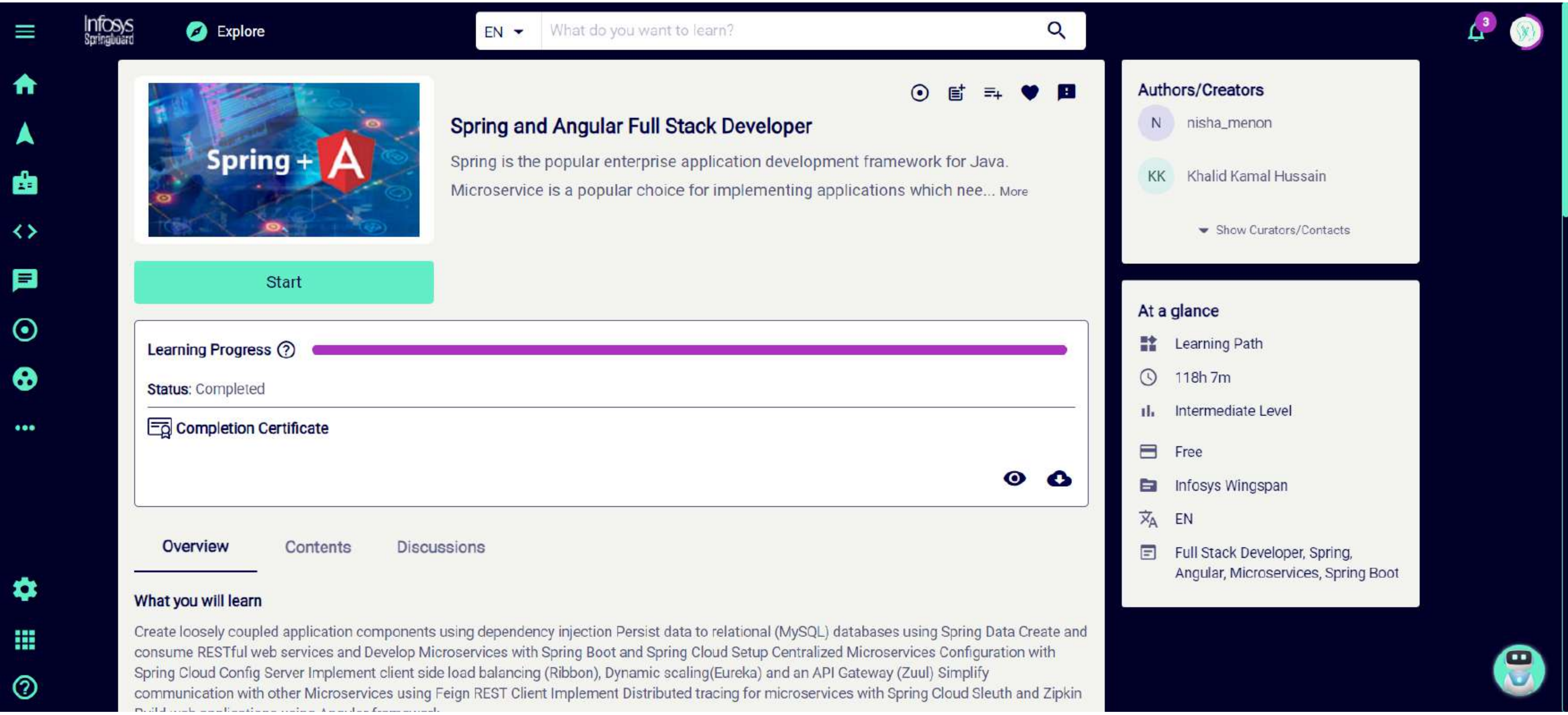


Infosys Springboard

Explore

EN | What do you want to learn?

3

Start

## Spring and Angular Full Stack Developer

Spring is the popular enterprise application development framework for Java. Microservice is a popular choice for implementing applications which nee... More

Learning Progress

Status: Completed

Completion Certificate

Overview | Contents | Discussions

### What you will learn

Create loosely coupled application components using dependency injection Persist data to relational (MySQL) databases using Spring Data Create and consume RESTful web services and Develop Microservices with Spring Boot and Spring Cloud Setup Centralized Microservices Configuration with Spring Cloud Config Server Implement client side load balancing (Ribbon), Dynamic scaling(Eureka) and an API Gateway (Zuul) Simplify communication with other Microservices using Feign REST Client Implement Distributed tracing for microservices with Spring Cloud Sleuth and Zipkin

### Authors/Creators

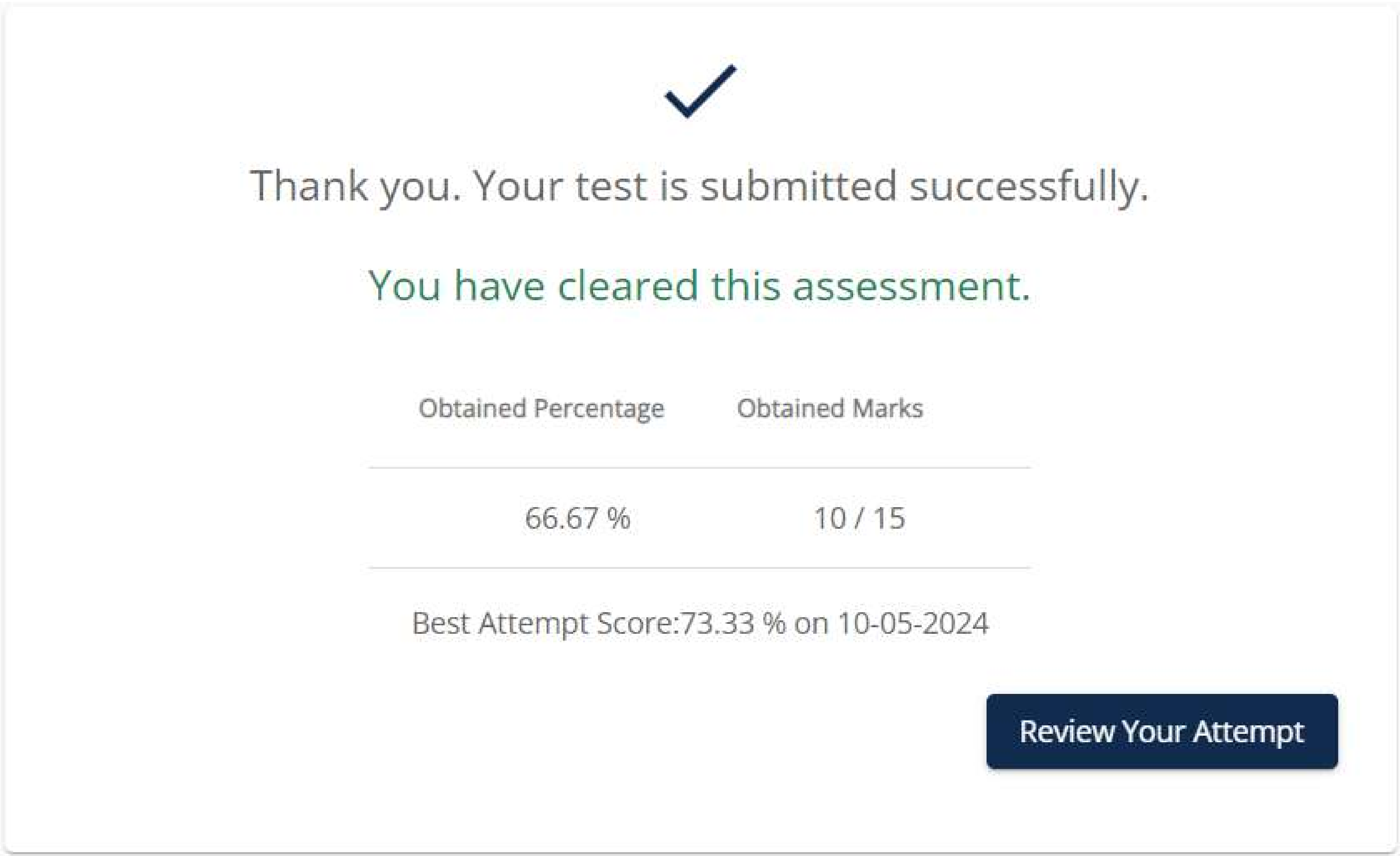
N nisha_menon

KK Khalid Kamal Hussain

Show Curators/Contacts

### At a glance

- Learning Path
- 118h 7m
- Intermediate Level
- Free
- Infosys Wingspan
- EN
- Full Stack Developer, Spring, Angular, Microservices, Spring Boot



Thank you. Your test is submitted successfully.

You have cleared this assessment.

| Obtained Percentage | Obtained Marks |
|---|---|
| 66.67 % | 10 / 15 |

Best Attempt Score:73.33 % on 10-05-2024

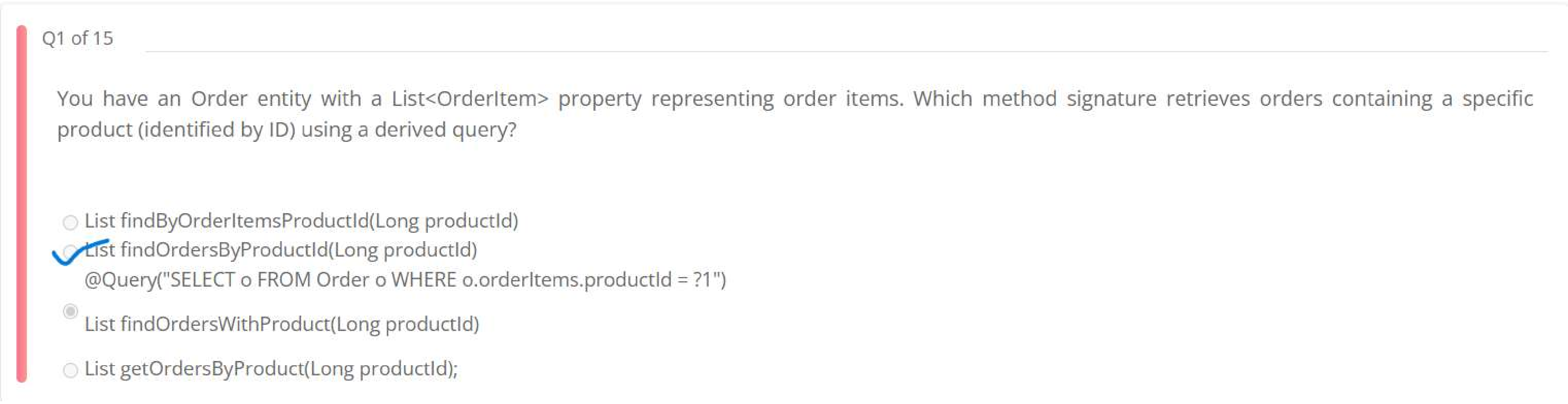
Result: CLEARED

Overall Percentage: 66.67% (10/15)

Spring Data Essential - Assessment

Section Score: 10/15

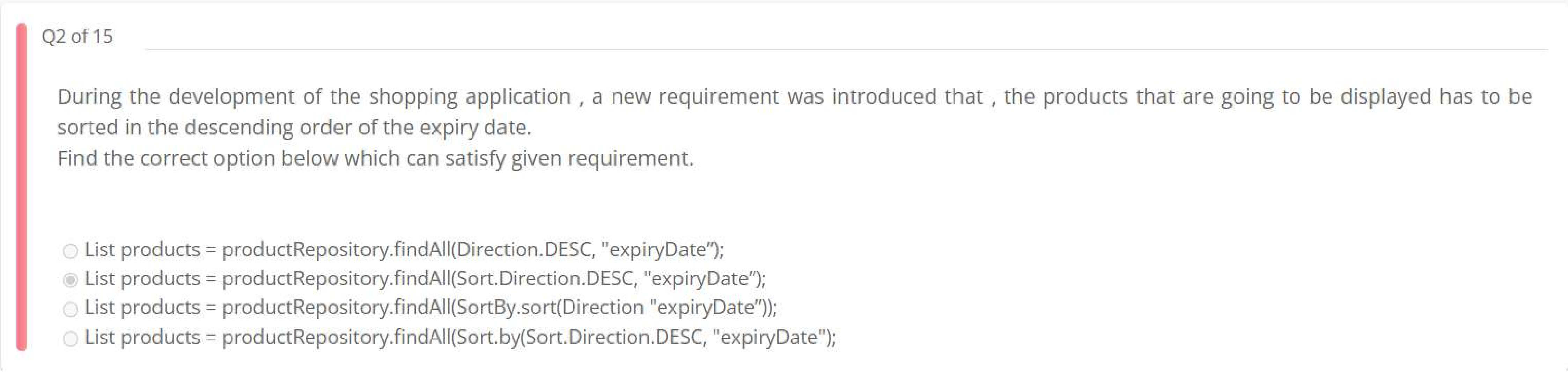
🟢 Correct | 🔴 Incorrect | ⚪ Unattempted



Q1 of 15

You have an Order entity with a List<OrderItem> property representing order items. Which method signature retrieves orders containing a specific product (identified by ID) using a derived query?

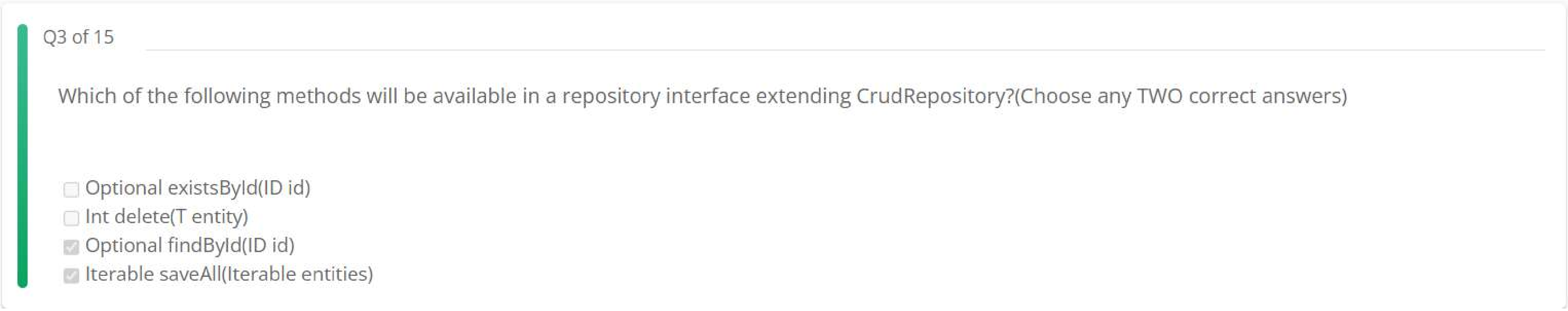
- ○ List findByOrderItemsProductId(Long productId)
- [✓ List findOrdersByProductId(Long productId)
  @Query("SELECT o FROM Order o WHERE o.orderItems.productId = ?1")]
- ● List findOrdersWithProduct(Long productId)
- ○ List getOrdersByProduct(Long productId);



Q2 of 15

During the development of the shopping application , a new requirement was introduced that , the products that are going to be displayed has to be sorted in the descending order of the expiry date.
Find the correct option below which can satisfy given requirement.

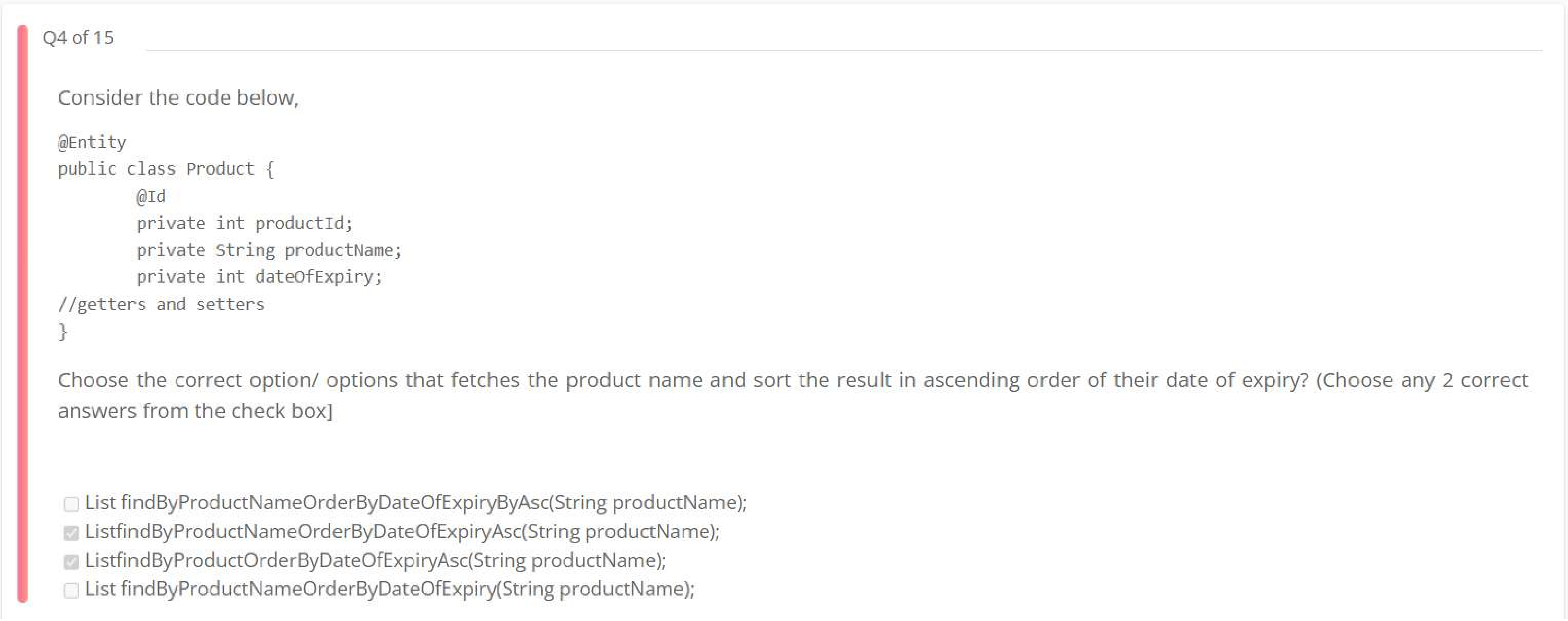
- List products = productRepository.findAll(Direction.DESC, "expiryDate”);
- [✓ List products = productRepository.findAll(Sort.Direction.DESC, "expiryDate”);]
- List products = productRepository.findAll(SortBy.sort(Direction "expiryDate”));
- List products = productRepository.findAll(Sort.by(Sort.Direction.DESC, "expiryDate");



Q3 of 15

Which of the following methods will be available in a repository interface extending CrudRepository?(Choose any TWO correct answers)

- ☐ Optional existsById(ID id)
- ☐ Int delete(T entity)
- ☑ Optional findById(ID id)
- ☑ Iterable saveAll(Iterable entities)



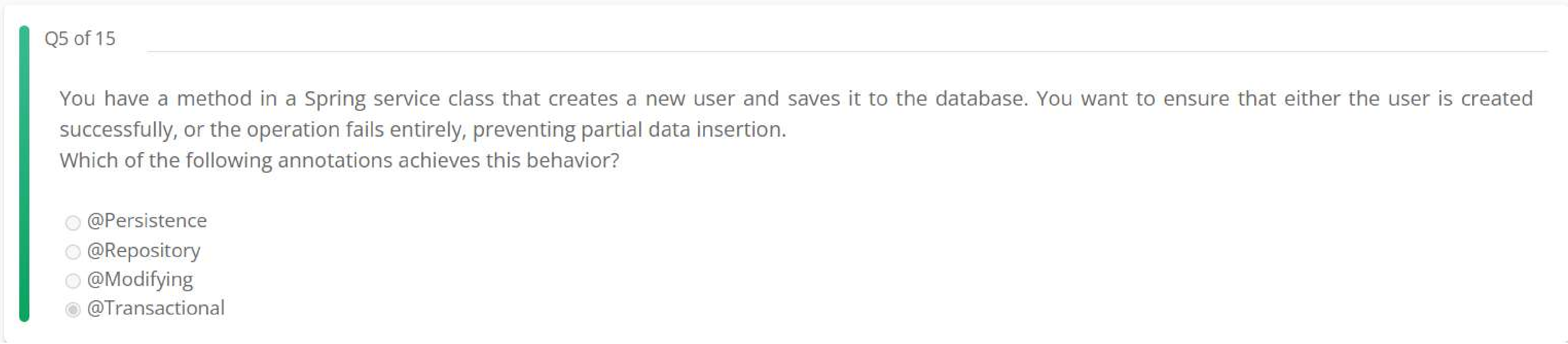
Q4 of 15

Consider the code below,

```
@Entity
public class Product {
        @Id
        private int productId;
        private String productName;
        private int dateOfExpiry;
//getters and setters
}
```

Choose the correct option/ options that fetches the product name and sort the result in ascending order of their date of expiry? (Choose any 2 correct answers from the check box]

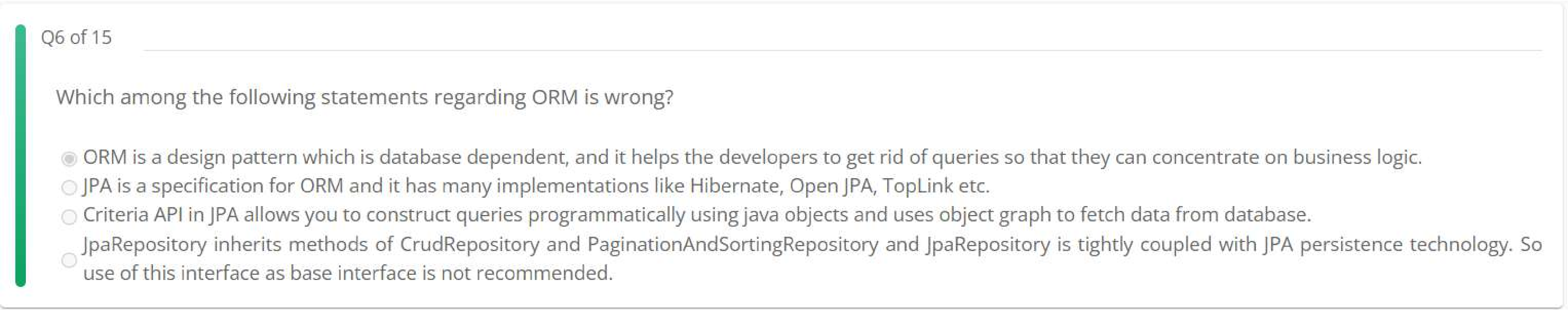
- [ ] List findByProductNameOrderByDateOfExpiryByAsc(String productName);
- [✓] ListfindByProductNameOrderByDateOfExpiryAsc(String productName);
- [✓] ListfindByProductOrderByDateOfExpiryAsc(String productName);
- [ ] List findByProductNameOrderByDateOfExpiry(String productName);



Q5 of 15

You have a method in a Spring service class that creates a new user and saves it to the database. You want to ensure that either the user is created successfully, or the operation fails entirely, preventing partial data insertion.
Which of the following annotations achieves this behavior?

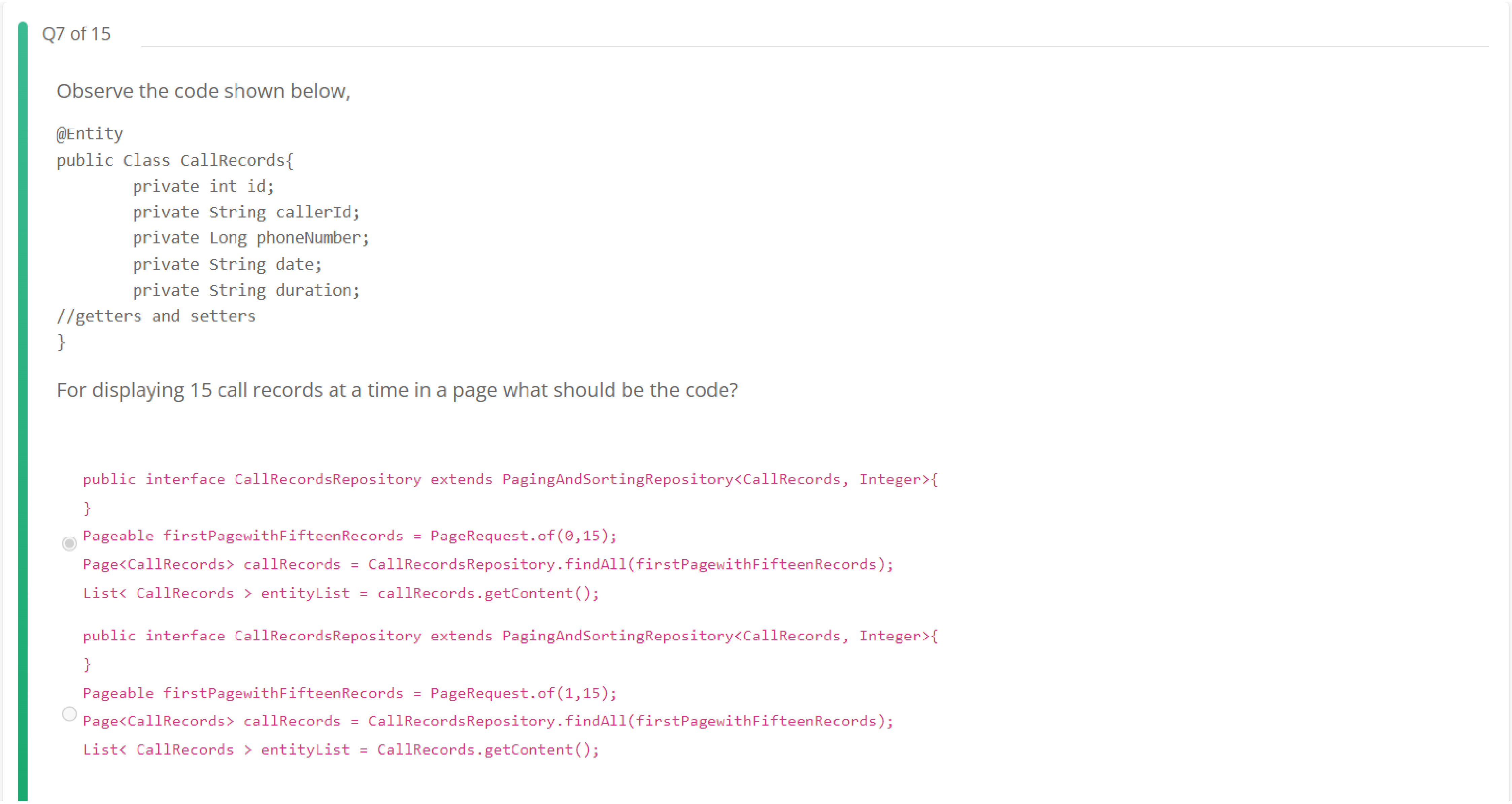
- @Persistence
- @Repository
- @Modifying
- [✓ @Transactional]



Q6 of 15

Which among the following statements regarding ORM is wrong?

- [✓ ORM is a design pattern which is database dependent, and it helps the developers to get rid of queries so that they can concentrate on business logic.]
- JPA is a specification for ORM and it has many implementations like Hibernate, Open JPA, TopLink etc.
- Criteria API in JPA allows you to construct queries programmatically using java objects and uses object graph to fetch data from database.
- JpaRepository inherits methods of CrudRepository and PaginationAndSortingRepository and JpaRepository is tightly coupled with JPA persistence technology. So use of this interface as base interface is not recommended.



Q7 of 15

Observe the code shown below,

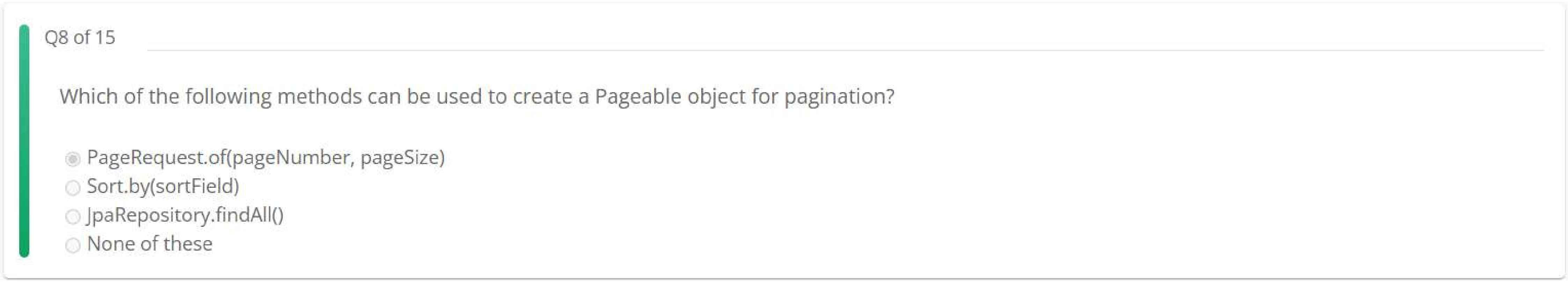
```
@Entity
public Class CallRecords{
        private int id;
        private String callerId;
        private Long phoneNumber;
        private String date;
        private String duration;
//getters and setters
}
```

For displaying 15 call records at a time in a page what should be the code?

[✓]

```
public interface CallRecordsRepository extends PagingAndSortingRepository<CallRecords, Integer>{
}
Pageable firstPagewithFifteenRecords = PageRequest.of(0,15);
Page<CallRecords> callRecords = CallRecordsRepository.findAll(firstPagewithFifteenRecords);
List< CallRecords > entityList = callRecords.getContent();
```

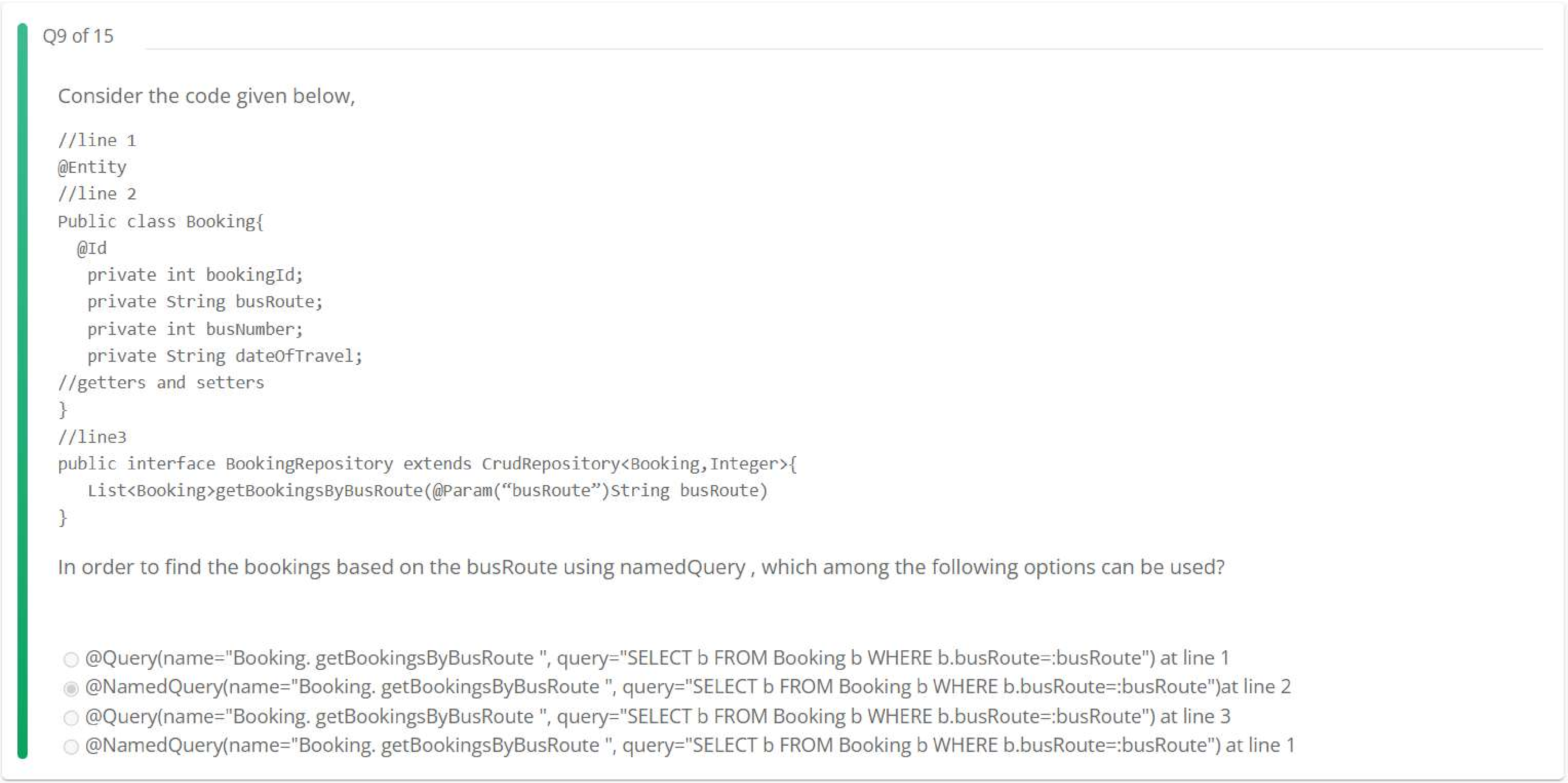
```
public interface CallRecordsRepository extends PagingAndSortingRepository<CallRecords, Integer>{
}
Pageable firstPagewithFifteenRecords = PageRequest.of(1,15);
Page<CallRecords> callRecords = CallRecordsRepository.findAll(firstPagewithFifteenRecords);
List< CallRecords > entityList = CallRecords.getContent();
```



Q8 of 15

Which of the following methods can be used to create a Pageable object for pagination?

- [✓ PageRequest.of(pageNumber, pageSize)]
- Sort.by(sortField)
- JpaRepository.findAll()
- None of these



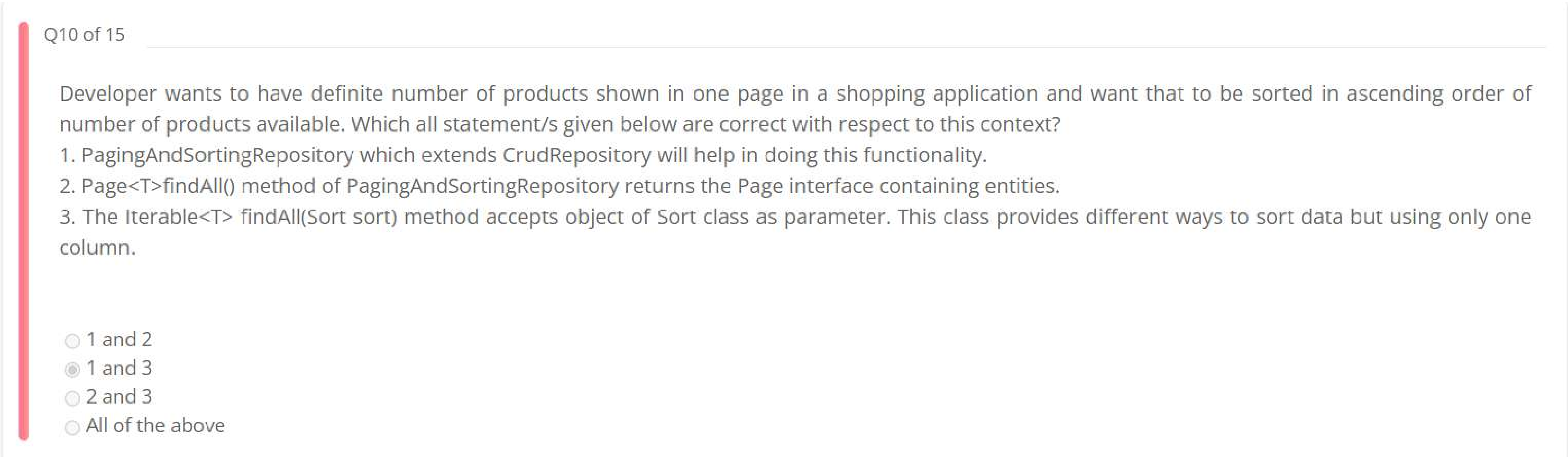
Q9 of 15

Consider the code given below,

```
//line 1
@Entity
//line 2
Public class Booking{
  @Id
   private int bookingId;
   private String busRoute;
   private int busNumber;
   private String dateOfTravel;
//getters and setters
}
//line3
public interface BookingRepository extends CrudRepository<Booking,Integer>{
   List<Booking>getBookingsByBusRoute(@Param(“busRoute”)String busRoute)
}
```

In order to find the bookings based on the busRoute using namedQuery , which among the following options can be used?

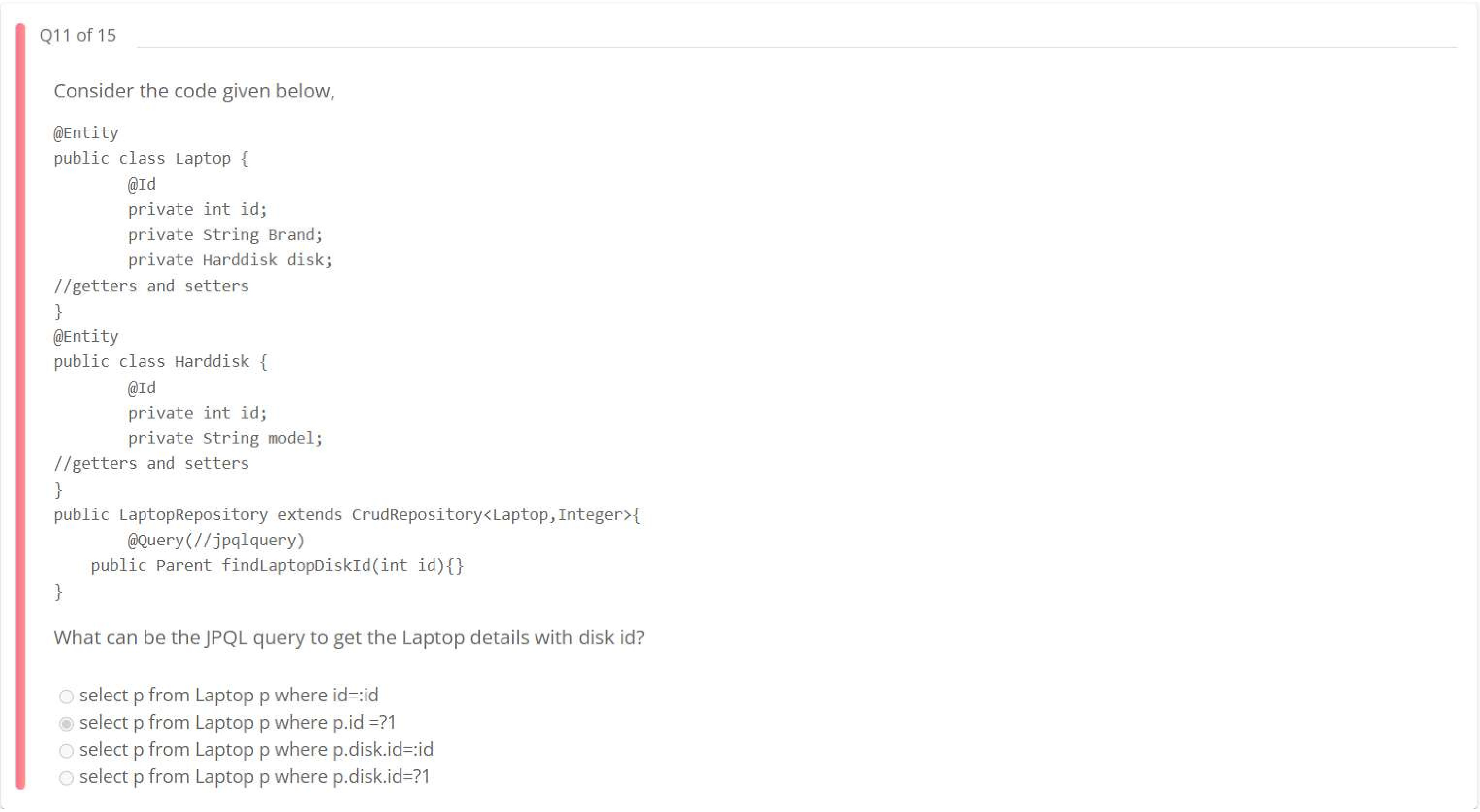
- @Query(name="Booking. getBookingsByBusRoute ", query="SELECT b FROM Booking b WHERE b.busRoute=:busRoute") at line 1
- [✓ @NamedQuery(name="Booking. getBookingsByBusRoute ", query="SELECT b FROM Booking b WHERE b.busRoute=:busRoute")at line 2]
- @Query(name="Booking. getBookingsByBusRoute ", query="SELECT b FROM Booking b WHERE b.busRoute=:busRoute") at line 3
- @NamedQuery(name="Booking. getBookingsByBusRoute ", query="SELECT b FROM Booking b WHERE b.busRoute=:busRoute") at line 1



Q10 of 15

Developer wants to have definite number of products shown in one page in a shopping application and want that to be sorted in ascending order of number of products available. Which all statement/s given below are correct with respect to this context?
1. PagingAndSortingRepository which extends CrudRepository will help in doing this functionality.
2. Page<T>findAll() method of PagingAndSortingRepository returns the Page interface containing entities.
3. The Iterable<T> findAll(Sort sort) method accepts object of Sort class as parameter. This class provides different ways to sort data but using only one column.

- 1 and 2
- [✓ 1 and 3]
- 2 and 3
- All of the above



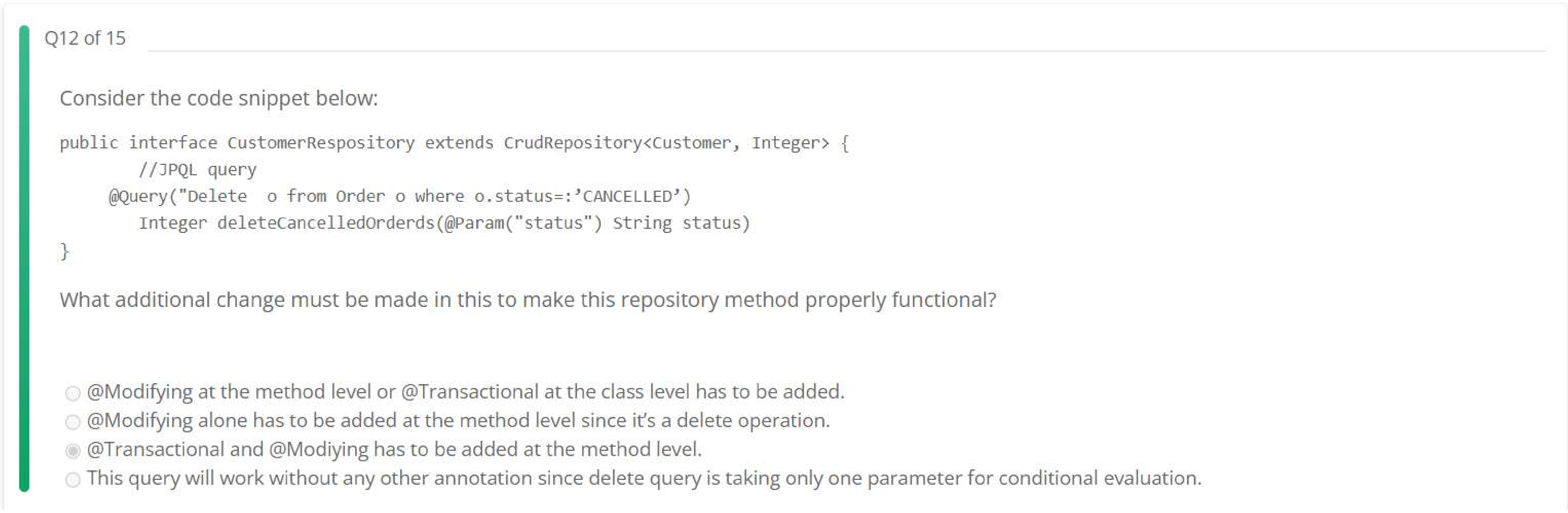
Q11 of 15

Consider the code given below,

```
@Entity
public class Laptop {
        @Id
        private int id;
        private String Brand;
        private Harddisk disk;
//getters and setters
}
@Entity
public class Harddisk {
        @Id
        private int id;
        private String model;
//getters and setters
}
public LaptopRepository extends CrudRepository<Laptop,Integer>{
        @Query(//jpqlquery)
    public Parent findLaptopDiskId(int id){}
}
```

What can be the JPQL query to get the Laptop details with disk id?

- select p from Laptop p where id=:id
- [✓ select p from Laptop p where p.id =?1]
- select p from Laptop p where p.disk.id=:id
- select p from Laptop p where p.disk.id=?1



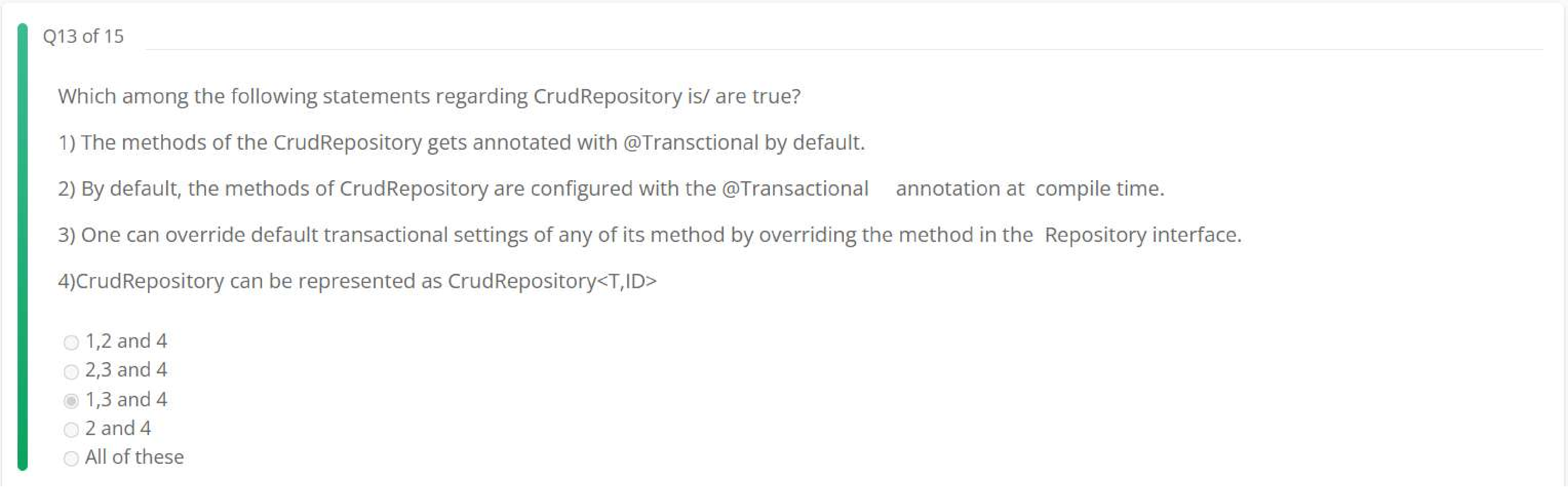
Q12 of 15

Consider the code snippet below:

```
public interface CustomerRespository extends CrudRepository<Customer, Integer> {
        //JPQL query
     @Query("Delete  o from Order o where o.status=:'CANCELLED')
        Integer deleteCancelledOrderds(@Param("status") String status)
}
```

What additional change must be made in this to make this repository method properly functional?

- @Modifying at the method level or @Transactional at the class level has to be added.
- @Modifying alone has to be added at the method level since it's a delete operation.
- [✓ @Transactional and @Modiying has to be added at the method level.]
- This query will work without any other annotation since delete query is taking only one parameter for conditional evaluation.



Q13 of 15

Which among the following statements regarding CrudRepository is/ are true?

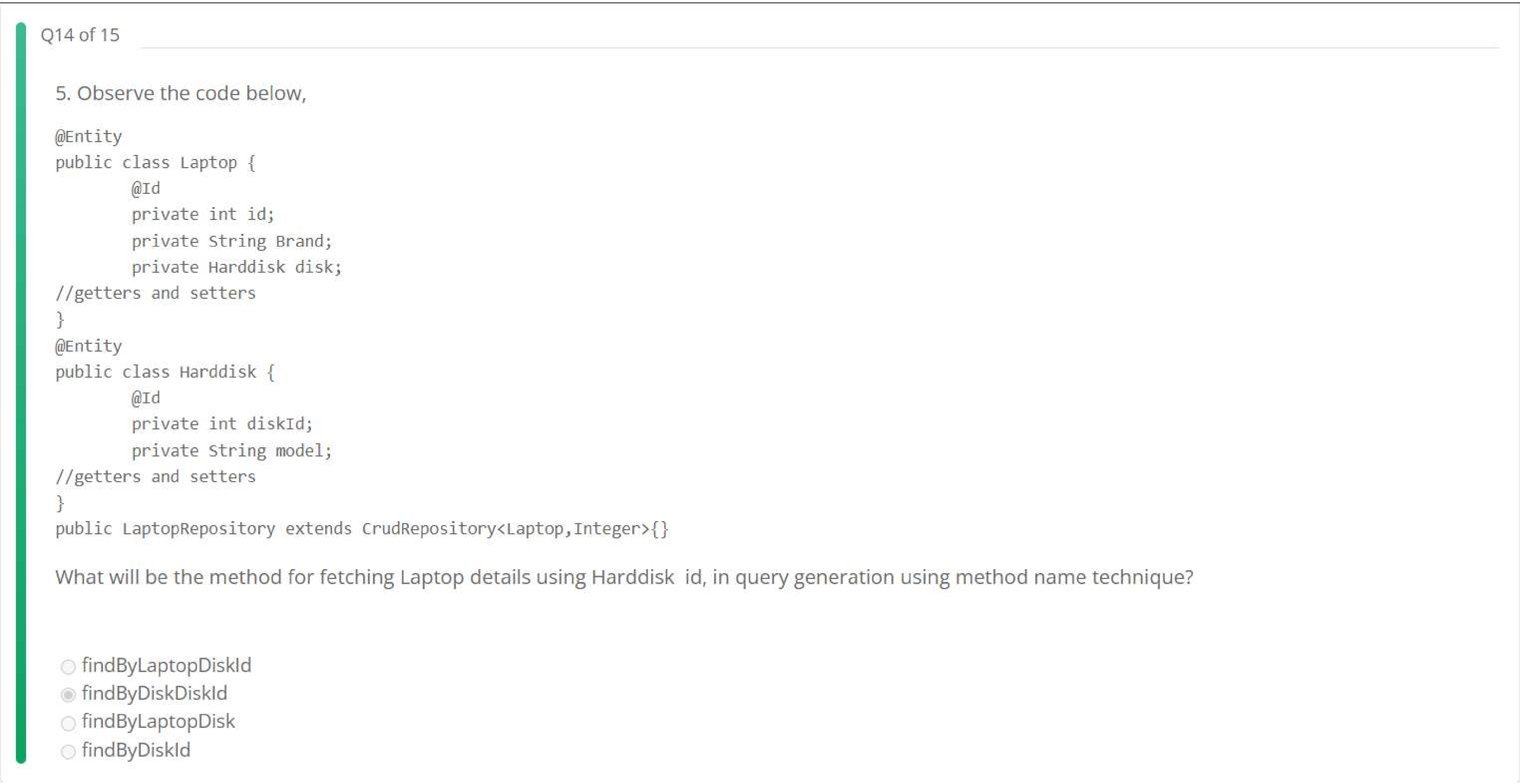
1) The methods of the CrudRepository gets annotated with @Transctional by default.

2) By default, the methods of CrudRepository are configured with the @Transactional annotation at compile time.

3) One can override default transactional settings of any of its method by overriding the method in the Repository interface.

4)CrudRepository can be represented as CrudRepository<T,ID>

- 1,2 and 4
- 2,3 and 4
- [✓ 1,3 and 4]
- 2 and 4
- All of these



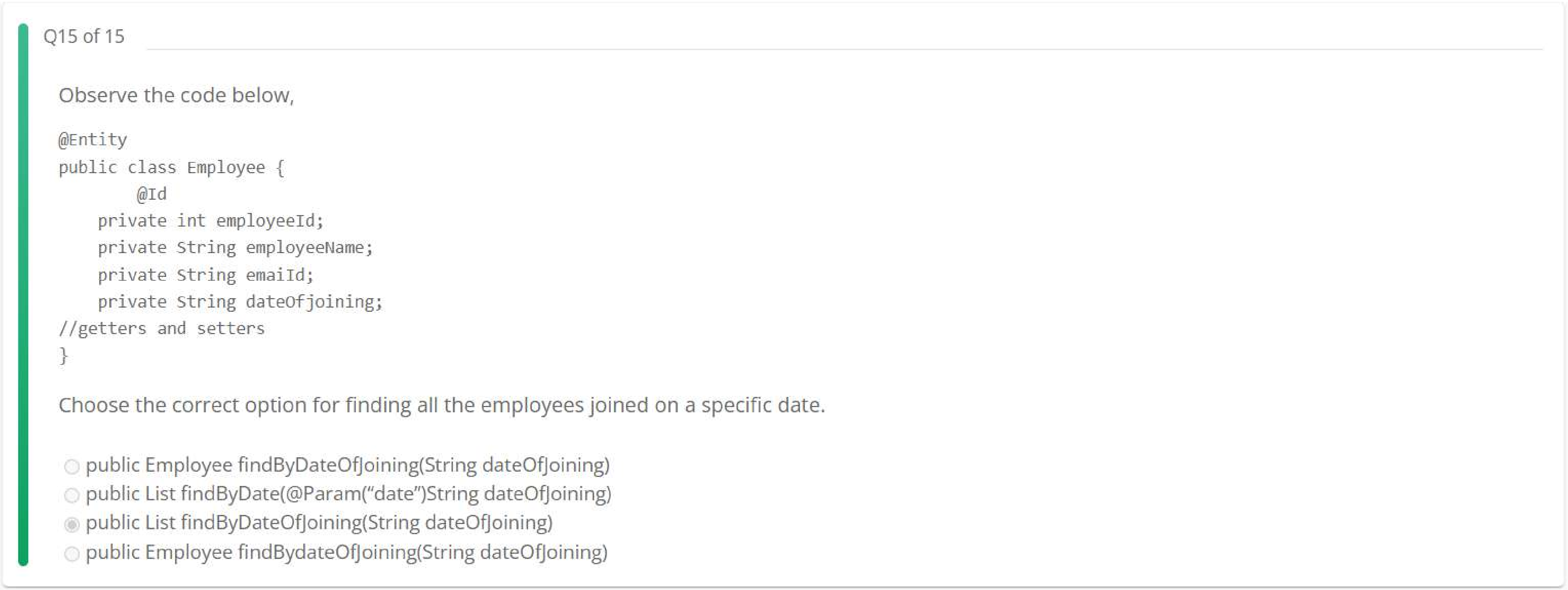
Q14 of 15

5. Observe the code below,

```
@Entity
public class Laptop {
        @Id
        private int id;
        private String Brand;
        private Harddisk disk;
//getters and setters
}
@Entity
public class Harddisk {
        @Id
        private int diskId;
        private String model;
//getters and setters
}
public LaptopRepository extends CrudRepository<Laptop,Integer>{}
```

What will be the method for fetching Laptop details using Harddisk  id, in query generation using method name technique?

- findByLaptopDiskId
- [✓ findByDiskDiskId]
- findByLaptopDisk
- findByDiskId



Q15 of 15

Observe the code below,

```
@Entity
public class Employee {
        @Id
    private int employeeId;
    private String employeeName;
    private String emailId;
    private String dateOfjoining;
//getters and setters
}
```

Choose the correct option for finding all the employees joined on a specific date.

- public Employee findByDateOfJoining(String dateOfJoining)
- public List findByDate(@Param("date")String dateOfJoining)
- [✓ public List findByDateOfJoining(String dateOfJoining)]
- public Employee findBydateOfJoining(String dateOfJoining)