UNIP
UNIVERSIDADE PAULISTA

SISTEMAS DA INFORMAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 3

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

SISTEMA DA INFORMAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 3

Questão 1

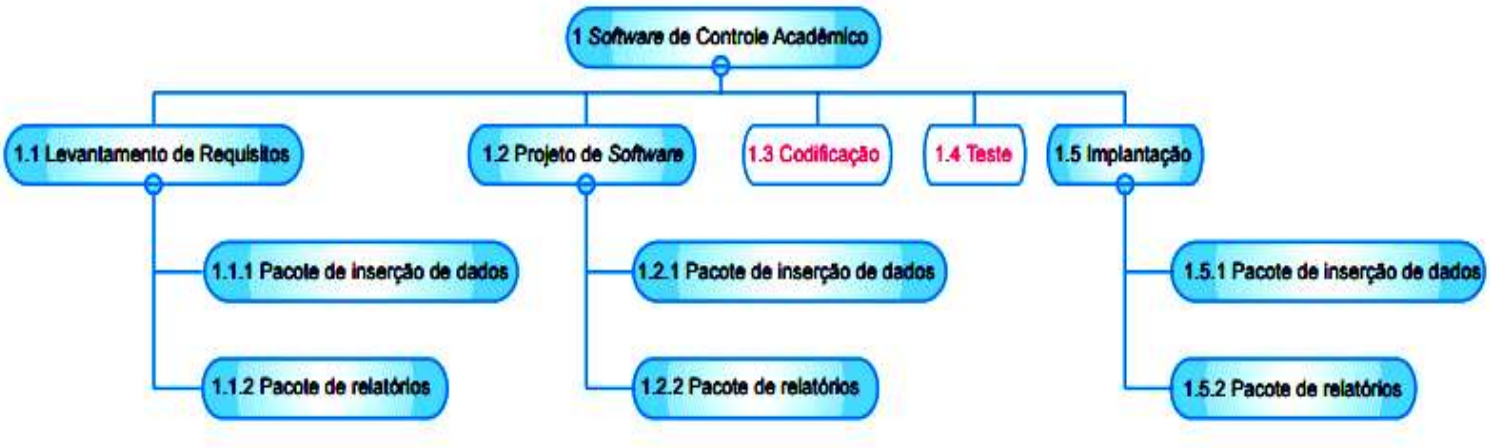
Questão 1.[1]

Leia o texto a seguir.

Conforme definido pelo Corpo de Conhecimento para Gerenciamento de Projeto de Software PMBOK, um dos artefatos de maior importância dentro do planejamento de um projeto de software é a Estrutura Analítica do Projeto (EAP).

PMI. *Um guia do conhecimento em gerenciamento de projetos.* Guia PMBOK. 4. ed. EUA: Project Management Institute, 2008.

A EAP apresentada na figura a seguir mapeia o processo de produção de um software de uma empresa e os pacotes de trabalho que fazem parte desse projeto de software. Percebe-se que as atividades de "codificação" e "teste" não possuem pacotes de trabalho. A empresa Alfa optou por terceirizá-las para uma parceria em uma operação de *outsourcing*.



Analisando-se o EAP da empresa, conclui-se que as atividades de "codificação" e "teste"

A. devem ser inseridas no planejamento do projeto, uma vez que a empresa é a responsável pela implantação do software (a terceirização foi uma opção da própria empresa).
B. devem ser inseridas no planejamento do projeto porque é obrigatório que um processo de software tenha as atividades de "codificação" e "teste" dentro de seu conjunto de atividades.
C. não devem ser inseridas no planejamento do projeto, porque, como elas foram terceirizadas, não é necessário realizar o controle de qualidade dessas atividades.
D. não devem ser inseridas no planejamento do projeto porque, como elas foram terceirizadas, não é necessário realizar o controle de produção - tempo, esforço e custo - dessas atividades.

[1]Questão 9 – Enade 2014.

E. não devem ser inseridas no planejamento do projeto, porque em um processo de software não é obrigatório que as atividades de "codificação" e "teste" estejam dentro de seu conjunto de atividades.

1. Introdução teórica

Estrutura Analítica do Projeto (EAP)

No início de um projeto de software ou de qualquer outra natureza, é importante visualizar o seu escopo, conhecer o que deve ser entregue para o cliente e saber o que deve ser feito para que o plano seja bem sucedido.

Uma das ferramentas utilizadas para gerenciar o escopo de um projeto é a Estrutura Analítica do Projeto (EAP), conhecida, em inglês, como *Work Breakdown Structure* (WBS). Normalmente, a EAP é apresentada como uma árvore hierárquica, sendo que cada nível dessa árvore apresenta os detalhes do projeto.

A EAP deve ser construída com o objetivo de detalhar as entregas do projeto e de delimitar o escopo dessas tarefas.

Segundo o *Project Management Institute* (PMI), no livro *Practice Standard for Work Breakdown Structures*, é importante que a EAP siga a chamada "regra dos 100%". Essa regra afirma que a EAP deve conter todo o trabalho definido pelo escopo de projeto, nem mais nem menos.

2. Análise das alternativas

A - Alternativa correta.

JUSTIFICATIVA. A EAP deve definir claramente o escopo do projeto e as atividades de codificação e de teste fazem parte do projeto, independentemente do fato de terem sido terceirizadas. Logo, seguindo a chamada "regra dos 100%", concluímos que a EAP deve conter as atividades de codificação e de teste. Além disso, tais atividades devem ser incluídas à EAP, uma vez que elas devem ser planejadas, executadas, verificadas e entregues.

B - Alternativa incorreta.

JUSTIFICATIVA. A inclusão das atividades de codificação e de teste é comum na maioria dos projetos de software. Essa situação ocorre em função de tais atividades fazerem parte do escopo e do planejamento do projeto.

C e D - Alternativas incorretas.

JUSTIFICATIVA. O fato de uma atividade ser terceirizada não significa que ela não deva ser controlada: pelo contrário, uma vez que a sua execução está nas mãos de terceiros, ela deve ser monitorada com mais cuidado.

E - Alternativa incorreta.

JUSTIFICATIVA. Uma vez que essas atividades fazem parte do escopo do projeto, elas devem ser inseridas no planejamento.

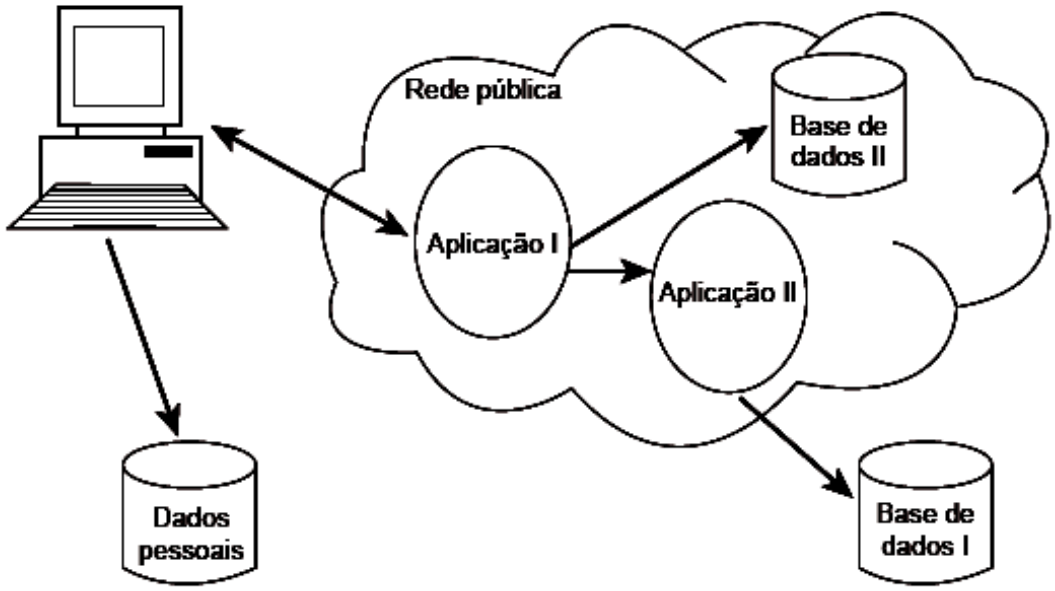
3. Indicações bibliográficas

- PROJECT MANAGEMENT INSTITUTE. *Project Management Institute Practice Standard for Work Breakdown Structures*. Newtown Square: Project Management Institute, 2006.
- PROJECT MANAGEMENT INSTITUTE. *Um guia do conhecimento em gerenciamento de projetos (Guia PMBOK)*. Pennsylvania: Project Management Institute, 2008.

Questão 2

Questão 2.[2]

Considere o arranjo computacional apresentado a seguir.



A característica fundamental esperada para tais sistemas de modo a ter o menor impacto sobre a experiência do usuário final é

A. a transparência entre as entidades do sistema.
B. a linguagem de programação orientada a eventos.
C. o hardware com elevada taxa de processamento de dados.
D. a base de dados deve estar localizada no mesmo espaço fixo.
E. a independência quanto à disponibilidade de conexão à rede de comunicação de dados.

1. Introdução teórica

Sistemas distribuídos

Até a década de 1970, computadores eram recursos caros, sofisticados e centralizados. O custo de produção de uma única máquina era tão elevado que, em geral, apenas governos ou grandes corporações tinham condições de adquirir esses equipamentos.

O uso dos computadores era feito por meio de lotes (*batches*), submetidos de uma só vez ao computador, usualmente na forma de um baralho de cartões perfurados. Ao final da execução, o usuário retirava os resultados em algum escaninho, ou seja, nenhum usuário

[2]Questão 15 – Enade 2014.

acessava o computador diretamente: todos os programas eram executados e controlados pelo "operador do sistema", que submetia os lotes ao computador central, controlava a sua execução, trazia os resultados impressos e colocava-os no escaninho público.

Com a evolução do uso dos sistemas em lotes, surgiram os sistemas interativos. Neles, o usuário acessa o computador e controla diretamente a execução do programa e de todos os periféricos ligados à máquina. Nas primeiras versões, os sistemas interativos eram monousuários, ou seja, apenas um único usuário podia utilizar o computador de cada vez.

Posteriormente, devido ao elevado custo dessas máquinas e à crescente demanda por serviços computacionais, surgiu o "compartilhamento temporal", ou, em inglês, o *time-sharing*. Nesse caso, uma única máquina era diretamente acessada por meio de um conjunto de terminais e os recursos eram distribuídos pelos diversos usuários, cada qual em seu terminal. Dessa forma, para cada usuário, parecia que havia um computador "só para ele". No entanto, na realidade, havia apenas um único computador rodando subdivisões independentes para cada usuário. Assim, uma máquina era capaz de atender a mais de um usuário ao mesmo tempo. Esses sistemas são chamados de sistemas multiusuários.

Com o aumento do número de máquinas e de usuários, houve a necessidade de interligar essas "ilhas de processamento" e surgiram as primeiras redes de computadores.

Com o desenvolvimento das redes de computadores e dos sistemas operacionais capazes de executar mais de um processo simultaneamente (sistemas operacionais multitarefa) e, também, com a capacidade de esses sistemas darem suporte a mais de um usuário simultaneamente (sistemas multiusuários), surgiu a possibilidade do compartilhamento de recursos tanto localmente quanto remotamente.

A comunicação entre computadores remotos possibilitou que sistemas fossem desenvolvidos de forma distribuída, tirando-se proveito não apenas dos recursos locais de uma única máquina, mas de várias máquinas remotas, postas em contato por meio da rede.

O resultado desse procedimento chama-se sistema distribuído, o qual é composto por recursos que executam, simultaneamente, em várias máquinas diferentes e se comunicam por meio da troca de mensagens, que são transportadas pela rede.

2. Análise das alternativas

A – Alternativa correta.

JUSTIFICATIVA. Em um sistema distribuído, no qual diferentes aplicações e bases de dados comunicam-se de forma complexa, é importante que existam interfaces bem definidas entre

as diversas entidades do sistema. Além disso, essas interfaces precisam ser transparentes, ou seja, devem permitir que as entidades se comuniquem sem expor a implementação interna. Dessa forma, a realização de alterações internas nessas entidades, como a correção de defeitos ou a execução de melhorias, não afetam a interface de acesso e fazem com que as aplicações que dependam dessas entidades continuem funcionando normalmente.

B – Alternativa incorreta.
JUSTIFICATIVA. Ainda que o uso de uma linguagem orientada a eventos seja útil na construção desse tipo de sistema, sua simples utilização não garante que o usuário final tenha a experiência adequada.

C – Alternativa incorreta.
JUSTIFICATIVA. Uma vez que as aplicações rodem em um sistema distribuído e se comuniquem por meio de uma rede pública, provavelmente há um ambiente bastante heterogêneo com relação ao hardware utilizado. Ainda há a influência da velocidade da rede pública, que está fora do controle da aplicação.

D – Alternativa incorreta.
JUSTIFICATIVA. A base de dados não precisa estar fisicamente próxima do local em que as aplicações são executadas, uma vez que elas podem ser acessadas remotamente pela rede.

E – Alternativa incorreta.
JUSTIFICATIVA. Um sistema distribuído, como o apresentado na figura do enunciado, certamente depende do acesso à rede para o seu funcionamento, uma vez que as aplicações rodam em diferentes máquinas e em diferentes locais (mas acessíveis via rede pública).

3. Indicações bibliográficas

- COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. *Distributed systems: concepts and design.* Harlow: Addison-Wesley, 2012.
- VERÍSSIMO, P.; RODRIGUES, L. *Distributed systems for system architects*. Boston: Kluwer Academic, 2001.

Questão 3

Questão 3.[3]

Para fins estatísticos, uma empresa precisa armazenar os trajetos que seus representantes comerciais percorrem entre pontos de venda. É importante que para cada local visitado sejam armazenadas, além da informação do próprio local, o local de origem do representante (ponto de venda anterior), o local de destino (ponto de venda posterior), as distâncias percorridas e os tempos de viagem. Esse procedimento permite que estes trajetos possam ser analisados, de forma rápida, do local de origem ao local de destino, bem como no sentido inverso, do local de destino (final do trajeto) ao local de origem (início do trajeto).

O analista responsável pelo sistema, que utilizará os dados armazenados e produzirá os relatórios estatísticos, projetou o seguinte esboço de uma classe que representa um ponto de venda:

```
public class Local {
     private String nome_estabelecimento;
     private String endereco;
     private Local origem;
     private Local destino;
     private float distancia_origem;
     private float tempo_origem;
}
```

A respeito do esboço da classe, avalie as afirmativas.

I. O esboço acima representa uma lista duplamente encadeada.

II. A utilização de um nó de uma estrutura de dados do tipo árvore de busca multivias de grau três seria a solução ideal para o problema porque providenciaria a economia de recursos de memória e de disco.

III. A utilização de uma árvore de pesquisa binária para a solução do problema é normal, desde que o atributo de ordenação da árvore seja `distancia_origem`.

É correto o que se afirma em

A. I, apenas.

B. II, apenas.

C. I e III, apenas.

D. II e III, apenas.

E. I, II e III.

[3]Questão 14 – Enade 2014.

1. Introdução teórica

Estruturas de dados: listas ligadas

A construção de um programa é profundamente influenciada pela escolha das estruturas de dados que dão suporte ao seu funcionamento. O livro intitulado *Algoritmos + Estruturas de Dados = Programas* (WIRTH, 1976) mostra como esses dois conceitos (algoritmos e estruturas de dados) estão relacionados e a importância deles para o trabalho do programador.

Com relação às estruturas de dados, podemos observar as listas (estruturas de dados lineares) e as árvores (estruturas de dados não lineares).

No seu formato mais simples, uma lista ligada pode ser vista como um conjunto de registros que apresentam ligações entre si, pelo menos em um sentido (também chamada de lista encadeada simples), como mostrado na figura 1.

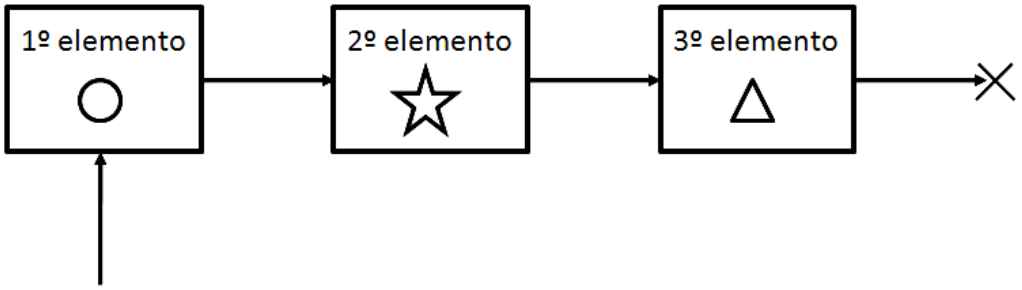


Figura 1. Representação gráfica de uma lista ligada simples com três elementos.

Cada membro da lista deve ter uma referência (ou ponteiro) para, pelo menos, outro elemento. Dessa forma, os elementos da lista estão "ligados" por referências. Isso permite que esses elementos sejam acessados de acordo com uma interface bem definida e de acordo com as referências.

O programador deve estar atento à forma como os elementos devem ser inseridos na lista: no início, no fim ou em algum ponto intermediário. Em alguns casos, pode ser necessário impor um critério de inserção, a fim de garantir que a lista apresente uma ordem particular.

Um exemplo bastante utilizado de lista é o chamado "lista duplamente ligada" (ou "lista duplamente encadeada"). Nesse caso, cada elemento da lista tem uma referência tanto para o elemento seguinte quanto para o elemento anterior, permitindo que o programa percorra a lista em dois sentidos. Caso ocorra a inserção de um elemento novo na

lista, o programador deve atualizar ambas as referências (exceto no caso de inserção nas extremidades).

Além das estruturas de dados lineares, como as listas ligadas, existem também estruturas de dados não lineares, como as árvores. Árvores são utilizadas para representar dados de natureza hierárquica (CELES, CERQUEIRA e RANGEL, 2004).

Tipicamente, em uma estrutura de dados linear, cada elemento apresenta uma ligação para, no máximo, outros dois elementos (o elemento posterior e o elemento anterior). Dessa forma, ao percorrermos esse tipo de estrutura, obtemos um caminho tipicamente linear. Em estruturas de dados não lineares, podem existir várias opções diferentes de percursos. Uma das estruturas de dados não lineares mais importantes corresponde às árvores.

Existem diversos tipos de árvores, construídos com diferentes finalidades, como as árvores binárias e as árvores B. Em uma árvore binária, cada elemento tem, além de uma referência para o nó pai, até duas referências para os nós filhos.

Formalmente, uma árvore pode ser definida de forma recursiva como um conjunto finito *T* de um ou mais nós com as características a seguir (KNUTH, 1997).

- Existe um nó especial chamado de raiz da árvore.
- Os nós restantes (excluindo o nó raiz) são particionados em $m \geq 0$ conjuntos T_1, T_2, ..., T_m e cada um deles também é uma árvore. As árvores T_1, T_2, ..., T_m são chamadas de subárvores da raiz.

2. Análise das afirmativas

I - Afirmativa correta.

JUSTIFICATIVA. A estrutura apresentada tem duas referências para objetos da mesma classe (a classe Local). Logo, trata-se de uma lista duplamente encadeada.

II - Afirmativa incorreta.

JUSTIFICATIVA. Uma árvore de busca multivias (também chamada de árvore B) é uma forma de organização de árvore especialmente projetada para grandes volumes de dados, nos casos em que é impossível armazenar todos esses dados na memória principal do computador. Nesses casos, o computador tem de utilizar alguma forma de armazenamento secundário, que, normalmente, apresenta tempos de acesso muito mais elevados do que os da memória principal.

III - Afirmativa incorreta.

JUSTIFICATIVA. Não é necessária a utilização de uma árvore para a resolução desse problema, uma vez que uma lista duplamente encadeada resolve o problema de forma eficiente e correta.

Alternativa correta: A.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- KNUTH, D. E. *The art of computer programming.* Upper Saddle River: Addison-Wesley, 1997, v. I.
- WIRTH, N. *Algorithms + Data Structures = Programs*. Englewood Cliffs: Prentice-Hall, 1976.

Questão 4

Questão 4.[4]

Leia o texto a seguir.

Uma função é denominada recursiva quando ela é chamada novamente dentro de seu corpo. Implementações recursivas tendem a ser menos eficientes, porém facilitam a codificação e seu entendimento.

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. *Introdução a estrutura de dados*. Rio de Janeiro, 2004 (com adaptações).

Considere a função recursiva f(), a qual foi escrita em linguagem C:

```
1 int f( int v[], int n){
2     if(n == 0)
3        return 0;
4     else{
5        int s;
6        s = f(v, n-1);
7        if( v[n-1] > 0 ) s = s + v[n-1];
8            return s;
9     }
10 }
```

Suponha que a função f() é acionada com os seguintes parâmetros de entrada:

```
f({2,-4,7,0,-1,4},6);
```

Nesse caso, o valor de retorno da função f() será

A. 8.

B. 10.

C. 13.

D. 15.

E. 18.

1. Introdução teórica

Recursividade

Em um programa computacional, quando temos uma função (ou procedimento) chamando a si mesma, chamamos esse mecanismo de recursividade. À primeira vista, o fato de uma função chamar a si mesma pode parecer estranho e talvez até mesmo errado: se uma função chama a si mesma de forma contínua, quando o processo irá parar?

[4]Questão 16 – Enade 2014.

Ao criar uma função recursiva, o programador deve evitar situações em que o programa nunca termine, com uma função chamando a si mesma sem nunca se estabelecer um critério de parada. Dessa forma, deve existir uma condição na qual ocorra recursividade e outra condição na qual a função retorne algum valor.

Em um programa bem comportado, sempre deve haver um momento em que o processo de recursividade é interrompido. A função retorna a um valor que vai ser utilizado em cada chamada anterior da função. Normalmente, queremos trabalhar com programas que devem levar um tempo finito para essa execução e, preferencialmente, o menor tempo possível.

Outro cuidado que devemos tomar ao utilizarmos recursividade, mesmo quando não temos uma situação com infinitas chamadas, é que, se o número de chamadas for muito grande, pode ocorrer uma situação chamada de estouro de pilha. Cada chamada para uma função implica a criação de um item a mais em uma região da memória chamada, a pilha de execução.

Se o número de elementos na pilha de execução crescer demasiadamente, a pilha pode estourar, levando ao fim da execução do programa. Esse problema não é exclusivo de programas que utilizam recursividade, mas é mais comum nesses casos, pois um número excessivo de chamadas a uma função pode levar ao estouro da pilha de execução.

Existem algumas técnicas de otimização que podem evitar situações de estouro de pilha. Uma das mais utilizadas é a chamada de recursividade final própria, ou, em inglês, *tail recursion*. Nesse caso, uma chamada pode ser feita sem a necessidade de se adicionar um quadro na pilha de chamadas, o que evita seu crescimento desenfreado.

2. Análise da questão

Para a resolução do problema, podemos construir uma tabela na qual simulamos a execução do programa. Observe que essa tabela será construída na ordem em que a função f(v,n) retorna aos valores, e não na ordem em que ela é chamada. Isso acontece porque essa é uma função recursiva e o primeiro valor a ser retornado é 0, na linha 3, quando n é igual a 0. A função f(v[0], 0) retorna a s=0 na linha 6. Como v[0]=2>0, sabemos que s=s+2=0+2=2. Sendo assim, f(v,1)=2. Esse valor é novamente retornado à linha 6. Com f(v[1],2), mas v[1]=-4<0, portanto, f(v,2)=2. Esse processo é repetido até que se chegue ao valor f(v,6)=13, conforme tabela 1.

Tabela 1. Resultado de f(v,n) para a execução do programa.

N	f(v,n)
0	0
1	2
2	2
3	9
4	9
5	9
6	13

Alternativa correta: C.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J.R. *Introdução à Estrutura de Dados.* Rio de Janeiro: Campus, 2004.
- CORMEN, T.; LEISERSON, C.; RIVEST, R.; STEIN, C. *Introduction to algorithms*. 3. ed. Cambridge: MIT Press, 2009.
- KNUTH, D. E. *The art of computer programming.* Uppers Saddle River: Addison Wesley, 1997, v. I.
- TOSCANI, L. V.; VELOSO, P.A.S. *Complexidade de algoritmos.* 2. ed. Porto Alegre: Bookman, 2008.

Questão 5

Questão 5.[5]

Nos anos 1970, os sistemas executavam em mainframes com aplicativos escritos em linguagens estruturadas e com todas as funcionalidades em um único módulo, com grande quantidade de linhas de código. Acessos a bancos de dados não relacionais, regras de negócios e tratamento de telas para terminais "burros" ficavam no mesmo programa. Posteriormente, uma importante mudança ocorreu: a substituição dos terminais "burros" por microcomputadores, permitindo que todo o tratamento da interface, e de algumas regras de negócios, passassem a ser feitas nas estações clientes. Surgiam as aplicações cliente-servidor. A partir dos anos 1990 até os dias atuais, as mudanças foram mais radicais, os bancos de dados passaram a ser relacionais e distribuídos. As linguagens passaram a ser orientadas a objetos, cuja modelagem encapsula dados e oferece funcionalidades através de métodos. A interface passou a ser web. Vive-se a era das aplicações em três camadas.

Considerando a evolução da arquitetura de software de sistemas de informação, conforme citado no texto acima, avalie as seguintes afirmações:

I. A separação dos sistemas em três camadas lógicas torna os sistemas mais complexos, requerendo pessoal mais especializado.

II. A separação dos sistemas em três camadas lógicas torna os sistemas mais flexíveis, permitindo que as partes possam ser alteradas de forma independente.

III. A separação dos sistemas em três camadas lógicas aumentou o acoplamento, dificultando a manutenção.

É correto o que se afirma em

A. II, apenas.

B. III, apenas.

C. I e II, apenas.

D. I e III, apenas.

E. I, II e III.

[5]Questão 18 – Enade 2014.

1. Introdução teórica

Arquitetura de software

Uma das maiores dificuldades ao se criar um novo software é garantir uma arquitetura que seja suficientemente flexível para poder crescer e incorporar novas funcionalidades, conforme as necessidades do cliente, mas que também seja robusta, para permitir que esse crescimento possa ser feito com o menor esforço possível.

Essas duas necessidades (flexibilidade e organização), com frequência, caminham em direções aparentemente opostas. Na realidade, o problema é que é muito mais fácil fazer um programa crescer de forma desordenada do que crescer de forma ordenada. A falta de organização pode ser falsamente percebida como flexibilidade. O desafio do arquiteto de software está em encontrar uma arquitetura que englobe tanto flexibilidade quanto organização.

Havia pouco conhecimento sobre o desenvolvimento de programas grandes e complexos e não havia muita preocupação com aspectos arquitetônicos de um programa. Frequentemente, a postura era a de solucionar o problema da forma mais rápida possível, sem nenhuma preocupação com o futuro do programa. Isso levou à geração de uma grande quantidade de códigos "não gerenciáveis": códigos com má qualidade e com má organização, difíceis de manter e de modificar.

O aumento da demanda por sistemas computacionais e o crescente uso de computadores levaram ao surgimento da engenharia de software. Um de seus objetivos é sistematizar a produção do software, incluindo boas práticas no projeto e na codificação de programas. Um dos aspectos fundamentais dessa área é a elaboração de uma arquitetura de software que seja suficientemente flexível para permitir o crescimento futuro, mas também organizada e gerenciável, para permitir que o desenvolvimento seja previsível, especialmente do ponto de vista do esforço necessário para a introdução de novas características, com o menor número possível de defeitos.

Uma das arquiteturas mais difundidas para o desenvolvimento de sistemas computacionais é a chamada arquitetura em camadas. Nessa arquitetura, cada camada do sistema tem responsabilidades específicas e uma interface bem definida. Por responsabilidade, queremos dizer que cada camada apresenta um objetivo claro e bem delimitado. Por interface, queremos dizer que o acesso aos recursos da camada se faz por meio de métodos (funções ou procedimentos, em alguns casos) claramente especificados,

padronizados e estáveis. No entanto, os detalhes técnicos de cada camada devem estar suficientemente isolados, de forma que, para a utilização da interface, não seja necessário conhecer os detalhes internos da sua implementação.

Ao se criar uma arquitetura de software mais sofisticada, surge a necessidade de mão de obra mais qualificada, bem como de um processo de gerenciamento mais adequado. Costuma-se dizer que o preço que se paga para se ter uma complexidade gerenciável é o aumento da complexidade de cada módulo do sistema. Contudo, esse aumento local da complexidade é cuidadoso e planejado e tem como objetivo elevar a qualidade do software desenvolvido, o que torna o desenvolvimento mais previsível e o programa mais flexível.

2. Análise das afirmativas

I - Afirmativa correta.
JUSTIFICATIVA. A divisão de um sistema em camadas certamente implica um aumento da complexidade do código, que deve ter uma arquitetura cuidadosamente projetada para separar responsabilidades adequadas para cada uma das camadas. Contudo, quando devidamente gerenciado, esse aumento de complexidade pode ser compensado, em longo prazo, pela maior facilidade no gerenciamento do desenvolvimento do software e pela maior facilidade de manutenção.

II - Afirmativa correta.
JUSTIFICATIVA. Em um sistema dividido em camadas e com interfaces adequadas, a manutenção e a extensão são muito mais fáceis. Com o correto isolamento conceitual entre cada uma das camadas, alterações podem ser feitas de forma transparente em uma das camadas sem a necessidade de alterações em outras camadas.

III - Afirmativa incorreta.
JUSTIFICATIVA. O objetivo da separação em camadas é diminuir o acoplamento, e não o aumentar. Cada camada deve ser acessível por uma interface bem definida, com o objetivo de desacoplar as camadas e facilitar a extensão e a manutenção do código.

Alternativa correta: C.

3. Indicações bibliográficas

- BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice.* Boston: Addison-Wesley, 2012.
- FAIRBANKS, G. *Just enough software architecture: a risk-driven approach.* Boulder: Marshall & Brainerd, 2010.

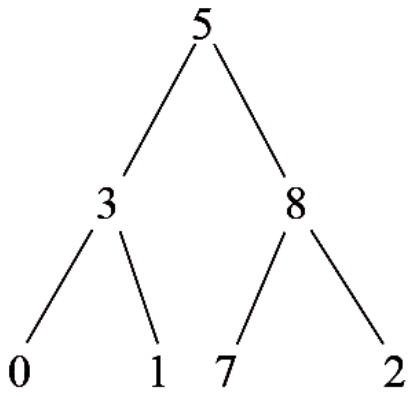
Questão 6

Questão 6.[6]

Existem várias maneiras de se percorrer uma árvore binária. A função a seguir, escrita em pseudocódigo, percorre uma árvore na ordem esquerda-raiz-direita, conhecida por varredura e-r-d recursiva. A função erd() recebe por parâmetro a raiz r de uma árvore e faz uso de seus elementos esq, dir e cont, que representam, respectivamente, ponteiros para uma subárvore à esquerda de r, uma subárvore à direita de r e o conteúdo de r.

```
função erd( árvore r)
{
     se( r != NULO )
     {
          erd( r->esq);
          escreva(r->conteúdo);
          erd(r->dir);
     }
}
```

Considere a árvore binária a seguir.



A sequência correta de exibição do conteúdo da árvore utilizando a função erd() é:

A. 5,3,8,0,1,7,2.

B. 0,1,7,2,3,8,5.

C. 0,3,5,1,7,8,2.

D. 0,3,1,5,7,8,2.

E. 2,7,8,5,0,3,1.

[6]Questão 20 – Enade 2014.

1. Introdução teórica

Árvores binárias

Listas ligadas, pilhas e vetores são estruturas de dados muito úteis para representação de vários tipos de informações em programas computacionais. Contudo, nem sempre conseguimos representar informações utilizando esses tipos de estruturas. Isso ocorre à medida que o relacionamento entre os nós começa a se tornar mais complexo, com mais possibilidades de interligação.

Uma das formas de estrutura de dados mais comuns para a representação de informações hierárquicas é a árvore (CELES, CERQUEIRA e RANGEL, 2004). Na computação, uma árvore corresponde a uma estrutura que contém um nó raiz (desenhado no topo) e uma série de nós filhos (que correspondem aos ramos).

Existem vários tipos de árvores, cada uma com uma finalidade específica. Um dos tipos mais comuns e úteis é a chamada árvore binária. Nesse caso, cada nó pode ter zero, um ou dois nós filhos. Os últimos elementos da árvore, normalmente desenhados na sua porção inferior, são chamados de nós folhas.

Uma das operações importantes que sempre devemos ser capazes de executar em uma estrutura de dados é chamada de percurso. Por exemplo, frequentemente queremos percorrer uma lista, visitando cada um dos seus elementos. Ou então percorrer um vetor, lendo cada um dos seus elementos ou fazendo outra operação com esses elementos. Também podemos percorrer uma árvore binária visitando cada um dos seus elementos.

É conveniente utilizar funções recursivas para situações em que queremos percorrer estruturas de dados como árvores binárias e listas ligadas. Devido a essas estruturas terem referências (ou ponteiros) para outros elementos da mesma classe (ou tipo), a utilização de funções que seguem essas referências é bastante conveniente.

2. Análise da questão

Para resolvermos a questão, enumeramos três pontos importantes da função "erd", mostrados na figura 1. Devemos observar que um número só é impresso quando a função "escreva" é chamada. Quando r for NULO, a função "erd" retorna sem efetuar nenhuma ação. Caso contrário, a função "erd" é chamada de forma recursiva nos pontos 1 e 3.

```
função erd( árvore r)
{
       se( r != NULO )
       {
           (1) erd( r->esq);
           (2) escreva(r->conteúdo);
           (3) erd(r->dir);
       }
}
```

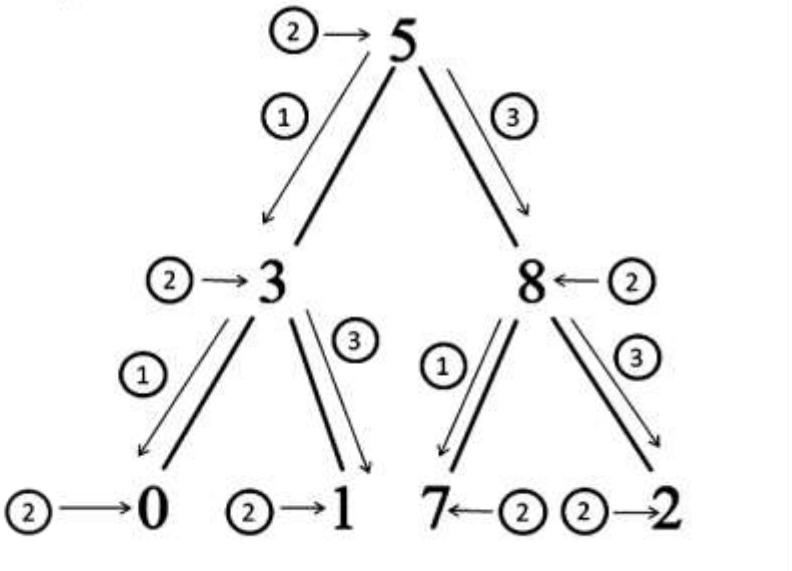


Figura 1. Representação gráfica de uma árvore binária ao lado da função "erd" utilizada para percorrer os elementos de uma árvore.

Para que a função "escreva" seja chamada, a função "erd" chamada no ponto 1 deve retornar, o que ocorre apenas quando r for igual a NULO. Logo, a primeira impressão deve ser a de um dos nós folhas da árvore. Nós folhas são os últimos nós de uma árvore computacional. Observe que, em computação, a árvore é desenhada "de cabeça para baixo", com a raiz no topo do desenho e as folhas na parte de baixo.

Na figura 2, temos o desenho da mesma árvore do enunciado, com a ordem de impressão dos elementos dentro de quadrados ao lado dos nós da árvore. Compare essa figura e a figura 1 e observe os sentidos das setas. O número dentro das circunferências na figura 1 mostra o ponto no código em que o programa toma determinada decisão. Observe que:

- no ponto 1, sempre percorremos o ramo esquerdo da árvore;
- no ponto 2, sempre imprimimos um elemento;
- no ponto 3, sempre percorremos o ramo direito.

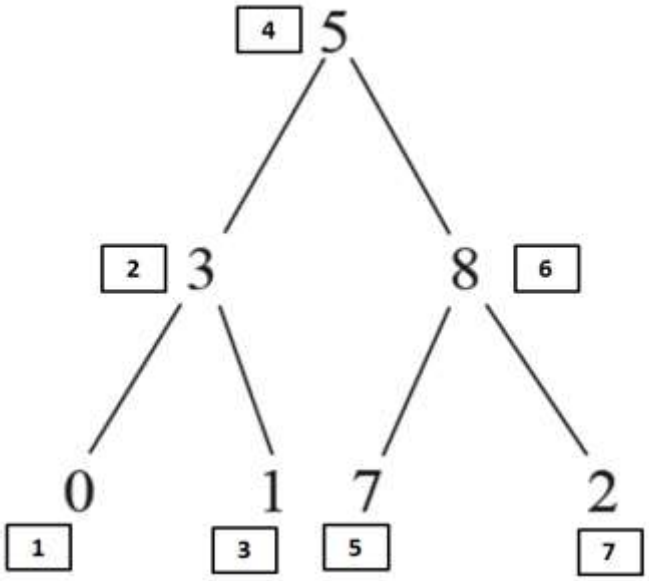


Figura 2. Ordem de impressão dos elementos da árvore binária.

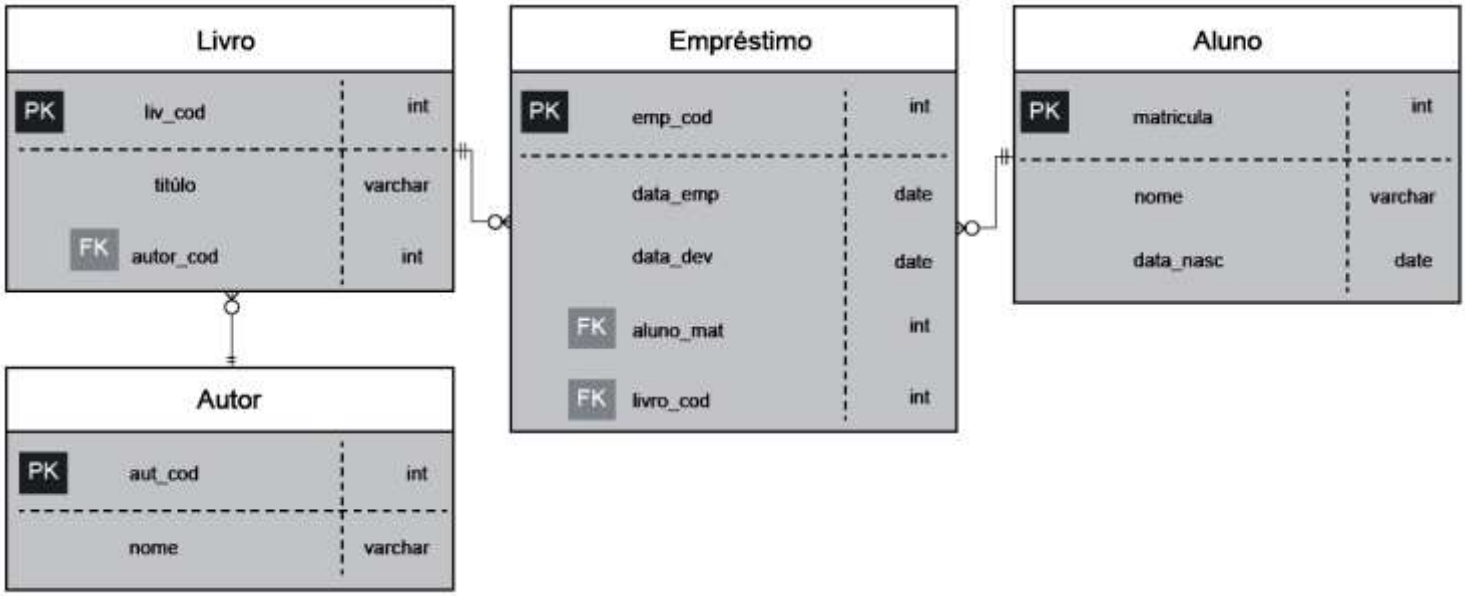
Alternativa correta: D.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- TUCKER, A. B.; NOONAN, R. E. *Linguagens de programação: princípios e paradigmas*. 2. ed. São Paulo: McGrawhill, 2008.

Questões 7, 8 e 9

Questão 7.[7]



O modelo de entidade relacionamento apresentado, representa, de forma sucinta, uma solução para persistência de dados de uma biblioteca. Considerando que um livro está emprestado quando possuir um registro vinculado a ele na tabela "Empréstimo", e essa tupla não possuir valor na coluna "data_dev", o comando SQL que deve ser utilizado para listar os títulos dos livros disponíveis para empréstimo é:

A.
```
select titulo from livro
except
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

B.
```
select titulo from livro
union
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

C.
```
select titulo from livro
except
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is not null
```

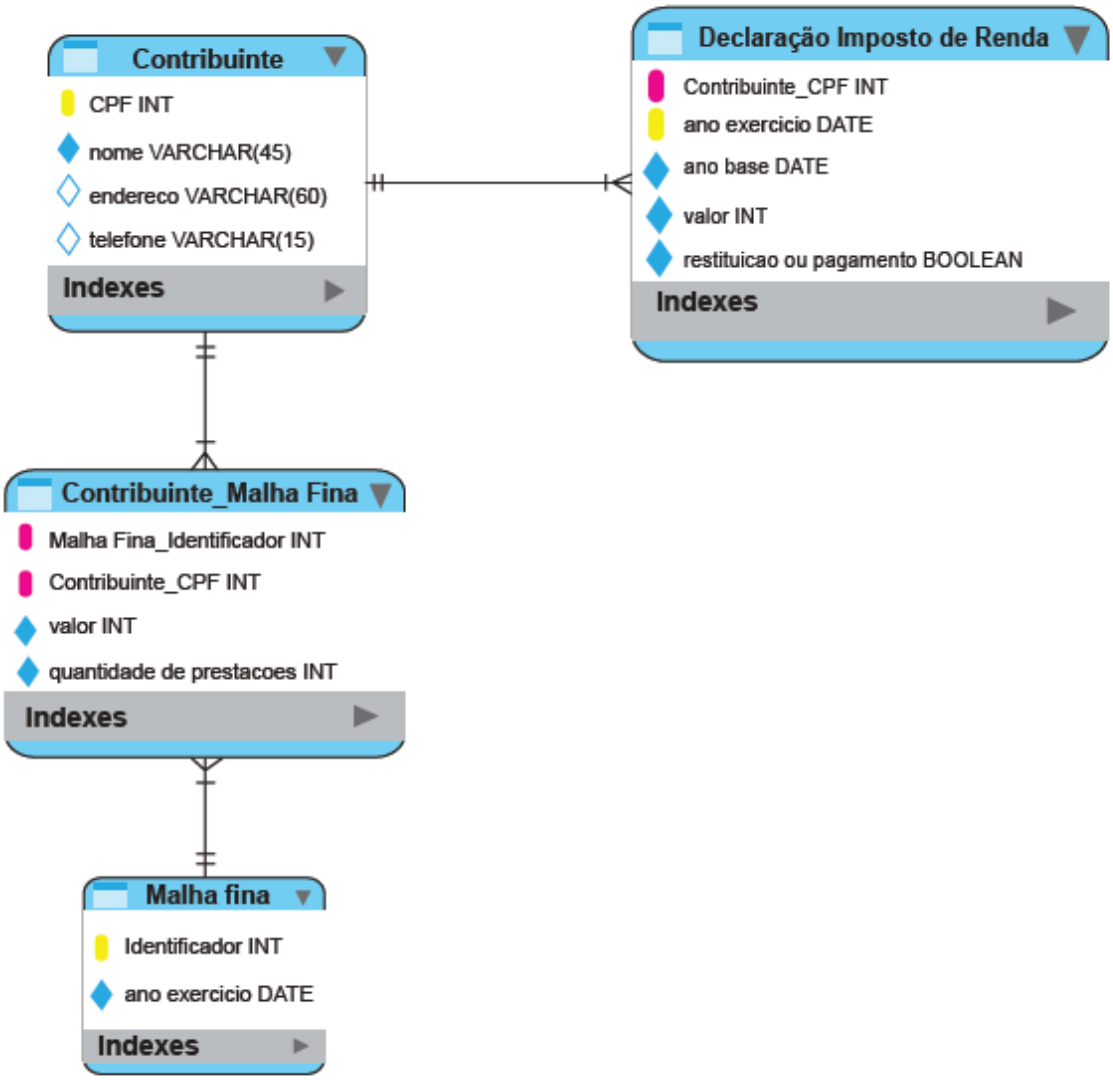
[7]Questão 11 – Enade 2014.

D. `select titulo from livro`
`union select l.titulo from emprestimo e left join livro l`
`on e.livro_cod = l.liv_cod where e.data_dev is null`

E. `select titulo from livro`
`except`
`select l.titulo from emprestimo e right join livro l`
`on e.livro_cod = l.liv_cod where e.data_dev is not null`

Questão 8.[8]

O modelo lógico de dados fornece uma visão da maneira como os dados serão armazenados. A figura a seguir representa o modelo lógico de um ambiente observado em um escritório contábil.



[8]Questão 21 – Enade 2014.

Em relação ao modelo, avalie as afirmativas a seguir.

I. A entidade Declaração Imposto de Renda é uma entidade fraca.

II. O relacionamento entre Contribuinte e Malha Fina é do tipo N:M (muitos para muitos).

III. O atributo CPF da entidade Contribuinte tem a função de chave estrangeira na entidade Declaração Imposto de Renda e no relacionamento Contribuinte_MalhaFina.

IV. A entidade Malha Fina não possui chave primária somente chave estrangeira.

V. O relacionamento Contribuinte_MalhaFina é um relacionamento ternário.

É correto apenas o que se afirma em:

A. I, II e III.

B. I, II e IV.

C. I, IV e V.

D. II, III e V.

E. III, IV e V.

Questão 9.[9]

Leia o texto a seguir.

O modelo relacional representa o banco de dados como uma coleção de relações (tabelas). Na terminologia formal do modelo relacional, uma linha é chamada de "tupla", o título da coluna é denominado "atributo" e a tabela é chamada de "relação". O tipo de dado que descreve os tipos de valores que podem aparecer em cada coluna é denominado "domínio". Um banco de dados relacional pode impor vários tipos de restrições nos dados armazenados.

ELMASRI, R. NAVATHE, S. B. *Sistema de Banco de Dados: Fundamentos e Aplicações*. Rio de Janeiro: LTC, 2002.

Restrições que permitem controlar situações como, por exemplo, "o salário de um empregado não deve exceder o salário do supervisor do empregado" e utilizam mecanismos chamados *triggers* (gatilhos) na sua implementação, são do tipo

A. restrições de domínio.

B. restrições de unicidade.

C. restrições de integridade referencial.

D. restrições de integridade da entidade.

E. restrições de integridade semântica.

[9]Questão 22 – Enade 2014.

1. Introdução teórica

1.1. Banco de dados relacionais

Frequentemente, problemas na área de computação lidam com grande quantidade de dados, gerados de forma artificial (por um programa), coletados do mundo externo (por exemplo, por meio de cadastros ou de instrumentos conectados ao computador) ou frutos de interação do usuário com o computador. Muitas vezes, queremos que esses dados sejam armazenados de forma permanente ou por um período de tempo potencialmente longo.

Podemos armazenar dados de várias formas. Por exemplo, um simples arquivo de texto gravado em um *pen drive* é uma forma válida de armazenamento de dados. Porém, conforme a quantidade de dados armazenados aumenta, pode surgir uma série de problemas: encontrar dados específicos em meio a uma grande quantidade de dados disponíveis, relacionar os dados entre si e atualizar e remover dados.

Com o aumento da capacidade de armazenamento dos computadores, bem como a sua popularização, várias técnicas de organização e de gerenciamento de dados foram desenvolvidas. Uma das formas que se tornou bastante importante no desenvolvimento de grandes sistemas é a dos bancos de dados relacionais.

Seguindo a abordagem de Ritchie (2002), podemos dizer, de forma simplificada, que os bancos de dados relacionais são uma coleção de tabelas interligadas.

As tabelas são formalmente chamadas de relações e cada uma de suas linhas (que são fisicamente registros da tabela) é chamada de tupla. Cada tabela tem várias colunas, chamadas de atributos. Como exemplo, considere a tabela 1 (ou relação) a seguir.

Tabela 1. Exemplo de uma tabela em um banco de dados.

Compositor	
Nome	**Gênero**
Ludwig van Beethoven	Clássico
Johann Sebastian Bach	Barroco
John Coltrane	Jazz
Miles Davis	Jazz
Antônio Carlos Jobim	MPB

Na tabela 1, "Compositor", os atributos (colunas) são "Nome" e "Gênero". Uma tupla dessa tabela seria: "Miles Davis; Jazz".

O diagrama de entidades e relacionamento apresenta uma notação especial para o projeto de banco de dados. Nesses diagramas, existem três elementos básicos: tipos de entidade, relacionamentos e atributos. O primeiro elemento, o tipo de entidade, é representado por um quadrado na maioria das notações, e é utilizado para representar objetos, coisas, pessoas e também eventos (MANNINO, 2008).

Em particular, uma entidade é um membro ou instância de um tipo de entidade. Os atributos são as propriedades dos tipos de entidade, enquanto os relacionamentos são as associações entre os tipos de entidades (MANNINO, 2008).

Outro aspecto importante no modelo relacional é que cada linha de uma tabela deve ter uma combinação única de elementos, não podendo haver linhas repetidas (MANNINO, 2008). Contudo, isso não quer dizer que não exista a possibilidade de que alguns valores individuais não se repitam, mas sim de que a linha toda seja única. Por exemplo, em uma tabela contendo o nome, o CPF e o salário de cada um dos funcionários de uma empresa, pode haver mais de um funcionário com o mesmo nome e salário (mas nunca com o mesmo CPF, que é único para cada indivíduo). Logo, cada linha dessa tabela é única.

Um conjunto de colunas com uma combinação única é chamado de superchave. Por isso, uma linha completa (com todas as colunas) é sempre uma superchave. Contudo, se pudermos remover elementos da superchave até o ponto em que temos uma combinação mínima de colunas (mas ainda única), dizemos que essa superchave mínima é uma chave candidata. Quando uma chave candidata é especialmente designada para uma tabela, dizemos que ela é uma chave primária (MANNINO, 2008).

Para ser possível referenciar tabelas diferentes, pode-se adicionar a chave primária de uma tabela em outra tabela, o que permite a seleção de um registro específico de uma tabela a partir de outra. A isso chama-se chave estrangeira (CORONEL e MORRIS, 2016; MANNINO, 2008).

Em um diagrama de entidade-relacionamento, existe a possibilidade de que uma entidade dependa de outra. Essa entidade pode precisar que parte da sua chave primária venha de outros tipos de entidades (ou até mesmo toda a chave). Nesse caso, dizemos que ela é uma entidade fraca (MANNINO, 2008).

Com o objetivo de garantirmos a integridade das entidades e dos dados armazenados em um banco de dados, introduzimos diversos tipos de restrições, especialmente ligados a cada tipo de integridade.

Assim, se quisermos garantir a integridade referencial, entende-se que a chave estrangeira de uma tabela deve conter os valores vindos da chave primária da tabela pai (ou da chave candidata) ou o valor nulo (CORONEL e MORRIS, 2016).

A integridade semântica significa que cada atributo (ou coluna) apresenta um valor consistente com o tipo de dado. A integridade da entidade significa que cada ênupla tem uma chave primária única não nula (CONRAD, MISENAR e FELDMAN, 2012).

1.2. Consultas e linguagem SQL

Durante muito tempo, para obter as informações contidas nas tabelas dos bancos de dados, era preciso saber fisicamente como essas informações estavam armazenadas e organizadas no banco. O programador precisava conhecer profundamente a forma como o banco funcionava para poder conseguir os dados necessários. Isso levava a um problema de acoplamento: qualquer mudança no banco afetaria a estrutura do código do programa, mesmo que essa mudança não fosse conceitual e fosse apenas armazenamento dos dados. Ao mesmo tempo, mudanças no código também afetavam e costumavam levar a modificações na estrutura do banco de dados.

Para simplificar o desenvolvimento de programas que utilizam bancos de dados, a linguagem SQL foi desenvolvida, partindo-se de um paradigma declarativo, em vez de um paradigma imperativo. Por exemplo, a linguagem C utiliza um paradigma imperativo, enquanto a linguagem Lisp utiliza um paradigma funcional e declarativo.

Dessa forma, o programador deve concentrar-se no resultado que ele busca (os registros desejados do banco), e não na forma como esses registros são encontrados no banco de dados. Além disso, a linguagem SQL foi desenvolvida especificamente a partir da consulta a banco de dados relacionais.

2. Análise das questões

Questão 7.

O resultado desejado corresponde a "encontrar todos os livros, exceto aqueles que estão emprestados". Um livro está emprestado se está presente na tabela "Empréstimo" e não ter valor na coluna "data_dev" (ou seja, essa coluna deve conter o valor NULL). Dessa forma, para selecionarmos todos os livros que estão emprestados, fazemos o que segue.

```
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

Lembrando que o comando "inner join" retorna apenas os registros que apresentam o mesmo valor na coluna "livro_cod" da tabela "empréstimo" e da coluna "liv_cod" na tabela "livro". Além disso, observe a condição "e.data_dev is null", ou seja, apenas os registros que não apresentarem valores em "data_dev".
Se quiséssemos selecionar todos os livros, independentemente de estarem emprestados ou não, deveríamos fazer:

```
select titulo from livro
```

Essa consulta vai retornar todos os títulos presentes na tabela livro. Excluiremos os registros que estão emprestados, retornados na consulta anterior. Isso pode ser feito utilizando-se o comando except, que elimina, da primeira consulta, os registros obtidos pela segunda consulta, conforme segue.

```
select titulo from livro
except
select l.titulo from emprestimo e inner join livro l
on e.livro_cod = l.liv_cod where e.data_dev is null
```

Alternativa correta: A.

Questão 8.

I – Afirmativa correta.
JUSTIFICATIVA. A entidade "Declaração de Imposto de Renda" depende da existência da entidade "Contribuinte" e tem como discriminador o atributo "ano exercício".

II – Afirmativa correta.
JUSTIFICATIVA. A tabela "Contribuinte_Malha Fina" faz o papel de uma tabela de ligação entre as entidades "Contribuinte" e "Malha Fina", justamente porque o relacionamento é do tipo N:M (muitos para muitos).

III – Afirmativa correta.
JUSTIFICATIVA. Tanto na entidade "Contribuinte_Malha Fina" quanto na entidade "Declaração Imposto de Renda", é necessário o atributo CPF, que aponta para a entidade "Contribuinte". Logo, esse atributo é uma chave estrangeira.

IV – Afirmativa incorreta.
JUSTIFICATIVA. O atributo "Identificador" é a chave primária da entidade "Malha Fina".

V – Afirmativa incorreta.
JUSTIFICATIVA. O relacionamento "Contribuinte_Malha Fina" é um relacionamento entre duas entidades, as entidades "Contribuinte" e "Malha Fina" e, portanto, é um relacionamento binário.

Alternativa correta: A.

Questão 9.

A – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não limita o tipo de dados de um atributo e, portanto, não é uma restrição de domínio.

B – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não garante unicidade.

C – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não menciona tipo de relacionamento entre entidades e, portanto, não pode ser vista como uma restrição de integridade referencial.

D – Alternativa incorreta.
JUSTIFICATIVA. A restrição descrita no enunciado não garante que mais de um empregado tenha os valores iguais em uma tabela, apenas garante que o empregado ganhe menos do que o seu supervisor. Por exemplo, dois funcionários poderiam ter o mesmo salário, desde que esse salário fosse menor do que o salário do supervisor.

E – Alternativa correta.

JUSTIFICATIVA. A restrição descrita no enunciado é uma regra de negócio. Logo, é uma restrição de integridade semântica.

3. Indicações bibliográficas

- CONRAD, E.; MISENAR, S.; FELDMAN, J. *CISSP Study Guide*. Waltham: Newnes, 2012.
- CORONEL, C.; MORRIS, S. *Database systems: design, implementation and management.* 12. ed. Boston: Cengage Learning, 2016.
- MANNINO, M. V. *Projeto, desenvolvimento de aplicações e administração de banco de dados.* 3. ed. Porto Alegre: AMGH, 2008.
- RITCHIE, C. *Relational database principles.* London: Thomson Learning, 2002.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database System Concepts.* 6. ed. New York: McGraw-Hill, 2011.

Questão 10

Questão 10.[10]

As classes costumam possuir relacionamentos entre si, chamados de associações, que permitem que elas compartilhem informações entre si e colaborem para a execução dos processos executados pelo sistema.

Com base nesse contexto, construa um diagrama de classes para representar as associações que seguem.

- Uma revista científica possui título, ISSN e periodicidade.
- Essa revista publica diversas edições com os seguintes atributos: número da edição, volume da edição e data da edição. Importante destacar que cada instância da classe edição relaciona-se única e exclusivamente a uma instância da classe revisão científica, não podendo relacionar-se com nenhuma outra.
- Um artigo possui o título e nome do autor. Um artigo é um conteúdo exclusivo de uma edição. E uma edição obrigatoriamente tem que possuir no mínimo 10 e no máximo 15 artigos.

1. Introdução teórica

Orientação a objetos: classes

Na orientação a objetos, utiliza-se o conceito de classes como uma forma de agrupar elementos que apresentam similaridades. Por exemplo, podemos imaginar a classe de todas as escolas de Ensino Médio do Brasil: ela agrupa enorme quantidade de diferentes escolas, de diferentes cidades e de diferentes tamanhos, com diversos alunos. Todas essas escolas têm uma característica em comum: são escolas do Ensino Médio. Contudo, isso não significa que todas as escolas são iguais: provavelmente uma escola bem é diferente da outra.

Essa ideia de agrupamento de elementos que pertencem a uma mesma categoria foi a que inspirou o desenvolvimento do conceito de classe na orientação a objetos. As classes correspondem às categorias de elementos, enquanto os objetos são instâncias de uma classe, de uma forma similar à ideia de que uma escola de Ensino Médio específica poderia ser considerada uma instância da classe "escolas do Ensino Médio".

A UML, acrônimo para a expressão Unified Modeling Language, possui um diagrama específico para trabalhar com classes, chamado de Diagrama de Classes. Os diagramas de

[10]Questão Discursiva 3 – Enade 2014.

classes "são utilizados para capturar as relações estáticas do software" (PILONE e PITMAN, 2006). Ainda segundo PILONE e PITMAN (2006), "uma classe representa um grupo de coisas que têm estado e comportamentos comuns".

A maioria dos programas orientados apresenta mais de uma classe, com diversos tipos de relações entre si. Uma dessas relações é chamada de associação. Podemos dizer que a associação é uma relação do tipo "...tem um...". Por exemplo, suponha a interface gráfica de um sistema operacional que pode ter uma classe para as janelas e outra classe para o ponteiro do mouse. Em dado instante, uma instância da classe janela pode conter um objeto do tipo ponteiro do mouse. Observe que se o usuário mover o ponteiro para outra janela, outro objeto vai "receber" o objeto. Um tipo mais forte de associação é chamado de agregação, que implica uma propriedade e, possivelmente, uma relação entre as linhas de vidas dos objetos envolvidos. Ainda mais forte do que a agregação é a composição, que indica uma relação do tipo "todo-parte". Por exemplo, no caso de uma interface gráfica, poderíamos dizer que uma janela possui uma barra de títulos e isso é representado por uma relação de composição (PILONE e PITMAN, 2006).

É possível especificar, no diagrama de classes, quantas instâncias de uma classe em particular estão envolvidas em uma associação, agregação ou composição, o que é chamado de multiplicidade (PILONE e PITMAN, 2006). O valor é indicado ao lado da classe de que é o alvo e, caso seja omitido, assume-se o valor 1. Exemplos de associações, agregações e composições são mostrados na figura 1.

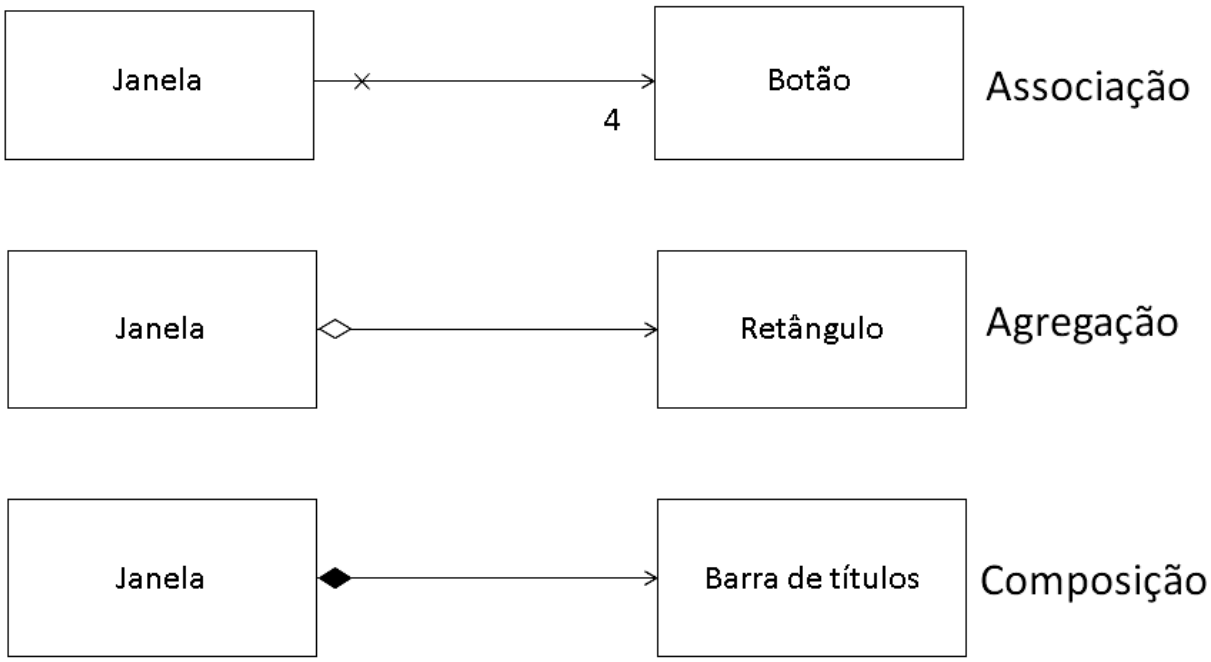


Figura 1. Exemplo de associação, agregação e composição.
Fonte. Adaptado de PILONE e PITMAN (2006).

2. Resposta padrão do INEP

A relação entre as classes Revista_Científica e Edição é muito forte, uma vez que, para existir a edição de uma revista, a revista também deve existir. Além disso, os artigos também fazem parte de uma edição, de forma que ambas as relações vão ser do tipo composição. Além disso, podemos representar os atributos de uma classe no diagrama de classes, dentro do mesmo retângulo, com a visibilidade marcada no lado esquerdo do nome do atributo (por exemplo, o símbolo "-" significa um atributo privado, ou seja, que está disponível apenas para membros da mesma classe).

A classe Revista_Científica possui como atributos: titulo_revista, issn_revista e periodicidade. A classe Edicao possui como atributos numero_edicao, volume_edicao e data_edicao. Finalmente, a classe Artigo possui como atributos titulo_artigo e nome_autor. Observe que uma edição deve possuir de 10 a 15 artigos, o que significa essa deve ser a multiplicidade da composição entre a classe Edicao e Artigo. Finalmente, a multiplicidade da composição entre Revista_Cientifica e Edicao deve ser 0.. ou 1..*, sendo que o último caso implicaria que uma revista deveria ter ao menos uma edição. O padrão de resposta do INEP é mostrado na figura 2.*

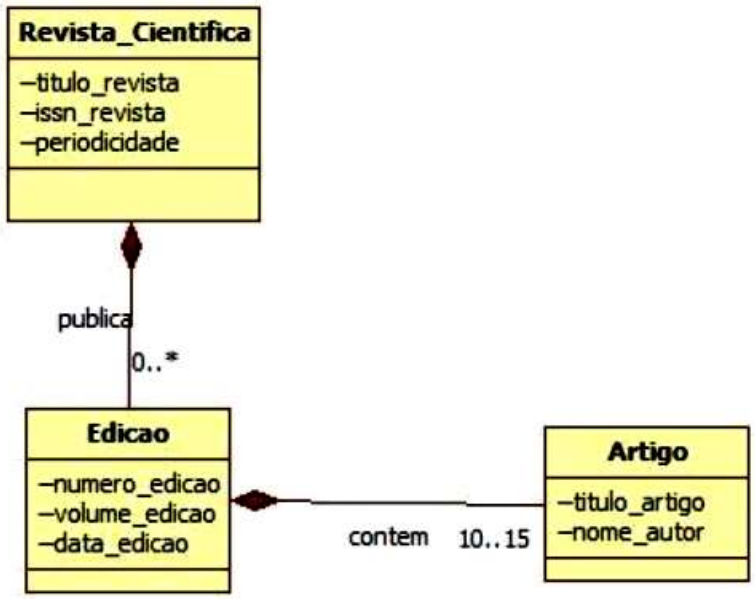


Figura 2. Diagrama de classes para o exercício.
Disponível em <http://download.inep.gov.br/educacao_superior/enade/padrao_resposta/2014/padrao_resposta_tecnologia_analise_desenv_sistemas.pdf>. Acesso em 18 set. 2017.

3. Indicação bibliográfica

- PILONE, D.; PITMAN, N. *UML 2: Rápido e Prático.* Rio de Janeiro: Alta Books, 2006.

Questão 11

Questão 11.[11]

Uma estrutura de dados do tipo pilha pode ser usada em um algoritmo que permite imprimir uma palavra de forma invertida. Exemplo: FELICIDADE deve ser impresso como EDADICILEF.

Considere as variáveis declaradas abaixo.

```
pilha[1..50]: caractere;
i, topo: inteiro;
palavra: string;
```

Em pseudocódigo, faça o que se pede nos itens a seguir.

A. Desenvolva a rotina push que inclui um elemento na pilha.
B. Desenvolva a rotina pop que retira um elemento da pilha.
C. Desenvolva a rotina que leia a palavra e, usando a pilha, a imprima de forma invertida.

1. Introdução teórica

Estruturas de dados: pilhas

A estrutura de dados denominada pilha tem comportamento muito similar ao comportamento de uma pilha de objetos na vida real. Por exemplo, suponha uma pilha de livros sobre uma mesa: idealmente, podemos adicionar livros ao topo da pilha ou retirar um livro do topo da pilha. Os demais livros que se encontram abaixo do livro que está no topo não estão (diretamente) acessíveis. Essa disciplina nos processos de inserção e de remoção garante que a informação sobre a ordem de inserção seja mantida. Dessa forma, o primeiro objeto a ser inserido é o último a ser removido e o último objeto a ser inserido é o primeiro a ser removido, o que costuma ser chamado, em inglês, de *LIFO (Last In, First Out)*.

[11]Questão Discursiva 4 – Enade 2014.

2. Padrão de resposta INEP

Parte A.

A primeira parte da questão corresponde à inserção dos elementos no topo da pilha. Do enunciado, é possível perceber a sugestão do uso de um vetor estático com 50 elementos, ou seja, a pilha vai armazenar no máximo 50 elementos. Uma variável deve conter o índice do elemento que é o topo da pilha, ou seja, o último elemento a ser inserido. Essa variável deve ser incrementada cada vez que inserimos um novo elemento na pilha. É importante controlarmos o tamanho máximo, uma vez que o vetor tem apenas 50 posições. Caso o valor da variável "topo" torne-se maior do que 50, o programa deve imprimir uma mensagem de erro. A variável c contém o caractere a ser depositado no topo da pilha. Isso é ilustrado no pseudocódigo abaixo.

```
função push(c:caractere)
início
  se topo >= 50
    escreva "Erro: Pilha Cheia"
  senão
    topo <- topo + 1
    pilha[topo] <- c
  fimse
fimfunção
```

Parte B.

A segunda função essencial para a implementação de uma pilha é a função pop, que retira o último elemento acrescentado ao topo da pilha. Essa função deve conter uma proteção para o caso de a pilha encontrar-se vazia, pois, nesse caso, não é possível remover nenhum elemento. Finalmente, é importante que a variável "topo" seja decrementada de uma unidade para cada remoção, uma vez que ela deve apontar para o elemento que se encontra imediatamente abaixo do elemento retirado, o qual vai ser o novo topo da pilha. O pseudocódigo a seguir ilustra essa função.

```
função pop(): caractere
início
  se topo <= 0
    escreva "Erro: Pilha Vazia"
  senão
    devolva(pilha[topo])
    topo <- topo -1
  fimse
fimfunção
```

Parte C.

Devido ao próprio funcionamento de uma pilha, em que o último elemento a ser acrescentado é o primeiro a ser retirado, basta acrescentarmos os caracteres na pilha um a um utilizando a função push e, depois, retirá-los utilizando a função pop. Como a pilha é uma estrutura de dados do tipo LIFO, depois de depositarmos todos os caracteres de uma palavra em uma pilha, eles vão ser retirados na ordem reversa a que foram colocados, como ilustrado na figura 1.

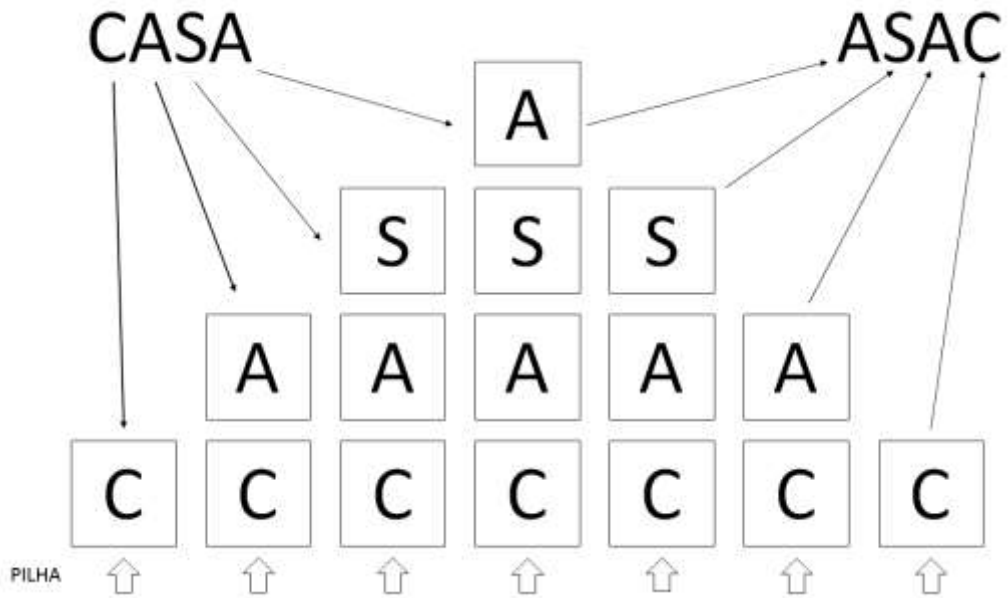


Figura 1. Uma pilha com a palavra casa.

O pseudocódigo do algoritmo é ilustrado logo a seguir.

```
função inverte()
início
      topo <- 0
      leia palavra
      para i <- 1 até tamanho(palavra) passo 1 faça
            push(palavra[i])
      fimpara
      para i <- 1 até tamanho(palavra) passo 1 faça
            imprima (pop())
      fimpara
fimfunção
```

Observe que o programa inicialmente lê toda a palavra, posteriormente insere essa palavra na pilha caractere a caractere (utilizando o comando push) e finalmente retira os caracteres na ordem inversa utilizando o comando pop.

Disponível em <http://download.inep.gov.br/educacao_superior/enade/padrao_resposta/2014/padrao_resposta_tecnologia_analise_desenv_sistemas.pdf>. Acesso em 18 set. 2017.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- FEOFILLOF, P. *Algoritmos em linguagem C*. Elsevier Brasil, 2009.

Questão 12

Questão 12.[12]

Matrizes multidimensionais são vetores capazes de armazenarem mais de uma posição de cada elemento que será indicado por dois ou mais índices. Um exemplo de matrizes multidimensionais são as matrizes matemáticas, que representam valores tabulados em linhas e colunas.

```
01 algoritmo “matriz”
02 var
03 i, j : inteiro;
04 m1 : vetor [1..3, 1..3] de inteiro;
05 m2 : vetor [1..3, 1..3] de inteiro;
06 inicio
07 para i de 1 ate 3 faça
08   para j de 1 ate 3 faça
09         m1[i,j] := i + 1;
10         m2[i,j] := j + 1;
11   fimpara;
12 fimpara;
13 para i de 1 ate 3 faça
14   para j de 1 ate 3 faça
15         se (m1[i,j] = m2[i,j]) então
16               m1[i,j] := 0;
17         senão
18               m2[i,j] := 1;
19         fimse;
20   fimpara;
21 fimpara;
22 fimalgoritmo
```

Considerando o algoritmo acima e com base no teste de mesa, faça o que se pede nos itens a seguir.

A. Apresente os dados dos vetores m1 e m2 ao término da execução da linha 12.

B. Apresente os dados dos vetores m1 e m2 ao término da execução da linha 21.

[12]Questão Discursiva 5 – Enade 2014.

1. Introdução teórica

Vetores e matrizes em linguagens de programação

Diversas linguagens de programação apresentam estruturas de dados lineares, frequentemente chamadas de vetores. Na linguagem C, por exemplo, os vetores correspondem a um conjunto de posições de memória vizinhas, todas de mesmo tamanho e de dado tipo.

Contudo, em virtude de alguns domínios do conhecimento, especialmente na Matemática, pode-se querer trabalhar com estruturas de dados com mais dimensões, como as matrizes. Por exemplo, na linguagem C, podemos declarar uma matriz de números inteiros de 9 elementos (3 por 3) da seguinte forma:

```
int M[3][3];
```

Devemos observar que, na linguagem C, os vetores e matrizes começam do índice zero, de forma que o primeiro elemento de uma matriz seria o elemento M[0][0]. A figura 1 ilustra como os elementos da matriz estão dispostos logicamente. Do ponto de vista físico, as matrizes são armazenadas em uma região contínua de memória, similar aos vetores.

Figura 1. Visualização de uma matriz de inteiros 3 por 3, em C.

2. Padrão de resposta INEP

Parte A.

A matriz m1 armazena o valor da linha (dada pelo incide i) + 1, tendo a seguinte forma:

2 2 2

3 3 3

4 4 4

A matriz m2 armazena o valor da coluna (dada pelo incide j) + 1, tendo a seguinte forma:

2 3 4

2 3 4

2 3 4

Parte B.

Observe que a linha 15 modifica todos os elementos da matriz m1 que têm o mesmo valor que o elemento de mesma posição da matriz m2. Ao observarmos a matriz m1 obtida no item anterior, percebemos que eles correspondem aos elementos da diagonal. Esses elementos são igualados a zero, obtendo-se:

0 2 2

3 0 3

4 4 0

Por outro lado, a matriz m2 é modificada apenas no caso contrário, ou seja, quando os elementos são diferentes, o que corresponde a todos os elementos exceto os da diagonal. Nesses casos, escreve-se 1, obtendo-se:

2 1 1

1 3 1

1 1 4

Disponível em <http://download.inep.gov.br/educacao_superior/enade/padrao_resposta/2014/padrao_resposta_tecnologia_analise_desenv_sistemas.pdf>. Acesso em 18 set. 2017.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. R. *Introdução à estrutura de dados.* Rio de Janeiro: Campus, 2004.
- FEOFILLOF, P. *Algoritmos em linguagem C*. Elsevier Brasil, 2009.

Questões 13 e 14

Questão 13.[13]

Leia o texto a seguir.

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).

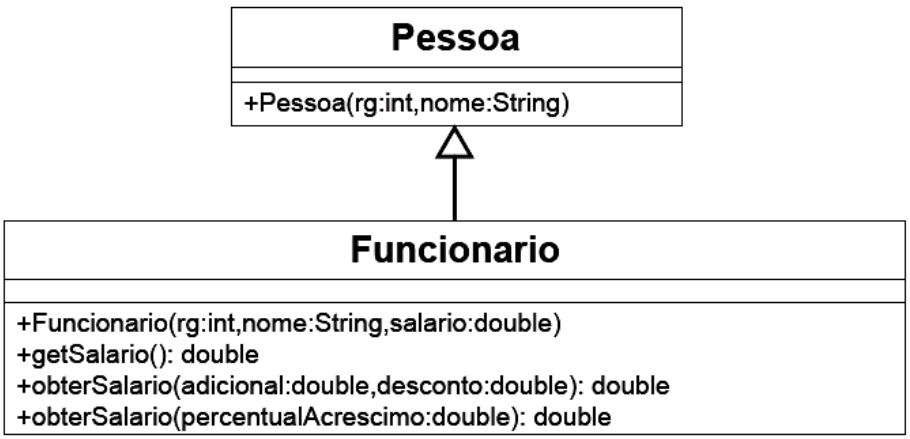
Se um projeto não tem a documentação apropriada ou se está com a documentação desatualizada, uma opção é a engenharia reversa que possibilita mapear códigos para diagramas UML. A seguir, é apresentado um código na linguagem de programação JAVA.

```
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 }
```

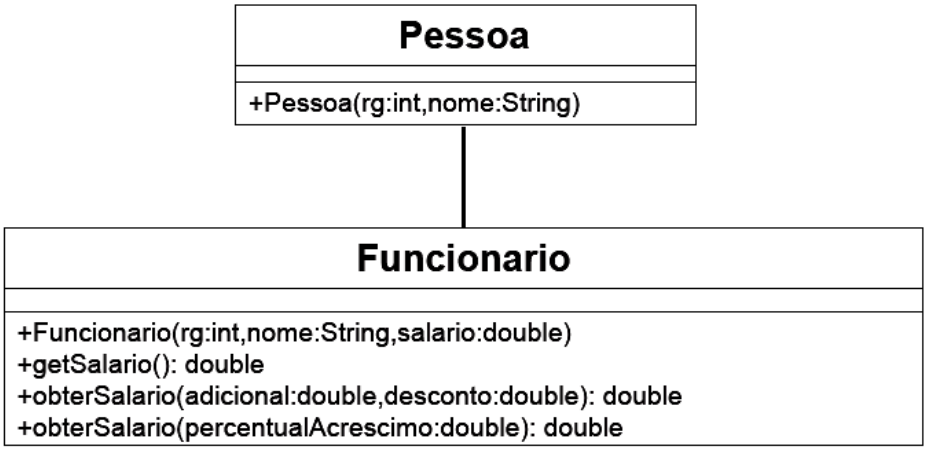
Utilizando a engenharia reversa nesse trecho de código, o diagrama UML de classes correspondente é

[13]Questão 29 – Enade 2014.

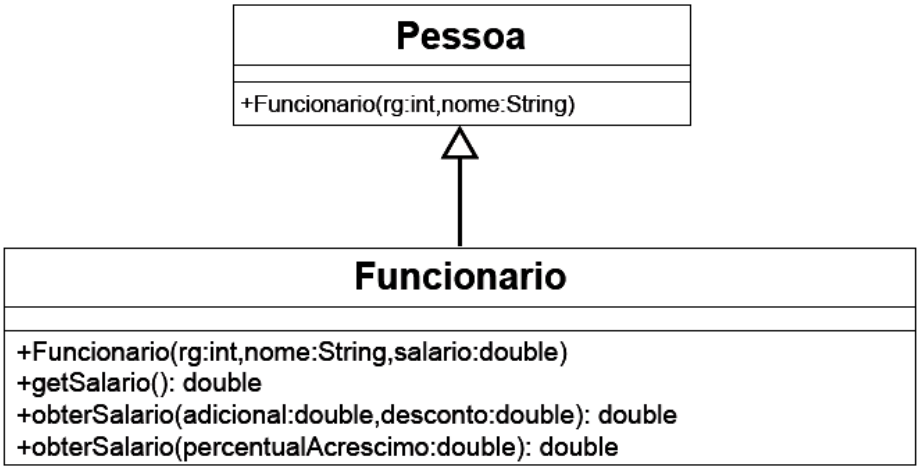
A.


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

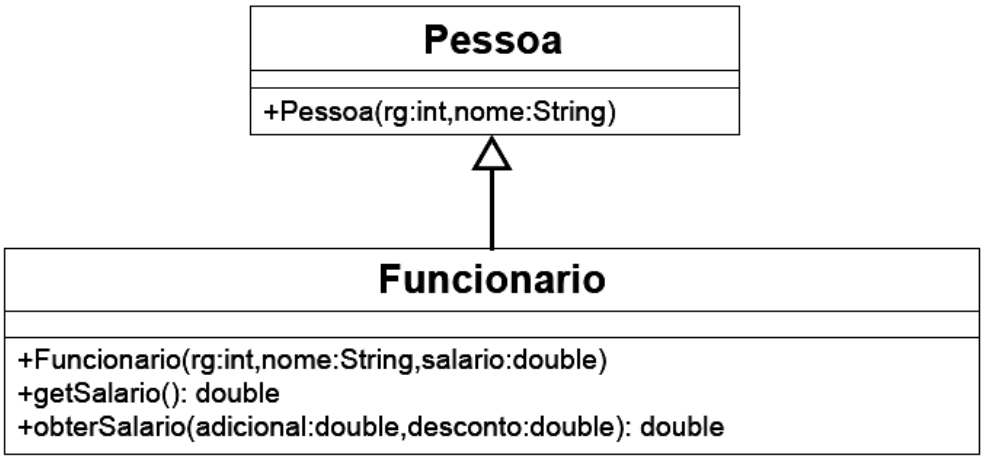
B.


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

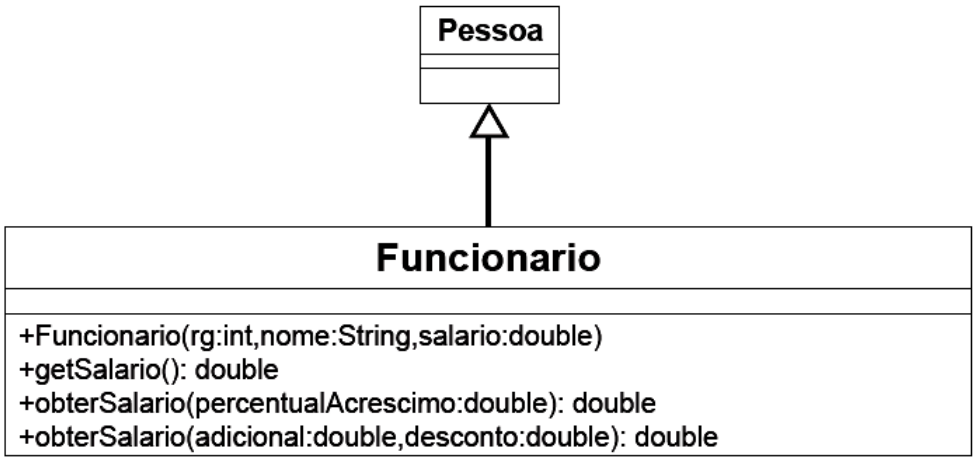
C.


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 14.[14]

Leia o texto a seguir.

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.

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

[14]Questão 17 – Enade 2014.

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 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".

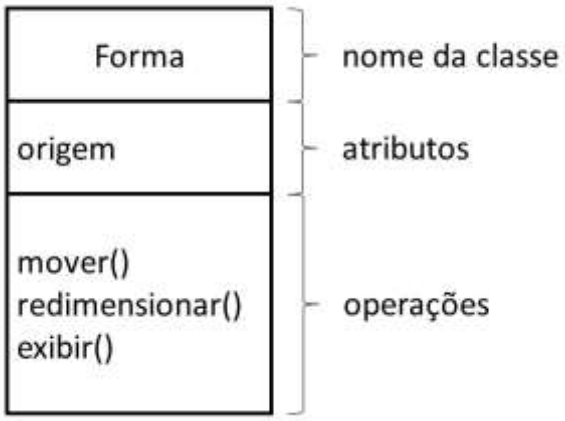


Figura 1. Representação de uma classe na UML.
Fonte. BOOCH, RUMBAUGH e JACOBSON, 2006 (com adaptações).

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 13.

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 14.

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.

ÍNDICE REMISSIVO