**Java e Orientação a Objetos**

1 – Atributos default são visíveis apenas entre classes de mesmos pacotes. Afirmativa correta. Outros nomes para a visibilidade default: **package**.

2 – Métodos protegidos são acessíveis em classes de mesmo pacote. Correto, assim como métodos e atributos default, também são visíveis dentro de um mesmo pacote.

3 – Classes internas podem ter qualquer nível de visibilidade. Correto, podem possuir os quatro níveis de visibilidade.

4 – Atributos protegidos de superclasses são acessíveis pelas subclasses. Correto.

5 – Acesso a métodos privados está restrita a própria classe que contém o método privado.

6 – Constantes em Java são declaradas com as palavras chaves **static** e **final**, e por convenção é utilizado letras maiúsculas.

7 – Um método definido como final não poderá ser sobrescrito.

8 – Um método da subclasse com visibilidade mais restritiva do que o da superclasse não será compilado. Exemplo: método foi definido como public na superclasse e definido como private na subclasse.

9 – A visibilidade do método da subclasse poderá ser menos restritivo que o método sobrescrito da super classe.

10 – Ao sobrescrever um método da superclasse, o método não poderá ter uma lista de argumentos diferentes.

11 – Uma classe abstrata não precisa necessariamente ter um método abstrato.

12 – Uma classe abstrata poderá somente ser herdada e nunca instanciada.

13 – Uma classe não poderá ser **abstratic e final** ao mesmo tempo. Ocorrerá erro de compilação.

14 – Métodos abstratos de superclasse deverão ser implementados em classes concretas, mas não em classes abstratas. Uma classe abstrata poderá herdar uma concreta. Uma classe será obrigatoriamente abstrata se possuir um método abstrato e podem possuir métodos com corpo (não abstratos).

15 – Não é possível sobrescrever métodos construtores. Super() é utilizado para chamar o construtor da super classe. Métodos construtores não possuem tipos de retorno. Uma chamada this() pode ser sem argumentos (chamando o construtor padrão dentro da mesma classe) ou ser chamada com argumentos para chamar um construtor especifico.

Como uma classe abstrata não pode ser instanciada, para acessar seu construtor, a subclasse deverá utilizar super(), ou então this() se estiver dentro da própria classe abstrata.

16 – Todos os métodos de interfaces devem ser abstratos e ter visibilidade somente publica. Uma interface não poderá ser instanciada, pois é considerada uma classe 100% abstrata. Interfaces podem ser utilizadas de forma polimórfica. Uma classe só pode estender uma única classe, pois não existe herança múltipla. Métodos de uma interface são públicos e abstratos, sempre. Uma interface não implementa outras interfaces, mas herda ou estende outras interfaces. O método de uma interface não poderá ser marcado como final.

17 – Membros estáticos (variáveis e método) pertencem à classe e não a instancia (objeto). Métodos estáticos não podem ser sobrescritos, mas sim redefinidos. Variáveis e métodos estáticas podem ter qualquer tipo de visibilidade.

18 - Da documentação Java, temos: "A classe Throwable  é a superclasse de todos os Erros e Exceções da linguagem Java. Apenas objetos que são instância dessa classe (ou uma de suas subclasses) podem ser lançados pela palavra chave throw ou capturados pela cláusula catch".

JavaEE

**O *GlassFish*é um projeto de servidor de aplicativos de código aberto iniciado pela Sun Microsystems para a plataforma Java EE e agora patrocinado pela Oracle Corporation.**

**Apache Geronimo é um servidor de aplicativos de código aberto desenvolvido pela Apache Software Foundation e distribuído sob a licença Apache.**

**O Apache Tomcat , muitas vezes referido como Tomcat Server , é um Servlet Container de Java de código aberto desenvolvido pela Fundação Apache Software (ASF). O Tomcat implementa várias especificações Java EE, incluindo Java Servlet , JavaServer Pages (JSP), Java EL e WebSocket , e fornece um ambiente de servidor web HTTP " Java puro " no qual  código Java pode ser executado.**

**JBoss  é um servidor de aplicativos criado por JBoss , agora desenvolvido pela Red Hat . O WildFly é escrito em Java e implementa a especificação Java Platform, Enterprise Edition (Java EE). Ele é executado em várias plataformas.**

***Containers:***

Os seguintes tipos de containers existem e executam no servidor J2EE:

 - **Container EJB**: container para acolher algumas ou todas as Enterprise Beans (EJBs) de uma aplicação

 - **Web container**: container para acolher algumas ou todas as JSPs e servlets de uma aplicação

Os seguintes tipos de containers existem e executam na máquina cliente:

 - **Application Client Containe**r: para executar uma aplicação "console", também chamadas de aplicações Java Standallone.

***Documentação da Oracle:***

The server and containers are as follows:

**Java EE server**: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.

**EJB container:**Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.

**Web container:** Manages the execution of web pages, servlets, and some EJB components for Java EE applications. Web components and their container run on the Java EE server.

**Application client container:** Manages the execution of application client components. Application clients and their container run on the client.

**Applet container:**Manages the execution of applets. Consists of a web browser and a Java Plug-in running on the client together.

***Injeção de Dependências e Contexto – CDI***

Especificada por JSR-299, é parte integrante do Java EE 6 e fornece uma arquitetura que permite aos componentes do Java EE, como os servlets, enterprise beans e JavaBeans, existirem dentro do ciclo de vida de uma aplicação com escopos bem definidos.   
Além disso, os serviços CDI permitem que os componentes do Java EE, como beans de sessão EJB e beans gerenciados do JavaServer Faces (JSF), sejam injetados e interajam de maneira acoplada flexível, disparando e observando eventos.

Se tratando do conteiner Java EE. Para se definir um alvo como injeção usa a **annotation** **"@Inject".**

Para o caso proposto, o container é que, supostamente de acordo a questão, possuir capacidade para inserir automaticamente a referências a outros elementos. Se o container não fosse explicitamente referenciado na questão, as outras alternativas não estariam necessariamente incorretas.

***Caracteristicas JEE:***

**Transações:**Enterprise Java Beans (EJB) suporta transações, integrado com a Java Transaction API (JTA) e possibilidade de transações distribuídas.

**Segurança:**autenticação e autorização de forma transparente. Os  desenvolvedores não precisam fazer a lógica de segurança, faz parte do JEE.

**Remotabilidade:**acesso remoto por vários protocolos de comunicação. por isso q aplicações EJB podem ser acessadas como Web Services.  
**Multithreading e Concorrência:**acesso por múltiplos usuários simultaneamente de modo controlado.  
**Persistência:**segue a especificação JPA.  
**Gerenciamento de Objetos:**injeção de dependências e controle de ciclo de vida, garantindo a escalabilidade.  
**Integração:**com os componentes da plataforma Java  
EE. Podemos, por exemplo, facilmente integrar os recursos do JSF em uma aplicação EJB.

**JSON** é API encontrada a partir do Java EE 7, e sua a manipulação na linguagem Java se dá através da biblioteca “org.json”.

O **JSON-P**, que é uma API Java EE 7, permite que aplicações Java EE analisem, transformem e consultem dados no formato JSON.

A **JAXB** é uma das APIs da plataforma Java EE e fornece suporte à manipulação de objetos Java e XML. Sua principal característica é a capacidade de vincular XML a objetos Java e vice-versa. E o melhor é que ela já está incluída na JDK 6.

Usando JAXB somos capazes de criar facilmente classes a partir de um schema (XSD). Isso mesmo. Com ela somos capazes de transformar um arquivo XSD em um conjunto de classes num piscar de olhos.

**Enterprise JavaBeans (EJB)** é um componente da plataforma JEE que roda em um container de um servidor de aplicação. Seu principal objetivo consiste em fornecer um desenvolvimento rápido e simplificado de aplicações Java, com base em componentes distribuídos, transacionais, seguros e portáveis. Atualmente, na versão 3.1, o EJB tem seu futuro definido conjuntamente entre grandes empresas como IBM, Oracle e HP, como também por uma vasta comunidade de programadores numa rede mundial de colaboração sob o portal do JCP.

**Servlet** - Gera conteúdo dinâmico por meio de um modelo de requisição-resposta

**JDBC** - Permite executar comandos SQL a partir de métodos de classes Java

**EJB** - Implementa lógica de negócios

**JPA** - É uma especificação baseada no conceito de POJOs para padronizar o mapeamento objeto/relacional (O/R) em Java

**JTA** - Fornece uma interface padrão para a demarcação de transações

**JMS** - Permite que componentes de aplicação JEE possam criar, enviar, receber e ler mensagens

**JSF** - É framework MVC para desenvolvimento de aplicações web e um modelo de interfaces gráficas baseado em eventos.

**Sempre que estivermos usando os métodos:**  
doGet, if the servlet supports HTTP GET requests  
doPost, for HTTP POST requests  
doPut, for HTTP PUT requests  
doDelete, for HTTP DELETE requests  
init and destroy, to manage resources that are held for the life of the servlet  
getServletInfo, which the servlet uses to provide information about itself

Estamos lidando com a classe **HttpServlet**.

**JBoss: o** Standalone Mode foi adicionado nesta versão. Quando o gerenciamento centralizado não é necessário, ou se você for um desenvolvedor trabalhando em um projeto usando JBoss Developer Studio (JBDS), modo standalone é o mais correto.  
O Domain Mode é uma das novidades trazidas pela JBoss AS7. Este modo fornece um único ponto de controle chamado de controlador de domínio para executar e gerenciar uma coleção de servidores.

**O RMI (Remote Method Invocation)** é uma interface de programação que permite a execução de chamadas remotas no estilo RPC em aplicações desenvolvidas em Java. É uma das abordagens da plataforma Java para prover as funcionalidades de uma plataforma de objetos distribuídos. Esse sistema de objetos distribuídos faz parte do núcleo básico de Java desde a versão JDK 1.1, com sua API sendo especificada através do pacote java.rmi e seus subpacotes1 .  
Através da utilização da arquitetura RMI, é possível que um objeto ativo em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais.

A API RMI fornece ferramentas para que seja possível ao programador desenvolver uma aplicação sem se preocupar com detalhes de comunicação entre os diversos possíveis elementos (hosts) de um sistema.

**Java Authentication and Authorization Service (JAAS):** autenticação e autorização de usuário ou grupo de maneira desacoplada. JEE permite restrições de segurança definidas em deployment, tornando aplicações portáteis para diferentes fornecedores. controle de acesso declarativo e pelo desenvolvedor, interpretado quando a aplicação é instalada no servidor, poupando os desenvolvedores da complexidade da segurança. O JEE permite também segurança externa, como Active Directory e LDAP, por exemplo, sem nenhuma alteração no código fonte da mesma. O **EJB Container** gerencia Enterprise Java Beans, responsável pela maioria dos serviços a requisitos não funcionais. ele faz acesso aos dados, visto que JTA não no Webcontainer.

A partir da interpretação do trecho JSF (JavaServer Faces), versão 2, no código a seguir, verifica-se que uma providência válida é configurar o managed-bean clientePage no arquivo faces-config.xml.

<f:view>

<h:form id=”clienteForm”> <h:outputLabel for=”informeNome” value=”Informe Nome”/> <h:inputText id=”informeNome” value =”#{clientePage.nome}”/>

<h:commandButton value=”Nome do Cliente” action=”#{clientePage.cliente}”/> </h:form> </f:view>