Ministério das Comunicações

PROCESSO SELETIVO **NÍVEL SUPERIOR**

Aplicação: 29/11/2008

CADERNO DE PROVAS – PARTE II

CONHECIMENTOS ESPECÍFICOS

NÍVEL III

ÁREA DE FORMAÇÃO 7:

INFORMÁTICA/ANÁLISE DE SISTEMAS

ATENÇÃO!

- Leia atentamente as instruções constantes na capa da Parte I do seu caderno de provas.
- Nesta parte do seu caderno de provas, que contém os itens relativos à prova objetiva de **Conhecimentos Específicos**, confira o nível, o número e nome de sua área de formação e o seu nome transcritos acima, no rodapé de cada página numerada desta parte do caderno de provas e na **folha de respostas**.
- A duração das provas é de **três horas e trinta minutos**, já incluído o tempo destinado à identificação — que será feita no decorrer das provas — e ao preenchimento da folha de respostas.

AGENDA (datas prováveis)

I 2/12/2008, após as 19 h (horário de Brasília) – Gabaritos oficiais preliminares das provas objetivas: Internet — www.cespe.unb.br.

II 3 e **4/12/2008** – Recursos (provas objetivas): exclusivamente no Sistema Eletrônico de Interposição de Recurso, Internet, mediante instruções e formulários que estarão disponíveis nesse sistema.

III 7/1/2009 – Resultado final das provas objetivas: Diário Oficial da União e Internet.

OBSERVAÇÕES

- Não serão objeto de conhecimento recursos em desacordo com o item 12 do Edital n.º 1 – MC, de 23/9/2008.
- Informações adicionais: telefone 0(XX) 61 3448-0100; Internet – www.cespe.unb.br.
-

cespe UnB
Centro de Seleção e de Promoção de Eventos

De acordo com o comando a que cada um dos itens de **51** a **120** se refira, marque, na **folha de respostas**, para cada item: o campo designado com o código **C**, caso julgue o item **CERTO**; ou o campo designado com o código **E**, caso julgue o item **ERRADO**. A ausência de marcação ou a marcação de ambos os campos não serão apenadas, ou seja, não receberão pontuação negativa. Para as devidas marcações, use a **folha de respostas**, único documento válido para a correção das suas provas.

CONHECIMENTOS ESPECÍFICOS

Acerca de segurança da informação, julgue os itens subseqüentes.

51 Disponibilidade, integridade e confidencialidade são princípios de segurança. A fim de garantir integridade, os dados e os recursos devem ser protegidos, evitando que sejam alterados de modo não-autorizado. Para garantir confidencialidade, os dados e os recursos devem ser protegidos, a fim de que os mesmos só sejam disponíveis aos indivíduos, programas ou processos autorizados.

52 A análise de riscos é parte da gerência de riscos e nela podem ser realizadas as seguintes atividades: identificar os recursos (*assets*), determinar os valores dos recursos, identificar vulnerabilidades e ameaças (*threat*), quantificar a probabilidade de cada ameaça ocorrer, estimar as perdas associadas às ameaças, estimar a freqüência de ocorrência das ameaças, reduzir, transferir ou aceitar riscos.

53 Na criptografia simétrica, o emissor e o receptor usam duas instâncias da mesma chave para cifrar e decifrar. Nesse tipo de criptografia, a chave deve ser mantida em segredo e protegida, pois a posse da chave possibilita decifrar mensagens cifradas com essa chave. Esse tipo de criptografia provê autenticação, pois, se duas pessoas usam a mesma chave, há como provar quem enviou cada mensagem.

54 Na criptografia assimétrica, para garantir a confidencialidade de uma mensagem, quem envia a mensagem deve cifrá-la com a chave privada do destinatário da mensagem. Se a autenticação for o serviço desejado por quem envia a mensagem, este deve cifrá-la com sua chave pública.

55 Na criptografia assimétrica, dados cifrados usando-se uma chave privada podem ser decifrados usando-se uma chave privada; dados cifrados usando-se uma chave pública podem ser decifrados usando-se uma chave pública; dados cifrados usando-se uma chave privada podem ser decifrados usando-se a correspondente chave pública; dados cifrados usando-se uma chave pública podem ser decifrados usando-se a correspondente chave pública.

56 Para criar uma assinatura digital para uma mensagem, pode-se usar uma função *hash* para calcular um valor a partir do conteúdo da mensagem e criptografar esse valor usando-se a chave pública de quem enviou a mensagem. A partir desse valor é possível, na recepção, verificar por quem a mensagem foi remetida e se a mensagem recebida é diferente da enviada.

57 Um certificado digital de um indivíduo, emitido por certificadora, contém a chave privada do indivíduo junto com outras informações de identificação.

58 Na escolha de senhas, constitui boa prática: usar tanto letras maiúsculas quanto minúsculas; incluir dígitos; combinar palavras, visando facilitar a memorização; usar senhas diferentes em diferentes máquinas; não reutilizar uma senha quando tiver de trocá-la; não usar palavras que estejam em dicionários; evitar informações pessoais.

Acerca do CMMI, julgue os seguintes itens.

59 Os modelos CMMI descrevem níveis de melhoria de processos denominado níveis de maturidade, e apresentam uma ordem para a melhoria dos processos em estágios. Nesses níveis, têm-se as áreas de processos, em que são definidos objetivos e práticas. Cada nível procura estabilizar parte dos processos empregados na organização. São possíveis o seguintes níveis de maturidade: inicial, gerenciado (*managed*), definido, quantitativamente gerenciado e otimizado.

60 A área de processo denominada garantia da qualidade de produto (*product quality assurance*) visa os seguintes objetivos: avaliar se processos, produtos e serviços estão de acordo com os processos, padrões e procedimentos esperados; identificar e documentar aquilo que não esteja aderindo ao esperado; registrar as atividades de garantia da qualidade; informar gerentes e outros membros do projeto sobre os resultados obtidos; estabelecer e manter critérios claros para as avaliações.

61 A área de processo denominada análise causal e resolução (*causal analysis* e *resolution*) encontra-se definida no nível de maturidade otimizado (*optimizing*) e objetiva identificar as causas dos defeitos e outros problemas e tomar ações para evitar que defeitos e problemas se repitam no futuro.

Acerca do MPS.BR, julgue os itens de **62** a **66**.

62 Um componente do MPS.BR é o modelo de referência, que contém os requisitos que os processos das unidades organizacionais devem atender para estar em conformidade com o modelo MPS.BR. Nesse modelo, são definidos níveis de maturidade que estabelecem patamares de evolução de processos em sete níveis de maturidade, cuja escala se inicia no nível de menor maturidade A e progride até o de maior maturidade G.

63 A capacidade do processo expressa o grau de refinamento e institucionalização com que o processo é executado na organização. A capacidade é representada por um conjunto de atributos de processo descrito em termos de resultados esperados. Uma vez que níveis de maturidade são acumulativos, a organização está no nível F, esta possui o nível de capacidade do nível F que inclui os atributos dos níveis G e F para os processos relacionados no nível F.

64 São processos no nível de maturidade E: verificação, validação, projeto e construção do produto. São processos no nível D: gerência de riscos, desenvolvimento para reutilização, análise de decisão e resolução. São processos no nível F: garantia da qualidade, gerência de configuração, medição. São processos no nível G: gerência de requisitos, gerência de projetos.

65 O processo denominado desenvolvimento para reutilização (DRU) reúne os componentes do produto, gerando um modelo integrado consistente com o projeto e demonstra que os requisitos funcionais são satisfeitos para o ambiente alvo ou equivalente. O processo denominado integração do produto (ITP) projeta, desenvolve e implementa soluções para atender aos requisitos.

66 O processo denominado verificação (VER) prevê que um produto ou componente atenderá a seu uso pretendido quando colocado no ambiente para o qual foi desenvolvido. Por sua vez, o processo análise de causas de problemas e resolução (ACP) identifica causas de defeitos e de outros problemas, assim como objetiva agir de modo a prevenir suas ocorrências no futuro.

Acerca dos modelos de processos, julgue os itens a seguir.

67 No modelo de processo de desenvolvimento em cascata, ciclo de vida é dividido em uma série de fases. Na fase de projeto, é definida tipicamente a arquitetura do sistema, quando estilos de arquitetura e padrões de projeto podem ser empregados. A princípio, em cada fase, podem ser produzidos artefatos variados, por exemplo: planos, códigos e modelos.

68 No modelo de processo de desenvolvimento iterativo, o ciclo de vida pode ser dividido em iterações. Em cada interação pode ocorrer análise, projeto, implementação e teste. O desenvolvimento iterativo tende a reduzir riscos, pois os componentes do *software* são progressivamente integrados.

69 Os processos ágeis (*agile*) de desenvolvimento são tipicamente processos iterativos, em que *softwares* são desenvolvidos em uma série de incrementos. Nesses processos, os usuários finais tipicamente participam na validação dos incrementos, podendo propor alterações e novos requisitos.

Acerca do *rational unified process* (RUP), julgue os seguintes itens.

70 A fase de elaboração (*elaboration*) tem os seguintes objetivos: desenvolver um produto que esteja em condições de migrar para uma comunidade de usuários; completar análise, projeto, implementação e teste das funcionalidades providas pelo sistema; desenvolver material de suporte e treinamento para os usuários e criar os artefatos necessários para a implantação e uso do sistema.

71 A fase de transição (*transition*) tem os seguintes objetivos: planejar a implantação do sistema; integrar subsistemas de modo a produzir o sistema; executar testes beta para validar se o sistema atende às expectativas dos usuários; finalizar material de suporte e treinamento; corrigir defeitos identificados nos testes; treinar usuários e obter *feedback* dos usuários.

72 São objetivos da fase de concepção (*inception*): preparar ambiente para o projeto; elaborar plano para o projeto; definir escopo do sistema; identificar atores e casos de uso; identificar as necessidades dos *stakeholders*; definir níveis de prioridade dos casos de uso; propor arquitetura candidata; e definir objetivos do esforço de teste.

73 A fase de construção (*construction*) tem os seguintes objetivos: detalhar casos de uso e requisitos do *software*; refinar a arquitetura proposta e demonstrar que essa arquitetura suporta os requisitos do sistema; testar e avaliar protótipos visando demonstrar que os principais riscos foram avaliados; e construir protótipos executáveis para a avaliação da arquitetura proposta.

Acerca do *extreme programming* (XP), julgue os itens **74** e **75**.

74 As seguintes práticas são recomendadas pelo XP: dividir o projeto em iterações; iniciar cada iteração com o seu planejamento; empregar cartões CRC no projeto simplificar o projeto seguir padrões de codificação; freqüentemente, testar e integrar; executar testes de unidade; programar em pares; incentivar participantes a trabalharem em diferentes partes do projeto.

75 No XP, não é necessário detalhar o planejamento para todo o projeto; tal planejamento pode ser atualizado a cada iteração; estórias menos relevantes são realizadas primeiro; as estórias são divididas em tarefas de programação; cada tarefa deve ser extensa de modo a ser executada ao longo de várias iterações.

Acerca dos modelos de gestão, julgue os itens a seguir.

76 O grupo de processos de iniciação no PMBOK confirma se um projeto deve ser iniciado, concede aprovação para alocação de recursos ao projeto; o termo de abertura é uma possível saída desse grupo de processos, e para ser concluído, deve ser aprovado pelo patrocinador do projeto, gerência sênior e principais *stakeholders*.

77 O grupo de processos de planejamento no PMBOK formula e revisa as metas e os objetivos do projeto, elabora os planos a serem seguidos para se cumprir os objetivos do projeto. O plano de gerenciamento do projeto e a definição do escopo são saídas desse grupo de processos, sendo o processo de planejamento do escopo parte desse grupo de processos.

78 O grupo de processos de execução no PMBOK executa plano de projeto, envolve a obtenção e a alocação de recursos humanos, avalia e controla desempenho do projeto, mantém o plano sob controle e assegura a sincronização entre a execução e os objetivos visados e costuma absorver parte significativa do tempo e dos recursos do projeto.

79 O modelo de desenvolvimento bazar tem as seguintes características: para minimizar a exposição dos usuários a *softwares* defeituosos, evita disponibilizar novas versões com freqüência; para facilitar a gestão, não envolve os usuários durante o desenvolvimento; testes são realizados pelos próprios desenvolvedores, para evitar a participação de usuários; e, no início de um projeto, não exige código já implementado.

Acerca de bancos de dados, julgue os itens subseqüentes.

80 Ao projetar o esquema de um banco de dados relacional, para cada tipo de relacionamento binário N: M, é correto criar uma forma para representar o relacionamento e nela inserir as chaves primárias das relações que representam os tipos de entidade no relacionamento. Alternativamente, é correto representar esse tipo de relacionamento inserindo-se apenas uma chave estrangeira em uma das relações no relacionamento.

81 Ao projetar o esquema de um banco de dados relacional, para cada tipo de entidade forte E, de um esquema ER, pode-se criar uma relação R com todos os atributos simples de E. Além disso, para um relacionamento R binário 1:1, no qual participam os tipos entidades S e T, pode-se escolher uma das relações e nela inserir, como chave estrangeira, a chave primária da outra relação.

82 No MySQL, pode-se criar uma base de dados com o comando `CREATE DATABASE`; pode-se, ainda, selecionar uma base de dados para uso com o comando `USEDATABASE`. No MySQL, em um comando `SELECT`, pode-se testar o valor `NULL` com um dos seguintes operadores: =, < ou >.

83 No MySQL, pode-se usar o atributo `AUTO_INCREMENT` para gerar um identificador para linhas em uma relação. O programa `mysqlmanager` possibilita monitorar e gerenciar servidores MySQL, enquanto o comando `CREATE TABLE PARTITION` particiona uma tabela. No MySQL não se pode executar múltiplos servidores em uma mesma máquina.

Considere os seguintes trechos de código em Java.

Arquivo – I

```
import java.util.*;
import java.io.*;
public abstract class Cadastro implements Serializable {
    protected Vector cadastro;
    protected static final String msg = "Erro cadastro";
    public abstract Object pesquisar (Object objeto);
    public abstract void inserir (Object cliente);
    public Cadastro(int tamanho) {
        if (tamanho <= 0)
            throw new IllegalArgumentException(msg);
        cadastro = new Vector(tamanho);
        }
}
```

Arquivo – II

```
import java.io.IOException;
public class CadastroCliente extends Cadastro {
    public CadastroCliente (int tamanho) {
        super(tamanho);
    }
    public Object pesquisar (Object objeto) {
        String nomeLido;
        for (int n = 1; n < cadastro.size(); n++) {
            nomeLido = ((Cliente)cadastro.elementAt(n)).getNome();
            if (nomeLido.compareTo((String)objeto)==0)
            return (Cliente)cadastro.elementAt(n);
        }
        return null;
    }
    public void inserir (Object cliente) {
        cadastro.addElement(cliente );
    }
}
```

Arquivo – III

```
import java.io.*;
public class Cliente implements Serializable {
    private String nome;
    public String getNome () {
        return nome;
    }
    public void setNome (String nome) {
        this.nome = nome;
    }
}
```

A partir dos trechos de códigos java acima apresentados, julgue os seguintes itens.

84 A classe `Cadastro`: não pode ser instanciada; o seu atributo `cadastro` poder ser acessado por código em uma subclasse dessa classe; a assinatura do seu construtor está errada, pois não informa sobre o lançamento da exceção `IllegalArgumentException`; um de seus atributos é uma constante; os métodos `writeObject` e `readObject` da interface `Serializable` não estão implementados, por isso a classe é abstrata.

85 Acerca da classe `CadastroCliente` é correto afirmar que: essa classe implementa `Serializable`; a classe pode ser instanciada; o construtor da classe `Cadastro` é explicitamente invocado; as conversões entre tipos (*typecast*) estão erradas; o *loop* controlado pela instrução `for` apresenta limite incorreto; o atributo `nomeLido` é um atributo local ao método pesquisar; o construtor da classe não pode lançar exceção.

Acerca da linguagem JavaScript, julgue os itens **86** e **87**.

86 Considerando o seguinte trecho de código, é correto afirmar que: não há erros de sintaxe no código; ao se pressionar um dos botões, será apresentado um alerta informando que a matriz `b` está vazia; quando pressionado o outro botão, serão apresentados os elementos da matriz `a`.

```
<HTML><BODY>
<SCRIPT TYPE="text/javascript">
function imprimir(matriz){
    if(matriz.length == 0) {
        alert("Matriz vazia.");
    }
    else {
       var indice = 0;
       do {
           document.write(matriz[indice]);
       }
       while(++indice < matriz.length);
    }
}
</SCRIPT>
<SCRIPT TYPE="text/javascript">
var a = new Array();
var b = new Array();
a[0] = "Maria";
a[1] = "Jose";
a[2] = "Pedro";
</SCRIPT>
<FORM>
<INPUT TYPE=button VALUE="Matriz A" onClick="imprimir(a);">
<INPUT TYPE=button VALUE="Matriz B" onClick="imprimir(b);">
</FORM>
</BODY></HTML>
```

87 Considerando o seguinte trecho de código, é correto afirmar que: não há erros de sintaxe no código; nele, há dois métodos, um deles é um construtor; em cada objeto, há três atributos de instância; são criados dois objetos; as informações `23:0:0` e `10:21:0` serão apresentadas quando da execução do código.

```
<HTML><BODY>
<SCRIPT TYPE="text/javascript">
function apresentar(){
    if(this.segundos >= 59){
        this.segundos = 0;
        if(this. minutos >= 59){
            this. minutos = 0;
            if (this. horas >= 23)
                    this. horas = 0;
               else this. horas++;
        } else this. minutos++;
    } else this.segundos++;
    document.write(this. horas

    + ":" + this. minutos

    + ":" + this.segundos + "<BR>");
}

function Relogio( horas, minutos,segundos){
    this. horas = horas;

    this. minutos = minutos;
    this.segundos = segundos;
    this.apresentar = apresentar;
}

var relogioA = new Relogio(22,59,59);
var relogioB = new Relogio(10,20,59);
relogioA.apresentar();
relogioB.apresentar();
</SCRIPT>
</BODY></HTML>
```

Acerca da linguagem de programação C, julgue os itens subseqüentes.

88 Não há erros de sintaxe no seguinte código e a sua execução apresenta como resultado: `11`, `11` e `12`.

```
#include <stdio. h>
int funcA(int *valor){
 (*valor)++;
  return *valor;
}
int funcB(int valor){
    valor++;
    return valor;
}

int main(int argc, char *argv[]) {
    int (*ptrA)(int*), (*ptrB)(int);
    ptrA = funcA;
    ptrB = funcB;
    int a = 10, b, c;
    b = ptrA(&a);
    c = (*ptrB)(a);
  printf("%d %d %d", a, b, c);
}
```

89 Considerando o seguinte trecho de código, é correto afirmar que: não há erros de sintaxe; `pacientes` é uma matriz de estruturas; são acessados campos no primeiro elemento de pacientes; o campo `evento` é um ponteiro para a cadeia de caracteres `nascimento`; há estruturas aninhadas.

```
enum e_data {
    nascimento,
    morte
};

typedef struct {
   short dia;
   short mes;
   enum e_data evento;
} t_data;

typedef struct {
   char *nome;
   t_data data;
} t_pessoa;

int main(int argc, char *argv[]) {
    t_pessoa pacientes[1];
    pacientes[0].nome = "Maria";
    pacientes[0].data. dia = 25;
    pacientes[0].data.mes = 11;
    pacientes[0].data.evento = nascimento;
}
```

Acerca da linguagem de programação C++, julgue os itens de **90** a **92**.

90 Considerando o seguinte trecho de código, é correto afirmar que: não há erros de sintaxe; `Pessoa` tem só um método *inline*; `Correntista` é subclasse de `Pessoa`; na classe `Correntista`, a visibilidade de `matricula` é *public*, uma vez que a herança entre `Correntista` e `Pessoa` é *public*; cada classe tem um construtor; um método na classe `Correntista` pode lançar uma exceção.

```
#include <stdexcept>
using namespace std;

class Pessoa {
  int matricula;
public:
   Pessoa(int matricula):matricula(matricula){}
   void setMatricula(int matricula){
   this->matricula = matricula;
}
 int getMatricula();
};
inline int Pessoa::getMatricula(){
 return matricula;
}
class Correntista:public Pessoa {
   float saldo;
public:
   Correntista(int matricula, float saldo):Pessoa(matricula),saldo(saldo){}
   void depositar(float) throw (invalid_argument);
};
void Correntista::depositar(float valor) throw (invalid_argument){
  if ( valor < 0 )
  throw invalid_argument("Valor incorreto.");
  saldo += valor;
}
```

91 Considerando o seguinte código, é correto afirmar que: não há erros de sintaxe; a classe `Encomenda` não pode ser instanciada; o atributo `prioridade` pode ser acessado a partir de métodos nas subclasses de `Encomenda`; a execução do código imprime os valores `5` e `20`.

```
#include <iostream>
using namespace std;

class Encomenda {
protected:
  int prioridade;
public:
  virtual void setPrioridade(int) = 0;
  int getPrioridade(){return prioridade;}
};

class EncomendaNormal:public Encomenda {
   void setPrioridade(int valor) {
    prioridade = valor;
   }
};

class EncomendaUrgente:public Encomenda {
   void setPrioridade(int valor) {
    prioridade = valor + 10;
   }
};

int main(int argc, char *argv[]) {
   Encomenda *ptrA, *ptrB;
   ptrA = new EncomendaNormal();
   ptrA->setPrioridade(5);
   ptrB = new EncomendaUrgente();
   ptrB->setPrioridade(10);
   cout << ptrA->getPrioridade();
   cout << ptrB->getPrioridade();
}
```

92 Considerando o seguinte código, é correto afirmar que: não há erros de sintaxe; o código no método `depositar` da classe `Estado` pode acessar o atributo privado `saldo`; a execução do código imprime o valor `20`.

```
#include <iostream>
using namespace std;

class Estado;

class Conta {
   float saldo;
public:
   Conta(float saldo):saldo(saldo){}
   friend class Estado;
   float getSaldo(){return saldo;}
};

class Estado {
protected:
    Conta conta;
public:
    Estado(Conta conta):conta(conta){}
    void depositar(float valor);
};

void Estado::depositar(float valor){
    conta.saldo += valor;
}

int main(int argc, char *argv[]) {
   Conta conta(10);
   Estado estado(conta);
   estado.depositar(10);
   cout << conta.getSaldo();
}
```

Considere as seguintes tabelas.

Tabela CLIENTES com colunas CPF, NOME, TELEFONE e RENDA.

CPF	NOME	TELEFONE	RENDA
123	Jose	1111	R$ 2.000,00
234	Maria	2222	R$ 3.000,00
345	Paulo	3333	R$ 1.500,00
456	Rosa	4444	R$ 4.000,00

Tabela CLIENTES_EMPRESTIMOS com colunas EMPRESTIMO e CLIENTE.

EMPRESTIMO	CLIENTE
100	234
400	123
200	456
500	345
300	123
150	456

Tabela EMPRESTIMOS com colunas CODIGO, IMOVEL, VALOR e SALDO.

CODIGO	IMOVEL	VALOR	SALDO
100	20	R$ 100.000,00	R$ 50.000,00
200	10	R$ 200.000,00	R$ 90.000,00
400	25	R$ 150.000,00	R$ 85.000,00
300	15	R$ 350.000,00	R$ 70.000,00
500	30	R$ 250.000,00	R$ 10.000,00
150	40	R$ 300.000,00	R$ 60.000,00

Tabela IMOVEIS com colunas CODIGO, VALOR, RUA, CIDADE e ESTADO.

CODIGO	VALOR	RUA	CIDADE	ESTADO
10	R$ 500.000,00	Sol	Natal	RN
20	R$ 400.000,00	Amparo	Recife	PE
15	R$ 800.000,00	Costeira	Salvador	BA
30	R$ 600.000,00	Primavera	Curitiba	PR
25	R$ 900.000,00	Flores	Fortaleza	CE
40	R$ 300.000,00	Rosa	Goiania	GO

A partir dessas informações, julgue os itens de **93** a **96**.

93 A seguir, tem-se um comando SQL correto e a relação que deverá ser produzida caso esse comando seja aplicado, na situação apresentada.

```
select cidade as CAPITAL
from imoveiswhere
valor > 500000
order by cidade desc;
```

CAPITAL
Curitiba
Fortaleza
Salvador

94 A seguir, tem-se um comando SQL correto e a relação que deverá ser produzida caso esse comando seja aplicado, na situação apresentada.

```
select nome as CLIENTE, sum(saldo) as TOTAL
from clientes, emprestimos, clientes_emprestimos
where cpf = cliente and codigo = emprestimo
group by nome
having sum(saldo) > 50000
order by nome desc, sum(saldo) asc;
```

CLIENTE	TOTAL
Rosa	R$ 150.000,00
Jose	R$ 155.000,00

95 A seguir, tem-se um comando SQL correta e a relação que deverá ser produzida caso esse comando seja aplicado, na situação apresentada.

```
select nome as CLIENTE, saldo as DEBITO
from clientes, emprestimos, clientes_emprestimos
where cpf = cliente and emprestimo = codigo and imovel in (
select codigo
from imoveis
where valor > 500000)
order by nome, saldo;
```

CLIENTE	DEBITO
Jose	R$ 70.000,00
Jose	R$ 85.000,00
Paulo	R$ 10.000,00

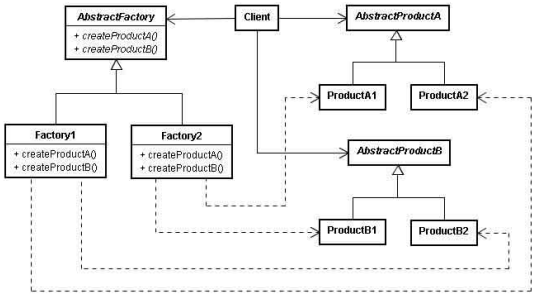
96 É correto afirmar que a seqüência de comandos SQL abaixo insere um registro em IMOVEIS, atualiza mais de um registro em EMPRESTIMOS e remove um registro de CLIENTES.

```
insert into imoveis (codigo, valor, rua, cidade, estado)
values (50, 200000, 'Almeida', 'Campinas', 'SP');
update emprestimos
where codigo > 200
set valor = 10000, saldo = 1000;
delete clientes
where cpf = 234;
```

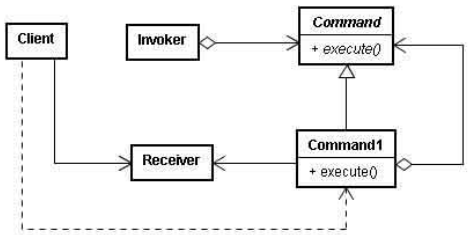
```
insert into imoveis (codigo, valor, rua, cidade, estado)
values (50, 200000, 'Almeida', 'Campinas', 'SP');
update emprestimos
set valor = 10000, saldo = 1000
where codigo > 200;
delete
from clientes
where cpf = 234;
```

Acerca de padrões de projeto (*design patterns*), julgue os itens de **97** a **100**.

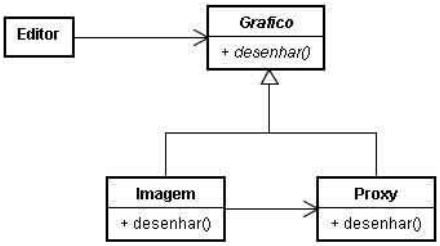
97 O seguinte diagrama UML documenta corretamente o padrão *abstract factory*.



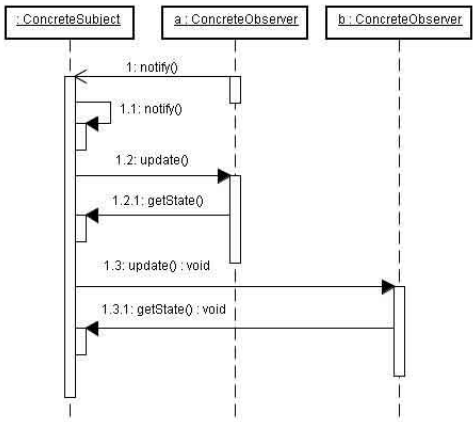
98 O padrão *command* está corretamente documentado no seguinte diagrama UML.



99 O padrão *proxy* está corretamente documentado no seguinte diagrama UML.



100 O seguinte diagrama UML documenta corretamente uma interação no padrão *observer*.



Acerca da *service-oriented architecture* (SOA), julgue os seguintes itens.

101 A orientação a serviços é uma estratégia de projeto com as seguintes características: a lógica de negócio se for automatizada, é particionada em serviços; os serviços apresentam forte acoplamento (*strong coupling*) e se comunicar pela troca de mensagens; para que serviços possam interagir, são disponibilizadas as suas descrições; promove reúso, pois serviços podem usar outros serviços; os serviços podem ser organizados em camadas com diferentes níveis de abstração.

102 Uma composição de serviços é um conjunto, em que cada serviço contribui na execução de uma tarefa. A montagem e a coordenação dos serviços pode ser uma atividade sob a responsabilidade de um controlador voltado para esse fim. Uma composição coordenada por um controlador pode, por sua vez, participar de outra composição maior.

103 Há um cabeçalho (*header*) em cada mensagem com o formato definido na especificação SOAP. Armazenados nesse cabeçalho, pode haver blocos de cabeçalho (*header blocks*) com informações relacionadas à entrega e ao processamento da mensagem. Por exemplo, blocos de cabeçalho podem conter informações de roteamento ou *workflow* associadas à mensagem.

104 No *framework* tecnológico chamado *web services*, as definições dos serviços podem ser feitas usando-se WSDL, as quais contribuem no sentido de possibilitar um forte acoplamento entre os serviços. Uma definição WSDL descreve a interface pública de um serviço, mas, alternativamente, é possível a descrição da interface de um serviço usando-se a linguagem UDDI.

Acerca de engenharia de *software*, julgue os itens a seguir.

105 A análise de requisitos localiza os requisitos funcionais e os não-funcionais. Nessa análise, são produzidos tipicamente artefatos que irão guiar o resto do processo de desenvolvimento. Na análise, os requisitos são definidos sem que o foco seja voltado para aspectos específicos da implementação a ser construída. Nos métodos de análise tipicamente recomendados, há informações de como organizar e representar as especificações dos requisitos.

106 A arquitetura de um sistema de *software* descreve os elementos que compõem o sistema e as interações entre eles. A arquitetura do *software* deve ser definida visando atender à especificação previamente estabelecida dos requisitos de *software*. Em um processo de desenvolvimento de *software*, a definição da arquitetura é tipicamente responsabilidade da disciplina de projeto.

107 Na disciplina de teste, o produto de *software* é executado para se verificar a presença de defeitos e aumentar a confiança na correção do produto, mas a execução de testes geralmente não é capaz de demonstrar que um *software* está correto. Atividades de teste, que podem ser realizadas durante o desenvolvimento, tipicamente são as seguintes: planejar atividades de teste, projetar testes, executar testes e avaliar resultados.

108 As técnicas de teste admitem a classificação como estrutural, funcional, com base em erros e com base em máquinas de estados finitos. Na técnica funcional, os requisitos de teste são estabelecidos com base em uma dada implementação. Na técnica estrutural, os requisitos são estabelecidos com base na especificação, sem necessidade de conhecimentos sobre uma dada implementação. Na técnica com base em erros, os requisitos são definidos explorando-se erros típicos durante um desenvolvimento. Na técnica com base em máquinas de estados finitos, para determinar requisitos de teste, são usadas máquinas de estado.

109 Ao longo do desenvolvimento, artefatos produzidos podem ser revisados, objetivando garantir que os mesmos apresentem, pelo menos, a qualidade mínima especificada. Não apenas o código, mas também outros artefatos podem ser revisados. Os defeitos encontrados pelas revisões referem-se à faltas (*fault*), enquanto os defeitos encontrados por testes são falhas do *software*, pois testes avaliam a qualidade comparando o comportamento esperado com o observado.

110 Um item de configuração de *software* é um item produzido no desenvolvimento para o qual é importante realizar o controle de alterações. Um conjunto de tais itens compõe uma configuração de *software*. A gerência de configuração envolve identificação e controle o qual abrange o controle das mudanças e das versões.

111 Em um processo de manutenção, há atividades que provêem serviços de manutenção ao *software*. Em um processo, a manutenção, ativada quando o *software* é modificado, pode ser adaptativa, corretiva, perfectiva ou preventiva. É corretiva se, por exemplo, objetiva corrigir desvios em relação ao padrão de codificação; é perfectiva se, por exemplo, visa incluir novas funcionalidades.

Acerca de tecnologias usadas no desenvolvimento de aplicações, julgue os itens subseqüentes.

112 Acerca do Tomcat é correto afirmar que: o diretório raiz da instalação de um servidor é, tipicamente, identificado utilizando-se a variável `$ CATALINA_TOMCAT`; server.xml é um arquivo de configuração; para que uma aplicação *web* possa ser instalada, tem que estar descompactada, não podendo estar em um arquivo WAR; um descritor de contexto é uma aplicação *web*.

113 Acerca do JBoss é correto afirmar: MBeans são componentes na arquitetura do servidor; *minimal*, *default*, *production* e *all* são possíveis configurações do servidor; informações de configuração estão no arquivo jboss-service.xml que deve ser armazenado no diretório data; o início da execução do servidor faz-se via comando `runjboss.bat` ou `runjboss.sh`.

114 Acerca do servidor Apache HTTP, é correto afirmar que: em máquinas com Windows, o servidor pode ser executado como um serviço; em máquinas com Unix, o httpd pode ser executado como um *daemon* em *background*; pode-se configurar o servidor via diretivas em httpd.conf.

115 Acerca do *framework* Hibernate, é correto afirmar que ele: possibilita o mapeamento entre o modelo relacional e o orientado a objetos e pode ser definido usando-se arquivos XML; ainda que, em um arquivo de configuração, pode-se declarar uma classe persistente, usando o elemento “class”; além disso, os relacionamentos entre as classes persistentes têm que ser unidirecionais e 1:1.

116 Quanto ao *framework* Hibernate é correto afirmar que: nas aplicações que o usam, as classes persistentes devem seguir o modelo JavaBeans, pois é impossível persistir as classes que seguem o modelo POJO (*plain old java object*); o *framework* só deve ser usado naquelas aplicações que usam intensamente procedimentos armazenados (*stored procedures*).

117 *ActionServlet* é a classe do controlador responsável por processar solicitações feitas pelos usuários e por mapear as requisições em ações, porque o *framework* usa o arquivo config-struts.xml; aplicações implementadas com o *framewok* podem conter *servlets*, mas não páginas JSP.

118 O Struts emprega o estilo de arquitetura MVC. Nele, os objetos da classe `Action` delegam requisições para objetos da classe `ActionServlet`; arquivos web.xml são usados para configurar aplicações; o método `autenticate()` em `ActionForm` é usado para validar dados de entrada preenchidos nos formulários.

119 No código em *cascading style sheets* (CSS) abaixo, vermelho é tanto a cor de fundo de textos destacados por <H4>, quanto a de textos em parágrafos da classe local; verde é a cor de fundo de textos em elementos da classe diretores; azul é a cor de elementos com identificador rua. Vermelho é também a cor dos elementos que são destacados por <H3> e têm o identificador bairro.

```
P, H4 {color: black; background-color:
red}
H1 {color: yellow}
P.local  {font-size: 18pt; color: red;
background-color: green}
.funcionarios  {color: blue; background-
color: red}
.diretores {color: yellow; background-
color: green}
#rua {color: blue}
H3#bairro {color: red}
```

120 A respeito de Ajax é correto afirmar: instâncias de `XMLHttpRequest` são usadas código em JavaScript, para se comunicar com servidores; a comunicação é possível apenas via método `GET`; a propriedade `onreadyreceive` identifica a função que processar respostas; `open` e `send` são usados para solicitar serviços; a propriedade `readyState` informa estado da solicitação no processamento da resposta; os dados podem ser enviados e recebidos só no formato XML.