UNIP
UNIVERSIDADE PAULISTA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTITUCIONAL ESPECÍFICO

TOMO 1

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 1

Questão 1

Questão 1.[1]

Um analista foi contratado para desenvolver um sistema de pesquisa de DVDs em lojas virtuais. O sistema deverá solicitar ao usuário um título de DVD, que será usado para realizar a pesquisa nas bases de dados das lojas conveniadas. Ao detectar a disponibilidade do DVD solicitado, o sistema armazenará temporariamente os dados das lojas (nome, preço, data prevista para entrega do produto) e exibirá as informações ordenadas por preço. Após analisar as informações, o cliente poderá efetuar a compra. O contratante deverá testar algumas operações do sistema antes de ele ser finalizado. Há tempo suficiente para que o analista atenda a essa solicitação e efetue eventuais modificações exigidas pelo contratante.

Com relação a essa situação, julgue os itens a seguir quanto ao modelo de ciclo de vida.

I. O entendimento do sistema como um todo e a execução sequencial das fases sem retorno produzem um sistema que pode ser validado pelo contratante.
II. A elaboração do protótipo pode ser utilizada para resolver dúvidas de comunicação, o que aumenta os riscos de inclusão de novas funcionalidades não prioritárias.
III. A definição das restrições deve ser a segunda fase a ser realizada no desenvolvimento do projeto, correspondendo à etapa de engenharia.
IV. Um processo iterativo permite que versões progressivas mais completas do sistema sejam construídas e avaliadas.

São corretos apenas os itens

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

1. Introdução teórica

Restrições de projeto

As restrições de projeto são requisitos de sistema capturados durante a fase de requisitos, nas etapas de concepção e de elaboração do sistema, conforme o processo RUP *(Rational Unified Process)*.

[1]Questão 11 – Enade 2008.

As restrições de projeto não podem ser confundidas com os objetivos do sistema, embora estejam diretamente relacionados. Um objetivo pode não ser alcançado devido a uma restrição que o limita ou que o impeça. Vejamos alguns exemplos:

- é objetivo do sistema permitir o acesso de fornecedores externos via internet, mas restrições de segurança não o permitem;
- o sistema tem de ler dados de um leitor de cartão de um modelo específico, mas o fabricante não fornece o *driver* necessário para o sistema operacional no qual o sistema será executado.

O protótipo é uma simplificação do sistema a ser desenvolvido, feito para permitir ao usuário antever, verificar, experimentar e validar o sistema futuro antes que ele seja realmente construído. Pode ser usado para:

- a demonstração de uma visão do sistema aos usuários;
- a validação dos requisitos;
- a clarificação de requisitos vagos, imprecisos ou indefinidos;
- a comunicação entre os membros da equipe e usuários.

Um processo de engenharia de *software* é dividido em fases, sendo que cada uma delas tem papel fundamental para que o objetivo seja cumprido. Em cada fase, são recomendadas as tarefas a serem distribuídas entre os vários integrantes das equipes.

Um processo iterativo e incremental é oposto ao antigo desenvolvimento sequencial. Cada passagem completa pelo processo é uma iteração, e cada nova iteração deve adicionar um ou vários novos incrementos. Desse modo, ao final de todas as iterações o sistema estará pronto.

Algumas vantagens do processo iterativo em relação ao processo sequencial são:

- redução dos riscos de se fazer toda uma etapa e não mais retornar a ela;
- aceleração no tempo de desenvolvimento porque serão trabalhados escopos menores e claros;
- possibilidade de sofrer menor impacto devido às constantes alterações e atualizações pedidas pelos usuários, facilitando a adaptação e mudança dos requisitos.

2. Indicações bibliográficas

- KRUCHTEN, P. *Introdução ao RUP.* São Paulo: Ciência Moderna, 2003.
- PRESSMAN, R. S. *Engenharia de software.* 6. ed. São Paulo: McGraw-Hill, 2006.
- SOMMERVILLE, I. *Engenharia de software.* 8. ed. São Paulo: Addison Wesley, 2007.

3. Análise dos itens

I – Item incorreto.
JUSTIFICATIVA. O item I é incorreto porque a execução sequencial das fases, sem retorno, não capturará as alterações e correções identificadas nas fases posteriores, sejam elas originadas internamente pelos usuários ou externamente por mudanças em legislações e regras.

II – Item correto.
JUSTIFICATIVA. O item II é correto porque o protótipo facilita a resolução de dúvidas de comunicação, mas, também, dá ao usuário a oportunidade de criar novas necessidades (prioritárias ou não) porque ele tem uma antevisão do que será o sistema.

III – Item incorreto.
JUSTIFICATIVA. O item III é incorreto porque na etapa de engenharia o objetivo é ter uma visão global do sistema como um todo (incluindo *hardware*, *software*, equipamentos e pessoas envolvidas). O detalhamento das restrições é feito em etapas posteriores.

IV – Item correto.
JUSTIFICATIVA. O item IV é correto porque o processo iterativo permite versões progressivas mais completas por meio de incrementos. A cada iteração, uma nova versão produtiva é completada e se aproxima mais do objetivo final de desenvolvimento do produto.

Alternativa correta: D.

Questão 2

Questão 2.[2]

Uma pizzaria fez uma ampliação de suas instalações e o gerente aproveitou para melhorar o sistema informatizado, que era limitado e não atendia a todas as funções necessárias. O gerente, então, contratou uma empresa para ampliar o software. No desenvolvimento do novo sistema, a empresa aproveitou partes do sistema antigo e estendeu os componentes de maneira a usar código validado, acrescentando as novas funções solicitadas.

Que conceito de orientação a objetos está descrito na situação hipotética acima?

A. Sobrecarga.

B. Herança.

C. Sobreposição.

D. Abstração.

E. Mensagem.

1. Introdução teórica

Sobrecarga, herança, hierarquia, sobreposição, abstração e mensagem

Sobrecarga é a provisão de mais de uma versão para um mesmo método. A diferenciação entre as versões é feita por assinaturas dos métodos, isto é, na lista de parâmetros que o método possui. Isso pode ser feito tanto na quantidade de parâmetros como no tipo de parâmetro informado (exemplos: *string, integer, double e float*). Os construtores também podem ser sobrecarregados.

Segue um exemplo de sobrecarga de construtor.

```
Point p1 = new Point(); // Construtor padrão
Point p2 = new Point(1,2); // Construtor sobrecarregado
```

Segue um exemplo de sobrecarga de método, alterando o tipo de dado.

```
int a = Math.abs(-10);
double b = Math.abs(-2.3);
```

[2]Questão 12 – Enade 2008.

Herança é o mecanismo utilizado em orientação a objeto para que uma classe (subclasse) herde as propriedades de outra classe (superclasse), isto é, seus atributos e seus métodos. Com a herança, podemos gerar uma hierarquia de classes na qual as no nível mais baixo herdam atributos e métodos da classe acima (suas características e seus comportamentos). A superclasse é a classe pai e a subclasse é a classe filha numa hierarquia de classes.

Segue um exemplo de herança.

Animal -> Mamífero -> Cachorro

O cachorro é um mamífero e também um animal. A classe Cachorro é uma subclasse de Mamífero. Mamífero é a superclasse de Cachorro. Cachorro é subclasse de Animal. Animal é superclasse tanto de Mamífero quanto de Cachorro.

A hierarquia é unidirecional, isto é, podemos falar que o cachorro é um mamífero, mas não podemos falar que o mamífero é um cachorro (um mamífero poderia ser uma baleia).

A sobreposição ocorre quando um método numa subclasse é declarado exatamente com o mesmo nome e lista de argumentos do método na superclasse, alterando ou não o código em seu interior.

Segue um exemplo de sobreposição.

```
class Mamifero {
    public void comer() {
     system.out.println("Mamifero comendo");
          }
}
class Cachorro extends Mamifero {
     public void comer() {
         system.out.println("Cachorro comendo");
         }
}
```

Abstração é a limitação de um amplo universo em um domínio específico. Com ela, focamos nos objetos, nas ações e nas prioridades que são relevantes para a aplicação específica, ignorando os demais pontos que são irrelevantes.

Segue um exemplo de abstração: em um domínio de determinado problema, podemos ter o conceito Veículo, que poderia significar grande quantidade de elementos (avião, carro, barco, submarino, disco voador, elevador, trem, metrô ou *jet-ski*). A abstração

reduz o universo ao qual se relaciona com a aplicação sendo desenvolvida. Por exemplo, em uma aplicação de Aluguel de Carros não usaríamos os conceitos de submarino e avião, dentre outros.

Mensagem é uma solicitação feita de um objeto para outro. Os objetos se comunicam por mensagens que geralmente são uma chamada de um método em algum dos objetos envolvidos no processo.

Segue um exemplo de mensagem: um objeto FormulárioCliente solicita uma consulta ao objeto PessoaFisica através do método ConsultarCliente(cpf).

2. Indicações bibliográficas

- GILLEANES, T. A. G. *UML 2 - uma abordagem prática*. São Paulo: Novatec, 2009.
- LARMAN, C. *Utilizando UML e padrões*. Porto Alegre: Bookman, 2007.
- SILVA, R. P. *Como modelar com UML 2*. Florianópolis: Visual Books, 2009.

3. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. A sobrecarga não permite o reaproveitamento ou a extensão de partes do sistema antigo, pois ela simplesmente gera novas versões dos métodos com assinaturas diferentes. Esses códigos terão de ser novamente testados e validados.

B – Alternativa correta.
JUSTIFICATIVA. A herança aproveita tudo que foi desenvolvido e aprovado na superclasse, possibilitando o uso nas subclasses como código já testado e validado.

C – Alternativa incorreta.
JUSTIFICATIVA. A sobreposição não aproveita partes antigas, mas as substitui. Esse novo código também terá de ser testado e validado.

D – Alternativa incorreta.
JUSTIFICATIVA. Abstração é um conceito que nada tem a ver com o reaproveitamento de código.

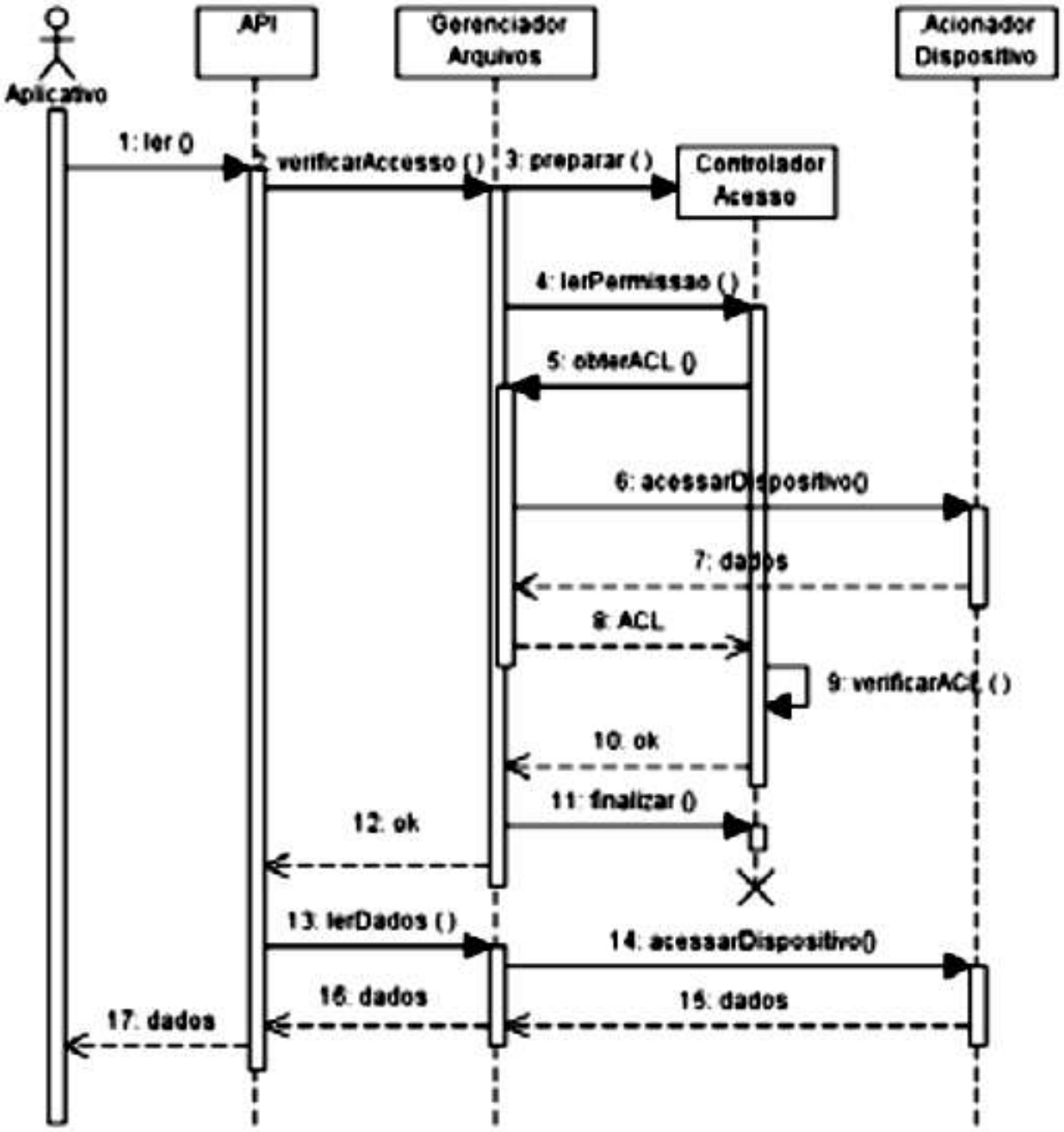
E – Alternativa incorreta.

JUSTIFICATIVA. A mensagem é um conceito que se refere à comunicação entre objetos, nada tendo a ver com o reaproveitamento de código em componentes já desenvolvidos.

Questão 3

Questão 3.[3]

Observe o diagrama a seguir.



Com relação ao diagrama acima, assinale a opção correta.

A. Para economizar tempo e memória, as mensagens de retorno 7: dados e 15: dados poderiam ser mescladas em uma única mensagem.

B. O objeto Controlador Acesso utiliza uma estrutura de repetição para verificar os atributos de acesso a um arquivo.

C. A mensagem 5: obterACL( ) pode levar à repetição da chamada 4: lerPermissao( ).

[3]Questão 13 – Enade 2008.

D. Sempre que um Aplicativo fizer uma leitura, será construído e destruído um objeto ControladorAcesso.
E. A mensagem 3: preparar( ) ocorre simultaneamente (em paralelo) à mensagem 4: lerPermissao( ).

1. Introdução teórica

Diagrama de Sequência da UML

A questão refere-se à correta leitura de um Diagrama de Sequência da UML. Trata-se de um diagrama comportamental direcionado à ordem temporal em que as mensagens são trocadas entre os objetos. Geralmente é formado com insumos vindos dos Diagramas de Caso de Uso e de Classes. O Diagrama de Caso de Uso fornece os atores e o Diagrama de Classes fornece os objetos envolvidos. Desse modo, determina a ordem em que os eventos ocorrem e as mensagens que são enviadas (métodos chamados), estabelecendo a interação entre os objetos.

O Diagrama de Sequência origina-se das funcionalidades identificadas no Diagrama de Caso de Uso. Um Caso de Uso, por sua vez, refere-se a um comportamento do sistema e descreve a funcionalidade executada por um ator. Baseia-se, também, no Diagrama de Classes para retirar dele os objetos identificados até o momento. Mas, além disso, o Diagrama de Sequência serve para validar e refinar os dois diagramas, podendo levar a complementações ou modificações.

A leitura do Diagrama de Sequência sempre é feita de cima para baixo e da esquerda para a direita. Adicionalmente, colocam-se números sequenciais para ordenar as mensagens. Portanto, no diagrama em questão, inicia-se a leitura em *1. Ler()*, *2. verificarAcesso()*, *3. preparar()* e assim por diante.

Os atores de um Diagrama de Sequência são os mesmos descritos nos Casos de Uso. São desenhados como bonecos magros. Os objetos são representados como quadrados ou estereótipos, mecanismos de extensão da UML, não usados no diagrama em questão.

Cada ator ou objeto tem uma linha de vida, que é a linha tracejada. Esta se refere geralmente a uma instância e ao tempo de vida em que o objeto ou ator existem no processo. A linha de vida é interrompida com um "X", como acontece com o objeto "Controlador Acesso", que é destruído nesse momento.

Quando um objeto está ativamente participando do processo ele tem o foco de controle. É a linha retangular fina que fica acima da linha tracejada.

Os objetos que são representados no topo do diagrama existem desde o início do processo. Os objetos que estão abaixo no diagrama são criados, e muitas vezes destruídos, durante o processo.

Um objeto pode enviar mensagem para si mesmo. É uma autochamada, como ocorre na mensagem *9. verificarACL()*.

2. Indicações bibliográficas

- GILLEANES, T. A. G*. UML 2 - Uma abordagem prática*. São Paulo: Novatec, 2009.
- LARMAN, C. *Utilizando UML e padrões*. Porto Alegre: Bookman, 2007.
- SILVA, R. P. *Como modelar com UML 2*. Florianópolis: Visual Books, 2009.

3. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. As mensagens 7 e 15 não são diferentes. São as mesmas mensagens disparadas em momentos distintos. É recomendado que mensagens diferentes tenham nomes diferentes para clareza do diagrama. Uma suposta economia de tempo e memória é irrelevante nesse caso.

B – Alternativa incorreta.

JUSTIFICATIVA. As estruturas de repetição no Diagrama de Sequência são obtidas por meio de fragmentos combinados: *loop* (para fazer o laço repetitivo) e *break* (para interromper a execução). O objeto "Controlador Acesso" não utiliza estrutura de repetição, apenas uma autochamada na mensagem *9. verificarACL()*. Na figura 1 há um exemplo de *loop* com *break.*

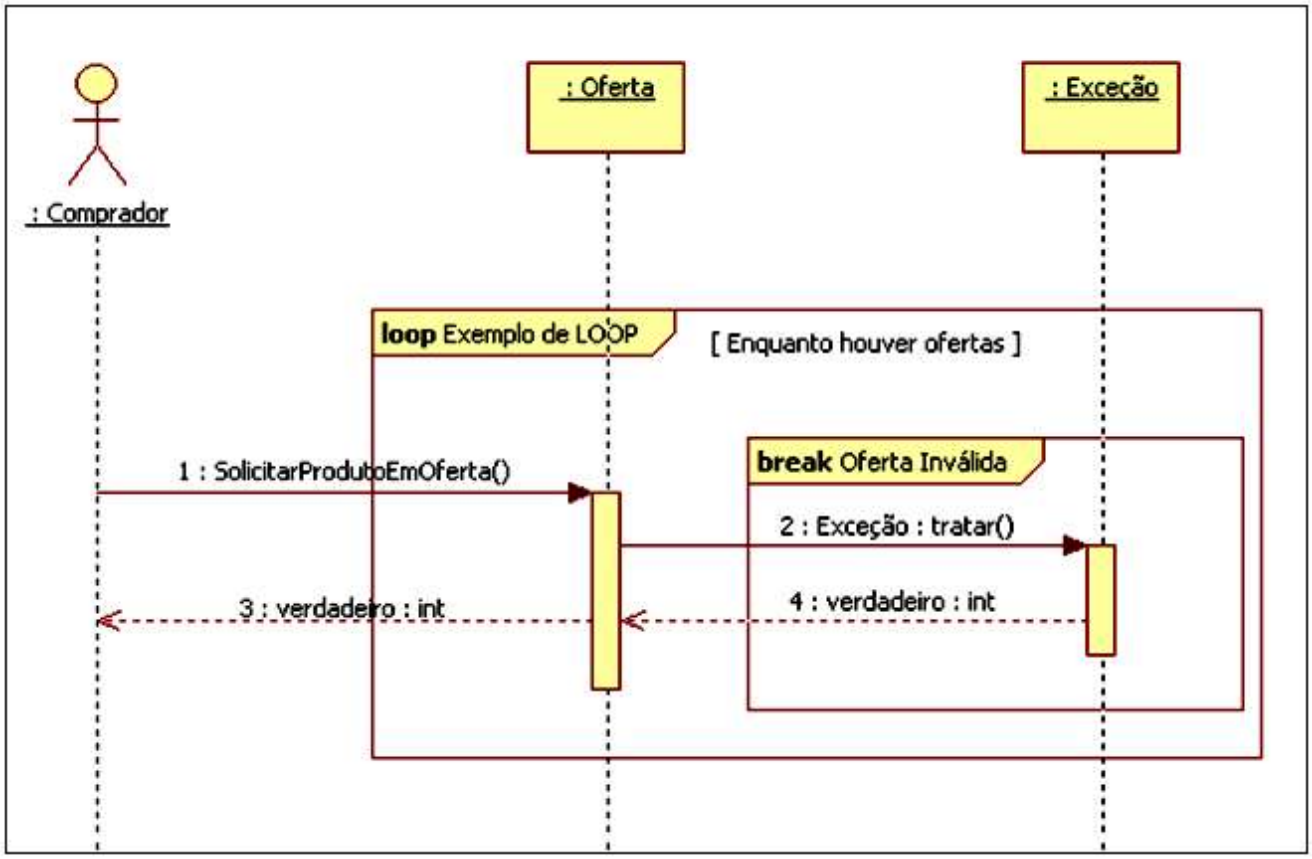


Figura 1. Exemplo de *loop* com *break.*

C – Alternativa incorreta.

JUSTIFICATIVA. A sequência de leitura do Diagrama de Sequência é de cima para baixo e da esquerda para a direita. Portanto, após a mensagem *5. obterACL()*, o objeto envia a mensagem para *6. acessarDispositivo()*, jamais para *4. lerPermissao()*. Após a mensagem 6, o objeto "Acionador Dispositivo" envia a mensagem *7. dados*, e assim por diante até o final do diagrama.

D – Alternativa correta.

JUSTIFICATIVA. As mensagens *1. Ler()*, *2. verificarAcesso()* e *3.preparar()* são executadas sequencialmente, culminando na criação do objeto "Controlador Acesso". Isso acontece sempre e incondicionalmente até a destruição do objeto após a mensagem *11. Finalizar()*. Não existe nenhuma interrupção entre a mensagem *1. Ler()* e a mensagem *11. Finalizar()* que poderia impedir ou evitar a criação e destruição do objeto.

E – Alternativa incorreta.

JUSTIFICATIVA. As mensagens *3.preparar()* e *4. lerPermissao()* são sequenciais. Para que uma mensagem seja paralela à outra, no diagrama de sequência é preciso utilizar o fragmento combinado par, conforme ilustrado na figura 2.

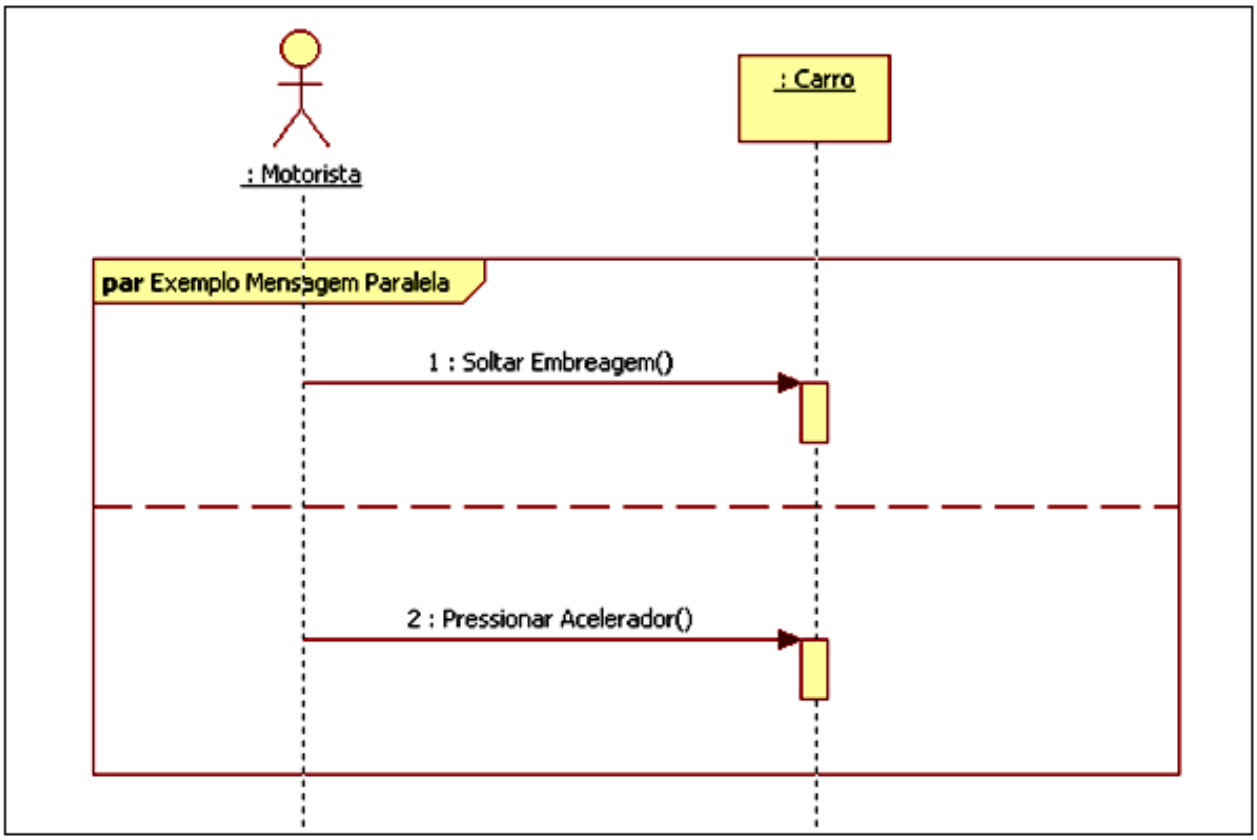


Figura 2. Fragmento combinado par.

Questão 4

Questão 4.[4]

Leia o código a seguir.

```
SUBROTINA xis()
   i = 0
   ENQUANTO (i < Gn) FACA
        i = i + 1
        SE (calc(i) <= Gn) ENTAO
             f1(i)
        SENAO
             f2(i)
        FIM  SE
   FIM  ENQUANTO
   Imprima("ok")
FIM SUBROTINA
```

Com relação ao código acima, considere que:

- a variável i é local e a variável Gn é global;
- não há nenhum tipo de documentação ou código fonte além do mostrado;
- a subrotina xis() faz parte de um programa;
- o critério de aceitação do teste é: a subrotina xis() não entra em laço infinito.

Na situação apresentada, é correto

I. aplicar testes de caixa-branca às rotinas calc(), f1() e f2() e, em seguida, usar o resultado para fazer um teste de mesa da subrotina xis().

II. aplicar testes de caixa-preta que forcem a chamada a xis() e depois medir a porcentagem de sucesso da subrotina xis().

III. aplicar testes de caixa-preta isoladamente ao código objeto das subrotinas calc(), f1() e f2() antes de aplicar um teste que envolva a sub-rotina xis().

Assinale a opção correta.

A. Apenas um item está certo.

B. Apenas os itens I e II estão certos.

C. Apenas os itens I e III estão certos.

D. Apenas os itens II e III estão certos.

E. Todos os itens estão certos.

[4]Questão 14 – Enade 2008.

1. Introdução teórica

Teste caixa-branca

Um teste caixa-branca é aquele no qual a estrutura interna do *software* (código) é analisada. Baseia-se nos caminhos internos, na estrutura e na implementação do produto. Requer conhecimento do código do produto em um teste para ser aplicado.

Para implementação do teste caixa-branca, seguem-se os passos abaixo.

- Analisar a implementação do produto a ser testado.
- Escolher os caminhos a testar.
- Selecionar valores de entrada que percorram estes caminhos.
- Determinar as saídas esperadas conforme as entradas escolhidas.
- Construir os casos de teste.
- Executar os casos de teste.
- Comparar as saídas encontradas com as saídas esperadas.
- Gerar um relatório para avaliar o resultado dos testes.

Um teste caixa-preta é aquele no qual um programa é tomado como uma entidade fechada e sua estrutura interna não é levada em conta: somente as especificações do programa e os requisitos do software são considerados no momento da execução dos testes. O teste funcional é um exemplo de um teste caixa-preta.

O teste tem esse nome porque considera o produto como uma caixa lacrada, da qual somente se conhecem a entrada e a saída, ou seja, não possuímos nenhum conhecimento de como o produto está construído internamente.

Para implementação do teste caixa-preta seguem-se os passos abaixo.

- Analisar a especificação dos requisitos.
- Escolher entradas válidas, conforme a especificação, para determinar o correto comportamento do produto.
- Escolher entradas inválidas para verificar se são manipuladas corretamente.
- Determinar as saídas esperadas conforme as entradas escolhidas.
- Construir os casos de teste.
- Executar os casos de teste.
- Comparar as saídas obtidas com as saídas esperadas.
- Gerar um relatório para avaliar o resultado dos testes.

O teste caixa-preta:

- utiliza a especificação funcional do produto como fonte primária de informação;
- pode ser usado em todas as fases de testes;
- é a técnica mais comumente utilizada.

  Para evitar o teste exaustivo, devemos selecionar diferentes critérios de teste.

2. Indicações bibliográficas

- BASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. *Base de conhecimento em teste de software.* São Paulo: Martins Fontes, 2008.
- MOLINARI, L. *Testes funcionais de software.* Florianópolis: Visual Books, 2008.
- PRESSMAN, R. S. *Engenharia de software.* 6. ed. São Paulo: McGraw-Hill, 2006.
- SOMMERVILLE, I. *Engenharia de software.* 8. ed. São Paulo: Addison Wesley, 2007.

3. Análise dos itens

I – Item incorreto.

JUSTIFICATIVA. Não é possível fazer o teste caixa-branca das rotinas *calc(), f1()* e *f2()* porque somente foi fornecido o código da subrotina *Xis().*

II – Item correto.

JUSTIFICATIVA. *Gn* é uma variável global e não temos ideia do seu valor. O mesmo acontece com o funcionamento das funções *calc(), f1()* e *f2()*. Só é possível descobrir executando a subrotina *Xis()* (caixa-preta). A subrotina *Xis()* vai rodar a primeira vez com i=0, depois i=2, i=3 etc., até que o valor de i e Gn sejam iguais, ou i alcance o seu limite superior máximo. Portanto, é um teste válido e possível de ser feito.

III – Item incorreto.

JUSTIFICATIVA. O teste caixa-preta é feito sobre um arquivo executável, página *web* ou biblioteca (jar ou DLL, por exemplo). Código objeto é um tipo de arquivo intermediário entre o código-fonte e o executável, o que torna este item falso, pois não pode ser testado.

Alternativa correta: A.

Questão 5

Questão 5.[5]

O conceito de máquina virtual (MV) foi usado, na década de 1970, no sistema operacional IBM System 370. Atualmente, centros de dados (datacenters) usam MVs para migrar tarefas entre servidores conectados em rede e, assim, equilibrar carga de processamento. Além disso, plataformas atuais de desenvolvimento de *software* empregam MVs (Java,.NET). Uma MV pode ser construída para emular um processador ou um computador completo. Um código desenvolvido para uma máquina real pode ser executado de forma transparente em uma MV.

Com relação a essas informações, assinale a opção correta.

A. O conceito de transparência mencionado indica que a MV permite que um aplicativo acesse diretamente o *hardware* da máquina.
B. Uma das vantagens mais significativas de uma MV é a economia de carga de CPU e de memória RAM na execução de um aplicativo.
C. Uma MV oferece maior controle de segurança, uma vez que aplicativos são executados em um ambiente controlado.
D. Para emular uma CPU *dual-core,* uma MV deve ser instalada e executada em um computador com CPU *dual-core.*
E. Como uma MV não é uma máquina real, um sistema operacional nela executado fica automaticamente imune a vírus.

1. Introdução teórica

Máquina virtual

Uma MV (máquina virtual) é um *software* que permite a execução de sistemas operacionais e aplicativos e é um ambiente operacional completo, que se comporta como se fosse um computador independente. É como se tivéssemos um computador dentro do outro, sendo este último "criado" por *softwares*. Ela se comporta exatamente como se fosse uma máquina física, contendo CPU, memória e HD próprios.

A máquina virtual é composta totalmente por *softwares*, não tendo componentes de *hardware*. Por isso, com a virtualização, um servidor pode manter vários sistemas operacionais em uso. A restrição para o número de máquinas virtuais possíveis é definida

[5]Questão 15 – Enade 2008.

pelos limites físicos de memória, espaço em disco e poder de processamento da máquina hospedeira.

Há várias vantagens no uso de máquinas virtuais, como as citadas abaixo.

- Elas ficam isoladas umas das outras, como se estivessem fisicamente separadas e geram um ambiente de computação completo para cada uma.
- Independência do *hardware*, isto é, no mesmo servidor físico podemos executar diferentes sistemas operacionais, como Windows ou Linux.
- Aumento da segurança, porque cada máquina virtual é independente da outra, assim podemos ter diferentes requisitos de segurança em diferentes ambientes virtualizados.
- Aumento da confiabilidade e disponibilidade, porque a parada de um ambiente não interrompe os demais.
- Redução do custo, porque podemos trocar vários servidores menores por um mais poderoso, que virtualiza cada um dos servidores menores substituídos.
- Melhoria do suporte a aplicações legadas. Ao migrar um sistema operacional dentro de uma empresa é possível manter o sistema operacional antigo funcionando em uma máquina virtual.

Temos, porém, algumas desvantagens no uso de máquinas virtuais, como as descritas a seguir.

- Aumento do gerenciamento. Como os ambientes se tornam mais complexos e heterogêneos, é necessária uma quantidade de produtos que permitam instanciar, monitorar, configurar e salvar os ambientes virtuais criados;
- Diminuição do Desempenho. A introdução de uma camada extra de *software* entre o sistema operacional e o *hardware* gera um aumento na carga do processamento. A adição de mais máquinas virtuais degrada o desempenho do *hardware* como um todo. É um desafio identificar quantos ambientes virtuais podem ser incluídos em uma determinada infraestrutura sem degradar a qualidade do serviço prestado.

Pesquisa feita pela *Forrest Consulting* em 2010, comentada na revista *Computer Word*, mostra que essa tecnologia trouxe ganhos para as empresas, mas também novos desafios, como a necessidade de gerenciamento, automação e a exigência de incluir a infraestrutura virtualizada dentro de um programa geral de gerenciamento de serviços de TI. A virtualização, no entanto, é uma técnica cada vez mais presente. O aumento do poder computacional gerou recursos ociosos que podem ser aproveitados por essa tecnologia.

2. Indicações bibliográficas

- MAIA, L. P. *Arquitetura de sistemas operacionais.* São Paulo: LTC, 2007.
- TANENBAUM, A. S. *Sistemas operacionais: projeto e implantação.* 3. ed. São Paulo: Artmed, 2008.
- TOSCANI, S. *Sistemas operacionais*. São Paulo: Artmed, 2010.
- Revista *Computer World* de 23.03.2010, publicada em <http://computerworld.uol.com.br/tecnologia/2010/03/23/virtualizacao-traz-beneficios-e-desafios-indica-pesquisa/>. Acesso em 27 ago. 2010.

3. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. A MV é um *software* que não acessa diretamente o *hardware* da máquina. Essa função é mantida pelo sistema operacional com o qual ela interage.

B – Alternativa incorreta.
JUSTIFICATIVA. A MV não permite a economia de carga de CPU ou memória RAM. Pelo contrário, como há uma camada adicional de software, existe uma sobrecarga (overhead) na máquina física.

C – Alternativa correta.
JUSTIFICATIVA. Cada MV se comporta como um ambiente completamente independente, o que reforça a segurança na execução de um aplicativo. Cada MV pode ter suas restrições de segurança independentes em função dos aplicativos que rodam em seu ambiente. Por exemplo, um vírus que for adquirido na máquina virtual não contamina a máquina real.

D – Alternativa incorreta.
JUSTIFICATIVA. A MV não emula processador e RAM. Ela usa o processador e memória instalados. Os processadores com mais de um núcleo (dual-core ou demais) são de 64 bits e para utilizá-los é necessário que o sistema hospedeiro da MV suporte 64 bits.

E – Alternativa incorreta.

JUSTIFICATIVA. As MVs estão sujeitas às mesmas ameaças de um computador físico, mas os vírus adquiridos na máquina virtual não migram para a máquina real. Portanto, as mesmas preocupações de segurança têm de ser mantidas nos dois tipos de ambientes (real e virtual), como um bom sistema de segurança, *firewalls*, antivírus etc.

Questão 6

Questão 6.[6]

O *Rational Unified Process* (RUP) é um processo de engenharia de *software* cujo objetivo é assegurar a produção de *software* de alta qualidade, satisfazendo as necessidades dos usuários no prazo e nos custos previstos. O RUP contém uma estrutura que pode ser adaptada e estendida, pois é formado por duas estruturas principais, denominadas dimensões, que representam os aspectos dinâmicos e estáticos do processo. O aspecto dinâmico é expresso em ciclos, fases, iterações e marcos. O estático, por sua vez, contém as disciplinas, os fluxos, os artefatos e os trabalhadores.

Com base na iteração do RUP, julgue as asserções e a relação proposta entre elas.

I. A cada iteração das fases do RUP, geram-se ou não artefatos de *software*

PORQUE

II. Os artefatos produzidos dependem da ênfase que é dada a cada disciplina.

Assinale a opção correta.

A. As duas asserções são proposições verdadeiras, e a segunda justifica a primeira.

B. As duas asserções são proposições verdadeiras, e a segunda não justifica a primeira.

C. A primeira asserção é uma proposição verdadeira, e a segunda, uma proposição falsa.

D. A primeira asserção é uma proposição falsa, e a segunda, uma proposição verdadeira.

E. As duas asserções são proposições falsas.

1. Introdução teórica

Iteração do RUP

Primeiramente, temos que entender o conceito de iteração do RUP. Uma iteração inclui as atividades de desenvolvimento que levam à liberação de um produto — uma versão do produto estável e executável junto com qualquer outro elemento periférico necessário para usar esse *release*. Logo, uma iteração de desenvolvimento pode ser comparada a uma passagem completa por todas as disciplinas, ou, pelo menos, requisitos, análise e *design*, implementação e teste. Os critérios de avaliação são estabelecidos quando cada iteração é planejada. O *release* terá planejado a capacidade que é demonstrável. A duração de uma iteração depende do tamanho e da natureza do projeto, mas é provável que diversos *builds*

[6]Questão 16 – Enade 2008.

sejam construídos em cada iteração, do modo especificado no plano de integração do *build* para a iteração.

Uma iteração é um *loop* completo de desenvolvimento que resulta em um *release* (interno ou externo) de um produto executável, um subconjunto do produto final em desenvolvimento, que cresce por incrementos, a cada iteração, para se tornar o sistema final.

A cada iteração, os artefatos são atualizados, de maneira similar a um *software* "em crescimento". Em vez de desenvolver artefatos uns após os outros, como em um *pipeline*, eles são desenvolvidos por meio do ciclo, apesar das taxas diferentes.

Um artefato é um produto de trabalho final ou intermediário produzido e usado durante o projeto. Pode ser um documento ou um modelo como, por exemplo, o Documento de Arquitetura de *Software*, as Especificações Suplementares e o Diagrama de Caso de Uso.

Os artefatos são utilizados nas disciplinas do RUP. Uma disciplina mostra todas as atividades que devem ser realizadas para produzir determinado conjunto de artefatos.

Uma iteração, portanto, passa por todo o ciclo de desenvolvimento, gerando inúmeros artefatos que são atualizados nas iterações seguintes. Portanto, a geração de artefatos em uma iteração é obrigatória por ser parte inseparável dos trabalhos da iteração.

Alguns exemplos de artefatos gerados em uma iteração do RUP são os citados abaixo.

- Modelagem de Negócio (Regras de Negócio, Casos de Uso de Negócio, Visão do Negócio, Especificação Suplementar de Negócios).
- Requisitos (Plano de Gerenciamento de Requisitos, Glossário, Visão, Especificações Suplementares e Modelo de Caso de Uso).
- Análise e *Design* (Modelo de Análise, Modelo de Design, Documento de Arquitetura de *Software*, Realização de Caso de Uso e Pacote de *Design*).
- Implementação (Componentes, Plano de Integração do *Build* e *Build*).
- Teste (Plano de Teste, Caso de Teste e *Script* de Teste).
- Gerenciamento de Projeto (Plano de Iteração, Lista de Riscos, Plano de Gerenciamento de Riscos e Plano de Aceitação do Produto).

2. Indicações bibliográficas

- KRUCHTEN, P. *Introdução ao RUP.* São Paulo: Ciência moderna, 2003.
- MARTINS, J. C. C. *Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML.* 4. ed. São Paulo: Brasport, 2007.

- Processo Unificado do RUP (*RUP-Rational Unified Process*, versão 2002.05.00 em Português). Disponível em <http://www.wthreex.com/rup/>. Acesso em 15 mar. 2021.

3. Análise das asserções

I - Asserção falsa.
JUSTIFICATIVA. A cada iteração, geram-se vários artefatos. Não são opcionais, são mandatórios.

II - Asserção verdadeira.
JUSTIFICATIVA. Cada artefato é escolhido em função da ênfase ou do peso de cada disciplina. Projetos de curta duração não gerarão vários artefatos, que seriam produzidos em um projeto com alguns anos de duração. Portanto, a quantidade de artefatos pode variar dependendo do ciclo de vida empregado, que pode ser incremental, evolutivo, liberação incremental ou *design* principal. O RUP os define conforme segue.
A estratégia incremental determina as necessidades do usuário e os requisitos do sistema e, em seguida, realiza o restante do desenvolvimento em uma sequência de *builds*. O primeiro *build* incorpora partes das capacidades planejadas, o próximo *build* adiciona mais capacidades e assim por diante até o sistema estar completo.
A estratégia evolutiva difere da incremental, pois reconhece que as necessidades do usuário não são totalmente entendidas e que os requisitos não podem ser definidos antecipadamente, sendo refinados em cada *build* sucessivo.
Alguns autores também definiram em fases as liberações de funcionalidade incremental para o cliente. Isso pode ser necessário quando há pressões de mercado sobre o tempo restrito, sendo que a liberação antecipada de recursos importantes pode render benefícios de negócio significativos.
A abordagem em cascata tradicional pode ser vista como um caso degenerado em que há apenas uma iteração na fase de construção. Ela chama-se design principal. Na prática, é difícil evitar iterações adicionais na fase de transição.

Alternativa correta: D.

Questão 7

Questão 7.[7]

Os alunos de uma disciplina deveriam escolher um sistema de média complexidade, contendo no mínimo 100 funcionalidades, para ser modelado em UML e codificado em uma linguagem orientada a objetos. Um dos grupos de alunos estabeleceu a seguinte estratégia para identificação e seleção do sistema.

I. Cada integrante do grupo deveria criar um nickname (apelido) em um software de chat.

II. O grupo deveria se reunir em um horário predeterminado.

III. Durante o chat, os seguintes procedimentos deveriam ser realizados:

- cada integrante deveria sugerir um ou mais sistemas e justificar sua escolha, e não poderia criticar as ideias dos outros;
- à medida que as ideias fossem digitadas, o líder deveria copiá-las para um editor de texto e controlar o tempo de sugestão;
- quando o limite de tempo fosse atingido, o líder disponibilizaria todas as sugestões para serem analisadas pelo grupo;
- as 5 melhores ideias seriam selecionadas e colocadas em votação para a escolha da melhor ideia, segundo critérios predefinidos.

Nessa situação, a estratégia utilizada pelo grupo de alunos é uma adaptação da técnica de levantamento e elicitação de requisitos denominada

A. *joint application design.*

B. PIECES (*perfomance* informação/dados economia controle eficiência e serviços).

C. *facilitaded application specification techniques*.

D. entrevista.

E. brainstorming.

1. Introdução teórica

Técnicas para captura (elicitação) de requisitos e resolução de problemas

Existem muitas técnicas para captura (elicitação) de requisitos e resolução de problemas. Entre elas, a questão cita as indicadas a seguir.

[7]Questão 17 – Enade 2008.

a) Jad *(Joint Application Design)* – é uma técnica de captura de requisitos baseada em reuniões. As técnicas JAD aplicam os conceitos de visões refinadas, sendo três as visões da aplicação: overview (geral), macro e detalhe.

- Na visão *overview*, usuários e analistas falam das dificuldades e dos problemas, tendo o primeiro contato com as necessidades da aplicação.
- Na visão macro, usuários e analistas falam dos inter-relacionamentos existentes entre as necessidades, as dificuldades e os objetivos da aplicação.
- Na visão detalhe, são focadas as particularidades de um objetivo.

Nessa técnica, o trabalho tem que ser feito em grupo, com equipes de no mínimo duas pessoas, em ambiente de reunião e o desenvolvimento deve seguir algum método.

b) Pieces *(Performance, Information, Economics, Control, Efficiency, Servic)* – é uma técnica usada para identificar problemas operacionais a serem resolvidos.

- No item desempenho *(performance),* indicamos se os tempos de resposta e throughput (taxa de transferência) são adequados.
- No item informação *(information),* indicamos se os usuários estão recebendo informações em tempo, acuradas e úteis.
- No item economia *(economy),* indicamos se os serviços prestados valem o seu custo.
- No item controle *(control),* indicamos se há controles efetivos para prover informação correta e segura.
- No item eficiência (efficiency), indicamos se o sistema atual faz bom uso dos recursos.
- No item serviços *(services),* indicamos se os serviços atuais são confiáveis, flexíveis e expansíveis.

c) *Fast (Facilitated Application Specification Technique)* – é uma técnica para captura de requisitos. Seu principal objetivo é cobrir o *gap* (falha de entendimento) entre o que o desenvolvedor pensa que vai produzir e o que os clientes pensam que vão receber. Essa técnica inicialmente foi uma melhoria da técnica JAD.

d) Entrevista – é uma interação entre duas ou mais pessoas (entrevistadores e entrevistados) por meio de perguntas e respostas para o levantamento de informações (requisitos ou quaisquer outras informações).

e) *Brainstorming* – traduzido, significa "tempestade cerebral". É uma técnica utilizada para gerar ideias relacionadas a um tópico específico, envolvendo um apanhado de ideias em um curto período de tempo.

As ideias somente são avaliadas no final do processo. Pensamentos incomuns e heterodoxos são permitidos e incentivados, não se permitindo críticas ou justificativas, de modo que todos se sintam livres para participar. Uma ideia iniciada por um dos participantes dispara novas ideias nos demais, por meio de associações, de modo que todos gerem e compartilhem conhecimento e experiência.

A sessão termina quando o suprimento de pensamentos e de ideias acaba, ou quando o tempo estimulado para a sessão chega ao fim. Então, todas as ideias são analisadas e agrupadas de acordo com sua aplicabilidade e prioridade.

As regras básicas do *brainstorming* são:

- atmosfera informal;
- foco inicial na quantidade;
- combinação e melhoria das ideias;
- geração de ideias incomuns.

A partir dessas definições, vamos analisar a questão. A técnica utilizada pela equipe é uma adaptação do *brainstorming*. As características que determinam essa técnica são as indicadas abaixo.

- Criação de *nicknames*, mostrando que os integrantes da reunião poderiam, por opção própria, ficar anônimos.
- Não poderia haver crítica às ideias dos outros.
- As ideias surgiriam e seriam documentadas para posterior análise.
- A análise somente ocorreria ao final da reunião.
- Haveria votação e seleção das 5 melhores ideias, conforme critérios pré-definidos.

2. Indicações bibliográficas

- MAGELA, R. *Engenharia de software aplicada: Princípios.* Porto Alegre: Alta Books, 2006.
- NOGUEIRA, M. *Engenharia de software.* São Paulo: Ciência Moderna, 2009.
- PRESSMAN, R. S. *Engenharia de software.* 6. ed. São Paulo: McGraw-Hill, 2006.
- SOMMERVILLE, I. *Engenharia de software.* 8. ed. São Paulo: Addison Wesley, 2007.

3. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. Não é aplicada a técnica JAD porque a reunião é feita online, sem os recursos visuais e presenciais de uma reunião JAD.

B – Alternativa incorreta.
JUSTIFICATIVA. Não se aplica a técnica PIECES porque se analisa apenas a implantação de um sistema, não se está procurando resolver problemas operacionais.

C – Alternativa incorreta.
JUSTIFICATIVA. Não se aplica a técnica FAST porque neste momento inicial não há qualquer *gap* a ser preenchido.

D – Alternativa incorreta.
JUSTIFICATIVA. Não é uma sessão de entrevista porque não se usa a técnica de perguntas e respostas.

E – Alternativa correta.
JUSTIFICATIVA. Utiliza-se o *brainstorming*, conforme explicações anteriores.

Questão 8

Questão 8.[8]

Uma indústria de alimentos compra sementes de vários fornecedores. No recebimento das cargas, as sementes passam por uma operação de classificação por cor, em uma esteira adquirida do fabricante MAQ, equipada com sensores e *software* de processamento de imagens. Na etapa seguinte do processo, as sementes são separadas em lotes, pelo critério de tamanho, e são, então, empacotadas. A separação dos lotes é realizada por um mecanismo robótico, controlado por computador e que, pelo fato de sofrer contínuo desgaste, necessita ser substituído a cada 1.000 horas de uso. Durante a última troca, em razão da indisponibilidade do equipamento produzido pela empresa MAQ, a indústria instalou, com sucesso, um equipamento robótico similar.

Considerando o processo descrito, julgue os itens a seguir, relacionados aos fatores de qualidade.

I. As operações de classificação e separação de sementes se interrelacionam e não podem falhar, pois essa falha acarretaria prejuízos. O atributo de qualidade correspondente a essas operações, e que deve ser observado pelo *software*, é a interoperabilidade.

II. Caso o responsável pela instalação do sistema robotizado não tenha encontrado dificuldade em fazê-lo comunicar-se com o equipamento de outra marca, é correto concluir que o sistema que controla o robô é portável.

III. A maneira como ocorre a interação com o sistema computacional sugere que alguns requisitos, como ergonomia, sejam observados na interface. Por isso, é correto concluir que o software utilizado pela indústria contempla o fator denominado usabilidade.

Assinale a opção correta.

A. Apenas um item está certo.

B. Apenas os itens I e II estão certos.

C. Apenas os itens I e III estão certos.

D. Apenas os itens II e III estão certos.

E. Todos os itens estão certos.

[8]Questão 18 – Enade 2008.

1. Introdução teórica

Interoperabilidade, confiabilidade, portabilidade e usabilidade

A norma ISO 9126 define interoperabilidade como a "habilidade de dois ou mais sistemas (computadores, meios de comunicação, redes, *softwares* e outros componentes de tecnologia da informação) de interagirem e de intercambiarem dados de acordo com um método definido, de forma a obter os resultados esperados". É, portanto, o intercâmbio coerente de informações e serviços entre sistemas, isto é, sua capacidade de interagir com os sistemas especificados.

A confiabilidade de um sistema é a probabilidade de um *software* operar sem ocorrência de falhas durante um período especificado de tempo em determinado ambiente. É o conjunto de atributos que evidenciam a capacidade do *software* de manter seu nível de desempenho sob condições estabelecidas durante um período de tempo estabelecido. Tem como subcaracterísticas: maturidade, tolerância a falhas, recuperabilidade e conformidade.

Portabilidade é a capacidade de transferência de *software* ou *hardware* de um ambiente para outro, com garantia de seu pleno funcionamento.

Usabilidade é a garantia do uso eficiente e confortável dos sistemas computacionais por seus diversos tipos de usuários. Engloba técnicas, processos, métodos e procedimentos para se projetarem *interfaces*, visando à facilidade do uso. É um atributo de qualidade que inclui inteligibilidade (fácil entendimento), apreensibilidade (fácil aprendizado), operacionalidade (fácil uso), atratividade (agradável aos sentidos).

Em termos de esforço (custo) despendido na implantação desses atributos de qualidade, podemos conceituar o que segue abaixo.

- Interoperabilidade **–** é o esforço exigido para fazer com que dois sistemas ou componentes se comuniquem adequadamente.
- Confiabilidade – é o esforço exigido para que a operação do sistema seja livre de falhas, num ambiente específico, em determinado período de tempo.
- Portabilidade – é o esforço exigido para transferir um sistema de um ambiente de *hardware* ou *software* para outro.
- Usabilidade – é o esforço exigido para aprender a operar o *software*, preparar a entrada e interpretar a saída de um sistema.

2. Indicações bibliográficas

- KOSCIANSKI, A.; SOARES, M. S. *Qualidade de software.* 2. ed. São Paulo: Novatec, 2007.
- PRESSMAN, R. S. *Engenharia de software.* 6. ed. São Paulo: McGraw-Hill, 2006.
- SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Addison Wesley, 2007.

3. Análise dos itens

I – Item incorreto.
JUSTIFICATIVA. Operações de classificação e separação de sementes não podem falhar. O atributo que corresponde a essas operações é a confiabilidade.

II - Item incorreto.
JUSTIFICATIVA. A substituição do robô por um equipamento de outra marca não significa portabilidade, porque é mantido o mesmo ambiente. O novo robô é que tem uma interface compatível com o sistema atual.

III – Item correto.
JUSTIFICATIVA. A ergonomia de *software* desenvolve conhecimentos sobre as capacidades, limites e outras características do desempenho humano que se relacionam com o projeto de interfaces entre indivíduos e outros componentes do sistema. É parte do fator de qualidade a usabilidade.

Alternativa correta: A.

Questão 9

Questão 9.[9]

Uma instituição de auxílio a desabrigados tem a preocupação de fornecer uma alimentação equilibrada a seus pensionistas. Para atingir esse objetivo, decidiu empregar um sistema informatizado e contratou um analista para projetá-lo. O analista, que deveria empregar UML na modelagem do sistema, recebeu as informações a seguir acerca das refeições.

- Café da manhã: dois tipos de carboidrato, duas vitaminas e duas proteínas.
- Almoço: dois tipos de carboidrato e de proteínas, quatro tipos de vitamina e um tipo de lipídio.
- Jantar: um tipo de carboidrato, uma proteína e uma vitamina.

Cada tipo de alimento deve ser acompanhado por seu nome, sua porção recomendável, por refeição, e seu valor calórico, por porção. O cálculo para descobrir a quantidade de calorias para cada pensionista é dado pelo produto do fator de atividade (FA) pela taxa de metabolismo basal (TMB). Esses dois valores são obtidos nas tabelas I e II a seguir.

Tabela I

	idade	taxa de metabolismo basal
mulheres	de 30 a 60 anos	8,7 × peso(kg) + 829
	acima de 60 anos	10,5 × peso(kg) + 596
homens	de 30 a 60 anos	8,7 × peso(kg) + 879
	acima de 60 anos	13,5 × peso(kg) + 487

[9]Questão 19 – Enade 2008.

Tabela II

descrição	valor do fator de atividade (FA)	
fica a maior parte do tempo sentado e não pratica atividades físicas programadas	mulheres e homens	1,2
caminha até o ponto de ônibus, mas sem atividades físicas programadas	mulheres	1,3
	homens	1,4
realiza atividades físicas três vezes por semana, cerca de 30 minutos por dia	mulheres	1,45
	homens	1,5
faz duas horas e meia de atividades físicas diárias	mulheres	1,5
	homens	1,6
pratica atividade física diária por mais de três horas	mulheres	1,7
	homens	1,8

O cardápio de cada pensionista deve ser gerado, a cada dia, com base no cálculo da quantidade de calorias recomendada para cada um deles e, depois, deve ser encaminhado para a cozinheira.

Considerando as necessidades da instituição no que se refere ao cardápio diário e a aspectos da modelagem conceitual com UML, julgue os itens a seguir, acerca da classe Refeição.

I. Para o cálculo da TMB, são precondições que a idade seja um valor maior do que 30 anos e que seja relacionada uma das descrições da tabela II para o valor de FA.

II. Essa classe tem um método denominado montarCardápioDiário() que será sobrescrito nas subclasses.

III. Suas subclasses não implementam o método para calcular a quantidade de calorias, utilizando a implementação já definida na classe pai.

IV. Essa classe possui associações um-para-um com a classe Pensionista e agregação com a classe Alimento.

V. O conceito de acoplamento é um critério importante durante a modelagem da classe Refeição, pois diminui a quantidade de seus relacionamentos, o que contribui para o seu reuso.

São corretos apenas os itens

A. I e II.

B. I e IV.

C. II e III.

D. III e V.

E. IV e V.

1. Introdução teórica

Métodos sobrescritos

Um método da superclasse é sobrescrito (na subclasse) quando tem a mesma assinatura (nome e parâmetros), o mesmo tipo de retorno e o seu código interno é reescrito, ficando diferente da superclasse. A sobrescrita é um recurso poderoso, porque uma superclasse pode definir o método e várias subclasses podem implementar uma versão conveniente.

Na figura 1, há um exemplo de uma hierarquia de classes com métodos sobrescritos.

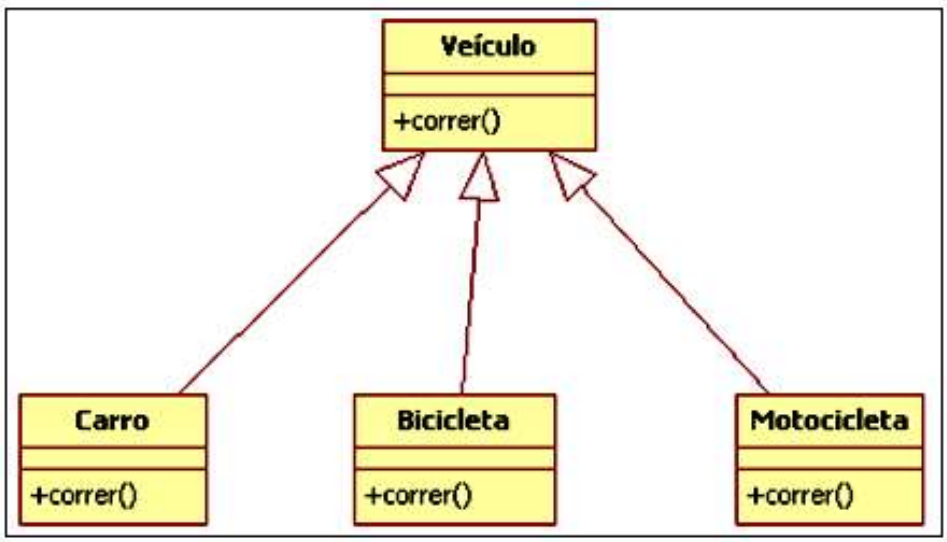


Figura 1. Subclasses com métodos sobrescritos.

Em uma associação "um-para-um" entre classes, uma ocorrência da classe A se relaciona com apenas uma ocorrência da classe B. Nessa associação, as classes de ambas as extremidades da associação devem ter apenas UMA ocorrência do objeto definido pela classe da outra extremidade.

Uma associação de agregação significa que a classe que contém o símbolo da agregação (um diamante não preenchido) deve ter uma ou mais ocorrências do objeto definida pela classe da extremidade oposta.

Os três relacionamentos citados estão representados na figura 2.

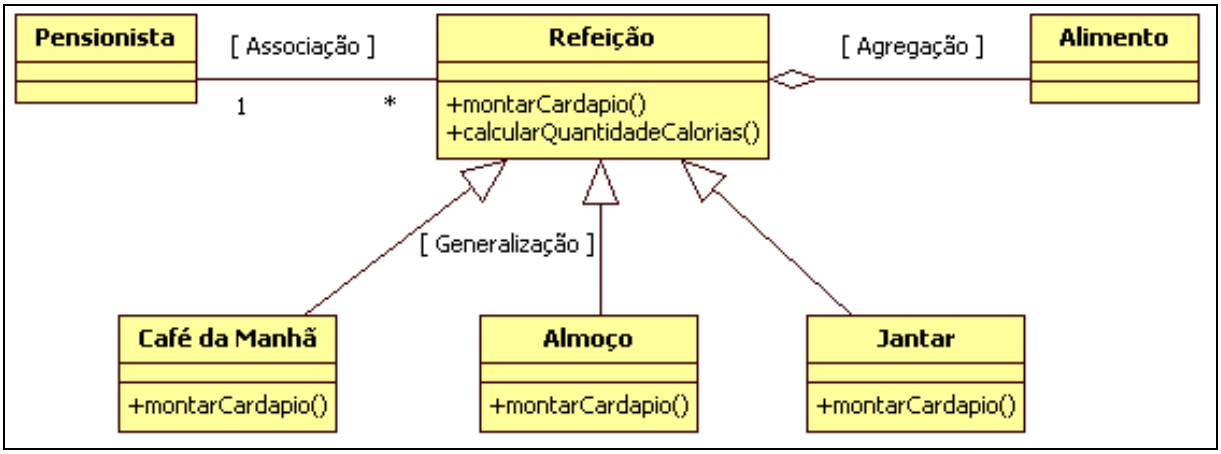


Figura 2. Diagrama de Classes.

O acoplamento mede o quanto uma classe depende ou está relacionada à outra classe. Quanto mais forte o acoplamento, mais problemas futuros de manutenção do *softwar*e serão gerados, causando os problemas abaixo.

- Classes difíceis de aproveitar. Sempre que uma for utilizada, todas as outras das quais ela depende devem estar presentes.
- Alterações nas classes relacionadas podem forçar mudanças locais.
- Dificuldades de compreensão isolada.

Uma regra geral para diminuir o acoplamento entre as classes é programar para uma interface. Fazendo isso, desacopla-se o código de uma implementação específica, tornando-a dependente apenas da interface.

O conceito de acoplamento está muito ligado ao conceito de coesão. Eles têm significados diferentes, mas estão inter-relacionados.

Acoplamento é o nível de interligação de duas classes, ou seja, o quanto uma classe conhece da outra. Ele é proporcionalmente ligado à quantidade de trabalho que se tem para mudar a implementação de uma classe. Elementos muito acoplados são muito dependentes, isto é, ao mudar um é necessário também mudar o outro. A coesão representa o quanto uma classe é específica para desempenhar um papel em um contexto. Ela mostra o quanto as tarefas que determinado elemento realiza estão relacionadas com um mesmo conceito. Segue um exemplo de acoplamento da classe Carro com a classe Motor:

```
package enade;
public class Carro {
    private Motor motor;
    public Carro(Motor motor) {
        if (motor == null) {
            System.out.println("O Carro está sem motor");
        }
        this.motor = motor;
    }
}
```

Não existe zero acoplamento porque as classes têm que se comunicar. Mas o alto acoplamento é indesejável, porque diminui a reusabilidade de objetos e aumenta a chance de erros quando mudanças são feitas em um objeto acoplado.

2. Indicações bibliográficas

- MAGELA, R. *Engenharia de software aplicada: princípios.* Porto Alegre: Alta Books, 2006.
- NOGUEIRA, M. *Engenharia de software.* São Paulo: Ciência Moderna, 2009.
- PRESSMAN, R. S*. Engenharia de software.* 6. ed. São Paulo: McGraw-Hill, 2006.
- SOMMERVILLE, I. *Engenharia de software.* 8. ed. São Paulo: Addison Wesley, 2007.

3. Análise dos itens

I - Item incorreto.

JUSTIFICATIVA. Para o cálculo da TMB, são precondições que a idade seja um valor maior do que 30 anos e que seja relacionada uma das descrições da tabela II para o valor de FA. O cálculo do TMB é feito multiplicando-se um fator (que varia conforme a idade e o sexo) pelo peso, somando o resultado a um valor informado. A TMB não depende do Fator de Atividade (FA). Ele depende do sexo, da idade (a partir de 30 anos) e do peso do pensionista. Portanto, a primeira parte do item está correta, mas a segunda parte está errada.

II - Item correto.

JUSTIFICATIVA. Essa classe tem um método denominado montarCardápioDiário() que será sobrescrito nas subclasses. O diagrama de classe da questão é o dado na figura 2. Como

cada cardápio é diferente, o método montarCardapio() da superclasse (Refeição) tem que ser sobrescrito em cada subclasse (Café da Manhã, Almoço e Jantar).

III - Item correto.
JUSTIFICATIVA. Suas subclasses não implementam o método para calcular a quantidade de calorias, utilizando a implementação já definida na classe pai. O cálculo da quantidade de calorias (figura 2) não varia em função do tipo de cada refeição, portanto pode ficar na superclasse sem necessidade de ser sobrescrita nas subclasses.

IV - Item incorreto.
JUSTIFICATIVA. Essa classe possui associações "um-para-um" com a classe Pensionista e agregação com a classe Alimento. A classe Refeição possui uma associação de agregação com Alimento, porque uma refeição é feita de alimentos (figura 2). Mas o relacionamento com a classe Pensionista é de "muitos-para-um", porque um Pensionista tem três refeições. A primeira parte do item está incorreta, mas a segunda parte está correta.

V - Item correto.
JUSTIFICATIVA. O conceito de acoplamento é um critério importante durante a modelagem da classe Refeição, pois diminui a quantidade de seus relacionamentos, o que contribui para o seu reuso. O acoplamento é um critério muito importante durante a modelagem. Um baixo acoplamento diminui a quantidade dos relacionamentos de uma classe e contribui enormemente para que ela seja reusável.

Como existem três itens corretos e nenhuma resposta (de A até E) corresponde a essa situação, a questão foi **anulada**.

Questão 10

Questão 10.[10]

Leia o algoritmo a seguir.

```
1  Algoritmo ENADE2008
2  variaveis
3    V[0..4] ← {2,0,4,3,1}:inteiro
4    I,J,A : inteiro
5  inicio
6    para I ← 0 ate 3 passo 1 faca
7      para J ← 0 ate 3-I passo 1 faca
8        se (V[J] > V[J+1] ) entao
9          A ← V[J]
10         V[J] ← V[J+1]
11         V[J+1] ← A
12       fim se
13       escreva V[0],V[1],V[2],V[3],V[4]
14     fim para
15   fim para
16 fim algoritmo
```

Com relação ao algoritmo acima, que manipula um vetor de inteiros, julgue os itens abaixo.

I. Quando as variáveis I e J valerem, respectivamente, 0 e 1, a linha 13 apresentará a sequência de valores 0,2,4,3,1.

II. Quando as variáveis I e J valerem, respectivamente, 1 e 0, a linha 13 apresentará a sequência de valores 0,2,3,1,4.

III. Quando as variáveis I e J valerem, respectivamente, 1 e 2, a linha 13 apresentará a sequência de valores 0,2,1,3,4.

Assinale a opção correta.

A. Apenas um item está certo.

B. Apenas os itens I e II estão certos.

C. Apenas os itens I e III estão certos.

D. Apenas os itens II e III estão certos.

E. Todos os itens estão certos.

[10]Questão 20 – Enade 2008.

1. Introdução teórica

Algoritmos

A resolução de questões envolvendo algoritmos demanda muito cuidado e atenção. A utilização de um correto acompanhamento de cada passo do processo é fundamental para acertar os resultados intermediários e finais.

Na questão proposta, trabalhamos com um vetor (que é uma matriz unidimensional) contendo índices variando de 0 (zero) a 4 (quatro), porque é um vetor de 5 elementos.

Chamaremos de V0 (Vetor zero) o vetor original proposto na questão. Os demais vetores (V1 até V4) mostrarão as modificações que ocorrerão a cada passo (P1 a P7) do desenvolvimento do algoritmo.

Seguem, abaixo, dois quadros: um contendo o detalhamento das alterações de cada valor do algoritmo ou passo a passo (quadro 1) e outro com os vetores resultantes (quadro 2).

Quadro 1. Passo a passo.

Passo	**Índices I e J e condição de parada de J**			**Teste SE V[J] > V[J+1]**			**Atualização de Valores**		**Vetor resultante**
	I	**J**	**3-I**	**V[J]**	**V[J+1]**	**A**	**V[J]**	**V[J+1]**	
P1	0	0	3	2	0	2	0	2	V1
P2	**0**	**1**	**3**	**2**	**4**	**-**	**-**	**-**	**V1**
P3	0	2	3	4	3	4	3	4	V2
P4	0	3	3	4	1	4	1	4	V3
P5	**1**	**0**	**2**	**0**	**2**	**-**	**-**	**-**	**V3**
P6	1	1	2	2	3	-	-	-	V3
P7	**1**	**2**	**2**	**3**	**1**	**3**	**1**	**3**	**V4**

Quadro 2. Vetores resultantes.

Vetor	Índices→		0	1	2	3	4	
V0	Valores →	**[**	**2**	**0**	**4**	**3**	**1**	**]**
V1	Valores →	[	0	2	4	3	1	]
V2	Valores →	[	0	2	3	4	1	]
V3	Valores →	[	0	2	3	1	4	]
V4	Valores →	[	0	2	1	3	4	]

A execução do algoritmo segue os passos a seguir.

- A variável I vai de 0 a 3, mas a questão somente testa valores de 0 e 1, correspondentes aos passos P2, P5 e P7, destacados em itálico;
- A variável J varia em função do valor de I, sendo calculada como 3-I. Como a questão testa I somente até 1, J será testada somente até 2 (3-1).

- A condição SE testa sempre se o valor contido no índice de J é maior que o valor contido no índice imediatamente acima (J+1). Quando o valor contido no índice J é MENOR ou IGUAL ao seguinte, o vetor não se altera;
- A variável A tem a função de guardar o valor contido no índice J para permitir a troca de valores. Serão trocados os valores contidos em J e em J+1, utilizando a variável A como depósito temporário;

Portanto, o algoritmo faz duas coisas:

- se o valor contido em J for MENOR ou IGUAL ao valor contido em J+1, mantém o vetor como estava anteriormente;
- se o valor contido em J for MAIOR que o valor em J+1 ele TROCA os valores: o que estava em J+1 vai para J; o que estava em J vai para J+1, utilizando a variável A como depósito temporário.

2. Indicações bibliográficas

- CORMEN, T. H. *Algoritmos - teoria e prática.* São Paulo: Campus, 2002.
- DASGUPTA, S.; PAPADIMITRIOU, C. H.; VAZIRANI, U. *Algoritmos.* São Paulo: McGraw-Hill, 2006.
- GOODRICH, M. T.; TAMASSIA, R. *Projeto de algoritmo*. Porto Alegre: Bookman, 2004.
- MEDINA, M.; FERTIG, C. *Algoritmos e programação - Teoria e Prática.* São Paulo: Novatec, 2005.

3. Análise dos itens

I - Item correto.

JUSTIFICATIVA. Quando as variáveis I e J são, respectivamente, 0 e 1, o vetor resultante é [0,2,4,3,1], que corresponde ao vetor V1 no passo P2.

II - Item correto.

JUSTIFICATIVA. Quando as variáveis I e J são, respectivamente, 1 e 0, o vetor resultante é [0,2,3,1,4], que corresponde ao vetor V3 no passo P5.

III - Item correto.

JUSTIFICATIVA. Quando as variáveis I e J são, respectivamente, 1 e 2, o vetor resultante é [0,2,1,3,4], que corresponde ao vetor V4 no passo P7.

Alternativa correta: E.

Questão 11

Questão 11.[11]

Leia o algoritmo a seguir.

```
1 Algoritmo ENADE2008
2  variaveis
3    varA, varB, varC: inteiro
4    varF : real
5    varS : literal
6    varL : logico
7  inicio
8    varS ← "1000"
9    varA ← 4
10   varF ← 3.5
11   varC ← 0
12   varL ← VERDADEIRO
13   se((varC < varA) E varL OU (varS > varC)) entao
14      varB ← varF/varA
15    senao
16      varB ← varA/varC
17    fim se
18 fim algoritmo
```

O código acima

A. não apresenta erros de nenhum tipo.

B. apresenta erros de atribuição de tipo inválido, divisão por zero e expressão relacional inválida.

C. apresenta erros de atribuição de tipo inválido, divisão por zero e estrutura condicional.

D. apresenta erros de estrutura condicional e expressão relacional inválida.

E. apresenta somente erro de divisão por zero.

1. Introdução teórica

Algoritmos

A resolução de questões envolvendo algoritmos demanda muito cuidado e atenção. Nessa questão, é importante entender o tipo de cada dado e os valores que podem receber, bem como quais comparações são possíveis.

Números inteiros não possuem componentes decimais ou fracionários, podendo ser positivos ou negativos. Exemplos de números inteiros: 0, 24, -12.

[11]Questão 21 – Enade 2008.

Números reais são aqueles que possuem componentes decimais ou fracionários, podendo também ser positivos ou negativos. Exemplos de números reais: 24.01, 144, -13.3, 0.0, 0.

Tipo literal é constituído por uma sequência de caracteres contendo letras, dígitos e/ou símbolos especiais. Exemplos: "Qual?", " ", "AbCdEfGhI", "1+2=3".

Tipo lógico é usado para representar apenas dois valores possíveis: verdadeiro ou falso. Exemplo: T – true, verdadeiro, F-false, falso.

O quadro 1 auxilia na visualização de cada variável com seu tipo de dado.

Quadro1. Variável e seu tipo de dado.

Tipo de Dado	Inteiro	Real	Literal	Lógico
varA	X			
varB	X			
varC	X			
varF		X		
varS			X	
varL				X

Trocas de dados possíveis e regras de funcionamento são as indicadas abaixo.

- Um número inteiro pode ser movido para um número real sem problemas;
- Um número inteiro ou real não pode ser movido para um literal diretamente, a não ser por meio de conversões;
- Um literal não pode ser movido para um número inteiro ou real;
- Um número inteiro ou real não pode ser dividido por zero;
- Expressões lógicas somente podem verificar se o resultado de um teste é falso ou verdadeiro;
- Dado lógico somente aceita Verdadeiro ou Falso.

Portanto, na questão proposta, temos as irregularidades citadas abaixo.

- Na linha 13 cada teste tem que retornar Falso ou Verdadeiro para ser válido. *varC* < *varA* é verdadeiro, pois *varC* = 0 e varA = 4. *varL* já contém o valor Verdadeiro. Porém a variável *varS* é uma literal que não pode ser comparada com *varC*, que é inteira. A estrutura condicional está errada.
- Na linha 14 a variável *varB* recebe o resultado de *varF*/*varA*, que é uma atribuição de tipo inválida, já que *varF* contém o valor 3.5 e *varA*, 4, cujo resultado da divisão é 0,875. Este valor não pode ser colocado em um número inteiro (se uma linguagem de programação permitir truncará o valor decimal e guardará 0 – zero - na variável).
- Na linha 16 a variável *varC* está zerada, provocando uma divisão por zero: *varB* = 4/0.

2. Indicações bibliográficas

- CORMEN, T. H. *Algoritmos - teoria e prática*. São Paulo: Campus, 2002.
- DASGUPTA, S.; PAPADIMITRIOU, C. H.; VAZIRANI, U. *Algoritmos.* São Paulo: McGraw-Hill, 2006.
- GOODRICH, M. T.; TAMASSIA, R. *Projeto de algoritmo.* Porto Alegre: Bookman, 2004.
- MEDINA, M.; FERTIG, C. *Algoritmos e programação - Teoria e Prática.* São Paulo: Novatec, 2005.

3. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. Na linha 13, a expressão relacional está errada, além dos erros de atribuição inválida e divisão por zero.

B – Alternativa correta.
JUSTIFICATIVA. Os três erros são apresentados nas linhas 13, 14 e 16.

C – Alternativa incorreta.
JUSTIFICATIVA. A estrutura condicional está correta, a expressão relacional é que está errada na linha 13.

D – Alternativa incorreta.
JUSTIFICATIVA. Faltou identificar o erro de divisão por zero.

E – Alternativa incorreta.
JUSTIFICATIVA. Houve falhas na identificação dos demais erros de expressão relacional e atribuição inválida.

ÍNDICE REMISSIVO