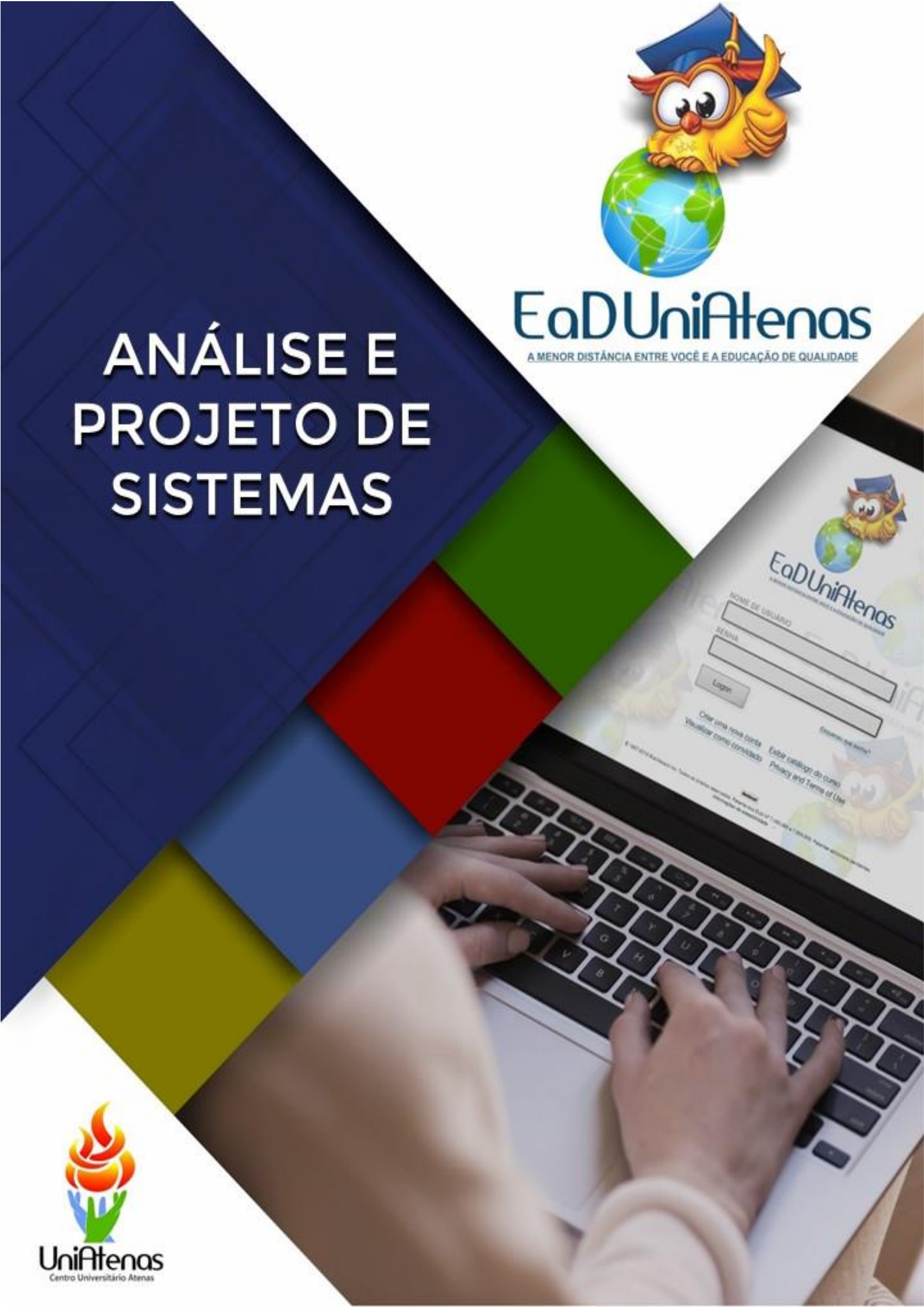
EaDUniAtenas

A MENOR DISTÂNCIA ENTRE VOCÊ E A EDUCAÇÃO DE QUALIDADE

ANÁLISE E PROJETO DE SISTEMAS



UniAtenas
Centro Universitário Atenas

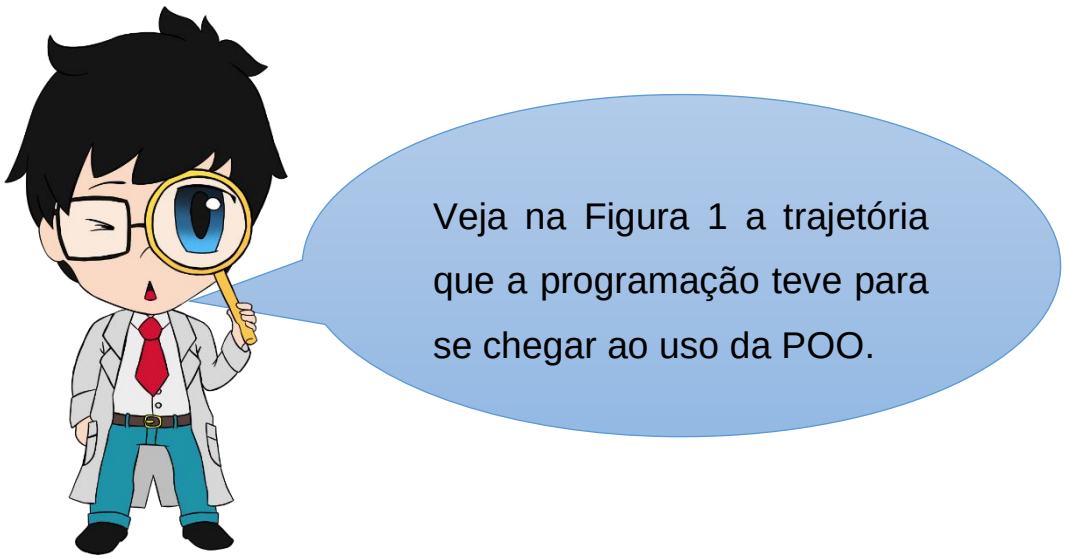
CONCEITO DE CLASSE E OBJETO NOTAÇÃO UML PARA CLASSE E OBJETO

OBJETIVO: Entender os princípios e os diagramas da UML

Introdução

Nos desenvolvimentos de sistemas, existem alguns fatores importantes como: o entendimento do código, fácil manutenção, reaproveitamento entre outros. Para isso, a Programação Orientada a Objetos também conhecida como POO, tem a intenção de ajudar nesses fatores, dando tempo e agilidade no desenvolvimento de um sistema para o programador.

A Programação Orientada a Objetos foi criada por Alan Kay, autor da linguagem Smalltalk. Antes mesmo da criação Orientada a Objetos, já existiam algumas aplicações, neste caso da linguagem Simula 67, criada por Ole Johan Dahl e Kristen Nygaard em 1967.



1950 – 1960 Era do Caos	1970 – 1980 Era da Estruturação	1990 até agora Era dos Objetos
Saltos, gotos, variáveis não estruturadas, variáveis espalhadas ao longo do programa	If-then-else Blocos Registros Laços-While	Objetos Mensagens Métodos Herança

Classes

Uma classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos - através de métodos - e os estados possíveis destes objetos - através de atributos.

Em outras palavras, uma classe se descreve nos serviços oferecidos por seus objetos onde os quais informam onde eles podem ser armazenados.

Classes não são diretamente suportadas em todas as linguagens, e são necessárias para que uma linguagem seja orientada a objetos.

Classes não são diretamente suportadas em todas as linguagens, e são necessárias para que uma linguagem seja orientada a objetos.

Uma classe representa um conjunto de objetos com características afins. Uma classe define o comportamento dos objetos através de seus métodos, e quais estados ele é capaz de manter através de seus atributos.

Características das classes

- Toda classe possui um nome;
- Possuem visibilidade, exemplo: public, private, protected;
- Possuem membros como: Características e Ações;
- Para criar uma classe basta declarar a visibilidade + digitar a palavra reservada class + NomeDaClasse + abrir e fechar chaves { }.

Listagem 1: Declaração de uma classe na linguagem Java

```
public class Teste{
//ATRIBUTOS OU PROPRIEDADES
//MÉTODOS
}
```

Na Listagem 2, são mostrados os componentes da classe, como métodos e atributos.

Listagem 2: Classe Caes

```
public class Caes {
```

```
        public String nome;
        public int peso;
        public String corOlhos;
        public int quantPatas;

        public void falar(){
                //MÉTODO FALAR
        }

        public void andar(){
                //MÉTODO ANDAR
        }

        public void comer(){
                //MÉTODO COMER
        }

        public void dormir(){
                //MÉTODO DORMIR
        }
}
```

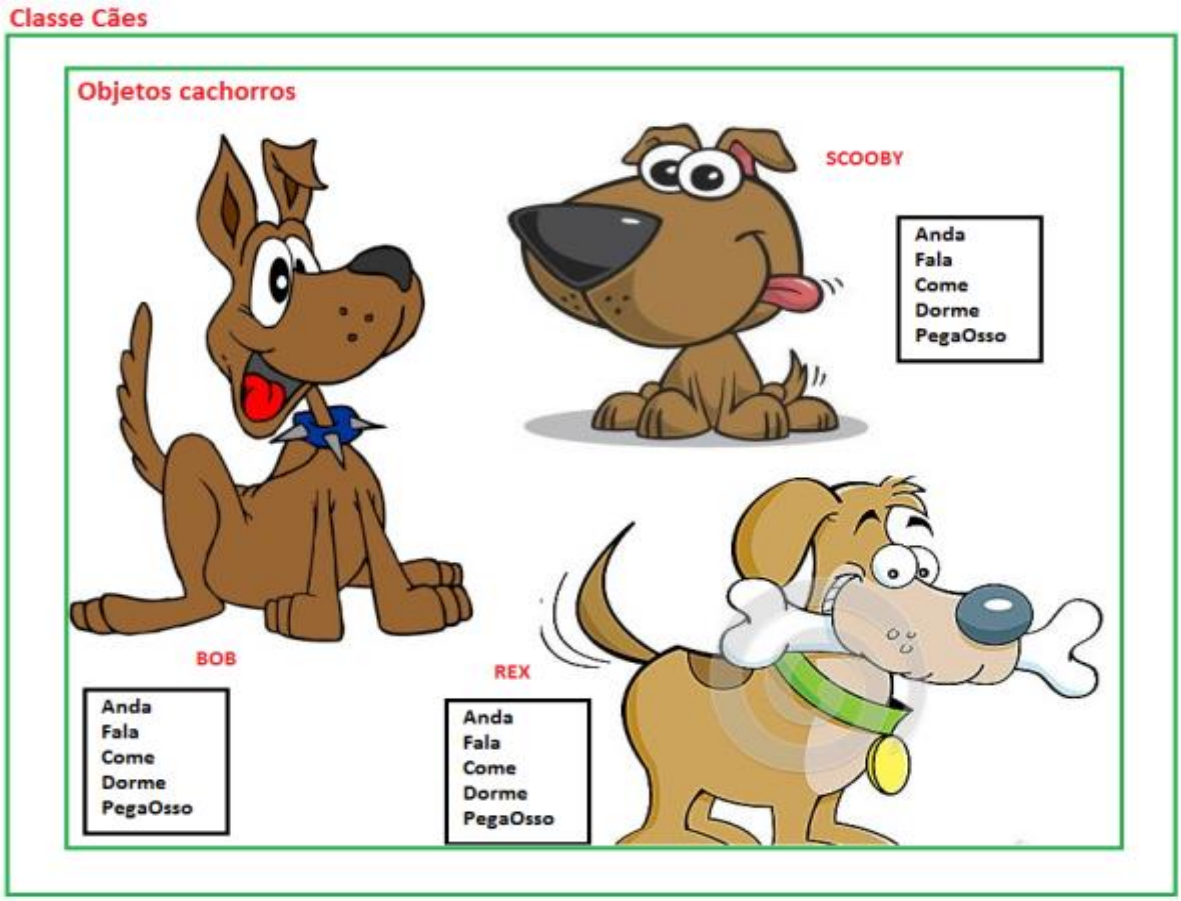
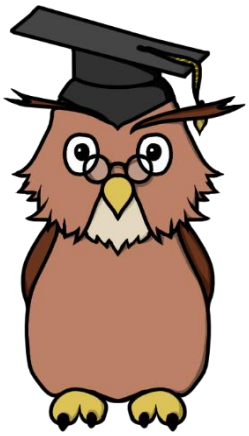


Figura 2: Demonstração da classe Cães



Na Listagem 2 e na Figura 2, mostra que a classe Cães de um modo genérico, tem os mesmos métodos independente de qualquer cachorro, sendo que a classe é sempre um molde/projeto para o objeto cachorro.

Objeto

Os objetos são características definidas pelas classes. Neles é permitido instanciar objetos da classe para inicializar os atributos e invocar os métodos. Veja no exemplo da Figura 3.

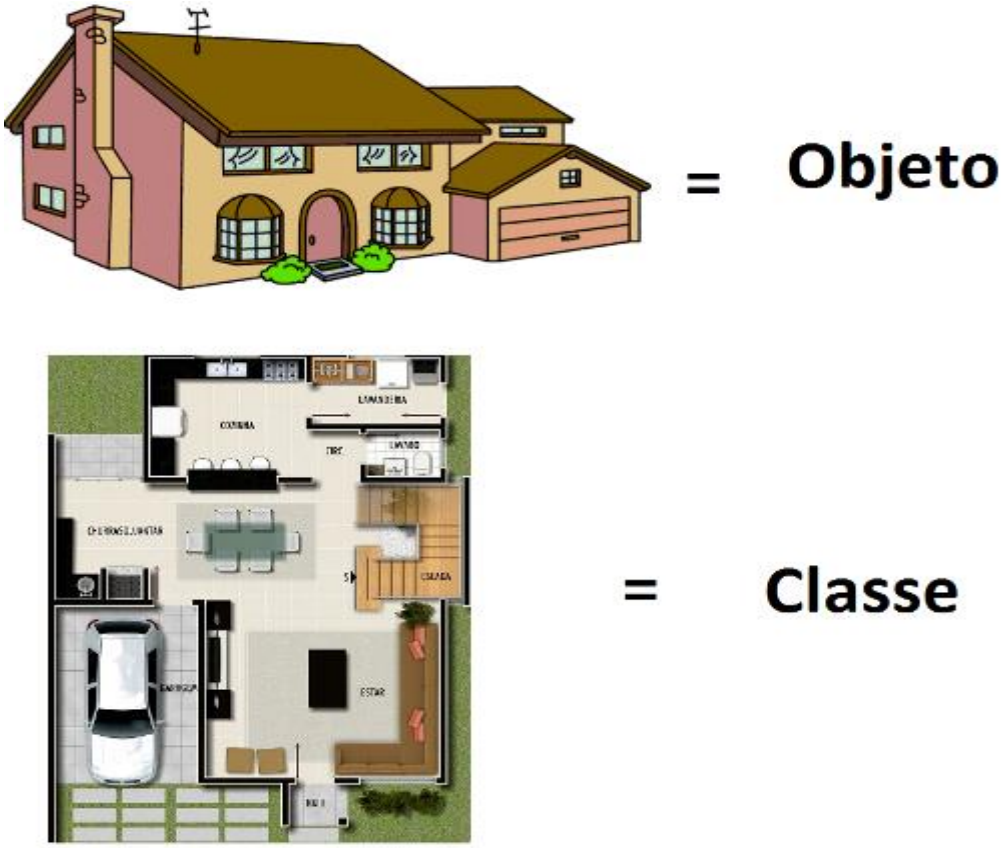


Figura 3: Diferença entre objeto e classe

Figura 3 mostra que todo objeto é algo que existe, uma coisa concreta, já a classe é considerada como um modelo ou projeto de um objeto, sendo algo que não consegue tocar.

Atributos

Os atributos são as propriedades de um objeto, também são conhecidos como variáveis ou campos. Essas propriedades definem o estado de um objeto, fazendo com que esses valores possam sofrer alterações. A Listagem 3 mostra as características de um cachorro, mas os valores que são guardados nas variáveis são diferentes variando para cada cachorro.

Listagem 3: Classe Cachorro

```
public class Cachorro{

        public String nome;
        public int peso;
        public String corOlhos;
        public int quantPatas;
}
```

Na Listagem 4 é instanciada três vezes a classe “Cachorro”, mostrando que cada cachorro instanciado tem características diferentes.

Listagem 4: Classe Testadora de Cachorro

```
public class TestaCaes {

    public static void main(String[] args) {
        Cachorro cachorro1 = new Cachorro();
        cachorro1.nome = "Pluto";
        cachorro1.corOlhos = "azuis";
        cachorro1.peso = 53;
        cachorro1.quantPatas = 4;

        Cachorro cachorro2 = new Cachorro();
        cachorro2.nome = "Rex";
        cachorro2.corOlhos = "amarelo";
        cachorro2.peso = 22;
        cachorro2.quantPatas = 3;

        Cachorro cachorro3 = new Cachorro();
        cachorro3.nome = "Bob";
        cachorro3.corOlhos = "marrom";
        cachorro3.peso = 13;
        cachorro3.quantPatas = 4;

    }

      }
```

Métodos

Os métodos são ações ou procedimentos, onde podem interagir e se comunicarem com outros objetos. A execução dessas ações se dá através de mensagens, tendo como função o envio de uma solicitação ao objeto para que seja efetuada a rotina desejada.

Como boas práticas, é indicado sempre usar o nome dos métodos declarados como verbos, para que quando for efetuada alguma manutenção seja de fácil entendimento.

Na Listagem 5 são declaradas as características e o método, nota-se que tem uma condição de acordo com o valor informado na variável “tamanho”.

Listagem 5: Classe Cachorro com método

```
class Cachorro{
        int tamanho;
        String nome;


        void latir(){
                if(tamanho > 60)
                        System.out.println("Wooof, Wooof!");
                else if(tamanho > 14)
                        System.out.println("Ruff!, Ruff!");
                else
                        System.out.println("Yip!, Yip!");
        }
}
```

A Listagem 6 mostra como uma variável pode mudar o estado de um objeto, comunicando-se com o método invocado.

Listagem 6: Classe Testadora

```
public class Testa_Cachorro {

        public static void main(String[] args) {
```

```
                Cachorro bob = new Cachorro();
                bob.tamanho = 70;
                Cachorro rex = new Cachorro();
                rex.tamanho = 8;
                Cachorro scooby = new Cachorro();
                scooby.tamanho = 35;

                bob.latir();
                rex.latir();
                scooby.latir();

        }
}
```

Construtores

O construtor de um objeto é um método especial, pois inicializa seus atributos toda vez que é instanciado (inicializado).

Toda vez que é digitada a palavra reservada **new**, o objeto solicita para a memória do sistema armazená-lo, onde chama o construtor da classe para inicializar o objeto. A identificação de um construtor em uma classe é sempre o mesmo nome da classe.

Na Listagem 7, o construtor recebe um parâmetro de uma String que será um argumento de entrada na classe testadora.

Listagem 7: Declaração de um construtor com parâmetro

```
class ConstrutorProg{
        private String nomeCurso;


        public ConstrutorProg(String nome)
        {
                nomeCurso = nome;
        }

        public String getNome()
        {
                return "Nome do Curso retornado "+nomeCurso;
```

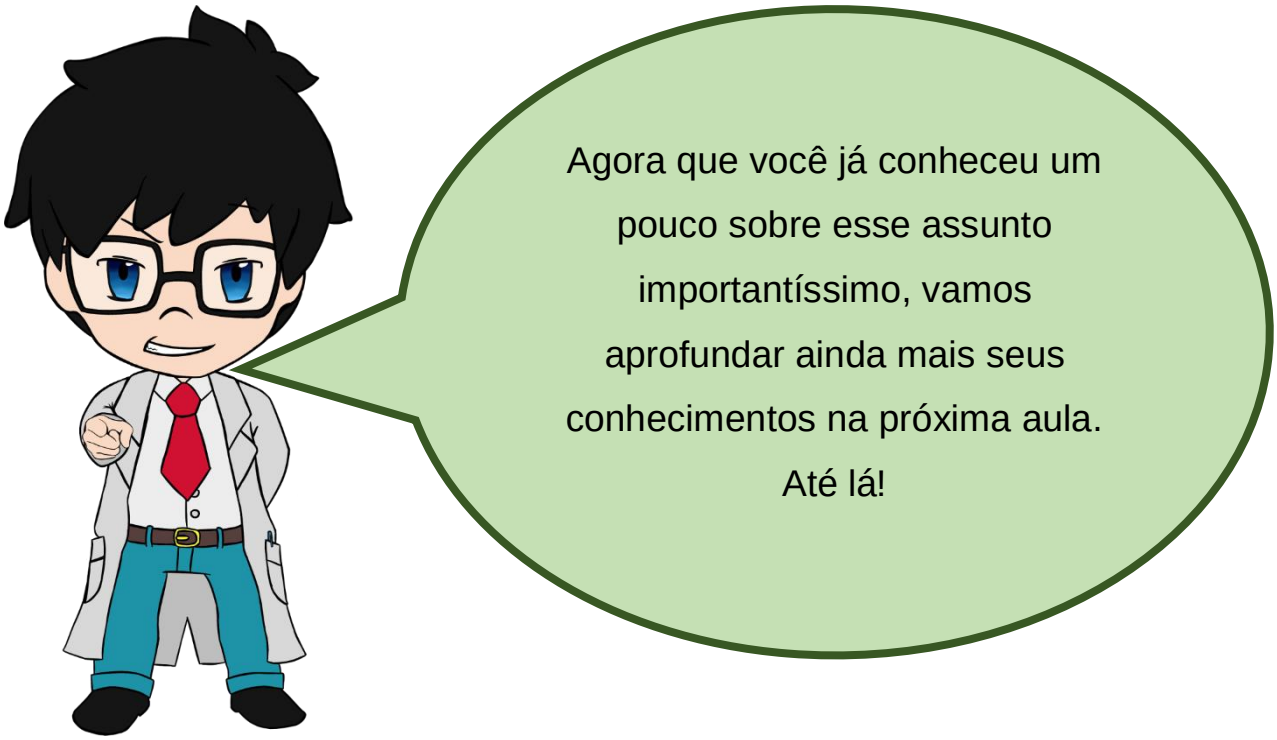
```
        }

}
```

Listagem 8: Classe Testadora do Construtor

```
public class Construtor {

        public static void main(String[] args) {

                ConstrutorProg cp = new ConstrutorProg("DevMedia - Java");
                System.out.println(cp.getNome());

        }
}
```

Na Listagem 8 é inicializada a classe “ConstrutorProg”, passando apenas um argumento no parâmetro que foi definido, se fosse apenas inicializado sem nenhum argumento estaria ocorrendo um erro de sintaxe pois já foi definido que no método construtor iria haver uma entrada de um parâmetro.

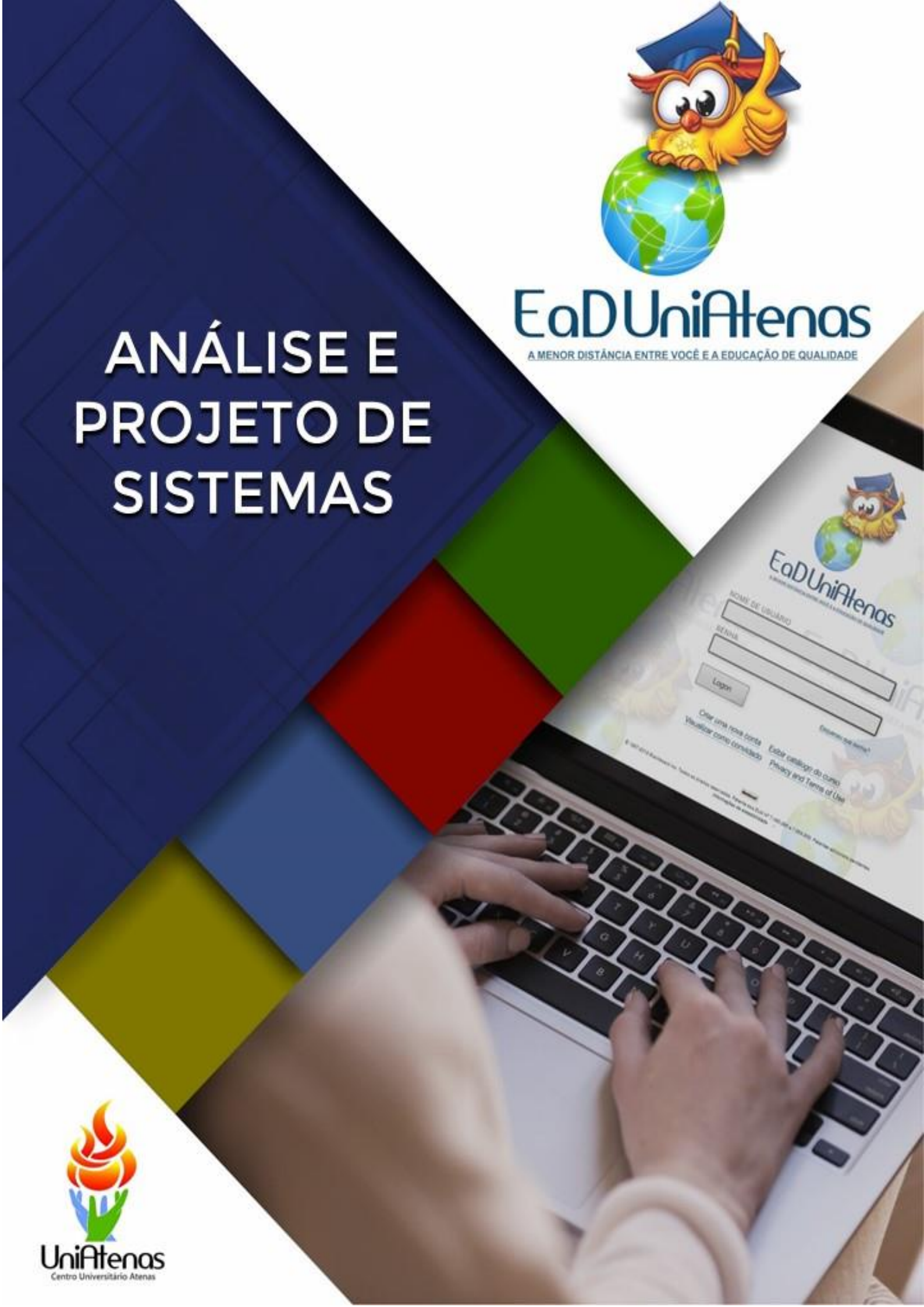
Sempre uma classe terá um construtor padrão, mesmo não sendo declarado o compilador irá fornecer um. Esse construtor não recebe argumentos e existe para possibilitar a criação de objetos para uma classe.


Agora que você já conheceu um pouco sobre esse assunto importantíssimo, vamos aprofundar ainda mais seus conhecimentos na próxima aula.
Até lá!

EaD UniAtenas

A MENOR DISTÂNCIA ENTRE VOCÊ E A EDUCAÇÃO DE QUALIDADE

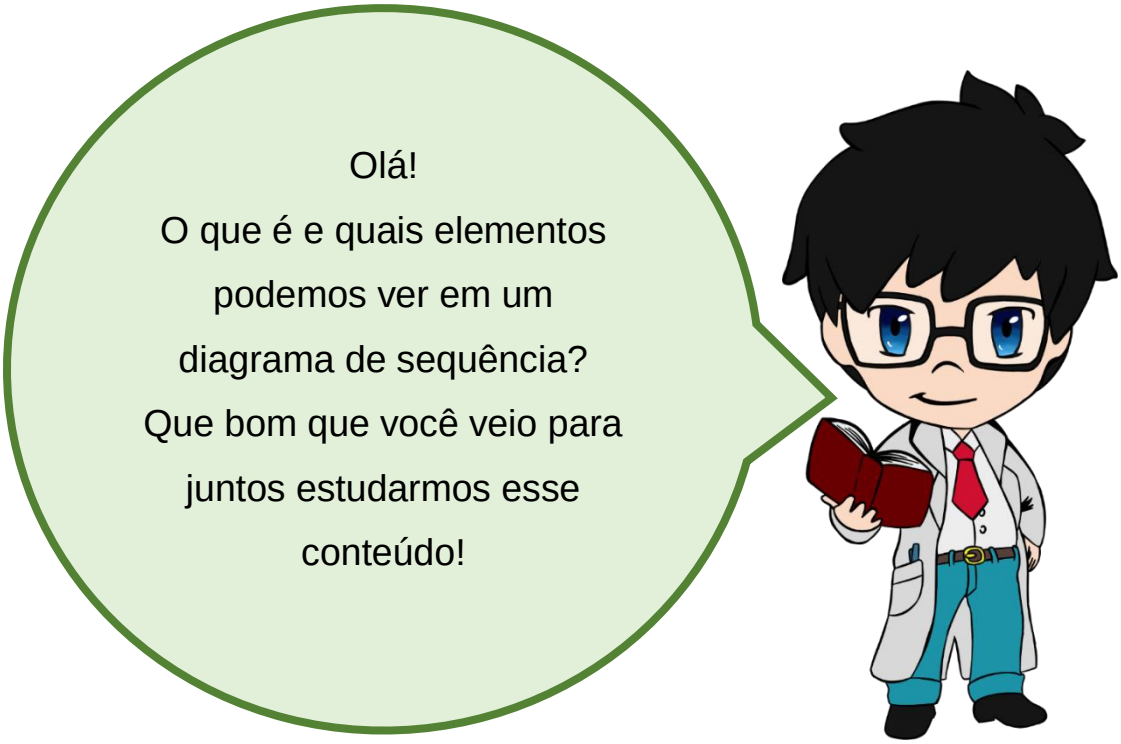
ANÁLISE E PROJETO DE SISTEMAS



UniAtenas
Centro Universitário Atenas

CONCEITO DE DIAGRAMAS DE SEQUÊNCIA

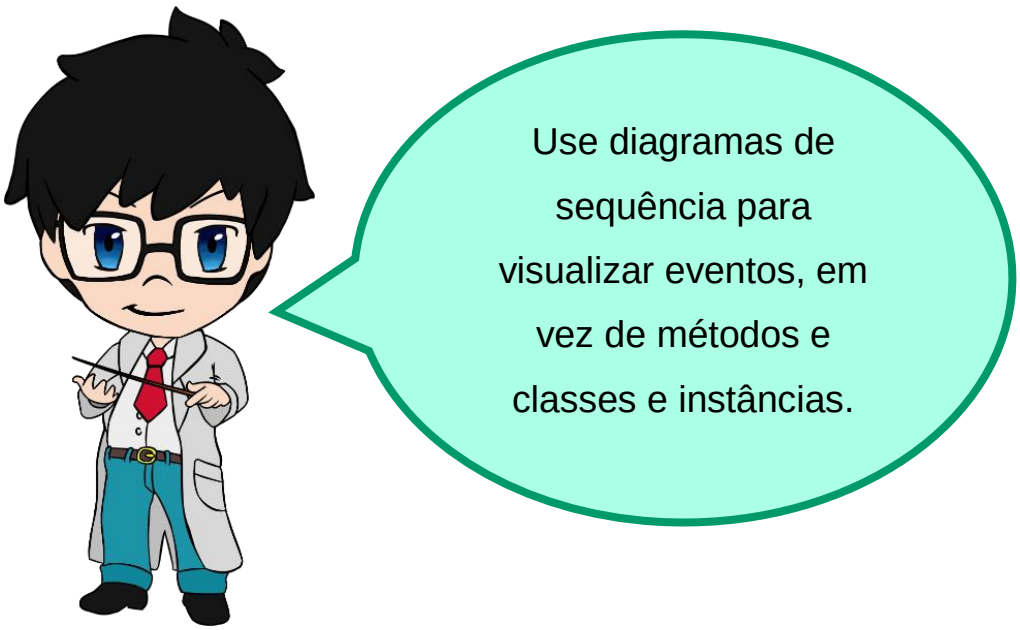
OBJETIVO: Entender os princípios e os diagramas da UML.



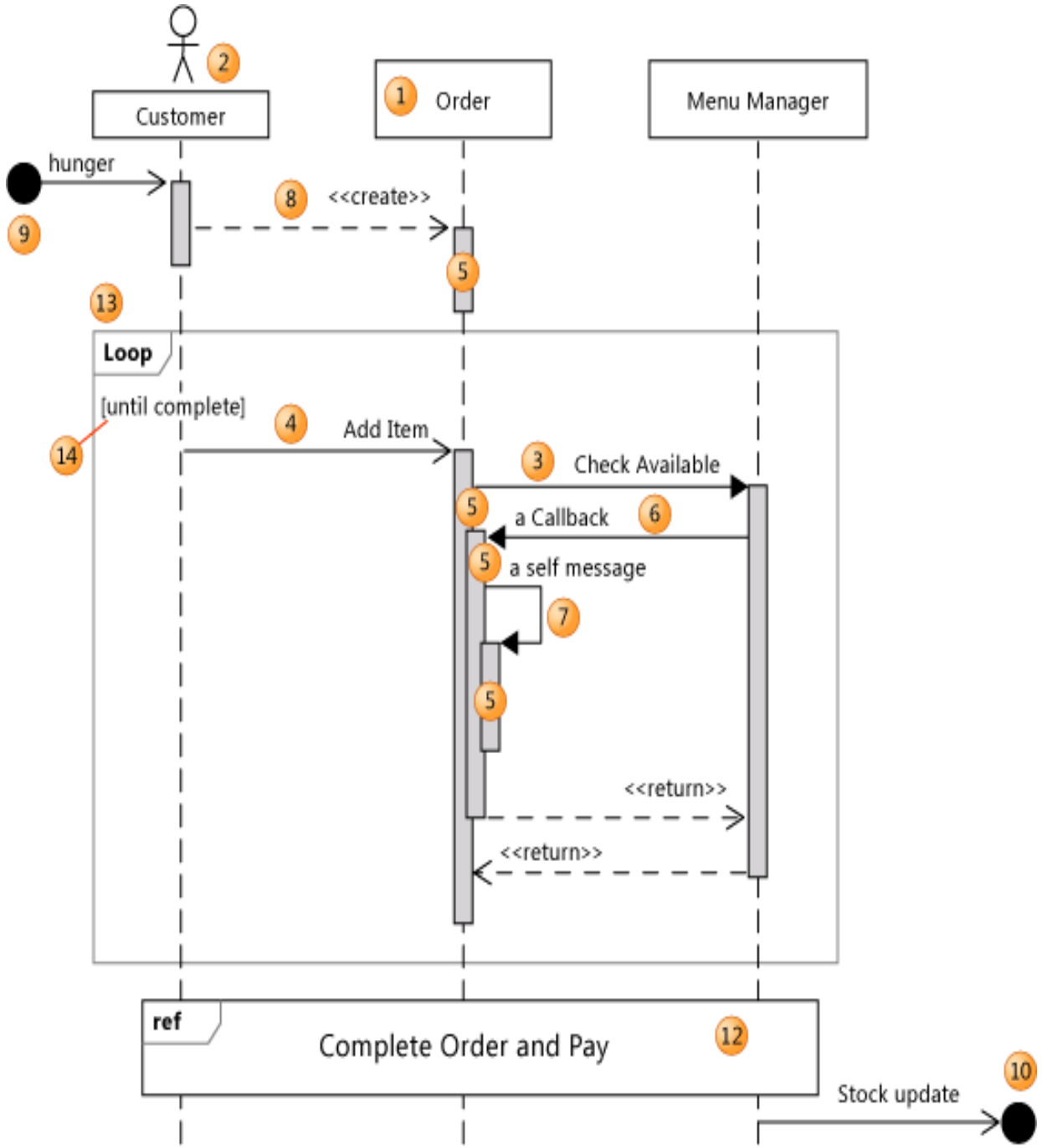
Introdução

Diagramas de sequência UML: referência

Um *diagrama de sequência* mostra uma interação, que representa a sequência de mensagens entre instâncias de classes, componentes, subsistemas ou atores. Tempo flui para baixo no diagrama e mostra o fluxo de controle de um participante para outro.



Mais de uma instância do mesmo tipo pode aparecer no diagrama. Também pode aparecer mais de uma ocorrência da mesma mensagem.


Customer
Order
Menu Manager
hunger
<<create>>
Loop
[until complete]
Add Item
Check Available
a Callback
a self message
<<return>>
<<return>>
ref
Complete Order and Pay
Stock update

Propriedade	Padrão	Elemento	Descrição
Nome	Um nome padrão	Todos	Identifica o elemento.
Nome Qualificado	Pacote: nome	Todos	Identifica exclusivamente o elemento. O nome qualificado do pacote que contém o prefixo.
Itens de Trabalho	0 associados	Todos	O número de itens de trabalho associados a esse elemento. Para associar itens de trabalho, consulte Vincular elementos de modelo e itens de trabalho.
Descrição	(em branco)	Todos	Você pode fazer anotações gerais sobre o item aqui.
Cor	(padrão para o tipo de elemento)	Linha da vida, mensagens	A cor da forma. Esta é uma propriedade da forma, e não o elemento exibe.
Tipo	(em branco)	Linha da vida	O tipo da instância que representa a linha da vida. Se houver um símbolo de referência exibido no

			cabeçalho da linha de vida, depois dessa classe ou interface existe separadamente no Gerenciador de modelos UML e pode ser exibido em um diagrama de classes.
Ator	Falso	Linha da vida	Indica se a linha da vida representa um componente de usuário, dispositivo ou software que é externo ao componente que trata do diagrama.
Espécie	Concluir -uma mensagem que contém o remetente e o receptor. Não encontrado - uma mensagem que possui um remetente não especificado. Perdidos -uma mensagem que possui um receptor não especificado.	Mensagem	Indica as extremidades de uma mensagem anexadas a uma linha da vida. Você não pode alterar essa propriedade. Ela é definida quando você cria a mensagem.

Classificar	AsynchCall -uma mensagem assíncrona. SynchCall -uma mensagem síncrona. Resposta -a parte de retorno de uma mensagem síncrona. CreateMessage - uma mensagem de criação de instância.	Mensagem	O tipo de mensagem. Você não pode alterar essa propriedade. Ele é determinado pela ferramenta que você usar para criar a mensagem.
Operação	(vazio)	Mensagem	Um método chamado pela mensagem na linha de vida de recebimento. Visível somente se a linha da vida de recebimento está vinculada a uma interface ou classe.
Refere-se a	Um diagrama de sequência	Uso de Interação	Diagrama de sequência chamado por esse uso de interação.
Operador de interação	Definir quando você usou o Circundar	Fragmento Combinado	O operador representado por esse fragmento ou uma

	com comando		coleção de fragmentos.
Proteção	(vazio)	Operando de interação em um fragmento combinado	A sequência no fragmento não ocorrerá a menos que a proteção seja true. Para selecionar o fragmento superior de qualquer fragmento combinado, clique abaixo do título do fragmento.
Min e Max	(sem restrição)	Loop de fragmento combinado	O número mínimo e máximo de vezes que o loop é executado.
Mensagens	(vazio)	Considere e Ignorar fragmentos combinados	As mensagens que são consideradas ou ignoradas neste fragmento.

Diagramas de sequência UML fazem parte de um modelo UML e existem somente em projetos de modelagem UML.

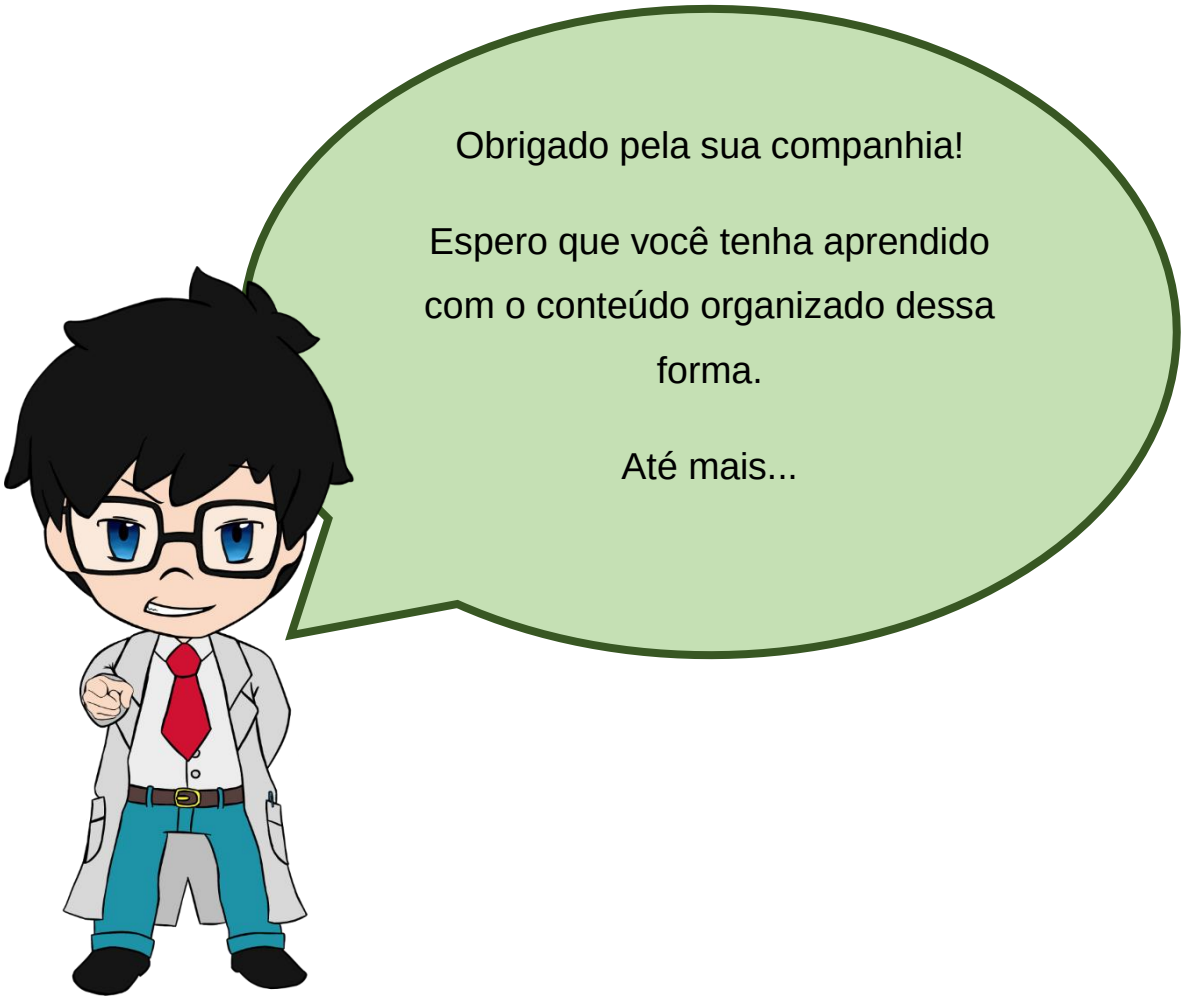
Diagramas de sequência de leitura

A tabela a seguir descreve os elementos que você pode ver em um diagrama de sequência.

Forma	Elemento	Descrição
1	Linha da vida	Uma linha vertical que representa a sequência de eventos que ocorrem em um participante durante uma interação, durante o tempo avança a linha para baixo. Este participante pode ser uma instância de uma classe, componente ou ator.
2	Ator	Um participante que é externo ao sistema que você está desenvolvendo. Você pode fazer com que um símbolo de ator aparecem na parte superior de uma linha da vida definindo seu atorpropriedade.
3	Mensagem síncrona	O remetente aguarda uma resposta a uma mensagem síncrona antes de continuar. O diagrama mostra a chamada e retorno. Mensagens síncronas são usadas para representar chamadas de função comuns dentro de um programa, bem como outros tipos de mensagem que se comportam da mesma maneira.

4	Mensagem assíncrona	Uma mensagem que não exigem uma resposta antes de continua o remetente. Uma mensagem assíncrona mostra apenas uma chamada do remetente. Use para representar a comunicação entre threads separados ou a criação de um novo thread.
5	Ocorrência de execução	Vertical sombreada retângulo que aparecerá na linha da vida do participante e representa o período em que o participante é executar uma operação. A execução começa em que o participante recebe uma mensagem. Se a mensagem inicial era uma mensagem síncrona, a execução termina com uma seta «retornar» de volta ao remetente.
6	Mensagem de retorno de chamada	Uma mensagem que retorna para um participante que está aguardando o retorno de uma chamada anterior. A ocorrência de execução resultante aparece na parte superior existente.
7	Mensagem Self	Uma mensagem de um participante a mesmo. A ocorrência de execução resultante aparece na parte superior da execução de envio.
8	Criar mensagem	Uma mensagem que cria um participante. Se um participante recebe uma mensagem de criar, deve ser o primeiro que recebe.
9	Mensagem encontrada	Uma mensagem assíncrona de um participante não especificado ou desconhecido.
10	Mensagem perdida	Uma mensagem assíncrona para um participante não especificado ou desconhecido.
11	Comentário	Um comentário pode ser anexado a qualquer ponto em uma linha da vida.
12	Uso de	Inclui uma sequência de mensagens que são definidos em outro diagrama.

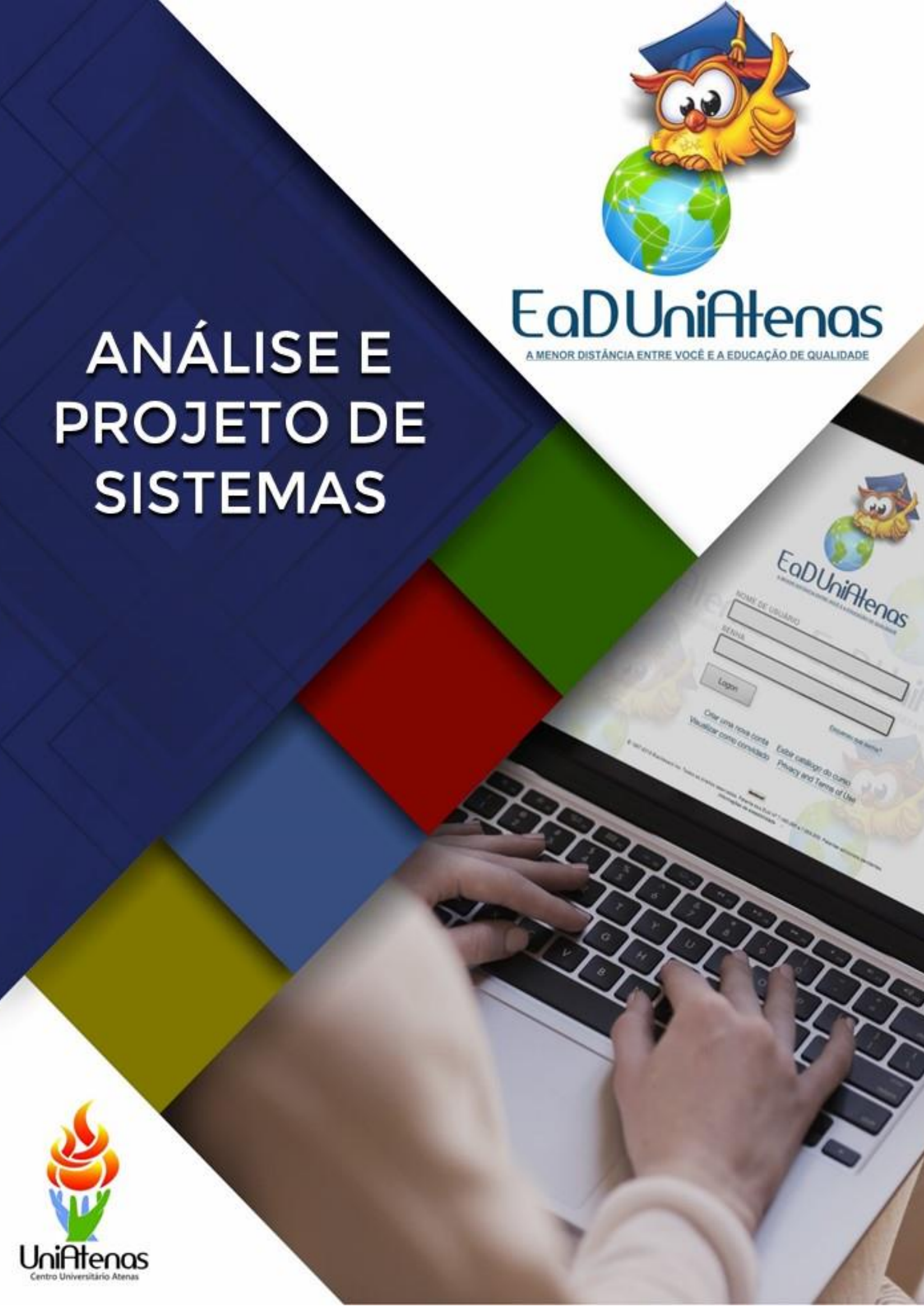
	Interação	Para criar um uso da interação, clique na ferramenta e arraste sobre as linhas de vida que você deseja incluir.
13	Fragmento Combinado	Uma coleção de fragmentos. Cada fragmento pode incluir uma ou mais mensagens. Há tipos diferentes de fragmentos combinados. Para obter mais informações, consulte Descrever o fluxo de controle com fragmentos em diagramas de sequência UML. Para criar um fragmento, uma mensagem de atalho, aponte para Circundar com, e, em seguida, clique em um tipo de fragmento.
14	Protetor de fragmento	Pode ser usado para indicar uma condição relevante se o fragmento ocorrerá. Para definir a proteção, selecione um fragmento, e em seguida, selecione o protetor e digite um valor.
X	Evento de destruição	Representa o ponto no qual o objeto é excluído ou não estar mais acessível. Aparece na parte inferior de cada linha da vida.
	Interação	A coleção de mensagens e linhas da vida que é exibida no diagrama de sequência. Para exibir as propriedades de uma interação, selecione-o no Gerenciador de modelos UML.
	Diagrama de Sequência	O diagrama que exibe uma interação. Para exibir suas propriedades, clique em uma parte vazia do diagrama. Note: Os nomes de diagrama de sequência, a interação que exibe e o arquivo que contém o diagrama pode ser todo diferente.


Obrigado pela sua companhia!
Espero que você tenha aprendido com o conteúdo organizado dessa forma.
Até mais...

EaD UniAtenas

A MENOR DISTÂNCIA ENTRE VOCÊ E A EDUCAÇÃO DE QUALIDADE

ANÁLISE E PROJETO DE SISTEMAS



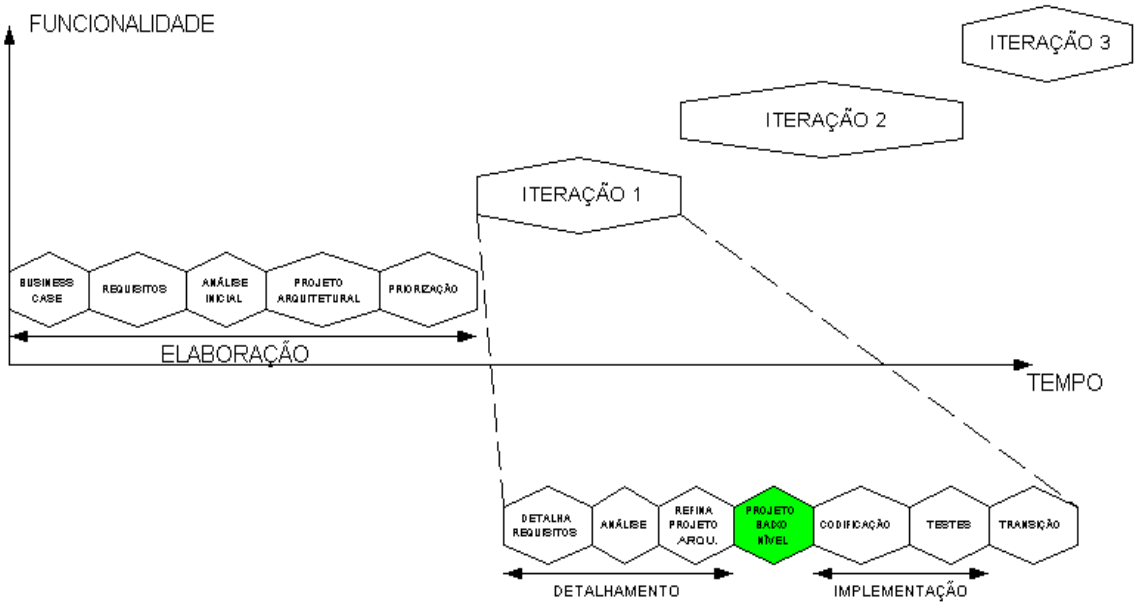
UniAtenas
Centro Universitário Atenas

DIAGRAMAS DE COLABORAÇÃO

OBJETIVO: Mostrar como as pós-condições dos contratos serão realizadas.

Introdução

Diagramas de Colaboração



Dois tipos de diagramas podem ser usados para mostrar as interações (mensagens) entre objetos:

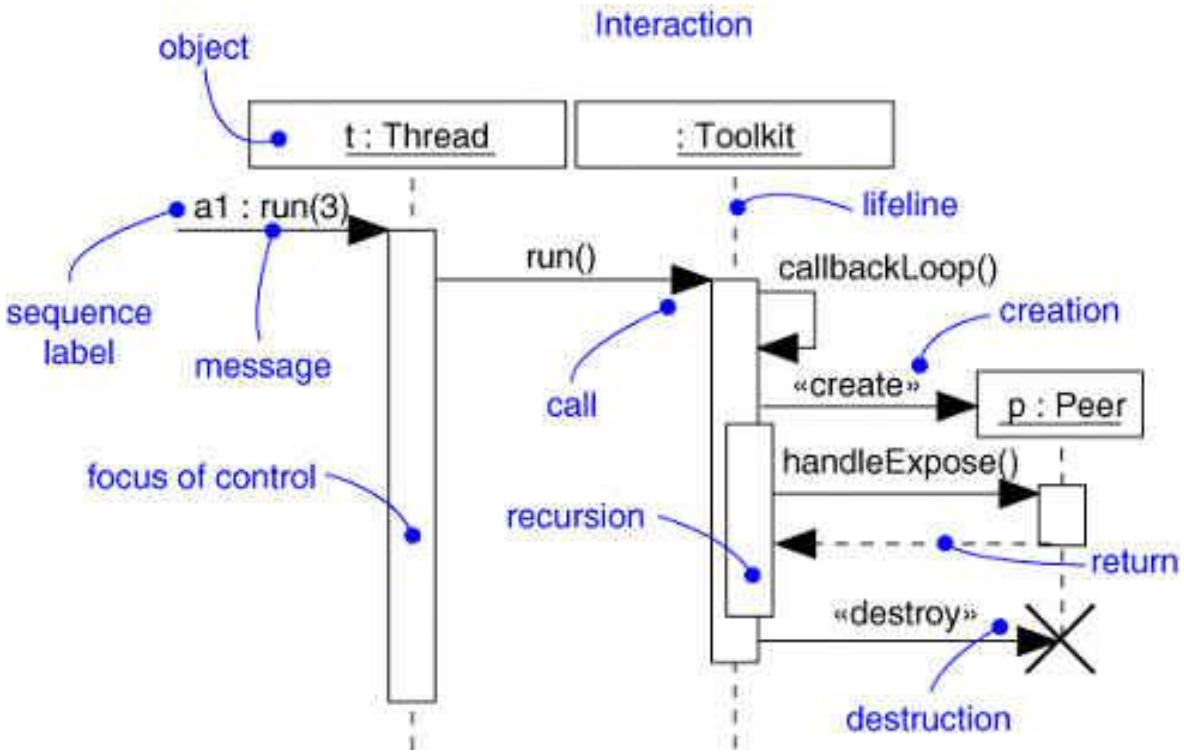
Diagramas de Sequência;

Diagramas de Colaboração.

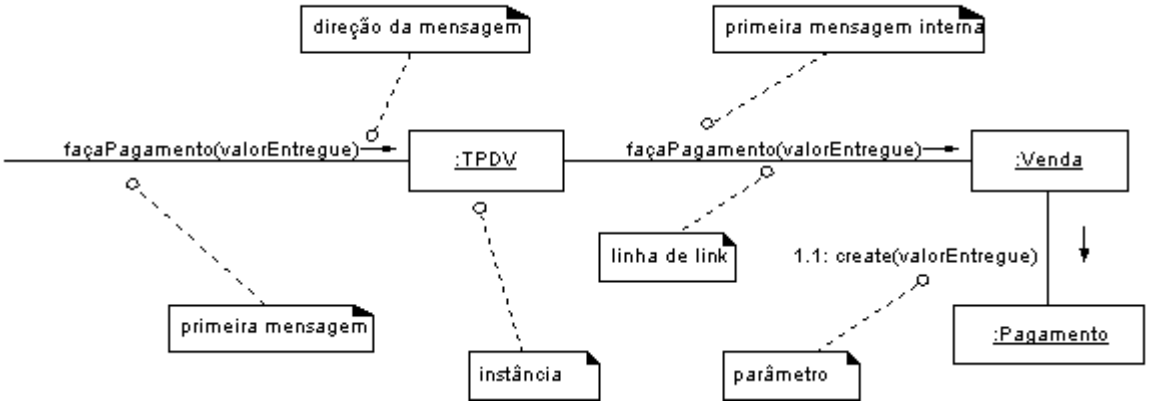
Os dois tipos de diagramas são chamados diagramas de interação. O objetivo é de mostrar como as pós-condições dos contratos serão realizadas.

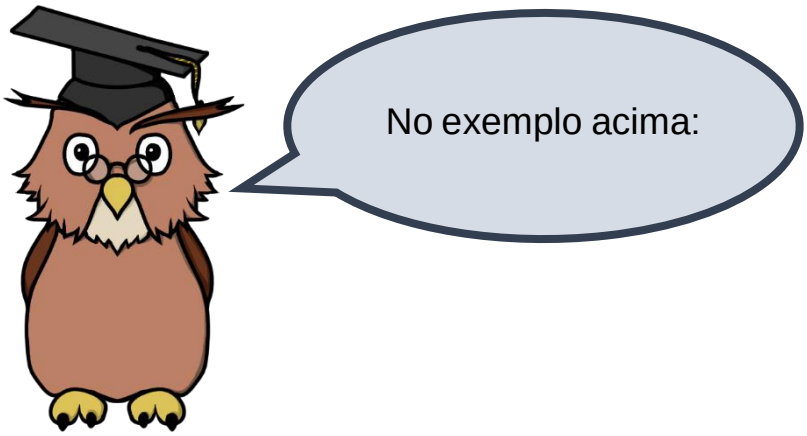
O diagrama de sequência é mais simples de usar quando se deseja mostrar apenas as sequências de interações já o diagrama de colaboração é mais adequado quando se deseja expressar mais detalhes da colaboração entre objetos.

Exemplo de um diagrama de sequência:



Exemplo de um diagrama de colaboração:





- A mensagem faça Pagamento é enviada a uma instância de uma TPDV.
- O TPDV envia a mensagem façaPagamento a uma instância de Venda.
- O objeto da classe Venda cria uma instância de um Pagamento.

Sobre a importância de diagramas de interação:

Uma das coisas mais difíceis de fazer no projeto de um sistema é a atribuição de responsabilidades a objetos e a consequente colaboração entre objetos, os diagramas de interação ajudam muito a construir o sistema e uma boa parcela do tempo deve ser dedicado à sua construção é principalmente aqui que bons princípios de projeto serão usados.

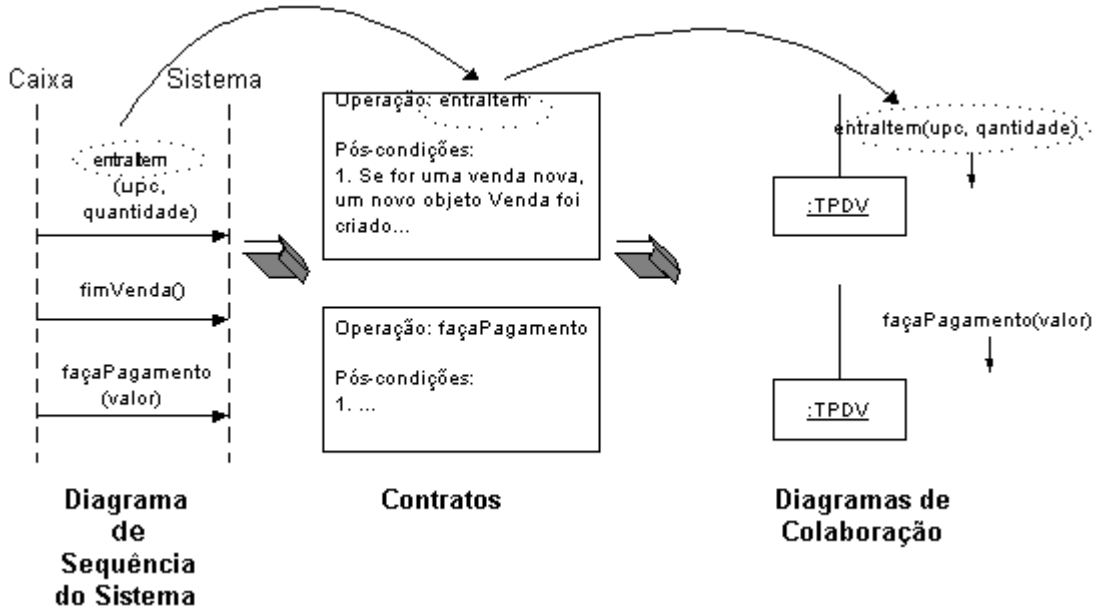
Esta seção discute apenas a notação empregada em diagramas de colaboração. Seções subsequentes tratarão da distribuição de responsabilidades entre objetos e apresentar padrões de projeto.

Como criar diagramas de colaboração

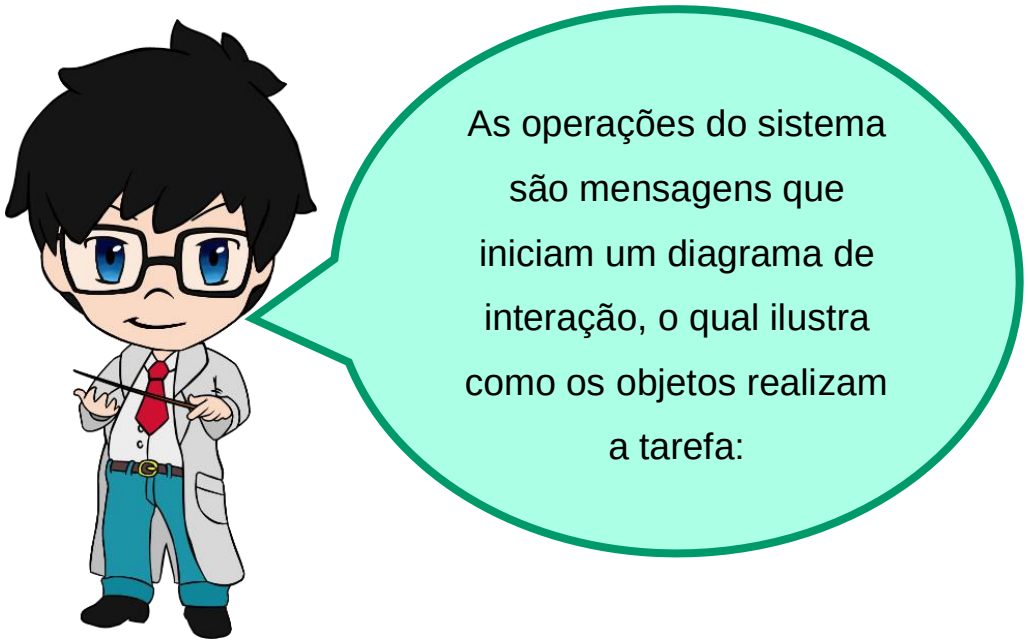
Criar um diagrama separado para cada operação do sistema sendo desenvolvida na iteração corrente. Para cada mensagem de operação do sistema, um diagrama é consttuído com essa mensagem inicial. Se o diagrama ficar complexo (não cabe numa única página), quebre-o em diagramas menores.

Usando o contrato das operações (principalmente as pós-condições) e os Use Cases como ponto de partida, projete um sistema de objetos interagindo entre si para realizar as tarefas. Aplique padrões de projeto para desenvolver um bom projeto.

Relação entre diagramas de colaboração e outros artefatos.

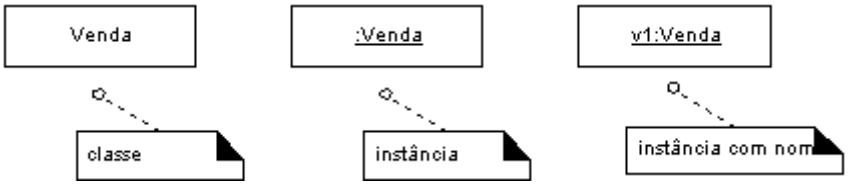


O Use Cases sugerem os eventos do sistema, os quais são mostrados explicitamente nos diagramas de sequência do sistema. Uma primeira aproximação do efeito das operações do sistema é descrita nos contratos.



Notação básica para diagramas de colaboração.

Classe e instâncias



Links

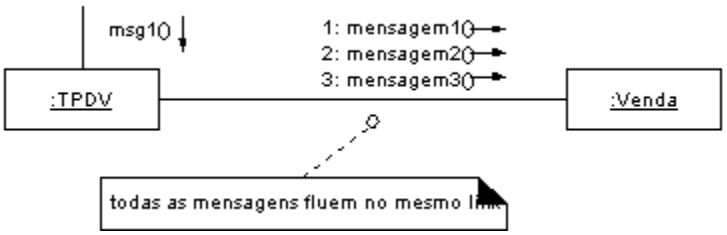
Um link é uma conexão entre dois objetos;

É uma instância de uma associação;

Indica alguma forma de navegabilidade e visibilidade.

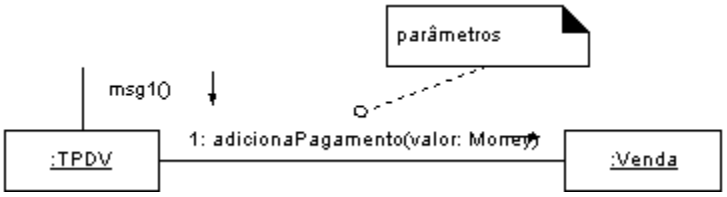
Mensagens

Observe o número de sequência das mensagens:

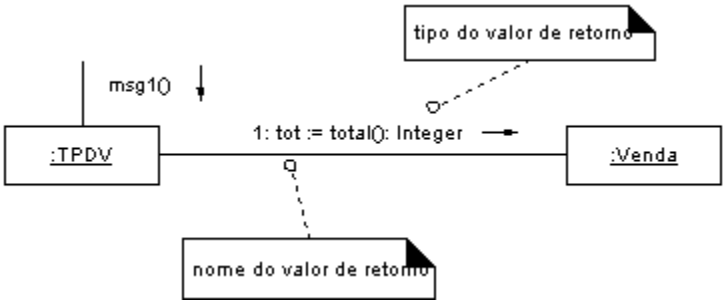


Parâmetros

O tipo do parâmetro é opcional.



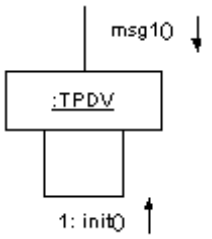
Valor de retorno



Sintaxe de mensagem

Pode-se usar a sintaxe UML (como acima) ou de outra linguagem (Java, por exemplo).

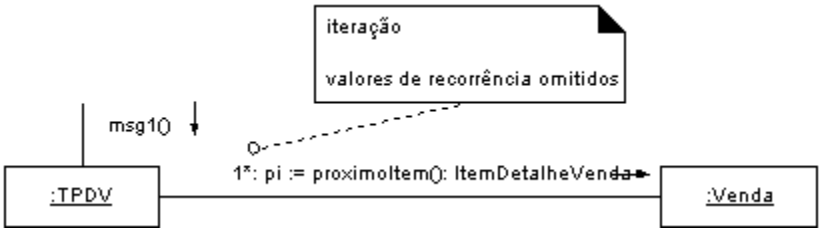
Mensagens a "this"



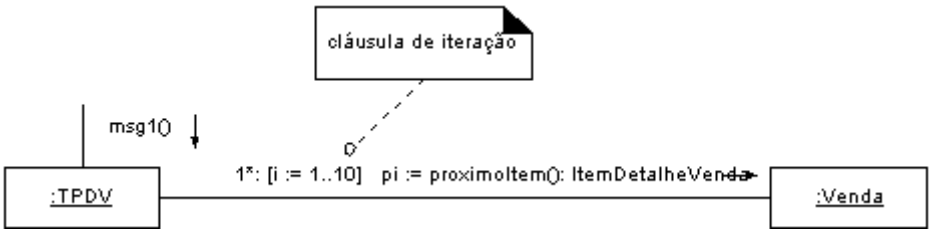
Iteração

A iteração é mostrada com um número de sequência e um *.

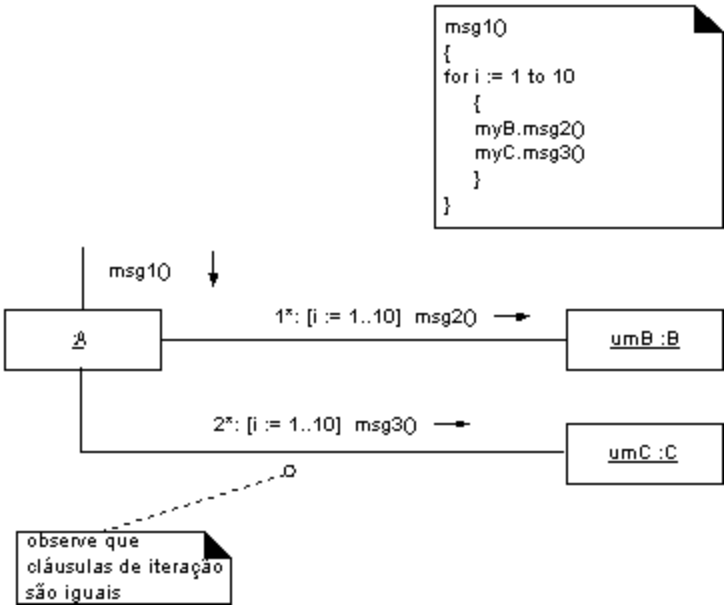
A mensagem é enviada repetidamente.



Valores de recorrência podem ser incluídos.



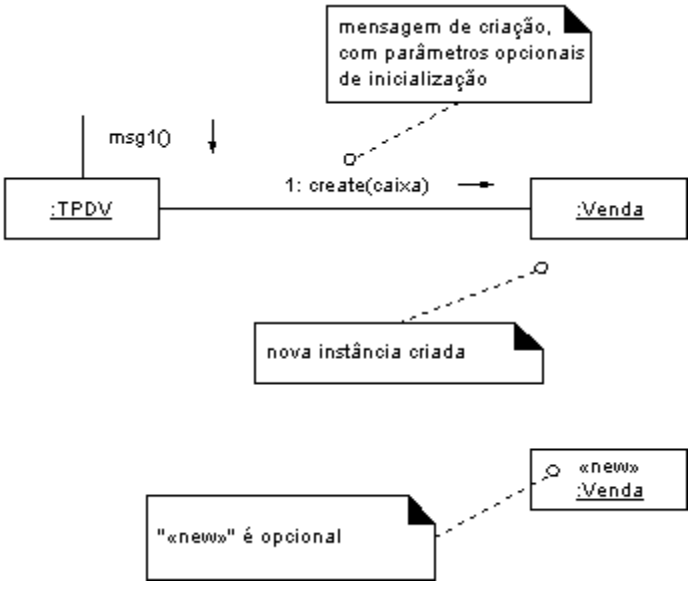
Mais de uma mensagem pode ser enviada na iteração.



Criação de instâncias

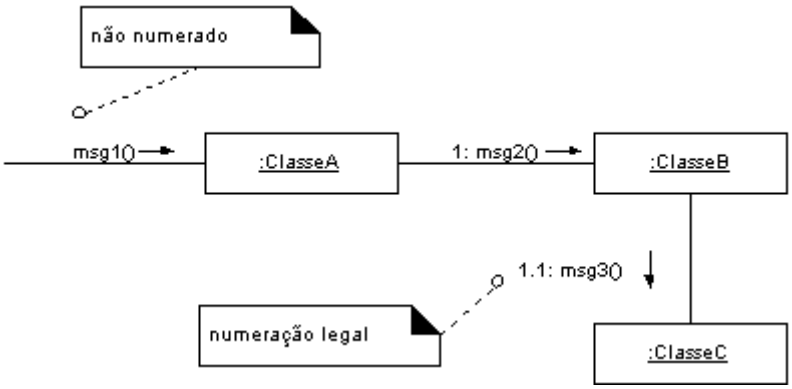
A mensagem de criação independente de linguagem é "create".

O estereótipo «new» pode ser usado.

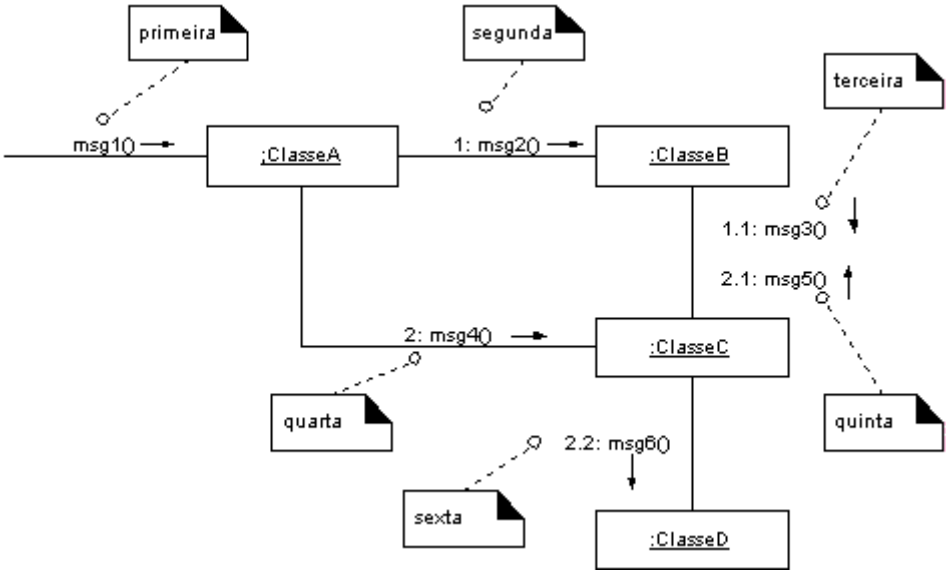


Sequenciamento de mensagens

A primeira mensagem não é numerada.

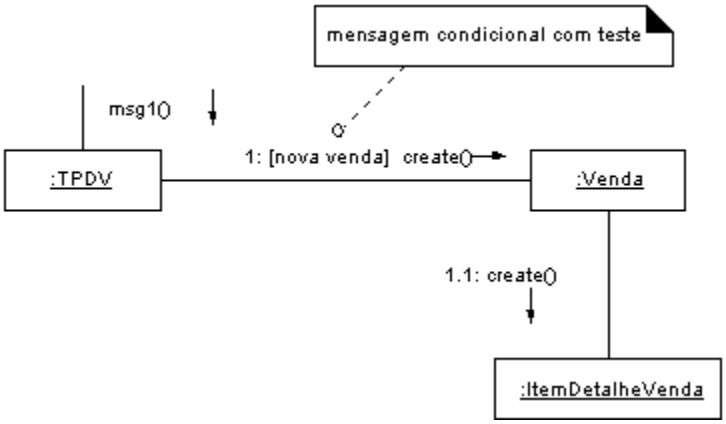


Tem várias alternativas para numerar as demais mensagens, incluindo um esquema hierárquico.

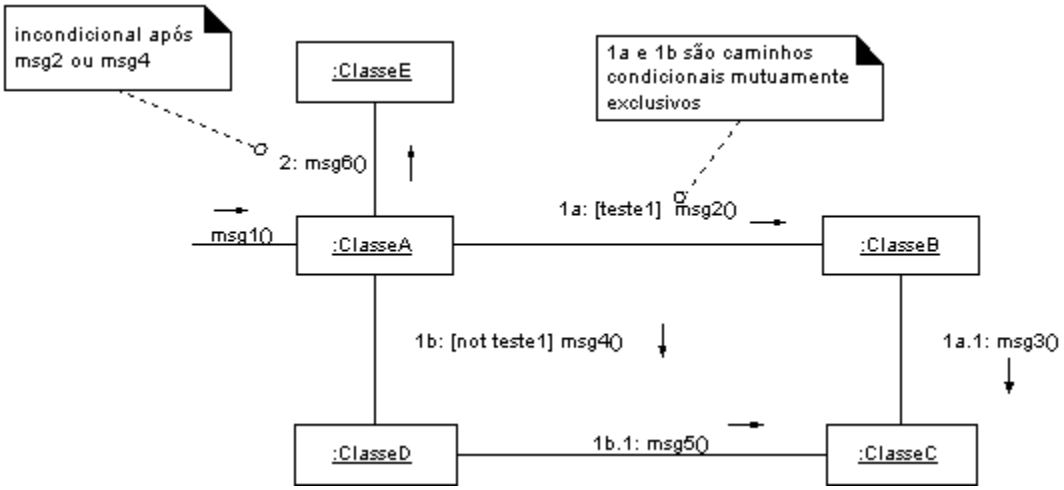


Mensagens condicionais

A mensagem só é enviada se o teste resultar em TRUE.

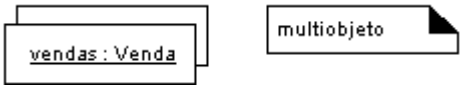


Caminhos condicionais mutuamente exclusivos.



Coleções

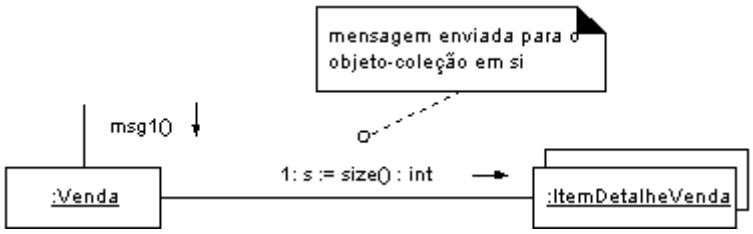
Um conjunto de instâncias (multiobjeto).



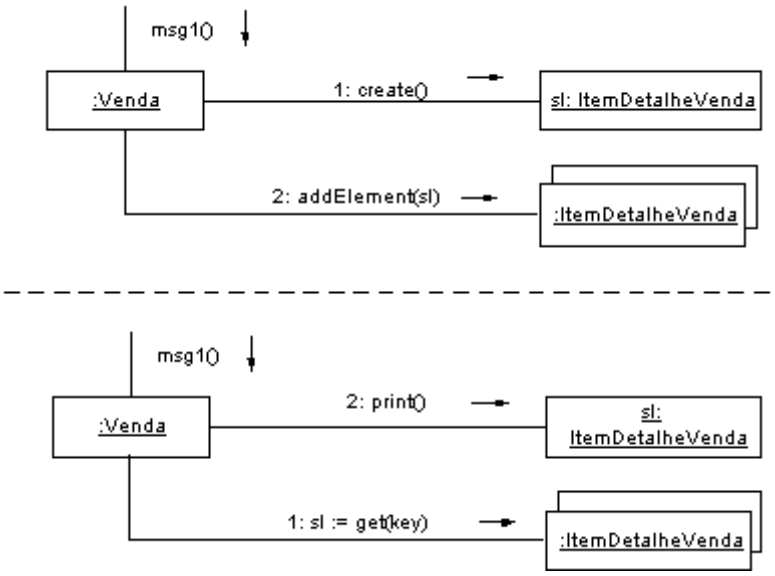
Mensagens para coleções.

Na UML 1.1, uma mensagem enviada para uma coleção vai para o objeto-coleção e *não* para todos os objetos da coleção.

Na UML 1.0, a mensagem ia para todos os objetos.



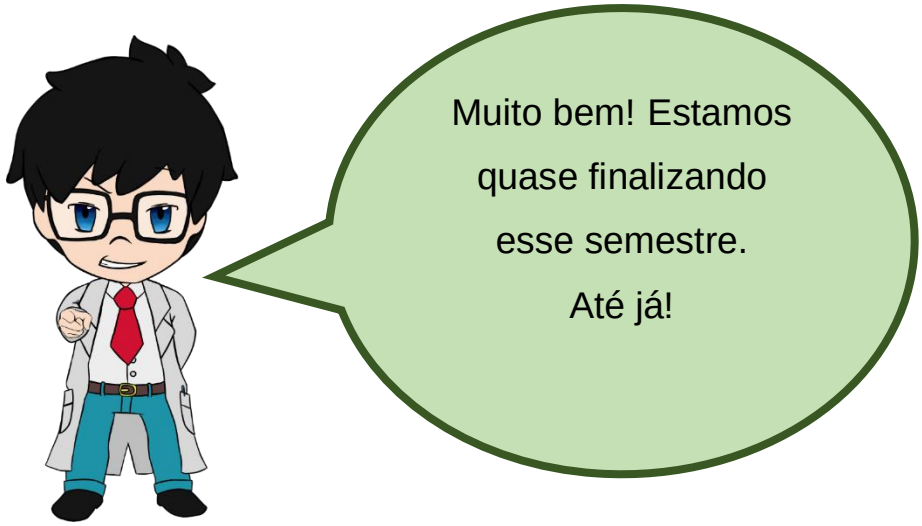
Outros exemplos de mensagens para objetos e coleções.



Mensagens para a classe (métodos estáticos)

Classe não está sublinhada.

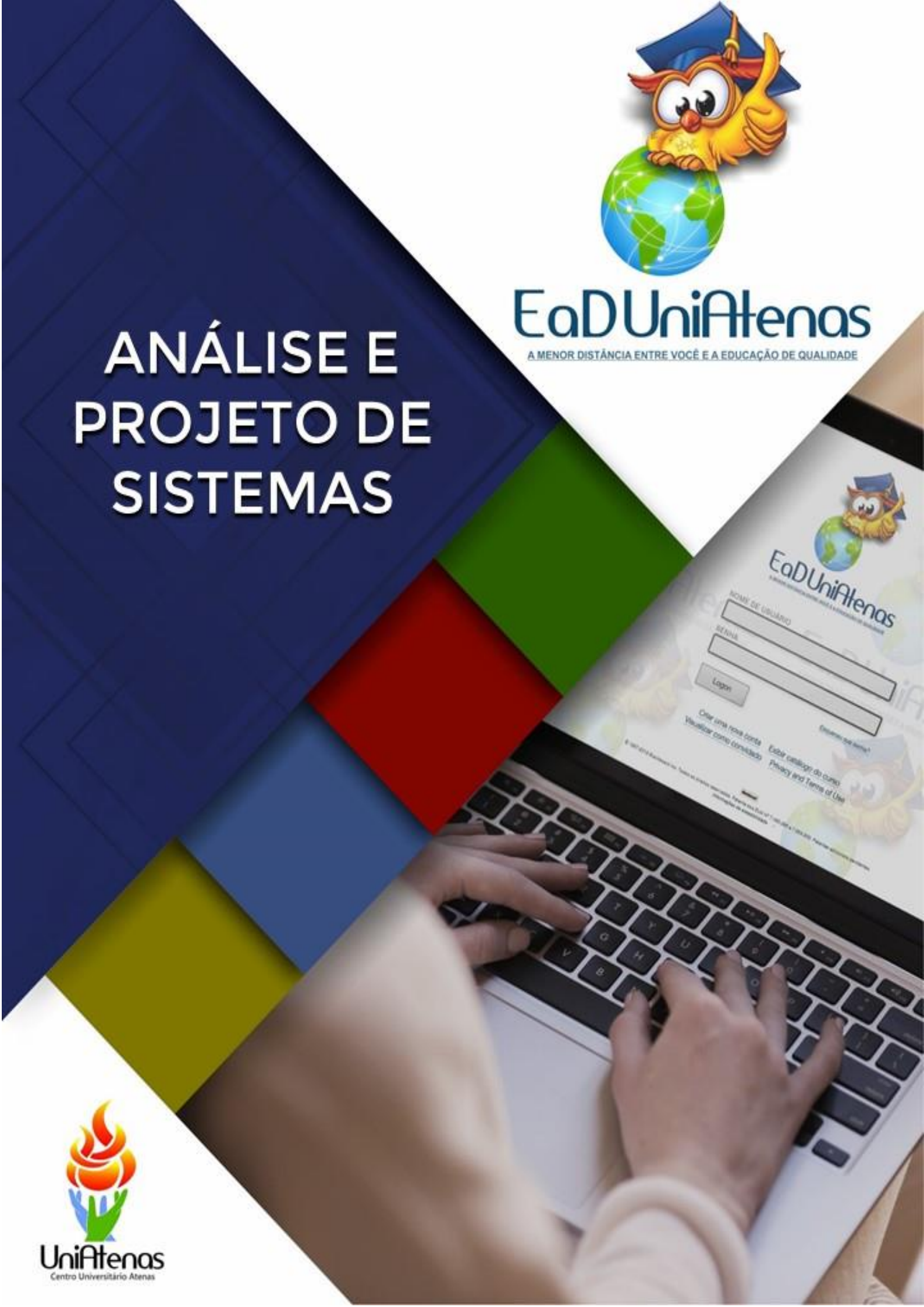
Sempre uma classe terá um construtor padrão, mesmo não sendo declarado o compilador irá fornecer um. Esse construtor não recebe argumentos e existe para possibilitar a criação de objetos para uma classe.



EaD UniAtenas

A MENOR DISTÂNCIA ENTRE VOCÊ E A EDUCAÇÃO DE QUALIDADE

ANÁLISE E PROJETO DE SISTEMAS

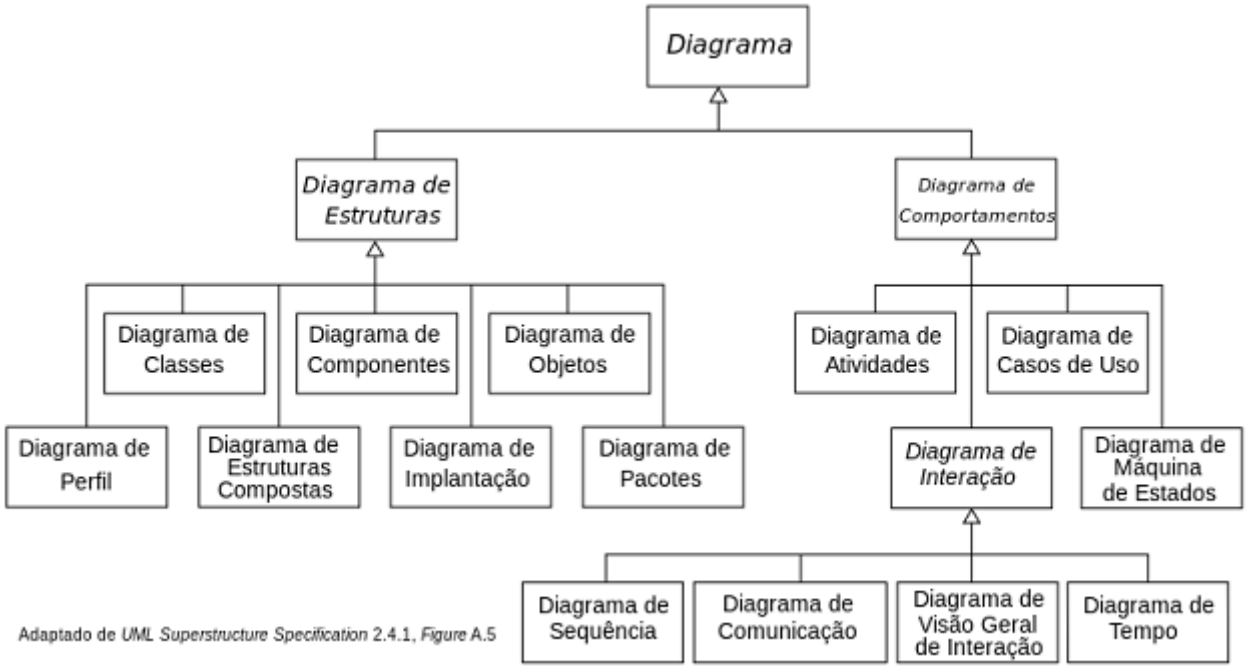


UniAtenas
Centro Universitário Atenas

DIAGRAMAS DE ATIVIDADES

OBJETIVO: Especificar o comportamento do software.

Introdução



Adaptado de *UML Superstructure Specification* 2.4.1, *Figure* A.5

No contexto da UML, o Diagrama de Atividades é um diagrama comportamental (que especifica o comportamento do software), e através dele podemos modelar partes do comportamento de um software.

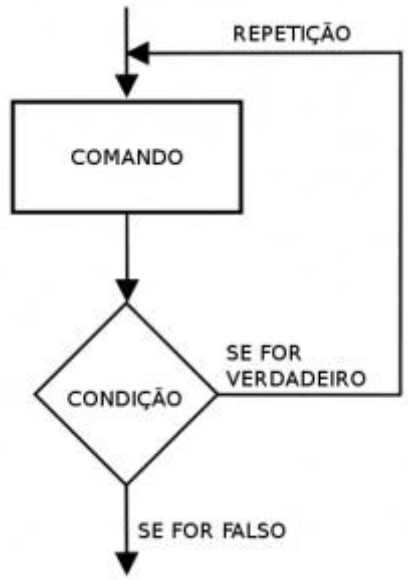
Em projetos de software utiliza-se modelos para representar tanto a estrutura quando o comportamento do sistema e com base neles construir, programar o modelo executável, que é o sistema materializado.

Estrutura representa aquilo que é estático, que não muda com o uso do sistema, não muda de estado, não movimenta.

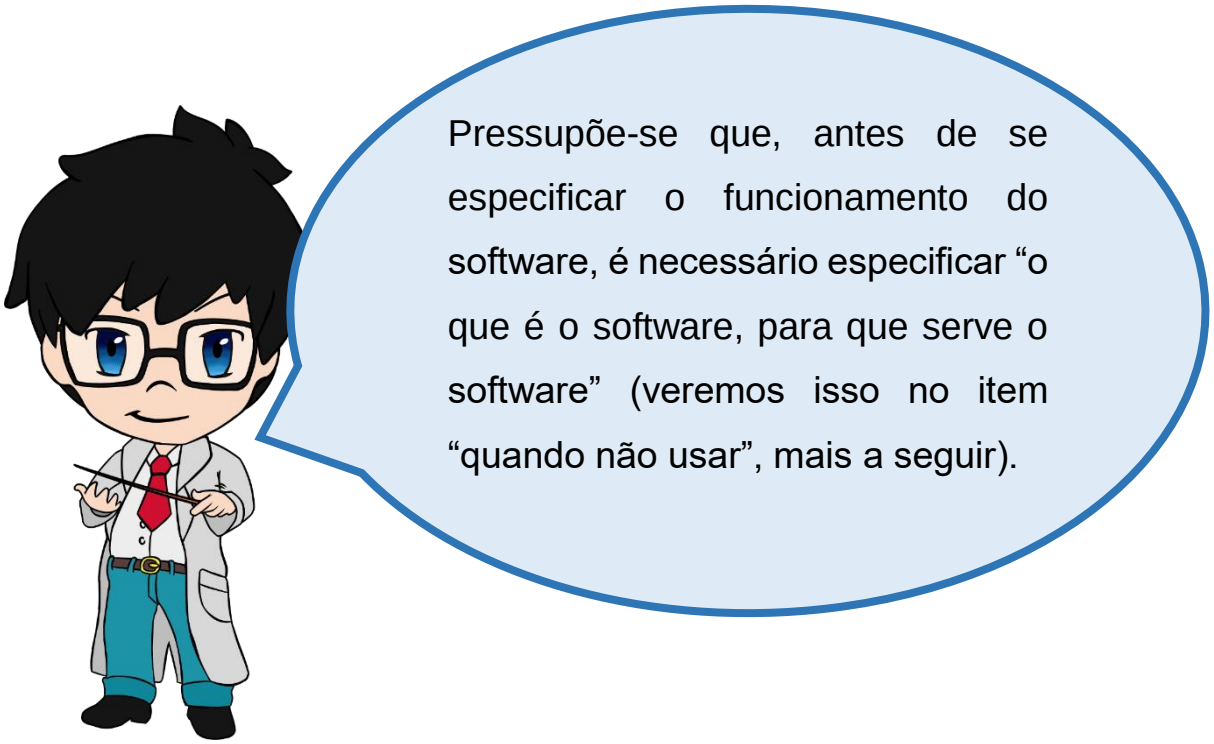
Comportamento é o que é dinâmico no sistema, que se altera a partir de ações do próprio sistema ou do usuário”.

O diagrama de atividades ilustra graficamente como será o **funcionamento** do software (em nível micro ou macro), como será a **execução** de alguma de suas partes, como será a **atuação** do sistema na realidade de negócio na qual ele está inserido.

Objetivos de utilização:



O diagrama de atividades, como citado, tem como objetivo principal a especificação do comportamento do software, **do ponto de vista funcional**, ou seja, das suas funcionalidades. É muito semelhante a um fluxograma, uma ferramenta utilizada há muitas décadas, principalmente na administração.



E ainda, como para qualquer outro modelo que segue a notação UML, o objetivo de um diagrama é especificar o que será posteriormente projetado, ou diretamente construído, diminuindo assim o nível de abstração do escopo, facilitando o entendimento sobre o que tem ser feito pelo programador, por exemplo.

Com isso, programadores, por exemplo, podem entender de uma maneira “mais lógica” e “menos abstrata” o que deverá ser codificado no modelo executável.

Quando usar:

Na prática, é um diagrama como o bombril: tem mil e uma utilidades. Mas não significa que isso é bom.

Usar uma faca para apertar um parafuso pode até funcionar, mas pode machucar a mão por não ser a ferramenta apropriada para isso, ou ainda, estragar o parafuso.

O diagrama de atividades apresenta uma simplicidade muito tentadora, e em função disso, muitos analistas utilizam o diagrama para modelagem de processos, modelagem de algorítimos, modelagem de sequência etc., quando na realidade, existem diagramas apropriados para isso na UML ou na BPMN.

Documentar o aspecto funcional (não estrutural) do software, quando é necessário representar o fluxo da informação que o software trabalhará, e quando existem condições/decisões que precisam detalhadas/descritas.

Mostrar **aspectos específicos** de alguma rotina do negócio que será automatizada pelo software, como um “zoom” em parte de alguma funcionalidade, por exemplo. Obs.: muito cuidado ao especificar toda uma funcionalidade (uma tela ou rotina batch por exemplo) num diagrama de

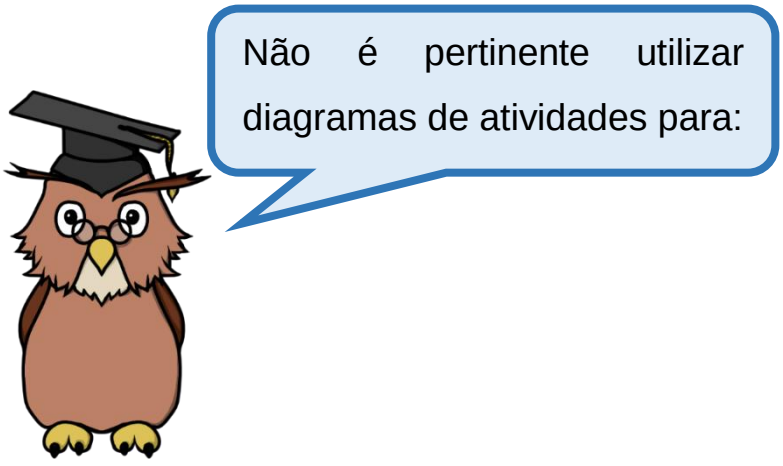
atividades. Geralmente isso gera diagramas de atividades super complexos, o que pode gerar efeito inverso (ou seja, “subtrair mais que somar”).

Mostrar como Funcionalidades vão realizar Requisitos Funcionais (funções executadas pelas funcionalidades), e a relação dos Requisitos Funcionais com as Regras de Negócio.

Documentar de forma **macro** como o sistema irá funcionar, mas orientado ao software, não ao processo de negócio. Mostrar como os módulos do sistema interagem entre si, as principais as informações trafegadas durante a execução do software, entradas e saídas principalmente.

É importante deixar claro que a documentação só tem razão de existir quando serve para explicar algo para alguém, e deixar uma memória viva e “útil” do projeto do software. Diferente disso é perda de tempo, dinheiro, e aumento da ineficiência do processo produtivo através do aumento desnecessário da complexidade do projeto.

Quando não usar



Elicitação de requisitos, pois o foco do diagrama de atividades é no **comportamento**, **no aspecto "funcional"** do software. E antes de se pensar no funcionamento do software é fundamental se pensar no **conceito do sistema**, focando no problema de negócio, e na solução que será aplicada para resolvê-lo. Para isso, utilize Digramas de Bloco ou Diagramas de Decomposição Funcional, por exemplo.

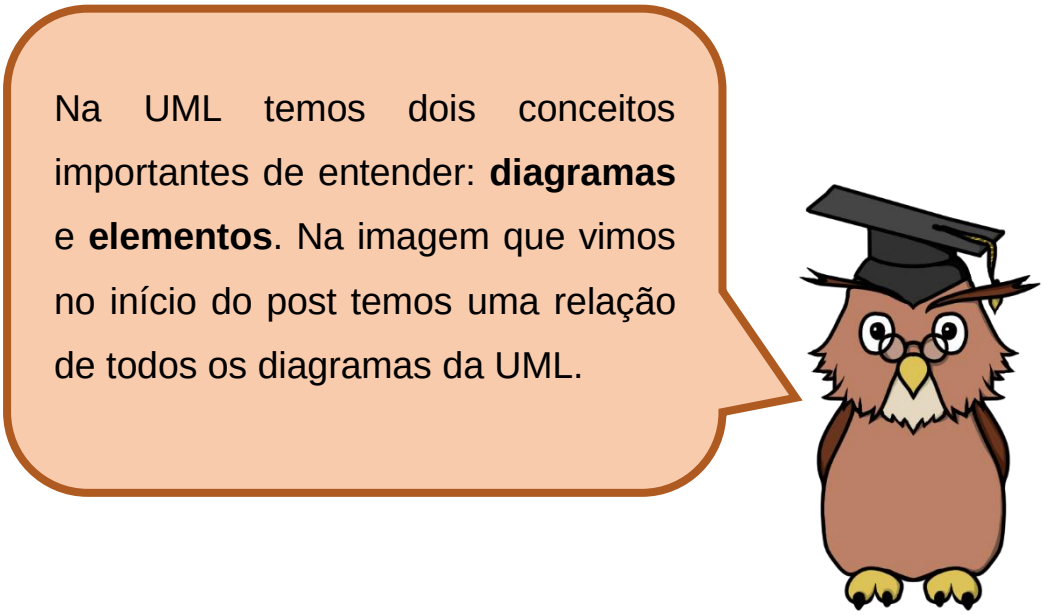
Um dos maiores problemas do Analista de Sistemas é dar atenção ao "micro" quando ainda não entendeu o "macro". Antes de pensar no funcionamento do software é necessário pensar no negócio que o software vai automatizar, qual o conceito dessa solução, e o que o software deverá fazer e não como ele deverá fazer. Isso se faz com Engenharia de Requisitos, e depois dos requisitos especificados, estes são "distribuídos" nas funcionalidades do software, que os realizam.

Quando o assunto é Modelagem de Processos de Negócio. **Modelar processos não é modelar software**. Software pode ser utilizado para realizar **algumas** atividades/tarefas do processo de negócio, mas **são coisas muito distintas**. Existem profissionais que utilizam diagramas de atividades para modelar processos, e diagramas BPMN para modelar software. São notações diferentes, e para quem está acostumado com ambos, pode gerar confusão.

Modelar o comportamento completo de telas e rotinas batch. Para essas finalidades, diagramas de Caso de Uso ou diagramas de Sequência **são opções muito melhores**. Utilize o primeiro quando o foco for mais no aspecto funcional da tela ou da rotina e quando usuários ou analistas menos técnicos estiverem envolvidos, e o segundo quando o

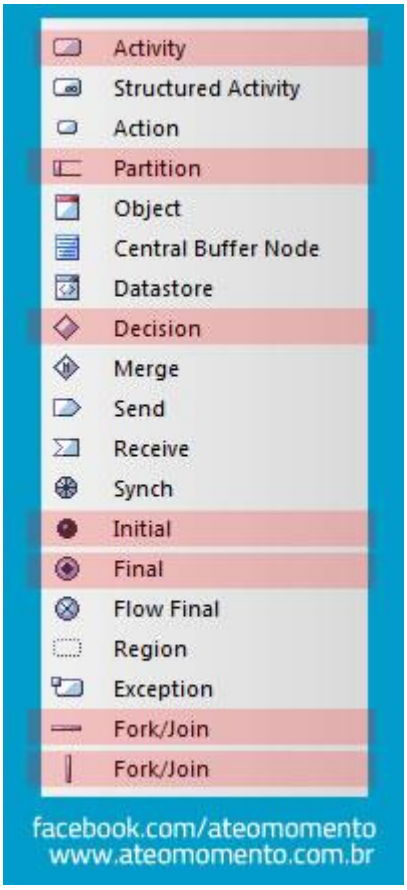
foco for mais no design (projeto) do software, quando programadores ou arquitetos estiverem envolvidos.

Como usar



As formas gráficas que compõe cada diagrama são chamadas “elementos”. Estes elementos são “o grande lance” da UML, é o que sustenta a ideia de “notação”, é a sintaxe contida nos diagramas. Cada elemento possui um objetivo específico, e a combinação destes elementos torna-se o diagrama, que gera a semântica do respectivo modelo.

Como tudo na vida, na UML também se aplica o Princípio de Pareto. Com 20% dos elementos fazemos 80% dos diagramas. Então, vou focar nos elementos mais utilizados do diagrama de atividades.



Activity – É a atividade propriamente dita. Usamos este elemento quando citamos uma atividade no diagrama. Por exemplo: “Processar Pedido” é uma atividade que seria ilustrada com esta forma.

Partition – É comum chamarmos de “Raia”, fazendo uma analogia com as raias de uma piscina. Podem ser representadas na vertical ou na horizontal. Ilustram fronteiras entre módulos, funcionalidades, sistemas ou sub-sistemas, conforme o nível de detalhe e foco do diagrama.

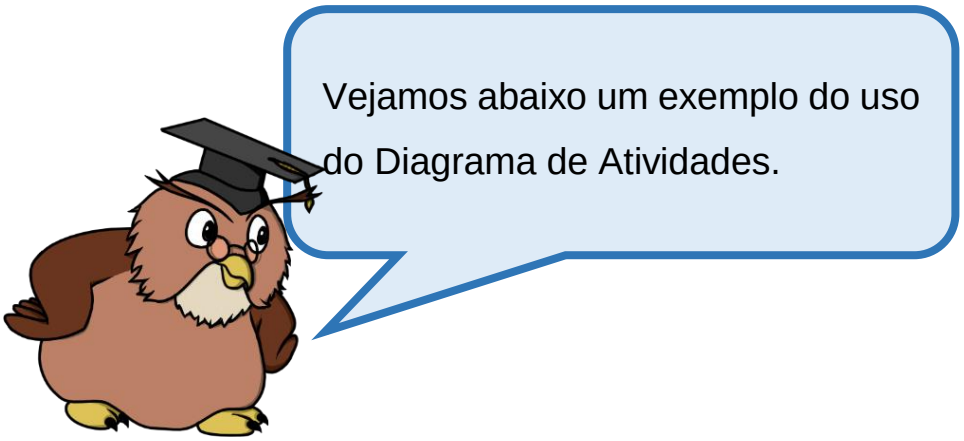
Decision – Representa uma decisão que pode desviar o fluxo ilustrado no diagrama. Utilizado quando lidamos com condições. Por exemplo: “Pagamento aprovado? Se sim, desvia para a atividade Gerar Boleto, se não, vai para atividade “Pagar novamente”.

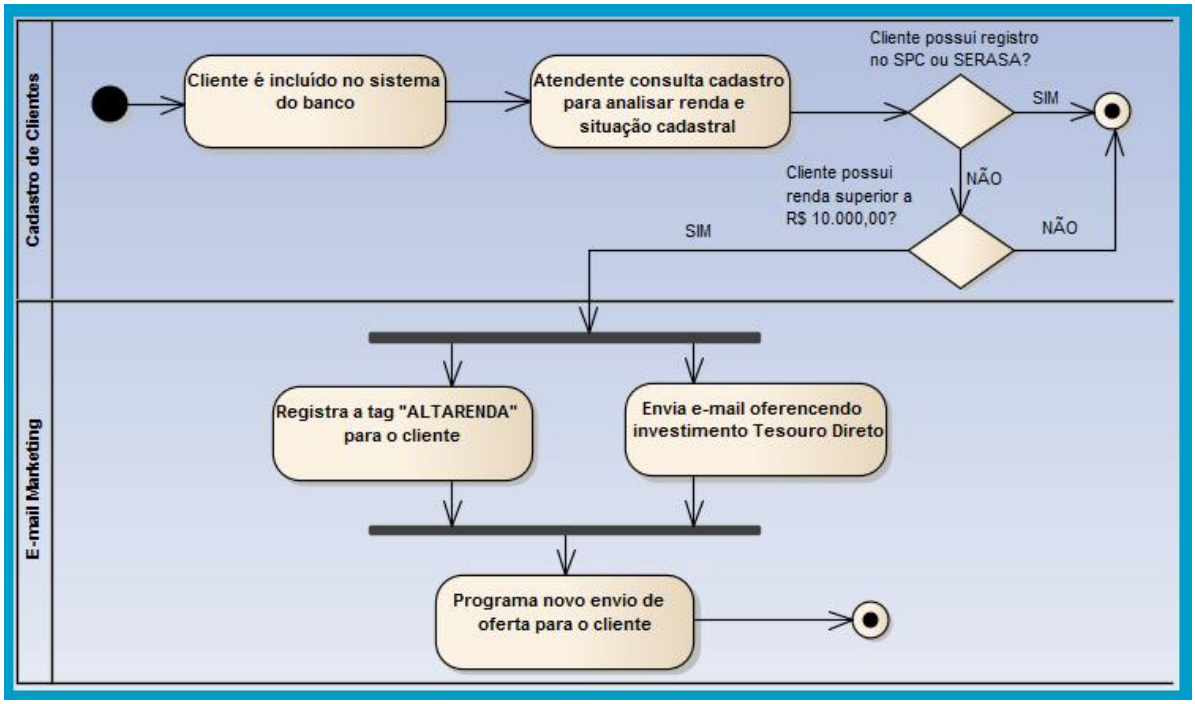
Initital – É o primeiro elemento do diagrama. Define o início do fluxo. Um diagrama de atividades pode ter mais de um elemento deste, pois seu início pode ser dar em mais de um “local”.

Final – É o último elemento do diagrama. Define o fim do fluxo. Um diagrama de atividades pode ter mais de um elemento deste também, pois o fim do fluxo pode ocorrer em vários partes do diagrama. O ideal é utilizar o elemento “Flow Final”, mas é um conceito mais avançado.

Fork/Join – Na imagem temos dois destes elementos, um na horizontal e outro na vertical. O objetivo é o mesmo para ambas as formas. O Fork tem como finalidade dividir o fluxo em mais de uma direção, e o Join tem finalidade inversa, ou seja, faz a união de várias direções do fluxo em uma única direção.

Exemplo de Uso:



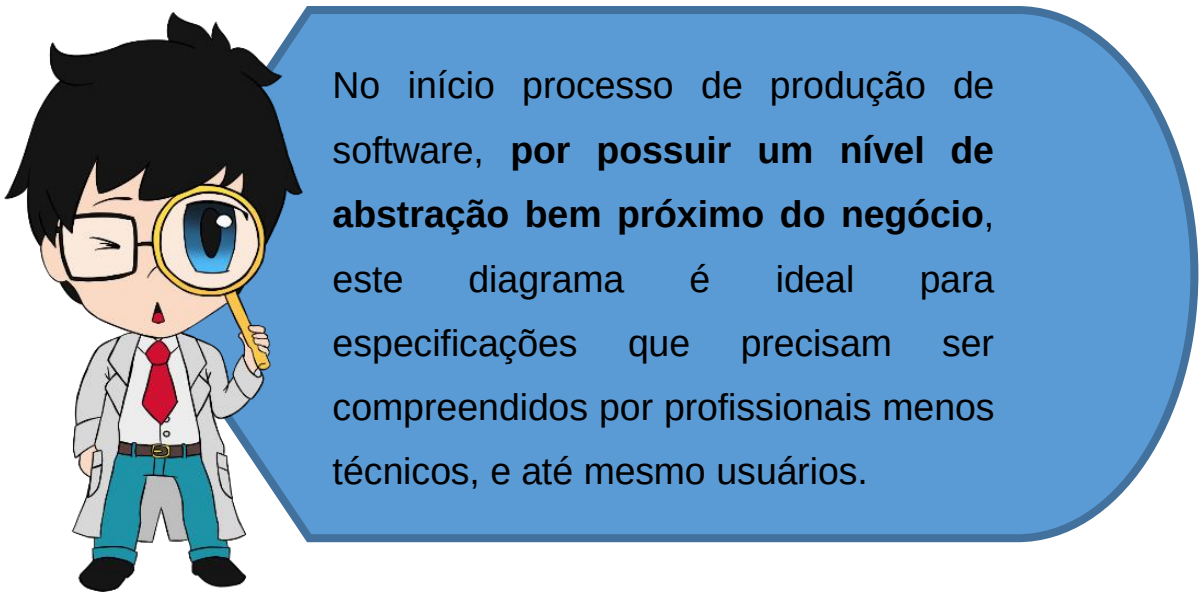


O exemplo é simplório, apenas para fins didáticos. Basicamente, temos referências a dois módulos nas duas Partitions (Cadastro de Cliente e E-mail Marketing), e trata-se de um fluxo do sistema, onde um cliente após ser cadastrado sofre uma avaliação, e dependendo do resultado da avaliação (feita através do software) o fluxo pode tomar caminhos diferentes.

Se todo o fluxo completar-se, antes de encerrar-se, o cliente vai para uma situação de “espera”, onde outro fluxo, por exemplo, tratará o envio de uma nova oferta ao cliente que passou em todas as etapas.

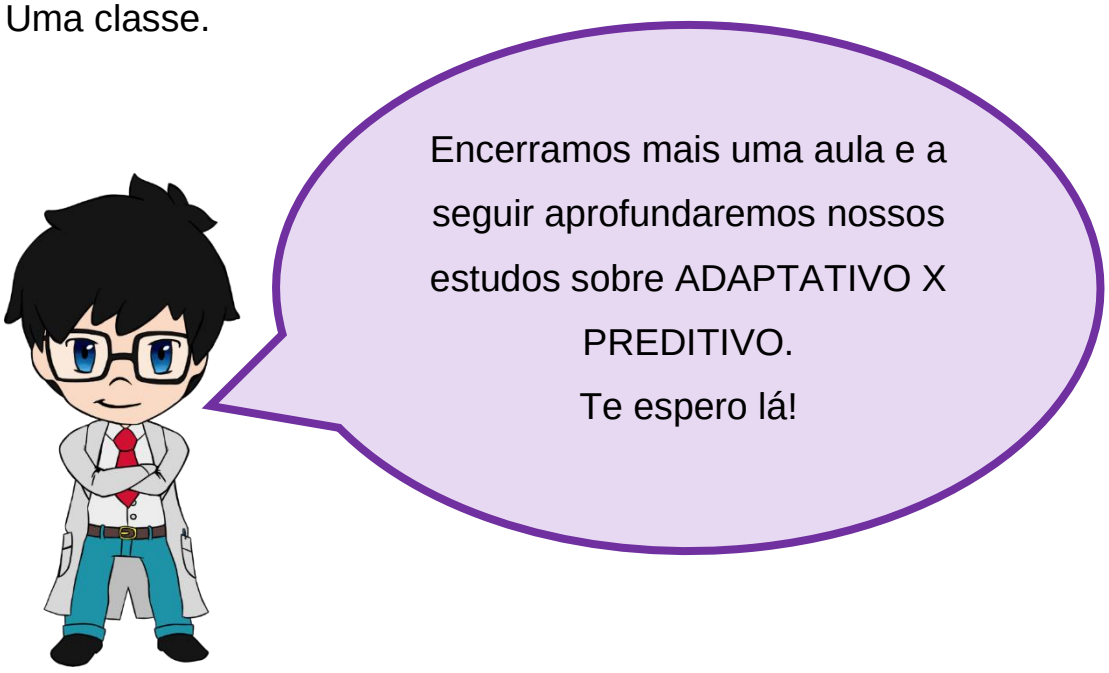
Conclusão

O diagrama de atividades é uma excelente opção para especificação de software, desde que empregado da maneira correta, para os fins adequados.



E ainda, auxilia muito na compreensão do escopo do software, servindo tanto para as disciplinas de análise e de projeto.

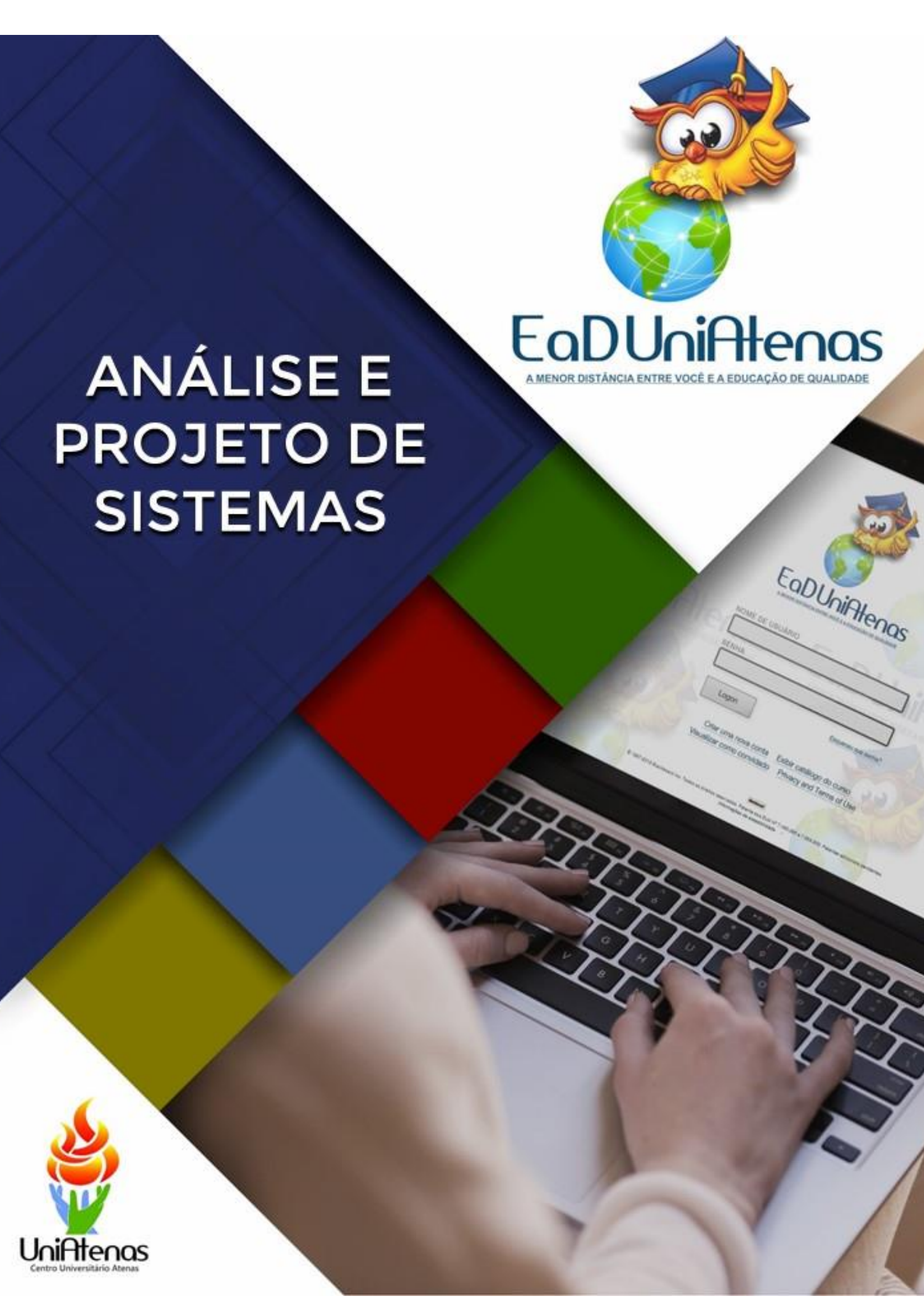
Uma classe.



EaD UniAtenas

A MENOR DISTÂNCIA ENTRE VOCÊ E A EDUCAÇÃO DE QUALIDADE

ANÁLISE E PROJETO DE SISTEMAS



UniAtenas
Centro Universitário Atenas

SISTEMAS ADAPTATIVO X PREDITIVO

OBJETIVO: Analisar as fases das quais o ciclo de vida do projeto se organiza.

INTRODUÇÃO

O ciclo de vida do projeto é o conjunto de fases nas quais este se organiza. Dependendo da organização e sobreposição entre as fases, se pode diferenciar vários modelos de ciclo de vida do projeto que vão desde a abordagem preditiva ou clássica, onde o produto e os entregáveis se definem ao princípio do projeto, passando pelo ciclo de vida iterativo ou incremental, que definem fases que vão incrementando o produto, até ao ciclo de vida adaptativo ou ágil, onde o produto se desenvolve depois de múltiplas iterações e o escopo detalhado para cada iteração se define no princípio da mesma.

CICLO DE VIDA PREDITIVO, CLÁSSICO OU ORIENTADOS AO PLANEJAMENTO

Os ciclos de vida preditivos (também conhecidos como clássicos ou orientados à planificação) são aqueles nos quais o escopo, prazo e custo se determinam o antes possível no ciclo de vida do projeto e os esforços se orientam para cumprir os compromissos estabelecidos em cada um destes fatores.

Normalmente estes projetos se organizam numa série de fases sequenciais ou consecutivas, onde cada uma das fases se enfoca num subproduto ou atividade concreta. Normalmente o trabalho de uma fase é muito diferente ao resto e pelo tanto a equipe de projeto vai variando dependendo das fases.

No início, a direção do projeto se centra em definir o escopo e definir um plano detalhado das atividades necessárias. A partir de aqui a direção do projeto orienta o seu trabalho a cumprir com o planejamento.

Qualquer mudança no escopo do projeto se deve gerenciar de forma explicita e, habitualmente, leva à revisão do planejamento e à aceitação formal do novo plano.

Se opta por ciclos de vida preditivos quando o produto a entregar está bem definido e existe um conhecimento bastante amplio sobre a forma de construir o produto. Este tem sido o modelo de trabalho mais habitual, ainda que nem por isso se ajusta às circunstâncias de todos os projetos e organizações.

CICLO DE VIDA ITERATIVO E INCREMENTAL

Os ciclos de vida iterativos e incrementais são aqueles nos quais se repetem as atividades do projeto em fases ou iterações e em cada uma delas se aumenta o entendimento do produto por parte da equipe do projeto. As iterações desenvolvem o produto através de uma série de ciclos repetidos que vão adicionando sucessivamente funcionalidades ao produto.

No final de cada iteração, se terá completado um entregável ou um conjunto de entregáveis. As futuras iterações podem melhorar os referidos entregáveis ou criar novos. O produto final será a acumulação de funcionalidades construída nas iterações.

Se opta por ciclos de vida iterativos e incrementais quando é necessário gerenciar objetivos pouco definidos ou de uma alta complexidade ou quando a entrega parcial do produto é a chave para o sucesso. Este tipo de ciclo de vida permite à equipe do projeto incorporar a retroalimentação e ir incrementando a experiência da equipe durante o projeto.

CICLO DE VIDA ADAPTATIVO OU ÁGIL

Os ciclos de vida adaptativos, também conhecidos como métodos orientados à mudança ou métodos ágeis, respondem a níveis altos de cambio e a participação continua dos interessados.

Existem dois modelos básicos para este tipo de ciclos de vida, aqueles centrado no fluxo (por exemplo, Kanban) e aqueles centrados em ciclos iterativos e incrementais (por exemplo, Scrum). No primeiro caso se estabelecem limitações muito claras sobre a concorrência entre atividades (Work in Progress) e no último as iterações muito rápidas (entre 1 e 4 semanas) onde se realiza o trabalho (Sprint).

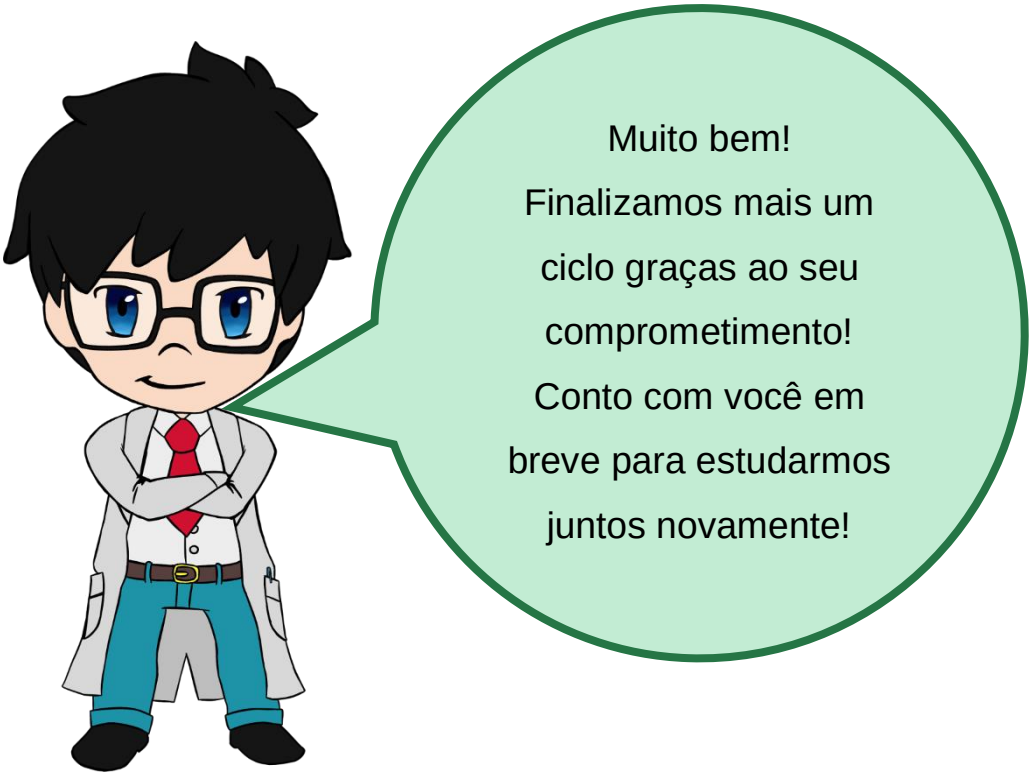
Habitualmente nos modelos ágeis o escopo global do projeto será decomposto num conjunto de requisitos ou trabalho a realizar (em ocasiões denominado Product Backlog). No início de uma iteração a equipe define as funcionalidades que serão abordadas nesse ciclo. No final de cada iteração o produto deve estar pronto para a sua revisão pelo cliente.

Este tipo de ciclo de vida requer equipes muito envolvidas que incluam os patrocinadores ou cliente para proporcionar continua retroalimentação.

Geralmente se opta por métodos ágeis em ambientes que mudam rapidamente, quando o escopo é confuso ou quando a contribuição de valor é muito mutante e com equipes altamente envolvidos.

CLASSE NÃO ESTÁ SUBLINHADA

Sempre uma classe terá um construtor padrão, mesmo não sendo declarado o compilador irá fornecer um. Esse construtor não recebe argumentos e existe para possibilitar a criação de objetos para uma classe.


Muito bem!
Finalizamos mais um ciclo graças ao seu comprometimento!
Conto com você em breve para estudarmos juntos novamente!