Questões 16 e 17

Questão 16.[16]

UML é uma linguagem padrão para desenvolver e documentar projetos de software e permite que desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados. Ela surgiu como uma proposta de ser uma linguagem para modelagem de dados que usava diversos artefatos para representar o modelo de negócio e um desses artefatos é o diagrama de classes.

PRESSMAN, R.S. *Engenharia de software*. 6. ed. Porto Alegre: Bookman, 2006 (com adaptações).

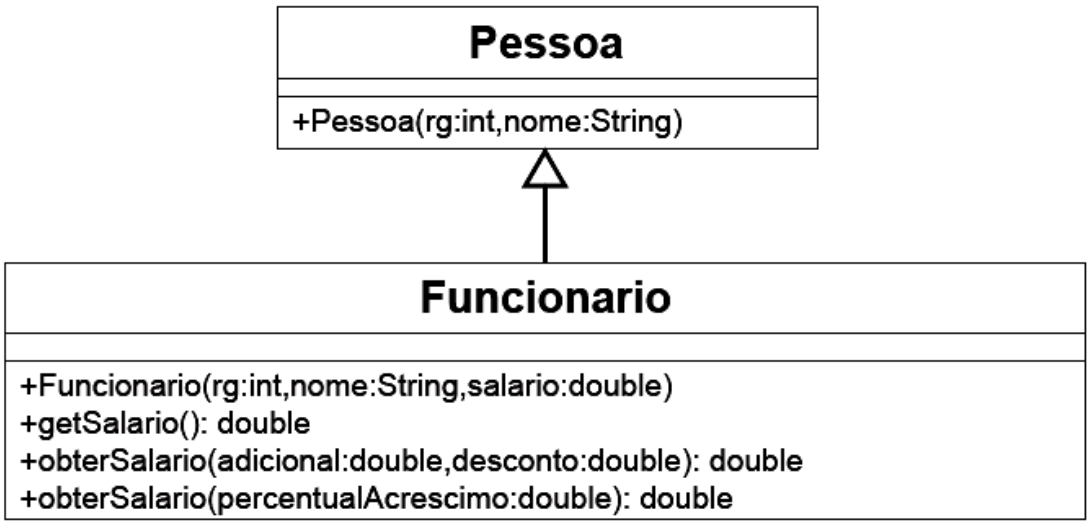
Se um projeto não tem a documentação apropriada ou se está com a documentação desatualizada, uma opção é a engenharia reversa que possibilita mapear códigos para diagramas UML. A seguir, é apresentado um código na linguagem de programação JAVA.

```
1 package default;
2
3 public class Funcionario extends Pessoa {
4    private double salario;
5
6    public Funcionario(int rg, String nome,
7                       double salario) {
8         super(rg, nome);
9         this.salario = salario;
10   }
11   public double getSalario() {
12        return salario;
13
14   }
15   public double obterSalario(double
16        percentualAcrescimo) {
17        double salarioReajustado = salario +
18             salario * percentualAcrescimo / 100;
19        return salarioReajustado;
20
21   }
22   public double obterSalario(double
23        adicional, double desconto){
24        return this.getSalario() + adicional - desconto;
25
26   }
27 }
```

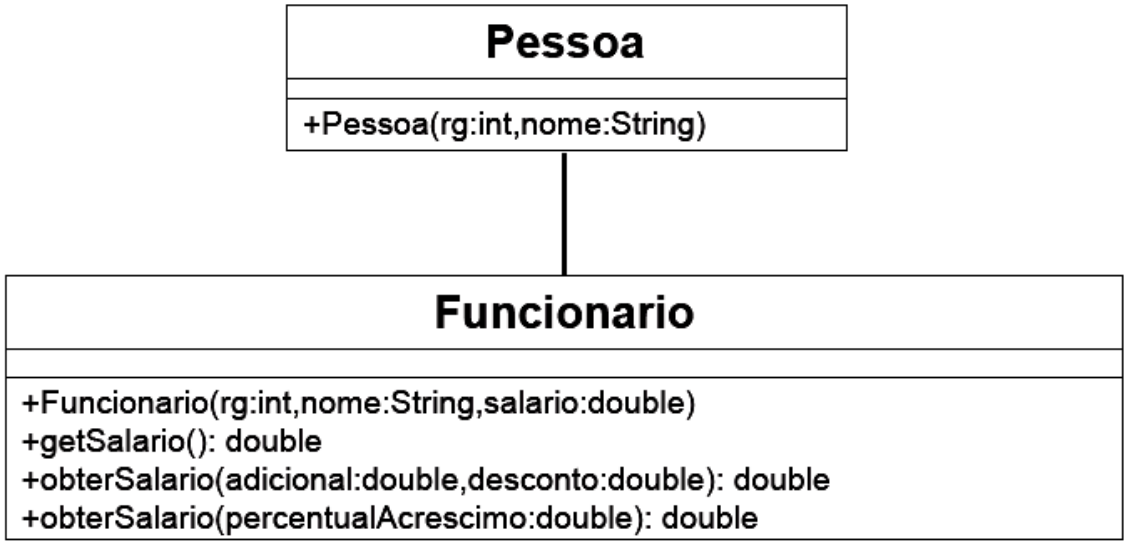
Utilizando a engenharia reversa nesse trecho de código, o diagrama UML de classes correspondente é

[16]Questão 29 – Enade 2014.

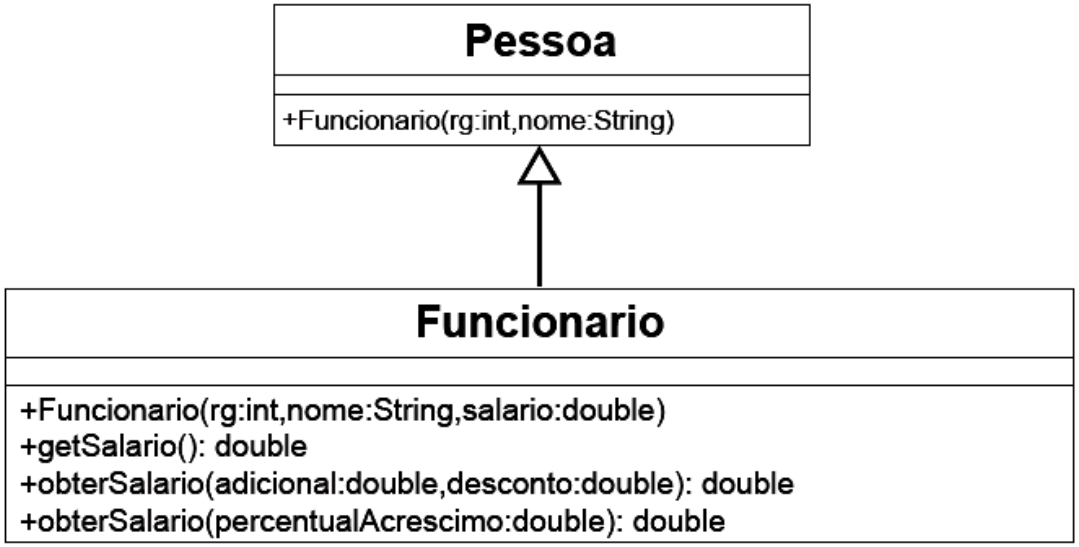
A.


Pessoa
+Pessoa(rg:int,nome:String)
Funcionario
+Funcionario(rg:int,nome:String,salario:double)
+getSalario(): double
+obterSalario(adicional:double,desconto:double): double
+obterSalario(percentualAcrescimo:double): double

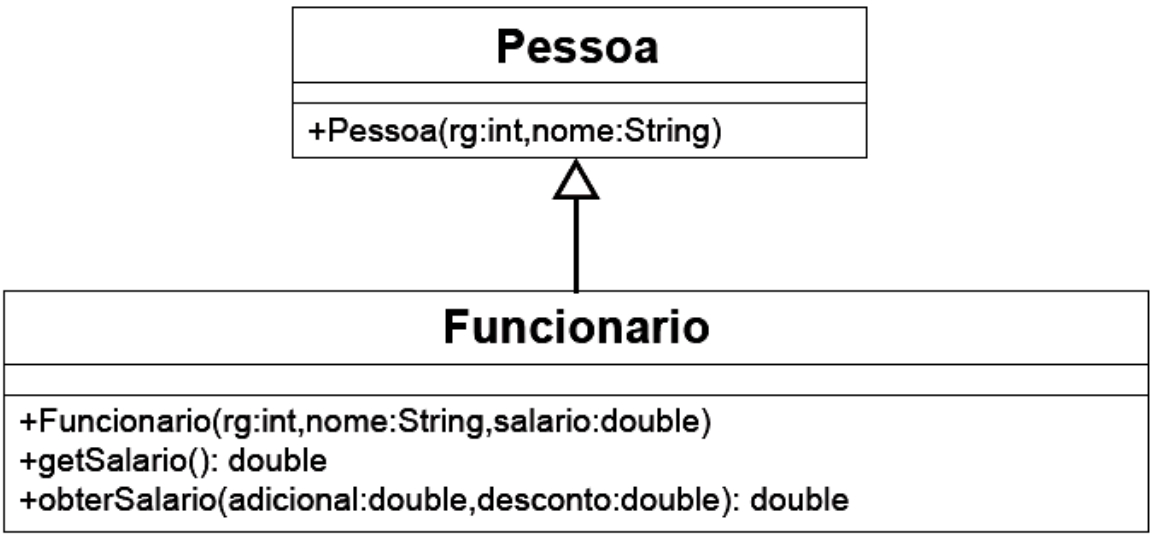
B.


Pessoa
+Pessoa(rg:int,nome:String)
Funcionario
+Funcionario(rg:int,nome:String,salario:double)
+getSalario(): double
+obterSalario(adicional:double,desconto:double): double
+obterSalario(percentualAcrescimo:double): double

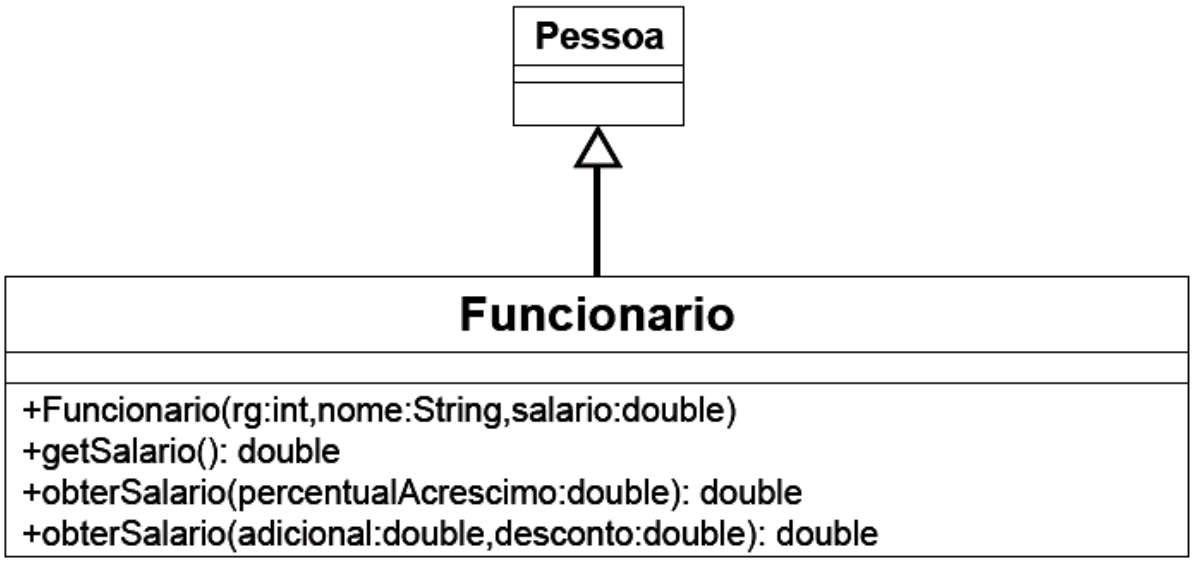
C.


Pessoa
+Funcionario(rg:int,nome:String)
Funcionario
+Funcionario(rg:int,nome:String,salario:double)
+getSalario(): double
+obterSalario(adicional:double,desconto:double): double
+obterSalario(percentualAcrescimo:double): double

D.



E.



Questão 17.[17]

Casos de uso podem ser organizados agrupando-os em pacotes do mesmo modo como são organizadas as classes. Também podem ser organizados pela especificação de relacionamentos de generalização, inclusão e extensão, existentes entre eles.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *UML - Guia do Usuário*. Campus, 2006 (com adaptações).

Considerando os relacionamentos existentes entre os casos de uso, avalie as afirmativas a seguir.

I. Para casos de uso, a generalização significa que o caso de uso filho herda o comportamento e o significado do caso de uso pai e no caso de uso filho deverá acrescentar ou sobrescrever o comportamento de seu pai.

[17]Questão 17 – Enade 2014.

II. Um relacionamento de inclusão entre casos de uso significa que o caso de uso base incorpora explicitamente o comportamento de outro caso de uso em uma localização especificada. O caso de uso base poderá permanecer isolado, mas, sob certas condições, seu comportamento poderá ser incluído pelo comportamento de outro caso de uso.

III. Um relacionamento estendido entre casos de uso significa que o caso de uso base incorpora implicitamente o comportamento de outro caso de uso em um local especificado indiretamente pelo caso de uso estendido. O caso de uso estendido nunca permanece isolado, mas é apenas instanciado como parte de alguma base maior que o estende.

IV. Um relacionamento estendido é utilizado para a modelagem de parte de um caso de uso que o usuário poderá considerar como um comportamento opcional do sistema e para a modelagem de um subfluxo separado, que é executado somente sob determinadas condições.

É correto apenas o que se afirma em

A. I e II.
B. I e IV.
C. II e III.
D. I, III e IV.
E. II, III e IV.

1. Introdução teórica

1.1. UML (Unified Modeling Language)

De acordo com Booch, Rumbaugh e Jacobson (2006), a UML (Unified Modeling Language) é definida como "uma linguagem-padrão para a elaboração da estrutura de projetos de software".

Os autores destacam os quatro objetivos básicos da UML, quanto aos artefatos de um sistema complexo de software: visualizar, especificar, construir e documentar.

O principal objetivo da UML é estabelecer um modelo ou representar conceitual e fisicamente um sistema (BOOCH, RUMBAUGH e JACOBSON; 2006).

1.1.1. UML para a visualização

Um projeto de software envolve mais do que o código-fonte. Além disso, um mesmo problema pode ser resolvido de diferentes formas, com códigos muito distintos. Ainda que programas com códigos diferentes possam atender aos mesmos requisitos, nem todas as soluções são igualmente úteis ou interessantes.

Se dispusermos apenas do código-fonte de um programa, sem nenhum tipo de documentação, o entendimento da solução fica muito mais difícil. Programas comerciais podem ser muito longos e conter milhões de linhas de código-fonte. Dessa forma, é importante ter uma maneira de visualizar o todo, a estrutura geral de um software, sem que, necessariamente, haja a obrigação de se ler o código-fonte de modo completo.

Desenhos e diagramas informais podem ajudar nessa visualização, mas a UML fornece uma linguagem-padrão que permite que diferentes pessoas e empresas façam e interpretem diagramas e modelos de uma mesma forma.

1.1.2. UML para especificação

Antes de escrevermos o código para um programa, é preciso conhecer quais são as necessidades do cliente. Além disso, grandes projetos envolvem problemas complexos de software.

Muitas empresas contratam profissionais especializados para encontrar a melhor solução, como os arquitetos de software. Esses profissionais precisam de uma linguagem para especificar, construir e documentar a solução e a UML oferece uma linguagem especificamente desenvolvida para esse propósito.

1.1.3. UML para construção

Ainda que a UML não seja uma linguagem de programação propriamente dita, é possível estabelecer um mapeamento entre os modelos construídos pela UML e o código-fonte em diversas linguagens de programação, como Java ou C# (BOOCH, RUMBAUGH e JACOBSON, 2006).

Essa facilidade do mapeamento vem do alto grau de expressividade aliado à baixa ambiguidade, o que não é o caso quando consideramos outras formas de especificação,

como um texto puro. Essa precisão faz com que existam ferramentas capazes de converter alguns diagramas UML em código-fonte com pouca ou nenhuma intervenção do usuário.

1.1.4. UML para documentação

De acordo com Booch, Rumbaugh e Jacobson (2006), um projeto de software envolve mais do que apenas o código-fonte. Empresas de software costumam produzir diversos outros elementos, chamados de artefatos, como:

- requisitos do software;
- documentações da arquitetura;
- projetos e planos de projeto;
- código-fonte;
- testes e planos de teste;
- protótipos.

A UML é utilizada para documentar diversos desses artefatos, de modo formal e com baixa ambiguidade.

1.2. Diagrama de classes

Formalmente, define-se uma classe como "uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica" (BOOCH, RUMBAUGH e JACOBSON; 2006).

É importante diferenciar as classes dos objetos: uma classe define um conjunto de objetos, mas não é um objeto. Por exemplo, podemos definir a classe das "xícaras de café" como sendo objetos cerâmicos, ocos, que servem para armazenar uma quantidade pequena de líquido quente. Essa classe abrange todas as possíveis xícaras, de diferentes formas e tamanhos. Se tomarmos um exemplo específico, como uma xícara de restaurante, dizemos que essa xícara é um objeto, ou uma instância específica de uma classe. Utilizamos uma classe para definir uma categoria de objetos similares.

A UML apresenta um diagrama específico para a representação das classes, chamado de diagrama de classes. Nesse diagrama, as classes são representadas por retângulos contendo nomes, atributos e operações, como ilustrado na figura 1. Os atributos representam propriedades associadas a cada um dos objetos que serão instanciados pela

classe. Por exemplo, a classe "Cliente" pode ter um atributo chamado de "nome". Outros atributos que essas classes podem ter são: "endereço", "telefone" e "email".

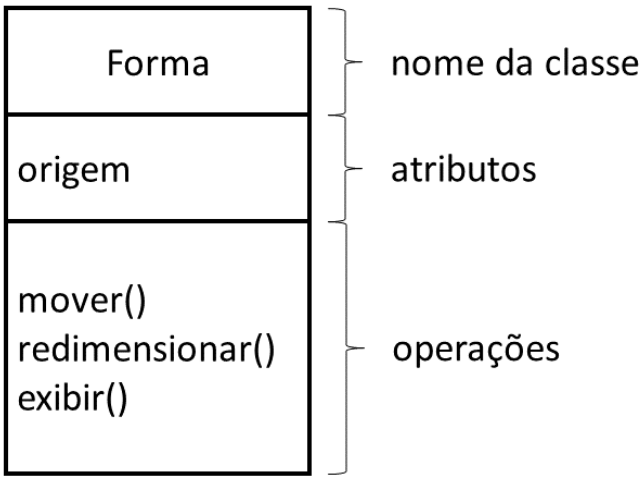


Figura 1. Representação de uma classe na UML.
Fonte. BOOCH, RUMBAUGH e JACOBSON, 2006 (com adaptações).

Além de atributos, uma classe também apresenta comportamentos associados. Por exemplo, suponha um editor de imagens que tenha uma funcionalidade de desenho de formas geométricas, como losangos. Essa funcionalidade pode ser implementada por meio de uma classe, chamada de "Losango", que apresenta uma série de comportamentos associados, como a movimentação do losango e a alteração do seu tamanho, entre outras. Essas operações são representadas no retângulo da classe, logo abaixo dos atributos.

As operações têm um nome e costumam terminar com parênteses "()": elas são similares a "funções" e podem receber parâmetros. Os parâmetros são representados nos parênteses e separados por vírgulas (no caso da existência de mais de um parâmetro).

As diversas classes que compõem um programa podem estar relacionadas entre si e existem, essencialmente, três tipos de relacionamentos entre classes:

- dependência;
- generalização;
- associação.

Utilizamos uma relação de dependência quando queremos dizer que uma classe necessita de outra classe para o seu funcionamento. Por exemplo, suponhamos uma classe chamada "Cachorro" e uma classe chamada "Animal". Podemos dizer que "Animal" representa uma generalização de "Cachorro", uma vez que um cachorro é um animal, mas nem todo animal é um cachorro. Finalmente, uma associação é um "relacionamento semântico entre dois elementos do modelo" (PENDER, 2004). Uma associação estabelece o

motivo pelo qual duas classes relacionam-se e estabelece as regras que controlam esse relacionamento.

1.3. Casos de uso

Segundo Booch, Rumbaugh e Jacobson (2006), um caso de uso

> *especifica o comportamento de um sistema ou de parte de um sistema e é uma descrição de um conjunto de sequências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator.*

Desenvolvedores de software precisam compreender as necessidades dos usuários e comunicá-las, sem precisar entrar em detalhes da implementação. A elaboração de casos de uso facilita esse processo de comunicação entre os diversos envolvidos em um projeto de software.

2. Análise das alternativas e das afirmativas

Questão 16.

A - Alternativa correta.
JUSTIFICATIVA. O trecho de diagrama de classes apresentado está correto. Nele, a classe Pessoa é uma generalização da classe Funcionário.

B - Alternativa incorreta.
JUSTIFICATIVA. O diagrama está incorreto, pois as classes Pessoa e Funcionário estão representadas com relacionamento de associação. Na realidade, essas duas classes deveriam apresentar relacionamento de generalização.

C - Alternativa incorreta.
JUSTIFICATIVA. O construtor da classe Pessoa está representado de forma errada. Deveria ser: +Pessoa(rg:int, nome:String), e não +Funcionário(re:int, nome:String).

D - Alternativa incorreta.
JUSTIFICATIVA. Falta um método na classe Funcionário.

E - Alternativa incorreta.
JUSTIFICATIVA. A classe Pessoa do diagrama não apresenta o seu construtor.

Alternativa correta: A.

Questão 17.

I - Afirmativa correta.
JUSTIFICATIVA. A generalização nos casos de uso tem papel similar à generalização nos diagramas de classes (BOOCH, RUMBAUGH e JACOBSON; 2006).

II - Afirmativa incorreta.
JUSTIFICATIVA. Como é o caso de uso-base que incorpora o incluído, o relacionamento incluído não pode ser isolado. Além disso, esse relacionamento tem seu comportamento incluído (por isso o nome) e, também, não é o caso de uso-base.

III - Afirmativa incorreta.
JUSTIFICATIVA. A descrição não corresponde a um relacionamento de extensão, pois o caso de uso-base pode permanecer isolado.

IV - Afirmativa correta.
JUSTIFICATIVA. O relacionamento de extensão é utilizado justamente para a modelagem de comportamentos opcionais, que podem ou não ocorrer em determinadas condições.

Alternativa correta: B.

3. Indicações bibliográficas

- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.; *UML: Guia do usuário.* Rio de Janeiro: Campus, 2006.
- PENDER, T. *UML: a Bíblia*. Rio de Janeiro: Campus, 2004.