UNIP
EDUCAÇÃO, POR SUA ESCOLHA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 10

CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 10

Christiane Mazur Doi

Doutora em Engenharia Metalúrgica e de Materiais, Mestra em Ciências - Tecnologia Nuclear, Especialista em Cálculo e Matemática Aplicada, Especialista em Língua Portuguesa e Literatura, Especialista em Recursos Gramaticais para Revisão Textual, Engenheira Química e Licenciada em Matemática, com Aperfeiçoamento em Estatística e MBA em Engenharia Econômica. Professora titular da Universidade Paulista.

José Carlos Morilla

Doutor e Mestre em Engenharia de Materiais, Mestre em Engenharia de Produção, Especialista em Engenharia Metalúrgica, Especialista em Física e Engenheiro Mecânico, com MBA em Gestão de Empresas. Professor adjunto da Universidade Paulista.

Tiago Guglielmeti Correale

Doutor e Mestre em Engenharia Elétrica e Engenheiro Elétrico - ênfase em Telecomunicações. Professor titular da Universidade Paulista.

Questão 1

Questão 1.[1]

Leia o texto a seguir.

Conceitualmente, cada processo tem sua própria CPU (Central Processing Unit) virtual. É claro que, na realidade, a CPU troca a execução, a todo momento, de um processo para outro, mas, para entender esse sistema, é muito mais fácil pensar em um conjunto de processos sendo executado (pseudo) paralelamente do que tentar controlar o modo como a CPU faz esses chaveamentos.

TANENBAUN, A. S. *Sistemas operacionais modernos*. 3. ed. São Paulo: Pearson, 2010 (com adaptações).

De acordo com o exposto, o conceito descrito denomina-se

A. thread.

B. multiprocessador.

C. multiprogramação.

D. processo monothread.

E. máquina de estados finitos.

1. Análise da questão

Sistemas operacionais: multiprogramação

A questão refere-se à parte inicial do estudo de sistemas operacionais e foca no conceito de multiprogramação. A fim de explicarmos esse conceito de modo didático, vamos imaginar um computador com apenas uma única CPU. Essa CPU tem apenas um único núcleo e não apresenta capacidade de paralelismo de execução, o que é incomum nos dias de hoje, mas facilita o entendimento das ideias aqui abordadas.

Vamos supor, ainda, que essa CPU hipotética possa executar somente uma instrução de cada vez, o que simplifica ainda mais o nosso cenário. Suponha que um sistema operacional execute nesse computador, e que vários processos sejam disparados simultaneamente.

Como uma máquina que pode executar apenas uma única instrução em dado instante de tempo poderia executar vários processos simultaneamente? Uma estratégia para isso é dividir a execução em pequenos intervalos de tempo, de modo que se "trabalhe" em cada processo apenas durante esse intervalo e se mude de processo quando o intervalo esgotar.

Assim, em vez de executar um processo do início ao fim, o computador executa um processo apenas por um intervalo de tempo e vai mudando de um processo para outro. Se essa troca for rápida, o usuário não vai perceber que, na realidade, apenas um processo está

[1]Questão 19 – Enade 2017.

utilizando a CPU em dado instante de tempo e "pode pensar" que vários processos estão sendo executados em paralelo.

Essa estratégia pode parecer pouco intuitiva: por que executar um processo de forma "fragmentada", em de executá-lo totalmente, do início ao fim?

Existem várias respostas, conforme segue.

- Em primeiro lugar, processos podem não ter necessariamente um fim, sendo executados continuamente.
- Em segundo lugar, quando um processo executa, ele pode ter de interromper essa execução para esperar algum dado vindo, por exemplo, do disco rígido, da rede ou de outro dispositivo externo.
- Em terceiro lugar, o processo também pode estar esperando uma entrada ou uma resposta do usuário. Sem dados disponíveis, o processo pode não ter como continuar sua execução (por exemplo, ele pode depender de algum dado para fazer um cálculo). Nesses casos, o trabalho da CPU no processo não pode ser continuado, pois é necessário aguardar determinada informação.

A CPU pode ficar ociosa, somente aguardando a chegada da informação. Contudo, isso é um desperdício de potencial computacional: a CPU poderia executar instruções, mas, por falta de dados, está ociosa.

Na abordagem de multiprogramação, a CPU pode mudar o processo em que está trabalhando em dado instante, interrompendo temporariamente a execução do processo atual. Ela passa a trabalhar em outro processo, enquanto o processo anterior fica aguardando a chegada da informação necessária.

Quando a informação estiver disponível, um mecanismo (como uma interrupção) notifica à CPU que o processo anterior pode ser retomado.

Em resumo, na multiprogramação, o sistema operacional deve gerenciar:

- quais processos estão aguardando alguma informação (e, por isso, encontram-se bloqueados);
- quais processos estão prontos para serem retomados;
- qual processo está sendo executado em um dado instante.

Trabalhando dessa forma, mesmo uma CPU muito primitiva, como a descrita no início desta explicação, poderia tirar proveito da multiprogramação.

2. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. Ainda que o conceito de thread esteja ligado ao paralelismo, ele não é a mesma coisa que multiprogramação. Na realidade, threads apresentam algumas similaridades com a ideia de processo. Contudo, threads podem ser executadas compartilhando o mesmo espaço de endereçamento, enquanto processos têm espaço de endereçamento próprio. Tipicamente, processos são criados de forma independente entre si, enquanto threads podem compartilhar recursos e tendem a ser mais "leves" do que um processo.

B – Alternativa incorreta.
JUSTIFICATIVA. A ideia de multiprocessamento é a de que um computador pode ter várias CPUs, que trabalham em conjunto. Dessa forma, computadores com diversos microprocessadores podem trabalhar com paralelismo real, uma vez que são capazes de executar várias instruções ao mesmo tempo.

C – Alternativa correta.
JUSTIFICATIVA. Uma das tarefas do sistema operacional é fazer com que a execução simultânea de processos ocorra de forma isolada e controlada, sem que o usuário tenha de compreender a execução em um nível específico de detalhamento. Assim, um usuário deve ter a impressão de que vários programas estão sendo efetivamente executados em paralelo.

D – Alternativa incorreta.
JUSTIFICATIVA. O texto descreve a utilização de multiprogramação e não está se referindo diretamente à estrutura de um processo com uma única thread.

E – Alternativa incorreta.
JUSTIFICATIVA. Uma máquina de estados finitos é um modelo conceitual de computação. Esse conceito é especialmente útil no estudo de linguagens formais, especificamente no estudo das linguagens regulares. O texto não descreve uma máquina de estado finita (ou autômato finito).

Alternativa correta: C.

3. Indicações bibliográficas

- DEITEL, H. M.; DEITEL, P.; CHOFFNES, D. R. *Sistemas operacionais*. 3. ed. São Paulo: Pearson Prentice Hall, 2005.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating system concepts*. 9. ed. Hoboken: Wiley, 2013.
- TANENBAUM, A. S. *Sistemas operacionais modernos.* 3. ed. São Paulo: Pearson Prentice Hall, 2009.

Questão 2

Questão 2.[2]

Leia o texto a seguir.

Os modelos de processo foram propostos para trazer ordem ao caos existente na área de desenvolvimento de software. A história mostra que esses modelos trouxeram considerável contribuição no trabalho da engenharia de software.

PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016 (com adaptações).

A respeito dos modelos de processo, avalie as afirmativas.

I. São atividades do modelo incremental: especificação, desenvolvimento e validação.

II. No modelo espiral, a fase de modelagem é responsável, entre outras atividades, pela estimativa, cronograma e análise de risco.

III. O modelo em cascata sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, iniciando na especificação de requisitos e finalizando com a entrega do software concluído.

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.

1. Análise da questão

Modelos de processos

Ao longo da evolução da engenharia de software, diversos modelos de processos foram propostos. O modelo de desenvolvimento chamado de cascata, ou sequencial e linear, é bastante conhecido e utilizado, mas também é alvo de muitas críticas.

No modelo em cascata, as diversas atividades que compõem o desenvolvimento de software são feitas sequencialmente, uma após a outra.

Tipicamente, nesse modelo, temos:

- fase de análise;
- fase de projeto;

[2]Questão 21 – Enade 2017.

- fase de codificação;
- fase de teste;
- fase de implantação;
- fase de manutenção.

O resultado de uma fase funciona como entrada para a fase seguinte. Dessa forma, erros podem ser acumulados e repassados para as fases posteriores, não havendo a possibilidade de retroalimentação ou de comunicação entre etapas consecutivas.

Os erros e os problemas acumulados ao longo do desenvolvimento do modelo em cascata tendem a gerar atrasos em projetos, estouro no orçamento e programas de qualidade ruim. Além disso, pode ocorrer elevado grau de pressão e frustração nas pessoas envolvidas no projeto e desconfiança por parte do cliente.

Outros modelos de processo foram propostos, em grande parte, para solucionar os problemas do modelo em cascata. Muitos desses modelos são baseados na ideia de iterações sucessivas. Assim, nesses casos, em vez de haver referência a um processo rígido, no qual as fases se seguem umas às outras de forma linear, o processo é quebrado em ciclos, em que algumas etapas são feitas várias vezes, mas, a cada vez que o ciclo roda, há refinamento do resultado anterior. Esses modelos são frequentemente chamados de evolutivos (ou evolucionários), pois o produto (software) evolui ao longo de sucessivas iterações, ou seja, não é criado de uma só vez.

Em um típico modelo iterativo, cada iteração envolve:

- fase de comunicação com o cliente;
- fase de planejamento;
- fase de modelagem;
- fase de construção (incluindo a realização de testes);
- fase de implantação.

Ao fim da iteração, deve existir um programa pronto e funcional para o cliente, ainda que o produto não tenha a totalidade das características desejadas. Novas iterações adicionam novas características. Assim, há a possibilidade de que, durante a interação com o cliente, exista a avaliação do produto ao longo do processo de desenvolvimento, e não apenas no final, como ocorre no modelo em cascata.

2. Análise das afirmativas

I – Afirmativa correta.
JUSTIFICATIVA. No modelo incremental, de natureza evolutiva, as atividades de especificação, desenvolvimento e validação são feitas diversas vezes e o software é desenvolvido em incrementos. Frequentemente, o produto vai sendo desenvolvido com base nas etapas anteriores, por meio da adição de novas características.

II – Afirmativa incorreta.
JUSTIFICATIVA. No modelo em espiral, as atividades de realização de estimativa, cronograma e análise de risco pertencem à fase de planejamento, e não à fase de modelagem.

III – Afirmativa correta.
JUSTIFICATIVA. No modelo em cascata, também chamado de sequencial e linear, o desenvolvimento é feito em atividades que se sucedem no tempo, sem a possibilidade de "retorno" ou retroalimentação. A ideia é a de que as atividades sejam feitas sequencialmente, e, ao final do processo, exista um software pronto.

Alternativa correta: D.

3. Indicações bibliográficas

- MACHADO, F. N. R. *Análise e gestão de requisitos de software: onde nascem os sistemas.* 3. ed. São Paulo: Érica, 2016.
- PAGE-JONES, M. *Fundamentos do desenho orientado a objeto com UML*. São Paulo: Makron Books, 2001.
- PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016.

Questão 3

Questão 3.[3]

Um desenvolvedor de software recém-formado foi contratado para a implementação de um projeto em uma empresa e, em reunião, recebeu várias explicações sobre como a gerência de configuração funcionava.

Considerando essa situação, avalie as afirmativas, referentes às informações dadas ao desenvolvedor.

I. Inicialmente, para ter acesso à base de desenvolvimento, o profissional deve realizar uma operação de *checkout* para baixar os arquivos do projeto que estão armazenados no servidor.

II. Na situação em que mais de um desenvolvedor estiver modificando um mesmo documento, ao se tentar realizar uma operação de *commit*, pode ser necessário realizar uma operação de *tag*(*release*) para resolução do conflito entre a versão local e a versão mais recente no repositório, caso algum desenvolvedor tenha submetido uma mudança no documento previamente.

III. No desenvolvimento de um novo caso de uso, em que diversos arquivos sejam modificados, é recomendada a criação de uma ramificação (*branch*).

IV. A versão estável é o ramo principal de desenvolvimento, que segue do começo do desenvolvimento até o momento presente.

É correto apenas o que se afirma em

A. I e II.

B. I e III.

C. II e IV.

D. I, III e IV.

E. II, III e IV.

[3]Questão 22 – Enade 2017.

1. Análise da questão

Gerenciamento da configuração e gerenciamento de versão

O processo de desenvolvimento de um software envolve a produção de diversos artefatos, sendo que um dos principais é o código-fonte. Muitos outros artefatos são desenvolvidos, como documentação, casos de uso e planos de projeto.

O desenvolvimento de qualquer software minimamente sofisticado requer a criação e a alteração de vários arquivos. Por exemplo, o desenvolvimento de um *site* utilizando a tecnologia Java pode envolver a criação de arquivos ".java", os quais contêm o código-fonte que vai dar origem a:

- arquivos com extensão ".class" (por meio de compilação);
- arquivos com extensões ".html" e ".css" (que vão dar origem às páginas *web* propriamente ditas);
- arquivos com extensões como ".jsp" (também relacionados às páginas *web*);
- arquivos ".xml" (que envolvem diversos tipos de configurações).

Esses arquivos são apenas aqueles relacionados à tecnologia de desenvolvimento em si. Há, também, os arquivos relacionados à análise e ao projeto (como, por exemplo, casos de uso e diagramas de classe).

Melhorias de um software podem requerer a modificação de muitos dos arquivos citados. Além disso, frequentemente, tais melhorias demandam o trabalho de uma equipe de pessoas, e não de um único indivíduo. Assim, é necessário que exista a coordenação da criação e da modificação de arquivos que são feitas por grande número de pessoas ao longo de grande período.

O trabalho de gerenciamento da configuração envolve a identificação, o controle e as implementações de mudanças, além de comunicação dessas mudanças aos grupos de pessoas interessadas. O gerenciamento de versão, um dos aspectos do gerenciamento da configuração, é auxiliado por programas que permitem armazenar tanto arquivos quanto informações de suas versões e do fluxo de trabalho nesses arquivos. Há uma ampla classe de programas que executam funções de gerenciamento de versões, com destaque para o *git*, utilizado em muitos projetos de código aberto.

2. Análise das afirmativas

I – Afirmativa correta.
JUSTIFICATIVA. Em muitos programas de gerenciamento de versão, a solicitação dos arquivos do projeto é feita por meio do comando checkout. A ideia básica é que esses arquivos serão mantidos em uma base de dados com diversas informações sobre as suas versões. Para trabalhar no projeto, o desenvolvedor utiliza uma ferramenta de controle versão, como, por exemplo, o Apache Subversion.

II – Afirmativa incorreta.
JUSTIFICATIVA. Quando dois desenvolvedores trabalham em paralelo sobre um mesmo conjunto de arquivos, pode acontecer de ocorrerem mudanças simultâneas em um mesmo arquivo. Isso provoca conflito entre as versões desenvolvidas. Esse conflito pode ser resolvido com um processo que chamamos de *merge,* ou seja, devemos "juntar" as modificações feitas pelos desenvolvedores em um único arquivo.

III – Afirmativa correta.
JUSTIFICATIVA. Se um novo caso de uso for ser criado e isso implicar a modificação de vários arquivos, pode ser necessária a criação de uma ramificação (*branch*, em inglês) para indicar o motivo claro das alterações em vários arquivos. Isso também auxilia nos processos de gerenciamento de versão e de gerenciamento de configuração.

IV – Afirmativa incorreta.
JUSTIFICATIVA. Normalmente, a versão estável não é igual à versão de desenvolvimento. O ramo da versão estável é, em geral, aquele que passou por vários testes e foi entregue ao cliente. Esse ramo não deve ser modificado, a não ser no caso de serem necessárias pequenas correções de defeitos (*bugs*). O ramo de desenvolvimento, por outro lado, é continuamente modificado, o que inclui a implementação de novas características pelos desenvolvedores. Com frequência, essa versão é considerada instável, uma vez que ainda não passou por processos de teste e de uso mais intensivo, como a versão estável.

Alternativa correta: B.

3. Indicações bibliográficas

- CRNKOVIC, I.; ASKLUND, U.; DAHLQVIST, A. P. *Implementing and integrating product data management and software configuration management*. Boston: Artech House, 2003.
- MACHADO, F. N. R. *Análise e gestão de requisitos de software: onde nascem os sistemas.* 3. ed. São Paulo: Érica, 2016.
- PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016.

Questão 4

Questão 4.[4]

Um software com defeito pode ser consequência de problemas no levantamento dos requisitos, uma vez que o requisito pode ser ambíguo porque o cliente não estava convicto de sua real necessidade ou porque a equipe o interpretou mal e registrou uma especificação de forma incorreta. Por esses motivos, as verificações, as validações e os testes são fundamentais para se certificar da qualidade do software resultante.

Considerando esse contexto, avalie as afirmativas.

I. O teste funcional certifica se o software desempenha as funções especificadas nos requisitos.

II. O teste de desempenho valida a conformidade da especificação do processo de desenvolvimento de software.

III. O teste de aceitação é realizado pelo cliente a fim de validar se aquilo que foi implementado é o que foi solicitado.

IV. O teste de instalação, invariavelmente, é executado no local determinado pelo cliente para instalação do software.

V. As técnicas de verificação e validação de software asseguram que o sistema que está sendo desenvolvido seja adequado ao seu propósito.

É correto apenas o que se afirma em

A. I e IV.

B. I, III e V.

C. II, III e V.

D. II, IV e V.

E. I, II, III e V.

1. Análise da questão

Testes de software

A questão aborda diferentes aspectos ligados ao teste do software.

O teste de software está ligado a dois conceitos fundamentais: validação e verificação.

[4]Questão 23 – Enade 2017.

Na verificação, estamos especialmente interessados em determinar se o software está sendo construído de acordo com a sua especificação. Na validação, focamos em saber se o software que está sendo construído atende ao usuário.

Diversos tipos de testes são feitos ao longo do desenvolvimento de um programa. Alguns testes são feitos de forma muito específica, como os testes unitários, nos quais o menor componente possível do software é testado de forma isolada. Outros testes podem ser feitos, com o objetivo de avaliar mais módulos e suas interações, o que leva aos testes de integração. Ao incluirmos todos os componentes no processo de teste, podemos testar o programa do ponto de vista das expectativas do cliente, no chamado teste de validação.

Finalmente, o teste de sistema engloba elementos que vão além do software em si, como interações com o hardware e questões de performance, entre outros aspectos.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. Os testes funcionais seguem uma abordagem do tipo "caixa-preta", na qual estamos interessados em nos certificarmos se a saída do programa está de acordo com as especificações e ignoramos qualquer conhecimento sobre sua estrutura interna e seu código-fonte.

II – Afirmativa incorreta.

JUSTIFICATIVA. Os testes de desempenho não estão ligados à especificação do processo de desenvolvimento, mas, sim, à performance da aplicação. Um exemplo de teste de desempenho é medir o tempo de resposta de uma aplicação web, ou seja, o tempo que o sistema demora para responder à dada solicitação do usuário.

III – Afirmativa correta.

JUSTIFICATIVA. Os testes de aceitação são testes do tipo "caixa-preta", mas feitos pelos próprios usuários (ou por um grupo selecionado de usuários), e não pelos desenvolvedores ou por uma equipe específica da empresa que se debruça sobre o software.

IV – Afirmativa incorreta.

JUSTIFICATIVA. O teste de instalação pode ser feito em um ambiente que é uma cópia do ambiente final no qual ele vai ser executado. Esse ambiente replicado torna o processo de teste menos inseguro, pois um erro não vai impactar na produção do cliente.

V – Afirmativa correta.

JUSTIFICATIVA. O objetivo de se utilizar tanto técnicas de validação quanto técnicas de verificação é justamente assegurar que o software entregue atenda às necessidades do cliente. Evidentemente, o desenvolvimento de software de qualidade depende de vários fatores, como uma boa equipe de desenvolvimento, um bom gerenciamento do projeto e uma análise adequada dos requisitos, entre muitos outros. Mas a utilização de técnicas cujo foco está no controle da qualidade é de suma importância para que o resultado seja satisfatório, visto que projetos de desenvolvimento tendem a ser complexos e ter muitos riscos envolvidos.

Alternativa correta: B.

3. Indicações bibliográficas

- KOIRALA, S.; SHEIKH, S. *Software testing.* Sudbury: Jones & Bartlett Publishers, 2009.
- MYERS, G. J.; BADGETT, T.; SANDLER, C. *The art of software testing*. 3. ed. Hoboken: Wiley. 2012.
- PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016.

Questão 5

Questão 5.[5]

Nas décadas de 1970 e 1980, muitos sistemas corporativos foram desenvolvidos com a linguagem Cobol, utilizando o Sistema Gerenciador de Banco de dados ADABAS e arquivos indexados do tipo ISAM e VISAM. Alguns desses produtos de implementação foram ou estão sendo descontinuados pelos seus fabricantes. Por isso, o trabalho de reengenharia desses sistemas, utilizando linguagens mais modernas, como Python, Java ou mesmo C++, associadas com sistemas de banco de dados mais atuais, apresenta-se como uma boa oportunidade de negócios.

Considerando esse cenário, avalie as afirmativas.

I. A dificuldade de reengenharia de sistemas antigos deve-se ao fato de que, na maioria das vezes, o desenvolvedor definia o sistema e esse já era o próprio processo da organização.

II. O custo de alteração para modernização de uma linha de código em Cobol é alto, por isso, fazer a manutenção desses sistemas é menos dispendioso.

III. Uma estratégia de conversão dos referidos sistemas para uma linguagem orientada a objetos é definir uma estrutura de classes e métodos e realizar o refatoramento do código.

É correto o que se afirma em

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

1. Análise da questão

Sistemas legados

Alguns setores da sociedade foram pioneiros na utilização da computação. Os primeiros computadores foram destinados basicamente às pesquisas científicas (de modo especial, às pesquisas voltadas ao desenvolvimento de sistemas militares). Esses computadores foram

[5]Questão 24 – Enade 2017.

rapidamente utilizados, também, por governos. Em seguida, bancos, instituições financeiras, empresas de grande porte passaram a utilizar "computação" internamente.

Conforme o custo e o tamanho dos computadores foram se reduzindo, seu uso tornou-se mais amplo em diferentes segmentos da economia. Finalmente, com o surgimento da computação pessoal, essa "ferramenta" ficou mais acessível para muitas pessoas.

Empresas e instituições que adotaram sistemas enfrentaram um problema: a manutenção de sistemas antigos. Na realidade, isso se tornou um problema bastante comum em muitas empresas, mesmo na atualidade.

Conforme o tempo passa e as tecnologias antigas ficam obsoletas, a manutenção de máquinas e sistemas torna-se mais complexa. Além disso, o mercado oferece menor quantidade de profissionais que conhecem o funcionamento dessas tecnologias, e o custo do trabalho desses profissionais tende a ser maior.

Em dado momento, pode ser interessante simplesmente abandonar o sistema antigo e desenvolver um novo sistema para substituir o anterior. A questão é que isso envolve investimento (que pode ser potencialmente elevado, dependendo da complexidade do sistema anterior), tempo e riscos. Sistemas antigos podem ser considerados mais estáveis pelo usuário, uma vez que boa parte dos seus defeitos pode ter sido identificado e corrigido ao longo dos vários anos de uso. Sistemas novos têm elevada chance de apresentarem defeitos, o que pode ser considerado ruim pelo usuário.

Assim, as empresas precisam continuamente decidir até que ponto vale a pena manter um sistema antigo ou, de modo alternativo, desenvolver um sistema completamente novo para substituir o anterior. Essa decisão também é afetada pelo grau de estabilidade do sistema antigo, pela sua criticidade para o negócio e pelos custos de novas melhorias e modificações no sistema atual.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. Durante muito tempo, empresas não dispunham de um processo de desenvolvimento de software bem definido. Uma boa parte dos primeiros sistemas foi construído sem o uso de metodologias específicas, com pouca ou nenhuma documentação e sem haver a preocupação com a manutenção ou com a expansão do software no futuro.

II – Afirmativa correta.

JUSTIFICATIVA. Muitas instituições, especialmente as ligadas ao setor financeiro, vêm investindo em informática há muito tempo e têm programas antigos, mas absolutamente vitais para o funcionamento do seu negócio. Devido aos riscos e aos custos envolvidos em uma migração de sistemas, muitas vezes, é mais interessante manter o código antigo do que ir para uma nova plataforma. Adicionalmente, muitas empresas têm a capacidade de executar o código antigo em máquinas novas.

III – Afirmativa correta.

JUSTIFICATIVA. Na conversão de sistemas antigos em sistemas novos, utilizam-se técnicas mais modernas de desenvolvimento. A orientação a objetos é uma ótima alternativa de abordagem para o desenvolvimento de sistemas computacionais, mas é importante termos em mente que não se trata da única.

Alternativa correta: E.

3. Indicações bibliográficas

- BELL, D. *The class diagram.* Disponível em <https://developer.ibm.com/ articles/the-class-diagram/>. Acesso em 04/02/2020.
- BLAHA, M.; RUMBAUGH, J. *Modelagem e projetos baseados em objetos com UML 2*. 2. ed. Rio de Janeiro: Elsevier, 2006.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do usuário*. 2. ed. Rio de Janeiro: Elsevier, 2005.
- PAGE-JONES, M. *Fundamentos do desenho orientado a objeto com UML*. São Paulo: Makron Books, 2001.
- PRESSMAN, R. S. *Engenharia de software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016.

Questão 6

Questão 6.[6]

O fragmento de código C++, a seguir, contém a implementação da inserção de um nó em uma árvore binária.

```
...
struct noArvore {
      int dado;
      noArvore *esquerda;
      noArvore *direita;
};
noArvore *insere(noArvore *arvore, int valor) {
      if (arvore == NULL) {
            arvore = new noArvore;
            arvore->esquerda = NULL;
            arvore->direita = NULL;
            arvore->dado = valor;
      } else if(valor < arvore->dado) {
            arvore->esquerda = insere (arvore->esquerda, valor);
      } else {
            arvore->direita = insere (arvore->direita, valor);
      }
      return(arvore);
}
...
```

Com base no exposto, assinale a opção que apresenta o fragmento de programa para imprimir os valores contidos nos nós, percorrendo a árvore em pré-ordem.

A.
```
void preorder(noArvore *raiz) {
      if (raiz != NULL) {
            cout << raiz->dado << " ";
            preorder(raiz->esquerda);
            preorder(raiz->direita);
      }
   }
```

B.
```
void preorder(noArvore *raiz) {
   if (raiz != NULL) {
      preorder(raiz->direita);
      preorder(raiz->esquerda);
      cout << raiz->dado << " ";
   }
```

[6]Questão 25 – Enade 2017.

```
    }
```

C.
```
  void preorder(noArvore *raiz) {
      if (raiz != NULL) {
        preorder(raiz->esquerda);
        cout << raiz->dado << " ";
        preorder(raiz->direita);
      }
    }
```

D.
```
void preorder(noArvore *raiz) {
    if (raiz != NULL) {
        preorder(raiz->esquerda);
        preorder(raiz->direita);
        cout << raiz->dado << " ";
    }
  }
```

E.
```
void preorder(noArvore *raiz) {
      if (raiz != NULL) {
        preorder(raiz->direita);
        cout << raiz->dado << " ";
        preorder(raiz->esquerda);
      }
    }
```

1. Análise da questão

Percurso de árvores binárias

Árvores binárias são estruturas de dados compostas por nós interligados de forma hierárquica. Em uma árvore que não esteja completamente vazia, um desses nós é chamado de raiz, e novos nós são adicionados como nós filhos desse nó principal. Além disso, nas árvores binárias, existe a restrição de que cada nó pode ter zero, um ou, no máximo, dois nós filhos.

O percurso de uma árvore binária envolve visitar cada um dos seus nós e processá-los uma única vez. Diferentemente de uma estrutura de dados linear, na qual existe uma forma óbvia de fazer um percurso, em uma árvore binária temos várias possibilidades de caminhos.

No caso de uma árvore binária, cada uma das possibilidades de processamento é uma combinação de três possibilidades:

- processamento de dado nó atual,
- visita ao nó filho esquerdo (se ele existir);
- visita ao nó filho direito (se ele existir).

A ordem na qual essas operações podem ser feitas define os tipos de percursos. Nesse sentido, temos as seguintes opções:

- percurso em pré-ordem;
- percurso em ordem;
- percurso em pós-ordem.

O percurso em pré-ordem é aquele no qual primeiro se processa o nó visitado, depois se visita o nó filho esquerdo e, posteriormente, visita-se o nó filho direito. Outra possibilidade é o percurso em-ordem, no qual primeiro se visita o nó filho esquerdo, depois se processa o nó atual e, em seguida, visita-se o nó direito. Finalmente, o percurso em pós-ordem é aquele no qual se visita o nó filho esquerdo, depois o nó filho direito e, por último, processa-se o nó atual.

2. Resolução da questão

Os fragmentos de código das alternativas apresentam possíveis soluções para o problema de percurso da árvore binária com o uso da recursividade. Devemos lembrar que o percurso em pré-ordem envolve primeiro o processamento do nó atual. No caso do exercício, isso se refere simplesmente a mostrar na tela o conteúdo do número inteiro armazenado em raiz->dado, o que pode ser feito pela linha de código indicada a seguir.

```
cout << raiz->dado << " ";
```

Depois disso, devemos visitar o nó esquerdo, chamando a mesma função (*preorder*, no código) de forma recursiva. Finalmente, devemos visitar o nó direito. Isso pode ser feito pelas linhas de código indicada a seguir.

```
preorder(raiz->esquerda);
```

```
preorder(raiz->direita);
```

Contudo, temos mais um problema a ser evitado: em linguagens como C e C++, nas quais trabalhamos com ponteiros, não podemos acessar o conteúdo de um ponteiro nulo, ou seja, de um ponteiro que aponta para NULL. Se tentarmos fazer isso, teremos um erro de execução. Assim, devemos proteger o código. Para isso, verificamos, previamente, se o ponteiro para o nó a ser processado é nulo ou não, o que pode ser feito pela indicação a seguir.

```
if( raiz != NULL )
```

Com base nisso, identificamos o código da alternativa A como correto.

Alternativa correta: A.

3. Indicações bibliográficas

- AGUILAR, L. J. *Programação em C++: algoritmos, estruturas de dados e objetos.* 2. ed. Porto Alegre: AMGH, 2011.
- 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: MaGrawhill, 2008.
- VEERARAJAN, T. *Discrete Mathematics with graph theory and combinatorics*. New Delhi: Tata McGraw-Hill, 2007.

Questões 7, 8 e 9

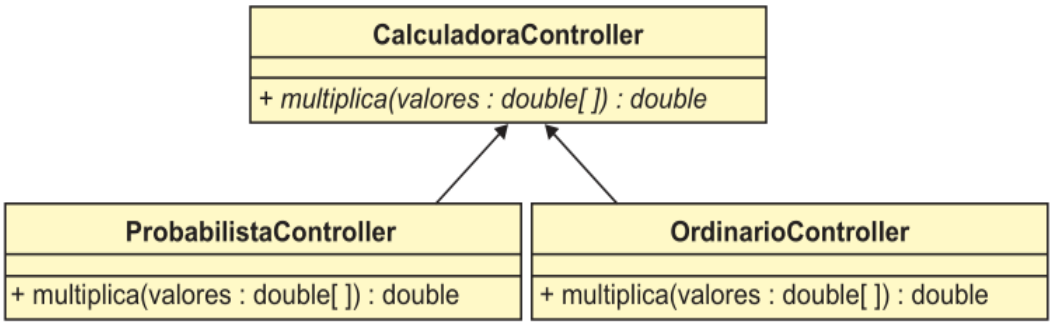
Questão 7.[7]

Para que se realize a multiplicação de probabilidades com maior facilidade, foi desenvolvida, utilizando-se o padrão MVC (*Model-View-Controller*), uma calculadora que pode ser configurada de dois modos: comum e probabilista. No primeiro, as multiplicações ocorrem de forma ordinária. No segundo, as multiplicações são feitas utilizando-se uma fórmula específica. A classe a seguir, escrita na linguagem Java, define o método `multiplica`. Ele é executado quando o botão da multiplicação é pressionado, e recebe como parâmetro os valores que o usuário deseja usar na operação, além de possuir uma variável de instância do tipo `CalculadoraController`. O método `alteraModo` é encarregado de alterar a instância para a qual essa variável faz referência conforme o modo selecionado pelo usuário. A instância interage com as partes do *model* apropriadas à solicitação realizada pelo usuário. Quando o usuário deseja usar o modo probabilista, a variável `modo` tem o valor 1, caso contrário, ela tem o valor 0.

```
public class Calculadora {
      private CalculadoraController c;
      private int modo;
      public double multiplica (double[] valores) {
            return c.multiplica (valores);
      }
      public void alteraModo () {
            if ( modo == 1)
                  c = new ProbabilistaController();
            else
            c = new OrdinarioController();
      }
}
```

O diagrama de classe que se segue mostra a hierarquia de *controllers* definida para a implementação da calculadora descrita. A hierarquia de classes de controle representa uma família de algoritmos intercambiáveis, por isso, o comportamento dos componentes da *view*, a cada instante, pode ser dinamicamente alterado, bastando trocar o tipo da instância referenciada pela variável `c`.

[7]Questão 26 – Enade 2017.



Considerando esse cenário, avalie as afirmativas.

I. A hierarquia exibida ilustra o uso do padrão de projetos *Strategy*.

II. O padrão composto MVC define a existência de um único *controller* e, portanto, a solução proposta não caracteriza o uso desse padrão.

III. O método `multiplica` da classe `Calculadora` chama o método `multiplica` de `CalculadoraController`, o que caracteriza uma conversão de interfaces e, portanto, o uso do padrão de projetos *Adapter*.

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.

Questão 8.[8]

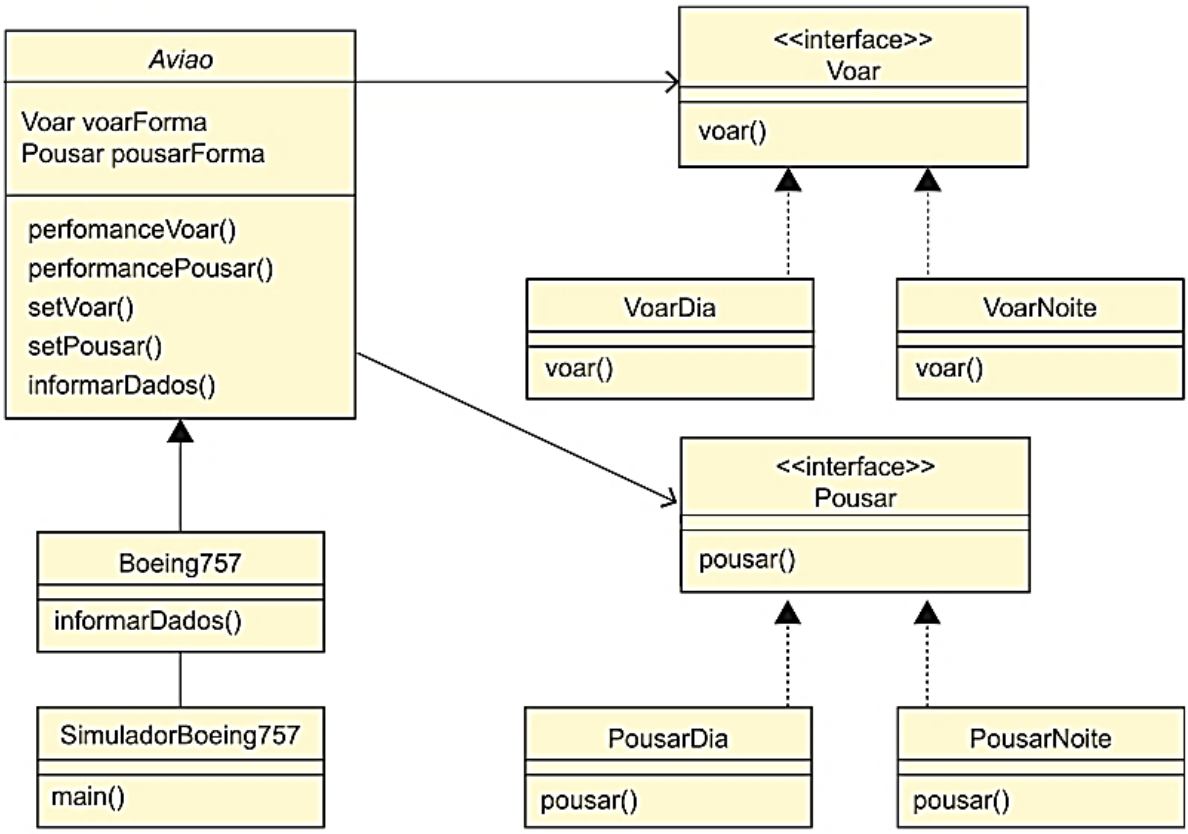
Leia o texto a seguir.

O diagrama de classe apresentado na figura a seguir mapeia um processo de um simulador de voo. Nesse diagrama, é utilizado o padrão de projeto Strategy, que define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. O padrão Strategy, que deixa o algoritmo variar independentemente dos clientes que o utilizam, aplica o princípio de projeto: "programar para uma interface, não para uma implementação".

LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo*. Porto Alegre: Bookman, 2007 (com adaptações).

Considere o diagrama de classe a seguir que descreve um simulador de voo.

[8]Questão 27 – Enade 2017.



Considere, ainda, o seguinte código em linguagem Java como implementação do simulador.

```
public interface Voar {
     public void voar();
}
public interface Pousar {
     public void pousar();
}
public class VoarDia implements Voar {
     public void voar() {
          System.out.println(“Voar durante o dia.”);
     }
}
public class VoarNoite implements Voar {
     public void voar() {
          System.out.println(“Voar durante a noite.”);
     }
}
public class PousarDia implements Pousar {
     public void pousar() {
          System.out.println(“Pousar durante o dia.”);
     }
}

public class PousarNoite implements Pousar {
     public void pousar() {
```

```
            System.out.println("Pousar durante a noite.");
      }
}
public abstract class Aviao {
      Voar voarForma;
      Pousar pousarForma;
      public Aviao() {}
      public abstract void informarDados();
      public void performanceVoar() {
                  voarForma.voar();
      }
      public void performancePousar() {
            pousarForma.pousar();
      }
      public void setVoar(Voar v) {
            voarForma = v;
      }
      public void setPousar(Pousar p) {
            pousarForma = p;
      }
}
public class Boeing757 extends Aviao {
      public Boeing757() {
            voarForma = new VoarDia();
            pousarForma = new PousarDia();
      }

      public void informarDados() {
            System.out.println("Informando dados de um Boeing 757.");
      }
}
public class SimuladorBoeing757 {
      public static void main(String[] args) {
            Boeing757 b757 = new Boeing757();
            b757.performanceVoar();
            b757.performancePousar();
            b757.setVoar(new VoarNoite());
            b757.setPousar(new PousarNoite());
            b757.performanceVoar();
            b757.performancePousar();
      }
}
```

Com base no diagrama de classe, no código em linguagem Java e no conceito do padrão de projeto *Strategy*, assinale a opção de correta.

A. A classe `Aviao` implementa as interfaces `Voar` e `Pousar`.

B. A classe `Boeing757` não aplica o princípio de herança, uma vez que esse princípio já foi aplicado pela classe `Aviao`, nas interfaces `Voar` e `Pousar`.

C. Por implementarem as interfaces `Voar` e `Pousar`, as classes `VoarDia`, `VoarNoite`, `PousarDia` e `PousarNoite` também são uma interface.

D. Na classe `SimuladorBoeing757`, os métodos `setVoar` e `setPousar` são os responsáveis por aplicar, respectivamente, as saídas "Voar durante o dia" e "Pousar durante o dia" para as saídas "Voar durante a noite" e "Pousar durante a noite".

E. Quando for executada, a classe `SimuladorBoeing757` apresentará um erro em tempo de execução porque os métodos `setVoar` e `setPousar` recebem respectivamente, como parâmetros, variáveis do tipo classe, o que não é permitido na orientação a objetos.

Questão 9.[9]

A área de desenvolvimento de *software* está se tornando cada vez mais complexa. Para lidar com essa realidade, os desenvolvedores contam com linguagens de programação baseadas no paradigma de orientação a objetos, cujos pilares são abstração, encapsulamento, herança e polimorfismo. No código a seguir, observa-se a implementação de classes relacionadas.

```
public abstract class Impressora {
      String nome;
      Impressora() {}
      Impressora(String n) {
            this.nome = n;
      }
      public void imprimir() {}
}
public class Laser extends Impressora {
      public Laser() {}
      public void imprimir() {
            System.out.println(“Imprimindo na Laser”);
      }
}
class Matricial extends Impressora {
      public Matricial () {}
      public void imprimir() {
            System.out.println(“Imprimindo na Matricial”);
      }
}
public class JatoDeTinta extends Impressora {
      public JatoDeTinta() {}
      public void imprimir(){
            System.out.println(“Imprimindo na Jato de tinta”);
      }
}
```

[9]Questão 30 – Enade 2017.

```
public class Main {
      public static void main(String args[]) {
            Impressora imp[] = new Impressora[3];
            imp[0] = new Laser();
            imp[1] = new JatoDeTinta();
            imp[2] = new Matricial();
            for(int i = imp.length - 1; i >= 0; i--){
                  imp[i].imprimir();
            }
      }
}
```

Com base nas informações do texto e no código apresentado, avalie as afirmativas a seguir.

I. A execução do código, via classe `Main`, resulta na seguinte saída:

Imprimindo na Laser

Imprimindo na Matricial

Imprimindo na Jato de tinta

II. O código faz uso da técnica denominada polimorfismo.

III. O código não será compilado, pois o vetor `imp` foi instanciado por meio da classe abstrata `Impressora`.

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.

1. Análise da questão

Orientação a objetos e padrões de projeto

As três questões apresentadas exploram conceitos fundamentais de orientação a objetos e de padrões de projeto.

Existem diferentes abordagens no que se refere à orientação a objetos, mas Booch et al. (2007) afirmam que "o conceito fundamental é que se deve modelar um sistema de software como uma coleção de objetos, sendo eles instâncias de uma classe dentro de uma hierarquia de classes".

De forma similar, outros autores, podem focar mais no aspecto de projeto: "o projeto orientado a objetos decompõe um sistema em objetos, sendo esses seus componentes básicos" (WIRFS-BROCK, WILKERSON e WIENER, 1990). Nesse mesmo texto, os autores focam nos mecanismos de abstração fornecidos pela orientação a objetos.

Page-Jones (2001) destaca que nove conceitos são fundamentais para a orientação a objetos, listados a seguir.

- Encapsulamento.
- Ocultação de informações e implementações.
- Retenção de estado.
- Identidade de objeto.
- Mensagens.
- Classes.
- Herança.
- Polimorfismo.
- Generalização.

Apenas o uso de uma linguagem orientada a objetos não garante a escrita de um software de qualidade. Projetistas experientes de software observaram que muitos problemas são recorrentes, ainda que ocorram variações, e podem ser resolvidos por estratégias similares. Além disso, algumas dessas estratégias parecem ser melhores do que outras: enquanto determinadas soluções podem apresentar problemas de manutenção no longo prazo, outras podem ser mais flexíveis e eficazes.

As melhores soluções deram origem aos chamados "padrões de projeto". Problemas similares foram catalogados, juntamente com as melhores estratégias para as suas soluções. Muitas dessas estratégias foram desenvolvidas com o objetivo de utilizar as ideias da orientação a objetos da melhor forma possível.

Em alguns livros, busca-se ensinar simultaneamente conceitos de orientação a objetos, UML e padrões de projeto. Outros livros têm uma abordagem mais diretamente focada nos padrões de projeto em si e funcionam como uma espécie de catálogo.

O padrão *Model-view-controller* (MVC) foi, de modo inicial, realizado na linguagem Smalltalk-80, justamente para o desenvolvimento de interfaces gráficas. A ideia básica é isolar três tipos de objetos:

- objetos relacionados à apresentação ou à interface gráfica, que correspondem ao tipo *View*;

- objetos relacionados à forma como a interface gráfica reage às ações do usuário, que correspondem ao tipo *Control*;
- objetos relacionados ao modelo da aplicação propriamente dita, que correspondem ao tipo *Model.*

Larman (2007) adiciona que o padrão MVC é uma aplicação de um princípio mais básico: o princípio da separação entre o modelo (relacionado ao domínio) e a visualização (relacionada à forma de apresentação). Esse conceito foi expandido ao longo do tempo e deu origem a um padrão arquitetural utilizado de maneira mais sofisticada em sistemas distribuídos, especialmente em sistemas *web*.

Gamma et al. (1995), no livro clássico sobre padrões de projeto, estabelecem quatro elementos essenciais da definição de um padrão de projeto:

- o nome do projeto (para referência);
- a descrição do problema (ou das classes de problemas) ao qual o padrão se aplica;
- o estabelecimento da estrutura da solução;
- a determinação das consequências da aplicação do padrão, ou seja, dos compromissos que fazemos ao adotarmos dado padrão.

Com relação à solução, devemos ter mente que ela não é uma descrição detalhada e concreta para um problema em específico, mas uma ideia abstrata de como resolver uma classe grande de problemas. Assim, as soluções apresentadas pelos padrões de projeto não focam em uma implementação específica, mas, sim, em guias gerais para a construção dessas soluções, propostas por projetistas ou programadores.

Um exemplo de padrão é o *Strategy* (estratégia). Esse padrão, também chamado de *policy* (política), é utilizado no caso em que existem diferentes algoritmos (ou políticas) que devem ser aplicados em diferentes contextos. A solução busca encapsular cada família de algoritmos em diferentes classes e estabelecer uma interface de acesso para eles.

2. Análise das afirmativas e das alternativas

Questão 7

I – Afirmativa correta.

JUSTIFICATIVA. A ideia do padrão de projetos *Strategy* é criar uma interface para acessar diferentes tipos de algoritmos. No caso do problema apresentado, isso envolve uma classe

para multiplicações comuns (chamada de ordinária no enunciado) e probabilistas, além da definição de uma interface única de acesso a essas classes.

II – Afirmativa incorreta.
JUSTIFICATIVA. É possível utilizar mais de um *controller* no padrão MVC.

III – Afirmativa incorreta.
JUSTIFICATIVA. Não ocorre conversão de interfaces nem é utilizado o padrão *Adapter*.

Alternativa correta: A.

Questão 8

A – Alternativa incorreta.
JUSTIFICATIVA. A classe Aviao é uma classe abstrata e não implementa as interfaces Voar e Pousar. O tipo de relação entre Aviao e Voar e entre Aviao e Pousar é chamado de associação.

B – Alternativa incorreta.
JUSTIFICATIVA. A classe Boeing757 utiliza o princípio de herança, o que pode ser visto tanto pelo diagrama de classes mostrado no enunciado quanto no código-fonte, uma vez que é utilizado a *keyword extends* na definição da classe Boeing757.

C – Alternativa incorreta.
JUSTIFICATIVA. Justamente por implementarem uma interface, essas classes não são uma interface.

D – Alternativa correta.
JUSTIFICATIVA. Os métodos setVoar e setPousar recebem objetos cujas classes implementam as interfaces de Voar e Pousar. A classe Boeing757, no seu construtor, inicializa as referências voarForma e pousarForma com objetos das classes VoarDia e PousarDia. Contudo, a classe SimuladorBoeing757 altera essas referências para objetos das classes VoarNoite e PousarNoite pelos métodos setVoar e setPousar. Esse comportamento é descrito na alternativa.

E – Alternativa correta.
JUSTIFICATIVA. Não existe um erro, pois esse é o mecanismo básico de passagem por referência da linguagem Java.

Questão 9

I – Afirmativa incorreta.
JUSTIFICATIVA. A saída do programa será a que segue.

Imprimindo na Matricial
Imprimindo na Jato de tinta
Imprimindo na Laser

Devemos observar que o laço de impressão começa da última posição do vetor (que corresponde a Matricial), indo até a primeira posição (que corresponde a Laser).

II – Afirmativa correta.
JUSTIFICATIVA. Podemos verificar que o método imprimir() é chamado para os objetos contidos no vetor imp. Observe que a classe Impressora é uma classe abstrata, e, mesmo que o método imprimir não tenha sido declarado como abstrato, ele está vazio. Isso não é exatamente a mesma coisa, mas, no contexto do enunciado, fica evidente que as classes que herdam a classe Impressora terão de implementar esses métodos (que é, de fato, o que ocorre). Assim, podemos dizer que o programa do enunciado utiliza polimorfismo.

III – Afirmativa incorreta.
JUSTIFICATIVA. O vetor imp é um vetor de referências para as diversas impressoras. Assim, cada elemento desse vetor deve conter um objeto que apresenta uma implementação do método imprimir(), mas o vetor em si é um vetor de referências.

Alternativa correta: A.

2. Indicações bibliográficas

- BOOCH, G. et al. *Object-oriented analysis and design with applications.* 3. ed. Boston: Pearson Prentice Hall, 2007.
- GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software.* Boston: Addison-Wesley, 1995.
- LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo*. Porto Alegre: Bookman, 2007.
- PAGE-JONES, M. *Fundamentos do desenho orientado a objeto com UML.* São Paulo: Pearson, 2001.
- WIRFS-BROCK, R.; WILKERSON, B.; WIENER, L. *Designing object-oriented software.* Englewoods Cliffs: Prentice-Hall, 1990.

Questão 10

Questão 10.[10]

Uma pesquisa está sendo realizada para identificar a renda média de um grupo de profissionais, em função de algumas variáveis de interesse. As variáveis utilizadas são "possui ensino superior", "possui pós-graduação" e "possui pelo menos 5 anos de experiência em sua área de atuação". Os conjuntos de pessoas representados por essas variáveis são denotados A, B e C, respectivamente. Dado que as variáveis não são mutuamente exclusivas, uma pessoa pode pertencer a mais de um conjunto. As expressões a seguir representam subconjuntos para os quais se deseja obter a renda médias das pessoas a eles pertencentes, tarefa que será realizada posteriormente.

Expressão 1	$x \in A \wedge x \notin (B \cup C)$
Expressão 2	$x \in A \ - \ (B \cup C)$
Expressão 3	$x \in A \wedge x \in B \wedge x \in C$

Considere que, em função das expressões dadas, o seguinte programa em linguagem de programação C foi escrito, para que posteriormente se realize os cálculos desejados. Considere, ainda, que A, B e C são estruturas de dados apropriadamente definidas e que contém as pessoas exatamente como especificado. A função `contem` opera sobre elas e devolve um valor booleano, indicando se a pessoa apontada por p existe ou não na estrutura especificada.

```
void determina_renda (pessoa *p) {
 /* condição 1 */
 if (contem (A, p) && !contem (B, p) && !contem (C, p)) {
 //renda será determinada posteriormente
 }
 /* condição 2 */
 else if (contem (A, p) || contem (B, p) || contem (C, p)) {
 //renda será determinada posteriormente
 }
}
```

Com base nessa situação e nas informações apresentadas, assinale a opção correta.

A. A condição 2 do programa é equivalente à expressão 3.

B. Pessoas com pós-graduação pertencem ao conjunto definido pela expressão 2.

C. A condição 1 do programa é verdadeira para todas as pessoas com ensino superior.

[10]Questão 29 – Enade 2017.

D. A condição 1 do programa e as expressões 1 e 2, que representam o mesmo subconjunto, são equivalentes.
E. A condição 2 do programa define o conjunto de pessoas que tem ensino superior, pós-graduação e pelo menos cinco anos de experiência em sua área de atuação.

1. Análise da questão

Operadores lógicos na linguagem C

A questão aborda, basicamente, o uso de operadores lógicos na linguagem de programação C.

Até o padrão C99, a linguagem C não apresentava um tipo booleano específico, ou seja, um tipo que correspondesse aos valores de verdadeiro (V) ou de falso (F). Assim, a linguagem C utilizava a seguinte estratégia:

- o zero (0) representa uma condição falsa;
- qualquer valor inteiro diferente de zero equivale a uma condição verdadeira.

Depois, com o padrão C99, surgiu o arquivo de cabeçalho "stdbool.h", que definiu algumas macros, como:

- bool (utilizado como um tipo);
- true (equivalente ao número inteiro um);
- false (equivalente ao número zero).

A linguagem C também tem os chamados operadores lógicos, que correspondem às seguintes operações lógicas típicas:

- o símbolo || corresponde ao operador lógico OU;
- o símbolo && corresponde ao operador lógico E;
- o símbolo ! corresponde ao operador lógico NÃO (negação lógica).

Esses operadores foram criados para permitir combinações de expressões, de forma similar ao que é feito na lógica matemática ou em circuitos digitais.

É importante diferenciarmos os operadores lógicos dos chamados operadores bit a bit (*bitwise operators*). No último caso, a linguagem C permite que os bits de uma variável sejam manipulados de forma individual. Assim, em vez de considerar uma variável como um valor lógico único (verdadeiro ou falso), nos operadores bit a bit, as variáveis envolvidas são consideradas como cadeias de valores binários, e as operações são feitas em cada um dos

bits isoladamente. O resultado também é uma cadeia de bits, correspondente ao resultado da operação em cada um dos bits isoladamente. Os principais operadores binários são:

- OU bit a bit, representado pelo símbolo |;
- E bit a bit, representado pelo símbolo &;
- negação, representada pelo símbolo ~;
- deslocamento de bits para a esquerda, representado pelo símbolo <<;
- deslocamento de bits para a direta, representado pelo símbolo >>.

2. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. A condição 2 do programa corresponde a um OU lógico entre os conjuntos A, B e C. A expressão 3 corresponde a um E lógico entre os conjuntos A, B e C.

B – Alternativa incorreta.
JUSTIFICATIVA. O conjunto definido pela expressão 2 exclui as pessoas do conjunto B, que são justamente aquelas que têm pós-graduação.

C – Alternativa incorreta.
JUSTIFICATIVA. Se uma pessoa tiver ensino superior (ou seja, pertence ao conjunto A), e também tiver pós-graduação (pertence simultaneamente aos conjuntos A e B), a primeira condição do programa será falsa, pois a expressão !contem(B, P) será avaliada como falsa. Assim, não é correto afirmar que a condição 1 do programa é verdadeira para todas as pessoas do conjunto A (que são aquelas com ensino superior).

D – Alternativa correta.
JUSTIFICATIVA. Tanto a expressão 1 quanto a expressão 2 representam o mesmo conjunto: das pessoas que têm apenas ensino superior, sem pós-graduação nem experiência de 5 anos na área de atuação. O programa, na condição 1, implementa o conjunto x pertence a A e x não pertence a B e x não pertence a C, que também é equivalente às expressões 1 e 2.

E – Alternativa incorreta.
JUSTIFICATIVA. Como a condição 2 do programa corresponde a um OU lógico entre os conjuntos A, B e C, ela representa qualquer pessoa que pertença a esses conjuntos,

simultaneamente ou não. Assim, se uma pessoa tiver apenas ensino superior (pertence apenas ao conjunto A), a condição vai retornar verdadeira.

3. Indicações bibliográficas

- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. *Introdução a estruturas de dados*. Rio de Janeiro: Campus, 2004.
- SENNE, E. L. F. *Primeiro curso de programação em C*. 3. ed. Florianópolis: Visual Books, 2006.

ÍNDICE REMISSIVO