Unidade II

3 UML

Quando trabalhamos com programas pequenos e simples, discutimos a sua estrutura diretamente a partir do código, eventualmente com algum texto adicional. Contudo, conforme lidamos com problemas maiores e mais complexos, essa abordagem deixa de ser viável. Além disso, surge a necessidade de comunicar aspectos do programa para pessoas que podem não estar diretamente envolvidas na sua implementação.

Fica evidente que precisamos de alguma forma para ilustrarmos e documentarmos as soluções nas quais trabalhamos, preferencialmente de forma visual. Se mostrarmos o código-fonte do programa, com todos os seus detalhes, poderemos gerar confusão para o usuário (especialmente no caso de grandes programas comerciais). Desse modo, fica claro que precisamos utilizar uma notação (ou uma linguagem) própria para trabalharmos com o projeto de sistemas orientados a objetos.

Uma das notações mais utilizadas é a linguagem UML. Segundo Booch, Rumbaugh e Jacobson (2005), a UML tem quatro objetivos principais: visualizar, especificar, construir e documentar os artefatos de um sistema complexo de software.

Para atingir esses objetivos, a UML é composta por uma série de diagramas, representações gráficas de diversos aspectos da modelagem de software. Esses diagramas podem ser divididos em dois grupos:

- diagramas focados nos aspectos estáticos do modelo;
- diagramas focados nos aspectos dinâmicos do modelo.

Booch, Rumbaugh e Jacobson (2005) listam os diagramas a seguir voltados para os aspectos dinâmicos.

- Diagrama de atividades.
- Diagrama de caso de uso.
- Diagrama de comunicação.
- Diagrama de gráfico de estados.
- Diagrama de sequência.

Ainda segundo Booch, Rumbaugh e Jacobson (2005), os diagramas a seguir são mais focados nos aspectos estáticos.

- Diagrama de artefatos.
- Diagrama de classe.
- Diagrama de componentes.
- Diagrama de estrutura composta.
- Diagrama de implementação.
- Diagrama de objetos.

Neste livro-texto, vamos explorar três desses diagramas, listados a seguir.

- Diagrama de caso de uso.
- Diagrama de classe.
- Diagrama de sequência.

Saiba mais

Para saber mais a respeito de UML, sugerimos a leitura do artigo indicado.

UML (Unified Modeling Language) e Unified Process (Unified Software Development Process). *Devmedia*, [s.d.]. Disponível em: http://www.devmedia.com.br/uml-unified-modeling-language-e-unified-software-development-process-unified-process/11217. Acesso em: 21 jul. 2020.

Além disso, vale a pena consultar o livro a seguir.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML*: guia do usuário. Tradução: Fábio Freitas da Silva e Cristina de Amorim Machado. 2. ed. Rio de Janeiro: Elsevier, 2005.

3.1 Diagrama de caso de uso

Um dos principais problemas com relação ao desenvolvimento de sistemas está em entendermos corretamente o que o cliente espera do sistema. A elaboração dos chamados casos de uso permite que

documentemos o que um sistema deve fazer, sem que nos preocupemos com a forma como isso deve ser feito (BOOCH; RUMBAUGH; JACOBSON, 2005).

No quadro 1, temos um exemplo simplificado de um caso de uso. Em um sistema hipotético, deve ser possível a inserção de relatórios dinâmicos, ou seja, de relatórios cujo conteúdo é parametrizado pelo usuário. O caso de uso apresenta o fluxo de eventos para a introdução desses relatórios no sistema, indicando cada um dos passos da inclusão.

Quadro 1 – Caso de uso: incluir relatório dinâmico

Incluir relatório dinâmico
Ator principal. Usuário gerente
Ator secundário. Nenhum
Resumo. Este caso de uso descreve os procedimentos que devem ser realizados para efetuarmos a inclusão de um relatório dinâmico
Precondição. Verificar se o usuário tem permissão para incluir relatórios dinâmicos
Pós-condição. Nenhuma
Fluxo de eventos
Fluxo principal de eventos
1. O sistema apresenta a opção Gerenciar Relatório.
2. O usuário seleciona a opção Gerenciar Relatório e em seguida a opção Incluir Relatório.
3. O sistema abre a tela de inclusão de relatório com os seguintes campos para a entrada de informações: Nome, Descrição e Conteúdo.
4. O usuário preenche o campo Nome com o nome do relatório.
5. O usuário preenche o campo Descrição com um texto contendo a descrição do relatório.
6. O usuário preenche o campo Conteúdo com um código que representa o conteúdo de um relatório dinâmico.
7. O sistema diz se a instrução cadastrada no campo Conteúdo é válida.
8. O usuário seleciona o botão Incluir.
9. O sistema verifica se todos os campos estão preenchidos.
10. Se todos os campos forem preenchidos, é efetivada a inclusão do relatório e apresentada a mensagem "Relatório incluído com sucesso!".
11. O caso de uso termina.
Fluxo excepcional de eventos
1. O usuário deve ser capaz de cancelar a inserção a qualquer momento selecionando o botão Cancelar.
Fluxo excepcional de eventos
1. Se o campo Conteúdo tiver um código inválido, o sistema deve exibir uma mensagem de erro dizendo ao usuário que ele deve preencher esse campo corretamente.
Fluxo excepcional de eventos
1. Se algum campo não estiver preenchido, o sistema deve apresentar uma mensagem para o usuário solicitando o preenchimento do respectivo campo.

A especificação de um sistema real é composta por vários casos de uso. A execução de um caso de uso pode requerer precondições (para que seja possível seguir o fluxo de eventos) e pós-condições.

No exemplo, é necessário verificar, antes da execução do caso de uso, se o usuário tem permissão para inserir relatórios dinâmicos no sistema.

Além do fluxo principal de eventos, que mostra o caminho normal de utilização, devemos também escrever os fluxos excepcionais de eventos, que representam outras possibilidades de interação. Por exemplo, suponha uma situação na qual um usuário está fazendo um cadastro em uma loja virtual e precisa entrar com um número de CPF. Imagine que o sistema deva validar esse número.

Se o número de CPF for válido, o sistema deve prosseguir normalmente com o processo de cadastro, sendo esse o fluxo principal de eventos. Se o CPF digitado não for válido, o sistema deve apresentar uma mensagem de erro para o usuário, solicitar que ele entre novamente com um número de CPF e impedir a continuidade do cadastro. Esse seria um fluxo excepcional de eventos, que ocorre devido à tentativa de cadastro com um número de CPF inválido.

É bastante provável que um sistema complexo seja composto por um grande número de casos de uso diferentes. É importante termos uma visão de como esses casos de uso se relacionam e de como os diversos atores estão envolvidos nesses casos de uso.

Na figura 23, temos um exemplo, bastante simplificado, de um diagrama de caso de uso. Nessa figura, existe um ator (o Gerente) se comunicando com três casos de uso: Emitir Relatório, Incluir Relatório Dinâmico e Excluir Relatório.

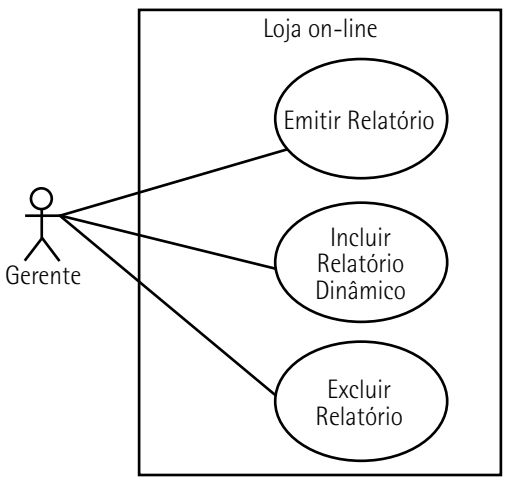

Figura 23 – Exemplo de diagrama de caso de uso

É interessante observarmos que o diagrama de caso e o caso de uso nos dão informações diferentes e complementares. Enquanto o caso de uso é uma descrição que nos ajuda a compreender uma funcionalidade em um cenário específico, o diagrama de caso de uso nos permite ver a inter-relação entre os diversos casos de uso e atores que atuam no sistema.

3.2 Diagrama de classe

Vimos que o conceito de classe é fundamental para a orientação a objetos. Em seu livro clássico sobre UML, Booch, Rumbaugh e Jacobson (2005) definem classe como uma descrição de um conjunto de objetos que comungam de mesmos atributos, mesmas operações, mesmo relacionamento e mesma semântica.

Na UML, os diagramas de classe permitem que representemos classes individuais por meio de retângulos. Além disso, esses diagramas apresentam uma notação própria para mostrarmos como essas classes se relacionam. A natureza desse tipo de diagrama faz com que ele capture mais os aspectos estáticos do que os aspectos dinâmicos de um sistema.

Na figura 24, temos um exemplo de representação de uma única classe. Nessa figura, o retângulo que representa a classe é dividido em três partes:

- na parte superior, temos o nome da classe (no caso, Endereço);
- nas duas partes inferiores, temos os atributos da classe – rua, número, bairro, complemento, cidade, estado e cep – seguidos dos seus diversos métodos – verificaDescontoEntrega() e calculaCustoEntrega().

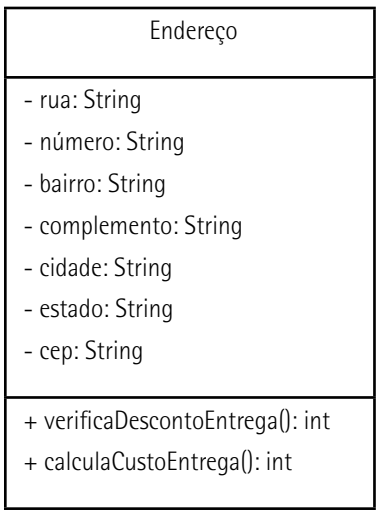

Figura 24 – Exemplo de representação de um endereço

Podemos interpretar que os atributos funcionam como as características de um objeto – no caso, um objeto do tipo Endereço. Assim, dado objeto dessa classe deve ter, a ele associados, uma rua, um número, um bairro, um complemento, uma cidade, um estado e um cep. Observamos que o diagrama informa o tipo de cada um desses atributos – no caso, todos são cadeias de texto (strings).

Os métodos estão relacionados ao comportamento exposto por uma classe e correspondem à principal forma como os objetos de determinada classe devem trocar informações com o mundo externo. De maneira bastante simplificada, os métodos são similares às funções ou aos procedimentos da análise estruturada, mas estão ligados a uma classe e atuam especialmente nos seus atributos.

Outro ponto que deve ser observado é o seguinte: a representação da UML permite que se informe, também, a visibilidade de métodos e atributos. No item em que abordamos o encapsulamento, mostramos que a questão da visibilidade é muito importante na orientação a objetos e, em virtude da sua importância, esse conceito foi diretamente introduzido na sua notação.

Os quatro níveis de visibilidade são similares aos vistos anteriormente (BOOCH; RUMBAUGH; JACOBSON, 2005), conforme exposto a seguir.

- **Público**. O método (ou o atributo) pode ser visto por qualquer outra classe, não importa se é uma subclasse da hierarquia ou qualquer outra. Utilizamos o símbolo + para a sua representação. Por exemplo, + calculaCustoEntrega(): int significa que o método calculaCustoEntrega() tem visibilidade pública e retorna um número do tipo int.

- **Protegido**. O método (ou o atributo) pode ser visto apenas por subclasses. É representado pelo símbolo #. Por exemplo, um atributo protegido poderia ser declarado como # sobrenome: String.

- **Privado**. A visibilidade limita-se à própria classe. É representada pelo símbolo -. Por exemplo, - nome: String significa que nome é um atributo privado da classe e é do tipo String.

- **Pacote**. A visibilidade limita-se ao pacote. É representada pelo símbolo ~.

A visibilidade em um diagrama de classes é representada por um símbolo adicionado antes do nome do método ou do atributo, dentro do retângulo que indica a classe.

Qualquer projeto real vai ter, muito provavelmente, bem mais do que apenas uma única classe. O diagrama de classes deve registrar, além das classes, o relacionamento entre as diversas classes e o relacionamento entre as classes e as interfaces (ou entre outras interfaces).

Existem diversos tipos possíveis de relacionamento entre as classes (FOWLER, 2005; PILONE; PITMAN, 2005): dependência, associação, agregação, composição e generalização. Além disso, há as chamadas classes de associação.

Na figura 25, temos um exemplo de um diagrama de classes com alguns relacionamentos.

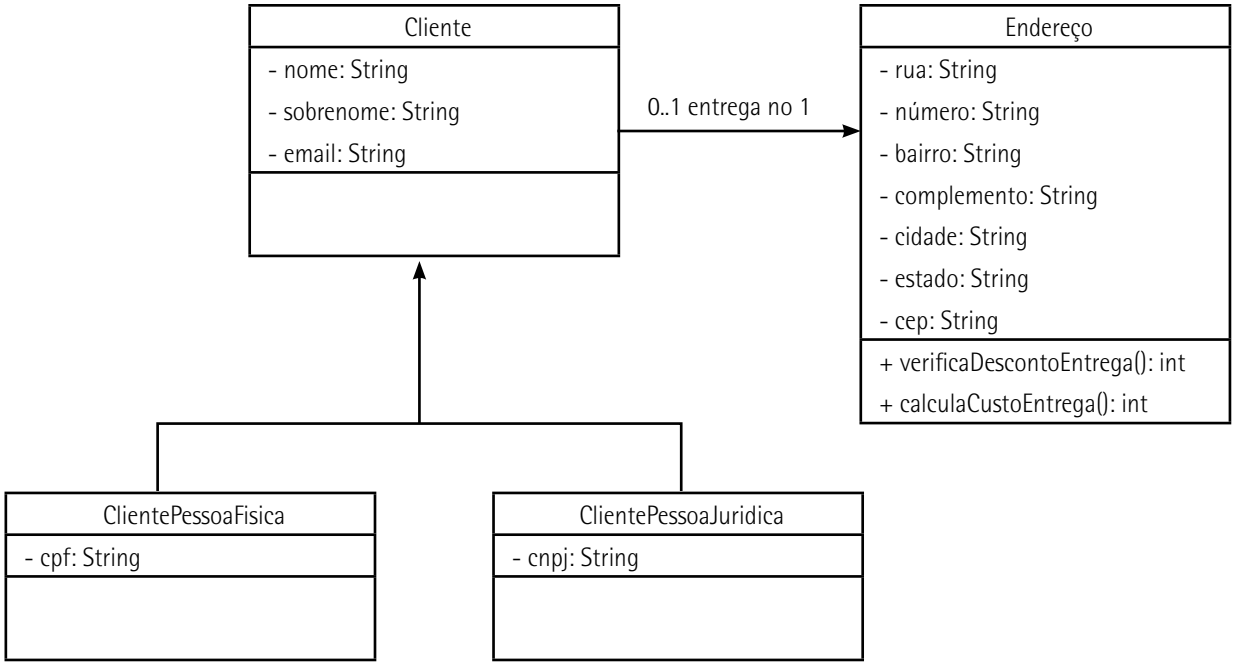

Figura 25 – Exemplo de um diagrama de classes com alguns relacionamentos

3.3 Diagrama de sequência

O diagrama de sequência é um tipo de diagrama de interação, o que faz dele um dos diagramas da UML cuja preocupação está atrelada aos aspectos dinâmicos do sistema (BOOCH; RUMBAUGH; JACOBSON, 2005).

Nesse diagrama, estamos preocupados em mostrar como os objetos do sistema trocam mensagens entre si ao longo do tempo. Isso contrasta com os objetivos de certos diagramas, como o de classes, em dois aspectos.

- Em primeiro lugar, estamos tratando da troca de mensagens entre objetos, enquanto o diagrama de classes centra-se no relacionamento entre classes.

- Em segundo lugar, o diagrama incorpora caráter temporal, algo que não é encontrado no diagrama de classes.

Na figura 26, temos um exemplo de diagrama de sequência em que um objeto da classe Cliente envia uma mensagem (ou seja, executa um método) para um objeto da classe Endereço. Nessa figura, os objetos são representados como retângulos dispostos horizontalmente. Uma linha tracejada, chamada de linha de vida, indica o tempo de vida dos objetos (PILONE; PITMAN, 2005). Dessa forma, o eixo vertical do diagrama refere-se à passagem de tempo, enquanto o eixo horizontal refere-se aos objetos envolvidos na comunicação (BOOCH; RUMBAUGH; JACOBSON, 2005). O retângulo branco que aparece sobre a linha tracejada é chamado de foco de controle (ou ocorrência de execução) e é utilizado para representar o fato de que o objeto em questão está executando uma ação (PILONE; PITMAN, 2005). No

exemplo em estudo, o objeto da classe Endereço calcula o custo da entrega e retorna esse valor para o objeto da classe Cliente.

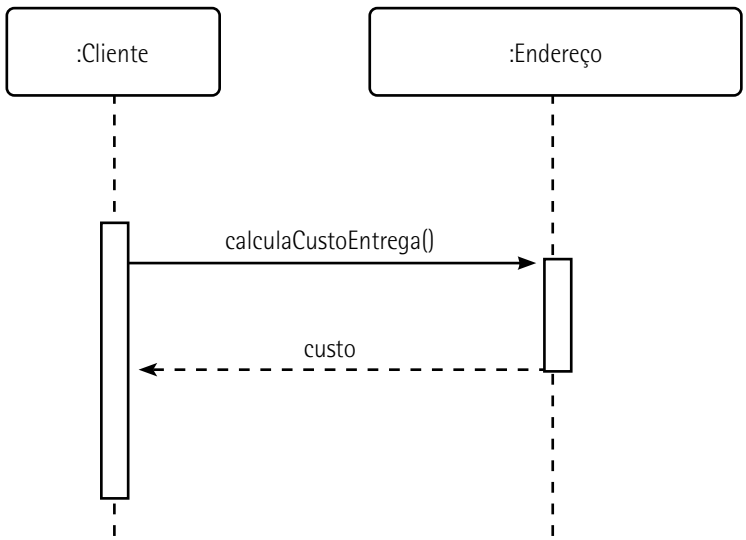

Figura 26 – Exemplo de um diagrama de sequência com a funcionalidade calcular custo de entrega

4 LINGUAGEM DE PROGRAMAÇÃO C#

A linguagem de programação C# é uma linguagem orientada a objetos desenvolvida pela Microsoft tendo em mente o ambiente de desenvolvimento .NET (TROELSEN; JAPIKSE, 2017). Foi inspirada em outras linguagens bastante populares, como Java e C++, mas a sua evolução e o fato de ela estar atrelada ao ambiente .NET deram-lhe um caráter peculiar.

Ainda que um dos seus principais usos seja para o desenvolvimento de aplicações web, a linguagem C# pode ser utilizada em diferentes contextos, como o desenvolvimento de aplicações Windows desktop, de jogos e de aplicações para celulares.

No próximo item, veremos como é possível fazer o download e a instalação do Microsoft Visual Studio para o desenvolvimento de aplicações utilizando C#.

4.1 Ambientes de desenvolvimento

Para facilitar e agilizar o desenvolvimento de aplicações, uma das principais ferramentas utilizadas por desenvolvedores são as IDEs (integrated development environments), ou ambientes de desenvolvimento integrado, em português. Esses ambientes correspondem a um conjunto de ferramentas que simplificam o processo de desenvolvimento de software, uma vez que permitem que o programador utilize uma interface coerente e integrada entre os diversos programas.

Um exemplo de IDE é o Microsoft Visual Studio. Desenvolvido pela Microsoft, essa IDE focaliza especialmente o ambiente .NET e pode ser utilizada com diferentes linguagens de programação, como as linguagens Visual Basic .NET, C# e F#. Além de diversas funcionalidades presentes em sua versão

padrão, é possível adicionarmos novas funcionalidades com o emprego de plug-ins. Um exemplo de uso bastante popular desse sistema é o desenvolvimento web na plataforma ASP.NET, na qual frequentemente utilizamos as linguagens Visual Basic .NET e C#.

Até agora, trabalhamos apenas com o Microsoft Visual Studio Code. Nas próximas seções, vamos utilizar o Microsoft Visual Studio 2019. No próximo item, veremos como é possível fazer a instalação do Microsoft Visual Studio 2019.

4.1.1 Microsoft Visual Studio 2019

Uma vez que o Microsoft Visual Studio foi desenvolvido pela Microsoft e é focado nos seus produtos e serviços, ele encontra-se disponível apenas na plataforma Windows. Vale notar que existem produtos similares que trabalham em outras plataformas, mas eles não fazem parte do escopo que estamos abordando agora.

Observação

O site oficial para download do Microsoft Visual Studio é: https://visualstudio.microsoft.com/pt-br/vs/. Não se esqueça de verificar a lista de requisitos de instalação, disponível em: https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements. É interessante checar a versão do sistema operacional e os requisitos mínimos de hardware.

Existem três versões disponíveis para a instalação. O nome do arquivo do instalador varia de acordo com a edição selecionada:

- vs_community.exe
- vs_professional.exe
- vs_enterprise.exe

Observação

É interessante vermos as diferenças entre as diversas opções de download no site. Observe que apenas a versão Community é gratuita. Mesmo sendo a mais limitada, ela apresenta muitas funcionalidades diferentes e é útil para diversos tipos de projeto.

Para iniciarmos a instalação, basta selecionarmos (clicando duas vezes) o binário que foi baixado para a máquina. O instalador do Visual Studio 2019 usa a mesma lógica de design que seu antecessor e permite a instalação de conjuntos de componentes, cada um direcionado para um cenário de

desenvolvimento específico. Cada conjunto de componentes é chamado de carga de trabalho. Observamos um exemplo de tela de seleção de cargas de trabalho na figura 27.

As cargas de trabalho facilitam a instalação e a manutenção do programa e permitem que os desenvolvedores instalem apenas o que realmente precisam, sem componentes desnecessários, o que ajuda a economizar muito espaço no disco.

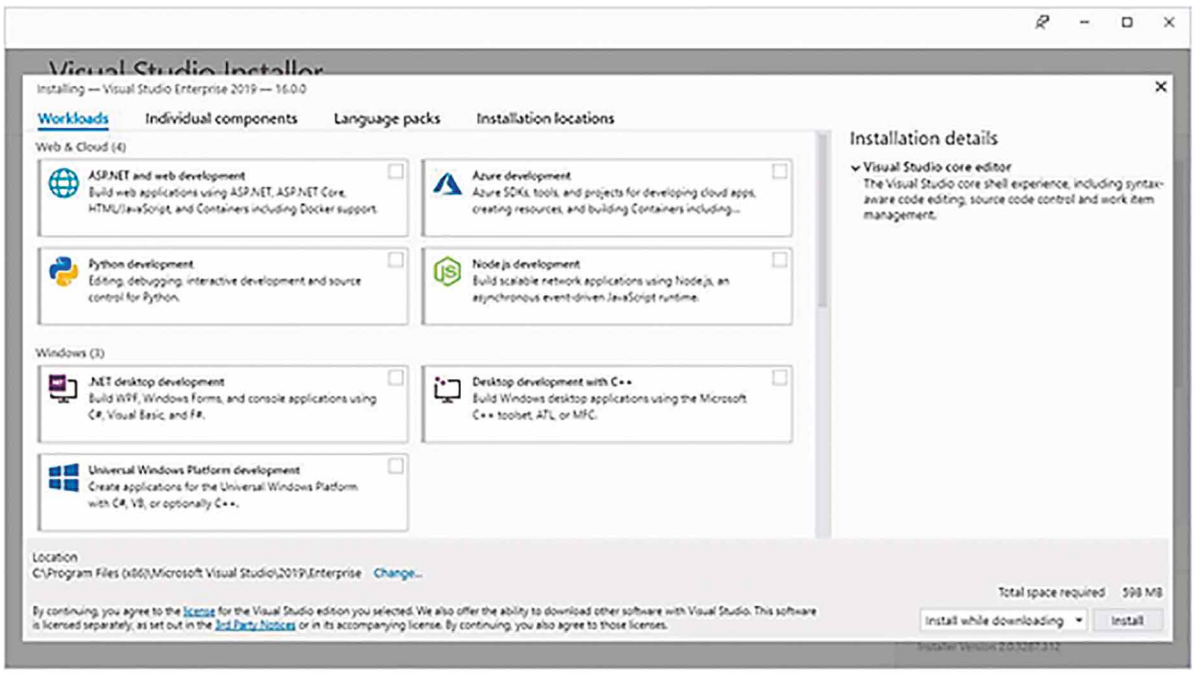

Figura 27 – Seleção de itens de instalação no Visual Studio 2019

Para os fins instrucionais deste livro-texto, são necessárias as cargas de trabalho mencionadas a seguir.

- Desenvolvimento de desktop .NET.
- Desenvolvimento da Plataforma Universal do Windows.

Observação

O leitor não é obrigado a fazer a mesma carga aqui apresentada. Ele pode selecionar apenas aquelas de que precisa. Posteriormente, o usuário poderá instalar cargas de trabalho adicionais, conforme elas forem necessárias.

Resumo

Iniciamos a unidade II estudando alguns aspectos da UML, especialmente os que se referem à escrita de casos de uso e à utilização dos diagramas de caso de uso em um projeto. Vimos que os casos de uso permitem a descrição de cenários de emprego de um sistema sem que nos preocupemos com itens da sua construção. Isso fornece ao analista e ao usuário uma forma de se comunicar e de focalizar os problemas sem considerar, imediatamente, uma solução tecnológica específica. Verificamos que existe um diagrama de casos de uso que permite investigar, de modo visual, como os diversos casos de uso de um sistema se relacionam.

Constatamos que a UML apresenta tanto diagramas focados em capturar aspectos dinâmicos do sistema quanto diagramas focados em aspectos estáticos.

Um dos diagramas com foco em aspectos estáticos é o diagrama de classes. Ele visa a mostrar como as diversas classes de um sistema se relacionam e inclui diversos tipos de relacionamento justamente para esse fim.

O diagrama de sequência focaliza aspectos dinâmicos do sistema e pretende indicar como os objetos se comunicam ao longo do tempo. Esse diagrama tem a capacidade de mostrar o tempo de vida de um objeto no sistema. Ele pode acompanhar um objeto desde a sua criação até a sua destruição ou centrar-se apenas em um período da vida de um objeto, normalmente ilustrando alguma funcionalidade específica.

Retomamos o tema dos ambientes de desenvolvimento integrados, ou IDEs, coleções de programas que auxiliam no desenvolvimento de aplicações. Por disponibilizarem ferramentas como depuradores e editores de texto especializados em códigos já integrados, esses ambientes tornam o trabalho do programador mais rápido e produtivo. Falamos sobre uma IDE em particular, o Microsoft Visual Studio 2019, e apontamos como realizar a sua instalação, a fim de que seja possível a elaboração dos próximos exemplos contidos neste livro-texto.

Finalmente, iniciamos o estudo da linguagem C#, para que possamos aprender como criar uma aplicação para a plataforma desktop. Especificamente, veremos como fazer uma aplicação Windows Forms no Microsoft Visual Studio 2019.

Exercícios

Questão 1. A representação da UML possibilita que informemos a visibilidade de métodos e atributos. Nesse contexto, considere, no quadro a seguir, os quatro níveis de visibilidade (indicados por nível I, nível II, nível III e nível IV) e suas características.

Quadro – Quatro níveis de visibilidade e suas características

Nível de visibilidade	Característica
Nível I	O método (ou o atributo) pode ser visto por qualquer outra classe. Utilizamos o símbolo "+" para sua representação.
Nível II	O método (ou o atributo) pode ser visto apenas por subclasses. Utilizamos o símbolo "#" para sua representação.
Nível III	A visibilidade limita-se à própria classe. Utilizamos o símbolo "-" para sua representação.
Nível IV	A visibilidade limita-se ao pacote. Utilizamos o símbolo "~" para sua representação.

As indicações nível I, nível II, nível III e nível IV presentes no quadro referem-se, respectivamente, aos termos:

A) Público, Privado, Protegido e Pacote.

B) Protegido, Público, Protegido e Pacote.

C) Público, Protegido, Pacote e Privado.

D) Privado, Protegido, Público e Pacote.

E) Público, Protegido, Privado e Pacote.

Resposta correta: alternativa E.

Análise da questão

As indicações nível I, nível II, nível III e nível IV estão especificadas no quadro a seguir.

Quadro – Quatro níveis de visibilidade e suas características (com especificações)

Nível de visibilidade	Característica
Público	O método (ou o atributo) pode ser visto por qualquer outra classe. Utilizamos o símbolo "+" para sua representação.
Protegido	O método (ou o atributo) pode ser visto apenas por subclasses. Utilizamos o símbolo "#" para sua representação.
Privado	A visibilidade limita-se à própria classe. Utilizamos o símbolo "-" para sua representação.
Pacote	A visibilidade limita-se ao pacote. Utilizamos o símbolo "~" para sua representação.

Questão 2. A UML visa a visualizar, especificar, construir e documentar os artefatos de um sistema de software. No sentido de alcançar tais metas, a UML é composta de uma série de diagramas, representações gráficas de diversos aspectos da modelagem de software. Em relação a esses diagramas na UML, considere as afirmativas a seguir.

I – O diagrama de caso de uso, equivalente ao próprio caso de uso, refere-se a uma descrição que auxilia a compreensão de uma funcionalidade em um cenário específico e permite a visualização da inter-relação entre os vários atores que atuam em um sistema.

II – O diagrama de classes possibilita que representemos classes individuais por meio de retângulos: em virtude da sua natureza, esse tipo de diagrama captura mais os aspectos estáticos do que os aspectos dinâmicos de um sistema.

III – O diagrama de sequência é um tipo de diagrama de interação: seu foco está associado aos aspectos estáticos de um sistema e à apresentação de como os objetos do sistema trocam mensagens entre si ao longo do tempo.

É correto o que se afirma em:

A) I, apenas.

B) II, apenas.

C) III, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa B.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: o diagrama de caso de uso e o caso de uso fornecem informações distintas e complementares. O caso de uso é uma descrição que ajuda a compreensão de uma funcionalidade em um cenário específico. O diagrama de caso de uso permite que vejamos a inter-relação entre os diversos casos de uso e atores que atuam em um sistema.

II – Afirmativa correta.

Justificativa: na UML, os diagramas de classes possibilitam que representemos classes individuais por meio de retângulos. Adicionalmente, esses diagramas apresentam uma notação própria para mostrar como as classes se relacionam. A natureza desse tipo de diagrama faz com que ele capture mais os aspectos estáticos do que os aspectos dinâmicos de um sistema.

III – Afirmativa incorreta.

Justificativa: o diagrama de sequência é um tipo de diagrama de interação. Trata-se de diagrama da UML centrado nos aspectos dinâmicos de um sistema. Nesse tipo de diagrama, o foco é mostrar como os objetos do sistema trocam mensagens entre si ao longo do tempo.