UNIP
EDUCAÇÃO, POR SUA ESCOLHA

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 1

CQA - COMISSÃO DE QUALIFICAÇÃO E AVALIAÇÃO

CIÊNCIA DA COMPUTAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 1

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.

Larissa Rodrigues Damiani

Doutora e Mestra em Ciências pelo programa de Engenharia Elétrica – Microeletrônica e Engenheira Elétrica – modalidade Eletrônica (ênfase em Telemática). Professora titular 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]

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

Conceitos de programação orientada a objetos

A orientação a objetos é um paradigma de programação que modulariza o código-fonte de um sistema em torno de objetos, que são entidades que combinam características (atributos) e comportamentos (métodos) relacionados em uma única unidade.

Os objetos são instanciados a partir de classes, que funcionam como "receitas" ou "moldes" para criar objetos. Por exemplo, a classe `Caneta` pode ser usada para criar os objetos `esferografica` e `hidrografica`, cada um com suas próprias características, mas definidos pela mesma estrutura.

No contexto de orientação a objetos, a **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 métodos construtores também podem ser sobrecarregados.

Segue um exemplo de sobrecarga de método construtor.

[1]Questão 12 – Enade 2008.

```
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);
```

A **herança** é o mecanismo utilizado 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 classes de nível mais baixo herdam atributos e métodos da classe de nível mais alto (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 -> Mamifero -> Cachorro

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

A hierarquia é unidirecional. Desse modo, podemos falar que o cachorro é um mamífero, mas não podemos falar que o mamífero é necessariamente um cachorro (um mamífero poderia ser uma baleia, por exemplo).

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, feita para o método `comer()`.

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

A **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.

A **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 `FormularioCliente` solicita uma consulta ao objeto `PessoaFisica` por meio do método `ConsultarCliente(cpf)`.

2. Análise das alternativas

A – Alternativa incorreta.
JUSTIFICATIVA. A sobrecarga não permite o reaproveitamento nem 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. A 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.

3. Indicações bibliográficas

- FUGERI, S. *Programação orientada a objetos: conceitos e técnicas*. São Paulo: Érica, 2014.
- GILLEANES, T. A. G. *UML 2: uma abordagem prática*. São Paulo: Novatec, 2018.
- RANGEL, P. CARVALHO JUNIOR, J. G. de. *Sistemas orientados a objetos: teoria e prática com UML e Java*. Rio de Janeiro: Brasport, 2021.

Questão 2

Questão 2.[2]

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, avalie as afirmativas 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.

É correto apenas o que se afirma em

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

1. Introdução teórica

1.1. 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. Essas restrições são

[2]Questão 11 – Enade 2008.

limitações ou condições impostas ao sistema, que podem afetar hardware, software, dados e procedimentos operacionais. Exemplos de restrições de projeto incluem prazos de entrega, orçamento disponível, requisitos de qualidade e necessidade de usar tecnologias específicas.

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 deve 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.

1.2. Protótipo do sistema

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.

O protótipo 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 os usuários.

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

1.3. Processos iterativos

Os processos iterativos de software dividem o projeto de um sistema de software em ciclos curtos e repetidos (iterações ou sprints), em que o código é desenvolvido, testado e refinado progressivamente, até que a versão final seja alcançada.

Um processo iterativo 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 a mudança dos requisitos.

2. Análise das afirmativas

I – Afirmativa incorreta.

JUSTIFICATIVA. A execução sequencial das fases, sem retorno, não capturará as alterações nem as correções identificadas nas fases posteriores, sejam elas originadas internamente pelos usuários ou externamente por mudanças em legislações ou em regras.

II – Afirmativa correta.

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

III – Afirmativa incorreta.

JUSTIFICATIVA. Na etapa de engenharia, o objetivo é ter uma visão global do sistema, incluindo hardware, software, equipamentos e pessoas envolvidas. O detalhamento das restrições é feito em etapas posteriores.

IV – Afirmativa correta.

JUSTIFICATIVA. 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 de desenvolvimento do produto.

Alternativa correta: D.

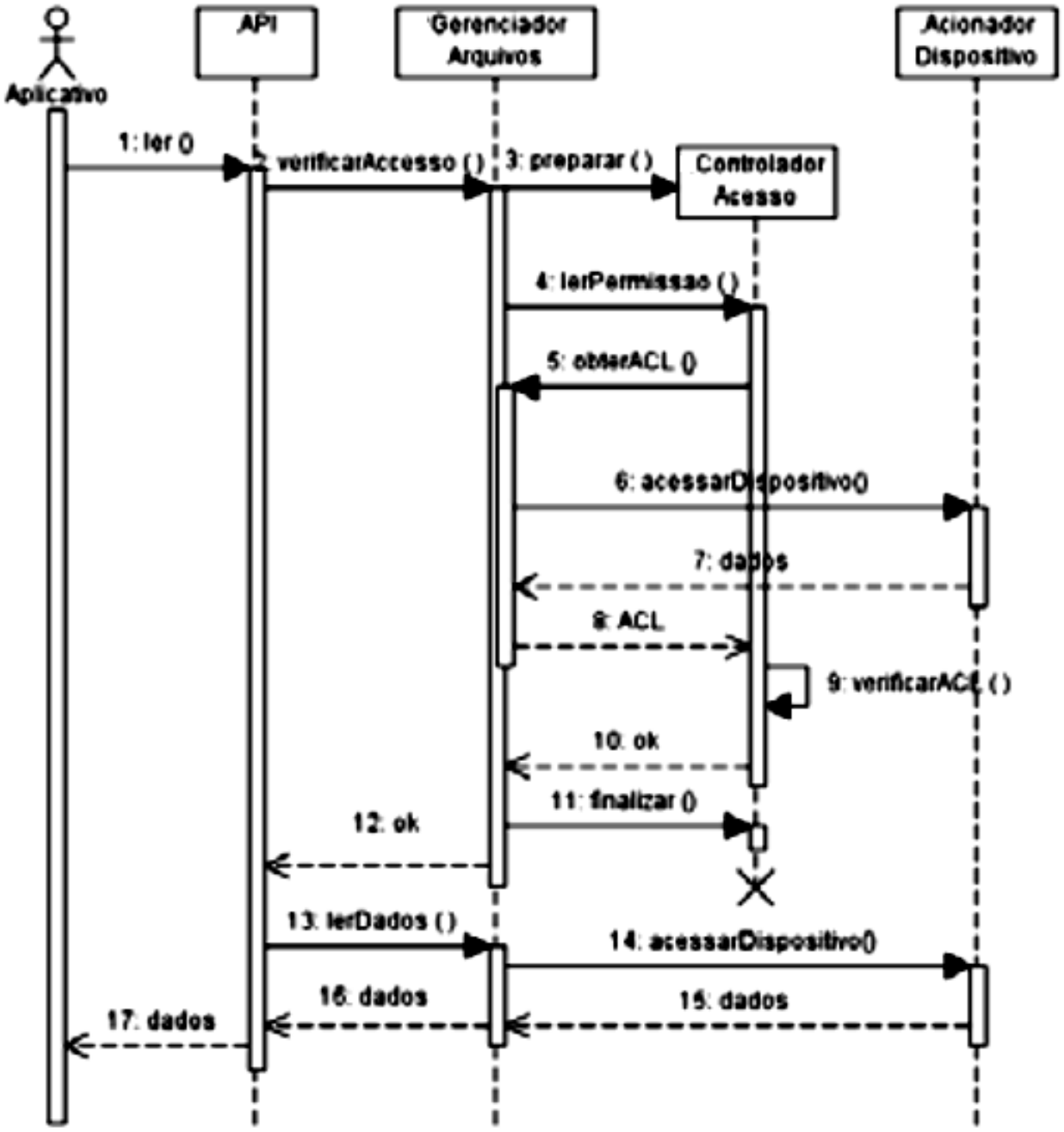
3. Indicações bibliográficas

- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software.* São Paulo: McGraw-Hill, 2021.
- SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Pearson, 2021.

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

O Diagrama de Sequência da UML (*Unified Modeling Language*) é 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 com 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. Além disso, o Diagrama de Sequência serve para validar e refinar os dois diagramas, podendo levar a complementações ou a 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 palito. 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. Ela se refere geralmente a uma instância e ao tempo de vida em que o objeto ou o 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 mensagens para si mesmo. Esse processo é denominado autochamada, como ocorre na mensagem *9. verificarACL()*.

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 melhorar a clareza do diagrama. Uma suposta economia de tempo e de 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.*

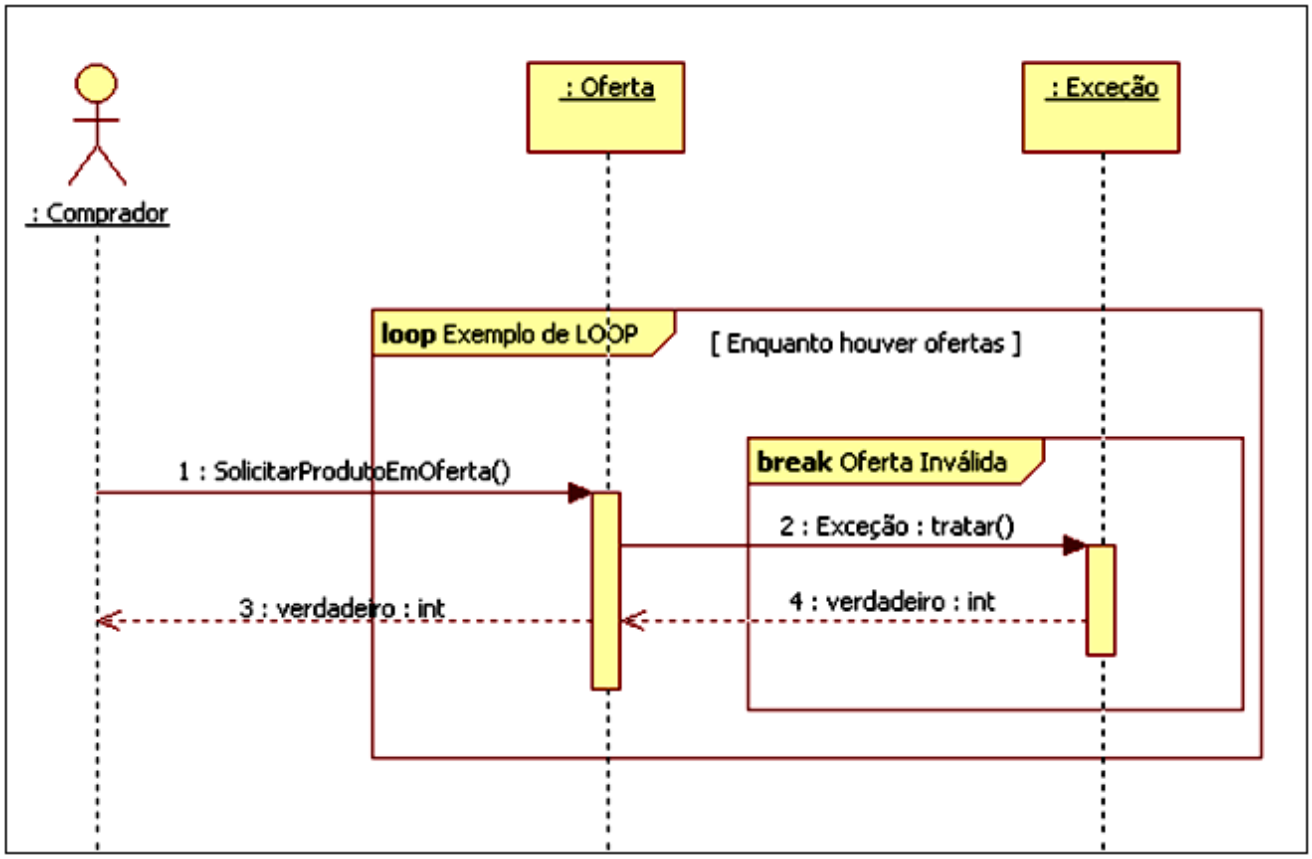


Figura 1. Exemplo de *loop* com *break.*

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 a destruição do objeto.

E – Alternativa incorreta.

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

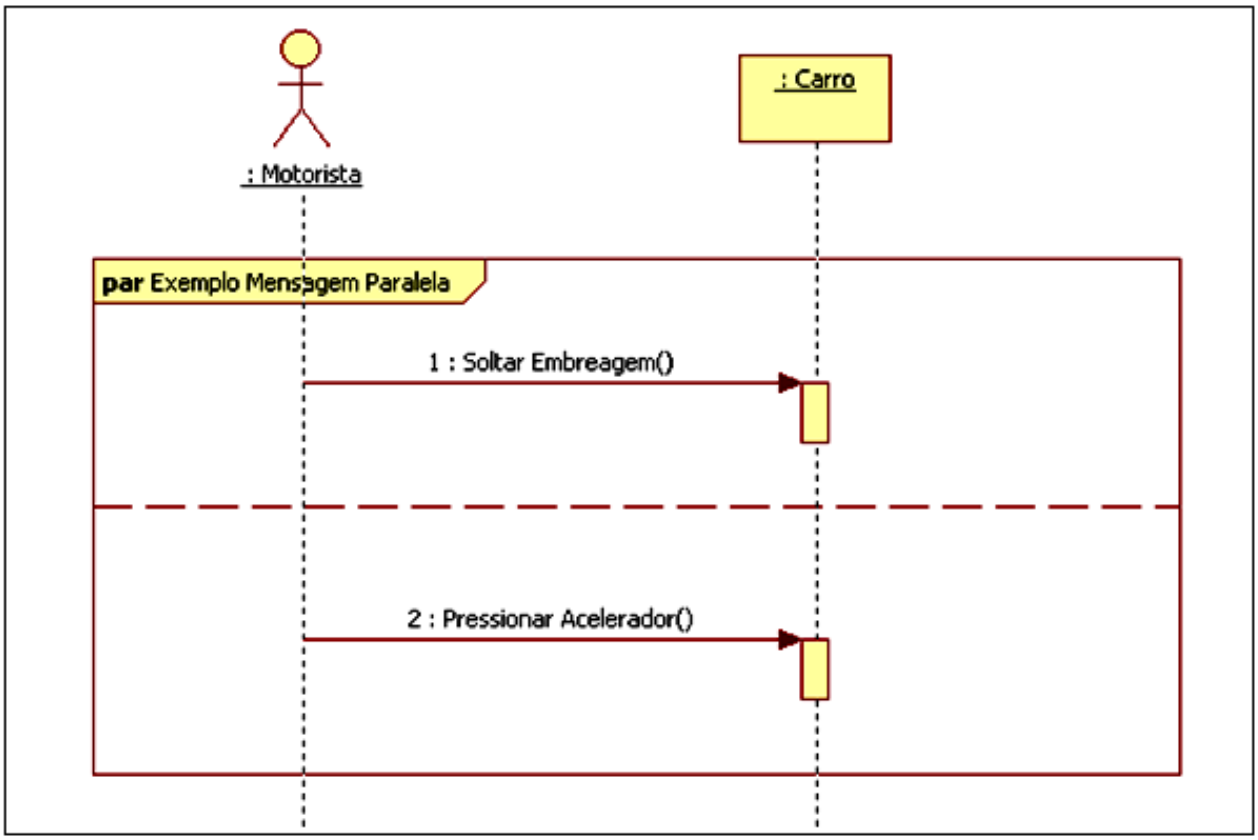


Figura 2. Fragmento combinado par.

3. Indicações bibliográficas

- FUGERI, S. *Programação orientada a objetos: conceitos e técnicas*. São Paulo: Érica, 2014.
- GILLEANES, T. A. G. *UML 2: uma abordagem prática*. São Paulo: Novatec, 2018.
- RANGEL, P. CARVALHO JUNIOR, J. G. de. *Sistemas orientados a objetos: teoria e prática com UML e Java*. Rio de Janeiro: Brasport, 2021.

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 subrotina `xis()` faz parte de um programa;
- a variável `i` é local à subrotina `xis()`;
- a variável `Gn` é global, e foi inicializada em outra parte do programa;
- o código-fonte disponível para a análise é apenas o da subrotina `xis()`;
- 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 (com adaptações).

1. Introdução teórica

Teste de caixa-branca

Um teste de 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. Para ser aplicado, requer conhecimento do código-fonte do produto em teste.

Para implementação do teste de 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 de 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 de 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 temos nenhum conhecimento de como o produto está construído internamente.

Para implementação do teste de caixa-preta, seguimos os passos abaixo.

- Analisamos a especificação dos requisitos.
- Escolhemos entradas válidas, conforme a especificação, para determinar o correto comportamento do produto.
- Escolhemos entradas inválidas, para verificar se são manipuladas corretamente.
- Determinamos as saídas esperadas, conforme as entradas escolhidas.
- Construímos os casos de teste.
- Executamos os casos de teste.
- Comparamos as saídas obtidas com as saídas esperadas.
- Geramos um relatório para avaliar o resultado dos testes.

O teste de 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. Análise das afirmativas

I – Afirmativa incorreta.

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

II – Afirmativa correta.

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 com `i=2`, com `i=3` etc., até que o valor de `i` e de `Gn` sejam iguais, ou até que `i` alcance o seu limite superior máximo. Portanto, é um teste válido e possível de ser feito.

III – Afirmativa incorreta.

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 a afirmativa falsa, pois ele não pode ser testado.

Alternativa correta: A.

3. Indicações bibliográficas

- BASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. *Base de conhecimento em teste de software.* São Paulo: Martins Fontes, 2008.
- POLO, R. C. *Validação e teste de software*. Curitiba: Contentus, 2020.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software.* São Paulo: McGraw-Hill, 2021.
- SOMMERVILLE, I. *Engenharia de software*. 8. São Paulo: Pearson, 2021.

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 (MV)

Uma máquina virtual (MV) é um software de ambiente computacional que permite a execução de sistemas operacionais e de aplicativos sobre um hardware hospedeiro. Ela 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

[5]Questão 15 – Enade 2008.

em uso. A restrição para o número de máquinas virtuais possíveis é definida 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.
- Há independência do hardware real, isto é, no mesmo servidor físico, podemos executar diferentes sistemas operacionais, como Windows ou Linux.
- Elas promovem aumento da segurança, pois cada máquina virtual é independente da outra. Assim, podemos ter diferentes requisitos de segurança em diferentes ambientes virtualizados.
- Há aumento da confiabilidade e da disponibilidade, pois a parada de um ambiente não interrompe os demais.
- Elas promovem a redução do custo, pois podemos trocar vários servidores menores por um mais poderoso, que virtualiza cada um dos servidores menores substituídos.
- Há melhoria do suporte a aplicações legadas. Ao migrar um sistema operacional em 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.

- 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.
- 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.

2. 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 de memória RAM. Pelo contrário, como há uma camada adicional de software, existe uma sobrecarga (overhead) na máquina física hospedeira.

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 a 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.

3. Indicações bibliográficas

- MACHADO, F. B. *Arquitetura de sistemas operacionais.* Rio de Janeiro: LTC, 2013.
- 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. 2025.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos.* São Paulo: Pearson Education do Brasil, 2016.
- TOSCANI, S. *Sistemas operacionais*. São Paulo: Artmed, 2010.

Questão 6

Questão 6.[6]

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 (TMB)**
Mulheres	de 30 a 60 anos	$8{,}7 \cdot peso(kg) + 829$
	acima de 60 anos	$10{,}5 \cdot peso(kg) + 596$
Homens	de 30 a 60 anos	$8{,}7 \cdot peso(kg) + 879$
	acima de 60 anos	$13{,}5 \cdot peso(kg) + 487$

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 atividades físicas diárias por mais de três horas	Mulheres	1,7
	Homens	1,8

[6]Questão 19 – Enade 2008.

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 reúso.

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, pois 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.

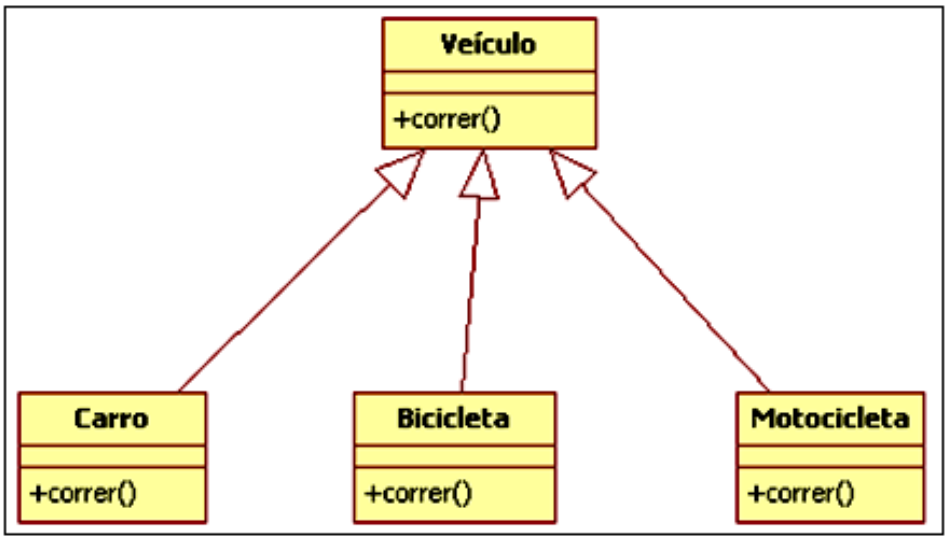


Figura 1. Subclasses com métodos sobrescritos.

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.

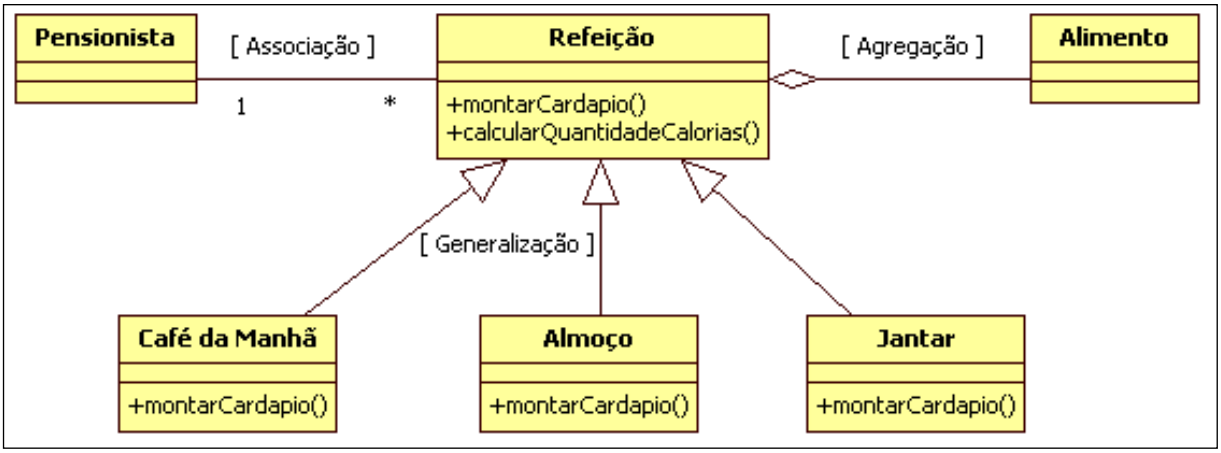


Figura 2. Diagrama de Classes.

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

- 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, pois diminui a reusabilidade de objetos e aumenta a chance de erros quando mudanças são feitas em um objeto acoplado.

3. Análise das afirmativas

I - Afirmativa incorreta.

JUSTIFICATIVA. 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). Ela depende do sexo, da idade (a partir de 30 anos) e do peso da pessoa. Portanto, a afirmativa está incorreta.

II - Afirmativa correta.
JUSTIFICATIVA. A afirmativa faz sentido no contexto da questão. 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 - Afirmativa correta.
JUSTIFICATIVA. O método para o cálculo da quantidade de calorias (figura 2) não varia em função do tipo de cada refeição e, portanto, pode ficar na superclasse, sem necessidade de ser sobrescrito nas subclasses.

IV - Afirmativa incorreta.
JUSTIFICATIVA. A classe Refeição tem uma associação de agregação com Alimento porque uma refeição é feita de alimentos (figura 2). No entanto, o relacionamento com a classe Pensionista é de "muitos-para-um" porque um Pensionista tem três refeições.

V - Afirmativa correta.
JUSTIFICATIVA. 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**.

3. Indicações bibliográficas

- FUGERI, S. *Programação orientada a objetos: conceitos e técnicas*. São Paulo: Érica, 2014.
- GILLEANES, T. A. G. *UML 2: uma abordagem prática*. São Paulo: Novatec, 2018.
- RANGEL, P. CARVALHO JUNIOR, J. G. de. *Sistemas orientados a objetos: teoria e prática com UML e Java*. Rio de Janeiro: Brasport, 2021.

Questão 7

Questão 7.[7]

Leia o algoritmo a seguir.

```
01. Algoritmo ENADE2008
02. Variaveis
03.     V[0..4] <- {2,0,4,3,1}: inteiro
04.     I, J, A: inteiro
05. Inicio
06.     para I <- 0 ate 3 passo 1 faca
07.         para J <- 0 ate 3-I passo 1 faca
08.             se (V[J] > V[J+1]) então
09.                 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.

[7]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 com 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 negrito.
- 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. Análise das afirmativas

I - Afirmativa correta.
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 - Afirmativa correta.
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 - Afirmativa correta.
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.

3. Indicações bibliográficas

- BHARGAVA, A. Y. *Entendendo algoritmos*. São Paulo: Novatec, 2017.
- CORMEN, T. H. *Algoritmos - teoria e prática.* São Paulo: LTC, 2012.

- DASGUPTA, S.; PAPADIMITRIOU, C. H.; VAZIRANI, U. *Algoritmos.* São Paulo: McGraw-Hill, 2006.
- MEDINA, M.; FERTIG, C. *Algoritmos e programação - Teoria e Prática.* São Paulo: Novatec, 2005.

Questão 8

Questão 8.[8]

Leia o algoritmo a seguir.

```
01. Algoritmo ENADE2008
02. variaveis
03.     varA, varB, varC: inteiro
04.     varF: real
05.     varS: literal
06.     varL: logico
07. inicio
08.     varS <- "1000"
09.     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.

[8]Questão 21 – Enade 2008.

1. Introdução teórica

Algoritmos

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

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

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 um número 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.

2. Análise da questão

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. Desse modo, a expressão relacional 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` contém o valor 4. O resultado da divisão é 0,875, e este valor não pode ser colocado em um número inteiro. Se a linguagem de programação permitir, truncará o valor decimal e guardará 0 na variável.
- Na linha 16, a variável `varC` está zerada, provocando uma divisão por zero: `varB = 4/0`.

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.

4. Indicações bibliográficas

- BHARGAVA, A. Y. *Entendendo algoritmos*. São Paulo: Novatec, 2017.
- CORMEN, T. H. *Algoritmos - teoria e prática.* São Paulo: LTC, 2012.
- DASGUPTA, S.; PAPADIMITRIOU, C. H.; VAZIRANI, U. *Algoritmos.* São Paulo: McGraw-Hill, 2006.
- MEDINA, M.; FERTIG, C. *Algoritmos e programação - Teoria e Prática.* São Paulo: Novatec, 2005.

ÍNDICE REMISSIVO

Questão 1	Sobrecarga, herança, hierarquia, sobreposição, abstração e mensagem.
Questão 2	Restrições de projeto.
Questão 3	Diagrama de Sequência da UML.
Questão 4	Testes de software.
Questão 5	Máquina Virtual (MV).
Questão 6	Métodos sobrescritos.
Questão 7	Algoritmos.
Questão 8	Algoritmos.