MINISTÉRIO DA CIÊNCIA E TECNOLOGIA (MCT)
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Concurso Público
NÍVEL SUPERIOR

Aplicação: 25/1/2009

CARGO: Tecnologista da Carreira de Desenvolvimento Tecnológico Classe: Tecnologista Pleno 2 Padrão I

MANHÃ (TS04)

LEIA COM ATENÇÃO AS INSTRUÇÕES ABAIXO.

1 **Ao receber este caderno, verifique se ele contém setenta e cinco itens, correspondentes às provas escritas objetivas, corretamente ordenados de 1 a 75, e dez temas referentes à prova escrita discursiva — devendo seu texto ser escrito com base unicamente no tema sorteado —, acompanhada de espaço para rascunho.**

2 Quando autorizado pelo aplicador, no momento da identificação, escreva, no espaço apropriado da folha de respostas, com a sua caligrafia usual, a seguinte frase:

A maior capacidade é a confiabilidade.

3 Caso o caderno esteja incompleto ou tenha qualquer defeito, ou haja divergência quanto ao cargo ou sigla do cargo, registrados nessa capa, no rodapé de cada página numerada deste caderno, na folha de respostas e na folha de texto definitivo da prova escrita discursiva, solicite ao aplicador mais próximo que tome as providências cabíveis, pois não serão aceitas reclamações posteriores.

4 Não serão distribuídas folhas suplementares para rascunho nem para texto definitivo.

5 Não utilize lápis, lapiseira (grafite), borracha e(ou) qualquer material de consulta que não seja fornecido pelo CESPE/UnB.

6 Não se comunique com outros candidatos nem se levante sem autorização de um aplicador.

7 Nos itens das provas objetivas, recomenda-se não marcar ao acaso: cada item cuja resposta divirja do gabarito oficial definitivo receberá pontuação negativa, conforme consta em edital.

8 A duração das provas é de **quatro horas e trinta minutos**, já incluído o tempo destinado à identificação — que será feita no decorrer das provas —, ao preenchimento da folha de respostas e à transcrição do texto definitivo da prova escrita discursiva para a folha de texto definitivo.

9 Você deverá permanecer obrigatoriamente em sala por, no mínimo, **uma hora** após o início das provas e poderá levar este caderno de provas somente no decurso dos últimos **quinze minutos** anteriores ao horário determinado para o término das provas.

10 Ao terminar as provas, chame aplicador mais próximo, devolva-lhe a sua folha de respostas e a sua folha de texto definitivo da prova escrita discursiva e deixe o local de provas.

11 A desobediência a qualquer uma das determinações constantes no presente caderno, na folha de respostas ou na folha de texto definitivo da prova escrita discursiva poderá implicar a anulação das suas provas.

AGENDA (datas prováveis)

I **27/1/2009**, após as 19 h (horário de Brasília) – Gabaritos oficiais preliminares das provas escritas objetivas: Internet — www.cespe.unb.br.

II **28 e 29/1/2009** – Recursos (provas escritas objetivas): exclusivamente no Sistema Eletrônico de Interposição de Recurso, Internet, mediante instruções e formulários que estarão disponíveis nesse sistema.

III **25/2/2009** – Resultado final das provas escritas objetivas, resultado provisório da prova escrita discursiva e convocação para a prova oral (todos os cargos de Tecnologista) e para a defesa pública de memorial (cargos de Tecnologista Pleno 2, 3 e Sênior): Diário Oficial da União e Internet.

IV **26 e 27/2/2009** – Recursos (prova escrita discursiva): exclusivamente no Sistema Eletrônico de Interposição de Recurso, Internet, mediante instruções e formulários que estarão disponíveis nesse sistema.

V **7 e 8/3/2009** – Realização da prova oral e defesa pública de memorial.

OBSERVAÇÕES

- Não serão objeto de conhecimento recursos em desacordo com o item 12 do Edital n.º 2/2008, de 18/8/2008.
- Informações adicionais: telefone 0(XX) 61 3448-0100; Internet – www.cespe.unb.br.
-

cespeUnB
Centro de Seleção e de Promoção de Eventos

CONHECIMENTOS ESPECÍFICOS

Acerca de arquiteturas de sistemas, julgue os seguintes itens.

26 No modelo de arquitetura cliente-servidor, processos clientes interagem com processos servidores para acessar serviços. Os servidores podem ser clientes de outros servidores. Alguns serviços são prestados por processos servidores que podem ser executados em vários computadores hospedeiros.

27 Nos sistemas distribuídos *peer-to-peer*, não há distinção entre os processos clientes e os servidores. Os processos envolvidos em uma tarefa interagem cooperativamente como pares, sem distinção entre clientes e servidores. Há sistemas *peer-to-peer* nos quais diversos processos são executados em diferentes máquinas e *middleware* gerenciam os recursos distribuídos.

28 Em um sistema classificado como de tempo real, tipicamente há requisitos rígidos quanto à operação do mesmo. Alguns desses sistemas são classificados como *hard*, outros como *soft*. Os sistemas da categoria *hard* servem para garantir que as tarefas críticas serão completadas em tempo e, embora façam pouco uso de memória secundária, frequentemente empregam memória virtual.

No que se refere a bancos de dados, julgue os itens de **29** a **34**.

29 No modelo relacional, um conjunto de atributos FK do esquema da relação é uma chave estrangeira de R1 que faz referência à relação R2 se satisfizer as seguintes condições: os atributos de FK têm os mesmos domínios dos atributos da chave primária PK de R2; um valor de FK em uma tupla t1 do estado corrente de R1 é igual a um valor de PK para uma tupla t2 no estado corrente de R2 ou é *null*.

30 O modelo relacional representa os bancos de dados como coleções de relações, em que cada relação é um conjunto de tuplas. Em uma relação, duas tuplas podem ter a mesma combinação de valores para todos os seus atributos. Em um esquema de relação, tem-se o nome da relação e sua lista de atributos. Um esquema de banco de dados é um conjunto de esquemas de relações.

31 No modelo relacional, as chaves primárias podem ter o valor *null*, o esquema de uma relação pode ter mais de uma chave candidata e esta pode ser indicada como chave primária, e uma chave estrangeira pode se referir à sua própria relação. Se SK for uma superchave do esquema de relação R, então duas tuplas distintas em qualquer estado dessa relação R podem ter o mesmo valor de SK.

32 No mapeamento do modelo orientado a objetos para o relacional, cada associação muitos-para-muitos pode ser mapeada para tabelas distintas, mas deve-se, sempre que possível, combinar várias classes e relacionamentos em uma única tabela. Cada classe de associação pode ser mapeada para tabelas distintas e, tanto na herança simples quanto na múltipla, tabelas distintas podem ser usadas para cada superclasse e subclasse.

33 Há sistemas de gerenciamento de banco de dados (SGBDs) orientados a objetos que identificam cada objeto via um identificador único (OID). Nesses casos, cada OID é tipicamente imutável, isto é, o valor de um OID não deve ser modificado. Por isso, o valor de um OID normalmente não é igual ao endereço físico de armazenamento do objeto e independe de valores modificáveis de atributos do objeto.

34 Há SGBDs que empregam um protocolo de efetivação em duas fases (*two phase commit*) para evitar a ocorrência de problemas caso máquinas falhem durante o processamento de transações distribuídas. Esse protocolo garante que, se a falha ocorrer na primeira fase da transação, esta seja recuperada e efetivada; se ocorrer na segunda fase, a transação seja revertida (*rollback*).

Julgue os itens de **35** a **42**, referentes a redes de computadores.

35 No modelo OSI, os protocolos na camada de transporte controlam a comunicação entre aplicações, possibilitando o estabelecimento, a gerência e o término de conexões entre aplicações cooperantes. Os protocolos na camada de rede, por sua vez, são responsáveis pela recuperação de erros na comunicação fim-a-fim e pelo controle de fluxo durante a comunicação.

36 No modelo OSI, protocolos na camada de aplicação são responsáveis por prover, às aplicações, independência quanto à representação dos dados. Esses protocolos podem prover uma representação para ser usada na transferência de informação e mecanismos para tradução entre as representações locais e aquela usada para a troca de informação. As facilidades providas por protocolos de aplicação incluem a criptografia e a compressão de dados.

37 No endereçamento IPv4 em redes TCP/IP, as redes que empregam a numeração 10.0.0.0 podem usar um serviço de tradução de endereços (NAT) para que suas máquinas possam acessar a Internet. Definindo-se máscaras apropriadas, uma rede pode ser dividida em sub-redes e as sub-redes, por sua vez, podem ser divididas em outras sub-redes. Endereços na classe D visam possibilitar que datagramas sejam enviados para grupos de máquinas.

38 Nas redes TCP/IP, a tecnologia *classless inter-domain routing* (CIDR) possibilita a implementação de uma estrutura hierárquica de redes com diferentes tamanhos. Essa tecnologia tende a reduzir as entradas nas tabelas de roteamento, pois possibilita, via agregação de rotas (*routing aggegation*), que cada entrada represente rotas para várias redes.

39 Em redes TCP/IP, números de portas bem conhecidos são atribuídos aos servidores e qualquer número de porta livre pode ser atribuído para cada processo cliente. O TCP e o UDP usam a mesma faixa de números de portas, mas, em teoria, enquanto o TCP usa um dado número de porta para se comunicar com uma aplicação, esse mesmo número de porta pode ser usado pelo UDP para se comunicar com outra aplicação.

40 A recepção das mensagens TCP é confirmada pelo protocolo, ao passo que as aplicações UDP podem ser responsáveis por identificar datagramas perdidos e retransmiti-los. O UDP controla o fluxo de dados transmitidos, enquanto o TCP não o faz. O TCP provê um serviço de comunicação orientado a conexões e o UDP provê um serviço não-orientado a conexões.

41 Depois de estabelecida uma conexão TCP, uma conexão de controle FTP é criada e por meio dela são transmitidos os comandos FTP. Para que os dados sejam transferidos, são estabelecidas conexões de dados, sendo que uma mesma conexão TCP é usada para várias conexões de dados. Sempre que uma transferência de dados é encerrada, a conexão de dados é mantida, sendo reaproveitada em futuras transferências de dados.

42 Um cliente inicia uma sessão HTTP estabelecendo uma conexão TCP com o servidor com o qual deseja se comunicar. Por meio de mensagens de requisição HTTP enviadas para esse servidor, o cliente especifica o tipo de ação desejada. Nessas mensagens, há a linha de requisição (*request line*), na qual é identificado o método de ação desejado, o URI do recurso sobre o qual a ação é aplicada e a versão do HTTP que o cliente está usando.

Acerca do SQL Server, julgue os itens subsequentes.

43 O comando Transact-SQL chamado `CREATE DATABASE` pode ser usado para criar um banco de dados e arquivos para armazená-lo. Quando da execução do comando, é possível informar os tamanhos máximos dos arquivos em disco usados para armazenar dados (*data file*) e *log* (*log file*). Podem existir múltiplos arquivos de dados, mas apenas um arquivo de *log*. Quando o tamanho máximo do arquivo de *log* não é informado, a este é atribuído automaticamente o valor de 100 *megabytes*.

44 No SQL Server, restrições podem ser aplicadas a tabelas ou a colunas. Entre as restrições suportadas, tem-se: `NOT NULL`, que especifica que não são aceitos valores *null* na coluna; `DOMAIN`, que define os domínios dos valores armazenados nas colunas; `UNIQUE`, que especifica que, nas colunas identificadas, não pode haver linhas com os mesmos valores não-nulos; `PRIMARY KEY`, que informa a coluna, ou conjunto de colunas, que identificam as linhas na tabela.

45 O comando Transact-SQL chamado `CREATE INDEX` é usado para criar índices em uma tabela. Ao se executar esse comando, `UNIQUE` informa não ser possível que duas linhas tenham o mesmo valor de índice. Quando há um índice `UNIQUE` definido, os comandos `UPDATE` e `INSERT`, que gerariam linhas com valores de índice duplicados, passam a ser revertidos (*rollback*). O dono de uma tabela pode criar índices apenas quando essa tabela não tiver dados armazenados.

46 Os comandos Transact-SQL chamados `COMMIT TRANSACTION` e `ROLLBACK TRANSACTION` terminam transações iniciadas com `BEGIN TRANSACTION`. O comando `ROLLBACK TRANSACTION` pode retornar a execução de uma transação para o seu início ou para um ponto que tenha sido definido executando-se o comando `SAVE TRANSACTION`.

47 O comando Transact-SQL chamado `CREATE VIEW` cria uma tabela virtual que possibilita o acesso a dados em uma ou mais tabelas. Já o comando `SELECT` define uma visão que pode acessar apenas uma única tabela ou visão. Gatilhos `INSTEAD OF` podem ser criados para tornar uma visão atualizável.

48 Um procedimento armazenado (*stored procedure*) é criado executando-se o comando Transact-SQL chamado `CREATE PROCEDURE`. Pode-se executar um procedimento armazenado por meio do comando `EXECUTE`. Quando da execução desse comando, é possível passar parâmetros para o procedimento na forma `@parameter_name = value` ou na ordem em que foram listados no `CREATE PROCEDURE`.

49 Um gatilho pode ser dos tipos `INSTEAD OF` ou `AFTER`. Os gatilhos `AFTER` podem ser definidos em tabelas ou visões, enquanto os gatilhos `INSTEAD OF` só podem ser definidos para tabelas. Pode haver somente um gatilho `AFTER` para cada tipo de ação de gatilhamento `INSERT`, `UPDATE` ou `DELETE`, mas pode haver vários gatilhos `INSTEAD OF` para cada um desses tipos de ação.

```
package classes;

public class Canvas {
    private static final int MAXIMO = 100;
    private boolean[][] pixels;
    public Canvas(int largura, int altura) {
        if (largura > MAXIMO || altura > MAXIMO)
            throw new IllegalArgumentException();
        pixels = new boolean[largura][altura];
        for (int i=0; i < largura; i++)
            for (int j=0; j < altura; j++)
                pixels[i][j]=false;
    }
    void setPixel(int x, int y, boolean valor) {
        if (x > pixels.length || y > pixels[0].length )
            throw new IllegalArgumentException();
        pixels[x][y] = valor;
    }
    boolean getPixel(int x, int y){
        if (x > pixels.length || y > pixels[0].length )
            throw new IllegalArgumentException();
        return pixels[x][y];
    }
}

package classes;

public class Main {
    public static void main(String[] args) {
        try{
            Canvas canvas = new Canvas(10,100);
            canvas.setPixel(8, 30, true);
            System.out.println(canvas.getPixel(20,30));
        }
        catch(Exception exp){
            System.out.println("Excecao interceptada.");
        }
    }
}
```

A respeito do código Java apresentado, julgue os itens **50** e **51**.

50 Há uma linha de código na qual uma instância da classe `Canvas`, que não é abstrata, é criada. O atributo `MAXIMO` é uma constante e a memória para a matriz `pixels` é alocada no construtor de `Canvas`. Pode-se invocar os métodos `setPixel` e `getPixel` a partir de métodos em classes armazenadas no mesmo pacote no qual encontra-se `Canvas`.

51 É apresentada a mensagem `Excecao interceptada` quando da execução do método `Main`. É atribuído o valor `false` a todos os membros da matriz no construtor da classe `Canvas`. O enunciado `pixels.length` não resulta no tamanho total da matriz `pixels`.

```
package controladoras;
import comandos.Comando;
import java.sql.*;

public class Controladora implements IPersistencia {
    private String url, usuario, senha;
    private Connection conexao;
    public Controladora(String url, String usuario, String senha){
        this.url = url;
        this.usuario = usuario;
        this.senha = senha;
    }
    public void executar(Comando comando) throws SQLException {
        if (conexao == null || conexao.isClosed())
            conexao = DriverManager.getConnection(url, usuario, senha);
        comando.executar();
    }
    public void encerrar() throws SQLException {
        if (conexao == null) throw new IllegalArgumentException();
        conexao.close();
    }
    public void setDriver(String driver) throws ClassNotFoundException {
        Class.forName(driver);
    }
}

package comandos;
import java.sql.*;

public abstract class Comando {
    protected String comando;
    protected Connection conexao;
    public Comando(Connection conexao) {
        this.conexao = conexao;
    }
    public abstract void executar() throws SQLException;
}


package entidades;

public class Pedido {
    String item;
    public Pedido(String item){
        this.item = item;
    }
    public String getItem() {
        return item;
    }
}
```

Considerando esse código Java, julgue os itens de **52** a **55**

52 No construtor da classe `Controladora`, a palavra `this` possibilita acessar os quatro atributos de instância dessa classe. No método `executar`, a exceção `IllegalArgumentException` pode ser lançada mesmo não tendo sido listada após a palavra `throws` na declaração do método.

53 Na classe `Comando`, da qual não é possível criar objetos, a visibilidade do atributo `conexao` possibilita que o mesmo seja acessado a partir de códigos em subclasses de `Comando`. A visibilidade da classe `Pedido` é `public`, portanto, a visibilidade de `item` também é `public`. As classes estão armazenadas em diferentes pacotes.

54 Acerca da classe `ComandoInserir` apresentada a seguir, é correto afirmar que o método construtor da classe `Comando` é invocado, que o código declara incorretamente a classe `ComandoInserir`, pois a mesma é abstrata, e que há enunciados no método executar que podem lançar a exceção `SQLException`.

```
package comandos;
import entidades.Pedido;
import controladoras.*;
import java.sql.*;

public class ComandoInserir extends Comando {
    public ComandoInserir(Pedido p, Connection c) {
        super(c);
        comando  = "INSERT INTO PEDIDO VALUES(" + "'" + p.getItem() + "')";
    }
    public void executar() throws SQLException {
        Statement st = conexao.createStatement();
        st.executeUpdate(comando);
        st.execute("commit;");
    }
}
```

55 O código apresentado a seguir declara corretamente a interface implementada por `Controladora`

```
package controladoras;
import comandos.Comando;
import java.sql.*;
public interface IPersistencia {
    void encerrar() throws SQLException;
    void setDriver(String driver) throws ClassNotFoundException;
    void executar(Comando comando) throws SQLException;
}
```

No que se refere a JavaBeans, julgue os itens a seguir.

56 Quando uma propriedade `constrained` é modificada, os `beans` interessados são notificados via uma exceção `PropertyChangeEvent`. Por sua vez, a modificação de uma propriedade `bound` pode ser vetada por `beans` interessados. Para que um `bean` vete a modificação de uma propriedade `bound`, ele deve lançar uma exceção da classe `PropertyVetoException`.

57 A API JavaBeans provê *design patterns* que possibilitam que ferramentas descubram quais eventos cada `bean` pode notificar. Para um `bean` ser uma fonte de evento, ele deve prover métodos por meio dos quais possam ser acrescentados e removidos os identificadores dos objetos interessados no evento. Os *design patterns* definidos para esses métodos são os seguintes.

```
public void add<EventListenerType>(<EventListenerType> a)
public void remove<EventListenerType>(<EventListenerType> a)
```

Acerca de JDBC, julgue os itens subsequentes.

58 Em java.sql, o método `getConnection`, em `DriverManager`, pode ser invocado para se estabelecer uma conexão; o método `executeQuery`, em `Statement`, retorna uma instância de `ResultQuery`; e a interface `ResultSetMetaData` contém métodos por meio dos quais pode-se obter informações tais como tipos e propriedades de colunas retornadas por consultas.

59 Em java.sql, métodos em `DatabaseMetaData` possibilitam acessar informações sobre bancos de dados. O método `getMetaData`, em `Connection`, retorna um objeto `DatabaseMetadata`, e o método `getSchemas`, em `DatabaseMetaData`, retorna uma instância de `Schema`.

Julgue os itens seguintes, que se referem a programação concorrente em Java.

60 É possível controlar a execução concorrente de métodos usando-se a palavra-chave `synchronized` nas declarações desses métodos, ao se declarar construtores e métodos estáticos. Em uma classe com métodos `syncronized`, um mesmo *lock* controla o acesso aos atributos estáticos e às instâncias da classe.

61 O método `Thread.start` pode ser usado para executar um *thread*. Já o método `Object.wait` suspende a execução do *thread* corrente. Este método pode ser invocado em um *loop* em que se teste a condição aguardada, mas não se pode invocá-lo dentro de métodos `synchronized`. O método `Thread.sleep` suspende o *thread* corrente por um período de tempo.

Acerca de JSP, julgue os itens que se seguem.

62 São possíveis os seguintes componentes em uma página JSP: código de apresentação no formato HTML, diretivas usadas para instruir o compilador quando da tradução da página, *scriptlets* compostos por segmentos de código Java, e ações usadas para, por exemplo, interagir com componentes JavaBean.

63 Em uma página JSP, ações são definidas usando-se XML e são executadas quando da compilação da página; o nome de uma ação identifica o tipo de ação a ser executada. Por exemplo, a ação `<jsp:useBean>` associa um objeto JavaBean a um identificador, enquanto a ação `<jsp:forward>` despacha a solicitação para um outro recurso no contexto da aplicação.

```
#include <iostream>
#include <stdexcept>

using namespace std;

class Conta {
protected:
    double saldo;
public :
    Conta(double valor) throw (invalid_argument){
        if (valor < 0) throw  invalid_argument("Erro.");
        saldo = valor;
    }
    double getSaldo() const {return saldo;}
    void depositar(double) throw (invalid_argument);
    virtual void sacar(double) throw (invalid_argument) = 0;
};

void Conta::depositar(double valor) throw (invalid_argument){
    if (valor < 0) throw  invalid_argument("Erro.");
    saldo = saldo + valor;
}

class ContaEspecial:public Conta {
        double limite;
public:
    ContaEspecial(double valor,double limite)
        throw (invalid_argument):Conta(valor){
    this->limite = limite;
    }
    void depositar(double) throw (invalid_argument);
    void sacar(double valor) throw (invalid_argument);
};

void ContaEspecial::sacar(double valor) throw (invalid_argument){
    if (valor < 0 || valor > limite)
        throw  invalid_argument("Erro.");
    saldo = saldo - valor;
}
```

Tendo como referência o trecho de código C++ acima, julgue os itens a seguir.

64 A classe `Conta` não pode ser instanciada. O código na classe `ContaEspecial` pode acessar o atributo saldo da classe `Conta`. O construtor da classe `Conta` é invocado quando a classe `ContaEspecial` é instanciada. Há métodos `inline` na classe `Conta`.

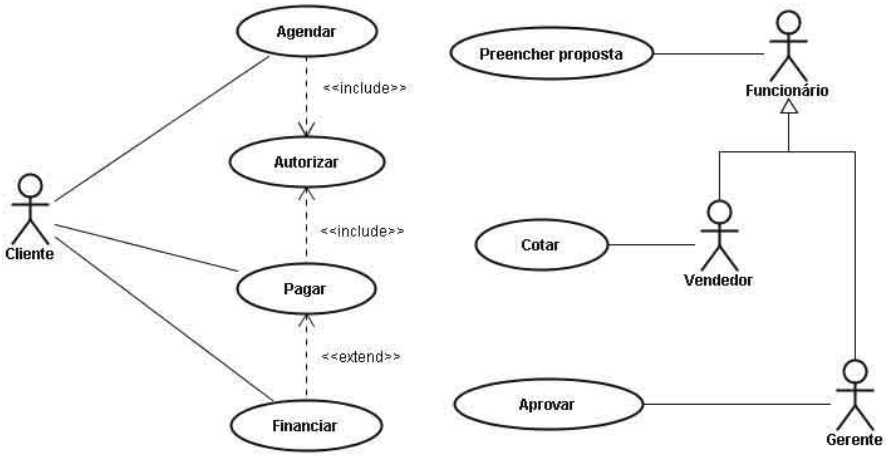
65 A visibilidade do atributo `limite` não permite que o mesmo seja acessado em código que não esteja na classe `ContaEspecial`. As assinaturas dos métodos que lançam exceções estão incorretas, pois é usada a palavra `throw` em vez de `throws`.

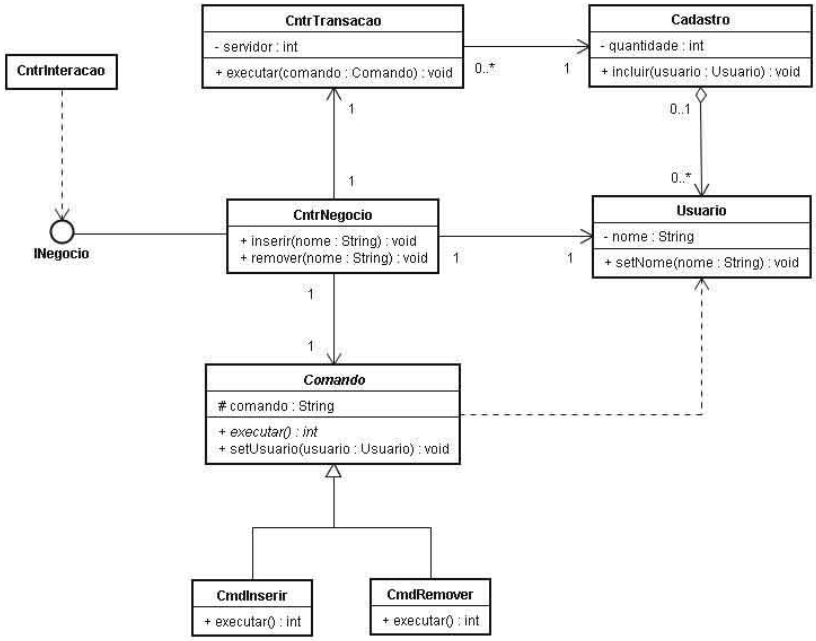
66 A execução da seguinte função imprime o valor 50.

```
nt main(int argc, char *argv[]) {
    Conta contaA(10);
    contaA.depositar(40);
    cout << contaA.getSaldo() << "\n";
}
```

67 A execução da seguinte função imprime o valor 130.

```
int main(int argc, char *argv[]) {
    Conta *ptr = new ContaEspecial(100,50);
    ptr->depositar(40);
    ptr->sacar(10);
    cout << ptr->getSaldo();
    delete ptr;
}
```



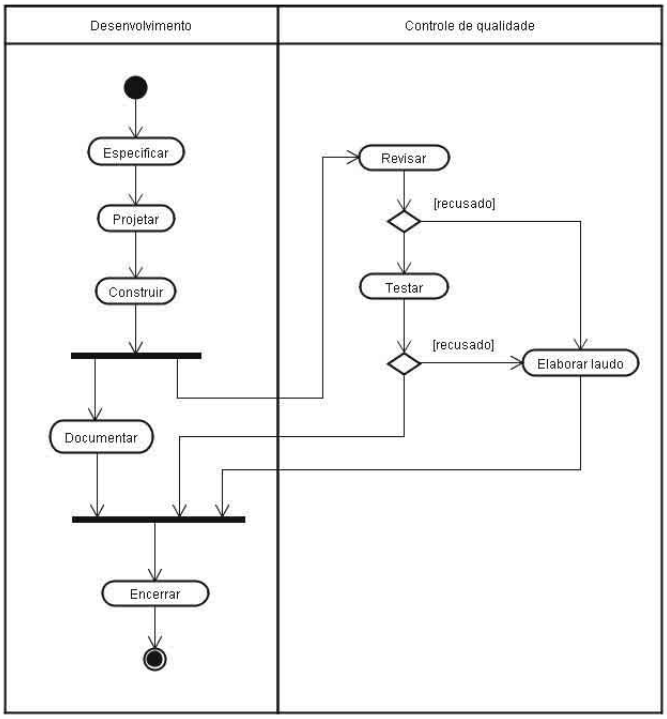


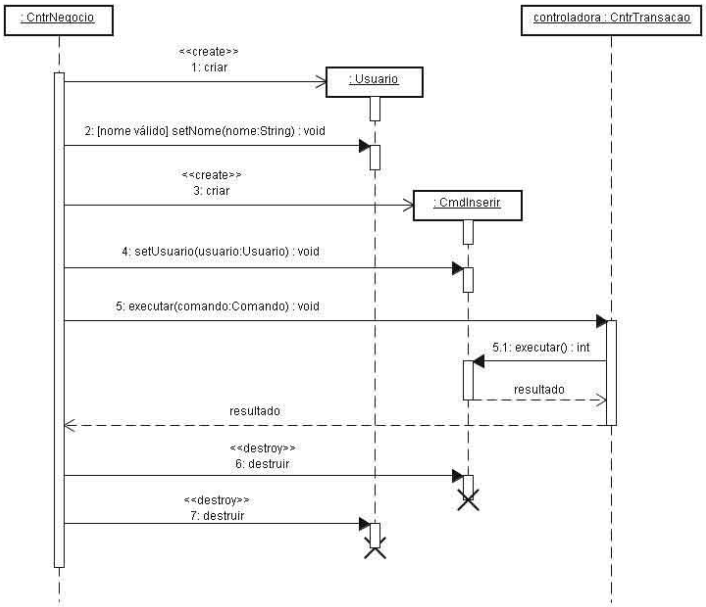
Quanto aos diagramas UML mostrados acima, julgue os próximos itens.

68 Quanto ao diagrama de casos de uso, as seguintes afirmações estão corretas: os casos Agendar e Pagar contêm trechos em comum; o caso Financiar é não abstrato; a ausência de relacionamento entre um ator e o caso Autorizar não configura um erro; o ator Gerente herda a associação com o caso Preencher proposta, mas não com o caso Cotar.

69 No que se refere ao diagrama de classes, é correto afirmar que o atributo comando da classe Comando pode ser acessado no método executar da classe CmdInserir; a classe CmdRemover não é abstrata; a classe Comando é abstrata e não pode ser instanciada; há uma herança simples; a relação entre Comando e Usuario é de dependência.

70 Acerca do diagrama de classes, é correto afirmar que a classe CntrNegocio implementa a interface INegocio; podem existir instâncias de Usuario mesmo que não existam instâncias de Cadastro; a cada instância de Cadastro podem estar ligadas várias instâncias de CntrTransacao; o atributo nome não pode ser diretamente acessado fora da classe Usuario; o método remover pode ser invocado fora da classe CntrNegocio.





Tendo como referência os diagramas UML mostrados acima, julgue os itens que se seguem.

71 Com relação ao diagrama de atividades, as seguintes afirmações estão corretas: há condições de guarda especificadas nos desvios; as atividades Documentar e Revisar podem ser concorrentes; as atividades Projetar e Elaborar laudo estão em raias distintas; quando a atividade Encerrar for executada, as atividades Documentar e Revisar terão sido executadas; a barra de separação (*fork*) contém uma transição de entrada e duas de saída.

72 Quanto ao o diagrama de sequência, é correto afirmar que dois dos objetos existiam antes da interação descrita no diagrama; controladora é o nome de uma instância da classe CntrTransacao; a instância da classe Usuario é anônima; uma mensagem assíncrona é enviada da instância da classe CntrNegocio para a instância da classe CntrTransacao; ao final da interação descrita, existem apenas duas instâncias de classes.

Acerca do ciclo de desenvolvimento de sistemas, julgue os seguintes itens.

73 No modelo de desenvolvimento em cascata, a especificação e a análise de requisitos, o projeto, a implementação, o teste, a operação e a manutenção são possíveis fases em um ciclo de vida. Cada fase produz artefatos e a fase seguinte não deve começar antes que a anterior tenha terminado. Esse modelo pode ser usado quando os requisitos são bem compreendidos e há pouca chance de mudanças radicais durante o desenvolvimento.

74 O modelo de desenvolvimento evolucionário pode-se basear no desenvolvimento de uma versão inicial que é refinada em várias versões até chegar ao sistema adequado ou em protótipos descartáveis que são construídos visando a compreensão dos requisitos. Atividades relacionadas a especificação, desenvolvimento e validação são intercaladas, em vez de separadas em fases. Esse modelo pode produzir sistemas mal-estruturados devido às mudanças contínuas.

75 No modelo de desenvolvimento iterativo, um ciclo de vida pode ser dividido em fases e estas em iterações, que produzem incrementos. Uma das características desse modelo é o fato de a especificação ser desenvolvida em conjunto com o *software*. Tipicamente, não há uma especificação completa do sistema até o incremento final ser especificado.

PROVA DISCURSIVA

- Nesta prova, que vale **vinte e cinco** pontos, faça o que se pede, usando o espaço para rascunho indicado no presente caderno. Em seguida, transcreva o texto para a **FOLHA DE TEXTO DEFINITIVO DA PROVA ESCRITA DISCURSIVA**, no local apropriado, pois **não será avaliado fragmento de texto escrito em local indevido**.
- Qualquer fragmento de texto além da extensão máxima de **trinta** linhas será desconsiderado.
- Na **folha de texto definitivo**, identifique-se apenas no cabeçalho da primeira página, pois **não será avaliado** texto que tenha qualquer assinatura ou marca identificadora fora do local apropriado.
- Quando comunicado pelo aplicador o número do tema sorteado, preencha com esse número, obrigatoriamente, o campo denominado TEMA SORTEADO de sua FOLHA DE TEXTO DEFINITIVO DA PROVA ESCRITA DISCURSIVA e acerca do qual você redigirá a sua PROVA ESCRITA DISCURSIVA.

TEMA 1– Arquitetura de sistemas

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- sistema cliente-servidor;
- sistemas de tempo real;
- sistemas distribuídos.

TEMA 2– Sistema gerenciador de banco de dados (SGBD)

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- estrutura de dados: lista, pilha e fila, estrutura de árvore, grafos;
- estrutura de um SGBD;
- tipos de banco de dados.

TEMA 3– Modelos de banco de dados

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- modelo conceitual;
- modelo lógico;
- processo de modelagem.

TEMA 4– Linguagem SQL

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- linguagem de manipulação, definição, controle e consulta de dados (DML, DDL, DCL, DQL);
- cláusulas;
- operadores lógicos e relacionais.

TEMA 5– Banco de dados relacional (BDR)

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- normalização em BD;
- modelo e diagrama de entidade de relacionamento (MER, DER);
- integridade referencial.

TEMA 6– Banco de dados distribuídos

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- arquitetura básica;
- processamento de consultas distribuídas;
- transações e controle de concorrência.

TEMA 7– Processamento paralelo

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- arquiteturas de *hardware* para processamento paralelo;
- sistemas de *software* para processamento paralelo;
- sistemas distribuídos.

TEMA 8– Programação orientada a objetos

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- criação de classes, objetos, métodos e variáveis, pacotes, reúso com herança e composição, operadores e controle de fluxo;

- programação Java;
- programação C++.

TEMA 9– Desenvolvimento de sítios na Internet

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- hipertexto e padrão HTML;
- páginas estáticas e dinâmicas;
- linguagens para aplicativos na Web.

TEMA 10 – Redes de computadores

Ao elaborar seu texto, aborde, necessariamente, os seguintes tópicos:

- arquiteturas de redes de computadores;
- modelo OSI;
- protocolos de comunicação TCP/IP, FTP e HTTP.

RASCUNHO

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	