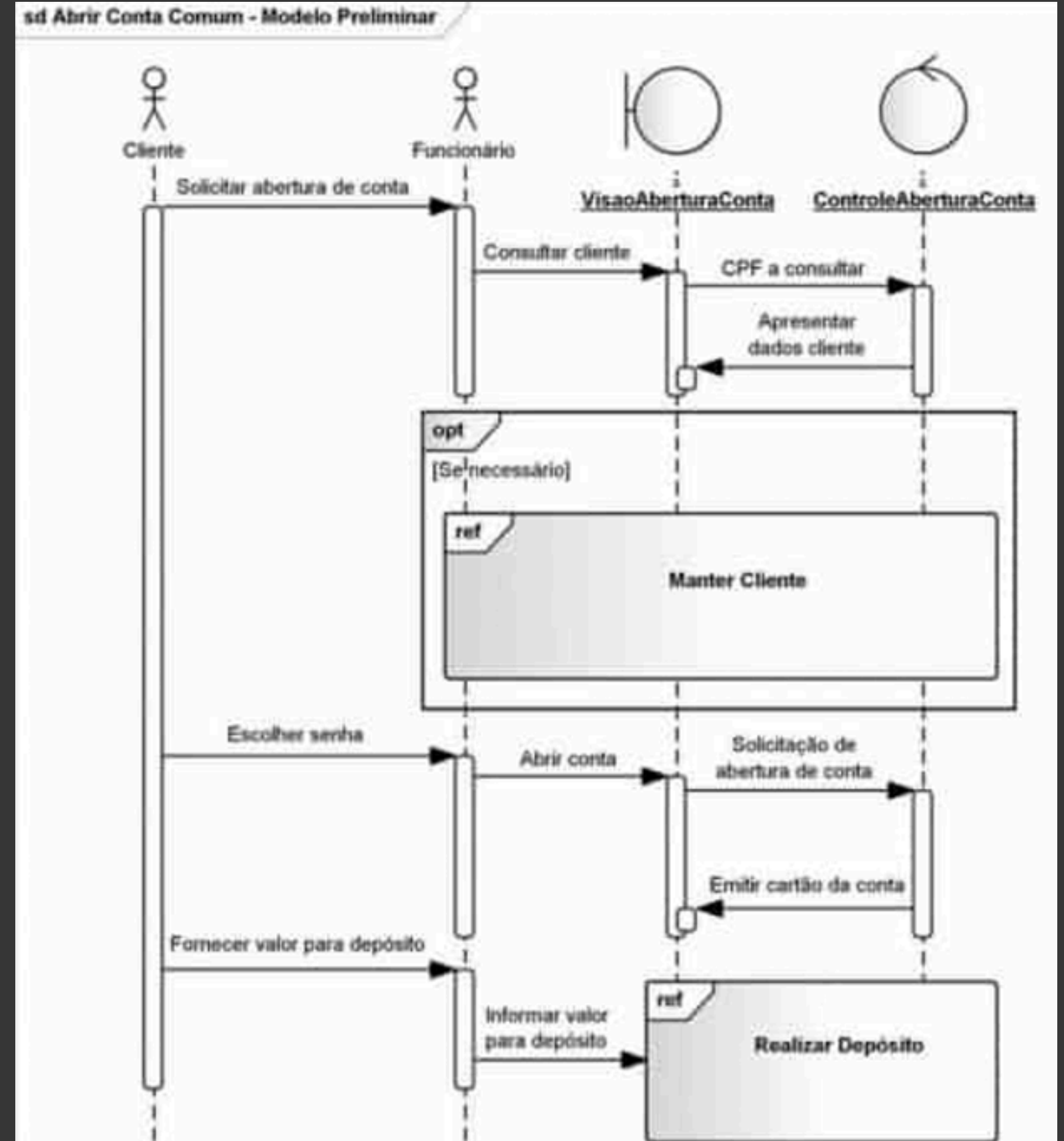
# Invariante de Estado (StateInvariant)

sd Gerar Pedido

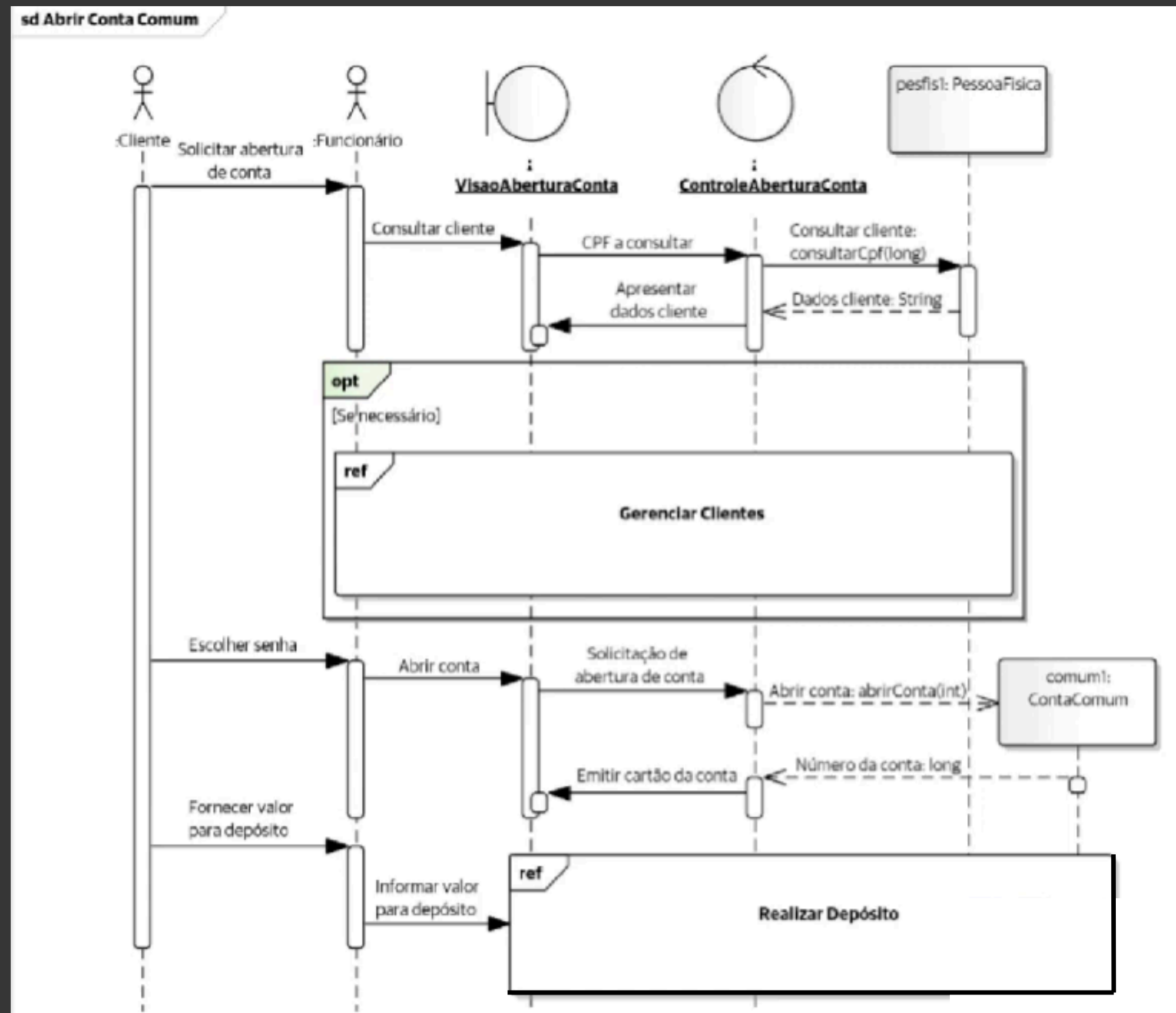


# EXEMPLOS DE DIAGRAMAS DE SEQUÊNCIA PARA O SISTEMA DE CONTROLE BANCÁRIO

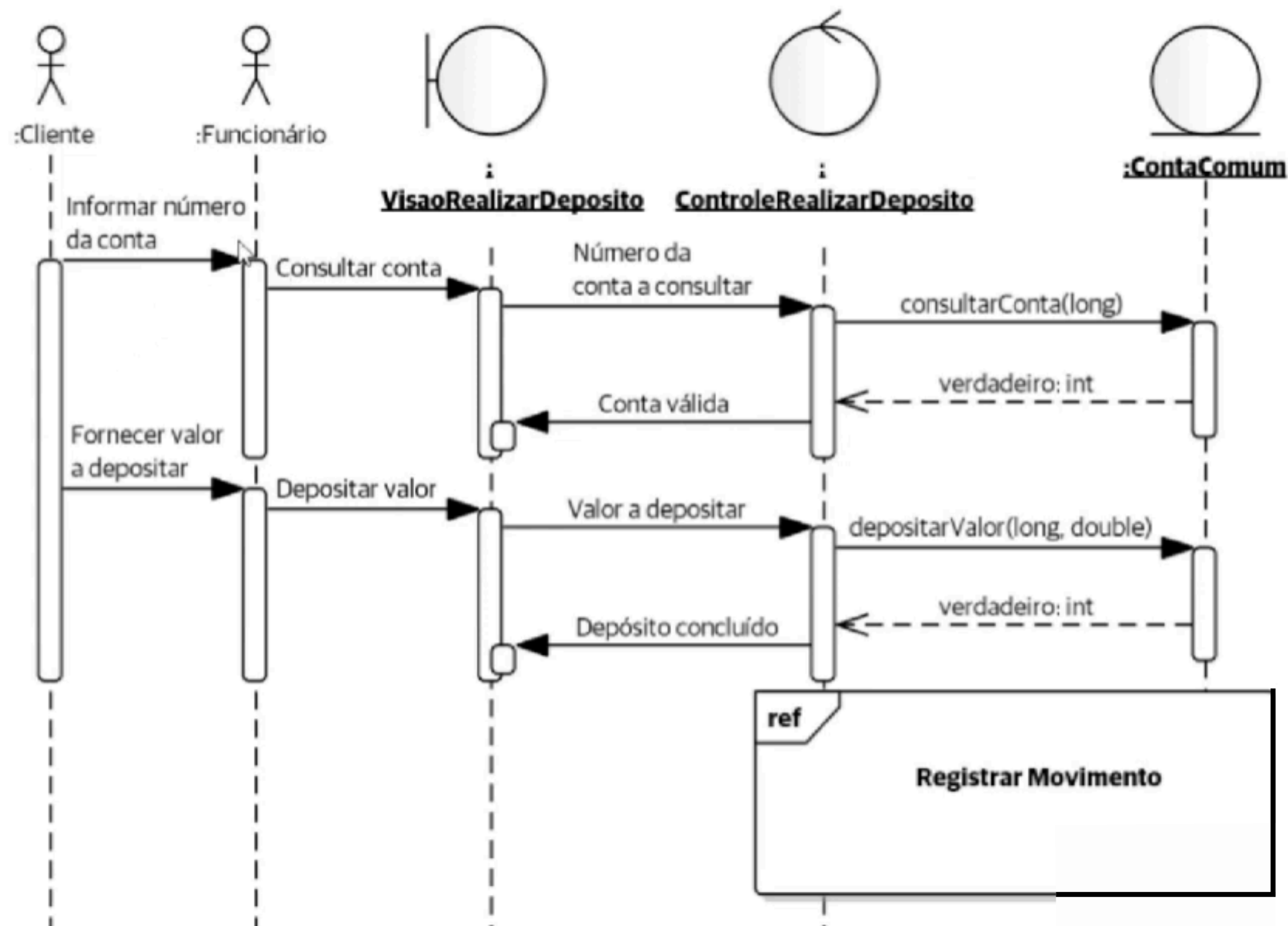
# Processo de Abertura de Conta Comum – Modelo Preliminar



# Processo de Abertura de Conta Comum – Modelo Detalhado

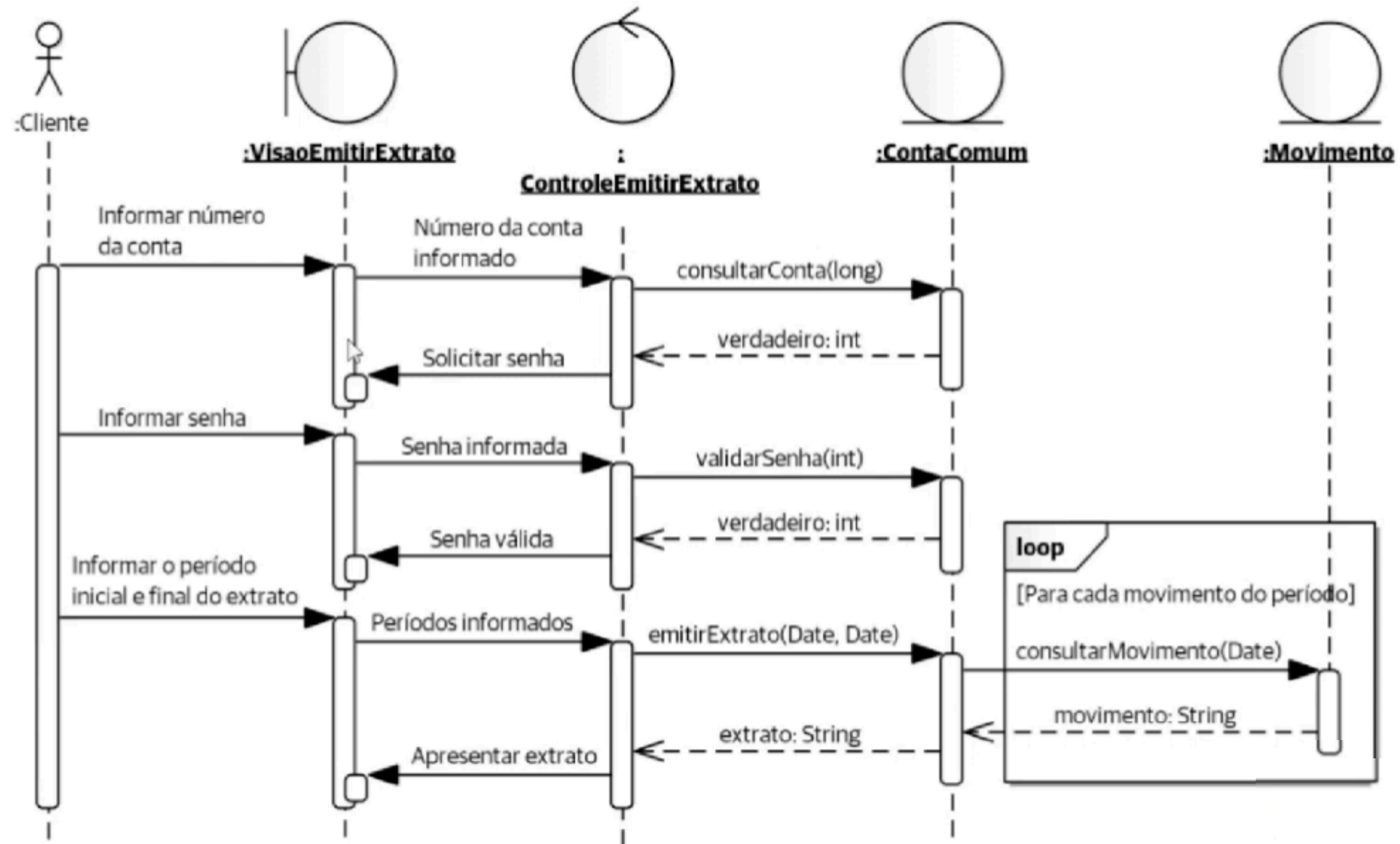


## sd Realizar Depósito



## Processo de Realizar Depósito

## sd Emitir Extrato



## Processo de Emissão de Extrato





# EXEMPLO DE APLICAÇÃO DO DIAGRAMA



Descrição do Código de Comunicação Cliente-Servidor com TCP  
Este código implementa uma simulação de conexão cliente-servidor utilizando o protocolo TCP. Ele inclui as classes e métodos necessários para gerenciar a troca de mensagens entre cliente e servidor por meio de segmentos TCP. O objetivo é criar uma comunicação confiável usando os princípios do protocolo TCP, como envio e confirmação de mensagens.

# ESTRUTURA DO CÓDIGO - CLASSES PRINCIPAIS

---

## 1 TCPSegment

- Representa um segmento TCP contendo:
- Número de sequência (seq\_num).
- Número de confirmação (ack\_num).
- Dados (data).
- Métodos:
- `__str__`: Converte o segmento em string no formato "seq\_num,ack\_num,data".
- `from_string`: Reconstrói um objeto TCPSegment a partir de uma string formatada.

## 2 TCPConnection:

- Gerencia o estado da conexão e os segmentos enviados/recebidos.
- Estados incluem:
- `CLOSED`: Conexão inativa.
- `SYN_SENT`: Solicitação de conexão enviada.

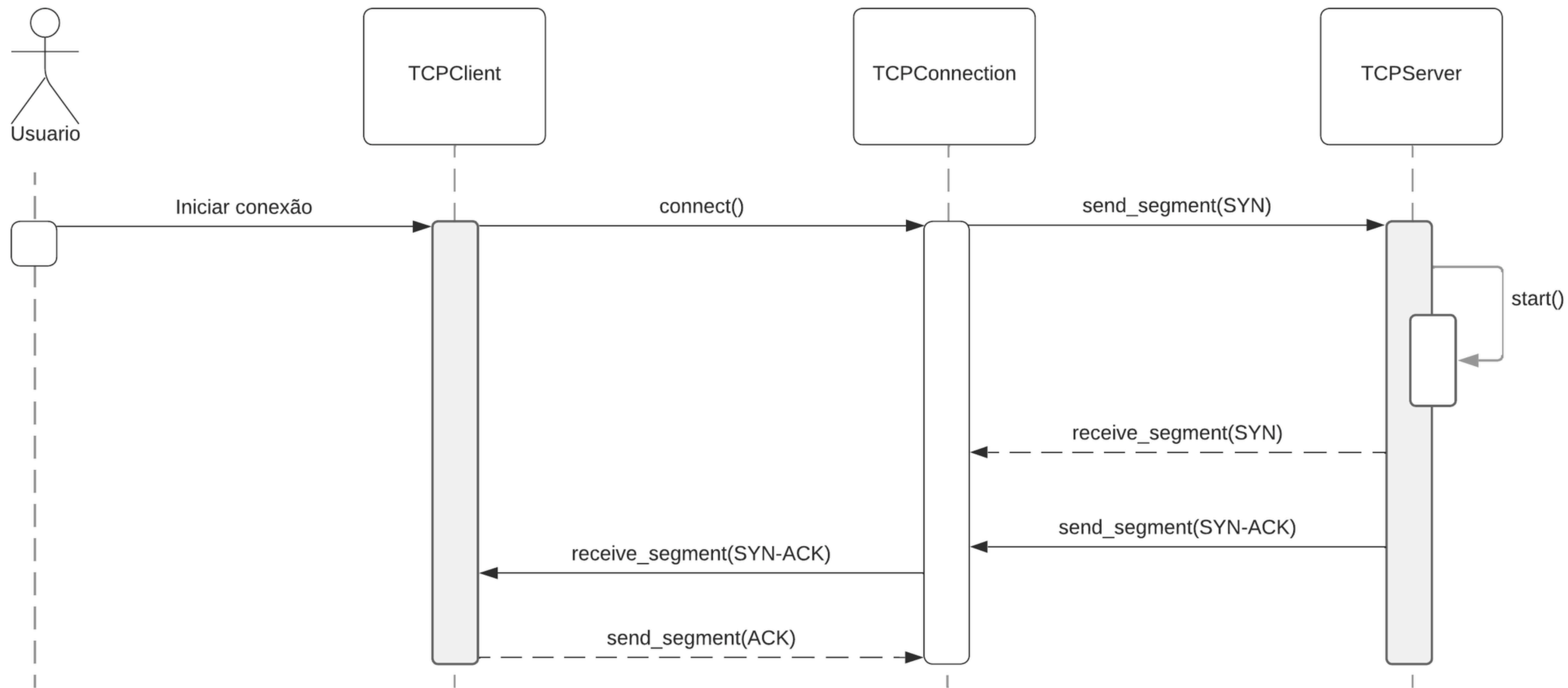
## 3 TCPClient

- Representa o cliente na comunicação TCP.
- Métodos principais:
- `connect`: Conecta ao servidor.
- `send_segment`: Envia segmentos TCP.
- `send_message`: Envia mensagens formatadas em segmentos.
- `receive_message`: Recebe respostas do servidor.
- `close_connection`: Fecha a conexão.

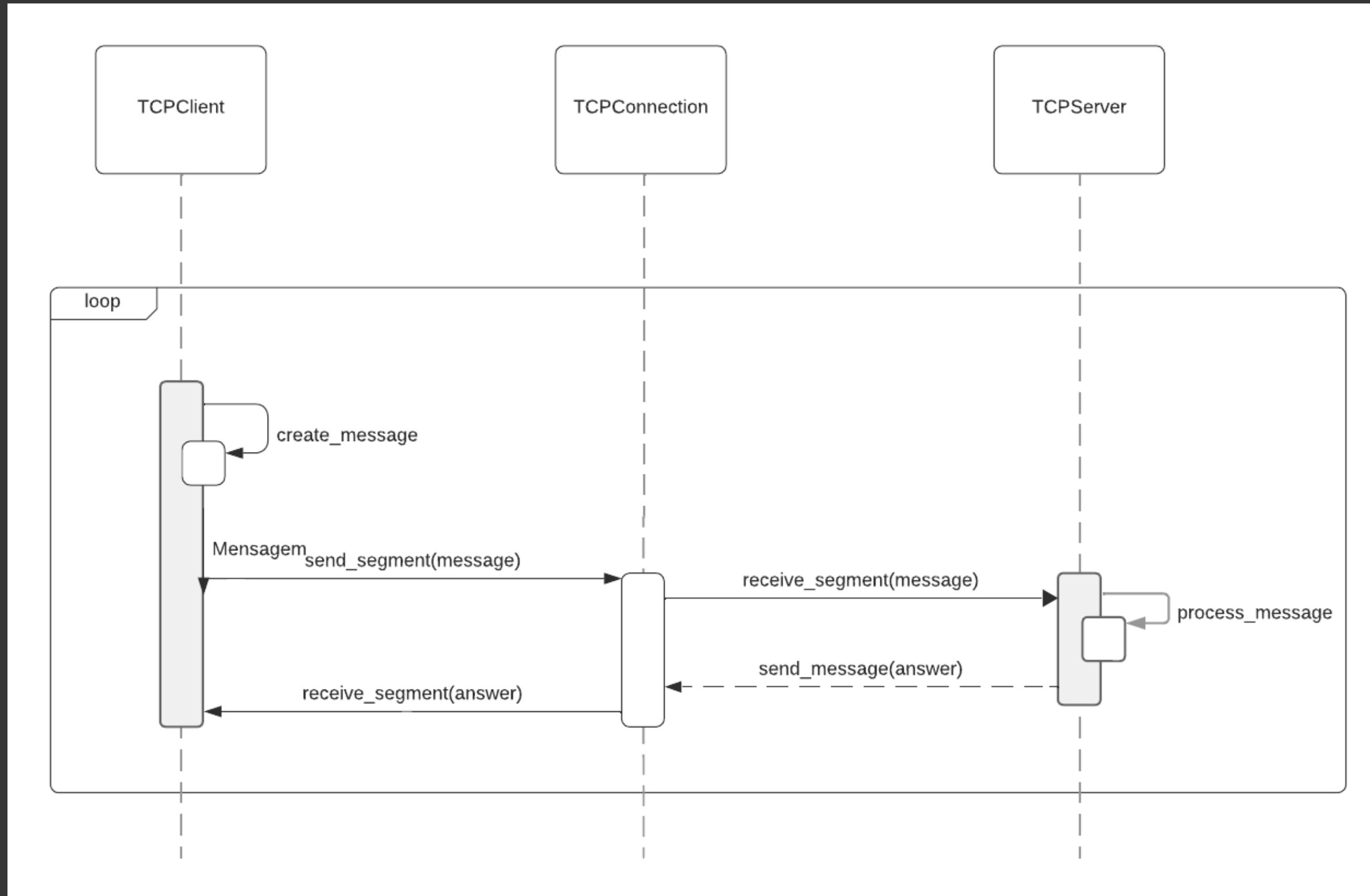
## 4 TCPServer

- Representa o servidor na comunicação TCP.
- Métodos principais:
- `start`: Inicia o servidor e aguarda conexões de clientes.
- `handle_client`: Lida com as mensagens de cada cliente conectado.
- `send_segment`: Envia segmentos TCP de resposta para o cliente.

# DIAGRAMA - CONEXÃO



Age Group	Gender	Believe U.S. should take more action to address climate change
18-29	Men	83%
	Women	78%
30-49	Men	75%
	Women	70%
50-59	Men	65%
	Women	60%
60+	Men	55%
	Women	50%



# CÓDIGO DO EXEMPLO PT1

```
teste_tcp.py > TCPSegment > __str__
1  import socket
2
3  class TCPSegment:
4      def __init__(self, seq_num, ack_num, data):
5          """
6          Inicializa um segmento TCP.
7
8          Parâmetros:
9          seq_num (int): Número de sequência.
10         ack_num (int): Número de confirmação.
11         data (str): Dados do segmento.
12         """
13         self.seq_num = seq_num # Número de sequência
14         self.ack_num = ack_num # Número de confirmação
15         self.data = data      # Dados
16
17     def __str__(self):
18         """
19         Retorna uma representação em string do segmento TCP.
20
21         Retorna:
22         str: Representação no formato "seq_num,ack_num,data".
23         """
24         return f"{self.seq_num},{self.ack_num},{self.data}"
```

```
@staticmethod
def from_string(segment_str):
    """
    Converte uma string representando um segmento TCP em um objeto TCPSegment.

    Parâmetros:
    segment_str (str): String no formato "seq_num,ack_num,data".

    Retorna:
    TCPSegment: Objeto TCPSegment com os valores extraídos.
    """
    seq_num, ack_num, data = segment_str.split(',', 2)
    return TCPSegment(int(seq_num), int(ack_num), data)

class TCPConnection:
    def __init__(self, address):
        """
        Inicializa uma conexão TCP.

        Parâmetros:
        address (tuple): Endereço IP e porta do servidor.
        """
        self.address = address # Endereço IP e porta
        self.state = 'CLOSED'  # Estado da conexão
        self.segments = []     # Segmentos enviados/recebidos

    def send_segment(self, segment):
        """
        Envia um segmento TCP.

        Parâmetros:
        segment (TCPSegment): Objeto TCPSegment a ser enviado.
        """
        print(f"Enviando segmento: {segment}")
        self.segments.append(segment)
```

# CÓDIGO DO EXEMPLO PT2

```
def recebe_segment(self, segment):
    """
    Recebe um segmento TCP.

    Parâmetros:
    segment (TCPSegment): Objeto TCPSegment recebido.
    """
    print(f"Recebendo segmento: {segment}")
    self.segments.append(segment)

class TCPClient:
    def __init__(self, server_address, server_port):
        """
        Inicializa um cliente TCP.

        Parâmetros:
        server_address (str): Endereço IP do servidor.
        server_port (int): Porta do servidor.
        """
        self.server_address = server_address
        self.server_port = server_port
        self.client_socket = None
        self.connection = TCPConnection((server_address, server_port))

    def connect(self):
        """
        Conecta ao servidor TCP.
        """
        try:
            self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.client_socket.connect((self.server_address, self.server_port))
            print(f"Conectado ao servidor {self.server_address}:{self.server_port}")
            self.connection.state = 'SYN_SENT'
            syn_segment = TCPSegment(seq_num=0, ack_num=0, data='SYN')
            self.send_segment(syn_segment)
        except Exception as e:
            print(f"Erro ao conectar: {e}")
            self.client_socket = None
```

```
def send_segment(self, segment):
    """
    Envia um segmento TCP ao servidor.

    Parâmetros:
    segment (TCPSegment): Objeto TCPSegment a ser enviado.
    """
    if self.client_socket:
        try:
            self.client_socket.sendall(str(segment).encode())
            print("Segmento enviado com sucesso.")
        except Exception as e:
            print(f"Erro ao enviar segmento: {e}")

    def send_message(self, message):
        """
        Envia uma mensagem ao servidor.

        Parâmetros:
        message (str): Mensagem a ser enviada.
        """
        if self.client_socket:
            try:
                seq_num = len(self.connection.segments)
                data_segment = TCPSegment(seq_num=seq_num, ack_num=0, data=message)
                self.send_segment(data_segment)
            except Exception as e:
                print(f"Erro ao enviar mensagem: {e}")
```

# CÓDIGO DO EXEMPLO PT3

```
def receive_message(self):
    """
    Recebe uma mensagem do servidor.

    Retorna:
    str: Mensagem recebida do servidor.
    """
    if self.client_socket:
        try:
            response = self.client_socket.recv(1024)
            segment_str = response.decode()
            segment = TCPSegment.from_string(segment_str)
            self.connection.receive_segment(segment)
            return segment_str
        except socket.error as e:
            print(f"Erro ao receber mensagem: {e}") #importante
            return None
        except Exception as e:
            print(f"Erro inesperado ao receber mensagem: {e}")
            return None

def close_connection(self):
    """
    Encerra a conexão com o servidor.
    """
    if self.client_socket:
        self.client_socket.close()
        print("Conexão encerrada.")
    self.client_socket = None
```

```
class TCPServer:
    def __init__(self, host='0.0.0.0', port=5500):#ngrok
        """
        Inicializa um servidor TCP.

        Parâmetros:
        host (str): Endereço IP do servidor.
        port (int): Porta do servidor.
        """
        self.host = host
        self.port = port
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connection = TCPConnection((host, port))

    def start(self):
        """
        Inicia o servidor TCP e aguarda conexões.
        """
        self.server_socket.bind((self.host, self.port))
        self.server_socket.listen(5)
        print(f"Servidor iniciado em {self.host}:{self.port}")

        while True:
            print("Aguardando conexão...")
            client_socket, client_address = self.server_socket.accept()
            print(f"Conexão recebida de {client_address}")
            self.handle_client(client_socket)
```



# CÓDIGO DO EXEMPLO PT4

```
def handle_client(self, client_socket):
    """
    Lida com as mensagens do cliente.

    Parâmetros:
    client_socket (socket): Socket do cliente.
    """
    try:
        while True:
            try:
                message = client_socket.recv(1024)
                if not message:
                    break
                segment_str = message.decode()
                segment = TCPSegment.from_string(segment_str)
                print(f"Segmento recebido: {segment}")
                self.connection.receive_segment(segment)
                response_segment = TCPSegment(seq_num=0, ack_num=segment.seq_num + 1, data='ACK')
                self.send_segment(client_socket, response_segment)
            except socket.error as e:
                print(f"Erro na comunicação com o cliente: {e}")
                break
    except Exception as e:
        print(f"Erro ao lidar com o cliente: {e}")
    finally:
        client_socket.close()
        print("Conexão com o cliente encerrada.")
```

```
def send_segment(self, client_socket, segment):
    """
    Envia um segmento TCP ao cliente.

    Parâmetros:
    client_socket (socket): Socket do cliente.
    segment (TCPSegment): Objeto TCPSegment a ser enviado.
    """
    try:
        client_socket.sendall(str(segment).encode())
        print("Segmento enviado com sucesso.")
    except Exception as e:
        print(f"Erro ao enviar segmento: {e}")
```

```
# Criando a conexão
if __name__ == "__main__":
    print("Escolha uma opção:")
    print("1 - Servidor")
    print("2 - Cliente")
    opcao = input("Opção: ")

    if opcao == "1":
        # Inicializa e inicia o servidor TCP
        server = TCPServer(host="localhost", port=5500)
        server.start()
    elif opcao == "2":
        # Inicializa o cliente TCP e conecta ao servidor
        client = TCPClient(server_address="localhost", server_port=5500)
        client.connect()
        while True:
            msg = input("Digite a mensagem (ou 'sair' para encerrar): ")
            if msg.lower() == "sair":
                client.close_connection()
                break
            client.send_message(msg)
            response = client.receive_message()
            print(f"Resposta do servidor: {response}")
    else:
        print("Opção inválida.")
```





# Conclusão e Considerações Finais

Diagramas de sequência são uma ferramenta valiosa para modelar e visualizar o fluxo de interações em sistemas complexos. Embora existam algumas limitações, suas vantagens superam os desafios, especialmente para projetos de software, análise de sistemas, comunicação de processos e diversas outras áreas que exigem clareza e organização na representação de processos e interações.





## Referências Bibliográfica:

GUEDES, Gilleanes T. A.. **UML 2**: uma abordagem prática.. 3. ed. São Paulo: Novatec, 2018.

[https://youtu.be/PpsEaqJV\\_A0?si=kZs9xcjVhz3tdCaK](https://youtu.be/PpsEaqJV_A0?si=kZs9xcjVhz3tdCaK) (What is TCP/IP)

Chatgpt, Copilot, Gemini, Llama 3

<https://youtu.be/uwoD5YsGACg?si=pCrZ4DdZqi3ixKgS> (TCP vs UDP Comparison)

<https://www.fortinet.com/br/resources/cyberglossary/tcp-ip-model-vs-osi-model> (Modelo OSI e TCP/IP)

[https://youtu.be/F27PLin3TV0?si=8SzGP5-10JJjhD\\_Y](https://youtu.be/F27PLin3TV0?si=8SzGP5-10JJjhD_Y) (TCP Walkthrough)



---

# OBRIGADO PELA ATENÇÃO

## EQUIPE

Caio Rodrigues

Fábio Borges

Gabriel Duarte

Guilherme Bandeira

Isabela Avelino

Vitor Braga

Vivian Rodrigues

---