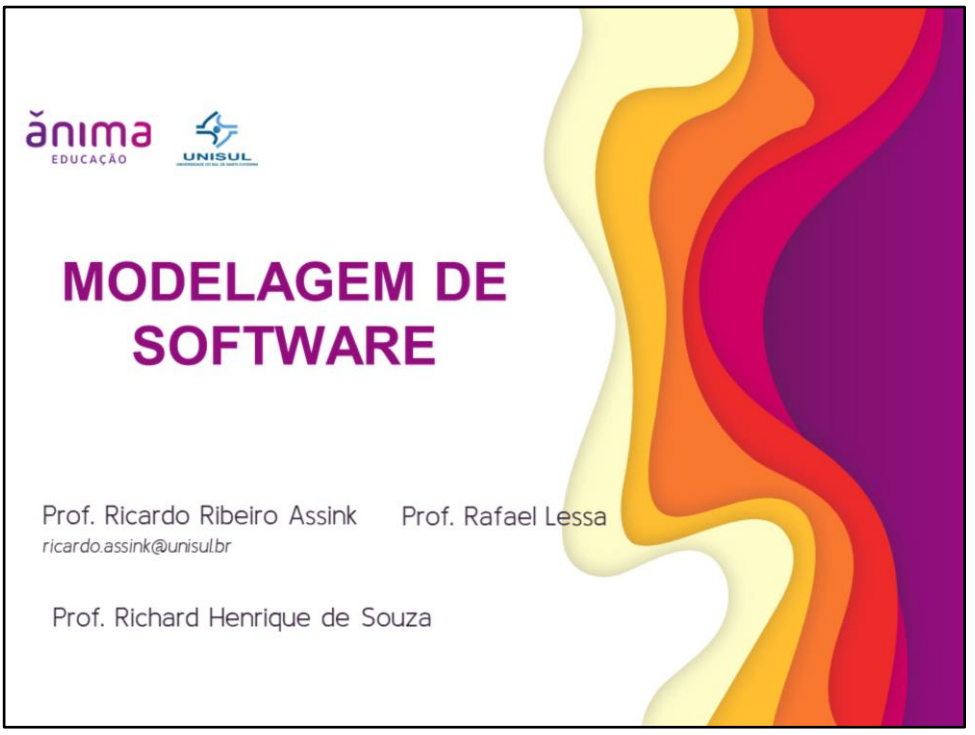
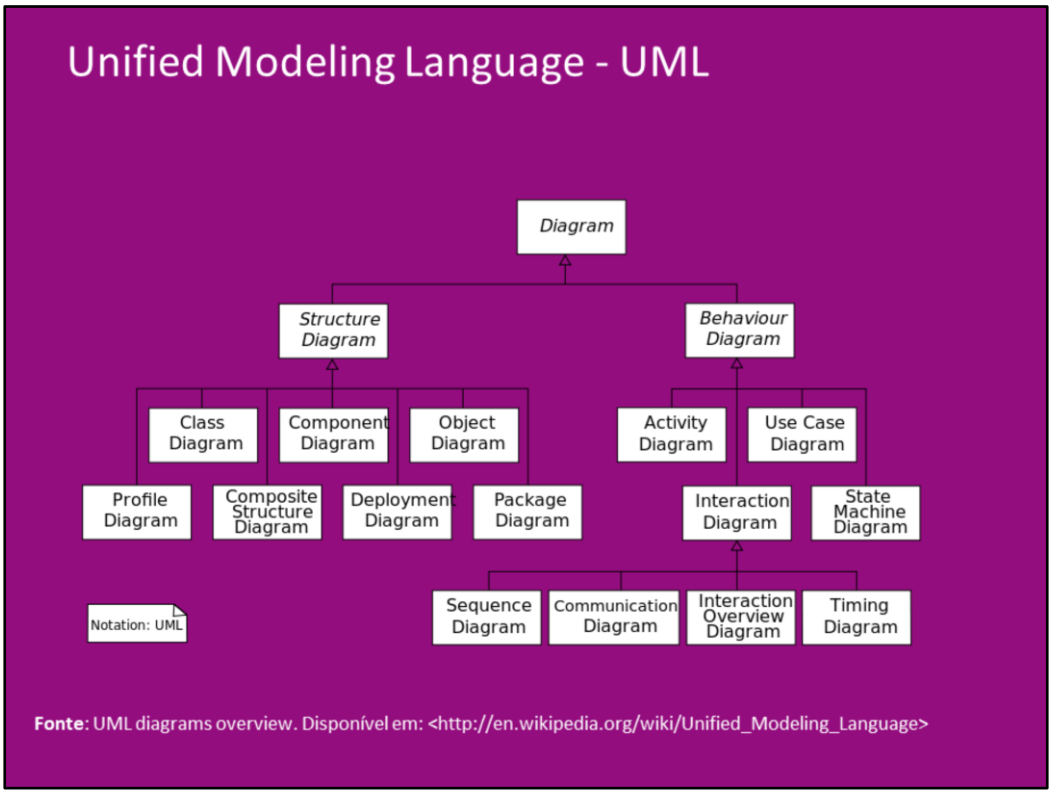

ânima
EDUCAÇÃO
UNISUL
MODELAGEM DE SOFTWARE
Prof. Ricardo Ribeiro Assink
ricardo.assink@unisul.br
Prof. Rafael Lessa
Prof. Richard Henrique de Souza

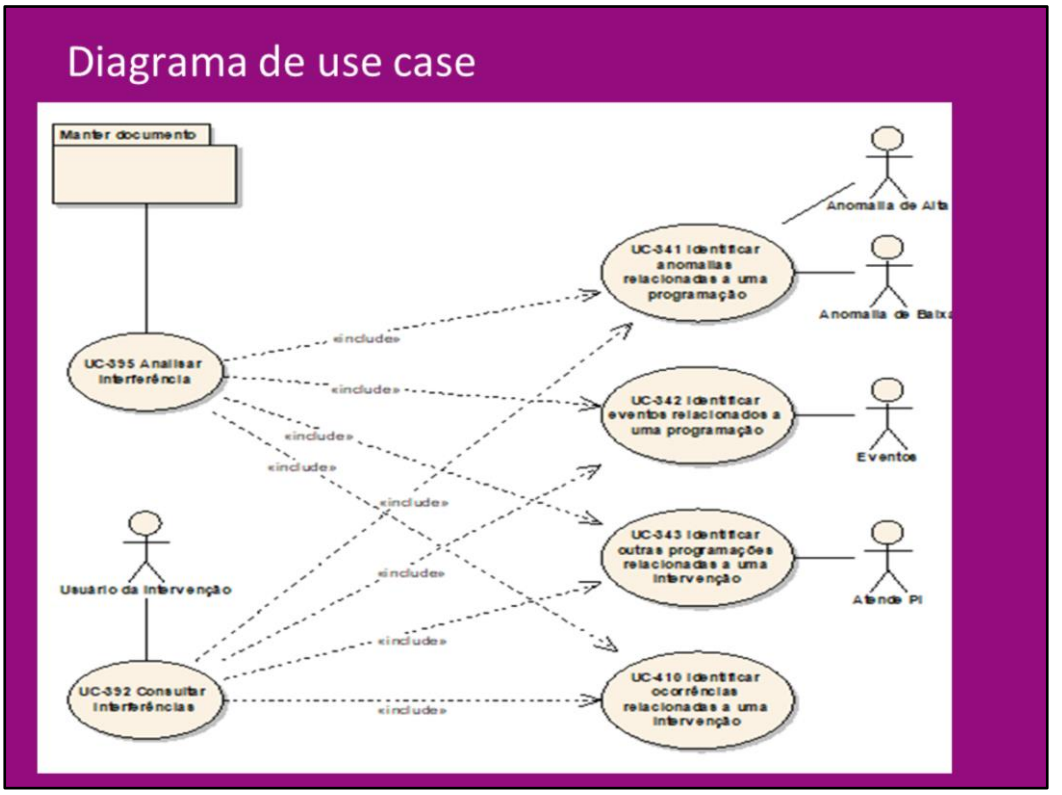
Agenda

Modelagem UML e a Orientação a Objetos

- *Unified Modeling Language* – UML
- Digrama de sequencia



Descrição textual do DIAGRAMA
Notação UML
Primeiro temos o elemento “Diagram” no TOPO do diagrama, ligados a outros dois elementos: “Structure Diagram” e “Behaviour Diagram”, essa ligação é a de herença, onde “Diagram” é o elemento genérico.
O Elemento “Structue Diagram” está ligado aos elementos: “ Profle Diagram”, “Class Diagram”, “composite Structure Diagram”, “Component Diagram”, “Deployment Diagram”, “Object Diagram” e “Package Diagram”, também a ligação indica o relacionamento de herança, onde “Structue Diagram” é o elemento genérico.
O elemento “Behaviour Diagram” está ligado aos elementos “Activity Diagram”, ”Use case Diagram”, ”State Machine Diagram”, ”Interaction Diagram” , também a ligação indica o relacionamento de herança, onde “Behaviour Diagram” é o elemento genérico.
O elemento ”Interaction Diagram” está ligado aos elementos: “Sequence Diagram”, ”Communication Diagram”, ”Interaction Oberview Diagram”, ”Timing Diagram” , também a ligação indica o relacionamento de herança, onde “Interaction Diagram” é o elemento genérico.

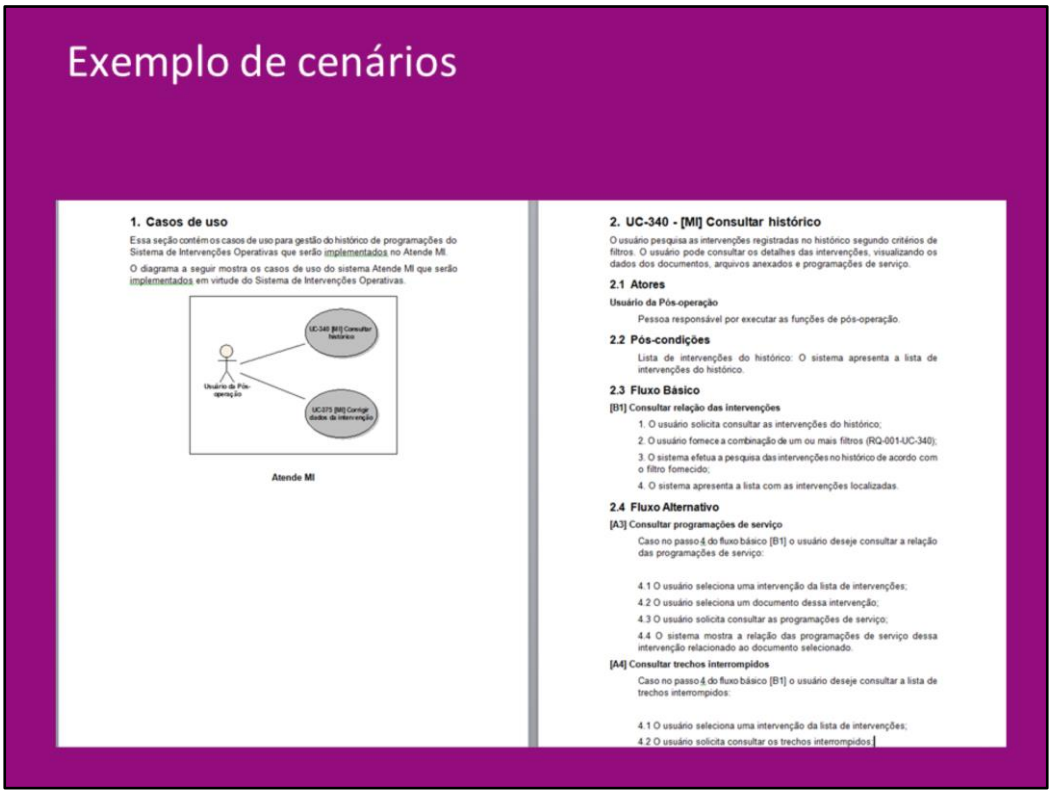


#PraCegoVer
O diagrama de casos de uso apresenta 6 casos de uso: UC-395 Analisar Interferência, UC-392 Consultar Interferências, UC-341 Identificar anomalias relacionada a uma programação, UC-342 Identificar eventos relacionados a uma programação, UC- 343 Identificar outras programações relacionadas a uma intervenção, UC-410 Identificar ocorrências relacionadas a uma intervenção
Também tem 5 atores Usuário da intervenção, Anolalia de alta, Anomalia de baixa, Eventos, Atende PI
e 1 pacote: Manter documento

O caso de uso UC- 343 Identificar outras programações relacionadas a uma intervenção tem um relacionamento de include com os casos de uso: UC-341 Identificar anomalias relacionada a uma programação, UC-342 Identificar eventos relacionados a uma programação, UC- 343 Identificar outras programações relacionadas a uma intervenção, UC-410 Identificar ocorrências relacionadas a uma intervenção
O caso de uso , UC-392 Consultar Interferências tem um relacionamento de include com os casos de uso: UC-341 Identificar anomalias relacionada a uma programação, UC-342 Identificar eventos relacionados a uma programação, UC- 343 Identificar outras programações relacionadas a uma intervenção, UC-410 Identificar ocorrências relacionadas a uma intervenção

Em dotUML

```
UseCaseDiagram [frame=true framecolor=steelblue
   label="Usecase Diagram"] {
  attribute usecase [fillcolor=paleturquoise]
  actor a1 as "Usuário da intervenção"
  actor a2 as "Anolalia de alta"
actor a3 as "Anomalia de baixa"
actor a4  as "Eventos"
actor a5  as "Atende PI"

       usecase u1 as "UC395 Analisar Interferência"
    usecase u2 as "UC392 Consultar Interferências"
    usecase u3 as "UC341 Identificar \n anomalias
   relacionada a uma programação"
    usecase u4 as "UC342 Identificar \n  eventos
   relacionados a uma programação"
    usecase u5 as "UC- 343 Identificar outras  \n
   programações relacionadas a uma intervenção"
    usecase u6  as "UC-410 Identificar ocorrências \n
   relacionadas a uma intervenção"


     u1 -i-> u3
  u1 -i-> u4
  u1 -i-> u5
  u1 -i-> u6
  u2 -i-> u3
  u2 -i-> u4
  u2 -i-> u5
  u2 -i-> u6
  a1 -- u2
  a2 -- u3
  a3 -- u3
  a4 -- u4
  a5 -- u5
 note "Manter documento" top of u1
}
```



#PraCegoVer

1. Casos de uso

Essa seção contém os casos de uso para gestão do histórico de programações do Sistema de Intervenções Operativas que serão implementados no Atende MI.

O diagrama a seguir mostra os casos de uso do sistema Atende MI que serão implementados em virtude do Sistema de Intervenções Operativas.

A figura contêm dois casos de uso:

UC-340 – [MI] Consultar histórico

UC-275 – [MI] Corrigir dados da Intervenção

A figura com têm um ator

Ator: Usuário de Pós-operação

Sistema Atende Mi

Em dotUML:

```
UseCaseDiagram [frame=true
  framecolor=steelblue label="Usecase
  Diagram"] {
```

```
  attribute usecase [fillcolor=paleturquoise]
  actor a1 as "Usuário de \n Pós-operação"
  system s as "Sistema Atende Mi" {
    usecase u340 as "UC - 340 - \n [MI]
  Consultar histórico"
    usecase u275 as "UC-275 – \n [MI]
  Corrigir dados da Intervenção"
    }
    a1 -- u340
    a1 -- u275
}
```

1. UC-340 – [MI] Consultar histórico

O usuário pesquisa as intervenções registradas no histórico segundo critérios de filtros. O usuário pode consultar os detalhes das intervenções, visualizando os dados dos documentos, arquivos anexados e programações de serviço.

1. Atores

Usuário de Pós-operação

Pessoa responsável por executar as funções de pós-operação

1. Pós-condições

Lista de intervenções do histórico: O sistema apresenta a lista de intervenções do histórico.

1. Fluxo Básico

[B1] Consultar relação das intervenções

1. O usuário solicita consultar as intervenções do histórico.

2. O usuário fornece a combinação de um ou mais filtros (RQ-001-UC-340)

3. O sistema efetua pesquisa das intervenções no histórico de acordo com o filtro fornecido.

4. O sistema apresenta a lista com as intervenções localizadas.

1. Fluxo Alternativo

[A3] Consultar programações de serviço

Caso no passo 4 do fluxo básico [B1] o usuário deseje consultar a relação das programações de serviço.

4.1 O usuário seleciona uma intervenção da lista de intervenções.

4.2 O usuário seleciona um documento dessa intervenção.

4.3 O usuário solicita consultar as programações de serviço.

4.4 O sistema mostra a relação das programações de serviço dessa intervenção relacionado ao documento selecionado.

[A4] Consultar trechos interrompidos

Caso no passo 4 do fluxo básico [B1] o usuário deseje consultar a relação a lista de trechos interrompidos.

4.1 O usuário seleciona uma intervenção da lista de intervenções.

4.2 O usuário solicita consultar os trechos interrompidos.

4.3 O sistema mostra a relação dos trechos interrompidos.

[A5] Imprimir documento

Caso no passo 4.3 do fluxo alternativo [A1] o usuário desejar imprimir o documento

4.4 O usuário solicita a impressão do documento

4.5 O sistema envia o documento para a impressora.

[A1] Consulta dos detalhes de documento

Caso no passo 4 do fluxo básico [B1] o usuário deseje consultar detalhes de um documento específico.

4.1 O usuário seleciona uma intervenção da lista de intervenções.

4.2 O usuário seleciona um documento da intervenção selecionada.

4.3 O sistema mostra os dados do documento selecionado.

[A2] Execução do download de arquivo em anexo.

Caso no passo 4 do fluxo básico [B1] o usuário desejar fazer o download de um arquivo especifico.

4.1 O usuário seleciona uma intervenção da lista de intervenções

4.2 O usuário seleciona o anexo para executar o download.

4.3 O sistema efetua o download do arquivo selecionado na máquina local do usuário.

1. Fluxo de Exceção

[E1] Erro ao executar o download do arquivo

Caso no passo 4.3 do fluxo alternativo [A2] o sistema não conseguir gravar o arquivo na máquina local do usuário.

download do arquivo.

4.3.2 Encerra o caso de uso.

1. Requisitos específicos

RQ-001-UC-340 Filtros para consulta

(Validação)

1. Os filtros para consulta das intervenções no histórico são descritos no item 1.1 do documento “ATECH 611.01.00018 – Especificação de campos das telas”.

RQ-002-UC-340 Filtros para consulta

(Relatório)

1. A consulta de histórico é limitada a um período máximo de 1 mês.

1. Requisitos Realizado

RQ-176-Terminal de histórico

(Restrição de Design)

1. Devem ser incorporados ao terminal do Atende MI as funcionalidades de consulta de histórico das intervenções da distribuição (PropTec. 5.3).

RQ-180 – Consulta detalhes de intervenção

(Funcional)

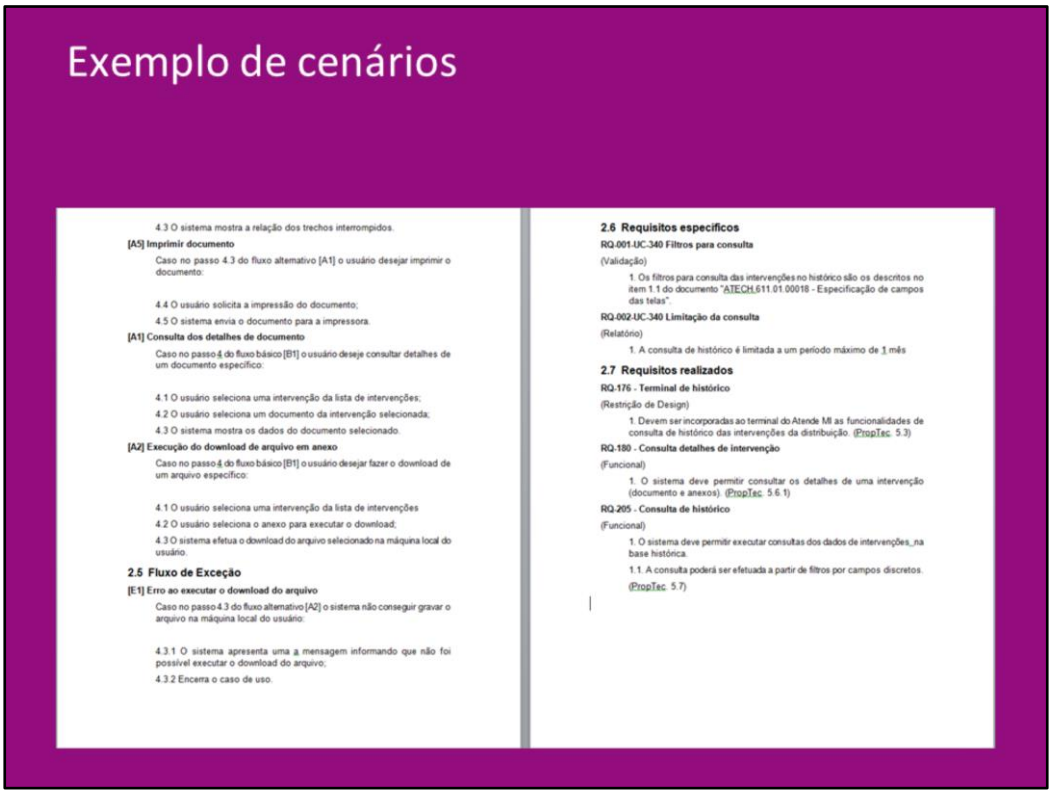
1. O sistema deve permitir consultar os detalhes de uma intervenção (documento e anexos) (PropTec. 5.6.1)

RQ-205-Consulta de histórico

(Funcional)

1. O sistema deve permitir executar consultas dos dados de intervenções na base histórica.

1.1 A consulta poderá ser efetuada a partir de filtros por campos discretos (PropTec. 5.7)



#PraCegoVer

1. Casos de uso

Essa seção contém os casos de uso para gestão do histórico de programações do Sistema de Intervenções Operativas que serão implementados no Atende MI.

O diagrama a seguir mostra os casos de uso do sistema Atende MI que serão implementados em virtude do Sistema de Intervenções Operativas.

A figura contêm dois casos de uso:

UC-340 – [MI] Consultar histórico

UC-275 – [MI] Corrigir dados da Intervenção

A figura com têm um ator

Ator: Usuário de Pós-operação

Sistema Atende Mi

1. UC-340 – [MI] Consultar histórico

O usuário pesquisa as intervenções registradas no histórico segundo critérios de filtros. O usuário pode consultar os detalhes das intervenções, visualizando os dados dos documentos, arquivos anexados e programações de serviço.

1. Atores

Usuário de Pós-operação

Pessoa responsável por executar as funções de pós-operação

1. Pós-condições

Lista de intervenções do histórico: O sistema apresenta a lista de intervenções do histórico.

1. Fluxo Básico

[B1] Consultar relação das intervenções

1. O usuário solicita consultar as intervenções do histórico.

2. O usuário fornece a combinação de um ou mais filtros (RQ-001-UC-340)

3. O sistema efetua pesquisa das intervenções no histórico de acordo com o filtro fornecido.

4. O sistema apresenta a lista com as intervenções localizadas.

1. Fluxo Alternativo

[A3] Consultar programações de serviço

Caso no passo 4 do fluxo básico [B1] o usuário deseje consultar a relação das programações de serviço.

4.1 O usuário seleciona uma intervenção da lista de intervenções.

4.2 O usuário seleciona um documento dessa intervenção.

4.3 O usuário solicita consultar as programações de serviço.

4.4 O sistema mostra a relação das programações de serviço dessa intervenção relacionado ao documento selecionado.

[A4] Consultar trechos interrompidos

Caso no passo 4 do fluxo básico [B1] o usuário deseje consultar a relação a lista de trechos interrompidos.

4.1 O usuário seleciona uma intervenção da lista de intervenções.

4.2 O usuário solicita consultar os trechos interrompidos.

4.3 O sistema mostra a relação dos trechos interrompidos.

[A5] Imprimir documento

Caso no passo 4.3 do fluxo alternativo [A1] o usuário desejar imprimir o documento

4.4 O usuário solicita a impressão do documento

4.5 O sistema envia o documento para a impressora.

[A1] Consulta dos detalhes de documento

Caso no passo 4 do fluxo básico [B1] o usuário deseje consultar detalhes de um documento específico.

4.1 O usuário seleciona uma intervenção da lista de intervenções.

4.2 O usuário seleciona um documento da intervenção selecionada.

4.3 O sistema mostra os dados do documento selecionado.

[A2] Execução do download de arquivo em anexo.

Caso no passo 4 do fluxo básico [B1] o usuário desejar fazer o download de um arquivo especifico.

4.1 O usuário seleciona uma intervenção da lista de intervenções

4.2 O usuário seleciona o anexo para executar o download.

4.3 O sistema efetua o download do arquivo selecionado na máquina local do usuário.

1. Fluxo de Exceção

[E1] Erro ao executar o download do arquivo

Caso no passo 4.3 do fluxo alternativo [A2] o sistema não conseguir gravar o arquivo na máquina local do usuário.

4.3.1 O sistema apresenta uma a mensagem informando que não foi possível executar o download do arquivo.

4.3.2 Encerra o caso de uso.

1. Requisitos específicos

RQ-001-UC-340 Filtros para consulta

(Validação)

1. Os filtros para consulta das intervenções no histórico são descritos no item 1.1 do documento “ATECH 611.01.00018 – Especificação de campos das telas”.

RQ-002-UC-340 Filtros para consulta

(Relatório)

1. A consulta de histórico é limitada a um período máximo de 1 mês.

1. Requisitos Realizado

RQ-176-Terminal de histórico

(Restrição de Design)

histórico das intervenções da distribuição (PropTec. 5.3).

RQ-180 – Consulta detalhes de intervenção

(Funcional)

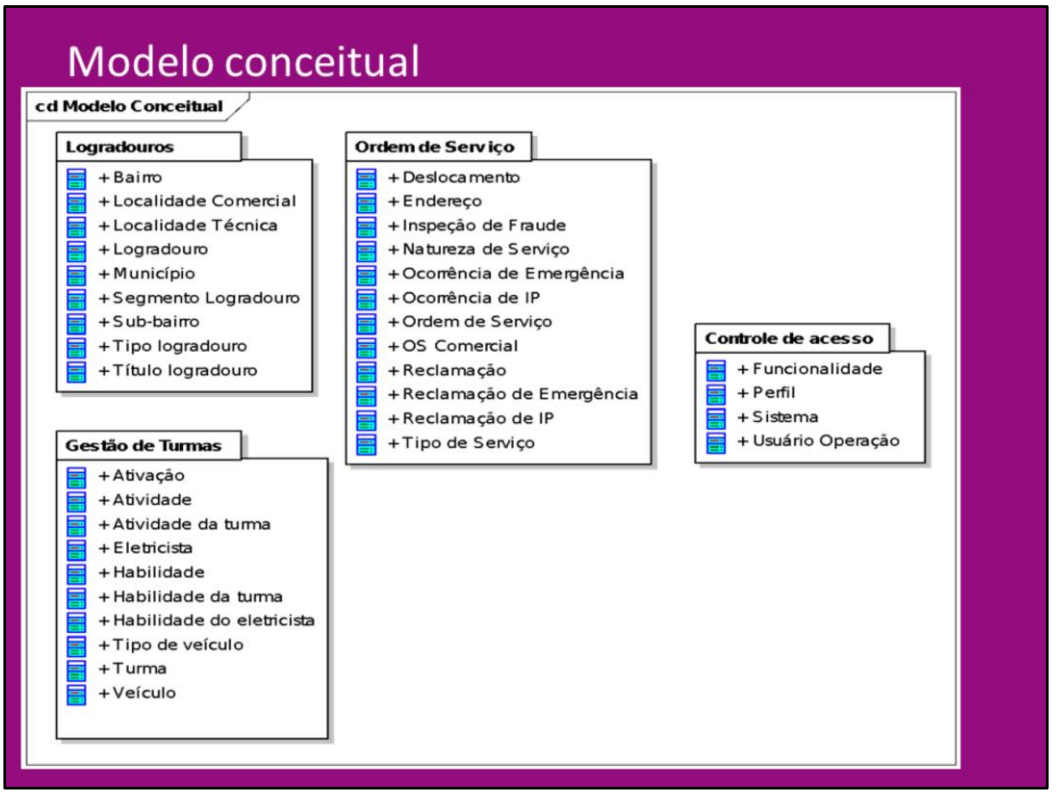
1. O sistema deve permitir consultar os detalhes de uma intervenção (documento e anexos) (PropTec. 5.6.1)

RQ-205-Consulta de histórico

(Funcional)

1. O sistema deve permitir executar consultas dos dados de intervenções na base histórica.

1.1 A consulta poderá ser efetuada a partir de filtros por campos discretos (PropTec. 5.7)



#PraCegoVer
O diagrama cd Modelo Conceitual contêm 4 classes: Logradouros, Ordem de Serviço, gestão de Turmas e Controle de acesso.
A classe logradouros contém os atributos: bairro, Localidade comercial, localidade técnica, logradouro, Munícipio, segmento logradouro, sub-bairro, tipo logradouro, Título logradouro.
A Classe Ordem de Serviço contém os atributos: Deslocamento, Endereço, inspeção de Fraude, Natureza de serviço, Ocorrência de Emergência, Ocorrência de IP, Ordem de Serviço, OS comercial, Reclamação, Reclamação de Emerhência, Reclamação de IP , Tipo de Serviço.
A classe gestão de Turmas contém os seguintes atributos: Ativação, atividade, atividade da turma, eletricista, habilidade, habilidade de eletricista, tipo de veiculo, turma, veiculo.
A classe de Controle de acesso contém os seguintes atributos:. Funcionalidade, Perfil, Sistema, Usuário Operação.

Em dotUML:

```
ClassDiagram [frame=true framecolor=steelblue label="Class
  Diagram"] {

 class Logradouros {
   bairro
   Localidade_comercial
```

```
  localidade_tecnica
  logradouro
  Municipio
  segmento_logradouro
  sub_bairro
  tipo_logradouro
  Titulo_logradouro.
}

class Ordem_de_Servico {
  Deslocamento
  Endereco
  inspecao_de_Fraude
  Natureza_de_servico
  Ocorrencia_de_Emergencia
  Ocorrencia_de_IP
  Ordem_de_Servico
  OS_comercial
  Reclamacao
  Reclamacao_de_Emergencia
  Reclamacao_de_IP
  Tipo_de_Servico
}

   class Gestao_de_Turmas {
 Ativacao
 atividade
 atividade_da_turma
 eletricista
 habilidade
 habilidade_de_eletricista
 tipo_de_veiculo
 turma
 veiculo
}

class Controle_de_Acesso {
 Funcionalidade
 Perfil
 Sistema
  Usuario
  Operação
}

  }
```

Comunicação entre objetos

Conceitualmente, objetos se comunicam através da troca de mensagens.

Mensagens definem:

- O nome do serviço requisitado
- A informação necessária para a execução do serviço
- O nome do requisitante.

Comunicação entre objetos

Na prática, mensagens são implementadas como chamadas de função.

- Nome = o nome do procedimento.
- Informação = a lista de parâmetros.
- Requisitante = o procedimento que realizou a chamada.

Diagramas de interação

Uma colaboração é um nome dado à interação entre duas ou mais classes.

Um colaboração é representada por um conjunto de interações entre objetos selecionados em um contexto limitado, focando a relação entre os objetos.

Esta colaboração pode mostrar a implementação de uma operação ou a realização de um use case.

Diagramas de interação

Uma colaboração pode ser descrita com mais de um diagrama de interação, com cada um mostrando uma visão diferente.

Normalmente pode-se escolher entre utilizar o diagrama de **colaboração** ou o diagrama de **sequência**.

Basicamente, estes diagramas mostram o mesmo fato em perspectivas diferentes.

Diagrama de colaboração

O **diagrama de colaboração** enfatiza o objeto e sua cooperação com os outros objetos, entre eles, as mensagens são apresentadas.

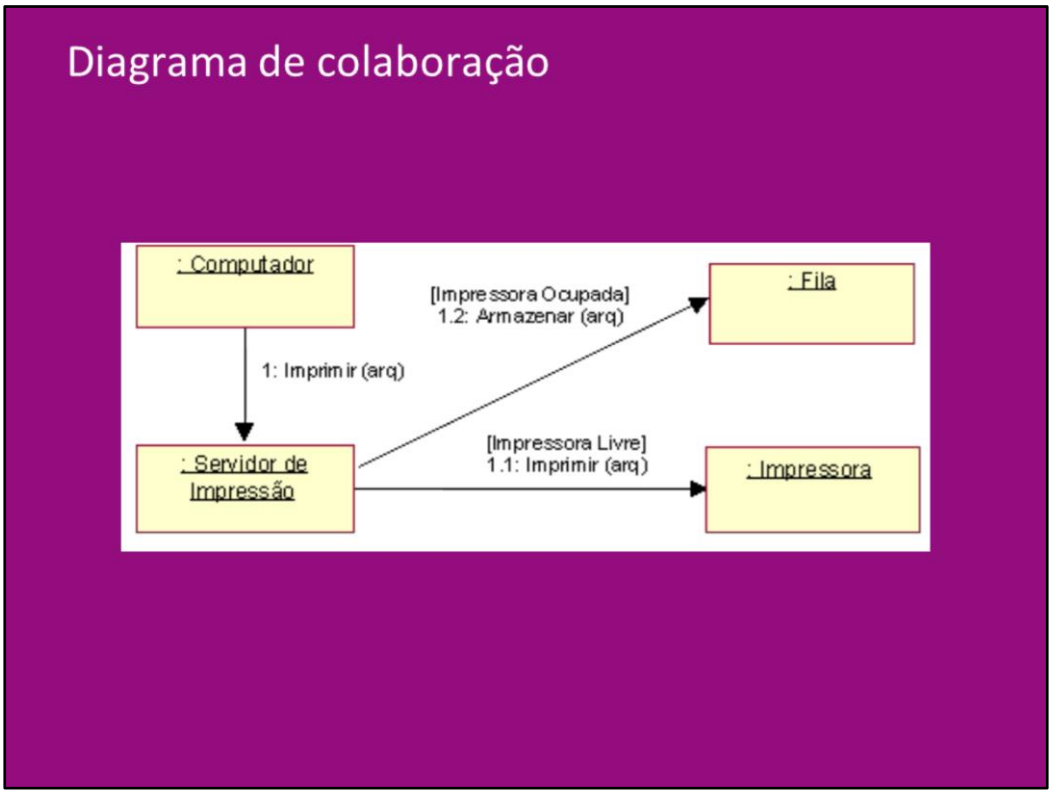
No diagrama de colaboração, além de mostrar a troca de mensagens entre os objetos, percebe-se também os objetos com os seus relacionamentos.

Este diagrama mostra a cronologia das mensagens, seus nomes e respostas, e seus possíveis argumentos.

Diagrama de colaboração

O diagrama de colaboração é desenhado como um diagrama de objeto, onde os diversos objetos são mostrados juntamente com seus relacionamentos.

A cronologia é apresentada através da numeração das mensagens. A relação do objeto pode existir temporariamente ou permanentemente.



#PraCegoVer
Há 4 objetos: “: Computador”, “:Servidor de Impressão”, “: Fila” , “ : Impressoa”
Entre :Computador e : Servidor de Impressão tem uma mensagem: 1: Imprimir(arq), a mesagem é direcionada do :computador para : Servidor de Impressão.
Entre : Servidor de Impressão e :Fila tem uma mensagem: 1.2: Armazenar (Arq), a mesagem é direcionada do :Servidor de Impressão para : Fila. E Tem um texto de guarda na mensagem: [Impressora Ocupada]
Entre : Servidor de Impressão e :Impressora tem uma mensagem: 1.1: Imprimir (Arq), a mesagem é direcionada do :Servidor de Impressão para : Impressora. E Tem um texto de guarda na mensagem: [Impressora Livre]

Como o dotUML não tem suporte para o diagrama de colaboração, então irei usar o mais próximo que é o de sequencia, o qual será visto após a explicação do diagrama e colaboração.
Em dotUML:

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Colaboração Diagram"] {
// Usaremos o mesmo código do diagrama de sequencia

   lifeline ": Computador" as c
lifeline ": Servisor de impressão" as s
lifeline ": Fila" as f
lifeline ": Impressora" as i
```

```
   c --> s  "1. Imprimir(arq)"
s --> i  "[Impressora Livre] \n 1.1 Imprimir (arq)"
s --> f  "[Impressora Ocupada] \n 1.2 Armazenar (arq)"

   }
```

Diagrama de colaboração

Sintaxe:

```
PredecessorCondition SequenceExpression
                ReturnValue      :=      MessageName
(ArgumentList)
```

PredecessorCondition [opcional]: numeração da mensagem que deve ter sido enviada antes da mensagem atual ser executada. Ex: 1.1, 1.2

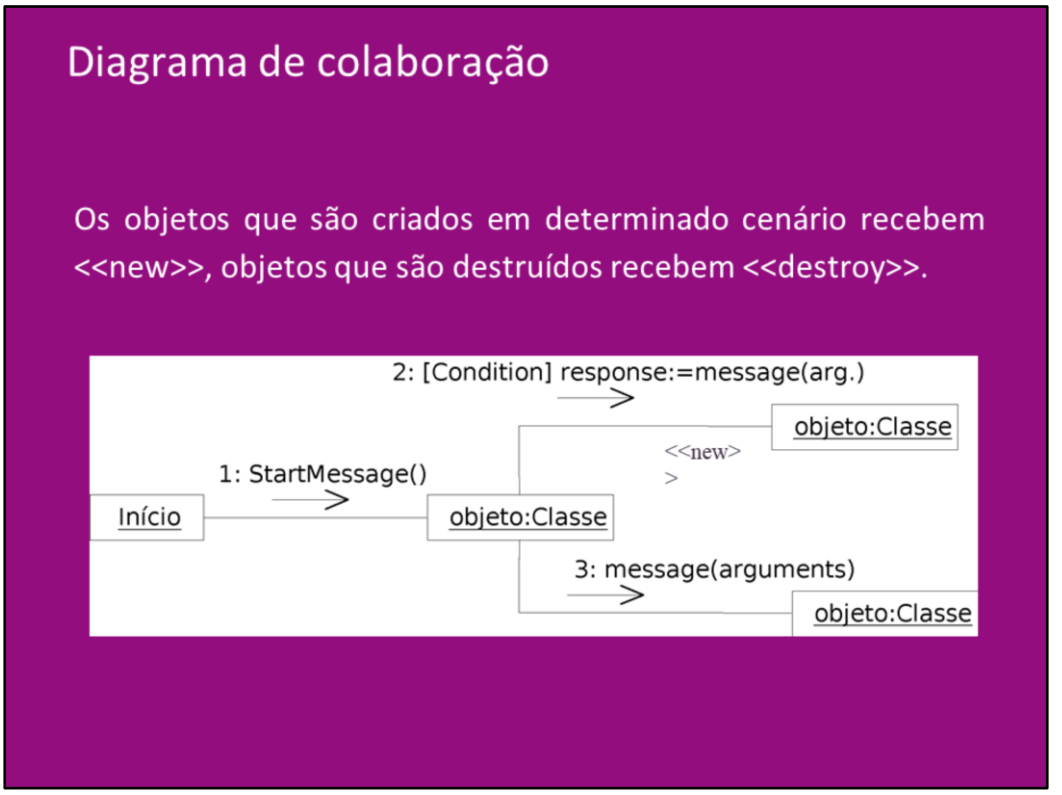
Sequence Expression: ordem ascendente de numeração.

Exemplos:

1.2.* [i:=1..n]

1.2.* || [i:=1..n]

1.2.* [x>5]

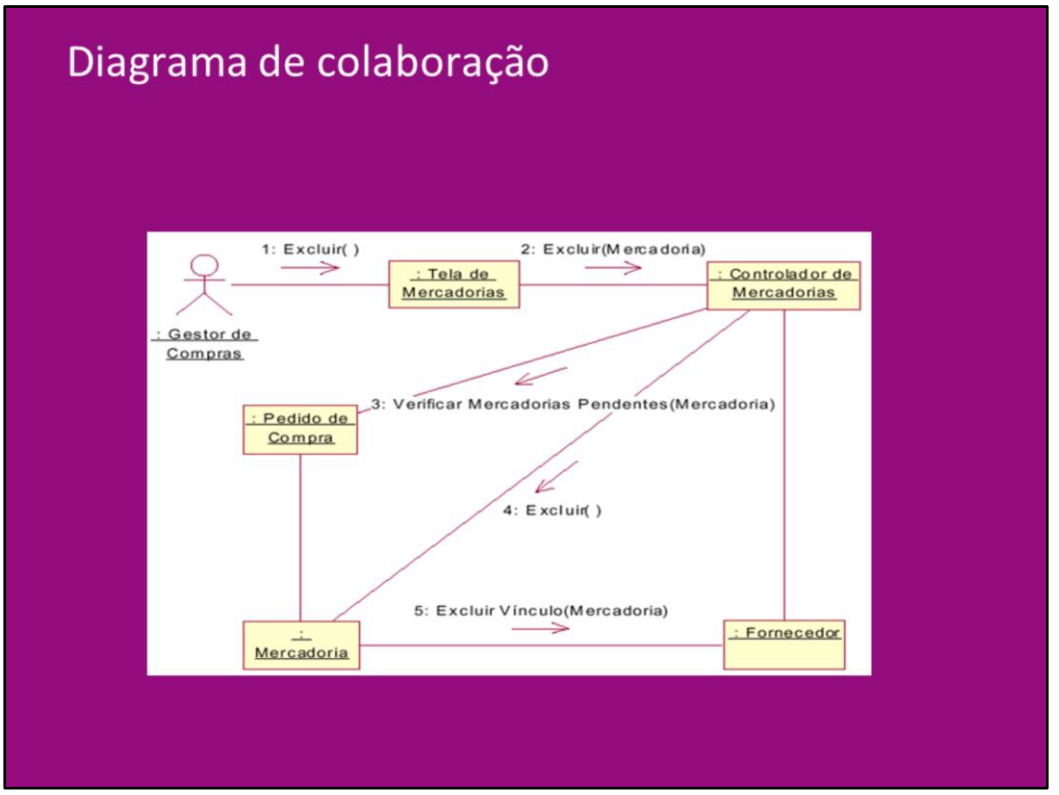


#PraCegoVer
O Diagrama apresenta 4 objetos: “: inicio”, “: objeto1:Classe”, ““: objeto2:Classe””, “: objeto3:Classe”.
Entre os objetos “: inicio” e “: objeto1:Classe” tem a mesagem 1: StartMessage(), e a mesagem é direcionada do “: inicio” para “: objeto1:Classe”.
Entre os objetos “: objeto1:Classe” e “: objeto2:Classe” tem a mesagem 2: [Condition] response:= Message(arg.) e a é caracterizado como <<new>>, e a mesagem é direcionada do “: objeto1:Classe” para “: objeto2:Classe”.
Entre os objetos “: objeto1:Classe” e “: objeto3:Classe” tem a mesagem 3: Message(arguments), e a mesagem é direcionada do “: objeto1:Classe” para “: objeto3:Classe”.

Em dotUML:

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Colaboração Diagram"] {
// Usaremos o mesmo código do diagrama de sequencia

   lifeline "Inicio" as i
lifeline "objeto1: Classe" as o1
lifeline "objeto2: Classe" as o2
lifeline "objeto3: Classe" as o3
```

```
    i --> o1 "1: StartMessage()"
 o1 -c-> o2 "2: [Condition] response:=message(arg.)"
 note left of o2  "<<new>>"
 o1 --> o3 "3: message(arguments)"

}
```



#PraCegoVer
O diagrama tem 6 objetos: “: Gestor de compras”, “:Tela de Mercadorias”, “:Controlador de Mercadorias”, “:Pedido De Compra”, “: Mercadoria”, “: Fornecedor”
Sendo que o objeto “: Gestor de compras” representa o usuário pelo símbolo de ator.
O objeto “: Gestor de compras” manda a mensagem “1: Excluir() “ para o objeto “:Tela de Mercadorias”.
O objeto “:Tela de Mercadorias” manda a mensagem “2: Excluir(Mercadoria) “ para o objeto “:Controlador de Mercadorias”.
O objeto “:Controlador de Mercadorias” manda a mensagem “3: Verifica Mercadorias(Mercadoria) “ para o objeto “:Pedido De Compra”.
O objeto “:Controlador de Mercadorias” manda a mensagem “4: Excluir() “ para o objeto “:Mercadoria”.
O objeto “:Mercadoria “manda a mensagem 5: Excluir Vinculo(Mercadoria) “ para o objeto “:Fornecedor”.
O objeto “:Fornecedor “manda a mensagem de retorno da operação para o objeto “:Controlador de Mercadorias”.

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
```

```
   label="Colaboração Diagram"] {
// Usaremos o mesmo código do diagrama de sequencia

   actor ": Geestor de Compras" as user
lifeline ": Tela de Mercadorias" as tela
lifeline ": Controlador de Mercadorias" as control
lifeline ": Pedido de Compra" as pedido
lifeline ": Mercadoria" as mercadoria
lifeline ": Fornecedor" as fornecedor

   user --> tela "1: Excluir()"
tela --> control "2: Excluir(Mercadoria)"
tela --> pedido "3: Verificar Mercadorias Pendentes
   (Mercadoria)"
note over pedido,mercadoria "Classes Associadas"
tela --> mercadoria "4: Excluir()"
mercadoria --> fornecedor "5: Excluir Vinculo(Mercadoria)"
note over control,fornecedor "Classes Associadas"
}
```

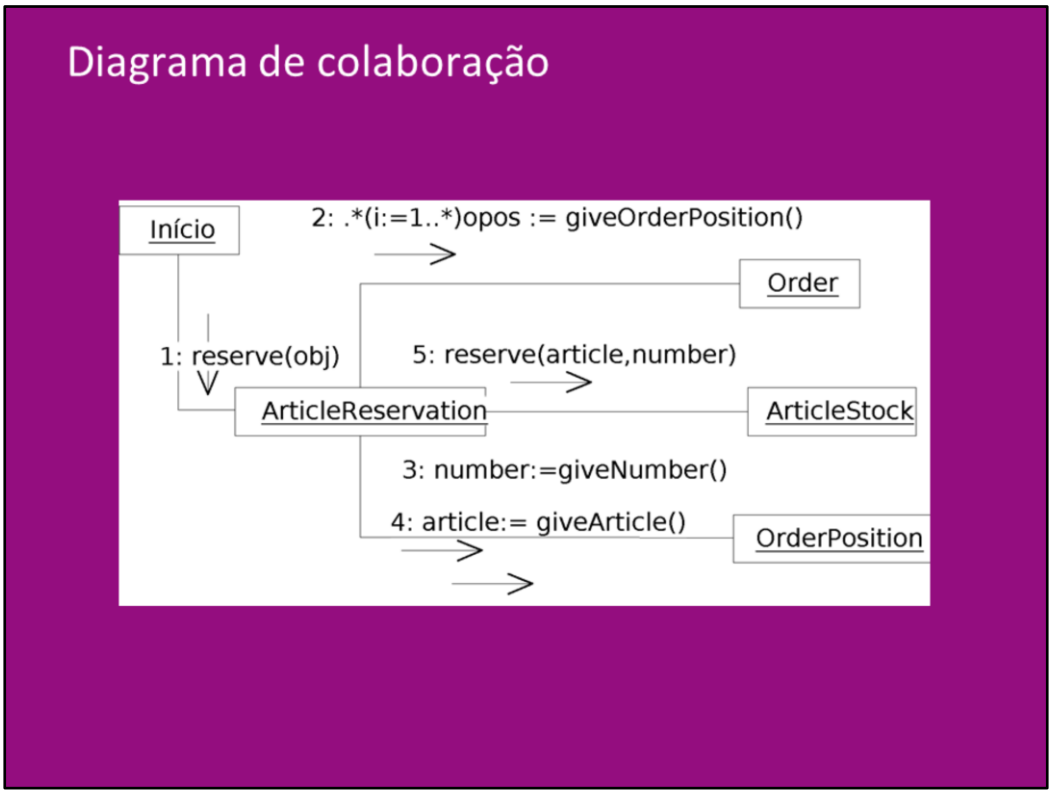
Diagrama de colaboração

As setas de mensagens são desenhadas entre os objetos para mostrar o fluxo de mensagens entre eles.

Nos diagramas de colaboração a ordenação das mensagens é mostrada apenas pela numeração delas.

Pode conter objetos ativos que executam os métodos paralelamente com outros.

Este diagrama é útil sempre que você quer se referir a uma interação particular.



#PraCegoVer
O diagrama tem 5 objetos: Início, ArticleReservation, Order, ArticleStock, OrderPosition
O objeto Início envia a mensagem 1:reserve(obj) para o objeto ArticleReservation
O objeto ArticleReservation envia a mensagem 2:*(i:=1..*)opôs:=giveOrderPosition() para o objeto Order
O objeto ArticleReservation envia a mensagem 3:number:=giveNumber() para o objeto OrderPosition
O objeto ArticleReservation envia a mensagem 4:article:=giveArticle() para o objeto OrderPosition
O objeto ArticleReservation envia a mensagem 5: reserve(article, number) para o objeto ArticleStock

Em dotUML:

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Colaboração Diagram"] {
// Usaremos o mesmo código do diagrama de sequencia

   lifeline "Inicio" as i
lifeline "articleReservation" as a
lifeline "Order" as o
lifeline "articleStock" as s
lifeline "OrderPosition" as op
```

```
 i --> a "1: reserve(obj)"
 a --> o "2: *(i:= 1..*) opos :=giveOrderPosition()"
 a --> op "3: number :=giveNumber()"
 a --> op "4: article :=giveArticle()"
 a --> s "5: reserve(article, number)"
}
```

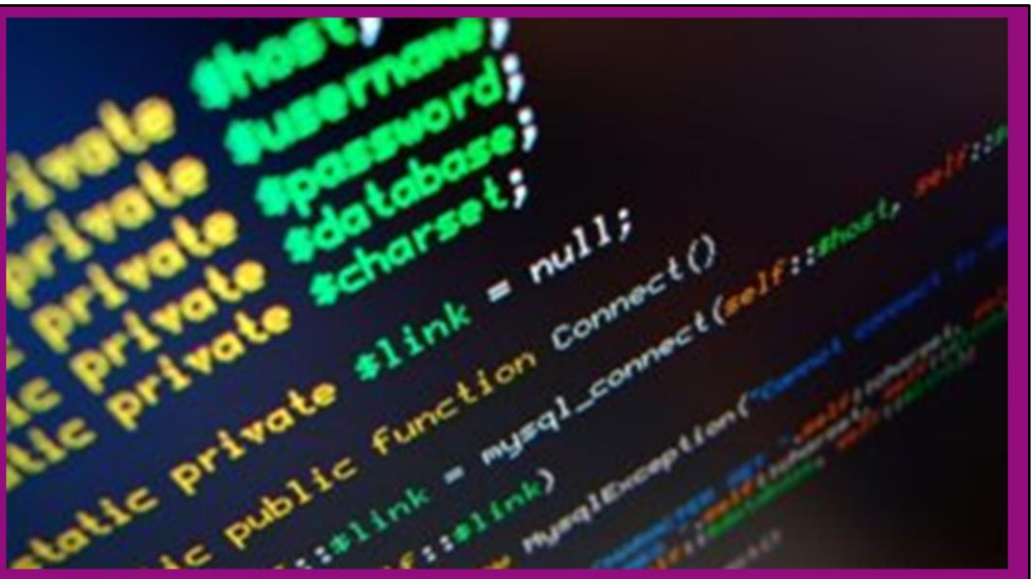
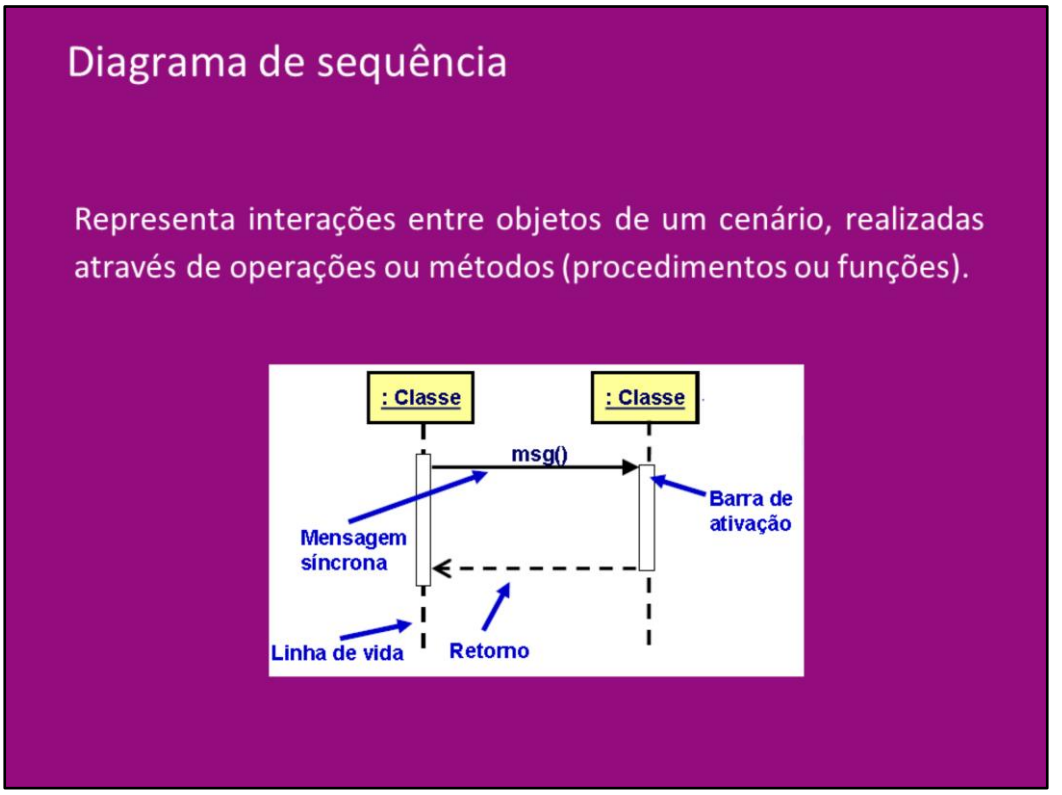


Diagrama de sequencia

Diagrama de sequência

Como um projeto pode ter uma grande quantidade de métodos em classes diferentes, pode ser difícil determinar a sequência global do comportamento. O diagrama de sequência representa essa informação de uma forma simples e lógica.

Ele registra o comportamento de um único caso de uso e exibe os objetos e as mensagens passadas entre esses objetos no caso de uso.



#PraCegoVer
O diagrama de sequencia tem a mesma finalidade do diagrama de colaboração, a diferença é no visual, onde os objetos são dispostos um do lado do outro e cada mensagem entre os objetos fica abaixo uma das outras a primeira mensagem fica no topo de modo que cada nova mensagem fica abaixo, ou seja, a sequencia é vista visualmente de cima para baixo, desse modo não há a necessidade de numerar as mensagens, como no diagrama de colaboração.
No exemplo do diagrama aparece dois objetos da mesma classe: ":Classe" e ": Classe"
Uma linha com uma seta sai de um objeto para outro, essa linha indica que é uma mensagem síncrona,
Essa linha está com um texto: msg(), indicando que a mensagem enviada é msg()
Isso significa que o objeto que recebe a mensagem tem esse método implementado.
De cada objeto sai uma linha tracejada de cima para baixo, chamada de linha de vida. Onde a mensagem toca a linha, essa linha então muda para uma barra, chamada de barra de ativação, que seria o processamento da mensagem.
Quando a mensagem finaliza o processamento, pode-se indicar isso com uma linha tracejada com uma flexa na ponta do objeto que recebeu a mensagem anterior (neste caso msg()) para o objeto que enviou a mensagem anterior.

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
```

```
lifeline ": Classe" as c1 autoactivate
lifeline ": Classe" as c2 autoactivate
c1 --> c2 "msg()"
 note left "mensagem \n síncrona"
note right of c2 "Barra de \n ativação"
c2 -r-> c1

   note left of c2 "Retorno "
note over c1,c2 "Linha de Vida "
}
```

Diagrama de Seqüência

- Interação entre os objetos
- Determina a seqüência de eventos que ocorrem em um determinado processo
  - Quais condições devem ser satisfeitas ...
  - Quais métodos devem ser disparados ...
  - E em qual ordem ...

- Baseia-se no Diagrama de Casos de Uso
  - 1 Caso de Uso N Diagramas de Seqüência
- Baseia-se, também, no Diagrama de Classes
  - Fornecem as classes e os métodos associados

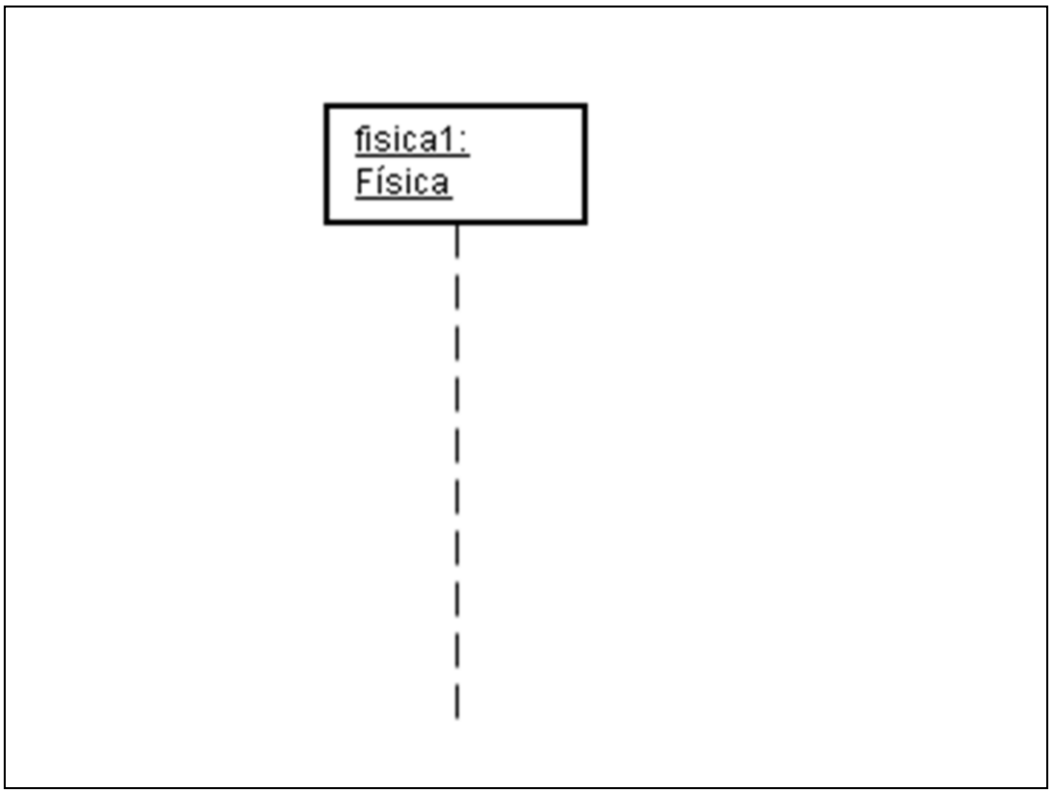
Diagrama de Seqüência

- **Componentes - ATORES**

- Exatamente os mesmos dos Casos de Uso
- Interagem , Solicitam serviços , Eventos , Processos
- Não são obrigatórios no Diagrama de Seqüência

Diagrama de Seqüência

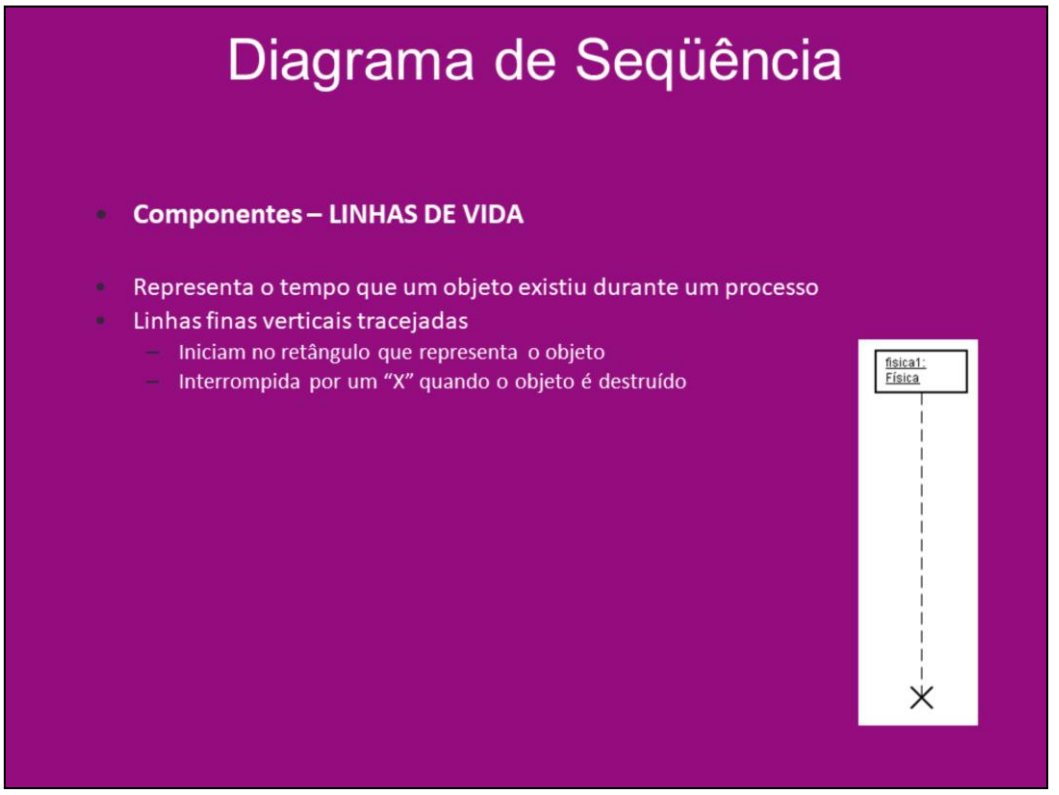
- **Componentes - OBJETOS**
  - Representam as instâncias das classes
  - Retângulos contendo um texto
    - Primeira parte, em minúsculo, o nome do objeto
    - Segunda parte, em letras iniciais maiúsculas, o nome da classe
    - Informações separadas por dois pontos (:)
  - Linha de vida
    - Linha vertical tracejada



#PraCegoVer
Exemplo de um objeto com linha de vida
Neste exemplo o objeto é fisica1: Física
A Figura mostra um retângulo com uma linha tracejada saindo na parte de baixo.
Dentro do retângulo é descrito o nome do objeto fisica1: Física

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "fisica1: Fisica" as f1 autoactivate

   }
```



#PraCegoVer
Exemplo de um objeto com linha de vida
Neste exemplo o objeto é fisica1: Física
A Figura mostra um retângulo com uma linha tracejada saindo na parte de baixo.
Dentro do retângulo é descrito o nome do objeto fisica1: Física
No Fim da linha tracejada (na parte de baixo) tem um X indicando o fim da linha de vida do objeto.

Em dotUML

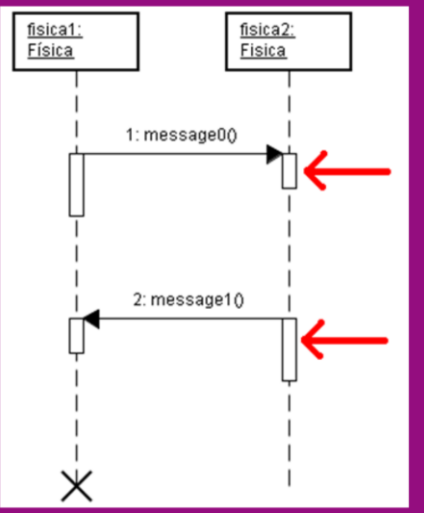
```
SequenceDiagram [frame=true
  framecolor=steelblue
  label="Sequence Diagram"] {
lifeline "fisica1: Fisica" as f1
  autoactivate
}
```

Diagrama de Seqüência

- **Componentes – FOCO DE CONTROLE/ATIVAÇÃO**
- Indica os períodos em que um determinado objeto está participando ativamente do processo
  - Executando um ou mais métodos do processo
- Representados por extensões mais grossas/largas da Linha de Vida

Diagrama de Seqüência

- **Componentes – FOCO DE CONTROLE/ATIVAÇÃO**



#PraCegoVer
Há dois objetos: fisica1: Fisica, fisica2:Fisica
O objeto fisica1: Fisica envia uma mensagem 1: menssage0() para o objeto fisica2: Fisica, onde a mensagem inicia a barra de vida é ativada.
O objeto fisica2: Fisica envia uma mensagem 2: menssage1() para o objeto fisica1: Fisica, onde a mensagem inicia a barra de vida é ativada.

Em dotUML

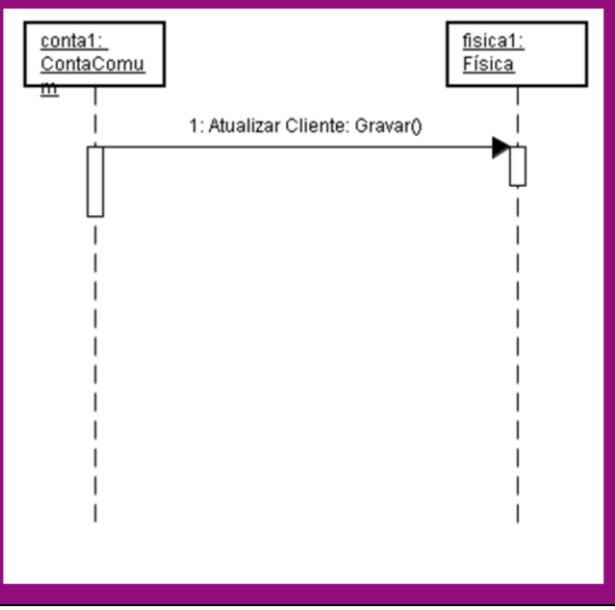
```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "fisica1: Fisica" as f1 autoactivate
lifeline "fisica2: Fisica" as f2 autoactivate

   f1 --> f2  "1: message0()"
f1 --> f2  "2: message1()"
}
```

Diagrama de Seqüência

- **Componentes – MENSAGENS/ESTÍMULOS**
- Demonstram a ocorrência de eventos que normalmente forçam a chamada de um método em algum dos objetos envolvidos no processo
- Mensagens entre:
  - Ator e Ator
  - Ator e Objeto
  - Objeto e Objeto
  - Objeto e Ator

Diagrama de Seqüência

- **Mensagem com disparo de método entre objetos.**



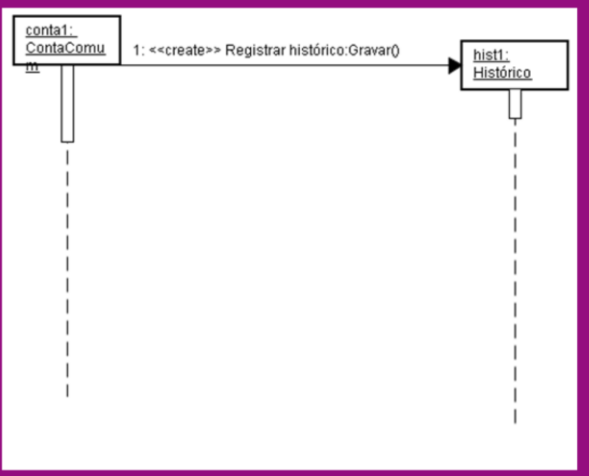
#PraCegoVer
Tem dois objetos: conta1: ContaComum, fisica1: Fisica
O objeto conta1: ContaComum envia uma mensagem 1: Atualizar Cliente: Gravar() para o objeto fisica1: Fisica, onde a mensagem inicia a barra de vida é ativada.

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "conta1: ContaComum" as c autoactivate
lifeline "fisica1: Fisica" as f autoactivate

   c --> f "1: Atualizar Cliente: Gravar()"
}
```

Diagrama de Seqüência

- **Mensagem que instancia um novo objeto**



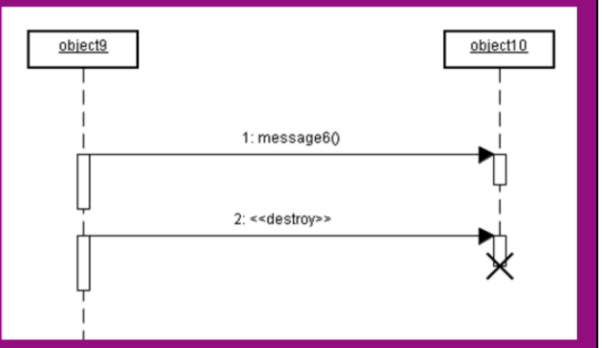
#PraCegoVer
Tem dois objetos: conta1: ContaComum, hist1: Histórico
O objeto conta1: ContaComum envia uma mensagem 1: <<create>> Registrar histórico : Gravar () para o objeto hist1: Histórico, onde o objeto irá aparecer e iniciar com a barra de vida é ativada.

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "conta1: ContaComum" as c autoactivate
lifeline "hist1: Historico" as h autoactivate

   c -c-> h "1:<<create>> Registrar histórico Gravar()"
}
```

Diagrama de Seqüência

- **Mensagem que dispara um método destrutor – elimina um objeto não mais necessário.**



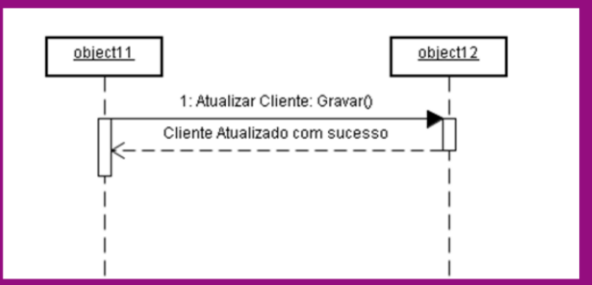
#PraCegoVer
Tem dois objetos: objeto9, objeto10
O objeto objeto9envia uma mensagem 1: message6() para o objeto objeto10, onde a mensagem inicia a barra de vida é ativada..
O objeto objeto9envia uma mensagem 2: <<destroy>>para o objeto objeto10, onde a mensagem inicia a barra de vida é ativada e logo em seguida tem um X indicando o fim do objeto10.

Em dotUML:

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "Objeto9" as o9 autoactivate
lifeline "Objeto10" as o10 autoactivate

   o9 -c-> o10 "1:  menssage6()"
o9 -d-> o10 "2: <<destroy>>"

   }
```

Diagrama de Seqüência

- **Mensagem de Retorno ... Linha tracejada.**
- **Podem retornar valores ou status...**



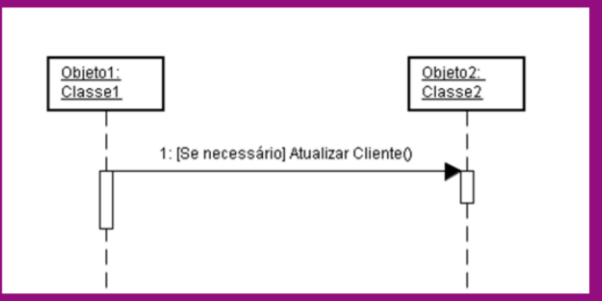
#PraCegoVer
Tem dois objetos: objeto11, objeto12
O objeto objeto11envia uma mensagem 1: Atualiza Cliente: Gravar() para o objeto objeto12, onde a mensagem inicia a barra de vida é ativada.
Em seguida o objeto12 envia um retorno indicado por uma linha tracejada, informando que Cliente Atualizado com sucesso para o objeto11,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto12.

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "Objeto11" as o11 autoactivate
lifeline "Objeto12" as o12 autoactivate

   o11 --> o12 "1:  Atualizar Cliente: Gravar()"
o12 -r-> o11 "Cliente Atualizado com sucesso"

   }
```

Diagrama de Seqüência

- **Mensagem com Condição de Guarda**
- **Entre colchetes [ ]**

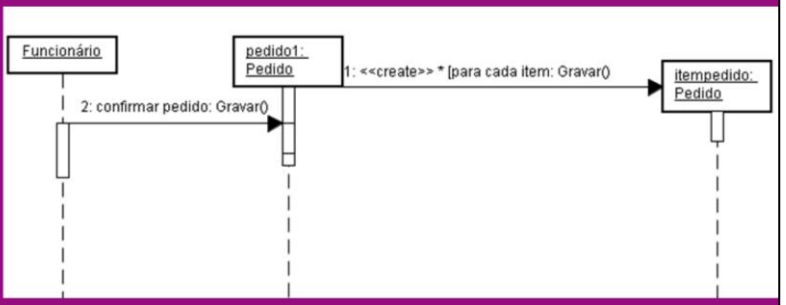


#PraCegoVer
Tem dois objetos: objeto1: Classe1, objeto2: Classe2
O objeto objeto1: Classe1 envia uma mensagem 1: [Se necessário] Atualiza Cliente() para o objeto objeto2: Classe2, onde a mensagem inicia a barra de vida é ativada.

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "Objeto1: Classe1" as o1 autoactivate
lifeline "Objeto2: Classe2" as o2 autoactivate

   o1 --> o2 "1: [Se necessário] Atualizar Cliente()"

   }
```

Diagrama de Seqüência

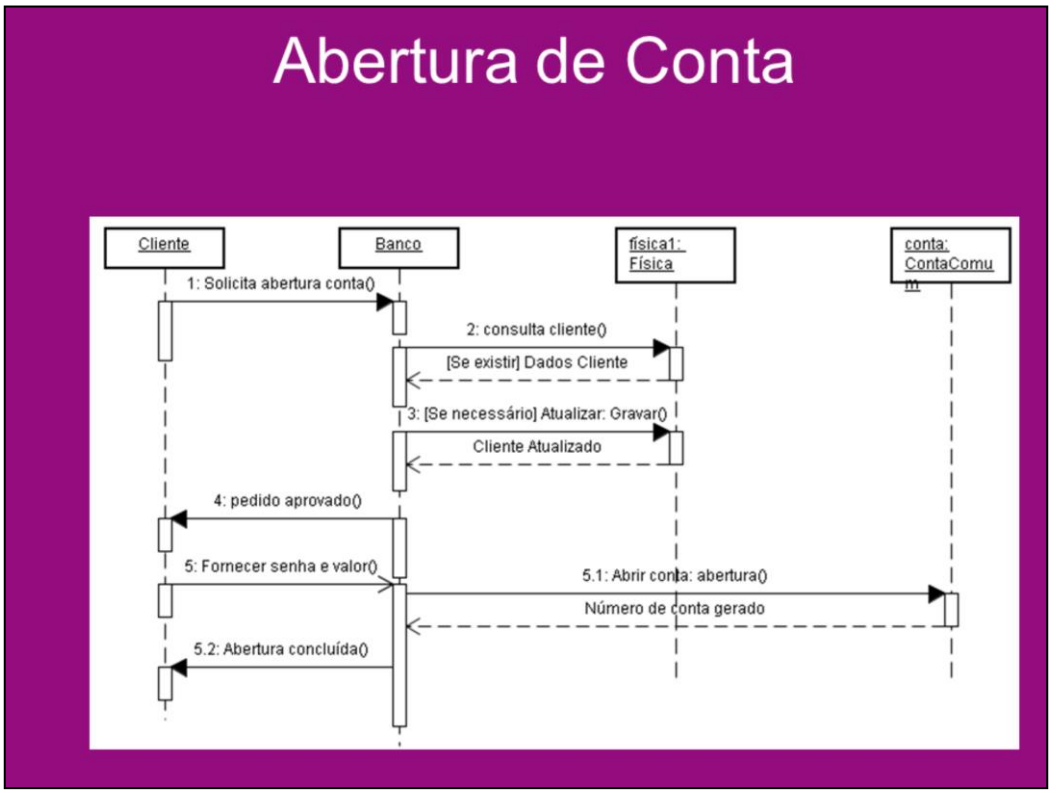


- **Mensagem com Condição de Guarda**
- **Disparo de mensagem a vários objetos (*)**

#PraCegoVer
Tem três objetos: Funcionario, pedido1: Pedido, itemPedido: Pedido
O objeto pedido1: Pedido envia uma mensagem 1: <<create>> *[para cada item]: Gravar() para o objeto itempedido: Pedido, onde o objeto itempedido: Pedido é inciado.
O objeto Funcionario envia uma mensagem 2: confirmar pedido: Gravar() para o objeto pedido1: Pedido, onde a mensagem inicia a barra de vida é ativada.

Em dotUML:

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "Funcionario" as f autoactivate
lifeline "pedido1: Pedido" as p1 autoactivate
lifeline "itemPedido: Pedido" as i autoactivate
p1 -c-> i "1. <<create>> *[Para cada item]: Gravar()"
activate f
f --> p1 "2. Confirmar pedido: Gravar()"
}
```

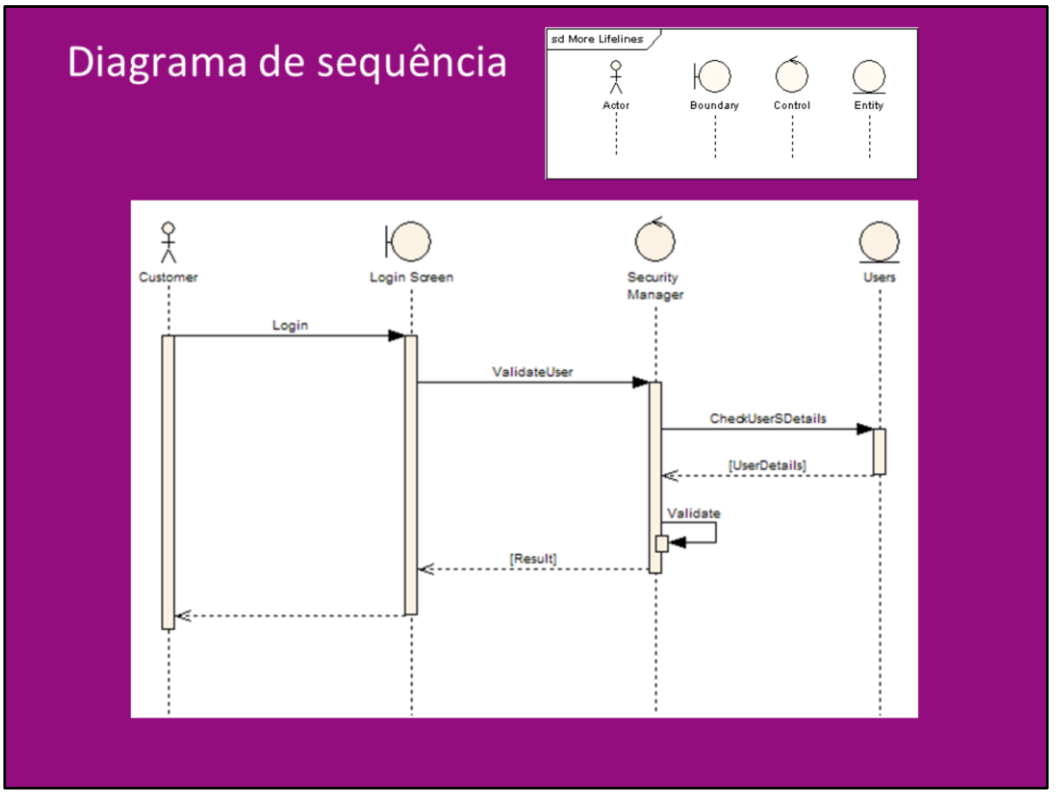


#PraCegoVer
Tem quatro objetos: Cliente, Banco, fisica1: Fisica, conta: ContaComum
O objeto cliente envia uma mensagem 1: Solicita abertura conta() para o objeto banco, onde a mensagem inicia a barra de vida é ativada.
O objeto banco envia uma mensagem 2: consulta cliente() para o objeto fisica1: Fisica, onde a mensagem inicia a barra de vida é ativada.
Em seguida o fisica1: Fisica envia um retorno indicado por uma linha tracejada, informando que [Se existir] Dados cliente para o objeto banco,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto fisica1: Fisica.
O objeto banco envia uma mensagem 3: [Se necessário]Atualizar: Gravar() para o objeto fisica1: Fisica, onde a mensagem inicia a barra de vida é ativada.
Em seguida o fisica1: Fisica envia um retorno indicado por uma linha tracejada, informando que Cliente Atualizado para o objeto banco,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto fisica1: Fisica.
O objeto banco envia uma mensagem 4: pedido aprovado() para o objeto cliente, onde a mensagem inicia a barra de vida é ativada.
O objeto cliente envia uma mensagem 5: Fornecer senha e valor() para o objeto banco, onde a mensagem inicia a barra de vida é ativada.
O objeto banco envia uma mensagem 5.1: Abrir conta: abertura() para o conta: ContaComun, onde a mensagem inicia a barra de vida é ativada.
Em seguida o conta: ContaComun envia um retorno indicado por uma linha

tracejada, informando que Número de conta gerado para o objeto banco,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto conta: ContaComun.
O objeto banco envia uma mensagem 5.2: Abertura concluída() para o objeto cliente, onde a mensagem inicia a barra de vida é ativada.

Em dotUML:

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
lifeline "Cliente" as c autoactivate
lifeline "Banco" as b autoactivate
lifeline "fisica: Fisica" as fisica autoactivate
lifeline "conta: ContaComum" as conta autoactivate

   c --> b "1: Solicita abertura conta()"
b --> fisica "2: consulta cliente() "
fisica -r-> b "[Se existir] Dados Cliente"
b --> fisica "3: [Se necessário] Atualizar : Gravar() "
fisica -r-> b "Cliente Atualizado"

   b --> c "4: pedido aprovado()"
c --> b "5: Fornecer senha e valor()"
b --> conta "5.1: Abrir conta: abertura()"
conta -r-> b "Número de conta gerado"
b --> c "5.2: Abertura concluída()"
}
```



#PraCegoVer
São apresentadas duas figuras, a primeira para mostrar os estereótipos das classes (Objetos): Actor (Ator), boundary (limite), Control (Controle), Entity (entidade).
Onde o boneco palito representa os atores do sistema (actor)
A boundary é representado circulo com um traço lateral (ou seja, por uma linha na vertical ligada a um circulo) , representando o limite do sistema, ou seja a interface que o usuário vai interagir.
O controle (Control) é representado por um circulo com uma flexa, é onde o processamento do sistema vai ser realizado, por exemplo, onde tem o método main no Java.
As entidades (entity) é representado por um circulo com um traço embaixo, é o objeto que representa as classes que irão armazenar dados, como por exemplo os dados dos alunos.

A segunda figura é um exemplo do diagrama de sequencia:
Onde temos quatro objetos: Customer, LoginScreen, Security Manager, Users
O Customer é representado pelo boneco palito, ou seja um ator.
LoginScreen é representado pelo circulo com um traço lateral, ou seja um objeto de limite
Security Manager é representado por um circulo com uma flexa, ou seja um objeto de controle
Users é representado por um circulo com um traço embaixo, ou seja um objeto de entidade.

Agora as sequencia das mensagens:
O objeto Customer envia uma mensagem: Login para o objeto LoginScreen, onde a mensagem inicia a barra de vida é ativada.
O objeto LoginScreen envia uma mensagem validaUser para o objeto Security Manager , onde a mensagem inicia a barra de vida é ativada.
O objeto Security Manager envia uma mensagem CheckUserSDetails para o objeto Users, onde a mensagem inicia a barra de vida é ativada.
Em seguida o objeto Users envia um retorno indicado por uma linha tracejada, informando que [UserDetails] para o objeto Security Manager ,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto Users.
O objeto Security Manager envia uma mensagem validate para o objeto Security Manager (isso mesmo, envia uma mensagem para si mesmo), onde a mensagem inicia a barra de vida é ativada em cima da barra de vida que já estava ativada, então temos barras sobrepostas.
Em seguida o Security Manager envia um retorno indicado por uma linha tracejada, informando que [Result] para o objeto LoginScreen,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto Security Manager (as duas barras terminam aqui).
Em seguida o LoginScreen envia um retorno indicado por uma linha tracejada, para o objeto Customer (neste caso não foi colocado nenhuma informação),
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto LoginScreen.

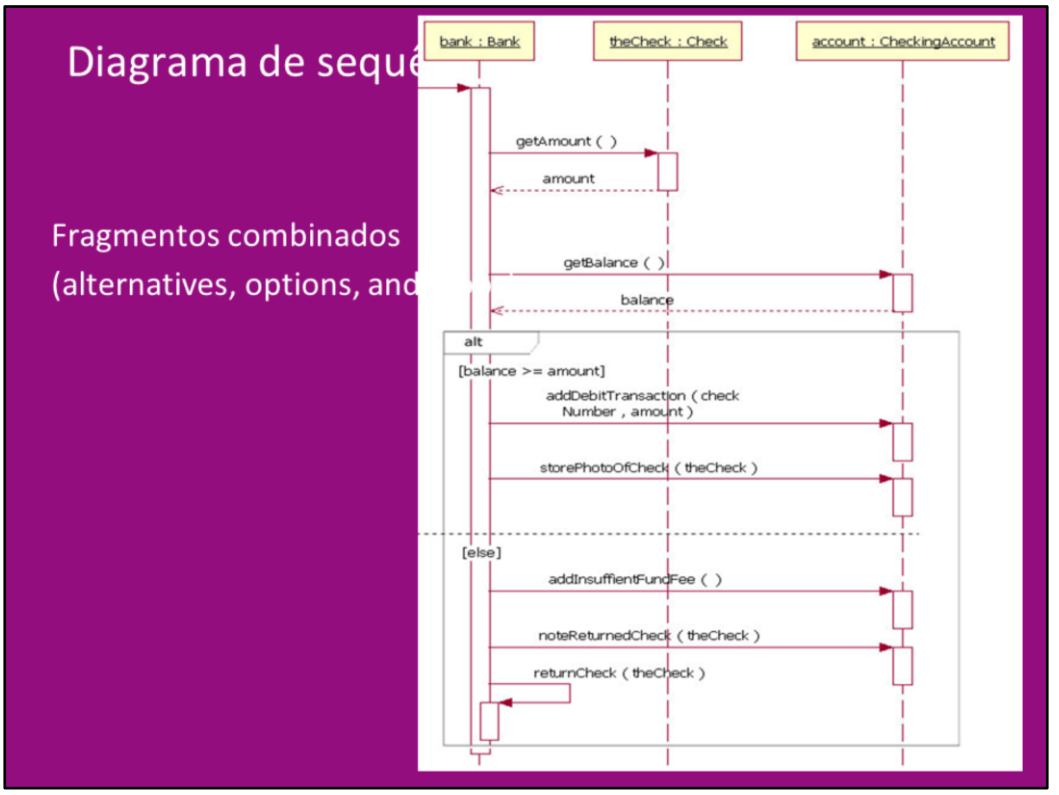
A primeira figura em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
actor "Customer" as customer
boundary "Login Screen" as login
control "Security Manager" as  seg
entity "Users" as users
}
```

A segunda figura em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
actor "Customer" as customer autoactivate
boundary "Login Screen" as login autoactivate
control "Security Manager" as  seg autoactivate
entity "Users" as users autoactivate
customer --> login "Login"
activate customer
login --> seg "ValidateUser"
seg --> users "CheckUsersDetails"
users -r-> seg "[UserDetails]"
```

```
seg --> seg "Validate"
seg -r-> login "[Result]"
login -r-> customer
deactivate customer
}
```



#PraCegoVer
É mostrado parte de um diagrama de sequencia para ilustras a inserção de fragmentos

O objeto envia uma mensagem getAmount() para o objeto , onde a mensagem inicia a barra de vida é ativada.
Em seguida o objeto envia um retorno indicado por uma linha tracejada, informando que amount para o objeto ,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto .

O objeto envia uma mensagem getBalance() para o objeto , onde a mensagem inicia a barra de vida é ativada.
Em seguida o objeto Users envia um retorno indicado por uma linha tracejada, informando que balance para o objeto ,
Nesse momento é desativada a barra de vida, voltando a ser uma linha tracejada no objeto .

Inicia o Fragmento alt:
Parte 1 do fragmento alt: é executado se: [balance >= amount] (ou seja, é executado se: [balance >= amount]
O objeto envia uma mensagem addDebitTransaction (check number, amount) para o objeto , onde a mensagem inicia a barra

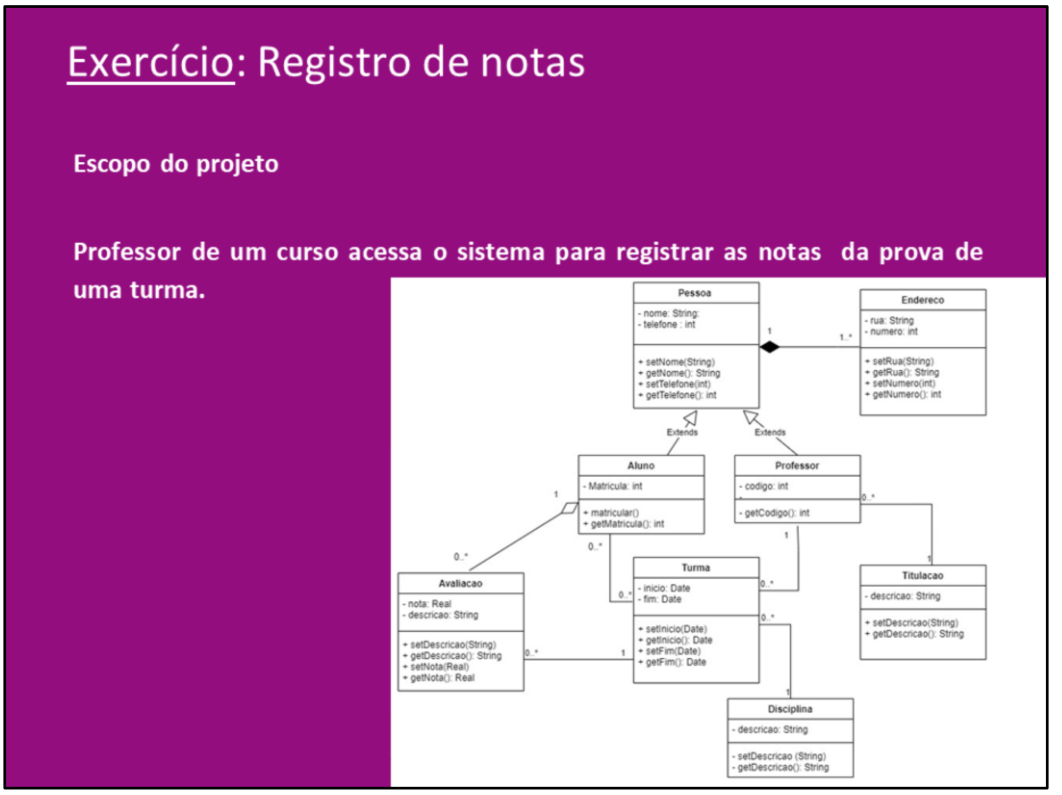
de vida é ativada.
O objeto bank: Bank envia uma mensagem storePhotoOfCheck(theCheck) para o objeto account : CheckingAccount , onde a mensagem inicia a barra de vida é ativada.

Fim da parte 1
Inicio da Parte 2 do Fragmento alt:
[Else]
O objeto bank: Bank envia uma mensagem addInsuffientFundFee() para o objeto account : CheckingAccount , onde a mensagem inicia a barra de vida é ativada.
O objeto bank: Bank envia uma mensagem noteReturnedCheck(the Check) para o objeto account : CheckingAccount , onde a mensagem inicia a barra de vida é ativada.
O objeto bank: Bank envia uma mensagem returnedCheck(the Check) para o objeto Bank (isso mesmo, uma mensagem para si mesmo), onde a mensagem inicia a barra de vida é ativada, neste caso uma barra sobre a outra.
Fim fragmento alt

Em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {

    lifeline "bank : Bank" as bank autoactivate
 lifeline "theCheck : Check" as check autoactivate
 lifeline "account : CheckingAccount" as account
   autoactivate

 bank --> check "getAmount()"
 activate bank
 check -r-> bank  "amount"
 bank --> account "getBalance()"
account -r-> bank "balance"
fragment alt "[balance >=amount]" {
bank --> account "addDebitTransaction (check Number,
   amount)"
bank --> account "storePhotoOfCheck(thecheck)"
deactivate account
case "else"
bank --> account "addInsuffientFundFee ()"
bank --> account "noteReturnedCheck (thecheck)"
deactivate account
bank --> bank "returnCheck(theCheck)"
deactivate bank
}
}
```

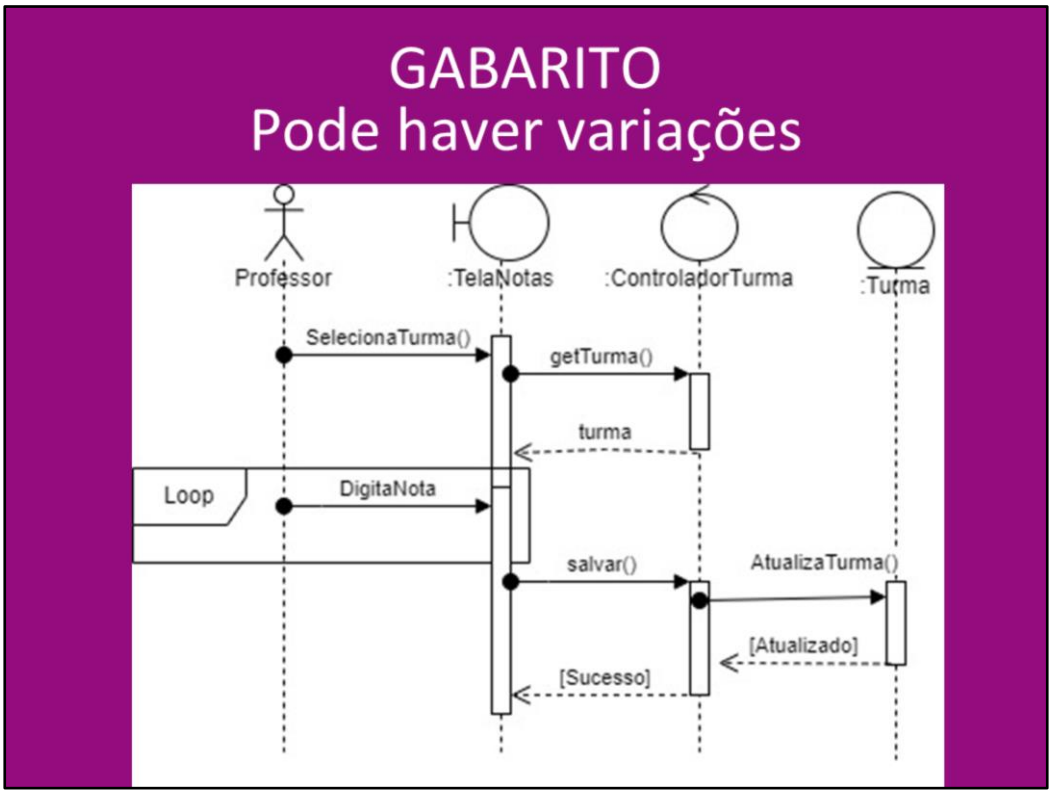
Vamos trabalhar?



#PraCegoVer
O diagrama apresenta 8 classes: Pessoa, Endereco, Aluno, Professor, Turma, Disciplina, Titulacao, avaliacao
As classes Pessoa e Endereco estão relacionados por meio de composição, onde o endereco é parte do cadastro do Pessoa.
O lado do losango fica do lado da classe Pessoa, sendo 1 a cardinalidade do lado da classe Pessoa e cardinalidade 1..* do lado do endereco.
As classes Pessoa e Aluno estão relacionados por meio de herença, Pessoa é a superclasse e o Aluno é a subclasse
As classes Pessoa e Professor estão relacionados por meio de herença, Pessoa é a superclasse e o Professor é a subclasse.
As classes Turma e Aluno estão relacionados por meio de associação, sendo 0..* a cardinalidade do lado da classe Aluno e cardinalidade 0..* do lado do Turma.
As classes Avaliacao e Aluno estão relacionados por meio de agregacao, onde o Avaliacao é parte do cadastro de Aluno.
O lado do losango fica do lado da classe Aluno,sendo 0..* a cardinalidade do lado da classe Avaliacao e cardinalidade 1 do lado do Aluno.
As classes Turma e Professor estão relacionados por meio de associação, sendo 1 a cardinalidade do lado da classe Professor e cardinalidade 0..* do lado do Turma.
As classes Turma e Disciplina estão relacionados por meio de associação, sendo 1 a cardinalidade do lado da classe Disciplina e cardinalidade 0..* do lado do Turma.
As classes Professor e Titulacao estão relacionados por meio de associação, sendo 0..* a cardinalidade do lado da classe Professor e cardinalidade 1 do lado da

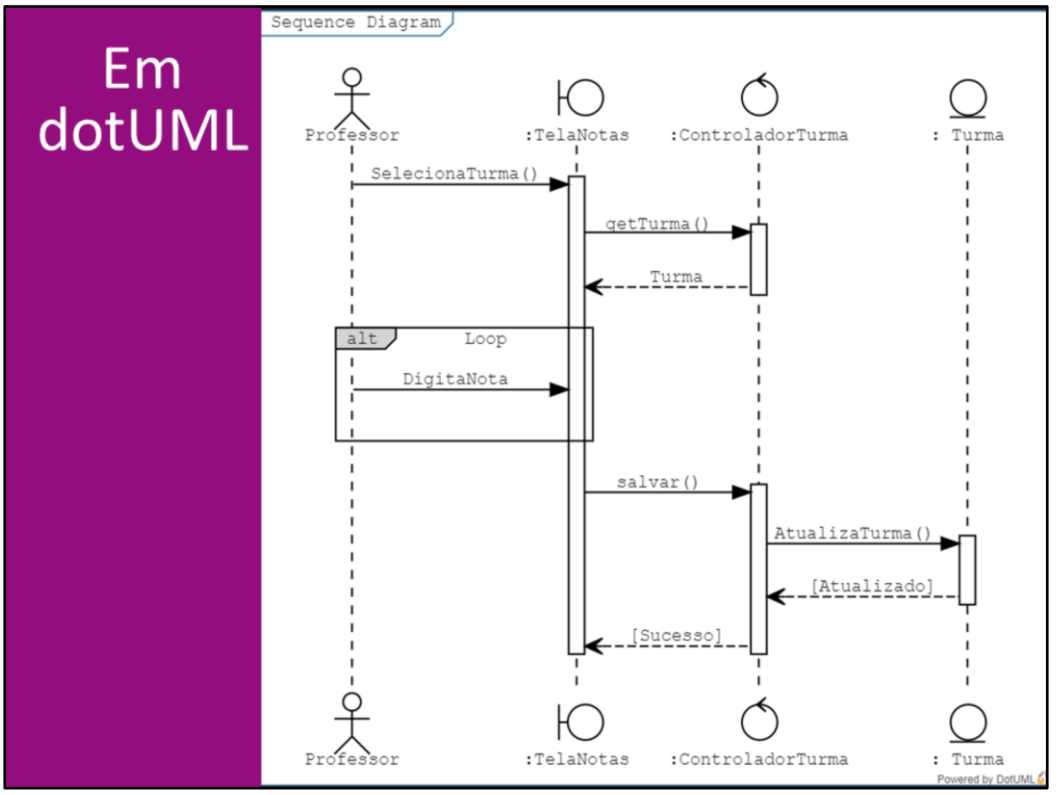
Titulacao.

A classe Pessoa contêm dois atributos:- nome: String, - telefone : int
A classe Pessoa contêm quatro operações: + setNome(String), + getNome(): String,+ setTelefone(int),+ getTelefone(): int
A classe Endereco contêm dois atributos:- rua: String, - numero: int
A classe Endereco contêm quatro operações: + setRua(String), + getRua(): String,+ setNumero(int),+ getNumero(): int
A classe Turma contêm dois atributos:- inicio: Date, - fim: Date
A classe Turma contêm quatro operações: + setInicio(Date),+ getInicio(): Date, + setFim(Date), + getFim(): Date
A classe Avaliacao contêm dois atributos:- nota: Real, - descricao: String
A classe Avaliacao contêm quatro operações: + setDescricao(String),+ getDescricao(): String,+ setNota(Real), + getNota(): Real
A classe Aluno contêm um atributo: - Matricula: int
A classe Aluno contêm duas operações: + matricular(), + getMatricula(): int
A classe Professor contêm um atributo:- codigo: int
A classe Professor contêm uma operação: - getCodigo(): int
A classe Disciplina contêm um atributo:- descricao: String
A classe Disciplina contêm duas operações: - setDescricao (String), - getDescricao(): String
A classe Titulacao contêm um atributo:- descricao: String
A classe Titulacao contêm duas operações: + setDescricao(String), + getDescricao(): String



Gabarito em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
  actor Professor
  boundary  ":TelaNotas" as tela  autoactivate //Se não
   quiser especificar o limite trocar boundary para lifeline
  control  ":ControladorTurma" as controle autoactivate
   //Se não quiser especificar o controle trocar boundary
   para lifeline
  entity ": Turma" as turma autoactivate //Se não quiser
   especificar como entidade  trocar entity para lifeline
  Professor --> tela "SelecionaTurma()"
  tela --> controle "getTurma()"
  controle -r-> tela "Turma"
  fragment alt "Loop" {
  Professor --> tela "DigitaNota"
  }
  tela --> controle "salvar()"
  controle --> turma "AtualizaTurma()"
  turma -r-> controle "[Atualizado]"
  controle -r-> tela "[Sucesso]"
  deactivate tela
}
```



Gabarito em dotUML

```
SequenceDiagram [frame=true framecolor=steelblue
   label="Sequence Diagram"] {
  actor Professor
  boundary  ":TelaNotas" as tela  autoactivate //Se não
   quiser especificar o limite trocar boundary para lifeline
  control  ":ControladorTurma" as controle autoactivate
   //Se não quiser especificar o controle trocar boundary
   para lifeline
  entity ": Turma" as turma autoactivate //Se não quiser
   especificar como entidade  trocar entity para lifeline
  Professor --> tela "SelecionaTurma()"
  tela --> controle "getTurma()"
  controle -r-> tela "Turma"
  fragment alt "Loop" {
  Professor --> tela "DigitaNota"
  }
  tela --> controle "salvar()"
  controle --> turma "AtualizaTurma()"
  turma -r-> controle "[Atualizado]"
  controle -r-> tela "[Sucesso]"
  deactivate tela
}
```

Referência bibliográfica

FOWLER, Martin e SCOTT, Kendall. Uml Essencial. 2a. Edição. Bookman. Porto Alegre, 2000.

SCHNEIDER, Geri. Applying use case: a practical guide. Addison-Wesley, 1998.

OESTEREICH, Bernd. Developing Software with UML. Addison-Wesley, 1999.


ânima
EDUCAÇÃO
UNISUL
CRÉDITOS
COORDENAÇÃO
Vera Rejane Niedersberg Schuhmacher
PROFESSORES
Rafael Lessa
Daniella Vieira