# Questões das vídeo-aulas

## Enterprise JavaBeans

### [01] AOCP - 2012 - BRDE

Sobre definições e características de Enterprise JavaBeans, analise as assertivas e assinale a alternativa que aponta as corretas.

I. A arquitetura Enterprise JavaBeans é uma arquitetura de componentes para o desenvolvimento e a implantação de aplicativos de negócios distribuídos baseados em componentes.

II. Aplicativos escritos utilizando a arquitetura Enterprise JavaBeans são escalonáveis, transacionais e seguros com multiusuários.

III. Aplicativos escritos utilizando a arquitetura Enterprise JavaBeans uma vez escritos e então implantados em qualquer plataforma de servidor, que suporta a especificação Enterpise JavaBeans.

IV. A arquitetura JavaBeans encontra-se presentes em outras linguagens de programação além da linguagem de programação java, esta arquitetura encontra- se em Object Pascal, Objective-C, Python, SuperCollider, Ruby, Smalltalk, entre outras.

a) Apenas I e II.

b) Apenas I e III.

c) Apenas I, II e III.

d) Apenas II, III e IV.

e) I, II, III e IV.

### [02] FCC - 2013 - TRT-15

O EJB é um modelo de componentes, especificado pela plataforma Java EE, elaborado para resolver problemas e desafios complexos de softwares corporativos. São componentes que atuam na camada servidor, classificados como componentes de negócio. Podem ser utilizados em diferentes situações como desenvolvimento distribuído, integração/conectividade com legado, processamento assíncrono baseado Fila / Mensagens, controle transacional e outros. Este componente é responsável pelas regras de negócio, ou seja, a persistência e o controle transacional.

### [03] NUCEPE - 2015 - SEFAZ-PI

O Enterprise JavaBeans (EJB) é um componente da plataforma Java EE que executa em um container de um servidor de aplicação. Quais os seus 3 (três) tipos fundamentais de beans?

a) Entity Beans, Session Beans e Message Driven Beans.

b) Entity Beans, Session Beans e Work Beans;

c) Session Beans, Message Driven Beans e Work Beans.

d) Session Beans, Progress Beans e Work Beans.

e) Entity Beans, Progress Beans e Work Beans.

## [04]

**CESPE - 2011 - TRE-ES**

Nos *beans* de entidade cuja persistência é gerenciada por contêiner, o desenvolvedor tem a responsabilidade de escrever todo o código JDBC para a interação com o banco de dados.

**ESAF - 2008 - CGU [adaptada]**

a) os *Entity Beans* foram substituídos pela "*Java Persistence API*" na versão EJB 3.0, porém, os *Entity Beans* de versões 2.x podem continuar utilizando o *"Container-Managed Persistence"* (CMP) por questões de compatibilidade.

## [05] FCC - 2015 - TRT-15

Em uma aplicação web que utiliza Enterprise JavaBeans (EJB) para implementar um carrinho de compras, utilizou-se um tipo de bean que mantém o estado durante uma sessão com o cliente. Nesta aplicação, para indicar ao servidor que a classe é um bean com estado de sessão deve-se utilizar, antes da declaração da classe, a anotação

a) @Session state="true"

b) @Stateful

c) @SessionState= "true"

d) @Stateless

e) @SessionRemote

## [06] FCC - 2014 - TCE-RS

Em uma aplicação Java EE há:

- uma classe EJB chamada ExemploSessionBean.java que não permite manter um estado de conversação com o cliente,
- uma interface chamada ExemploSessionBeanRemote.java e
- uma classe cliente desktop chamada Main.java,

todas criadas em seus respectivos projetos e em condições de execução ideais. Os fragmentos de código-fonte destas classes são apresentados a seguir, omitindo-se as declarações de pacotes e importação de classes:

```
ExemploSessionBean.java

        I
public class ExemploSessionBean implements ExemploSessionBeanRemote{
        II
    public String getMessageRemote() {
        return "Remote EJB";
    }
}

ExemploSessionBeanRemote.java

        III
public interface ExemploSessionBeanRemote {
    String getMessageRemote();
}

Main.java

public class Main {
        IV
    private static ExemploSessionBeanRemote ex;
    public static void main(String[] args) {
        System.err.print(ex.getMessageRemote());
    }
}
```

As lacunas I, II, III e IV são preenchidas correta e, respectivamente, pelas anotações

a) @Statefull, @Override, @Remote e @EJB.

b) @Stateless, @Override, @Remote e @EJB.

c) @EJB, @Stateless, @Remote e @EJBInjection.

d) @Stateless, @EntityManager, @RemoteInterface e @EJBInjection.

e) @Statefull, @EJBMethod, @RemoteInterface e @EJBInjection.

## [07] CESGRANRIO - 2013 - BNDES

Cada tipo de enterprise bean passa por diferentes fases durante seu ciclo de vida. Um desses tipos possui um estado denominado Passivo. Quando um bean entra nesse estado, o container EJB o desloca da memória principal para a memória secundária.

Qual tipo de bean se comporta dessa maneira?

a) Stateless Session Bean

b) Stateful Session Bean

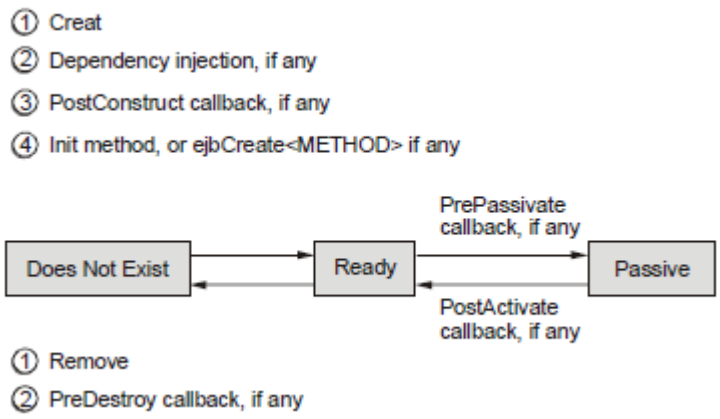
c) Web Service Bean

d) Singleton Session Bean

e) Message-Driven Bean

## [08] FCC - 2012 - TRT-AM

Analise a figura:




Foi representado o ciclo de vida de um

a) Stateful Session Bean.

b) Stateless Session Bean.

c) Singleton Session Bean.

d) Message-Driven Bean.

e) Embedded Session Bean.

## [09] FCC - 2011 - TRT-19

Tipo de session bean EJB 3.1 cujas instâncias não têm estado conversacional, isto é, todas as instâncias são equivalentes quando não estão envolvidas em atender um método invocado pelo cliente. Trata-se de

a) Stateful.

b) Stateless.

c) Singleton.

d) Message driven.

e) Entity.

## [10]

**ESAF - 2008 - CGU [adptada]**

e) os *Stateless Session Beans* são objetos distribuídos que não possuem estado, permitindo acesso concorrente aos mesmos. Assim, o conteúdo das variáveis de instância é preservado entre as chamadas de métodos.

**FCC - 2011 - TRT-24 [adptada]**

a) No Session Bean, toda vez que um método é invocado, o estado de suas variáveis se mantém apenas durante a invocação desse método.

**NUCEPE - 2015 - SEFAZ-PI [adptada]**
e) o Stateful Session Bean não mantém estado, permitindo que diversas chamadas a métodos sejam feitas de forma que uma chamada dependa da outra.

## [11] NC-UFPR - 2015 - COPEL

Sobre Session Beans, conforme a especificação EJB (Enterprise JavaBeans) 3.1, identifique as afirmativas a seguir como verdadeiras (V) ou falsas (F):

( ) Não é possível utilizar Multithreading em EJBs do tipo Singleton.

( ) Stateless Session Beans não armazena nenhuma informação sobre o estado transacional (conversacional), ou seja, nenhuma informação é automaticamente mantida entre as diferentes requisições.

( ) Recomenda-se utilizar um Stateful Session Bean ao construir um carrinho de compras de um e-commerce, embora seja possível usar um Stateless Session Bean, tendo um pouco mais de trabalho.

( ) Existem apenas três tipos de Session Beans: Stateful, Stateless e Singleton.

Assinale a alternativa que apresenta a sequência correta, de cima para baixo.

a) F – V – F – F.
b) F – F – V – V.
c) V – V – V – F.
d) F – V – V – V.
e) V – F – F – V.

## [12] CESPE - 2010 - TCU

Uma equipe de desenvolvimento de *software* recebeu a incumbência de desenvolver um sistema com as características apresentadas a seguir.

O sistema deverá ser integrado, interoperável, portável e seguro.

O sistema deverá apoiar tanto o processamento *online*, quanto o suporte a decisão e gestão de conteúdos.

O sistema deverá ser embasado na plataforma JEE (Java *enterprise edition*) v.6, envolvendo *servlets*, JSP (Java *server pages*), Ajax, JSF (Java *server faces*) 2.0, Hibernate 3.5, SOA e *web services*.

O líder da equipe iniciou, então, um extenso processo de coleta de dados com o objetivo de identificar as condições limitantes da solução a ser desenvolvida e tomar decisões arquiteturais e tecnológicas que impactarão várias características funcionais e não funcionais do sistema, ao longo de seu ciclo de vida. A partir dessa coleta, o líder deverá apresentar à equipe um conjunto de informações e de decisões.

Com relação às diferentes arquiteturas e tecnologias que, se escolhidas, impactarão as características do sistema descrito no texto, julgue o item.

A tecnologia EJB (*enterprise* Java *beans*) apresenta, na sua versão 3.1, melhorias que propiciam facilidades para o uso de *beans singleton* e que permitem o uso de *beans* de uma classe, sem necessidade de desenvolvimento de sua interface correspondente, e a invocação assíncrona de *beans* de sessão.

### [13]

**NUCEPE - 2015 - SEFAZ-PI**

O Message Driven Bean é usado para processar mensagens síncronas. Sempre há acoplamento entre a aplicação cliente e o Message Driven Bean.

**FCC - 2011 - TRT-24 [adaptada]**

b) Um Message-Driven Bean é um EJB que possui as interfaces home e remote e apenas um método que recebe qualquer tipo de mensagem.

**FCC - 2012 - TRE-CE [adaptada]**

O Message-Driven Bean utiliza a API JMS, facilita a quebra de acoplamento entre o cliente e o destino (Point-to-point ou Publish- subscriber), e é acionado de forma assíncrona.

### [14] VUNESP - 2009 - CETESB

Em um servidor de aplicações, o tipo de enterprise bean que é definido, sem nenhuma interface com o cliente, é

a) BPM Entity.

b) CMP Entity.

c) Stateful Session.

d) Stateless Session.

e) Message Driven.

## Java Persistence API

### [15] FCC - TCE-PR 2011

A JPA

a) pode ser usada fora de componentes EJB e fora da plataforma Java EE, em aplicações Java SE.

b) utiliza persistência gerenciada por contêiner (CMP), ou seja, as classes de entidade e persistência necessitam de um contêiner presente em um servidor de aplicações para serem executadas.

c) utiliza descritores XML para especificar informações do mapeamento relacional de objeto, mas não oferece suporte a anotações.

d) suporta consultas dinâmicas nomeadas nas classes de entidade que são acessadas apenas por instruções SQL nativas.

e) possui uma interface *EntityBeans* que padroniza operações *Create Read Update Delete* (CRUD) que envolvem tabelas.

## [16]

**CESPE - MPU - 2010**

A versão 3.0 da API de Persistência Java utiliza descritores de implantação, não permitindo uso de anotações.

**CESPE - MPU - 2010**

A API de Persistência Java é embasada em ideias contidas em *frameworks* líderes de mercado, como Hibernate, Oracle TopLink e Objetos de Dados Java.

**CESPE - CNJ 2013**

Os objetos mapeados na linguagem Java que devem ser persistidos como objetos precisam utilizar JPA (Java *persistence* API), pois o JPA permite realizar o mapeamento objeto/relacional automatizado e transparente e sua persistência em um banco de dados relacional.

## [17] FCC - 2015 - TRT - 9ª REGIÃO (PR)

a) são definidas as configurações do servidor de aplicação, como número de conexões simultâneas permitidas e dados de log.

b) é definido o mapeamento objeto-relacional entre as tabelas do banco de dados e as classes de entidade da aplicação.

c) são definidas as queries nomeadas, que são queries pré-definidas que podem ser chamadas pelo nome a partir de classes de acesso a dados da aplicação.

d) há o mapeamento de componentes da camada de apresentação para os respectivos componentes da camada de acesso a dados da aplicação.

e) são definidas as propriedades de conexão com o banco de dados, como o driver JDBC, a URL de conexão, o nome do usuário e a senha.

## [18] FCC - 2012 - TJ-PE

Sobre JEE e tecnologias relacionadas é correto afirmar que

a) O EntityManager é uma classe identificada com a anotação @Entity que representa o modelo das tabelas do banco de dados.

b) O EntityManager é o serviço central do JPA para todas as ações de persistência e oferece todas as funcionalidades de um DAO genérico.

c) Um servidor de aplicações Java EE possui um único contêiner conhecido como contêiner EJB.

d) Serlvets e JSP rodam no contêiner EJB do servidor de aplicação JEE.

e) Em aplicações que utilizam EJB com JPA, um arquivo persistence.xml pode definir uma única unidade de persistência.

## [19] FCC - 2012 - TJ-PE

Quando se utiliza JPA, um EntityManager mapeia um conjunto de classes a um banco de dados particular. Este conjunto de classes, definido em um arquivo chamado persistence.xml, é denominado

a) persistence context.

b) persistence unit.

c) entity manager factory.

d) entity transaction.

e) persistence provider.

## [20] FCC - 2014 - TRF 3

Considere a seguinte classe desenvolvida em uma aplicação Java que utiliza *JPA/Hibernate*:

```
import javax.persistence.*;
public class NewClassDao {
    public void conectar() {
        EntityManagerFactory a = Persistence.createEntityManagerFactory("conectar");
        EntityManager b = a.createEntityManager();
        EntityTransaction c = b.getTransaction();
        c.begin();
    }
}
```

É correto afirmar que os diversos métodos para executar operações de inserção, consulta, alteração e exclusão de registros no Banco de Dados (*persist, find, merge, remove, createQuery* etc) podem ser acessados por meio do objeto ...ˡ... . A *String* “conectar” refere-se ao nome da ..ˡ..ˡ... .

As lacunas I e II são preenchidas correta e respectivamente com

a) b − interface local

b) c − unidade de persistência

c) c − interface remota

d) a − interface local

e) b − unidade de persistência

## [21] CESGRANRIO - 2012 - Petrobras

Em aplicações Java Enterprise Edition 6, é comum o uso da API JPA. Nessa API, há o conceito de classe de entidade (entity class). Por definição, uma classe de entidade deve, obrigatoriamente, cumprir os seguintes requisitos, EXCETO

a) estar anotada com a anotação Entity ou representada em um descritor XML.

b) não ser qualificada com final.

c) ter as variáveis de instância persistentes qualificadas com private, protected, ou package-private.

d) ter ao menos um construtor, este sem argumentos (no-arg constructor).

e) ter o mesmo nome da tabela correspondente do banco de dados.

## [22] FCC - 2010 - TRT 8

Os três estados de objeto definidos pelo framework Hibernate são:

a) Temporário (Temporary), Permanente (Permanent) e Resiliente (Resilient).

b) Transiente (Transient), Persistente (Persistent) e Resiliente (Resilient).

c) Temporário (Temporary), Persistente (Persistent) e Destacado (Detached).

d) Transiente (Transient), Persistente (Persistent) e Destacado (Detached).

e) Transiente *(Transient)*, Permanente *(Permanent)* e Resiliente *(Resilient).*

## [23] FCC - 2011 - TRT - 19ª Região (AL)

Os estados do ciclo de vida de uma instância de uma entidade, definidos na JPA 2.0, são .

a) novo (new), gerenciado (managed), destacado (detached) e removido (removed).

b) ativo (active), inativo (inactive) e removido (removed).

c) novo (new), temporário (temporary), permanente (permanent) e destacado (detached).

d) novo (new), temporário (temporary) e destacado (detached)

e) gerenciado (managed), temporário (temporary), permanente (permanent) e destacado (detached).

## [24] FCC - 2007 - MPU

Objetos que têm uma representação no banco de dados, mas não fazem mais parte de uma sessão do *Hibernate*, o que significa que o seu estado pode não estar mais sincronizado com o banco de dados, são do tipo

a) transient.

b) detached.

c) attached.

d) persistent.

e) *consistent.*

## [25]

**CESPE - TRE GO 2015**

A anotação @Entity significa que determinada classe Java é uma entidade do banco de dados. Caso essa entidade tenha nome que não seja o da tabela, será necessário utilizar a anotação @Table.

**CESPE - TRE GO 2015**

Ao se declarar uma coluna que seja a chave primária de uma tabela, é necessário utilizar a anotação @Id.

## [26] FCC - 2015 - TRT-4 (RS)

Uma aplicação que trabalha com Hibernate e EJB possui uma classe POJO − Plain Old Java Object utilizada no mapeamento objeto-relacional com uma tabela do banco de dados. Nessa classe, há um atributo calculado chamado valorTotalPedido que, para ser utilizado apenas em tempo de execução e descartado após finalizar o seu serviço temporário, deverá ser anotado com

a) @Embedded

b) @TemporaryAttribute

c) @GeneratedValue

d) @Transient

e) @Basic

## [27] FGV - 2014 - PROCEMPA

Considere a seguinte classe com anotações JPA:

```
@Entity
@Table(name = "funcionario")
public class Funcionario implements Serializable {
    private static final long serialVersionUID = 2L;
    @Id
    @Column(name = "id", nullable = false)
    private Integer id;
    @Column(name = "nome")
    private String primaryKey;
    @ManyToOne
    private Funcionario chefe;
    // Restante da classe...
}
```

Sobre essa classe anotada, analise as afirmativas a seguir.

I. A anotação @Table é dispensável, neste caso.

II. A chave primária da tabela associada à classe Funcionario é nome.

III. A anotação @ManyToOne introduz, neste exemplo, um autorrelacionamento.

Após o exame das afirmativas, verifica-se que
a) somente I e II são verdadeiras.

b) somente I e III são verdadeiras.

c) somente II e III são verdadeiras.

d) somente II é verdadeira.

e) somente I é verdadeira.

## [28] FCC - 2011 - TRT - 23ª REGIÃO (MT)

Considere:

```java
@Entity
@Table(name = "domic")
@NamedQueries({
     @NamedQuery(name="Domic.findById", query="SELECT r FROM Domic r WHERE r.id=:id"),
     @NamedQuery(name="Domic.findByNome", query="SELECT r FROM Domic r WHERE r.nome=:nome")
})
public class Domic implements Serializable {

   private static final long serialVersionUID = 1L;
   @Id
   @Column(name = "id", nullable = false)
   private Integer id;
   @Column(name = "nome")
   private String nome;
   @OneToMany(cascade = CascadeType.ALL, mappedBy = "domicId")
   private Collection<Predio> predioCollection;
```

Em relação à JPA (*Java Persistence API*) é INCORRETO afirmar que

a) @NamedQuery é aplicada para definir várias consultas.

b) @Entity define que haverá correspondência da classe com uma tabela do banco de dados.

c) @Id define que o atributo que está mapeado com tal anotação corresponderá à chave primária da tabela.

d) @Column(name = "id", nullable = false) define que o atributo da classe mapeado com tal anotação deve estar associado à coluna cujo nome é "id", além de definir que tal campo não pode ser nulo.

e) @OneToMany indica que o atributo contém um conjunto de entidades que a referenciam.

## [29] FCC - 2014 - TRT - 13ª REGIÃO (PB)

Java Persistence API (JPA) é uma API padrão da linguagem Java para persistência de dados em bancos de dados relacionais.

Em uma aplicação que utiliza JPA

a) pode ser utilizada, como provedor de persistência, as bibliotecas EclipseLink, Hibernate, OracleTopLink, JBossSeam e JDBCProvider.

b) as classes de entidade do banco de dados permitem o mapeamento entre objetos da classe e tabelas do banco de dados, utilizando anotações como @Table, @Entity, @PrimaryKey, @Column, @Constraint, @Foreignkey e @EJB.

c) todas as operações realizadas nas tabelas do banco de dados, como inserção de dados, alteração, consultas e exclusão, são realizadas sem o uso de instruções SQL, ou seja, o desenvolvedor não precisa conhecer SQL para programar.

d) as configurações de acesso a banco de dados normalmente ficam no arquivo persistence.xml, ligado à aplicação, de forma que se for alterado o servidor de banco de dados não seja necessário alterar o código-fonte Java da aplicação.

e) as relações existentes entre as tabelas do banco de dados não são refletidas nas classes de entidade criadas na aplicação, o que torna a execução mais rápida. O mapeamento de relações é feito em tempo de execução pelas bibliotecas do provedor de persistência.

## [30] NC-UFPR - 2015 - COPEL

Em relação ao mapeamento objeto-relacional usando JPA (Java Persistence API) 2.0, assinale a alternativa correta.

a) Quando se usa a anotação @OneToOne(mappedBy="pai"), entende-se que a chave estrangeira desse mapeamento aponta para uma tabela chamada pai.

b) A anotação @Temporal indica que esse atributo é temporário, ou seja, é um sinônimo da anotação @Transient.

c) Quando usamos a anotação @GeneratedValue(strategy=GenerationType.SEQUENCE), devemos informar o valor da propriedade generator, que pode apontar para um @TableGenerator ou @SequenceGenerator.

d) A anotação @Version está Deprecated e portanto não deve ser utilizada, já que entra em conflito com a JTA (Java Transaction API)

e) A anotação @ManyToMany indica que o relacionamento é bidirecional, e mesmo que seja informado em apenas uma das classes, será possível realizar a navegação (e obter suas respectivas coleções) em ambos os lados.

## [31] UERJ - 2015 - UERJ

Java Persistence API (JPA) é uma especificação para frameworks de mapeamento objeto-relacional. Nesse contexto, considere que em um programa em Java existam duas classes, denominadas Pessoa e Projeto. Considere ainda que a classe Pessoa contém a declaração a seguir.

```
@ManyToMany
@JoinTable(name = "PESSOA_PROJETO",
     joinColumns = { @JoinColumn(name = "C1", referencedColumnName = "C2") },
     inverseJoinColumns = { @JoinColumn(name = "C4", referencedColumnName = "C3") })
private List<Projeto> projetos;
```

Dentre as opções abaixo, aquela que lista corretamente as colunas de chaves estrangeiras que devem ser definidas na tabela PESSOA_PROJETO é:

a) C1 e C2

b) C1 e C4

c) C2 e C3

d) C3 e C4

## [32] NC-UFPR - 2015 - COPEL

Quanto a JPA (Java Persistence API) 2.0 e seus modos de carregamento (FetchType) Lazy e Eager, identifique as afirmativas a seguir como verdadeiras (V) ou falsas (F):

( ) Eager é o comportamento padrão para relacionamentos muitos-para-muitos.

( ) É preciso cuidar do cascateamento ao usar Eager Load, pois muitos objetos podem ser carregados desnecessariamente.

( ) Lazy apresenta maior consumo de processamento e rede durante a inicialização da aplicação quando comparado comEager.

( ) Fazer cache de objetos instanciados via Lazy Load é geralmente desaconselhável, devido ao alto consumo de processamento.

( ) Essas formas de carregamento tornaram-se Deprecated na JPA 2.0.
Assinale a alternativa que apresenta a sequência correta, de cima para baixo.

a) V – F – V – F – V.

b) F – V – F – V – V.

c) V – F – V – V – F.

d) F – F – V – V – F.

e) F – V – F – F – F.

## [33] CESPE - 2015 - FUB

Ao se usar JPA, a forma de armazenamento dos dados das instâncias de uma classe e suas subclasses varia de acordo com a estratégia de herança adotada. Essa variação pode incluir todos os dados armazenados em uma tabela, bem como dados armazenados em tabelas distintas que usam uma coluna de referência.

## [34]

**CESPE - 2015 - TRE GO**

Para que uma classe Java efetue consultas em determinada entidade do banco de dados, é necessário elaborar o SQL e, depois, convertê-lo para JPQL (Java p*ersistence query language*).

**CESPE - 2010 - MPU**

A versão 3.0 da API de Persistência Java provê uma linguagem de consulta de persistência Java que é uma forma melhorada da linguagem de consulta do EJB.

**CESPE - 2015 - STJ**

JPQL (Java Persistence Query Language) é uma linguagem de manipulação de dados adotada para criar, alterar estrutura de tabelas e gatilhos utilizados na especificação JPA (Java Persistence API).

## [35] FCC - 2015 - TRE-AP

Em uma classe de entidade do banco de dados presente em uma aplicação que utiliza JPA existem as seguintes instruções:

```
@NamedQuery(name="Cliente.listarTodos",query="select c from Cliente c")
@Entity
public class Cliente {
// atributos e métodos
}
```

Considere que os atributos e métodos da classe Cliente estão implementados e mapeados adequadamente para a tabela Cliente do banco de dados.

Em uma classe de acesso a dados da mesma aplicação, que possui um objeto em válido do tipo EntityManager, para executar a *query* da classe de entidade Cliente e obter os dados retornados em uma lista, utiliza-se:
a) Query query = em.createNamedQuery("Cliente.listarTodos",Cliente.class);
List <Clientes> clientes = query.getResultList();

b)ResultSet query = em.createQuery("Cliente.listarTodos",Cliente.class);
ArrayList <Clientes> clientes = query.getResultList();

c)Query query = em.createNativeQuery("Cliente.listarTodos",Cliente.class);
List <Clientes> clientes = query.getList();

d)Query query = em.createQuery("Cliente.listarTodos",Cliente.class);
List <Clientes> clientes = query.getResultList();

e)List query = em.createNamedQuery("Cliente.listarTodos",Cliente.class);
ArrayList clientes = query.getResultSet();

## [36] UERJ - 2015 - UERJ

Java Persistence Query Language (JPQL) é uma linguagem de consulta que faz parte da especificação JPA. Considere uma aplicação em Java que usa JPA, na qual está definida uma classe de entidade denominada br.app.acme.Cliente.

Além disso, essa aplicação contém o trecho de código abaixo, que cria um objeto do tipo javax.persistence.Query, cuja referência é qry.

```
javax.persistence.Query qry = entityManager.createQuery(
        "SELECT OBJECT (c) FROM br.app.acme.Cliente c " +
        "WHERE c.uf = :uf");
qry.setParameter("uf", "Rio de Janeiro");
```

A expressão adequada para execução da consulta em JPQL representada pela referência qry é:

a) java.util.Collection clientes = qry.getResultList()

b) java.util.Collection clientes = qry.executeQuery()

c) br.app.acme.Cliente[] clientes = qry.getResultList()

d) java.util.Collection clientes = qry.getSingleResult()

### [37] FCC - 2015 - TRE-AP

Considere o fragmento de código a seguir, presente em uma classe ideal de acesso a dados de uma aplicação que utiliza JPA.

```
String jpql = "select e from Empregado e where e.cargo = :c";
Query q = entityManager.createQuery(jpql, Empregado.class);
...I...
List <Empregados> empregados = q.getResultList ();
```

Para completar corretamente o fragmento de código de forma que a consulta retorne os empregados cujo cargo seja Gerente, a lacuna Ideve ser preenchida por
a) q.setString("c", "Gerente");

b) q.setParameter("cargo", "Gerente");

c) q.setString(c, "Gerente");

d) q.setAttribute(c, "Gerente");

e) q.setParameter("c", "Gerente");

## Java DataBase Connectivity

### [38]

**CESPE - 2012 - TJ-RO**
JDBC, uma biblioteca vinculada a API da arquitetura JEE, define como um cliente pode acessar bancos de dados OO exclusivamente.

**CESPE - 2014 - TJ-SE**

JDBC faz conexão persistente entre as instâncias beans e as chamadas aos bancos de dados conectados, sendo, portanto, incompatível com sessões do tipo bean stateful.

**CAIP-IMES - 2014 - Prefeitura SP [Adaptada]**

O JDBC é um conjunto de interfaces e classes que tem como objetivo padronizar o modo com que um aplicativo qualquer se conecte com banco de dados. Possui independência da plataforma do Sistema Operacional e também visa a obter independência de banco de dados.

## [39] NUCEPE - 2015 - SEFAZ - PI

No Java, a classe DriverManager fornece os serviços básicos para gerenciamento de drivers JDBC. Quais três argumentos normalmente são passados como parâmetros em seu método *getConnection*?

a) String url, String user e String password.

b) String url, String port e String database.

c) String url, String user e String database.

d) String user, String port e String database.

e) String user, String password e String database.

## [40] CESPE - 2009 - IBAMA

```
1   import java.sql.*;
2   public class JoltReport {
3      public static void main(String args[]) {
4         String URL = "jdbc:odbc:CafeJolt";
5         String username = "";
6         String password = "";
7         try {
8            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
9         } catch (Exception e) {
10           System.out.println("“Error");
11           return;
12        }
13        //continua ....
```

As linhas de 7 a 12 permitem carregar o driver que informa às classes JDBC como se comunicar com a fonte de dados. No código, é usada a classe JdbcOdbcDriver e a linha 10 informa a ocorrência de erro no carregamento do driver JDBC/ODBC.

## [41] CESPE - 2009 - CEHAP-PB

```
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
```

Considerando a utilização da linha de código acima no estabelecimento de uma conexão com JDBC, assinale a opção correta.

a) EmbeddedDriver é o principal tipo de driver de conexão JDBC e ODBC em java.

b) Na linha de código, a chamada para Class.forName automaticamente cria uma instância de um driver e o registra com o DriverManager.

c) O trecho de código dado faz os dois passos necessários para a conexão a uma base com JDBC; o EmbeddedDriver faz todo o restante do trabalho de conexão.

d) Class.forName faz parte de outra classe denominada DriverProperty.JDBC.Main, que é utilizada com a tecnologia JDBC.

## [42] FCC - 2013 - TRT - 15ª Região

O método a seguir foi extraído de uma classe Java que permite o acesso a um banco de dados relacional.

```
public int conectar(String a, String b, String c, String d) {
   try {
      Class.forName(a);
      x = DriverManager.getConnection(b, c, d);
      y = x.createStatement();
      return 1;
   } catch (ClassNotFoundException ex) {
      return 2;
   } catch (SQLException ex1) {
      return 3;
   }
}
```

Sobre este método é correto afirmar que :

a) o parâmetro d refere-se ao endereço do banco de dados.

b) x é um objeto da interface SQLConnection.

c) y é um objeto da interface PreparedStatement.

d) o parâmetro b refere-se ao nome do usuário do banco de dados.

e) o parâmetro a refere-se ao driver JDBC.

## [43] FCC - 2012 - MPE-AP

Analise as linhas a seguir presentes em um programa Java que não apresenta erros.

```
a = DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=E:\\bd.mdb", "", "");
b = a.createStatement( );
c = b.executeQuery("select * from cliente where id = "+ valor +"");
```

Considere que os objetos a, b e c são de interfaces contidas no pacote java.sql. Pode-se concluir que esses objetos são, respectivamente, das interfaces

a) Connection, SessionStatement e Result.

b) DriverManager, PreparedStatement e RecordSet.

c) ConnectionStatement, PreparedStatement e RecordSet.

d) Connection, Statement e ResultSet.

e) DaoConnection, Statement e ResultSet.

## [44] FCC - 2013 - TRT - 12ª Região (SC)

Considere, abaixo, os métodos encontrados em classes de aplicações Java que acessam banco de dados.

**Método 1**

```
public int inserir(int varId, String VarNome, double varRenda) {
   int retorno;
   try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection(
           "jdbc:mysql://localhost:3306/bd007", "root", "1234");
      PreparedStatement st = conn
           .prepareStatement("insert into cliente (id,nome,renda) values (?,?,?");
      st.setInt(1, varId);
      st.setString(2, VarNome);
      st.setDouble(3, varRenda);
      retorno = st.executeUpdate();
   } catch (ClassNotFoundException ex) {
      retorno = 2;
   } catch (SQLException ex1) {
      retorno = 3;
   }
   return retorno;
}
```

**Método 2**

```
public int inserir(int varId, String VarNome, double varRenda) {
   int retorno;
   try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection(
           "jdbc:mysql://localhost:3306/bd007", "root", "1234");
      Statement st = conn.createStatement();
      retorno = st.executeUpdate("insert into cliente values (" + varId
           + "," + VarNome + "," + varRenda + ")");
   } catch (ClassNotFoundException ex) {
      retorno = 2;
   } catch (SQLException ex1) {
      retorno = 3;
   }
   return retorno;
}
```

Nas classes, nas quais estes métodos se encontram, foram importados todos os recursos necessários para a execução. O banco de dados, a tabela e o driver JDBC existem e funcionam corretamente.

É correto afirmar que

a) o Método 1 está incorreto, pois o método executeUpdate da interface PreparedStatement precisa receber como parâmetro a instrução SQL insert a ser executada.

b) o Método 2 está incorreto, pois o método executeUpdate da interface Statement não pode receber parâmetros. A instrução insert passada como parâmetro nesse método deveria ser passada como parâmetro para o método createStatement da interface Connection.

c) ambos os métodos estão corretos e executam a mesma operação, apresentando os mesmos resultados.

d) ambos os métodos estão incorretos, pois o método presente tanto na interface Statement como na interface PreparedStatement para incluir dados na tabela do banco de dados é o método executeInsert e não executeUpdate .

e) o Método 1 está incorreto, pois a instrução insert passada como parâmetro para o método PreparedStatement da interface Connection está incompleta. No lugar dos pontos de interrogação devem ser colocados os valores que devem ser incluídos nos campos id , nome e renda da tabela.

# Gabarito

| | |
|---|---|
| [01] | C |
| [02] | C |
| [03] | A |
| [04] | E-C |
| [05] | B |
| [06] | B |
| [07] | B |
| [08] | A |
| [09] | B |
| [10] | E-E-E |
| [11] | D |

| | |
|---|---|
| [12] | C |
| [13] | E-E-C |
| [14] | E |
| [15] | A |
| [16] | E-C-C |
| [17] | E |
| [18] | B |
| [19] | B |
| [20] | E |
| [21] | E |
| [22] | D |

| | |
|---|---|
| [23] | A |
| [24] | B |
| [25] | C-C |
| [26] | D |
| [27] | B |
| [28] | A |
| [29] | D |
| [30] | C |
| [31] | B |
| [32] | E |
| [33] | C |

| | |
|---|---|
| [34] | E-C-E |
| [35] | A |
| [36] | A |
| [37] | E |
| [38] | E-E-C |
| [39] | A |
| [40] | C |
| [41] | B |
| [42] | E |
| [43] | D |
| [44] | C |