Diagramas de Sequência: Interações Dinâmicas

Prof. Me. Lucas Bruzzone

Aula 14

Diagrama de Sequência

Objetivo

Mostrar a interação entre objetos ao longo do tempo, enfocando a ordem das mensagens

Características:

- Dimensão temporal explícita (eixo vertical = tempo)
- Foco na comunicação entre objetos
- Detalha implementação de casos de uso
- Mostra fluxo de controle e dados
- Útil para design detalhado
- Identifica responsabilidades dos objetos

Quando usar: Análise, design detalhado, documentação e debugging

Por que usar Diagramas de Sequência?

Benefícios práticos:

- **Visualização clara**: Fluxo temporal das interações
- **Validação de design**: Identifica problemas antes da implementação
- **Comunicação**: Facilita discussão entre equipe
- **Documentação viva**: Evolui com o sistema
- **Base para testes**: Cenários derivados do diagrama
- **Identificação de gargalos**: Performance e complexidade

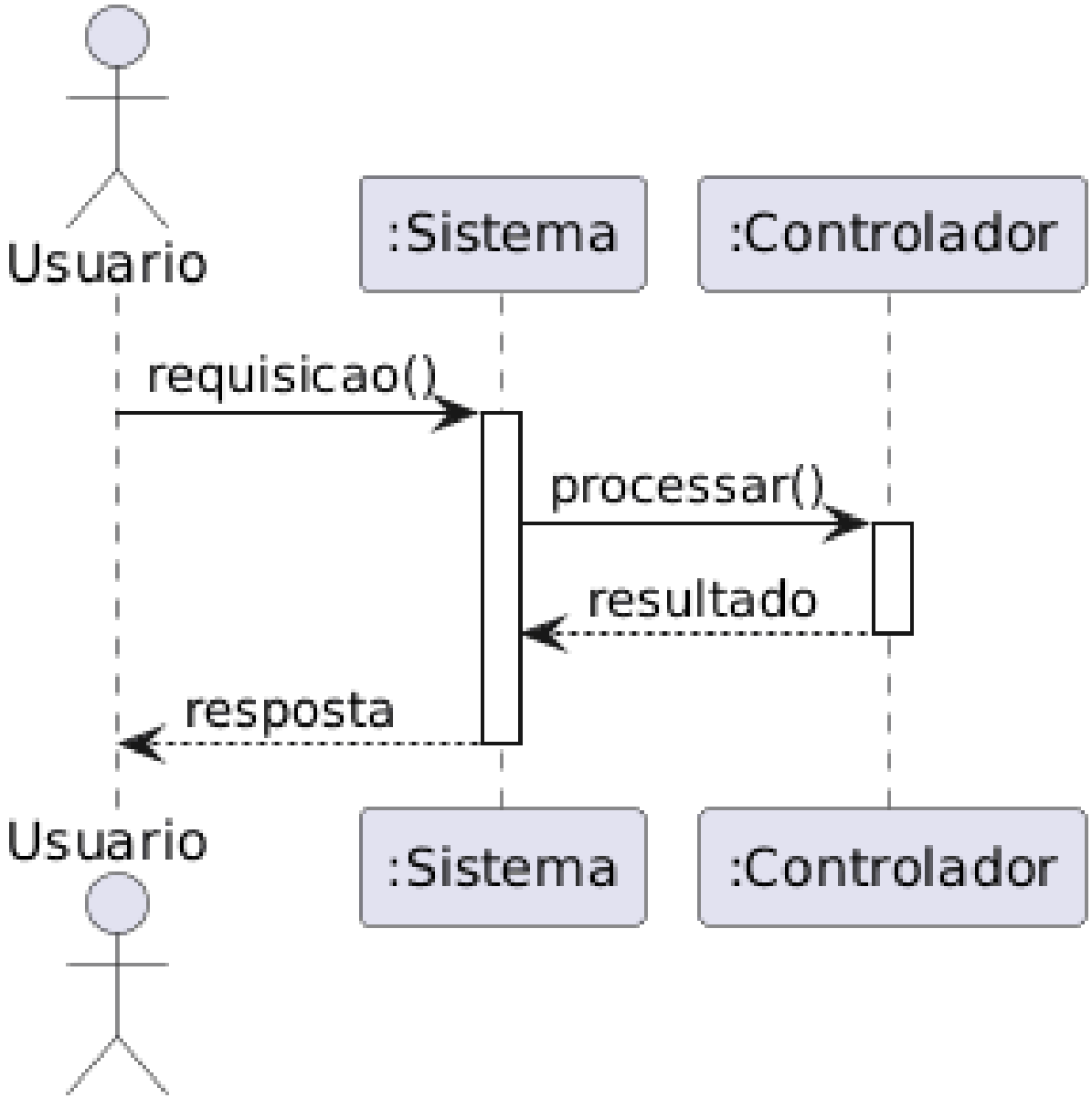
Um diagrama bem feito vale mais que páginas de documentação!

Componentes do Diagrama

Elementos principais:

- **Atores**: Usuários ou sistemas externos
- **Objetos/Participantes**: Instâncias de classes
- **Linha de vida**: Existência do objeto no tempo
- **Mensagens**: Comunicação entre objetos
- **Ativação**: Período de execução (foco de controle)
- **Fragmentos**: Estruturas de controle (if, loop, etc.)

Exemplo de Diagrama Básico



Atores e Objetos

Notação de objetos:

`nomeObjeto:NomeClasse`

Variações:

- `:Cliente` - objeto anônimo (somente classe)
- `joao:Cliente` - objeto nomeado específico
- `Cliente` - representa a própria classe

Posicionamento:

- Retângulos no topo do diagrama
- Ordenados da esquerda para direita (fluxo lógico)
- Ator geralmente à esquerda (iniciador)
- Objetos de sistema no centro/direita

Linha de Vida e Ativação

Linha de Vida:

- Linha vertical tracejada
- Representa existência do objeto no tempo
- Desce do objeto até fim da interação
- Pode ser interrompida (destruição do objeto)

Ativação (Focus of Control):

- Retângulo estreito sobre linha de vida
- Indica quando objeto está ativo/executando
- Processando ou controlando a interação
- Pode ser aninhada (chamadas recursivas ou síncronas)
- Altura = duração da execução

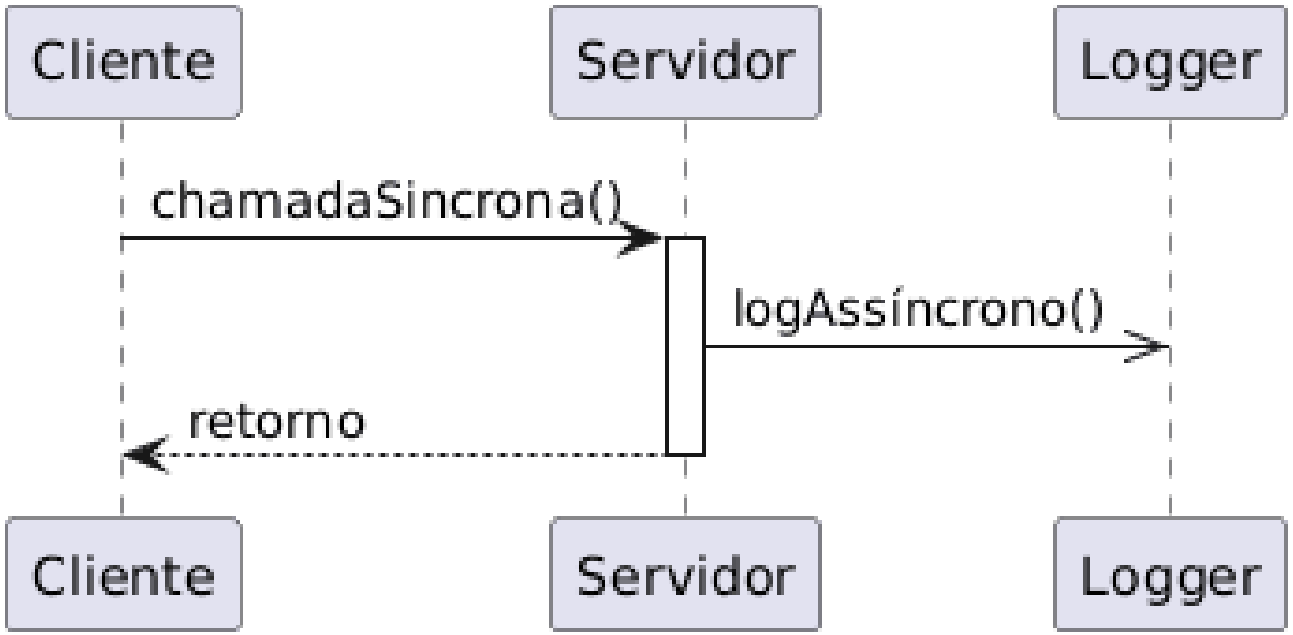
Dica: Quanto mais ativações um objeto tem, mais responsabilidades ele possui

Tipos de Mensagens

1. **Síncrona (Synchronous):**
   - Remetente espera resposta antes de continuar
   - Seta cheia: $\rightarrow$
   - Mais comum em sistemas tradicionais
   - Exemplo: Chamada de método que retorna valor

2. **Assíncrona (Asynchronous):**
   - Remetente não espera resposta
   - Seta aberta:
   - Comum em sistemas paralelos/distribuídos
   - Exemplo: Envio de email, logging, eventos

Exemplo - Mensagens Síncrona e Assíncrona



Observe:

- Chamada síncrona aguarda retorno
- Log assíncrono não bloqueia execução

Tipos de Mensagens (continuação)

3. **Retorno (Return):**
   - Resposta a uma chamada síncrona
   - Linha tracejada com seta aberta: ← - -
   - Opcional se óbvio pelo contexto
   - Pode incluir valor de retorno

4. **Criação (Create):**
   - Cria novo objeto
   - Estereótipo `<<create>>`
   - Seta aponta para o objeto criado
   - Objeto aparece no momento da criação

5. **Destruição (Destroy):**
   - Destrói/finaliza objeto
   - Estereótipo `<<destroy>>`
   - X grande no fim da linha de vida
   - Libera recursos

Sintaxe de Mensagens

Formato completo:

```
sequencia: retorno = nomeMsg(parametros): tipo
```

Exemplos práticos:

- `1: login(usuario, senha)`
- `2: resultado = validar(): boolean`
- `3: exibirMensagem(texto: String): void`
- `1.1: dados = buscarDados(id: int): Cliente`
- `4: total = calcularTotal(): double`

Numeração das mensagens:

- **Sequencial**: 1, 2, 3, 4... (simples e direto)
- **Hierárquica**: 1, 1.1, 1.2, 2, 2.1... (mostra aninhamento)
- **Opcional**: Em diagramas simples pode ser omitida

Exercício Rápido 1

Atividade

Identifique o tipo de mensagem em cada cenário:

1. Um método `calcularMedia()` que retorna um double
2. Um sistema que envia notificação por email e continua
3. Um construtor criando novo objeto Produto
4. Um método `fecharConexao()` que libera recurso
5. Um método `getCliente(id)` que busca no banco

Tempo: 2 minutos

Respostas: 1-Síncrona, 2-Assíncrona, 3-Criação, 4-Destruição, 5-Síncrona

Fragmentos de Interação

Estruturas de controle (Combined Fragments):

- **opt**: Opcional - executa se condição verdadeira (if)
- **alt**: Alternativa - escolhe um caminho (if-else)
- **loop**: Repetição - executa múltiplas vezes (while/for)
- **par**: Paralelo - execução concorrente
- **ref**: Referência a outro diagrama (reutilização)
- **break**: Interrompe o fluxo normal

Representação:

- Retângulo com cantos arredondados
- Rótulo no canto superior esquerdo (operador)
- Condição entre colchetes: `[condição]`
- Pode ser aninhado (fragmento dentro de fragmento)

Fragmento OPT - Opcional

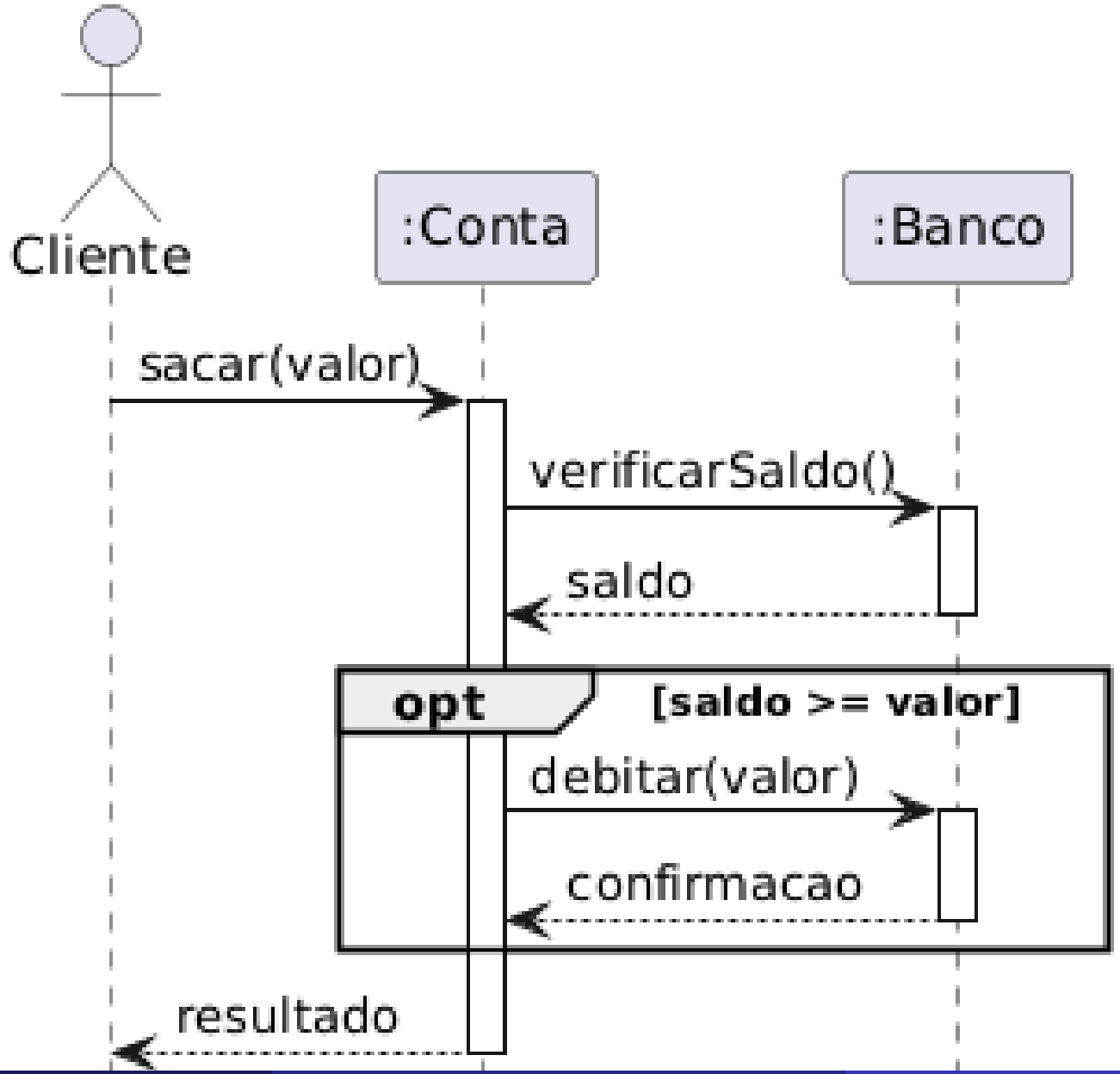
Estrutura:

```
opt [condição]
```

Quando usar:

- Execução condicional simples
- Validações antes de processar
- Ações opcionais baseadas em estado

Exemplo - Fragmento OPT



Fragmento ALT - Alternativa

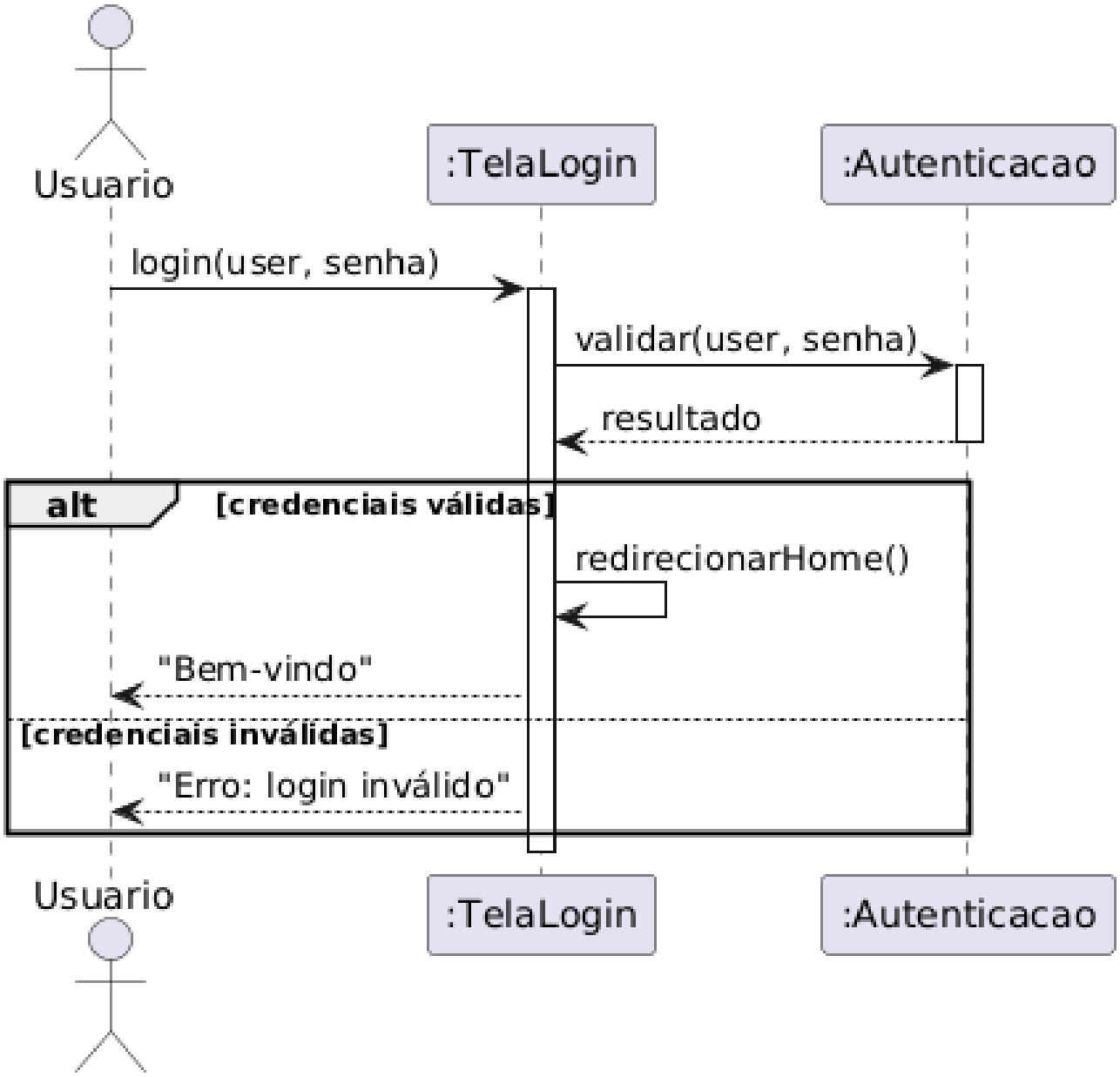
Estrutura:

```
alt [condição verdadeira]
else [condição falsa]
```

Características:

- Duas ou mais alternativas mutuamente exclusivas
- Equivale a estrutura if-else
- Separado por linha horizontal tracejada
- Apenas um caminho é executado

Exemplo - Fragmento ALT



Fragmento LOOP - Repetição

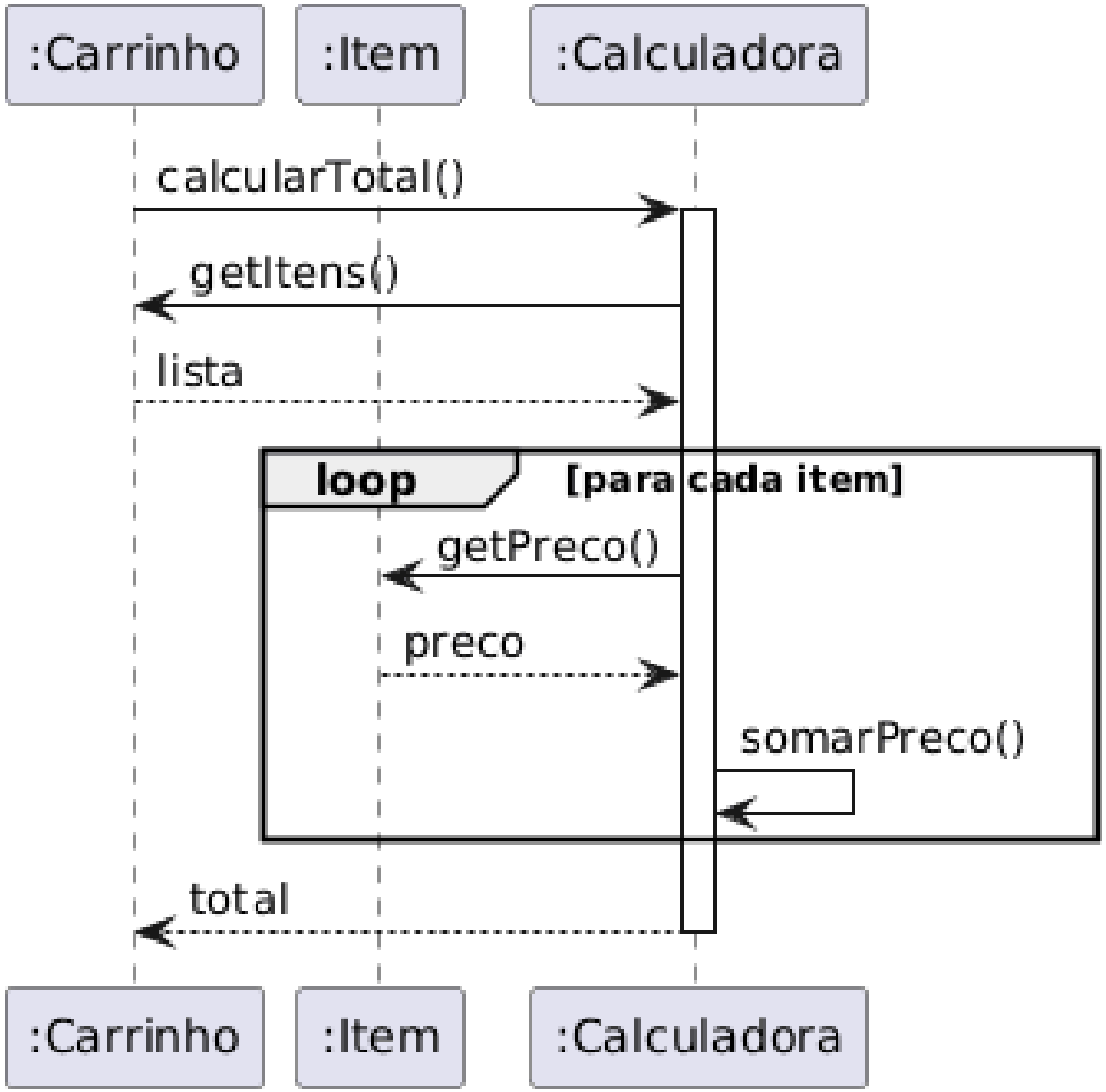
Estrutura:

```
loop [condição ou intervalo]
```

Condições comuns:

- `[i = 1..n]` - loop com contador fixo
- `[para cada item]` - iteração sobre coleção
- `[hasNext()]` - iteração com condição
- `[tentativas < 3]` - loop condicional
- `[enquanto conectado]` - loop baseado em estado

Exemplo - Fragmento LOOP



Outros Fragmentos Importantes

PAR - Execução Paralela:

- Mensagens executam simultaneamente
- Comum em sistemas distribuídos
- Exemplo: Carregar dados de múltiplas APIs

REF - Referência:

- Referencia outro diagrama de sequência
- Evita repetição e simplifica diagramas complexos
- Sintaxe: `ref ProcessarPagamento`
- Promove reutilização

BREAK - Interrupção:

- Interrompe execução normal
- Similar a break/return em código
- Exemplo: Erro fatal que aborta operação

Exercício Rápido 2

Atividade

Qual fragmento usar em cada situação?

1. Aplicar desconto apenas se cliente for VIP
2. Processar cada item do carrinho de compras
3. Login: redirecionar se válido, mostrar erro se inválido
4. Enviar email E enviar SMS simultaneamente
5. Abortar transação se detectar fraude

Tempo: 2 minutos

Respostas: 1-opt, 2-loop, 3-alt, 4-par, 5-break

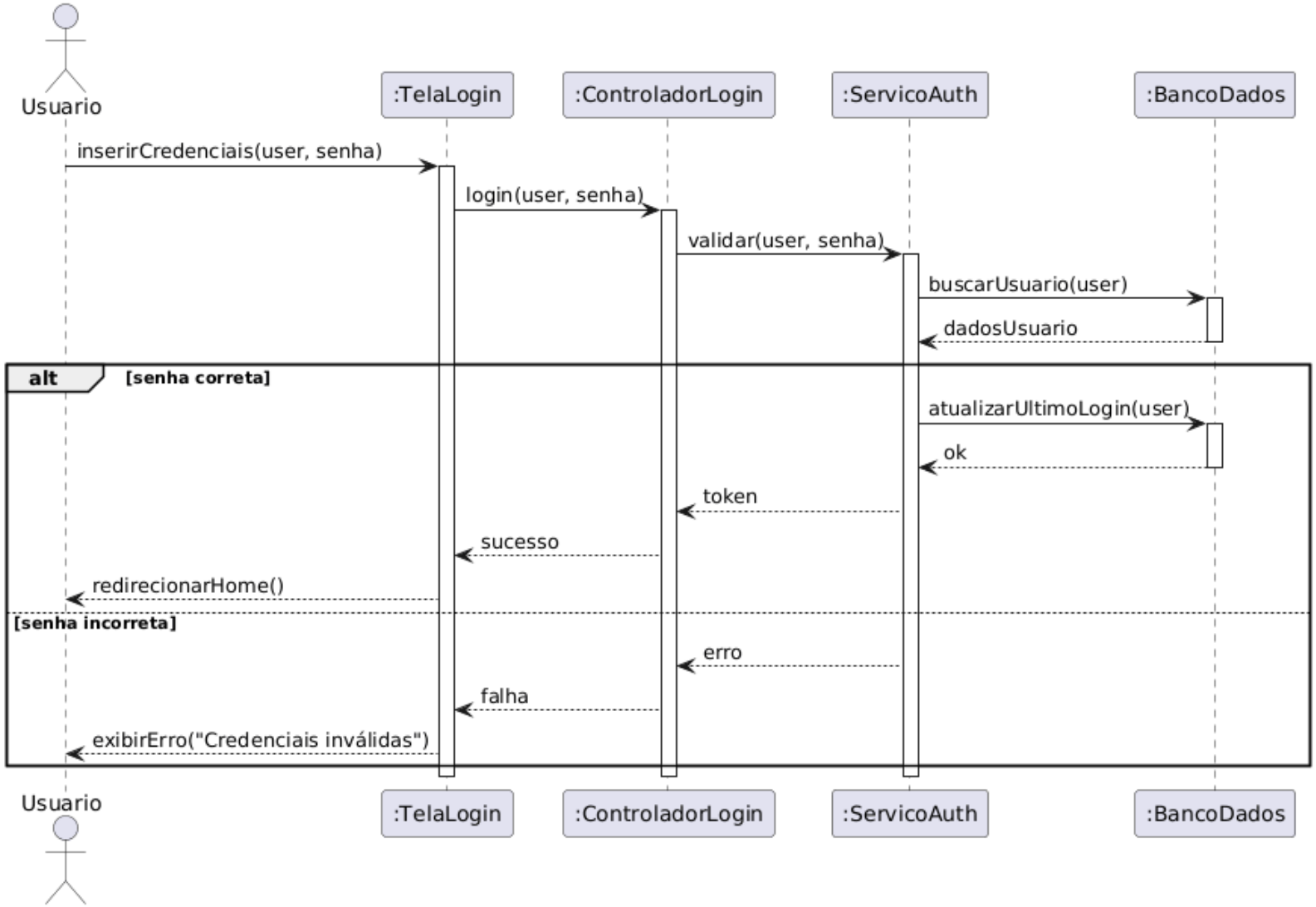
Exemplo Completo - Login no Sistema

Cenário: Usuário autentica no sistema

Participantes:

- Usuario (ator)
- :TelaLogin
- :ControladorLogin
- :ServicoAutenticacao
- :BancoDados

Diagrama - Login no Sistema



Análise do Exemplo - Login

Observações importantes:

- **Separação de responsabilidades**
  - UI: Interface com usuário
  - Controlador: Orquestração
  - Serviço: Lógica de negócio
  - BD: Persistência
- **Uso de ALT**: Dois caminhos distintos (sucesso/erro)
- **Mensagens síncronas**: Cada camada aguarda resposta
- **Retornos importantes**: Token, dados, confirmações

Benefício: Design modular, testável e manutenível

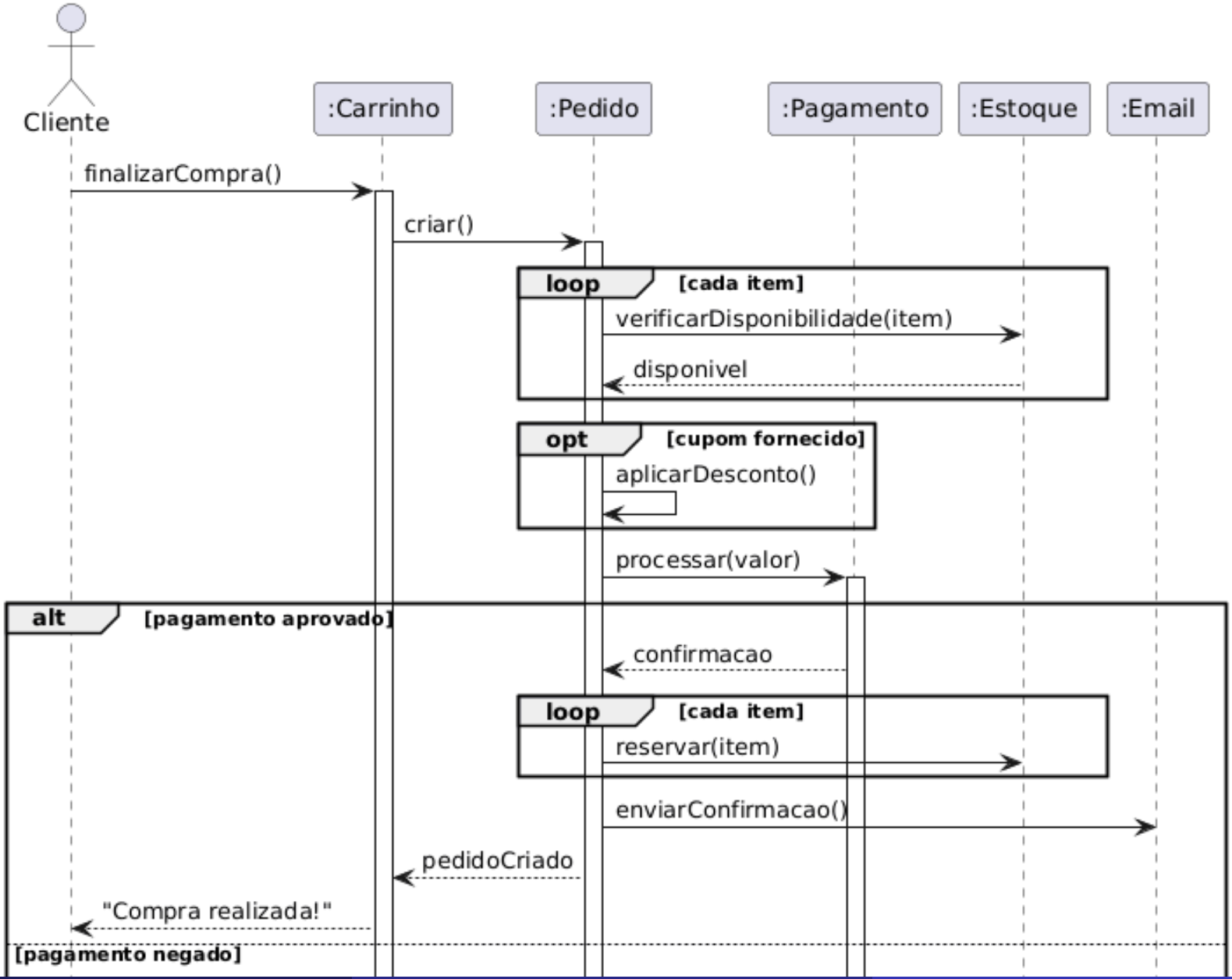
Exemplo Completo - Processamento de Pedido

Cenário: Cliente finaliza compra no e-commerce

Participantes:

- Cliente (ator)
- :Carrinho, :Pedido
- :ServicoPagamento, :Estoque
- :ServicoEmail

Diagrama - Processamento de Pedido



Análise do Exemplo - Pedido

Fragmentos utilizados:

- **LOOP**: Verificar cada item no estoque
- **OPT**: Aplicar desconto se cupom válido
- **ALT**: Processar sucesso ou falha do pagamento
- **LOOP aninhado**: Reservar itens após confirmação

Padrões identificados:

- Validação antes de processar (estoque)
- Operações condicionais (desconto)
- Tratamento de erro (pagamento negado)
- Notificação assíncrona (email)
- Transação completa com rollback implícito

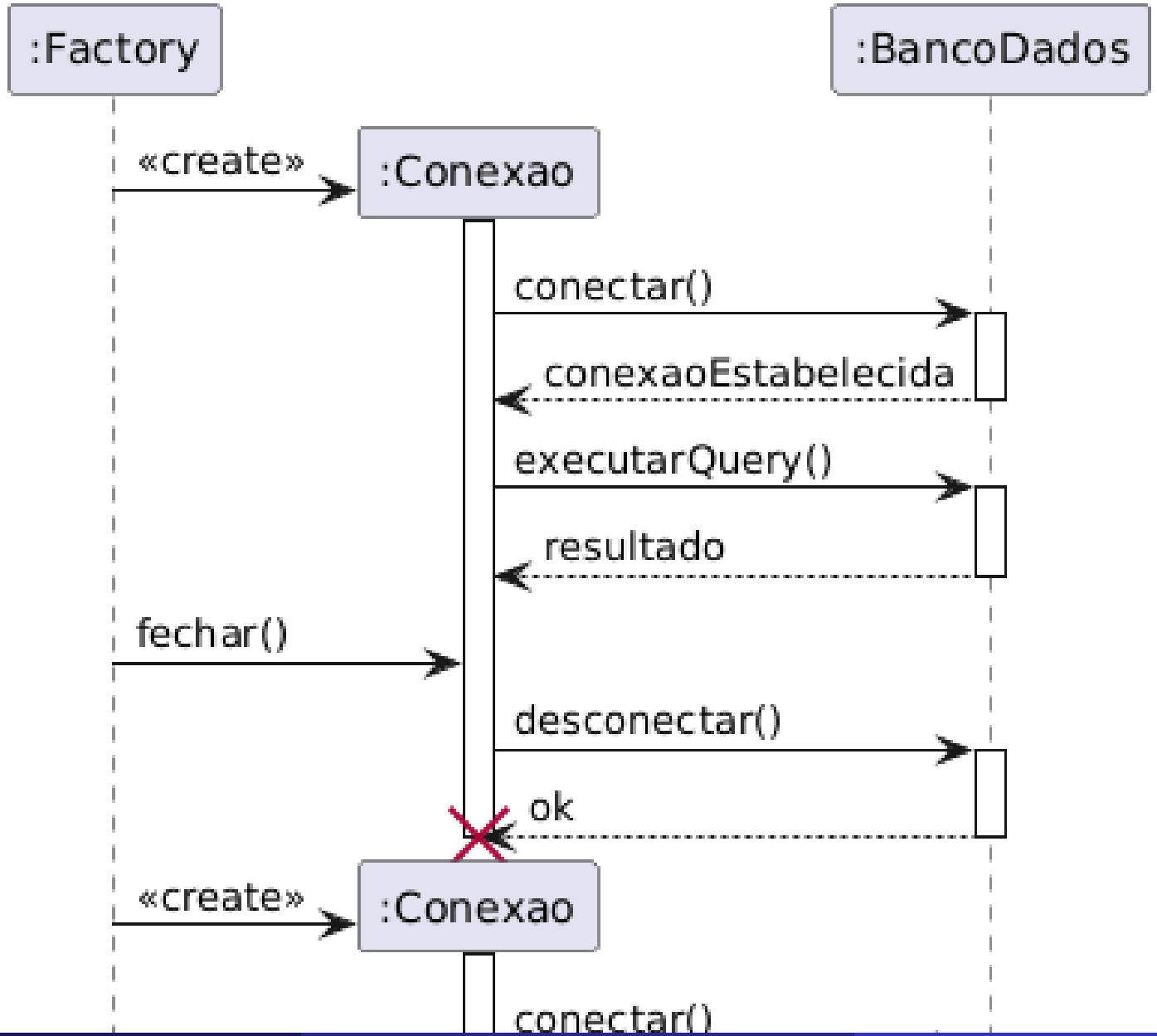
Exemplo - Criação e Destruição de Objetos

Cenário: Gerenciamento de conexão com banco de dados

Pontos importantes:

- **¡¡create¿¿**: Factory cria nova conexão
- **Ciclo de vida**: Conexão existe durante as operações
- **Destruição**: X marca fim da vida do objeto
- **Padrão**: Criar $\rightarrow$ Usar $\rightarrow$ Destruir (RAII)

Diagrama - Criação e Destruição



Relacionamento com Outros Diagramas

Casos de Uso:

- Diagrama de sequência detalha caso de uso
- Um caso de uso $\rightarrow$ múltiplas sequências (cenários)
- Fluxo principal $\rightarrow$ sequência principal
- Fluxos alternativos $\rightarrow$ sequências variantes

Diagrama de Classes:

- Objetos são instâncias das classes
- Mensagens tornam-se métodos nas classes
- Valida responsabilidades e colaborações
- Identifica métodos necessários

Fluxo de trabalho:

$$\text{Casos de Uso} \rightarrow \text{Sequência} \rightarrow \text{Classes} \rightarrow \text{Implementação}$$

Boas Práticas de Modelagem

- **Foque no essencial**: Não modele tudo, escolha cenários críticos
- **Nível adequado**: Nem muito abstrato, nem muito detalhado
- **Organize logicamente**: Esquerda $\rightarrow$ direita segue fluxo natural
- **Use fragmentos com critério**: Não complique sem necessidade
- **Numere quando necessário**: Mensagens complexas precisam ordem
- **Mantenha atualizado**: Diagrama evolui com sistema
- **Validação**: Peça feedback da equipe
- **Consistência**: Mesma notação em todos diagramas

Lembre-se: Clareza $>$ Completude

Erros Comuns e Como Evitar

- **Muito detalhado**
  - Erro: Incluir cada getter/setter
  - Solução: Abstrair operações triviais
- **Objetos em excesso**
  - Erro: 10+ objetos no diagrama
  - Solução: Agrupar ou usar REF
- **Mensagens vagas**
  - Erro: processar(), fazer(), executar()
  - Solução: Nomes específicos e descritivos
- **Ignorar retornos importantes**
  - Erro: Não mostrar dados críticos
  - Solução: Indicar retornos que afetam fluxo
- **Ordem temporal incorreta**
  - Erro: Mensagens fora de sequência lógica
  - Solução: Revisar cronologia

Quando NÃO usar Diagrama de Sequência

Situações onde outros diagramas são melhores:

- **Visão estrutural estática** → Use Diagrama de Classes
- **Estados de um objeto** → Use Diagrama de Estados
- **Fluxo de processo** → Use Diagrama de Atividades
- **Arquitetura do sistema** → Use Diagrama de Componentes
- **Interações muito simples** → Pode ser overkill

Regra prática:

Use sequência quando o TEMPO e a ORDEM das interações são importantes

Dicas Avançadas

Para sistemas complexos:

- **Divida e conquiste**: Múltiplos diagramas para cenários
- **Níveis de abstração**: Alto nível + detalhado quando necessário
- **Use REF**: Reutilize sequências comuns
- **Anote decisões**: Comentários explicam escolhas
- **Versione**: Controle evolução do design

Para análise de performance:

- Identifique gargalos (muitas ativações)
- Marque operações custosas
- Avalie chamadas síncronas vs assíncronas
- Considere paralelização (PAR)

Exercício Prático

Atividade

Baseado no trabalho do primeiro bimestre, no diagrama de classes e nos casos de uso das aulas anteriores, desenvolva diagramas de sequência da sua aplicação

Requisitos:

- Escolher 2-3 casos de uso principais do seu sistema
- Criar um diagrama de sequência para cada caso de uso
- Incluir pelo menos 4 participantes por diagrama
- Utilizar no mínimo 1 fragmento (opt, alt, ou loop)
- Aplicar numeração hierárquica nas mensagens
- Indicar retornos importantes
- Mostrar ativações dos objetos

Entrega: Diagramas + Documento explicativo

Dicas para o Exercício

- **Revise seus casos de uso**
  - Escolha fluxos principais e alternativos
- **Use as classes do diagrama anterior**
  - Objetos devem ser instâncias das suas classes
- **Identifique os atores**
  - Quem inicia a interação?
- **Mapeie a sequência temporal**
  - O que acontece primeiro, segundo, terceiro...
- **Identifique decisões e repetições**
  - Onde usar alt, opt, ou loop?
- **Valide a consistência**
  - Métodos existem nas classes?

Ferramentas para Diagramas de Sequência

Baseadas em texto (recomendadas):

- **PlantUML**: Sintaxe simples, integra com IDEs
- **Mermaid**: Suporte nativo no GitHub/GitLab
- **WebSequenceDiagrams**: Interface web amigável

Visuais (drag-and-drop):

- **Lucidchart**: Online, colaborativa
- **Draw.io**: Gratuita, integra Google Drive
- **Visual Paradigm**: Completa, profissional
- **StarUML**: Desktop, gratuita

Dica: PlantUML é excelente para versionamento com Git!

PlantUML - Exemplo Prático

Código simples:

```
@startuml
actor Usuario
participant ":Sistema" as Sys

Usuario -> Sys: login(user, senha)
activate Sys
Sys -> Sys: validar()

alt credenciais válidas
    Sys --> Usuario: "Bem-vindo"
else inválidas
    Sys --> Usuario: "Erro"
end
deactivate Sys
@enduml
```

Resumo da Aula

O que aprendemos:

- Componentes do diagrama de sequência
- Tipos de mensagens e sua notação
- Fragmentos de interação (opt, alt, loop, par, ref)
- Exemplos práticos de modelagem
- Boas práticas e erros comuns
- Relacionamento com outros diagramas UML
- Ferramentas para criação de diagramas

Diagramas de sequência são essenciais para entender e documentar o comportamento dinâmico do sistema

Próxima Aula

Diagrama de Atividades: Modelagem de Processos

Estudaremos como modelar fluxos de trabalho e processos de negócio