upGrad

#LifeKoKaroLift

Session 4 | Data Access Layer: Repositories

Course: Developing RESTful Web APIs Using Spring Boot

Lecture On: Data Access Layer: Repositories

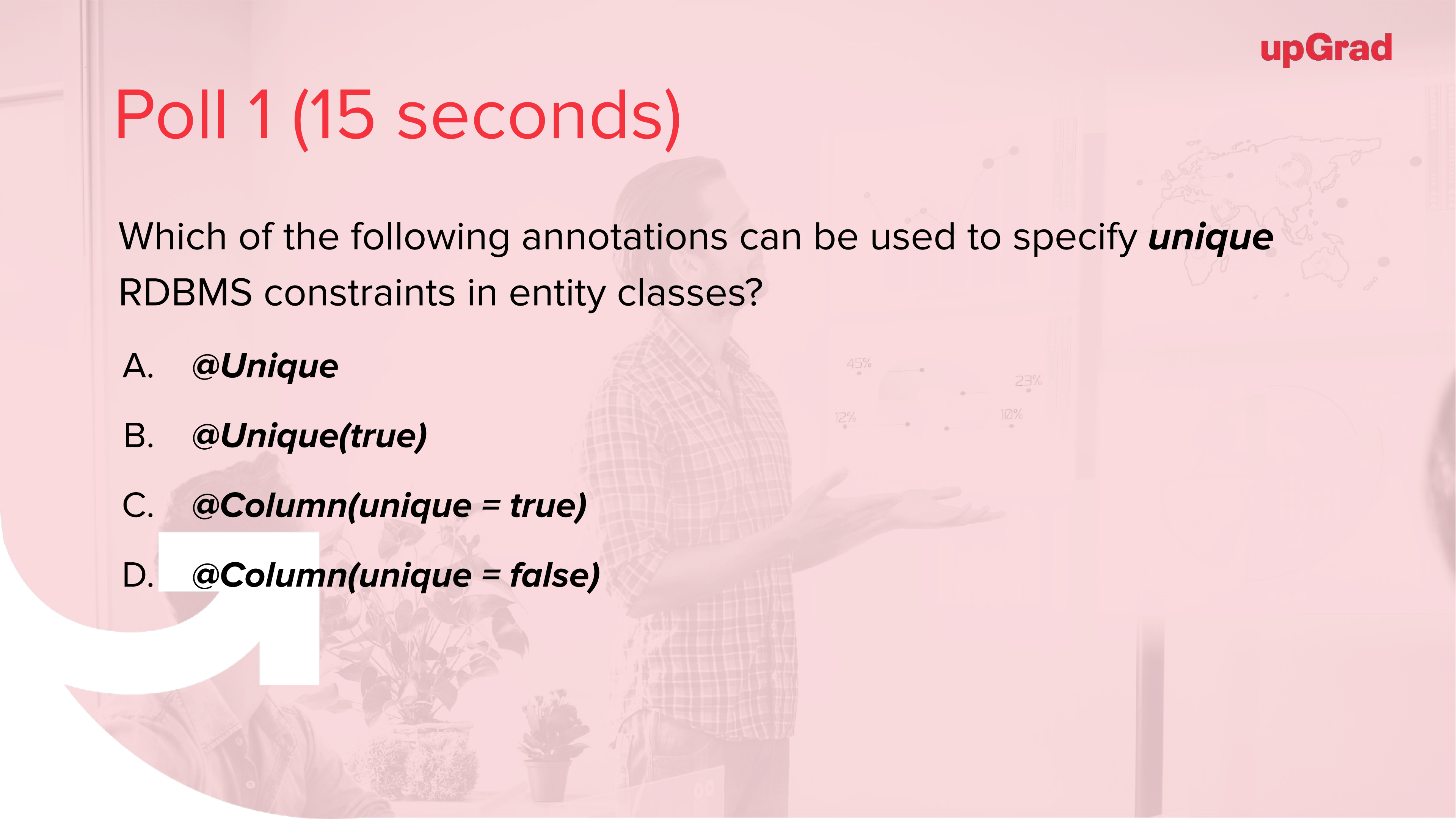
Instructor: Vishwa Mohan

upGrad

Topics covered in the previous class...

1. Introduction to Layered Architecture and Movie Booking Application
2. Different layers of the Movie Booking Application
3. Issues in designing the Data Access Layer using JDBC
4. Solving these issues using Object Relational Mapping (ORM) tools
5. Using Hibernate as an ORM tool and JPA (Java Persistence API) as Java ORM specification
6. Using Spring ORM to integrate Spring applications with JPA and Hibernate
7. Setting up the connection with database
8. Entity creation and basic annotations

upGrad



Poll 1 (15 seconds)

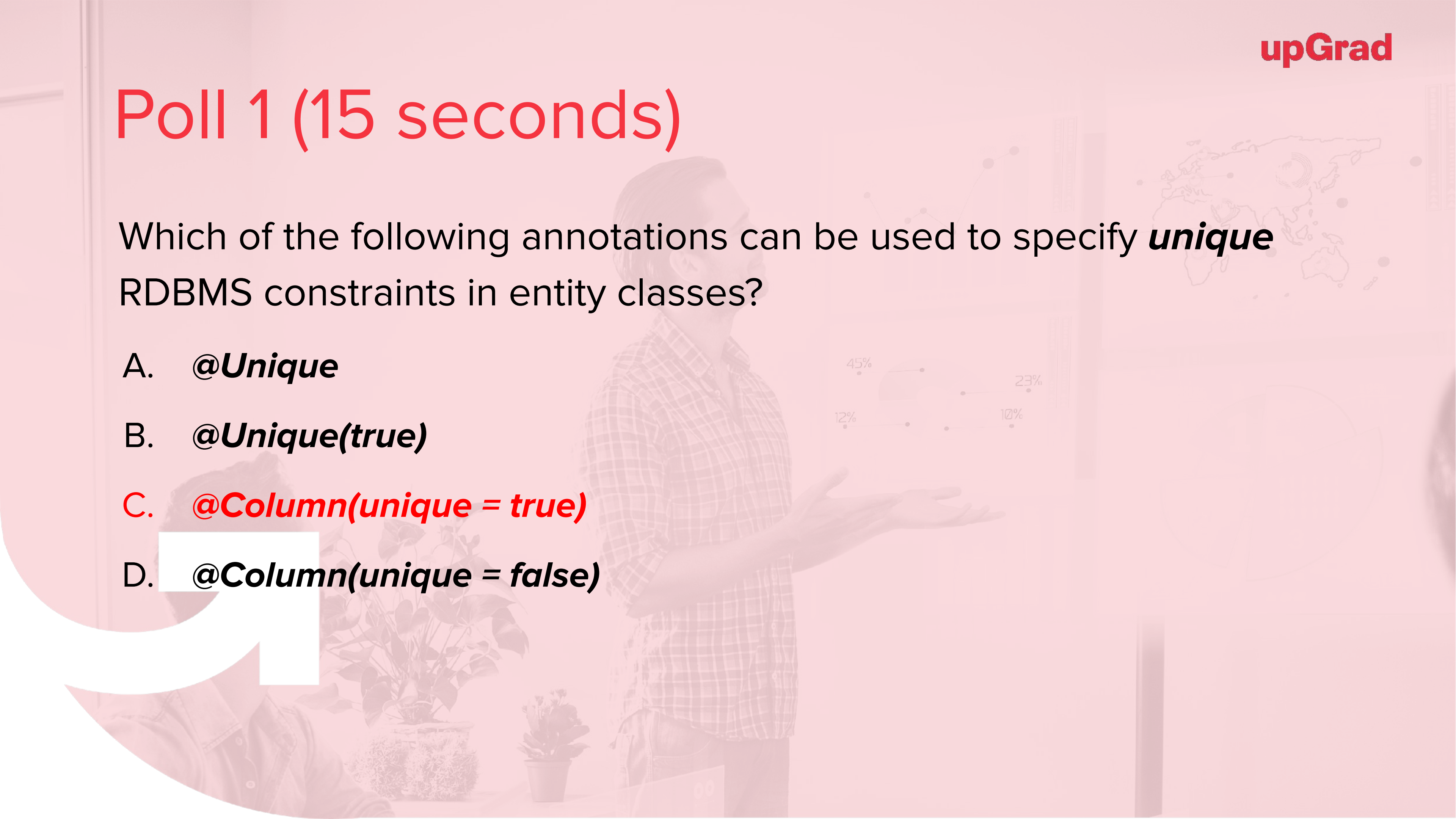
Which of the following annotations can be used to specify ***unique*** RDBMS constraints in entity classes?

A. ***@Unique***

B. ***@Unique(true)***

C. ***@Column(unique = true)***

D. ***@Column(unique = false)***



upGrad

Poll 1 (15 seconds)

Which of the following annotations can be used to specify ***unique*** RDBMS constraints in entity classes?

A. ***@Unique***

B. ***@Unique(true)***

C. ***@Column(unique = true)***

D. ***@Column(unique = false)***

upGrad

Homework Discussion

https://github.com/ishwar-soni/tm-spring-movie-booking-application/tree/session3-homework

upGrad

Today’s Agenda

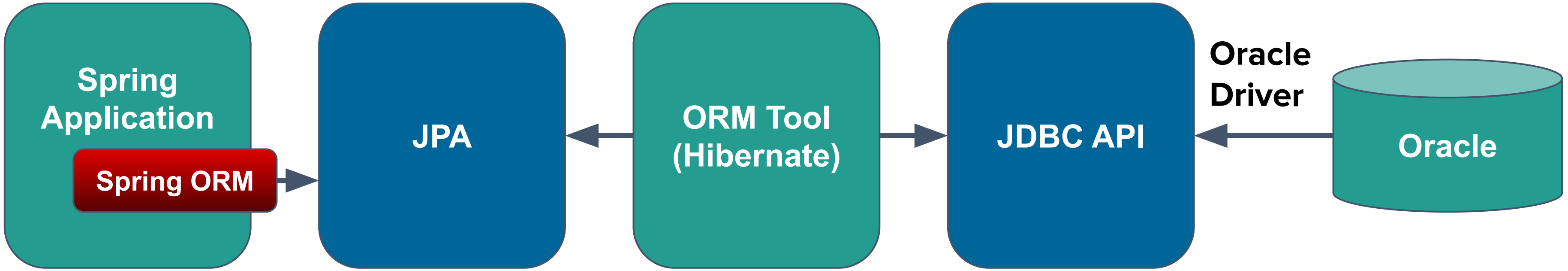
- **Create the Data Access Layer to Save and Retrieve Entities**
  - Creating the Data Access Layer using Spring ORM
  - Using Hibernate SessionFactory
  - Issues with Spring ORM
  - Introduction to Spring Data and Spring Data JPA
  - Introduction to JPA Repository
  - Creating the Data Access Layer using Spring Data JPA

Creating the Data Access Layer Using Spring ORM

Creating the Data Access Layer Using Spring ORM

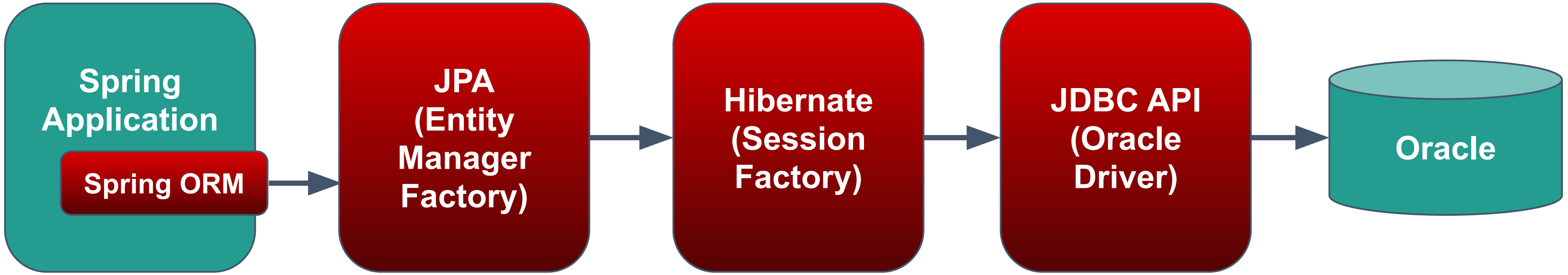
upGrad

- In the previous session, you learnt how Spring ORM helps you connect your Spring application with ORM specifications/tools, such as JPA and Hibernate.



Creating the Data Access Layer Using Spring ORM

- When you run the Spring Boot application with the required dependencies (***spring-boot-starter-data-jpa*** and ***ojdbc***) and database configuration (provided in the ***application.properties*** file), Spring Boot does the autoconfiguration and provides you the following set-up.



Creating the Data Access Layer Using Spring ORM

- Spring Boot will create an ***EntityManagerFactory*** bean, which you can use to get the ***EntityManager*** to exchange the entities with the RDBMS database.
- EntityManagerFactory and EntityManager are parts of the JPA specification, so it's just a wrapper around the Hibernate objects that do the actual work in the background.
- ***SessionFactory*** and ***Session*** are provided by the Hibernate which are used by EntityManagerFactory and EntityManager to save and retrieve the entities to and from the database.

Creating the Data Access Layer Using Spring ORM

upGrad

EntityManager	Session
It is a part of JPA Specification that makes your code loosely coupled with ORM tools.	It is a part of Hibernate that makes your code tightly coupled with Hibernate.
Internally, it uses Session to store and retrieve entities.	Internally, it uses JDBC API to store and retrieve entities.
It provides less control over CRUD operations.	It provide more control over CRUD operations.
It provides CRUD methods such as persist(), find(), merge(), remove(), etc.	It provides CRUD methods such as persist(), save(), saveOrUpdate(), get(), load(), merge(), update(), delete(), etc.

Creating the Data Access Layer Using Spring ORM

upGrad

- In this course, we will be using SessionFactory and Session to save and retrieve entities because they provide more control over CRUD operations.
- After learning how to work with SessionFactory and Session, you will be able to easily work with EntityManagerFactory and EntityManager.
- Let’s create the CustomerDao interface and the CustomerDaoImpl class to understand how to use SessionFactory and Session to save and retrieve entities.

Creating the Data Access Layer Using Spring ORM

- Let’s provide the CustomerDao interface for standard CRUD operations.

```
public interface CustomerDao {
   public Customer save(Customer customer);
   public Customer findById(int id);
   public Customer update(Customer customer);
   public void delete(Customer customer);
}
```

Creating the Data Access Layer Using Spring ORM

- Let’s provide the CustomerDaoImpl class to implement the CustomerDao interface.

```
@Repository
public class CustomerDaoImpl implements CustomerDao {
   ...
}
```

@Repository Annotation

upGrad

- The ***@Repository*** annotation is a special form of the ***@Component*** annotation (Recall @Component annotation).
- Similar to the @Component annotation, any class marked with the @Repository annotation will be instantiated as a Spring bean by Spring.
- A special feature of this annotation is that it converts checked JDBC exceptions into unchecked Spring exceptions.

SessionFactory

upGrad

```
@Repository
public class CustomerDaoImpl implements CustomerDao {

  private SessionFactory sessionFactory;

  @Autowired
  public CustomerDaoImpl(EntityManagerFactory entityManagerFactory) {
      this.sessionFactory = entityManagerFactory.unwrap(SessionFactory.class);
  }

  ...
}
```

As Spring Boot only creates the bean of EntityManagerFactory, which is a wrapper around SessionFactory, we need to unwrap it to get SessionFactory out of it.

session.save()

upGrad

```
@Repository
public class CustomerDaoImpl implements CustomerDao {
    ...
    @Override
    public Customer save(Customer customer) {
        Session session = this.sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();

        session.save(customer);

        transaction.commit();
        session.close();

        return customer;
    }
    ...
}
```

Open a new session and start a database transaction.

Save the entity into the database.

Commit the transaction and close the session.

session.save()

- In the previous code snippet, we used the save() method of the session to save the entity into the database.
- However, Session provides one more method, persist(), to do the same job.
- There is a subtle difference between these two methods. To understand this different, we need to first understand different states of the entities during CRUD operations.

Entity States

- Entities (or entity objects) can be in one the following three states:
  - ***Transient***: Newly created entity objects that do not have corresponding rows in the database table
  - ***Persistent***: Entity objects that have corresponding rows in the database table once the transaction is committed
  - ***Detached***: Entity objects that have corresponding rows in the database table but are not in the persistent state because the session is probably closed

session.save()

```
@Repository
public class CustomerDaoImpl implements CustomerDao {
    ...
    @Override
    public Customer save(Customer customer) {
        Session session = this.sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();

        session.save(customer);


        transaction.commit();
        session.close();

        return customer;
    }
    ...
}
```

Transient state
No corresponding row exists in the database.

Persistent state
Corresponding rows exist in the database table when transaction is committed.

Detached state
The session is now closed.

save() vs persist()

save()	persist()
It returns the generated ID.	It returns the void.
It also works outside transaction boundaries.	It only works inside transaction boundaries.
It saves detached entities as new rows.	It throws exceptions when we try to save detached entities.
It can only be used with session objects.	It is internally called by the JPA EntityManager.

session.get() and session.update()

upGrad

```java
@Repository
public class CustomerDaoImpl implements CustomerDao {
   ...
   @Override
   public Customer findById(int id) {
       Session session = this.sessionFactory.openSession();
       Transaction transaction = session.beginTransaction();
       Customer customer = session.get(Customer.class, id);
       transaction.commit();
       session.close();
       return customer;
   }

   @Override
   public Customer update(Customer customer) {
       Session session = this.sessionFactory.openSession();
       Transaction transaction = session.beginTransaction();
       session.update(customer);
       transaction.commit();
       session.close();
       return customer;
   }
   ...
}
```

get() vs load()

upGrad

- Hibernate provides two methods to retrieve entities from the database, get() and load().

get()	load()
If an entity is not found for the given ID, then the method returns null.	If an entity is not found for the given ID, then the method throws an exception.
It is slower than the load() method, as it fully initialises objects, including foreign key mapped objects.	It is faster than the get() method, as it does not fully initialise objects.

session.delete()

```
@Repository
public class CustomerDaoImpl implements CustomerDao {
   ...
   @Override
   public void delete(Customer customer) {
       Session session = this.sessionFactory.openSession();
       Transaction transaction = session.beginTransaction();

       Customer mergedCustomer = (Customer)session.merge(customer);
       session.delete(mergedCustomer);

       transaction.commit();
       session.close();
   }
}
```

Code Reference

session.delete()

- Using the session.delete() method, we can only delete entities that are in the persistent state.
- If we want to delete entities that are in the detached state, then we will need to first move them back to the persistent state using the merge() method.
- We cannot delete entities that are in the transient state because they do not have any corresponding rows that can be deleted from the database.

Poll 2 (15 seconds)

Fill in the blank.

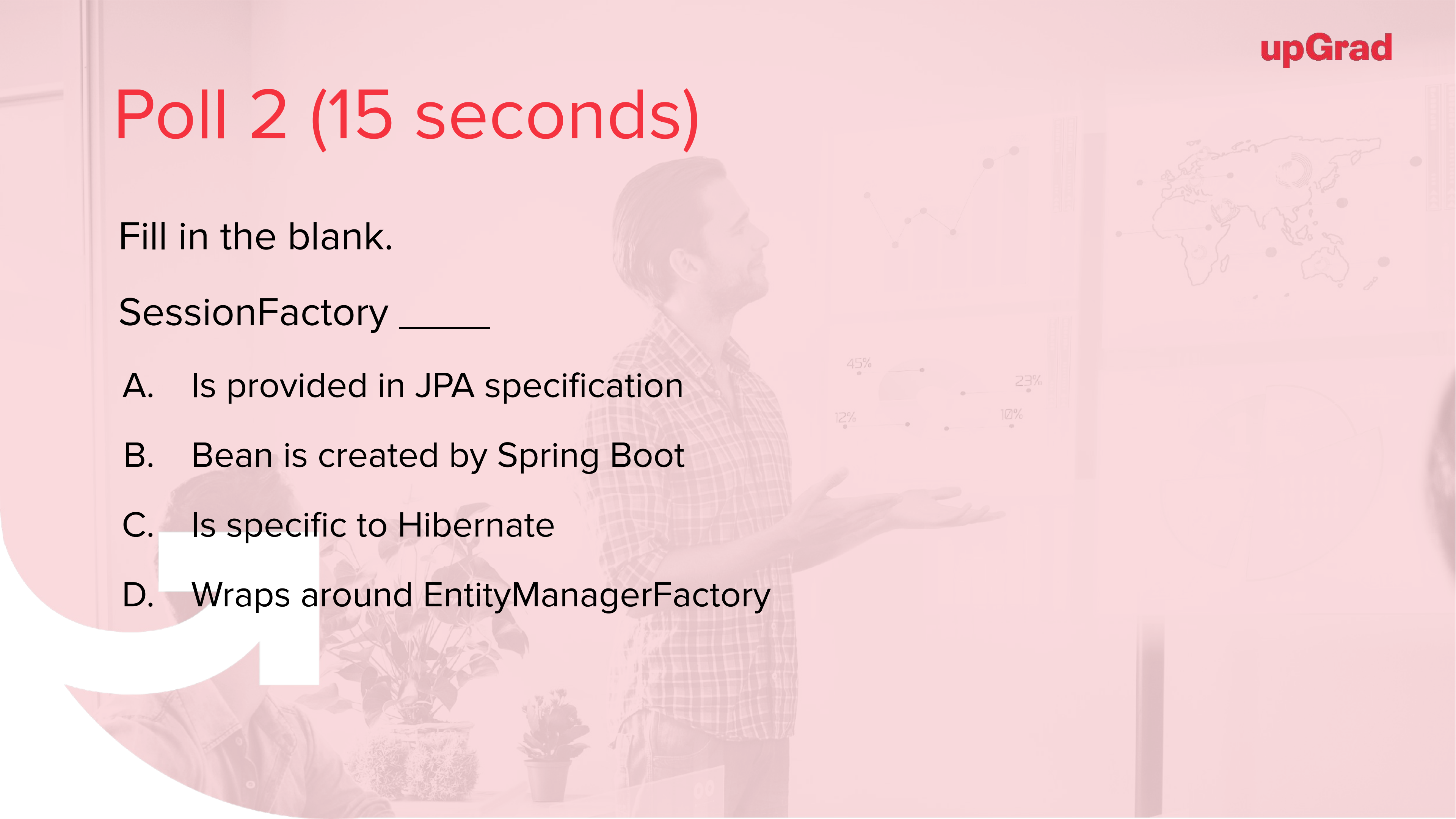
SessionFactory _____

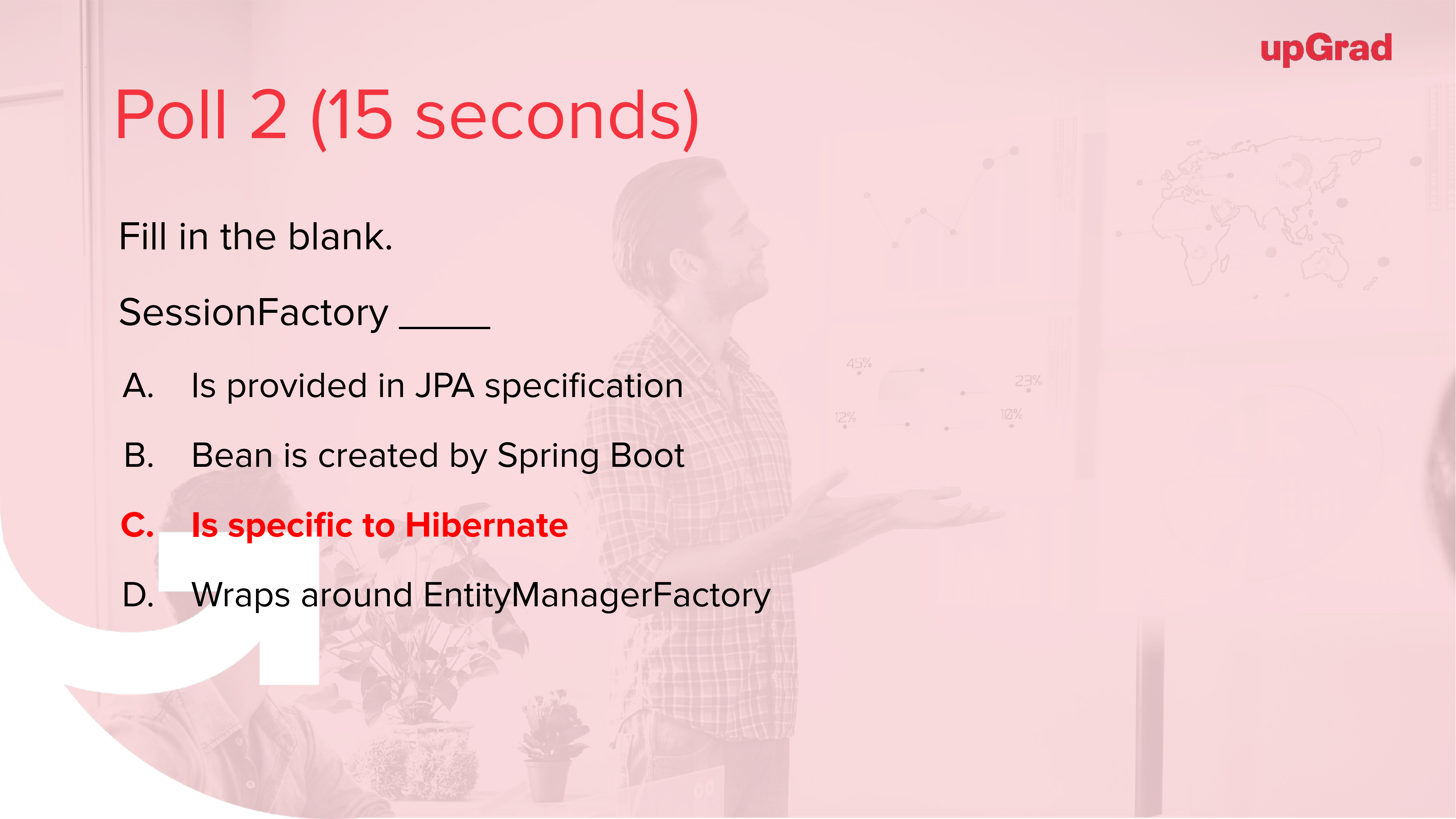
A. Is provided in JPA specification

B. Bean is created by Spring Boot

C. Is specific to Hibernate

D. Wraps around EntityManagerFactory





Poll 2 (15 seconds)

Fill in the blank.

SessionFactory _____

A. Is provided in JPA specification

B. Bean is created by Spring Boot

C. Is specific to Hibernate

D. Wraps around EntityManagerFactory

upGrad

Poll 3 (15 seconds)

Which of the following statements regarding the @Repository annotation are true? (Note: More than one option may be correct.)

A. It is used to mark entity classes.

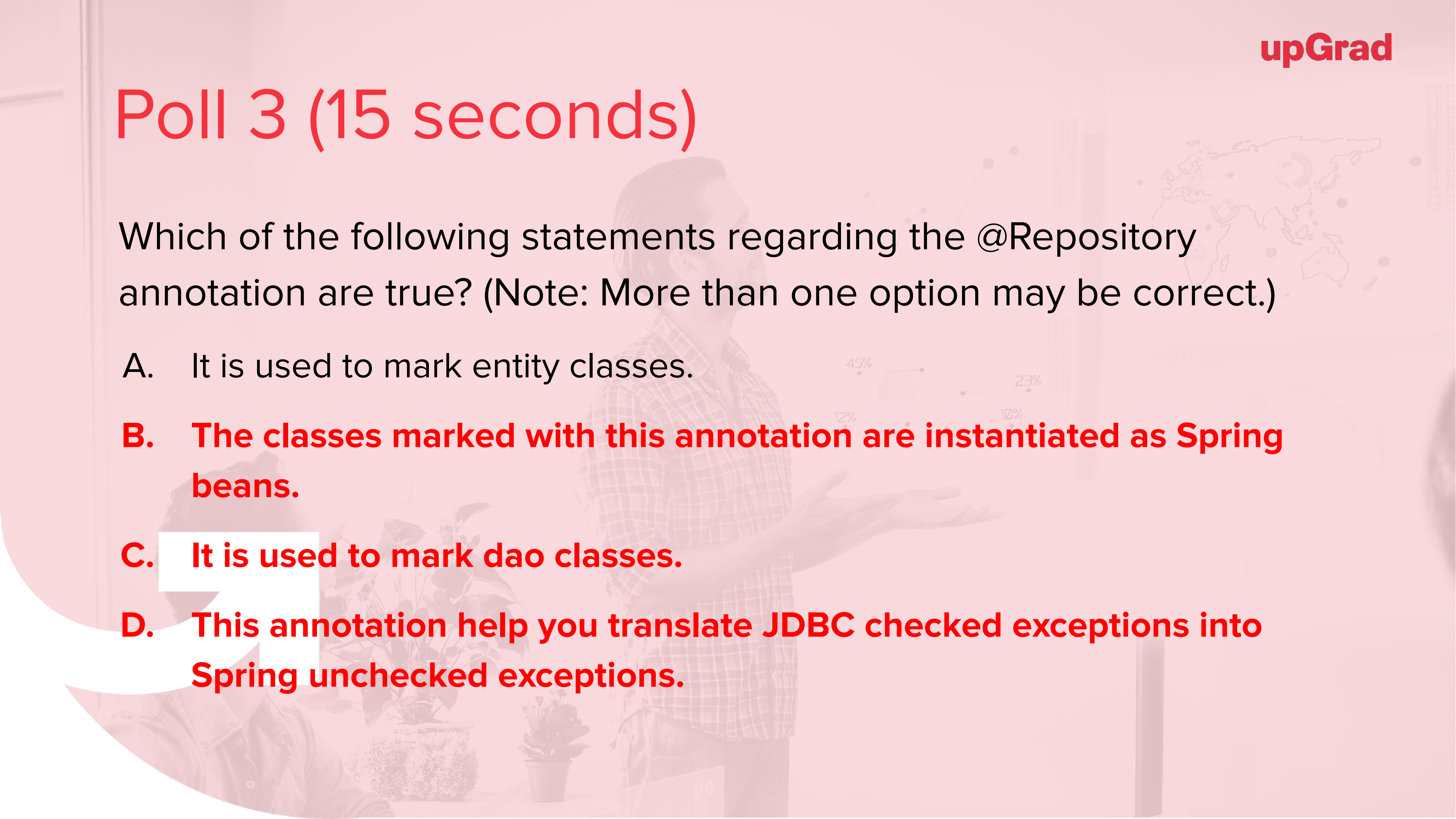
B. The classes marked with this annotation are instantiated as Spring beans.

C. It is used to mark dao classes.

D. This annotation help you translate JDBC checked exceptions into Spring unchecked exceptions.

upGrad

Poll 3 (15 seconds)



Which of the following statements regarding the @Repository annotation are true? (Note: More than one option may be correct.)

A. It is used to mark entity classes.

B. The classes marked with this annotation are instantiated as Spring beans.

C. It is used to mark dao classes.

D. This annotation help you translate JDBC checked exceptions into Spring unchecked exceptions.

upGrad

Poll 4 (15 seconds)

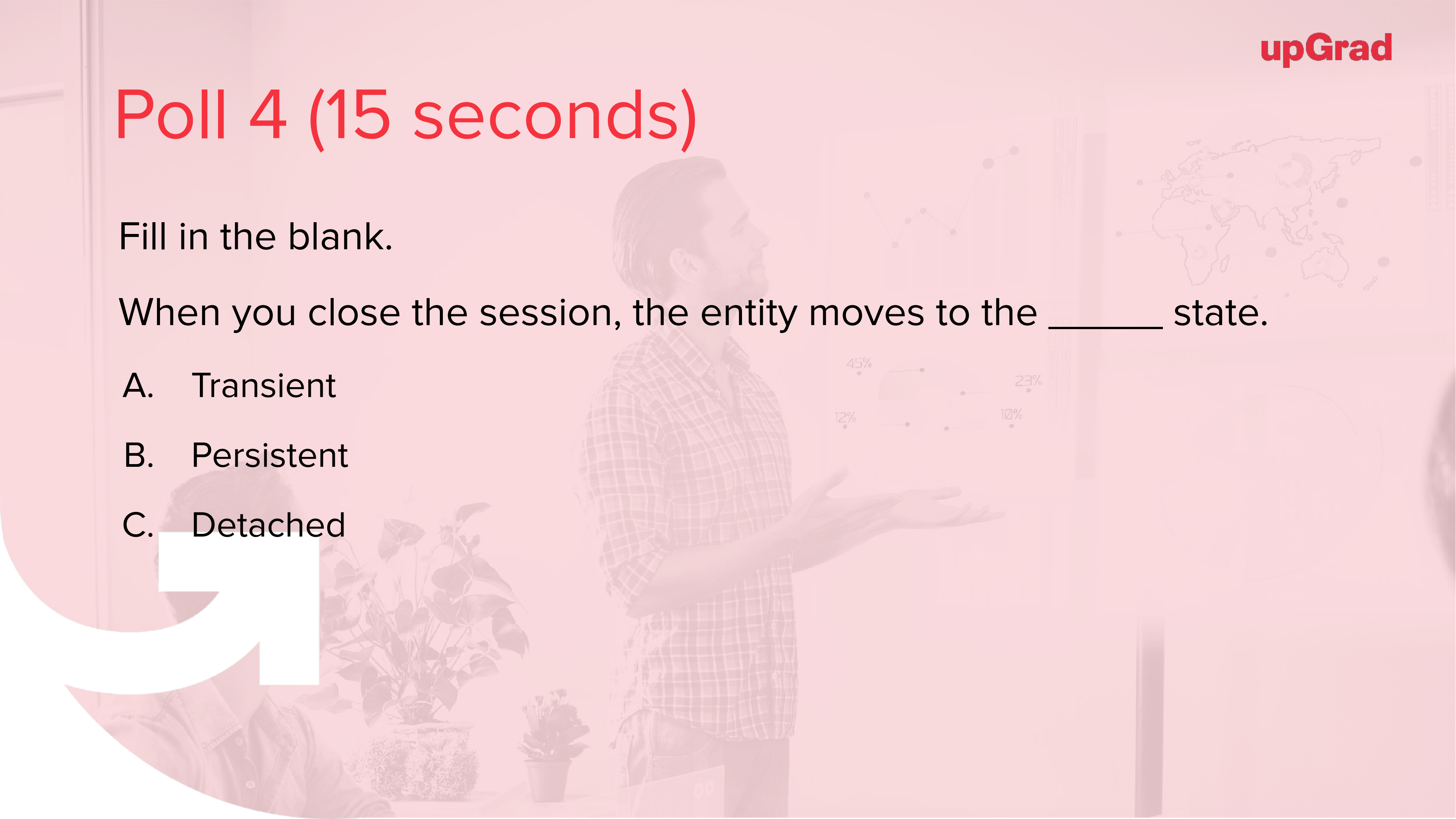
Fill in the blank.

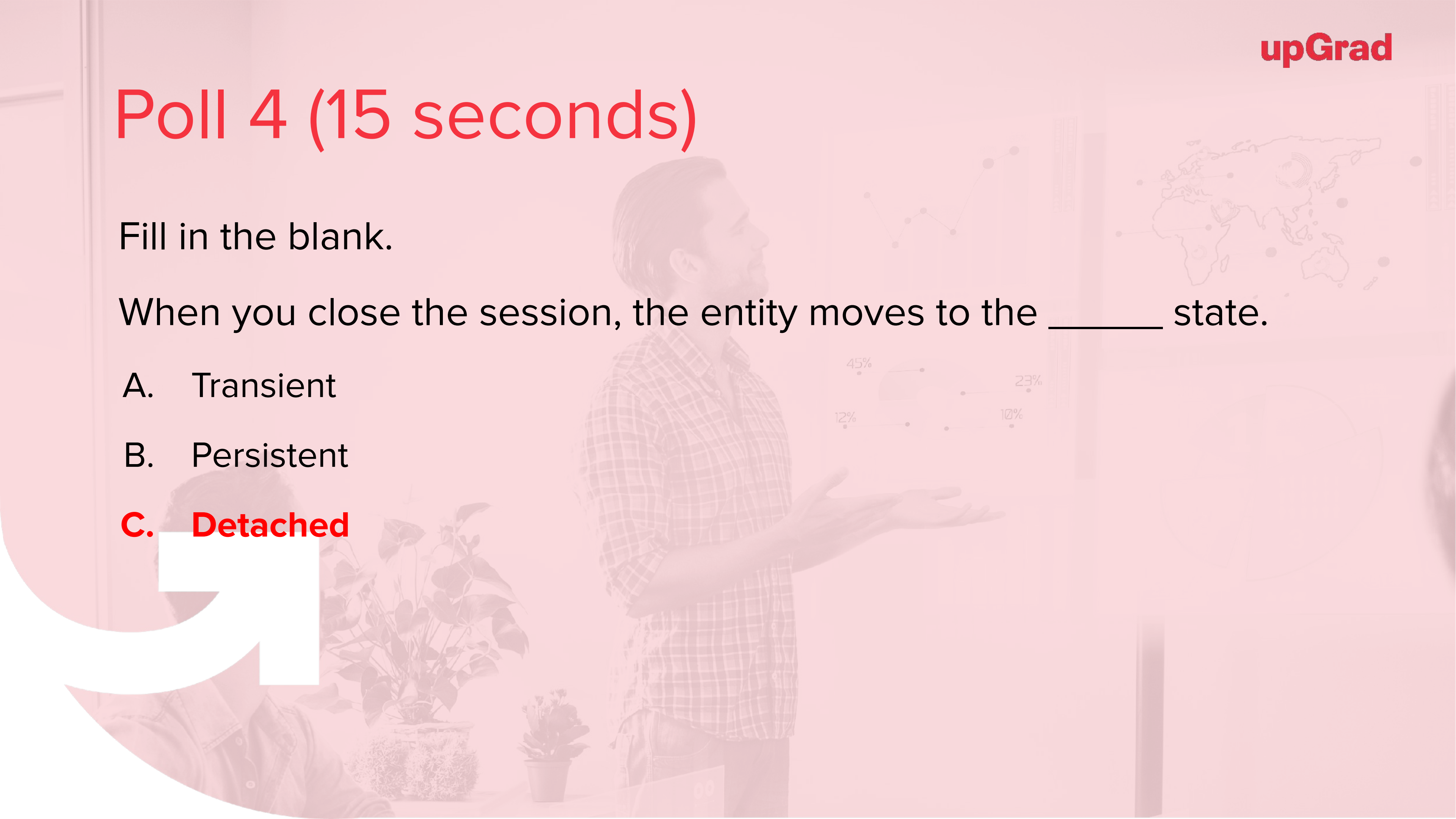
When you close the session, the entity moves to the ______ state.

A. Transient

B. Persistent

C. Detached





upGrad

Poll 4 (15 seconds)

Fill in the blank.

When you close the session, the entity moves to the ______ state.

A. Transient

B. Persistent

[✓ C. **Detached**]

upGrad

Poll 5 (15 seconds)

Which of the following statements regarding the save() method of the session class is true?

A. It returns the void.

B. It only works inside transaction boundaries.

C. It saves detached entities as new rows.

D. It is internally called by the JPA EntityManager.

Poll 5 (15 seconds)

Which of the following statements regarding the save() method of the session class is true?

A. It returns the void.

B. It only works inside transaction boundaries.

C. It saves detached entities as new rows.

D. It is internally called by the JPA EntityManager.

Hands-On Experience

upGrad

Let's create another dao class for the City entity.

TODO

- Use the following command to checkout to the current state:

  ```
  git checkout d56843f
  ```

- Create the CityDao interface with basic CRUD operations.
- Provide the CityDaoImpl class to implement the CityDao interface. This class has to be marked with the @Repository annotation.
- Use CityDaoImpl to perform CRUD operations on the City entity in the main() method.

Code Reference

Spring Data and Spring Data JPA

Issues with Spring ORM

- So far, you have learnt how to perform basic CRUD operations using EntityManager or Session without writing SQL queries. However, there are certain issues that may face while working with EntityManager and Session, which are as follows:
  - You need to write boilerplate code, such as opening and closing sessions, and starting and committing transactions.
  - These classes only provide methods to perform basic CRUD operations; you will need to write JPQL or HQL queries for other operations.
  - You need to manage entity states manually; otherwise, exceptions are thrown or can lead to duplication of data.

Introduction to Spring Data

- Spring Data helps you deal with the issues discussed previously
- Spring Data is a Spring Project, which provides a common interface to access data from both RDBMS databases and NoSQL databases.
- It not only provides a common interface but also makes data access so easy that you can create the Data Access Layer using only dao interfaces.
- With Spring Data, you simply need to create DAO interfaces. You need not provide any implementations for these interfaces; Spring Data will do that for you.
- Spring Data can provide several subprojects based on your requirements. In this course, we will be using Spring Data JPA to create the Data Access Layer.

Introduction to Spring Data JPA

- While using Spring Data JPA, we can create the entire Data Access Layer using only dao interfaces.
- The only requirement is that the dao interface should extend the ***JpaRepository*** interface. With this, you will have methods for all the basic CRUD operations.
- Let’s provide Dao interface for the Movie entity and see Spring Data JPA in action.

JpaRepository Interface

```
public interface MovieDao extends JpaRepository<Movie, Integer> {

}
```

- By doing this, you will have all the basic CRUD operations for the Movie entity.
- The JpaRepository interface is of the generic type, which accepts two types of parameters.
    - The first parameter specifies the type of Entity for which we are providing the DAO interface.
    - The second parameter specifies the type of the primary key for that entity.

JpaRepository Interface

upGrad

- The JpaRepository interface provides the following methods:

C	R	U	D

save(entity)
saveAll(entities)
	findById(id)
	findAll()
	findAllById(ids)
	findAll(pageable)
		save(entity)
		saveAll(entities)
			delete(entity)
			deleteById(id)
			deleteAll()
			deleteAll(entities)

JpaRepository Interface

upGrad

- Let’s use the MovieDao interface to perform some basic CRUD operations for the Movie entity.

Code Reference

Pagination Using JpaRepository

- The JpaRepository interface provides a method that returns data in the paginated form which is as follows:

```
Page<T> findAll(Pageable pageable)
```

- The argument is a pageable object, which can be created as shown below.

```
PageRequest.of(0, 2)

PageRequest.of(0, 2, Sort.by("duration").ascending())
```

- The first argument specifies the page number, and the second argument specifies the page size. The third argument is optional; it specifies how the sorting needs to be done, by column or by attribute. It also specifies the sorting direction.

Pagination Using JpaRepository

- Let’s check how to make paginated requests using JpaRepository.

Code Reference

Poll 6 (15 seconds)

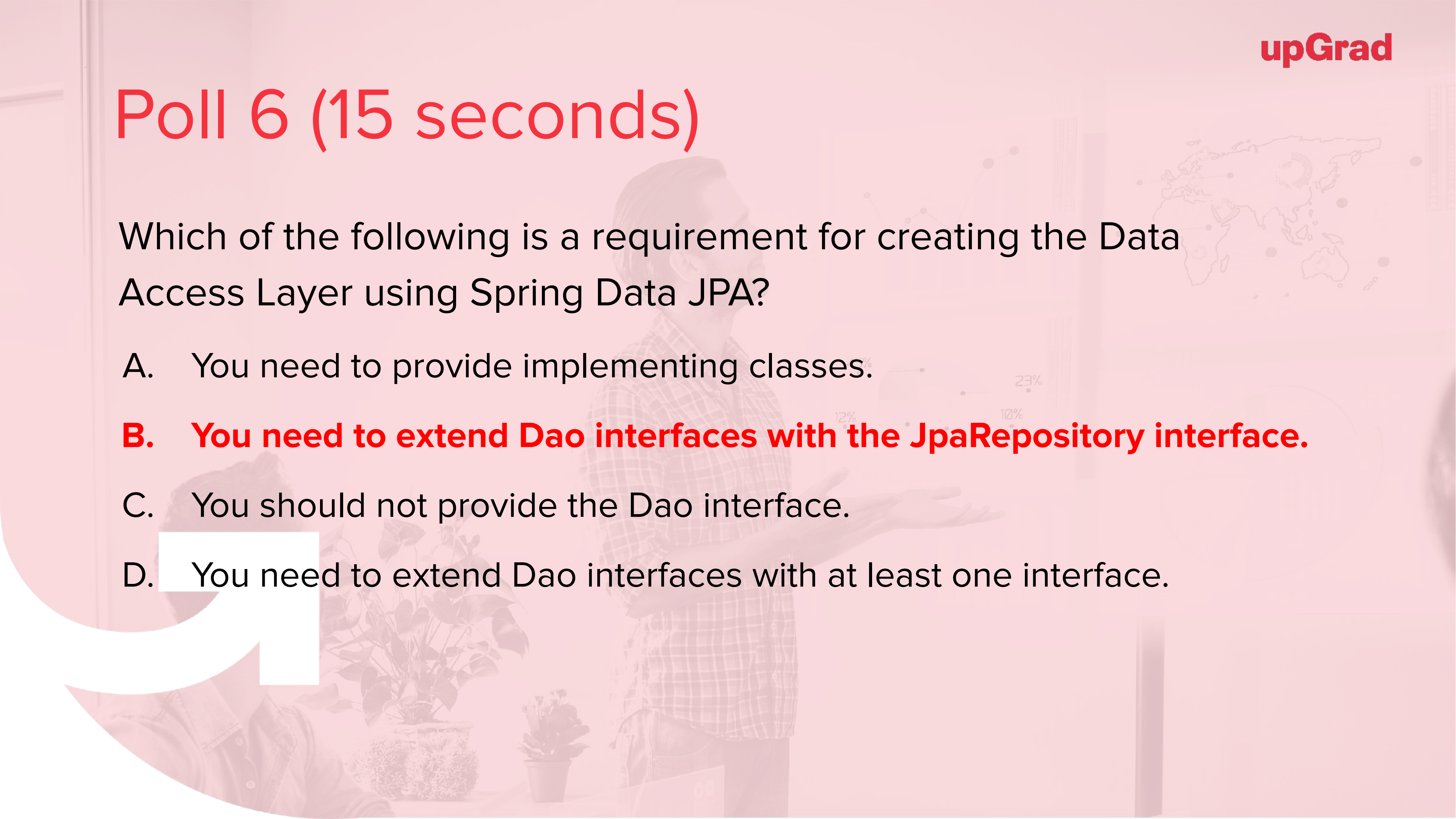
Which of the following is a requirement for creating the Data Access Layer using Spring Data JPA?

A. You need to provide implementing classes.

B. You need to extend Dao interfaces with the JpaRepository interface.

C. You should not provide the Dao interface.

D. You need to extend Dao interfaces with at least one interface.



upGrad

Poll 6 (15 seconds)

Which of the following is a requirement for creating the Data Access Layer using Spring Data JPA?

A. You need to provide implementing classes.

B. You need to extend Dao interfaces with the JpaRepository interface.

C. You should not provide the Dao interface.

D. You need to extend Dao interfaces with at least one interface.

upGrad

Poll 6 (15 seconds)

Which of the following is the correct way of creating a PageRequest? (Note: More than one option may be correct.)

A. PageRequest.of(pageNumber, pageSize)

B. PageRequest.of(pageSize, pageNumber)

C. PageRequest.of(pageNumber, pageSize, SortingCriteria)

D. PageRequest.of(pageSize, pageNumber, SortingCriteria)

Poll 6 (15 seconds)

Which of the following is the correct way of creating a PageRequest? (Note: More than one option may be correct.)

A. PageRequest.of(pageNumber, pageSize)

B. PageRequest.of(pageSize, pageNumber)

C. PageRequest.of(pageNumber, pageSize, SortingCriteria)

D. PageRequest.of(pageSize, pageNumber, SortingCriteria)

Hands-On Experience

Let’s create another dao interface using Spring Data JPA for the Theatre entity.

TODO

- Use the following command to checkout to the current state:

  ```
  git checkout 93a91c3
  ```

- Create the TheatreDao interface and extend it with the JpaRepository interface.
- Use the TheatreDao interface to perform CRUD operations on the Theatre entity in the main() method.
- Add at least five theatres into the database and use the TheatreDao interface to make paginated requests.

Code Reference

Custom Query Methods Using Spring Data JPA

Custom Query Methods Using Spring Data JPA

upGrad

- Even though Spring Data JPA provides query methods for basic CRUD operations, they are not sufficient for real-world applications, especially the retrieve methods.
- For example, you can only search for a movie using its ID. You cannot use any other field, such as movie name, because JpaRepository only provides the findById() method.
- If you want to search movies based on some other criteria, you will need to provide custom query methods.
- While providing custom query methods, you need to follow certain naming conventions so Spring Data JPA can provide implementations for those methods.

Naming Conventions for Custom Query Methods

upGrad

Query Method	**Example**	**Description**
Movie findById(int id)	-	Find a movie based on its ID
Movie findBy***Field*** (Type field)	Movie findByMovieName (String movieName)	Find a movie based on the movie name. This will fail if more than one movie exist with the same name.
List<Movie> findBy***Field*** (Type field)	List<Movie> findByMovieName (String movieName)	Find all the movies based on the movie name. This is safe when the column does not have a unique constraint.

Naming Conventions for Custom Query Methods

upGrad

Query Method	Example	Description
Movie findBy***Field1AndField2*** (Type1 field1, Type2 field2)	Movie findByMovieNameAndDuration (String movieName, int duration)	Find a movie based on the movie name and its duration. This will fail if more than one movie with the same provided criteria exists. Consider changing return type to List<Movie> for safety.
Movie findBy***Field1OrField2*** (Type1 field1, Type2 field2)	Movie findByMovieNameOrDuration (String movieName, int duration)	Find a movie based on the movie name or its duration. This will fail if more than one movie with the same provided criteria exists. Consider changing return type to List<Movie> for safety.

Custom Query Methods Using Spring Data JPA

upGrad

- You can find other method naming convention here.
- Let’s provide some of the custom query methods for the MovieDao interface.

Code Reference

upGrad

Poll 7 (15 seconds)

Suppose you have the Employee entity with one Age field. Which of the following custom query methods will you use to find all the employees who are younger than the specified age?

A. List<Employee> findEmployeesYoungerThan(int age)

B. List<Employee> findByAgeLessThan(int age)

C. Employee findByAgeLessThan(int age)

D. List<Employee> findAgeLessThan(int age)

upGrad

Poll 7 (15 seconds)

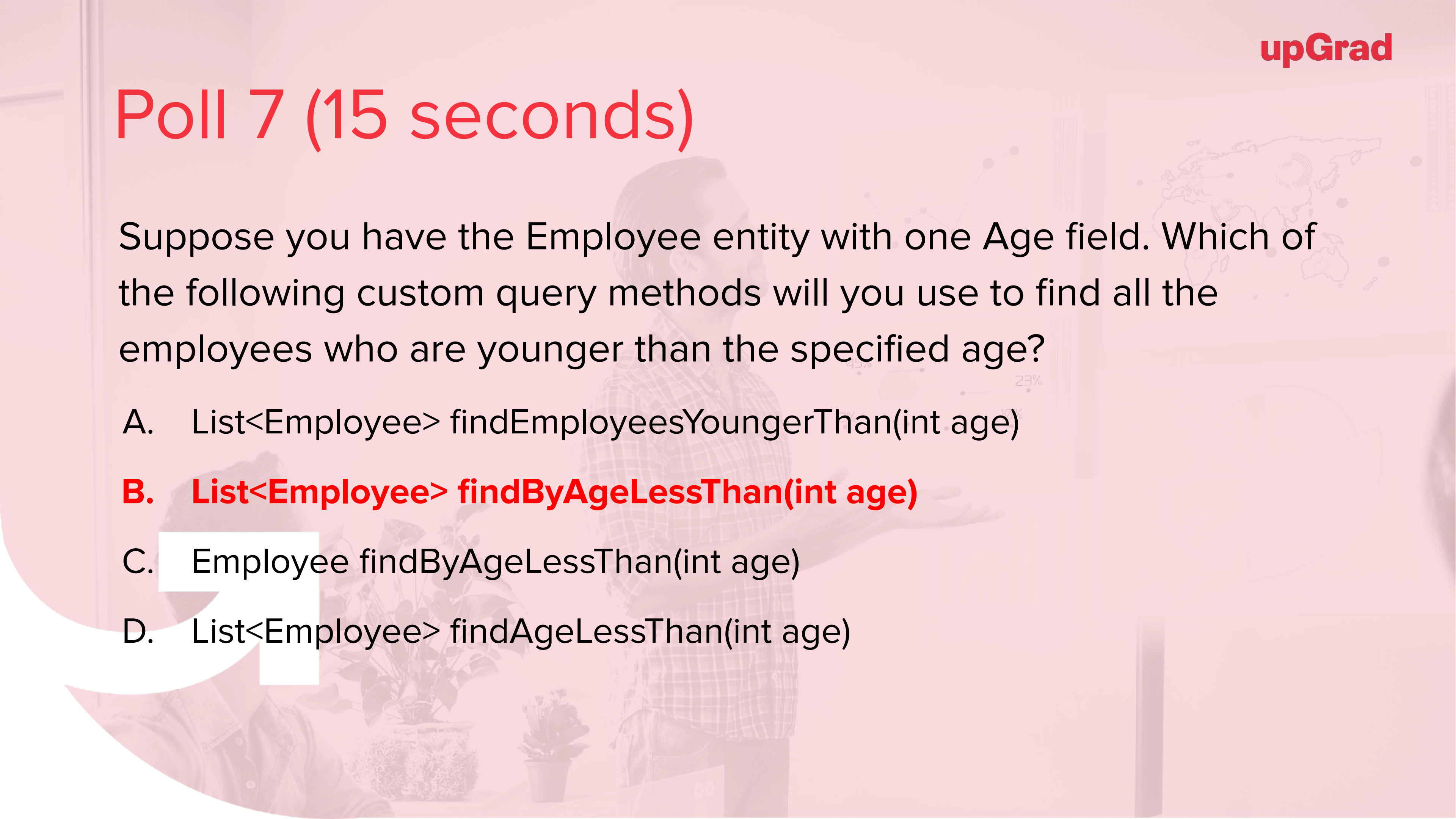
Suppose you have the Employee entity with one Age field. Which of the following custom query methods will you use to find all the employees who are younger than the specified age?

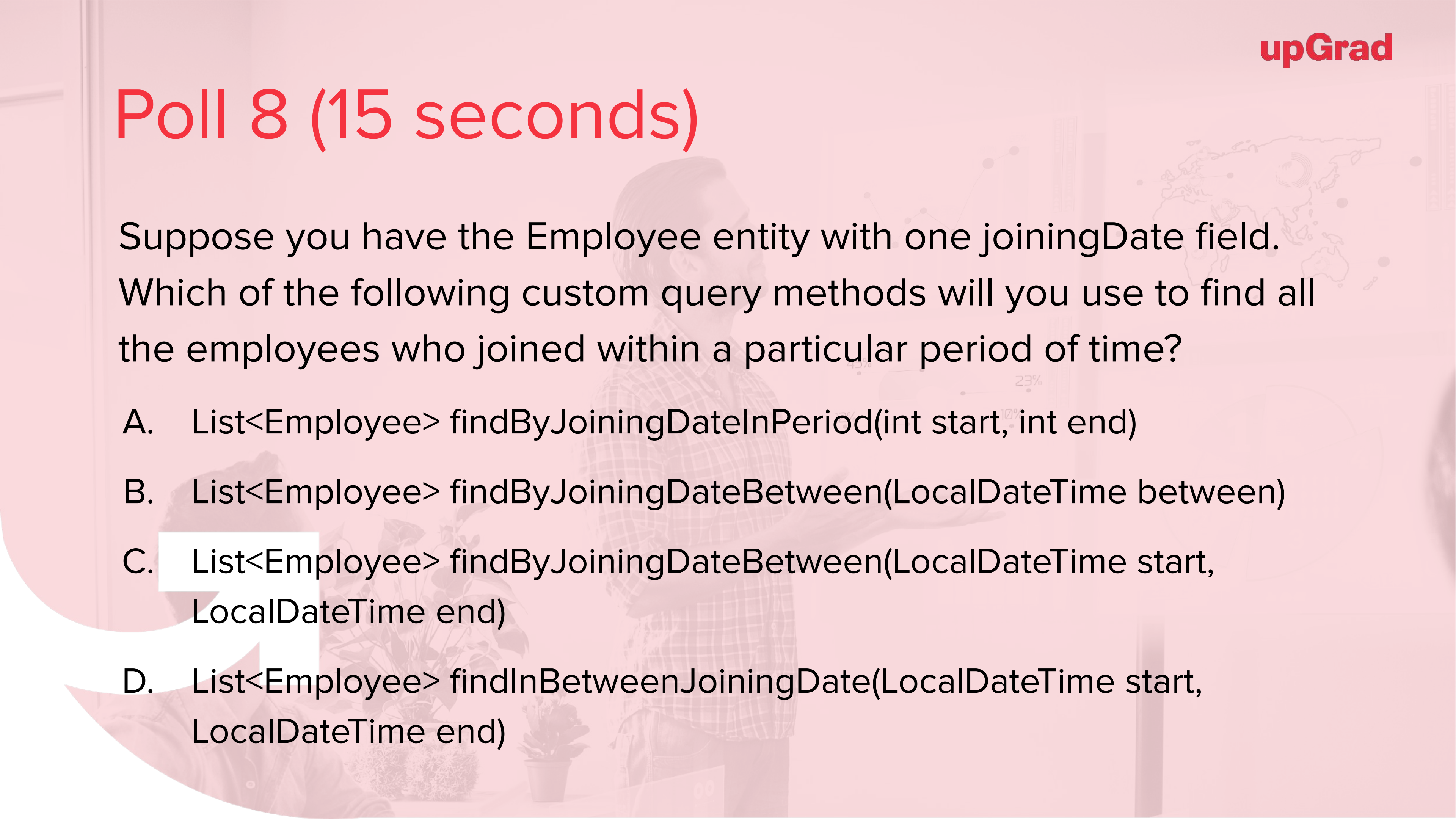
A. List<Employee> findEmployeesYoungerThan(int age)

B. List<Employee> findByAgeLessThan(int age)

C. Employee findByAgeLessThan(int age)

D. List<Employee> findAgeLessThan(int age)





Poll 8 (15 seconds)

Suppose you have the Employee entity with one joiningDate field. Which of the following custom query methods will you use to find all the employees who joined within a particular period of time?

A. List<Employee> findByJoiningDateInPeriod(int start, int end)

B. List<Employee> findByJoiningDateBetween(LocalDateTime between)

C. List<Employee> findByJoiningDateBetween(LocalDateTime start, LocalDateTime end)

D. List<Employee> findInBetweenJoiningDate(LocalDateTime start, LocalDateTime end)

upGrad

Poll 8 (15 seconds)

Suppose you have the Employee entity with one joiningDate field. Which of the following custom query methods will you use to find all the employees who joined within a particular period of time?

A. List<Employee> findByJoiningDateInPeriod(int start, int end)

B. List<Employee> findByJoiningDateBetween(LocalDateTime between)

C. List<Employee> findByJoiningDateBetween(LocalDateTime start, LocalDateTime end)

D. List<Employee> findInBetweenJoiningDate(LocalDateTime start, LocalDateTime end)

All the code used in today's session can be found in the link provided below (branch session4-demo):

https://github.com/ishwar-soni/tm-spring-movie-booking-application/tree/session4-demo

Important Concepts and Questions

Important Questions

upGrad

1. What is the difference between the load() method and the get() method of Hibernate Session?
2. How can one dao Implementation object (singleton) handle multiple requests?
3. What is the difference between the save() method and the persist() method of Hibernate Session?
4. Explain Spring Data and Spring Date JPA.
5. Explain some of the methods provided by the Spring Data JPA.
6. Explain some of the naming conventions to be followed while writing custom query methods.

upGrad

Doubt Clearance Window

Today, we learnt the following:

1. We can create the Data Access Layer using Spring ORM and JPA EntityManager or Hibernate Session without writing any SQL or JPQL query.
2. Creating the Data Access Layer using Hibernate Session or JPA EntityManager has certain limitations; you need to write the boilerplate code, manage the state of the entities and write JPQL queries for specific CRUD operations.
3. You can create the Data Access Layer using only the Dao interface with the help of Spring Data JPA.
4. Spring Data JPA provides some basic CRUD operations.
5. You can also provide more specific CRUD operations using custom query methods.

Homework

upGrad

In today’s session, you learnt how to create the Data Access Layer using Spring ORM and Spring Data JPA. We have already provided Dao interfaces for certain entities. You should try to provide Dao interfaces for the remaining entities using the Spring Data JPA.

TODOs

1. Use the following command to checkout to the current state:
   git checkout e948bab
2. Provide Dao interfaces for the following entities: (Note that you need not provide any custom query methods for these Dao interfaces.)
   a. Booking
   b. Language
   c. Status
   d. UserType

Homework

2. Provide the following custom query methods for the TheatreDao interface:
   a. findByTheatreName
   b. findByTicketPriceLessThan
   c. findByTheatreNameContaining
3. Use the TheatreDao interface to perform CRUD operations using the custom query methods in the main method.

Code Reference

upGrad

Tasks to Complete After Today’s Session

MCQs
Coding Questions
Homework
Project Checkpoint 3

Next Steps

Checkout this code (**session4-homework** branch) before proceeding to the next session.

upGrad
#RahoAmbitious

Thank You!