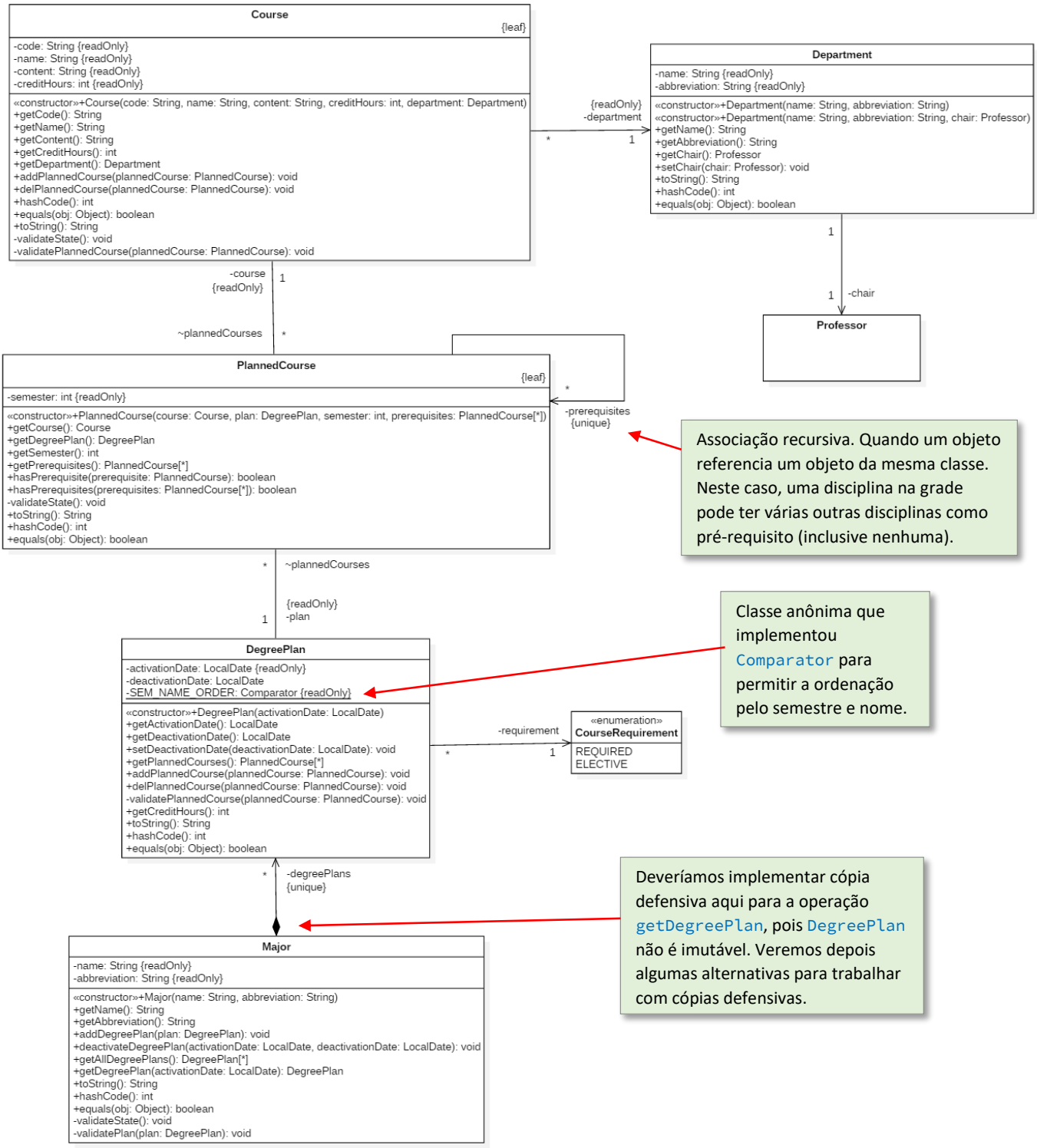
Roteiro 09 – Atividade 01/03

1. Se você executou a última atividade do roteiro 08, sua modelagem deve ter ficado similar à imagem abaixo.



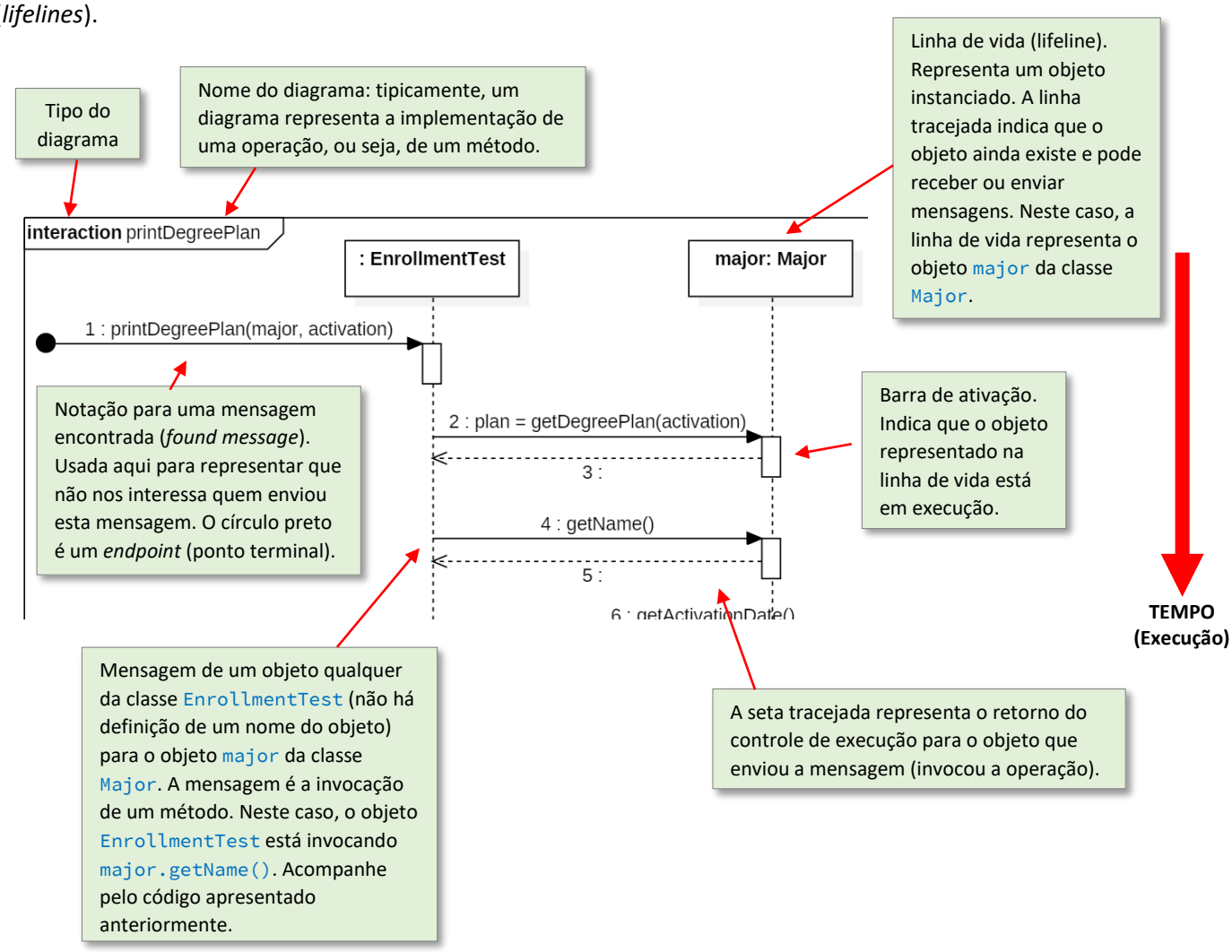
2. Considerando que você também implementou os testes solicitados, você deve ter criado um método para imprimir a grade curricular, ordenando as disciplinas pelo semestre em que elas são oferecidas e pelo código dentro do semestre. Algo similar ao apresentado na próxima página. No arquivo **EnrollmentTest.java**, disponibilizado neste roteiro, você poderá verificar a classe de teste criada para o roteiro anterior.

Roteiro 09 – Atividade 01/03

```java
/**
 * Imprime uma grade curricular.
 *
 * @param major o curso que tem a grade curricular
 */
private void printDegreePlan(Major major, LocalDate activation) {
    DegreePlan plan = major.getDegreePlan(activation);
    System.out.println(major.getName());
    System.out.println("-----------------------------------------");
    System.out.println("degree plan " + plan.getActivationDate());
    System.out.println(plan.getCreditHours() + " credits hours");
    System.out.println("---");
    for (PlannedCourse pc : plan.getPlannedCourses()) {
        System.out.print(pc.getCourse().getCode() + "."
                + pc.getCourse().getName() + ", " + pc.getSemester()
                + " [");
        for (PlannedCourse pr : pc.getPrerequisites()) {
            System.out.print(pr.getCourse().getCode());
        }
        System.out.println("]");
    }
    System.out.println("---");
}
```

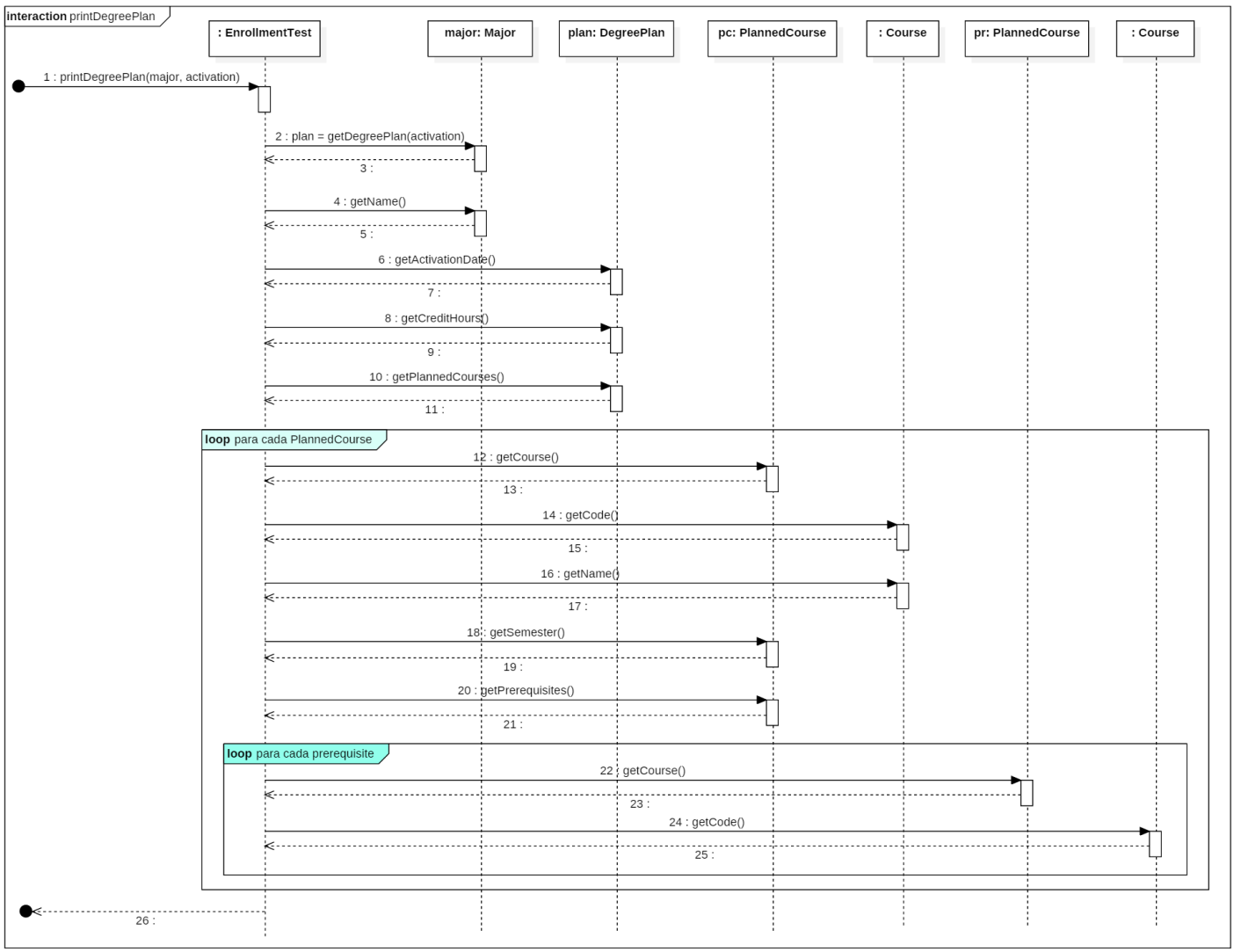
```
run:
Ciência da Computação
-----------------------------------------
degree plan 2017-01-01
52 credits hours
---
ALG.Algoritmos, 1 []
C1.Cálculo, 1 []
C2.Cálculo, 2 [C1]
ED.Estrutura de Dados, 2 [ALG]
POO.Programação Orientada a Objetos, 2 [ALG]
BD1.Banco de Dados, 3 [EDPOO]
ES1.Engenharia de Software, 3 [EDPOO]
MD.Matemática Discreta, 3 [ALG]
BD2.Banco de Dados, 4 [BD1]
ES2.Engenharia de Software, 4 [ES1]
TOP.Tópicos em Computação, 4 []
TCC1.Trabalho de Conclusão de Curso, 5 [ES2BD2]
TCC2.Trabalho de Conclusão de Curso, 6 [TCC1]
---
```

Utilizaremos este código como o exemplo para construirmos nosso primeiro diagrama de sequência. Um **diagrama de sequência** é o tipo mais comum de **diagrama de interação** da UML, o qual descreve uma interação por meio da sequência de mensagens que são trocadas entre os objetos. Os objetos são representados na forma de linhas de vida (*lifelines*).

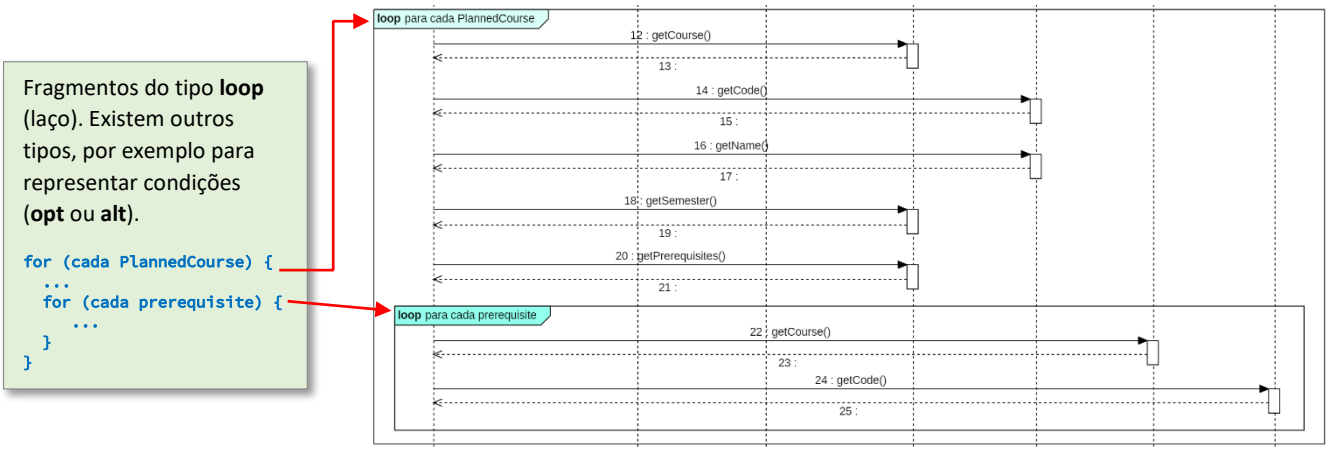


O diagrama de sequência permite que modelemos o algoritmo do método. Ele complementa o diagrama de classe (visão estática) com uma **visão comportamental**. A versão completa do diagrama pode ser vista a seguir.

Roteiro 09 – Atividade 01/03



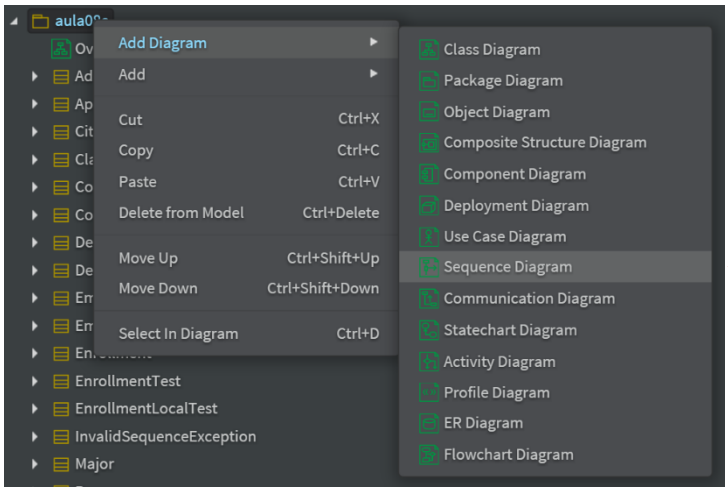
Compare o diagrama com o método `printDegreePlan(Major major, LocalDate activation)` da classe `EnrollmentTest`. Observe que as mensagens seguem exatamente a sequência dos métodos invocados pelos objetos que fazem parte da interação. Os dois quadros (*loop*) são chamados de fragmentos na UML e permitem a representação de laços e condições. Como um está dentro do outro, representam laços aninhados.

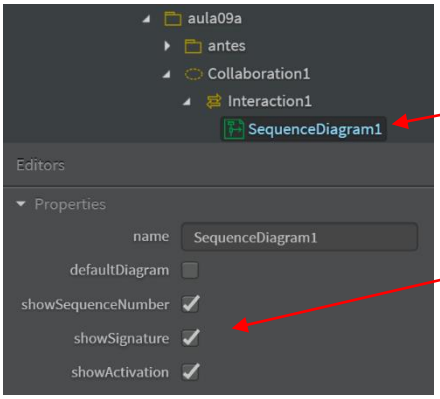


Agora, vamos desenhar este diagrama passo a passo na ferramenta CASE StarUML. Você pode adaptar as orientações a partir da sua própria implementação, uma vez que o diagrama será desenhado considerando que você tem um diagrama de classe com estas operações. As mensagens serão definidas a partir das operações definidas em cada classe.

Roteiro 09 – Atividade 01/03

3. Na ferramenta StarUML, clique com o botão direito sobre um modelo ou pacote existente (ex: utilize aquele em que seu diagrama de classe está modelado) e selecione a opção **[Add Diagram | Sequence Diagram]**.

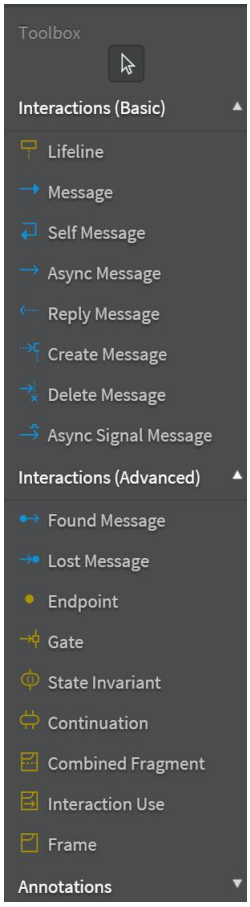




Você pode renomear. Um diagrama de sequência é um tipo de interação que, por sua vez, é uma representação da colaboração entre objetos.

Configuração da visualização (liga ou desliga):

1. Mostrar o número de sequência das mensagens
2. Mostrar a assinatura das operações (mensagens)
3. Mostrar a barra de ativação (execução)



Toolbox especializada para o diagrama de sequência.

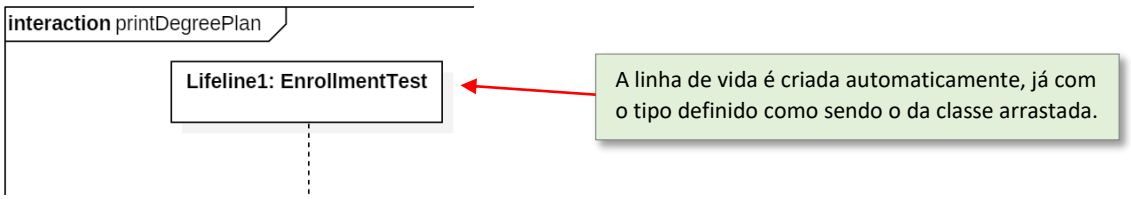
Existem diferentes tipos de mensagem. Neste momento, focaremos em mensagens síncronas (**Message**). Uma mensagem síncrona é aquela que bloqueia a execução do objeto chamador até que o objeto chamado finalize a operação invocada e retorne o controle de execução para o objeto chamador (**Reply Message**). Por este motivo, mensagens síncronas são também conhecidas como mensagens bloqueantes e representam uma chamada tradicional de procedimento.

Utilizaremos também o conceito de mensagem encontrada (**Found Message**) e mensagem perdida (**Lost Message**). Uma mensagem encontrada é aquela que chega de um emissor desconhecido ou de um emissor não mostrado no diagrama. Uma mensagem perdida é aquela que ou não chegou no destinatário ou que o destinatário não está mostrado no diagrama.

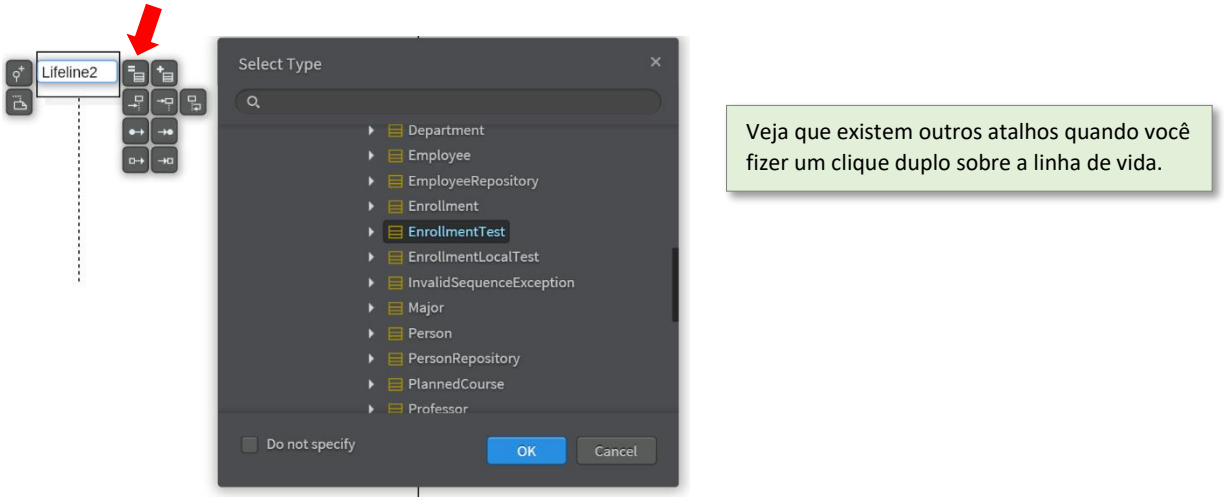
Ouro elemento utilizado será o fragmento (**Combined Fragment**) para representar áreas que estão dentro de um escopo específico (como um laço ou uma condição).

Roteiro 09 – Atividade 01/03

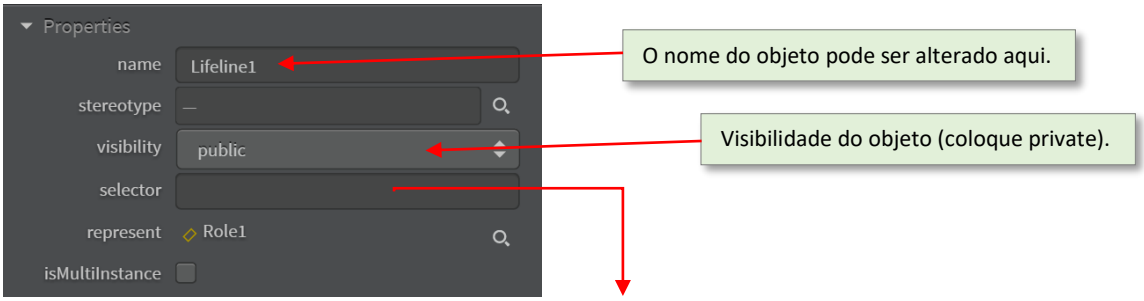
4. Para colocarmos a primeira linha de vida (*lifeline*), arraste a classe correspondente do *Model Explorer* para o diagrama.



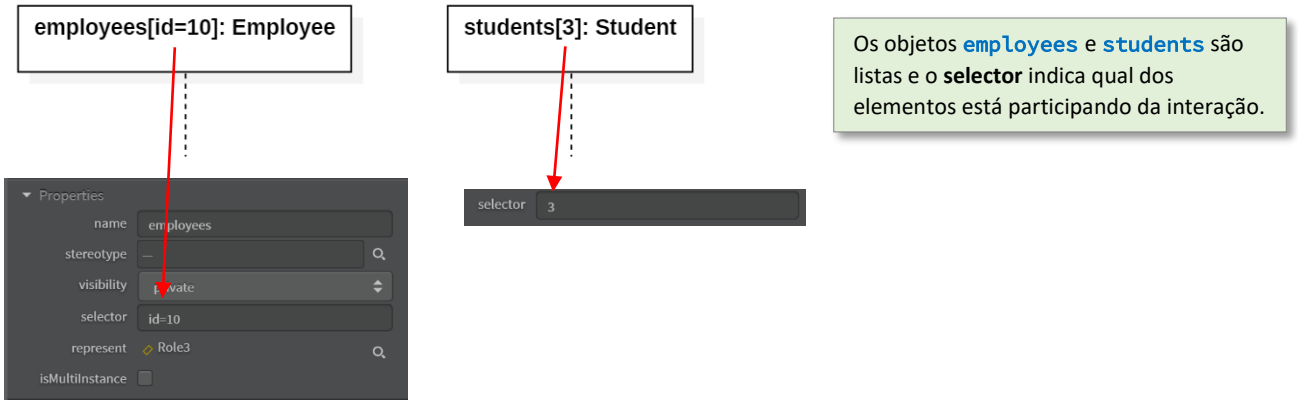
Outra opção seria clicar o elemento **Lifeline** da Toolbox e selecionar o tipo manualmente no *Model Explorer*.



5. Como o nome do nosso objeto `EnrollmentTest` não é relevante, iremos apagá-lo, ou seja, torná-lo **anônimo**. Clique na linha de vida e altere as propriedades.



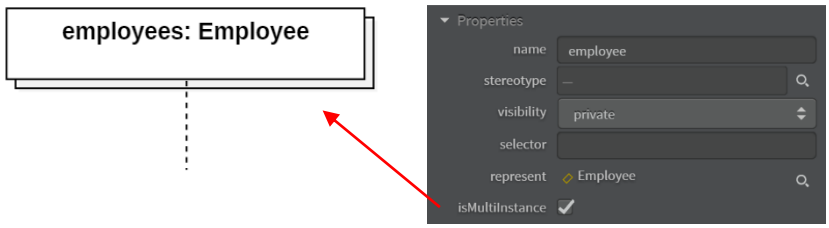
Se o objeto é multivalorado (multiplicidade > 1), então a linha de vida pode ter uma expressão (**selector**) que especifica qual ocorrência específico é representado por esta linha de vida. Se o **selector** é omitido, isto significa que um representante arbitrário dentre as ocorrências é escolhido.



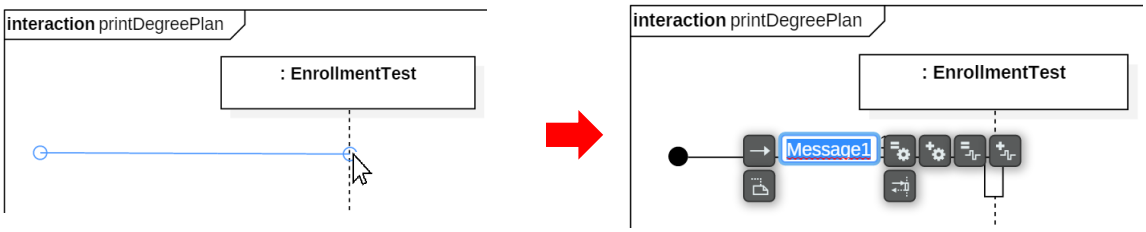
Roteiro 09 – Atividade 01/03

A propriedade **represent** indica qual o papel (role) desta linha de vida na colaboração. Como vimos antes, o diagrama de sequência é representado no contexto de uma **colaboração** (representada graficamente por uma elipse com a linha tracejada). Uma colaboração explica como um conjunto de instâncias cooperando entre si realiza uma ou mais tarefas em comum. Tipicamente, o nome do papel (role) é o mesmo nome do objeto representado na linha de vida. No momento, não focaremos nesta propriedade e você pode deixar os valores default.

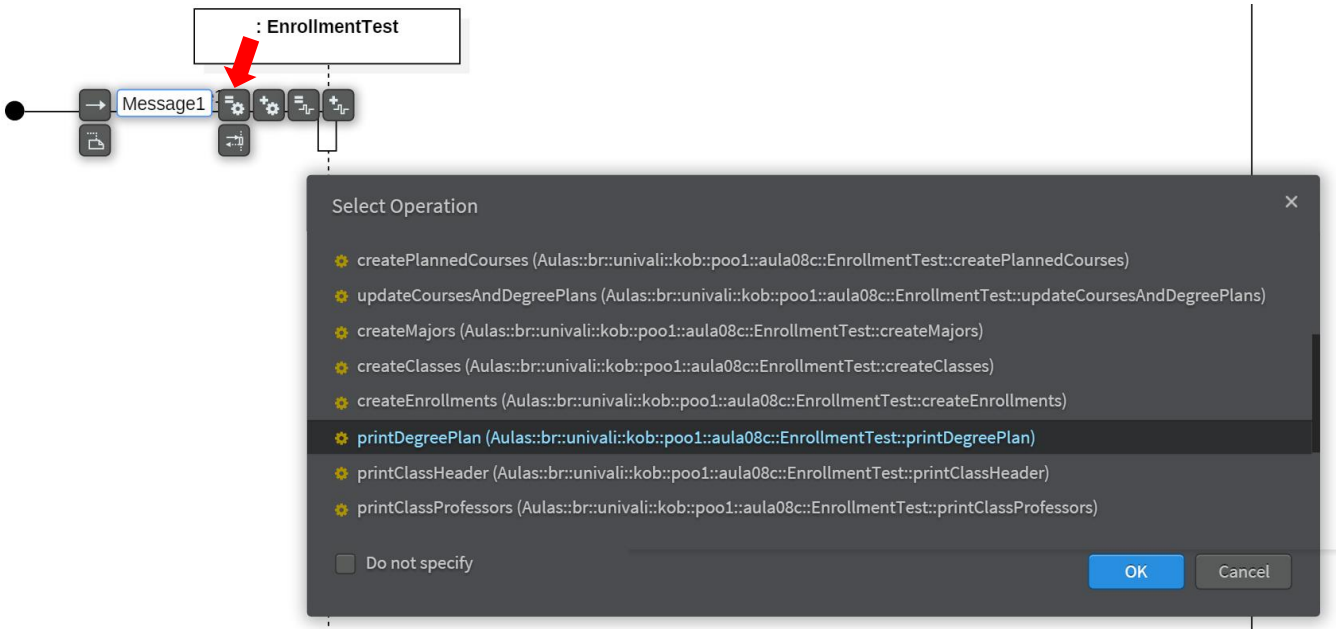
A propriedade **isMultiInstance** (**MultiObject**) indica se a linha de vida representa, na verdade, um conjunto de instâncias do objeto representado pela linha de vida (ex: uma coleção de objetos). Diferentemente do **selector**, não estamos indicando uma ocorrência em particular, mas estamos considerando toda a coleção.



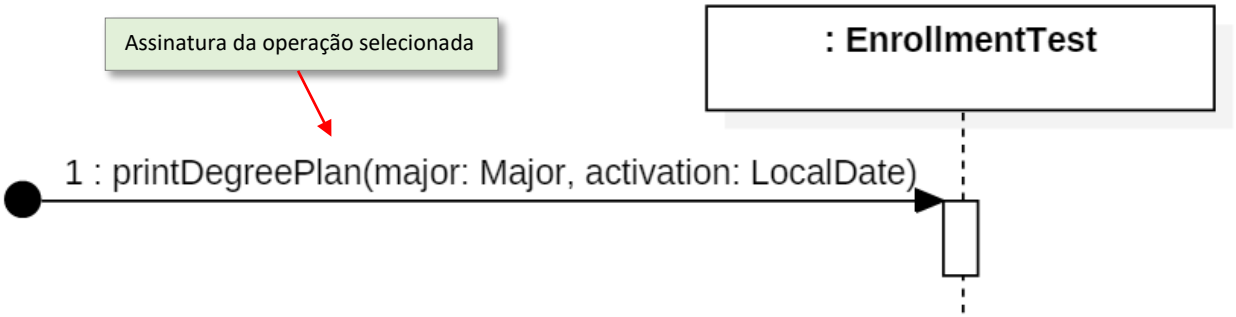
6. Voltando ao nosso diagrama, na Toolbox **Interactions (Advanced)**, selecione o elemento **FoundMessage**. Clique no diagrama e arraste até a linha de vida.



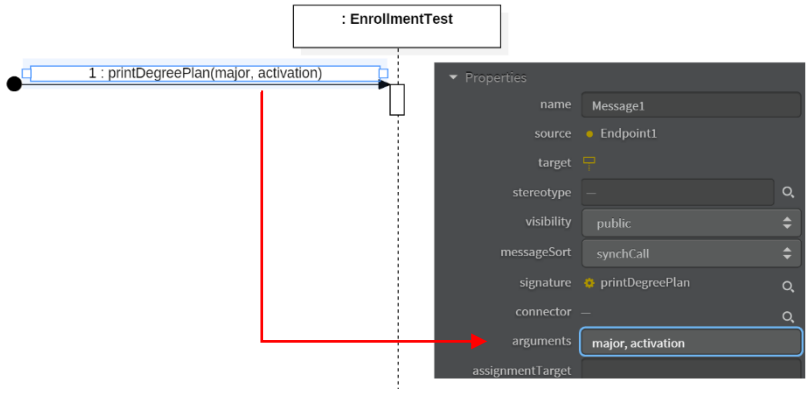
Utilize o atalho **Select Operation**, o qual abrirá uma janela de seleção para que você escolha qual operação de `EnrollmentTest` deve ser invocada. Como sua linha de vida é um objeto `EnrollmentTest`, a ferramenta já sabe que somente as operações desta classe podem ser invocadas.



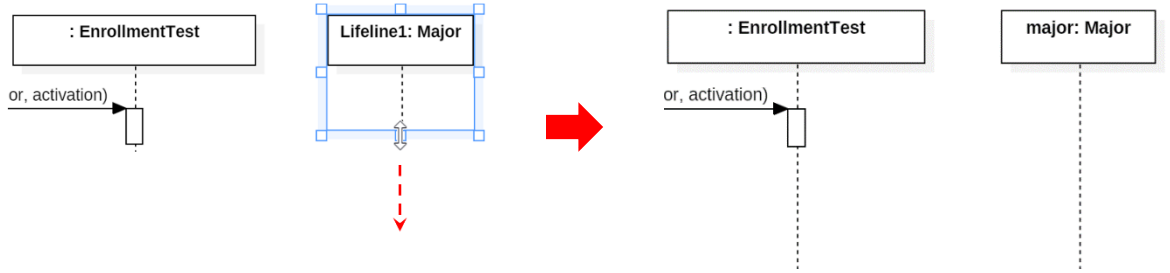
Roteiro 09 – Atividade 01/03



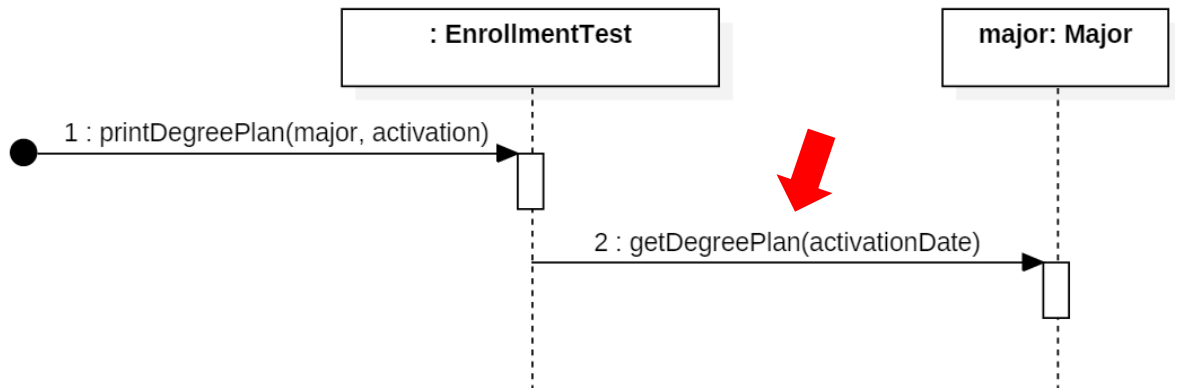
7. Para representar melhor nosso algoritmo, vamos utilizar o nome dos argumentos passados ao invés dos nomes dos parâmetros.



8. Agora arraste a próxima classe para o diagrama, como visto no passo 4. De acordo com nosso exemplo, será a classe `Major`. Altere o nome da linha de vida para major (como utilizado no método). Para ampliar a linha de vida (no tempo), basta selecionar a linha de vida, colocar o mouse sobre a parte inferior da área selecionada e arrastar.

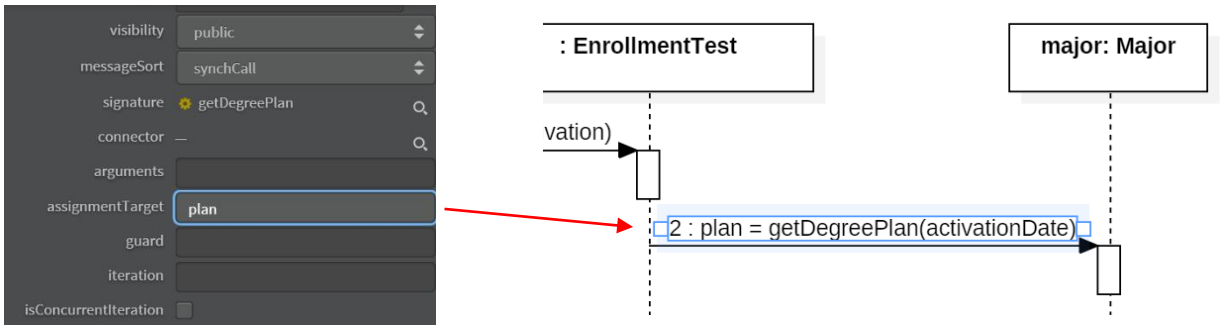


9. Agora, na Toolbox **Interactions (Basic)**, selecione o elemento **Message**. Clique na linha de vida chamadora (neste caso, `:EnrollmentTest`) e arraste até a linha de vida chamada (`major: Major`). O procedimento é o mesmo explicado no passo 6. Note que as operações apresentadas na janela são aquelas definidas na classe `Major` (quem recebe é quem executa a operação). Selecione a operação conforme nosso exemplo.

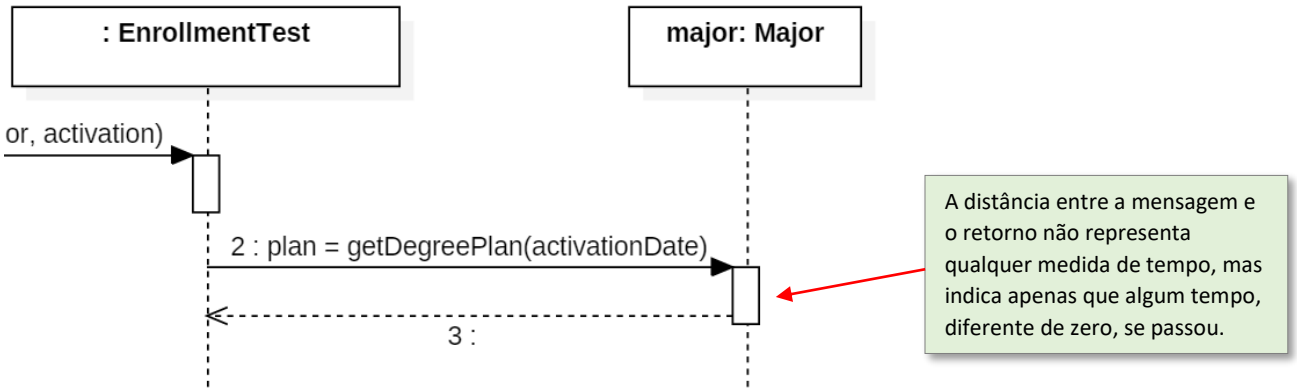


Roteiro 09 – Atividade 01/03

10. No caso desta última mensagem, queremos indicar que o retorno deve ser atribuído a uma variável local chamada `plan`. Para isso, basta preenchermos a propriedade **assignmentTarget** (destino da atribuição).

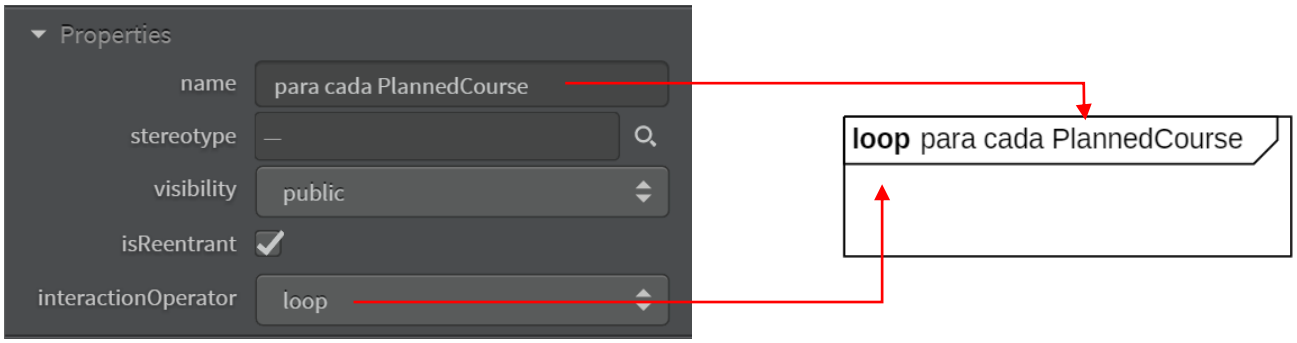


11. O próximo passo é desenhar o retorno do controle de execução para `:EnrollmentTest`. Para isso, utilize o elemento **Reply Message** na Toolbox **Interactions (Basic)**. O retorno será da linha de vida `major: Major` para a linha de vida `:EnrollmentTest`. Por ser um retorno de controle, não há operação associada. Note que a barra de ativação termina no ponto é que o retorno é enviado.



12. Continue o desenho do diagrama, conforme o exemplo apresentado inicialmente. Coloque todas as mensagens sem se preocupar agora com os fragmentos **loop** (laços).

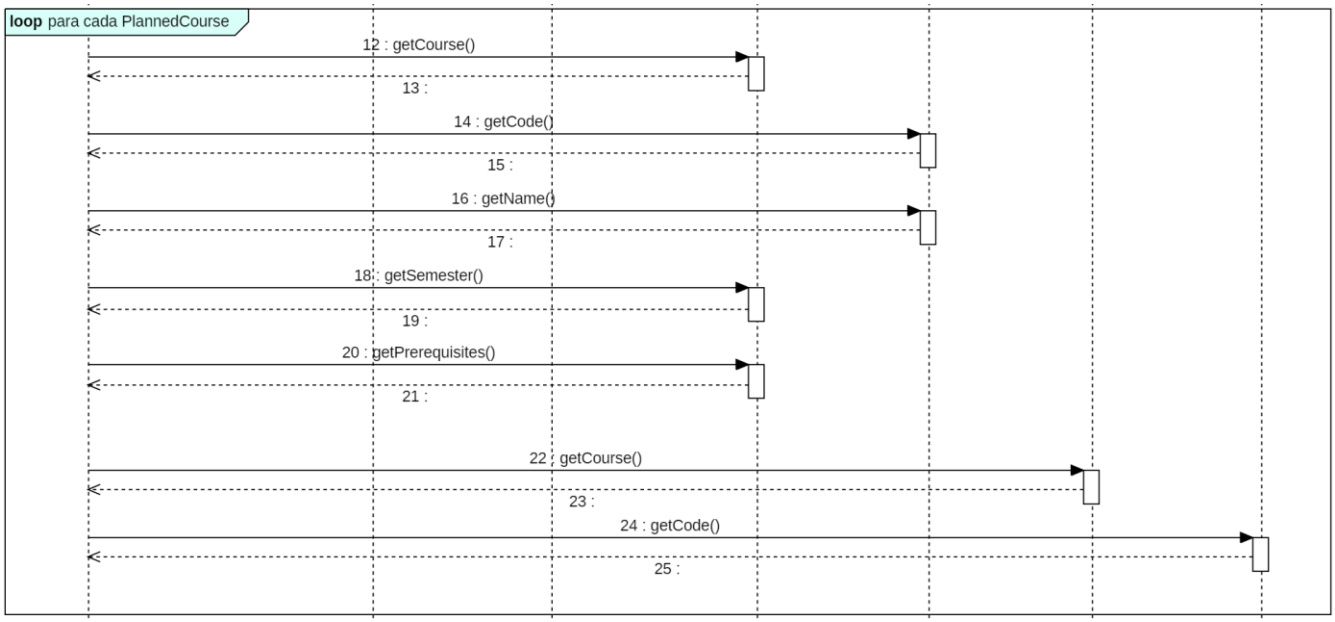
13. Podemos agora adicionar os fragmentos para representar os laços do nosso algoritmo. Note que temos um laço "**for**" dentro do outro. Selecione o elemento **Combined Fragment** na Toolbox **Interactions (Advanced)**.



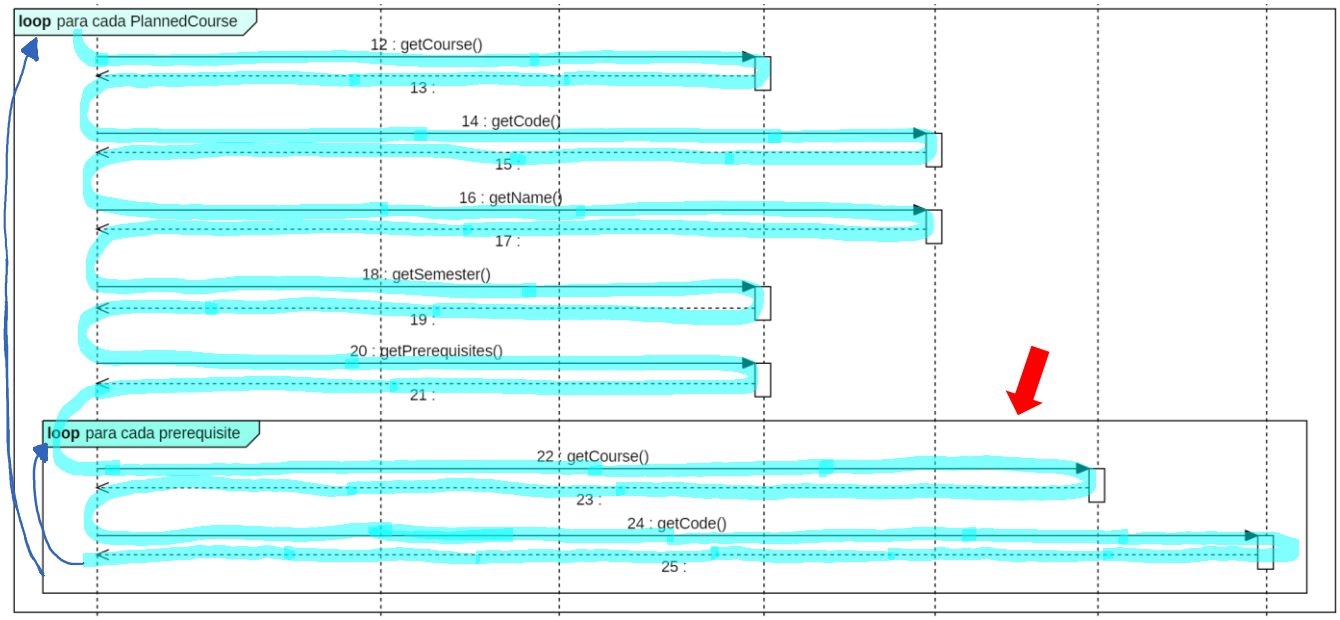
Existem vários tipos de fragmento, os quais permitem representar diferentes situações em nossos diagramas (laços, condicionais, paralelismo, etc.). Na propriedade **name**, coloque um texto que facilite o entendimento do algoritmo. Exemplos: "para cada elemento da lista X", "enquanto x for verdadeiro", "para os primeiros 10 elementos de X".

Roteiro 09 – Atividade 01/03

Agora, basta posicionar o fragmento de modo que as mensagens afetadas pelo laço estejam dentro da área do fragmento.

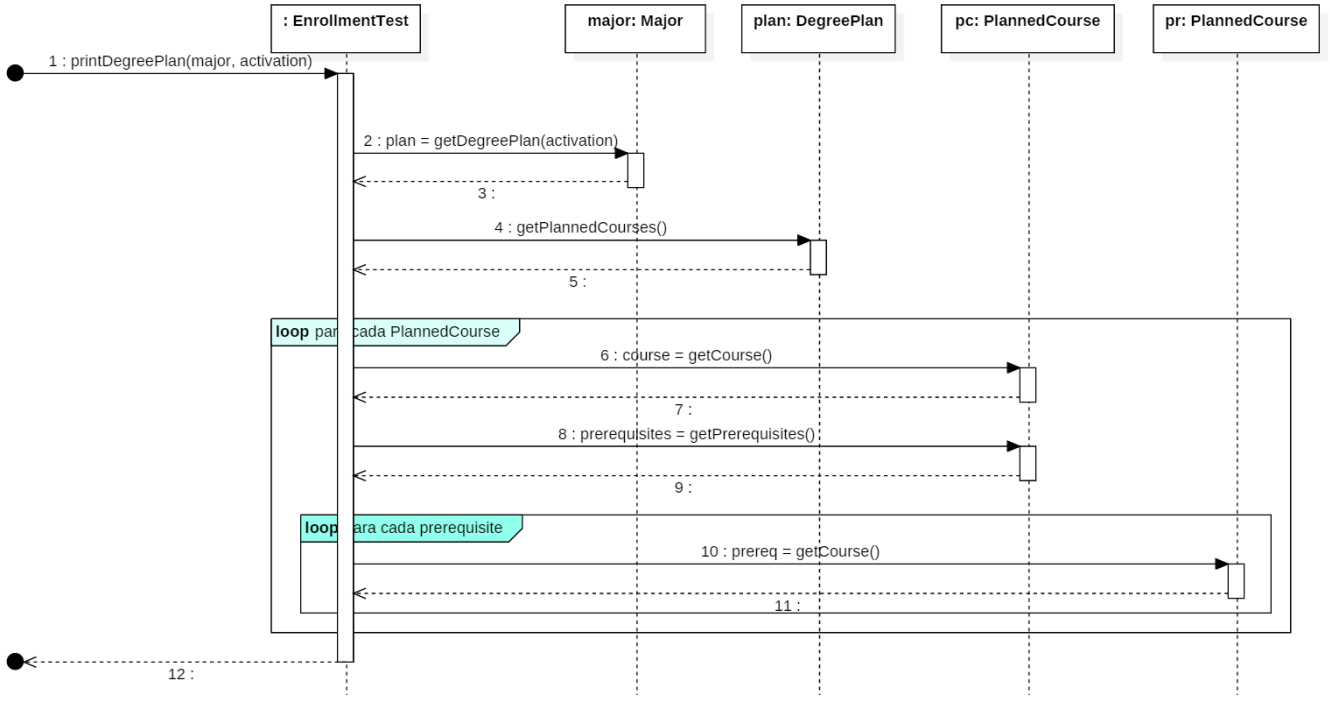


14. Siga o mesmo procedimento anterior para inserir o segundo fragmento. Para representar corretamente a ideia de que o segundo laço está aninhado no primeiro, você deve garantir que o segundo fragmento está dentro do primeiro.



Embora este exemplo tenha sido útil para mostrar a construção passo a passo de um diagrama de sequência, raramente modelaremos um diagrama com todos estes detalhes (repare que o diagrama é praticamente a programação do método). Diagramas de sequência são adequados para mostrar quais objetos se comunicam com quais outros objetos e quais disparam estas comunicações. Entretanto, devemos evitar que eles sejam utilizados para mostrar detalhes da complexidade de uma lógica procedural. Embora, na versão atual, já tenhamos desconsiderado detalhes de impressão, o diagrama poderia ter sido desenhado de modo ainda mais simples, como mostra a próxima figura.

Roteiro 09 – Atividade 01/03



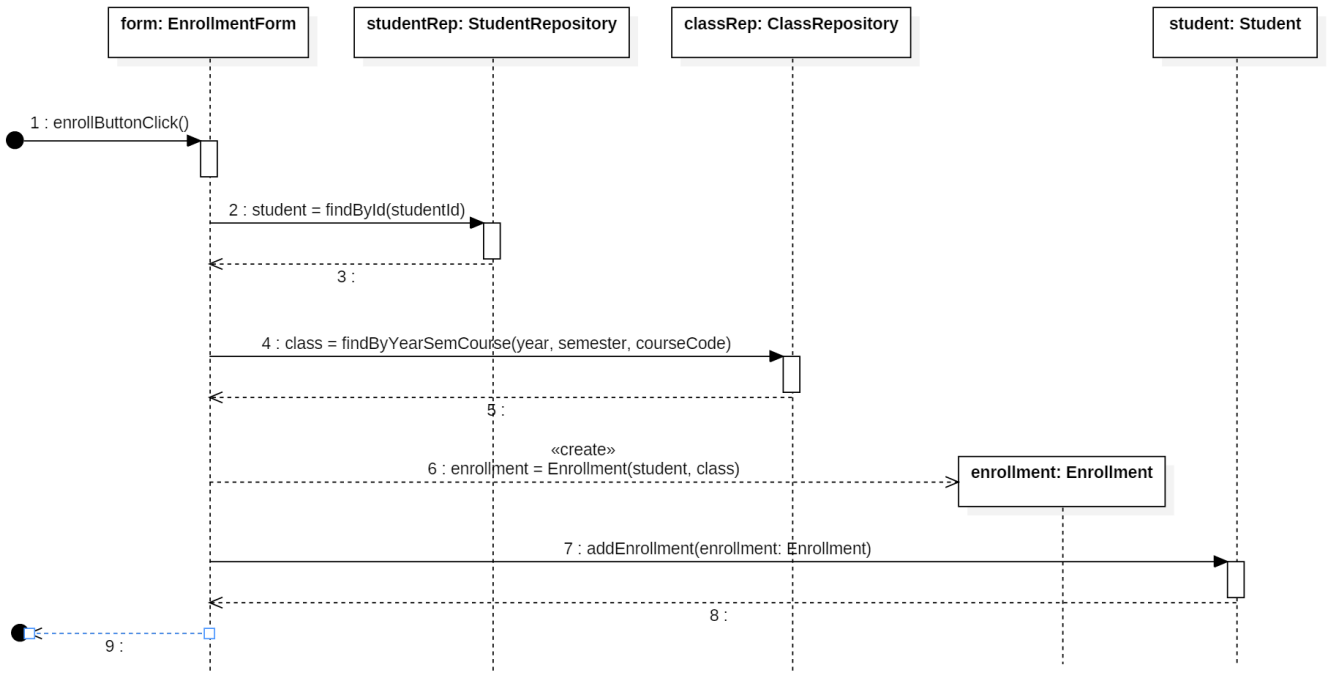
Veja que este diagrama já oferece ao programador, informações suficientes sobre como ele deve proceder. Detalhes sobre quais informações devem ser impressas podem estar definidas em um protótipo de tela ou na especificação dos requisitos.

Outra diferença mostrada neste diagrama é a barra de ativação (que indica um objeto em execução). Note que a barra entre mensagem inicial e a última (retorno) está correta, pois o objeto `:EnrollmentTest` está em execução durante todo o diagrama. A ferramenta não estica automaticamente a barra de ativação quando você move a mensagem de retorno de controle. Temos que fazer isso manualmente.

Durante esta disciplina, utilizaremos diferentes níveis de abstração para desenhar os diagramas de sequência, uma vez que nosso objeto é didático. O professor irá orientá-lo sobre qual nível de detalhe deverá ser empregado no desenho do diagrama.

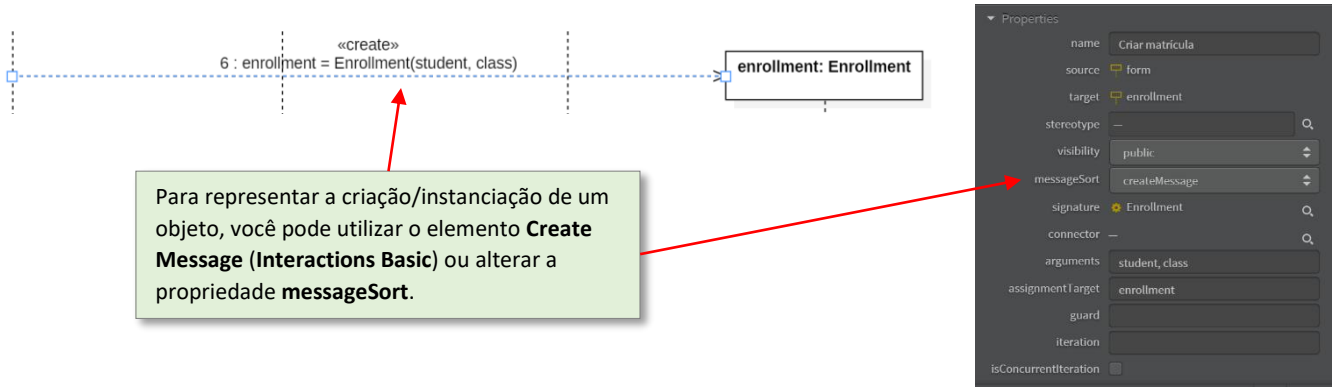
Roteiro 09 – Atividade 02/03

1. Não devemos desenhar duas vezes o mesmo algoritmo em diagramas de sequência diferentes. Por exemplo, se você já desenhou o diagrama de um determinado método relevante, você pode apenas enviar uma mensagem para sua respectiva linha de vida, sem precisar conhecer os detalhes de como o método foi implementado (exatamente como fazemos quando chamamos um objeto).



No diagrama acima, foi representado uma situação onde um formulário de matrícula efetua a matrícula de um estudante a partir dos dados fornecidos por um usuário: a) id do estudante; e b) ano, semestre e código da disciplina. Note que nem todas as mensagens foram desdobradas em como elas foram implementadas (seus métodos). Por exemplo, não precisamos saber neste diagrama como o repositório de estudantes ou de classe encontra o objeto que queremos. Precisamos apenas saber qual operação deve ser invocada (mensagem).

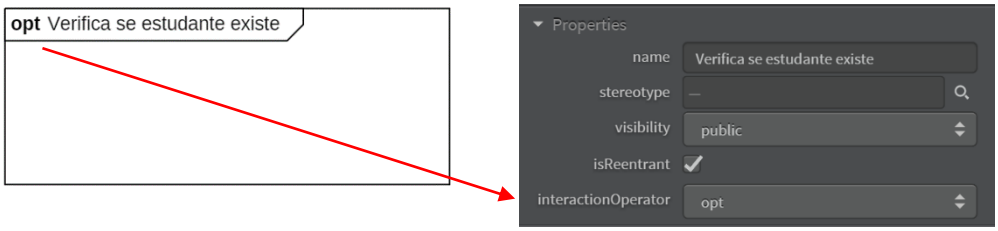
Outro ponto interessante é a demonstração de como instanciar um objeto em um diagrama UML. Note que a linha de vida `enrollment: Enrollment` só passou a existir após sua instanciação, o que faz sentido (como enviar uma mensagem para um objeto que ainda não existe).



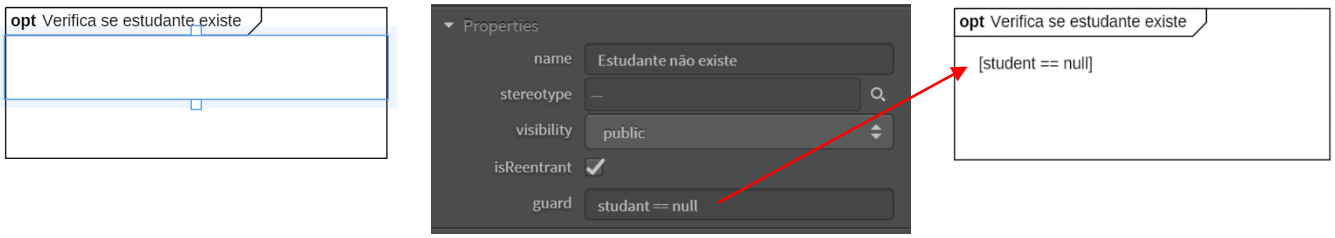
Crie o diagrama acima para exercitar o que foi visto até o momento. Você pode depois, implementar o formulário de matrícula utilizando uma interface modo texto. Todas as demais classes já devem ter sido implementadas nos roteiros anteriores.

Roteiro 09 – Atividade 02/03

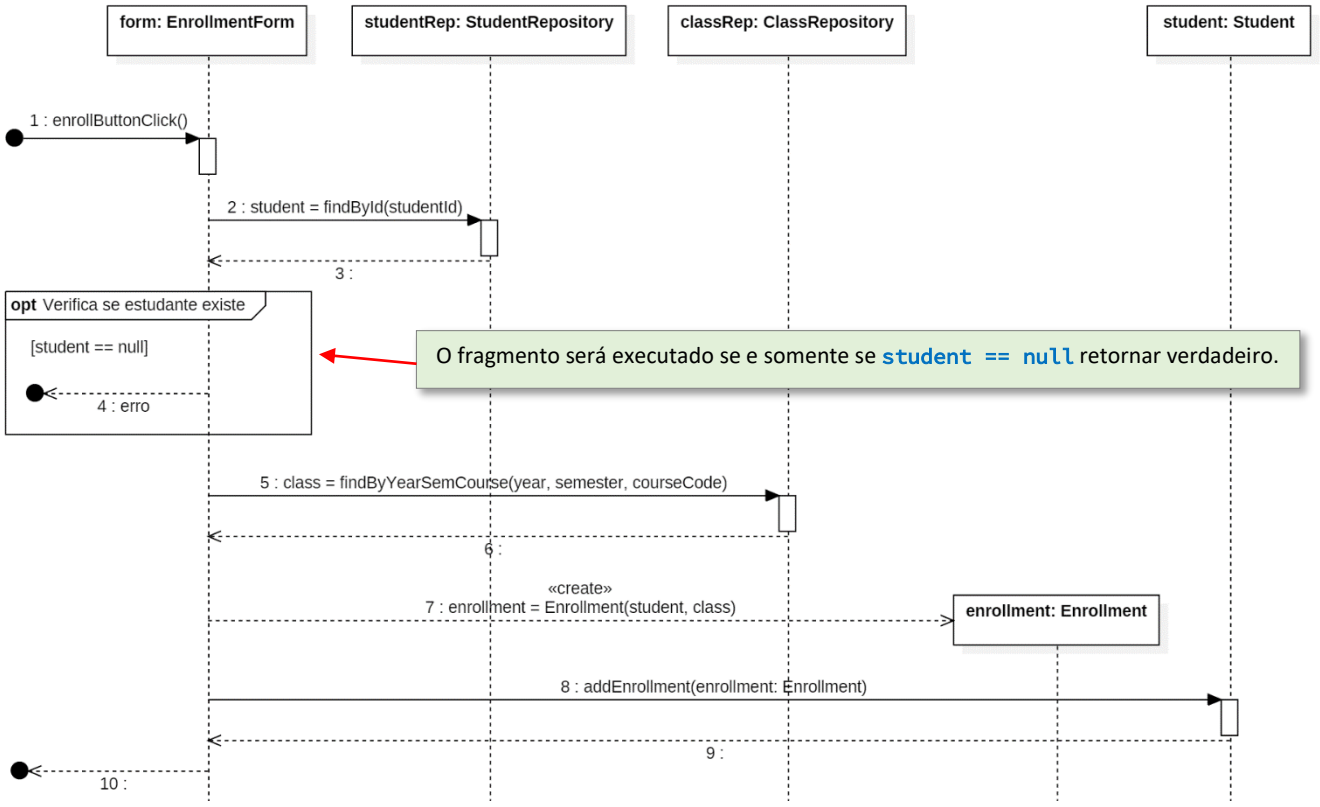
2. Você deve ter notado que o diagrama anterior poderia ter sido abortado em sua execução, caso não fosse encontrado um estudante com o código informado ou não existisse uma classe para o ano, semestre e código de disciplina fornecidos. Caso você quisesse representar estas condições, precisaríamos adicionar um fragmento condicional. O elemento é o mesmo utilizado para laços (**Combined Fragment**), mas você precisa alterar a propriedade **interactionOperator** para **opt** (*option*).



Note que o nome foi alterado para oferecer maior legibilidade ao objetivo deste fragmento na interação. O operador **opt** significa que o fragmento combinado representa uma escolha de comportamento onde todo o bloco do operando é executado ou nada acontece (IF-THEN). Clique na área interna do fragmento e preencha a propriedade **guard**. Um **guarda** em UML é uma expressão booleana.



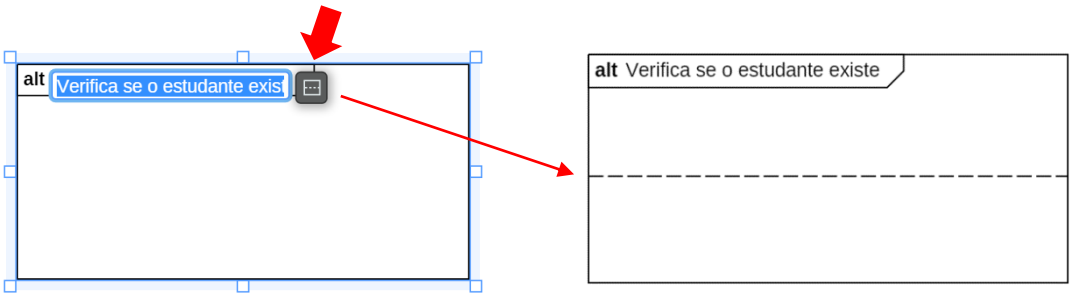
Agora, podemos completar nosso diagrama. Note que o objetivo foi apenas mostrar que se o estudante não for encontrado, o método deveria retornar com erro (por exemplo, uma exceção).



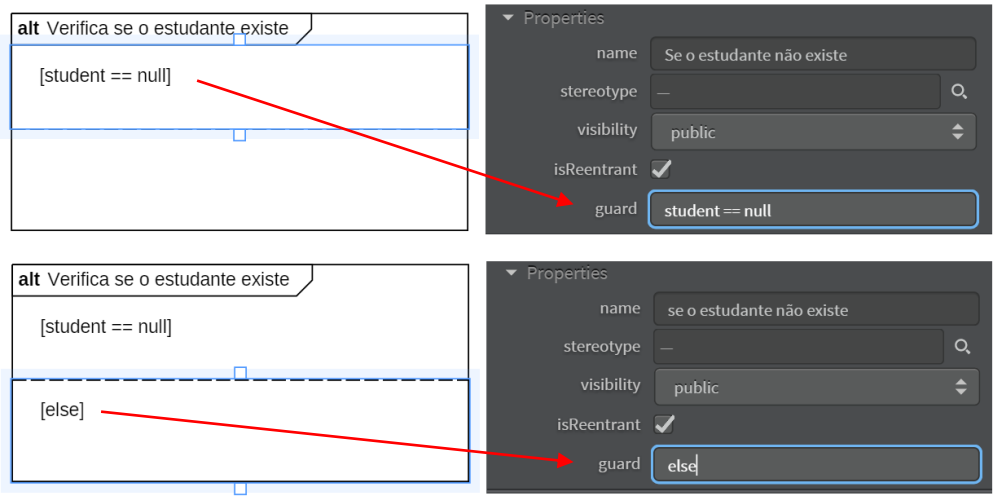
Você pode fazer algo similar para desenhar o retorno de um erro, caso a turma não seja encontrada.

Roteiro 09 – Atividade 02/03

3. Mas e se precisarmos representar mais condições. Por exemplo, um IF-THEN-ELSE ou um SWITCH? Para desenharmos estas situações, devemos utilizar o operador **alt** (*alternatives*). A alteração do operador pode ser feita via propriedades. Entretanto, para criarmos uma área para cada alternativa, precisamos fazer um clique duplo sobre o fragmento e utilizar o atalho indicado abaixo.



Cara área representa um operando (alternativa). Você pode adicionar quantos operandos forem necessários. Você pode ainda manualmente alterar as dimensões da área coberta pelo operando. Cada área tem seu próprio operando.



Agora basta organizar as mensagens de modo que elas fiquem dentro do seu operando, conforme o algoritmo.