1. How to deploy a servlet context listener within web application ?

1 point

A: Using <listener> tag in web.xml

B: Using <web-listener> tag

C: Using @WebListener annotation

D: A & C

A servlet context listener can be deployed within a web application using either the <listener> tag in the web.xml deployment descriptor or the @WebListener annotation in the Java code.

2. How many ServletConfig & ServletContext objects will be created by web container?

1 point

A Only single one per entire web server.

B One per servlet & one per web application

C. One per context & one per application

D One per every servlet.

* ServletConfig: Each servlet has its own ServletConfig object created by the web container when the servlet is initialized. This object provides the servlet with information about its configuration, such as its name, initialization parameters, and servlet context.
* ServletContext: There is only one ServletContext object created per web application by the web container. This object provides information about the entire web application, such as the context path, classloader, and attributes shared by all servlets within the application.

3. Consider following 2 servlets.

@WebServlet(value="/s1",loadOnStartup=1)

public class FirstServlet extends HttpServlet {...}

@WebServlet(value="/s2")

public class SecondServlet extends HttpServlet {...}

Which of the following is true ?

1 point

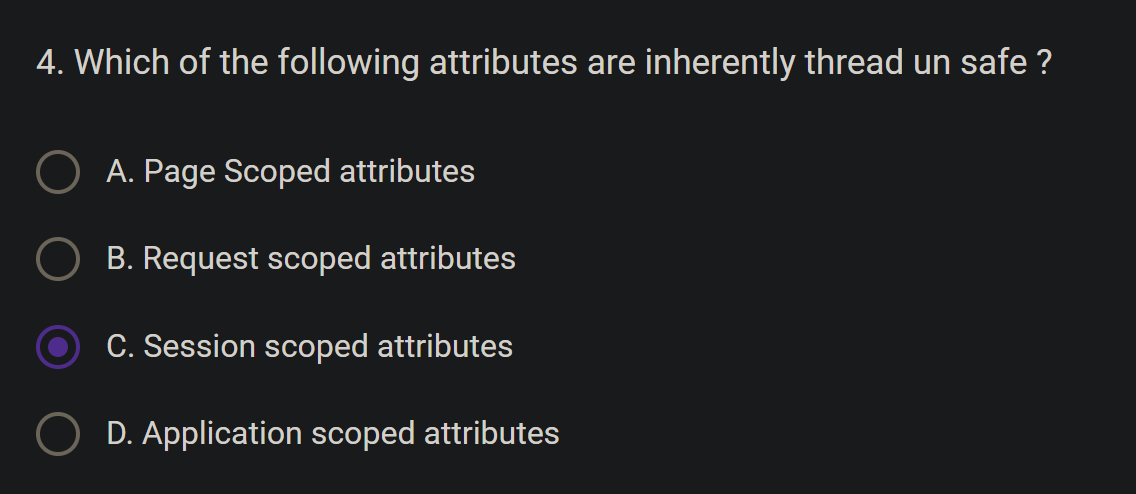
A. Web container doesn't invoke init method of the second servlet.

B Web container invokes init method of the second servlet at web app start up.

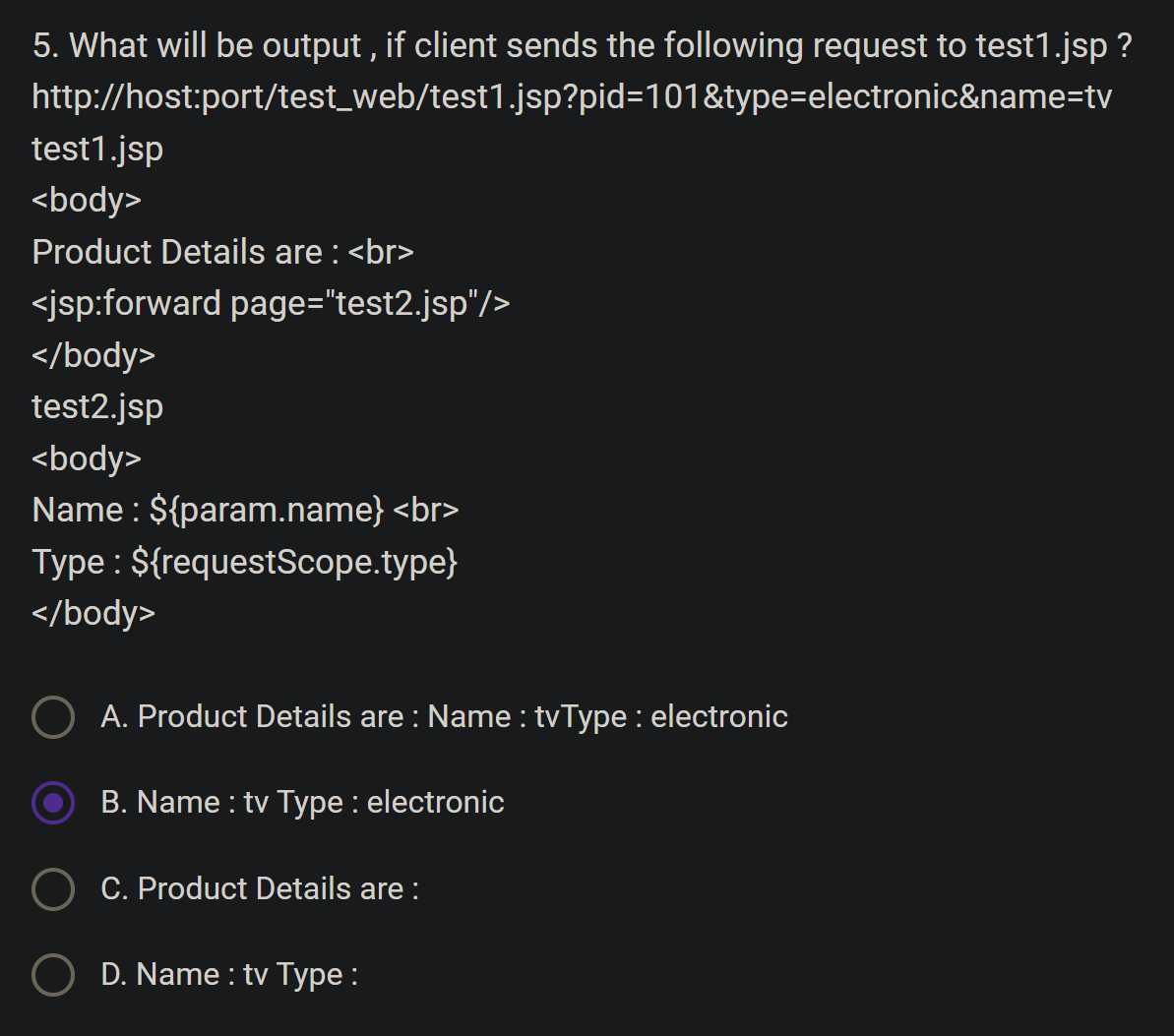
C. Web container invokes init method of the first servlet at web app start up.

D. Web Container starts life cycle of the first servlet , after clint request.

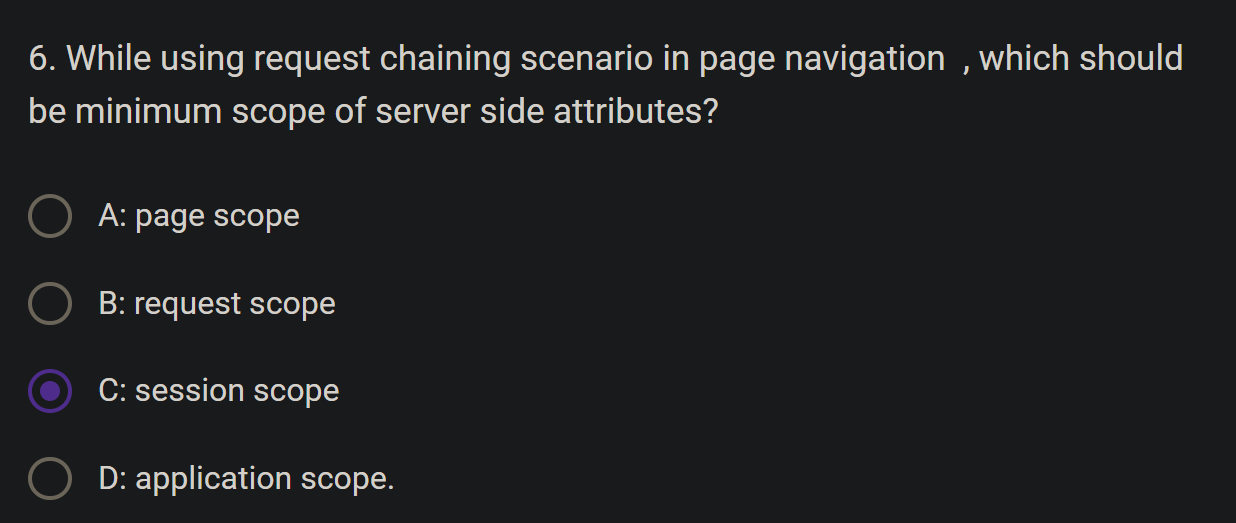
* C



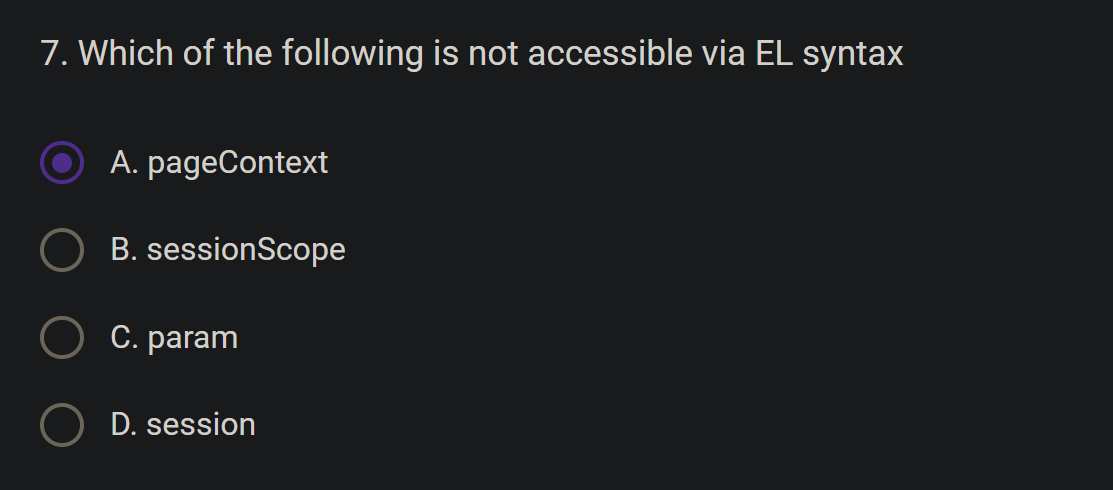
* D

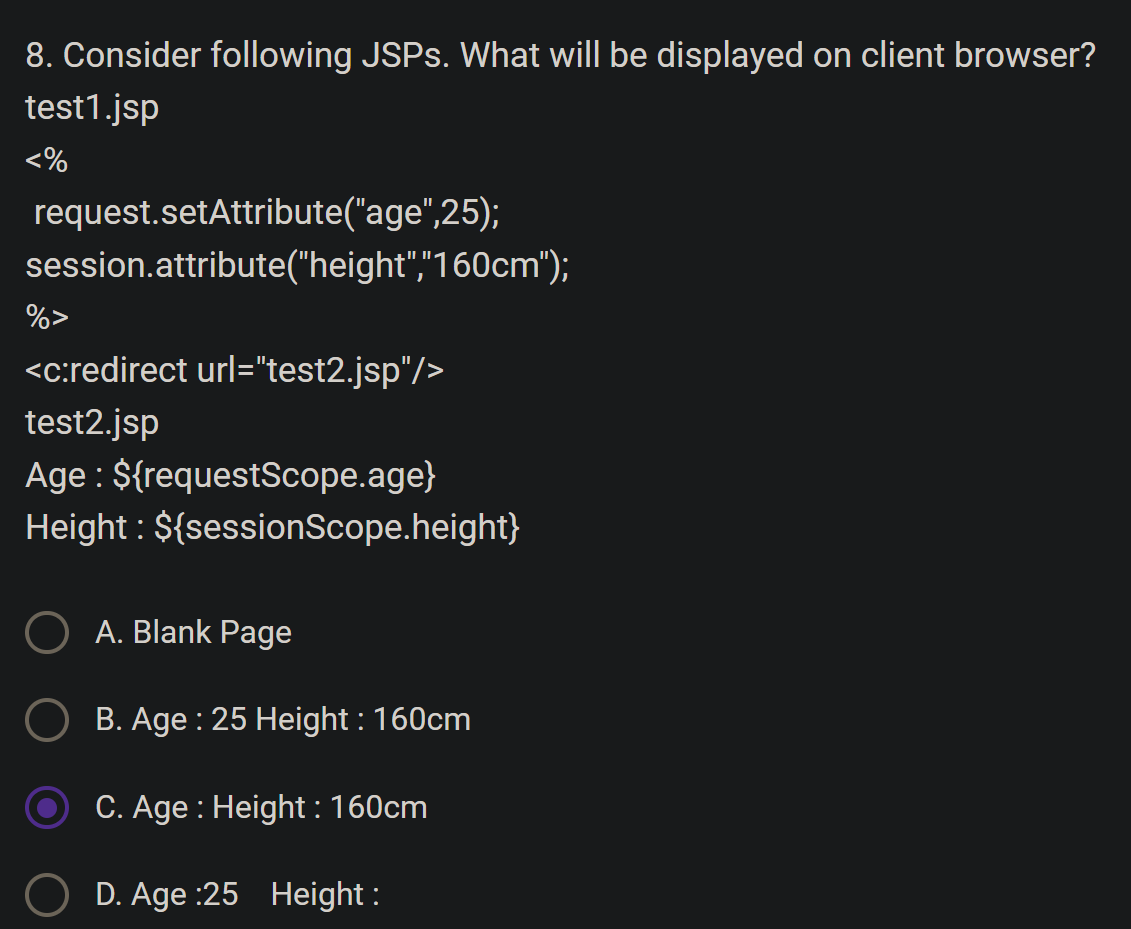


* D



Ans 🡺 Request scope





Explanation:

test1.jsp executes:

request.setAttribute("age", 25); sets the request attribute "age" to 25.

session.setAttribute("height", "160cm"); sets the session attribute "height" to "160cm".

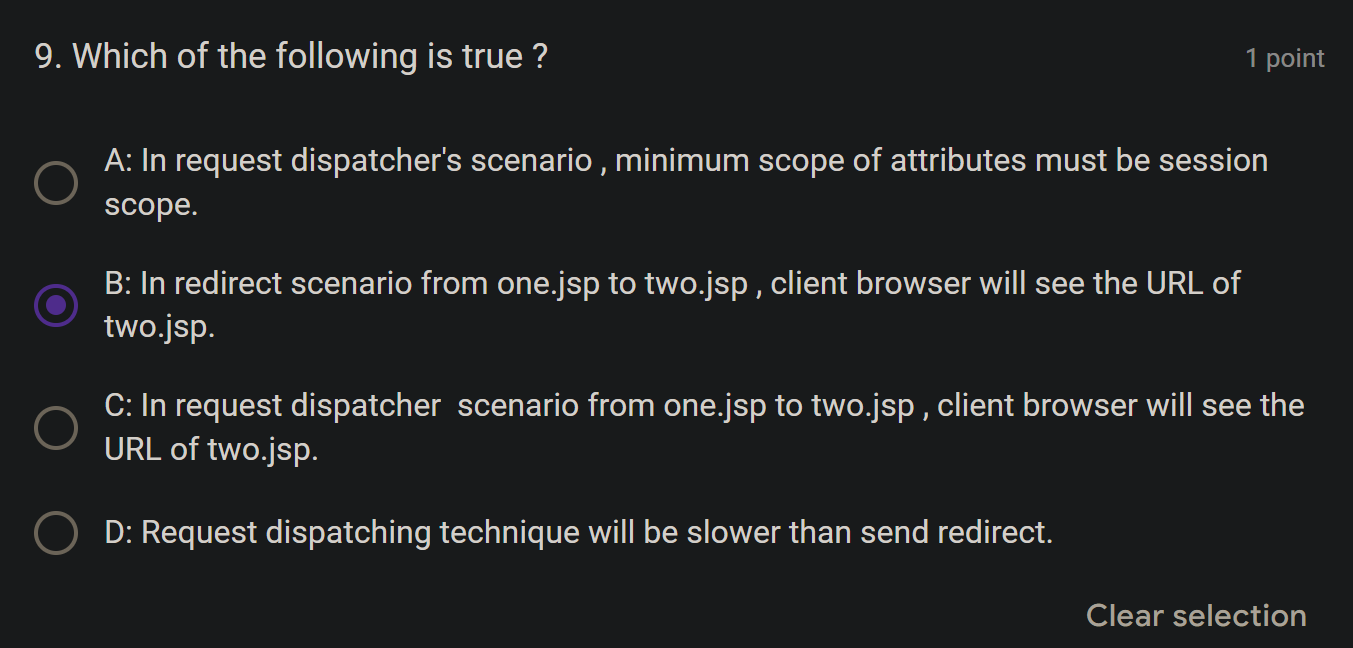
<c:redirect url="test2.jsp"/> initiates a redirect to test2.jsp, initiating a new request.

New request starts:

* The redirect creates a new, separate request to test2.jsp.
* Request attributes do not persist across redirects, so the "age" attribute is lost.
* Session attributes, however, do persist across redirects, so the "height" attribute remains accessible.

Key points:

* Redirects create new requests, discarding request attributes.
* Session attributes persist across redirects within the same session.



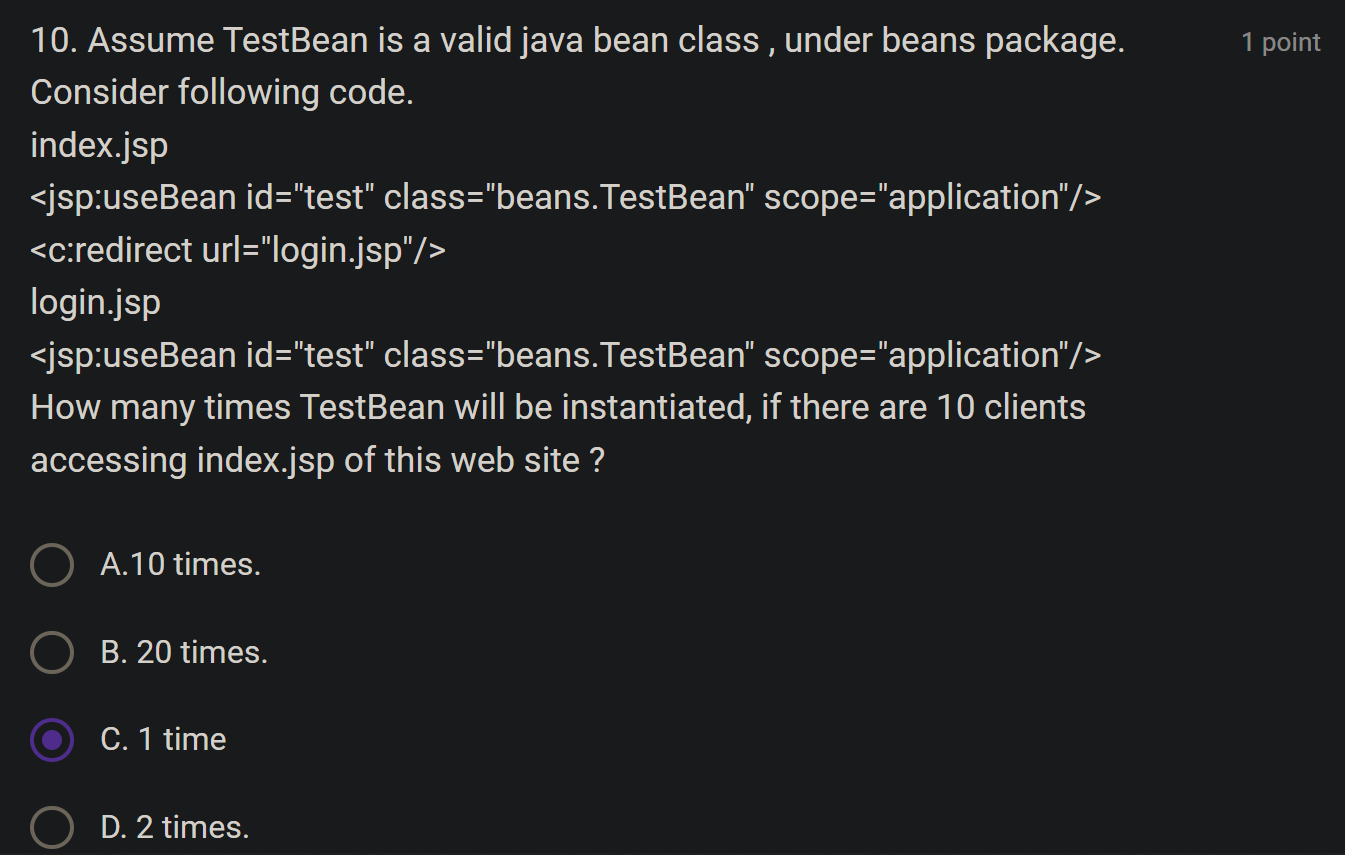
The correct answer is B: In redirect scenario from one.jsp to two.jsp, client browser will see the URL of two.jsp.

Here's why the other options are incorrect:

* A: In request dispatcher's scenario, minimum scope of attributes must be session scope. This is false. Request dispatchers can work with attributes in any scope, including request scope.
* C: In request dispatcher scenario from one.jsp to two.jsp, client browser will see the URL of two.jsp. This is false. Request dispatchers process the request internally on the server, so the client browser still sees the original URL of one.jsp.
* D: Request dispatching technique will be slower than send redirect. This is generally false. Request dispatchers often perform faster than redirects because they avoid the overhead of an additional request-response cycle.

Key differences between request dispatchers and redirects:

* Request dispatchers: Server-side forwarding, keeping the same request context and URL.
* Redirects: Client-side redirection, initiating a new request and changing the URL.



The TestBean will be instantiated only once, regardless of how many clients (10 or more) access index.jsp.

Here's why:

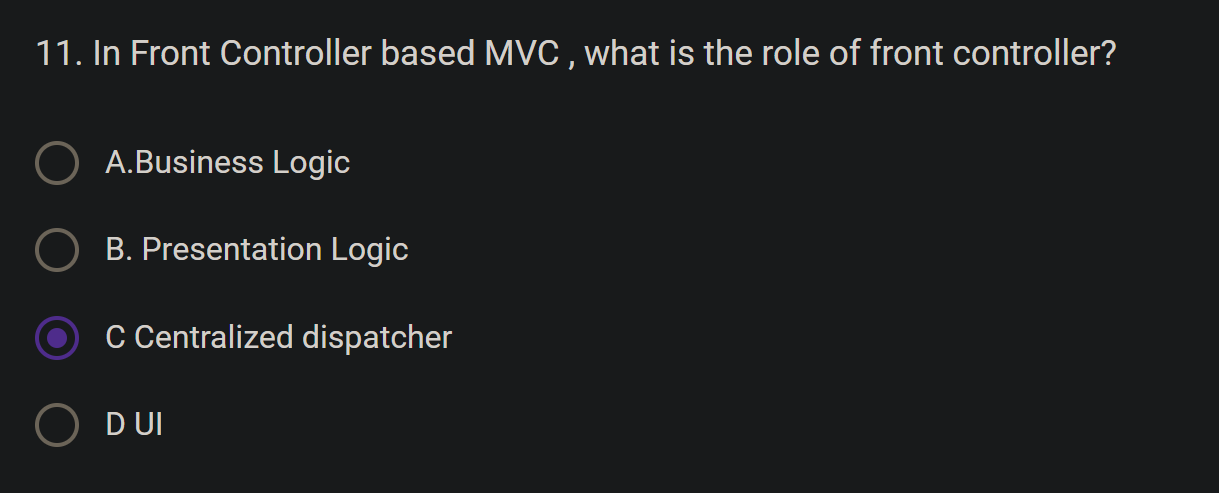
* Application Scope: The <jsp:useBean> tag in index.jsp specifies scope="application", meaning the TestBean instance will be accessible to all users and pages within the entire web application.
* Single Instance: The application server creates a single instance of TestBean and stores it in the application scope when the first client accesses index.jsp.
* Shared Instance: Subsequent requests, even from different clients, will reuse the same TestBean instance from the application scope.
* Redirect Doesn't Matter: The redirect to login.jsp doesn't affect the scope or lifetime of the TestBean. Since it's in application scope, it persists across redirects and different pages within the application.

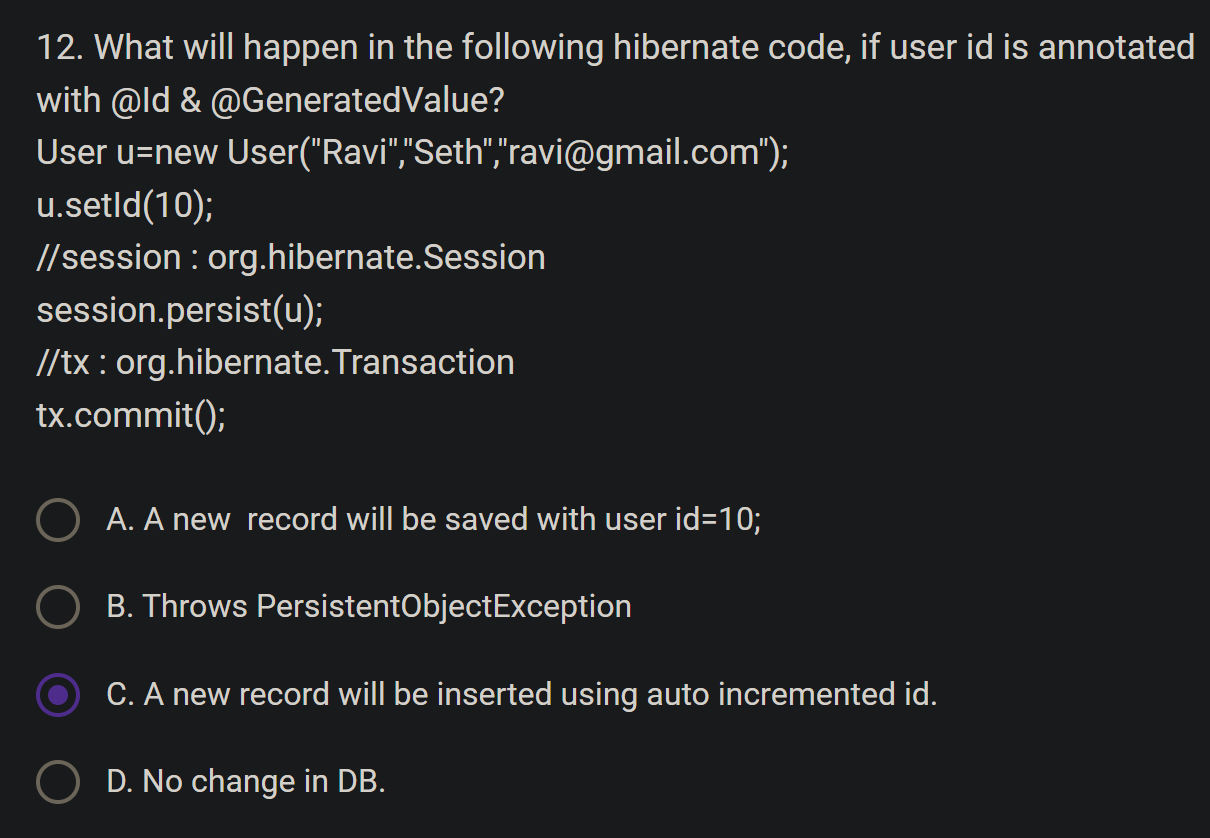
Key points:

* Application scope has the widest visibility and longest lifespan among scopes.

Application-scoped beans are shared across all users and requests.

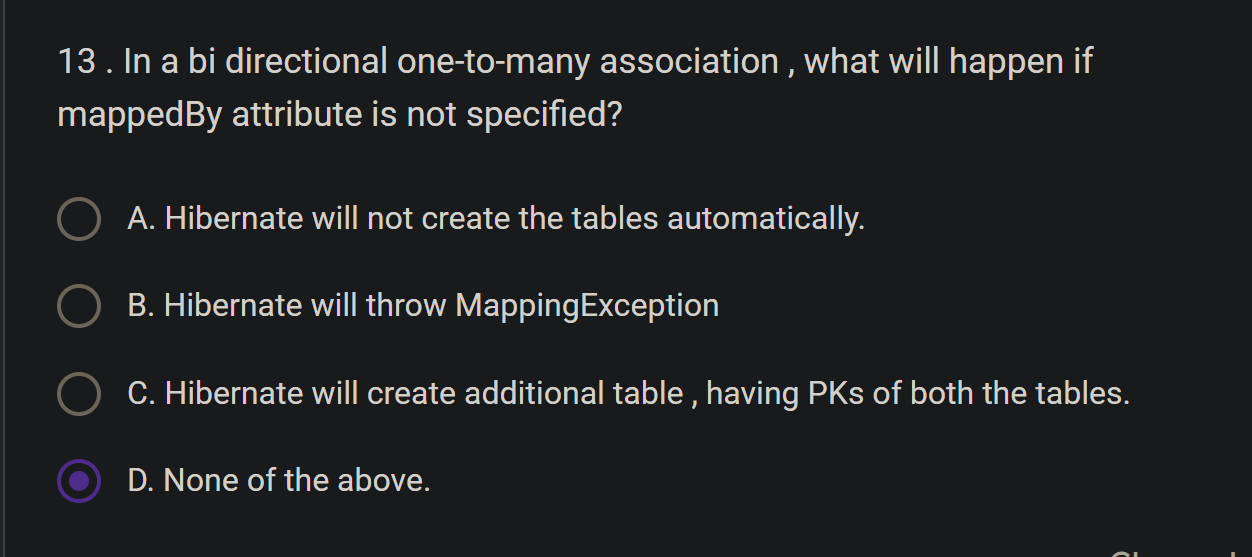
Redirects don't affect application-scoped beans.

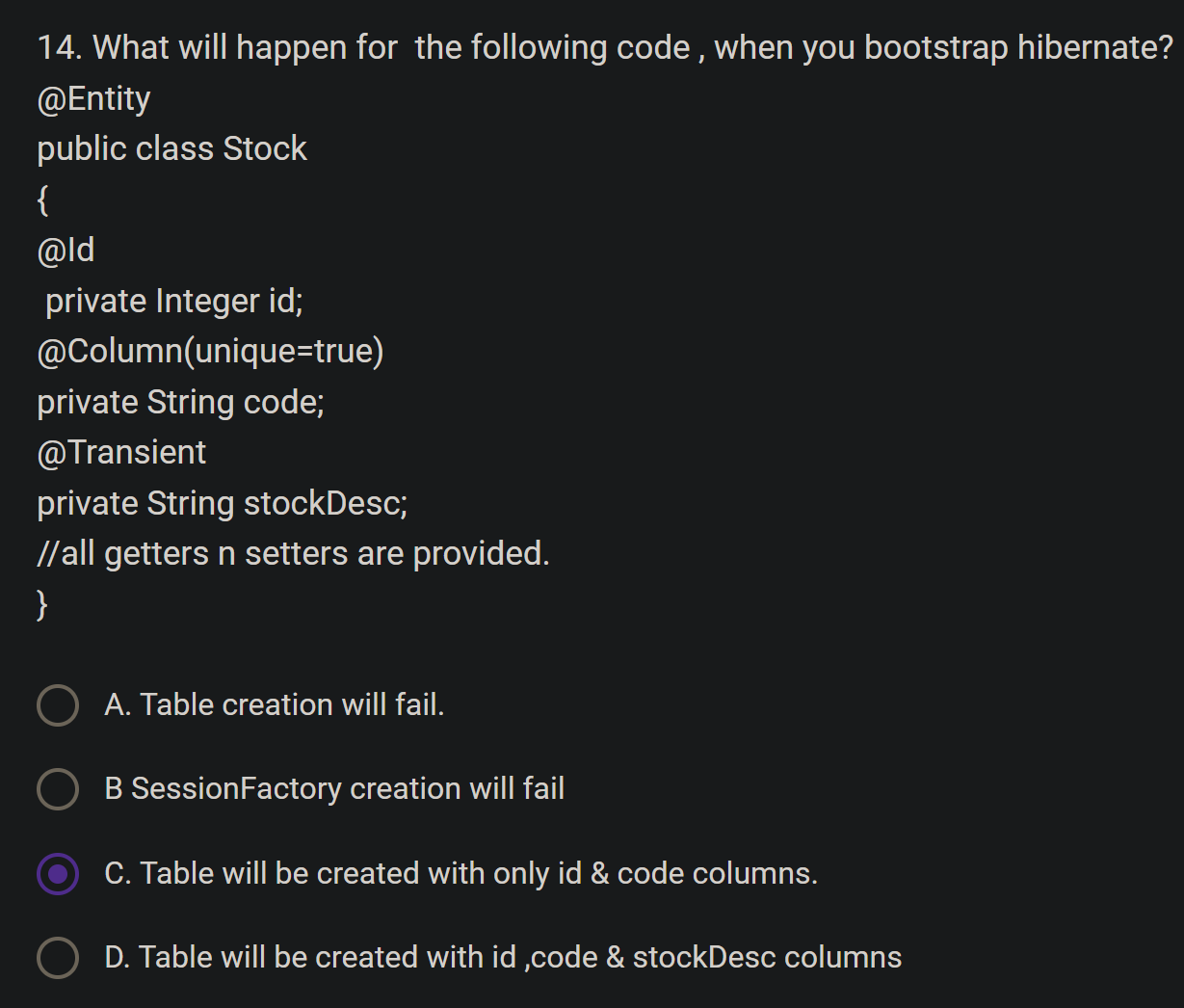


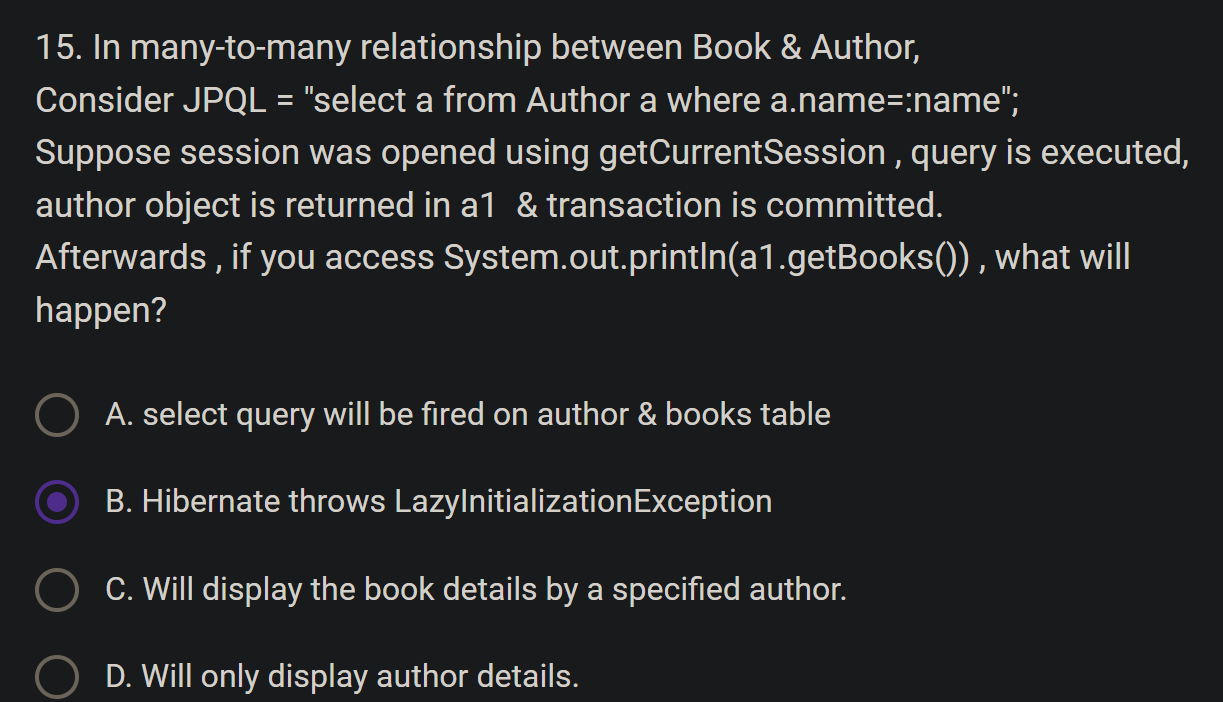


* User object creation: A new User object is created with the given name, surname, and email, but setting the ID manually with u.setId(10) will be ignored due to the presence of @GeneratedValue.
* Ignoring manually set ID: Hibernate will detect that the id property is annotated with both @Id (marking it as the primary key) and @GeneratedValue (indicating that its value should be automatically generated by the persistence provider). It will disregard the manually set value of 10.
* Generating primary key: Hibernate will use its configured ID generation strategy to create a new unique ID for the User object. The specific strategy used depends on the configuration, but common approaches include:
  + AUTO: Hibernate chooses the most appropriate strategy based on the underlying database.
  + IDENTITY: Relies on the database's auto-incrementing column feature.
  + SEQUENCE: Uses a database sequence to generate unique values.
  + TABLE: Employs a dedicated table to keep track of generated IDs.
  + Persisting the object: The session.persist(u) call signals Hibernate to persist the User object to the database, using the generated ID.
* Transaction commit: The tx.commit() call instructs Hibernate to execute the SQL statements necessary to insert the new User record into the database with the generated ID.

Therefore, the user will be persisted with a new, automatically generated ID, not the manually set value of 10. Hibernate takes precedence over manual ID setting when @GeneratedValue is present



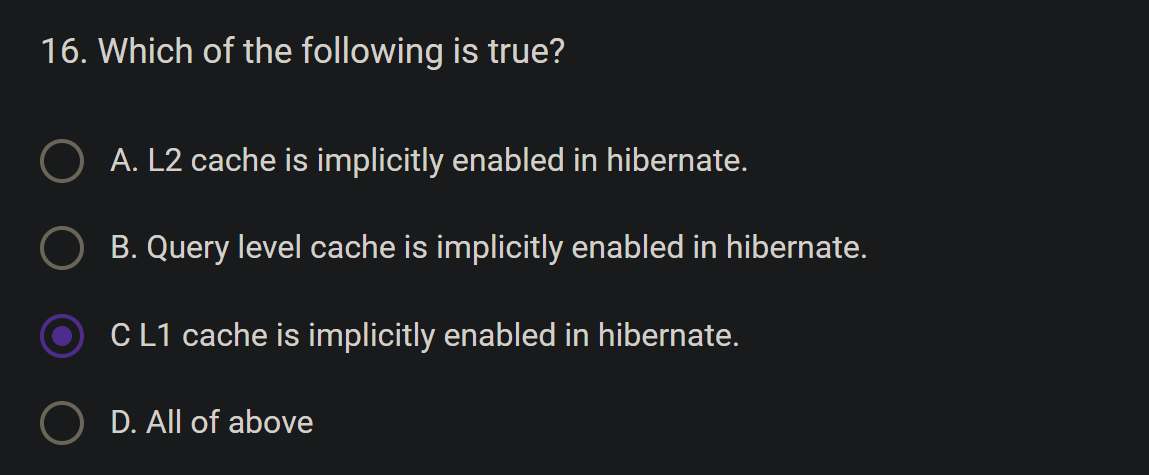




The correct answer is B: Hibernate throws LazyInitializationException.

Here's why:

* Lazy Loading: The getBooks() method in the Author entity is likely annotated with @ManyToMany(fetch = FetchType.LAZY), which means the associated books are not loaded eagerly when the author object is fetched. This is a common optimization to avoid loading unnecessary data until it's actually needed.
* Session Closed: The session was opened using getCurrentSession and committed after retrieving the author object. This means the session is now closed, and Hibernate can no longer interact with the database within that session's context.
* LazyInitializationException: When you try to access a1.getBooks() after the session is closed, Hibernate cannot fetch the associated books from the database because it doesn't have an active session. This leads to a LazyInitializationException being thrown.



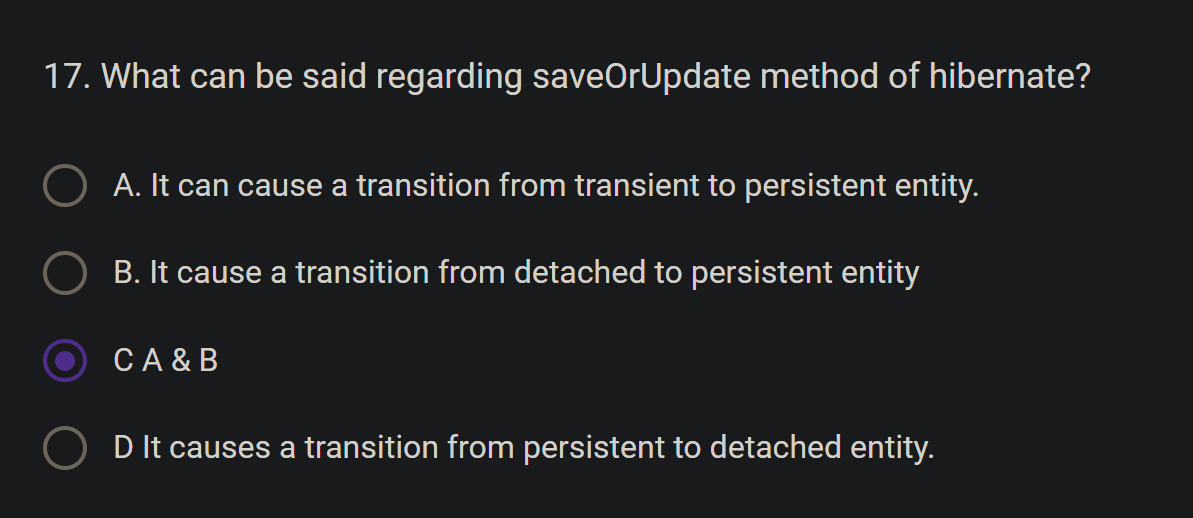
The correct answer is C: L1 cache is implicitly enabled in hibernate.

Here's why:

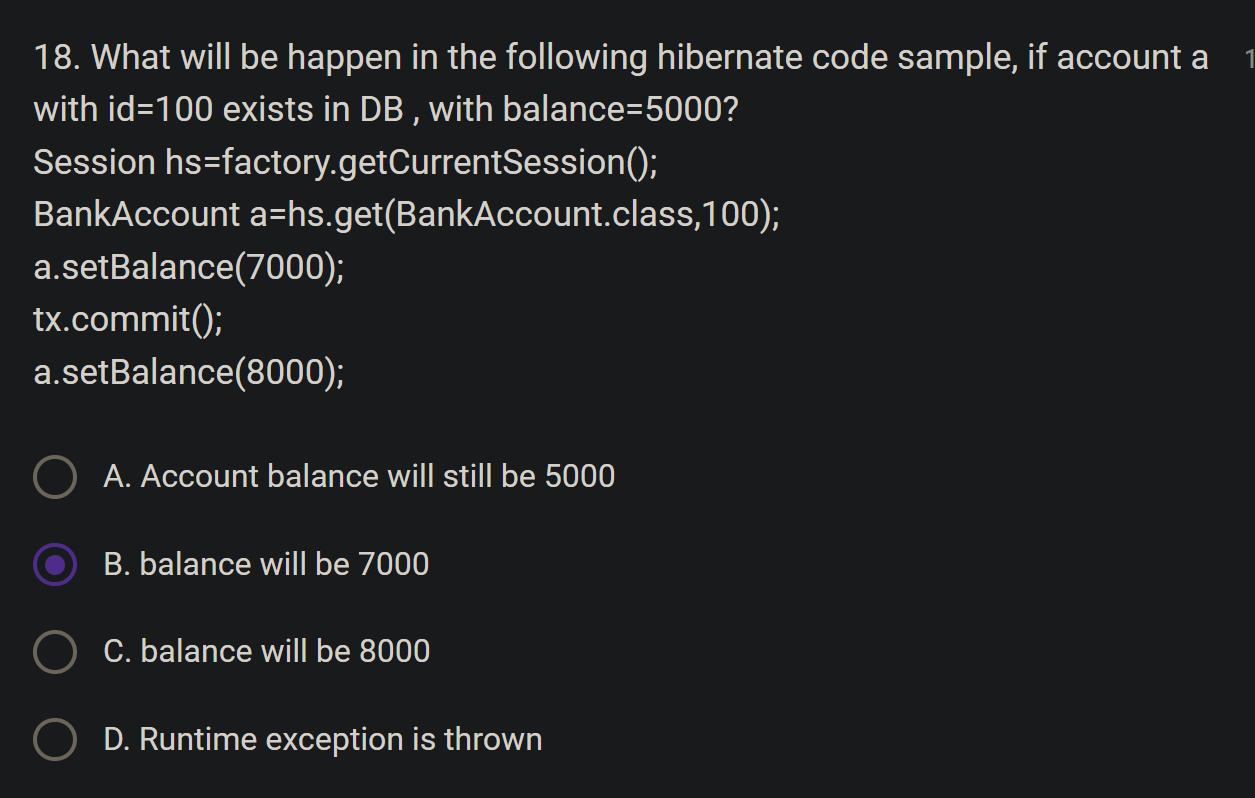
* L1 cache: The first-level cache, also known as L1 cache, is implicitly enabled in Hibernate and automatically stores the entities loaded within the current session. It provides fast access to recently accessed data without hitting the database.
* L2 cache: The second-level cache, or L2 cache, is not enabled by default in Hibernate. You need to explicitly configure it using a caching provider like Ehcache or Infinispan. L2 cache enables sharing of persistent data across different sessions within the same application server.
* Query level cache: Hibernate also offers query caching, but it's separate from L1 and L2 caches and also needs explicit configuration. Query cache stores the results of specific queries to further optimize database access.

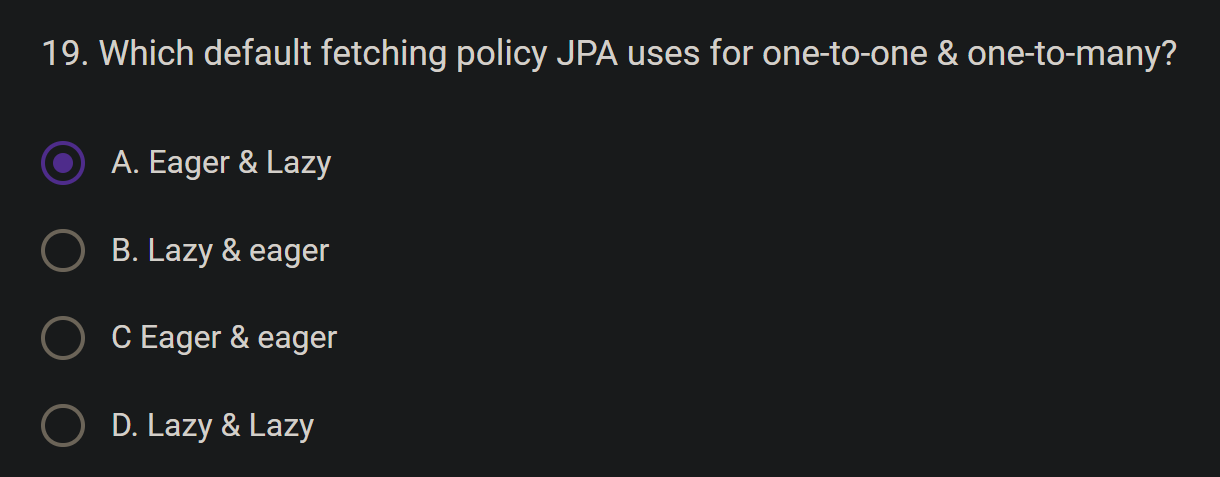
Remember:

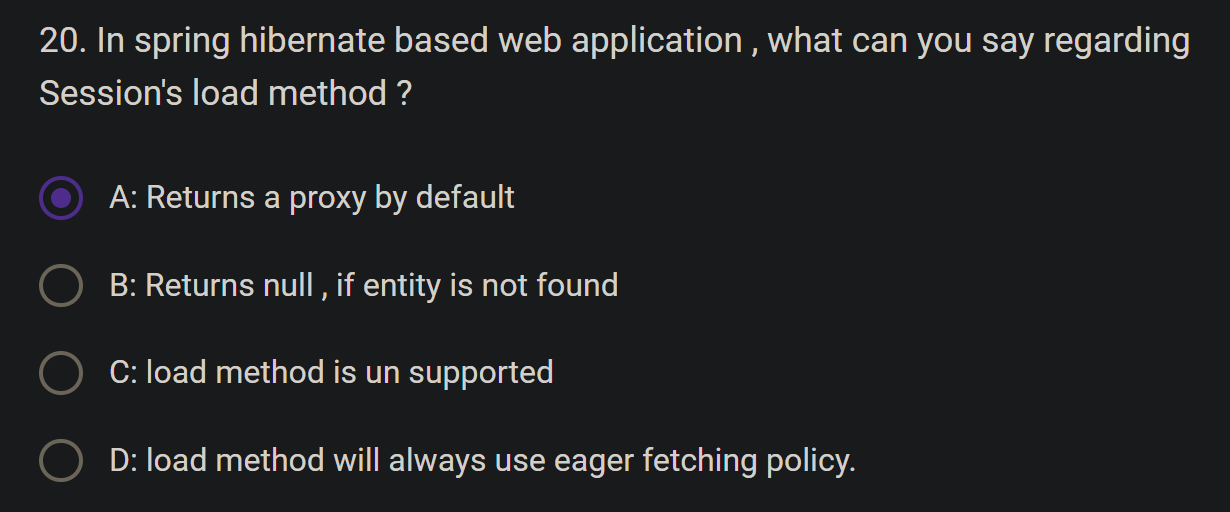
* L1 cache provides short-lived, session-scoped caching.
* L2 cache offers broader, application-scoped data sharing.
* Query cache optimizes specific queries based on their results.



The saveOrUpdate method of Hibernate can cause a transition from transient to persistent entity or from detached to persistent entity. It checks if the given entity is transient (i.e., not associated with any session) or detached (i.e., previously persistent but not associated with any session). If the entity is transient, it will cause a transition to persistent entity by inserting a new record into the database. If the entity is detached, it will cause a transition to persistent entity by updating the existing record in the database. Therefore, the correct answer is C (A & B).







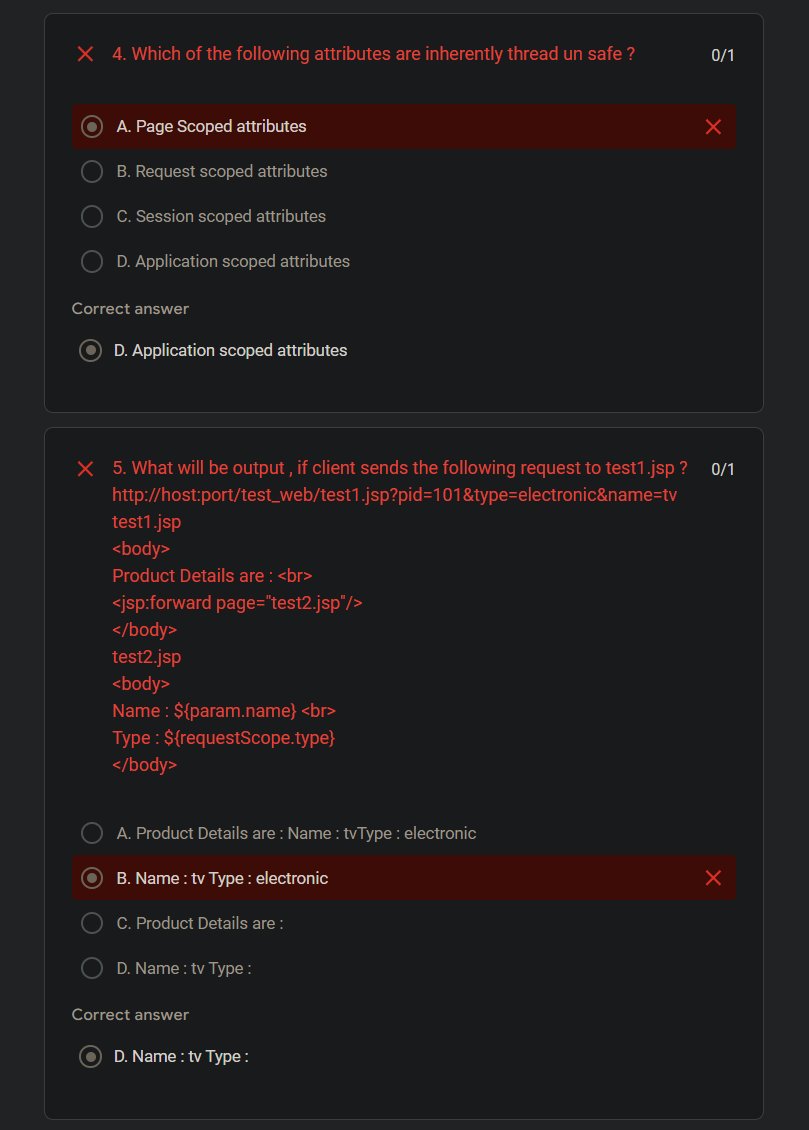
The correct answer is A: Returns a proxy by default.

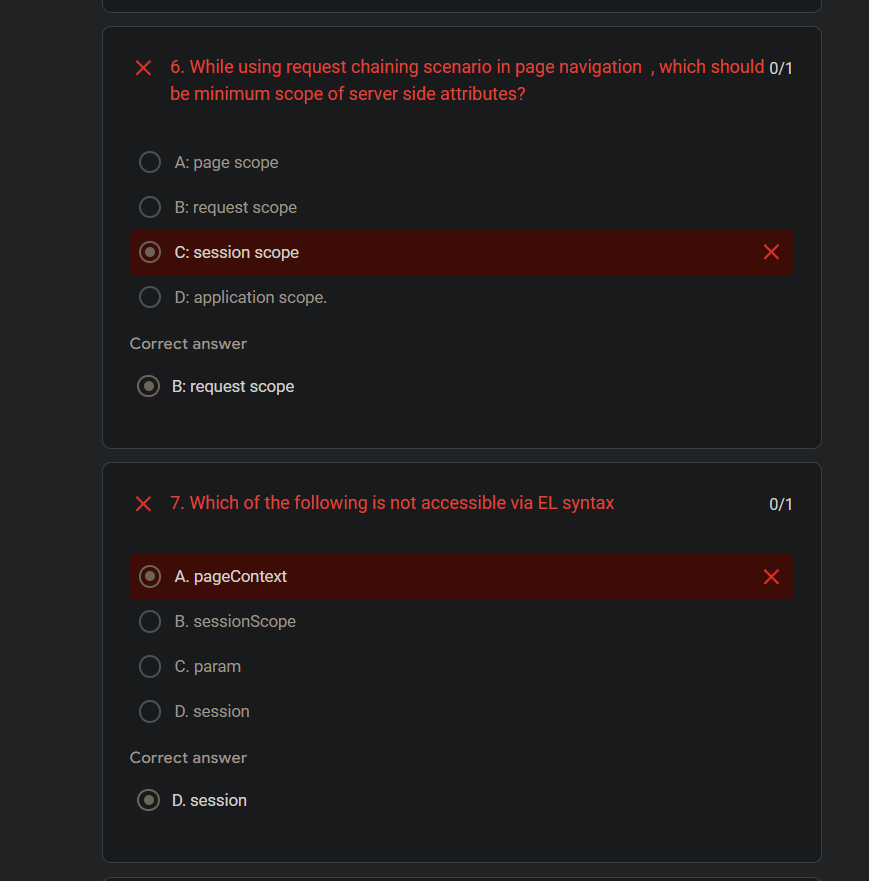
Here's a breakdown of the key characteristics of the Session.load() method in a Spring Hibernate web application:

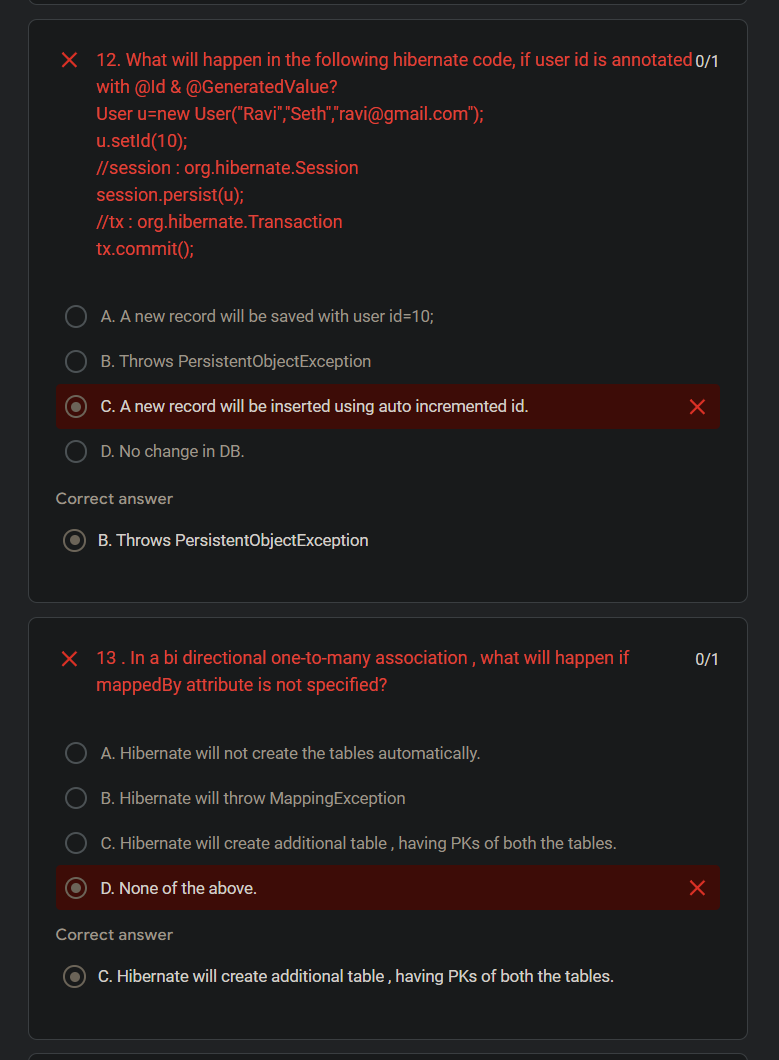
* Proxy by Default: When you call Session.load(EntityClass.class, id), Hibernate returns a proxy object instead of loading the actual entity from the database immediately. This proxy acts as a placeholder for the real entity.
* Lazy Loading: The actual database access occurs only when you try to access a property of the proxy object for the first time. This is known as lazy loading, and it helps optimize performance by avoiding unnecessary database queries.
* Exception if Not Found: If the entity with the specified ID doesn't exist in the database, Hibernate throws an ObjectNotFoundException when you attempt to access a property of the proxy. It doesn't return null.
* Eager Fetching: While Session.load() often uses lazy loading, it can also employ eager fetching in certain scenarios. For example, if you have a one-to-one association mapped with FetchType.EAGER, Hibernate might eagerly fetch the associated entity along with the main entity using Session.load().

Key Points:

* Session.load() is primarily used when you're confident the entity exists and want to avoid unnecessary database calls.
* It's best suited for retrieving entities by their primary key when you're certain of their existence.
* Be mindful of potential exceptions if the entity might not exist.
* Understand the role of proxies and lazy loading to avoid unexpected behavior.
* I hope this explanation clarifies the behavior of Session.load() in Spring Hibernate applications!







Above paper in text

1. How to deploy a servlet context listener within web application ?

1 point

A: Using <listener> tag in web.xml

B: Using <web-listener> tag

C: Using @WebListener annotation

D: A & C

2. How many ServletConfig & ServletContext objects will be created by web container?

1 point

A Only single one per entire web server.

B One per servlet & one per web application

C. One per context & one per application

D One per every servlet.

3. Consider following 2 servlets.

@WebServlet(value="/s1",loadOnStartup=1)

public class FirstServlet extends HttpServlet {...}

@WebServlet(value="/s2")

public class SecondServlet extends HttpServlet {...}

Which of the following is true ?

1 point

A. Web container doesn't invoke init method of the second servlet.

B Web container invokes init method of the second servlet at web app start up.

C. Web container invokes init method of the first servlet at web app start up.

D. Web Container starts life cycle of the first servlet , after clint request.

4. Which of the following attributes are inherently thread un safe ?

1 point

A. Page Scoped attributes

B. Request scoped attributes

C. Session scoped attributes

D. Application scoped attributes

5. What will be output , if client sends the following request to test1.jsp ?

http://host:port/test\_web/test1.jsp?pid=101&type=electronic&name=tv

test1.jsp

<body>

Product Details are : <br>

<jsp:forward page="test2.jsp"/>

</body>

test2.jsp

<body>

Name : ${param.name} <br>

Type : ${requestScope.type}

</body>

1 point

A. Product Details are : Name : tvType : electronic

B. Name : tv Type : electronic

C. Product Details are :

D. Name : tv Type :

6. While using request chaining scenario in page navigation , which should be minimum scope of server side attributes?

1 point

A: page scope

B: request scope

C: session scope

D: application scope.

7. Which of the following is not accessible via EL syntax

1 point

A. pageContext

B. sessionScope

C. param

D. session

8. Consider following JSPs. What will be displayed on client browser?

test1.jsp

<%

request.setAttribute("age",25);

session.attribute("height","160cm");

%>

<c:redirect url="test2.jsp"/>

test2.jsp

Age : ${requestScope.age}

Height : ${sessionScope.height}

1 point

A. Blank Page

B. Age : 25 Height : 160cm

C. Age : Height : 160cm

D. Age :25 Height :

9. Which of the following is true ?

1 point

A: In request dispatcher's scenario , minimum scope of attributes must be session scope.

B: In redirect scenario from one.jsp to two.jsp , client browser will see the URL of two.jsp.

C: In request dispatcher scenario from one.jsp to two.jsp , client browser will see the URL of two.jsp.

D: Request dispatching technique will be slower than send redirect.

10. Assume TestBean is a valid java bean class , under beans package.

Consider following code.

index.jsp

<jsp:useBean id="test" class="beans.TestBean" scope="application"/>

<c:redirect url="login.jsp"/>

login.jsp

<jsp:useBean id="test" class="beans.TestBean" scope="application"/>

How many times TestBean will be instantiated, if there are 10 clients accessing index.jsp of this web site ?

1 point

A.10 times.

B. 20 times.

C. 1 time

D. 2 times.

11. In Front Controller based MVC , what is the role of front controller?

1 point

A.Business Logic

B. Presentation Logic

C Centralized dispatcher

D UI

12. What will happen in the following hibernate code, if user id is annotated with @Id & @GeneratedValue?

User u=new User("Ravi","Seth","ravi@gmail.com");

u.setId(10);

//session : org.hibernate.Session

session.persist(u);

//tx : org.hibernate.Transaction

tx.commit();

1 point

A. A new record will be saved with user id=10;

B. Throws PersistentObjectException

C. A new record will be inserted using auto incremented id.

D. No change in DB.

13 . In a bi directional one-to-many association , what will happen if mappedBy attribute is not specified?

1 point

A. Hibernate will not create the tables automatically.

B. Hibernate will throw MappingException

C. Hibernate will create additional table , having PKs of both the tables.

D. None of the above.

14. What will happen for the following code , when you bootstrap hibernate?

@Entity

public class Stock

{

@Id

private Integer id;

@Column(unique=true)

private String code;

@Transient

private String stockDesc;

//all getters n setters are provided.

}

1 point

A. Table creation will fail.

B SessionFactory creation will fail

C. Table will be created with only id & code columns.

D. Table will be created with id ,code & stockDesc columns

15. In many-to-many relationship between Book & Author,

Consider JPQL = "select a from Author a where a.name=:name";

Suppose session was opened using getCurrentSession , query is executed, author object is returned in a1 & transaction is committed.

Afterwards , if you access System.out.println(a1.getBooks()) , what will happen?

1 point

A. select query will be fired on author & books table

B. Hibernate throws LazyInitializationException

C. Will display the book details by a specified author.

D. Will only display author details.

16. Which of the following is true?

1 point

A. L2 cache is implicitly enabled in hibernate.

B. Query level cache is implicitly enabled in hibernate.

C L1 cache is implicitly enabled in hibernate.

D. All of above

17. What can be said regarding saveOrUpdate method of hibernate?

1 point

A. It can cause a transition from transient to persistent entity.

B. It cause a transition from detached to persistent entity

C A & B

D It causes a transition from persistent to detached entity.

18. What will be happen in the following hibernate code sample, if account a with id=100 exists in DB , with balance=5000?

Session hs=factory.getCurrentSession();

BankAccount a=hs.get(BankAccount.class,100);

a.setBalance(7000);

tx.commit();

a.setBalance(8000);

1 point

A. Account balance will still be 5000

B. balance will be 7000

C. balance will be 8000

D. Runtime exception is thrown

19. Which default fetching policy JPA uses for one-to-one & one-to-many?

1 point

A. Eager & Lazy

B. Lazy & eager

C Eager & eager

D. Lazy & Lazy

20. In spring hibernate based web application , what can you say regarding Session's load method ?

1 point

A: Returns a proxy by default

B: Returns null , if entity is not found

C: load method is un supported

D: load method will always use eager fetching policy.