

**CS544**  
**Enterprise Architecture**  
**Exam 1 March 2018**

Name \_\_\_\_\_

Student ID \_\_\_\_\_

**NOTE: This material is private and confidential. It is the property of MUM and is not to be disseminated.**

1. [15 points] **Circle** which of the following is TRUE/FALSE concerning Spring Inversion of Control/Dependency Injection:

T F    Only Managed Beans can be injected in Spring, a POJO or JavaBean cannot.

EXPLAIN: \_\_\_\_\_.

T F    @Autowired works only on interfaces. It cannot work directly on classes.

EXPLAIN: \_\_\_\_\_.

T F    In practice, the IoC container is not exactly the same as Dependency Injection as it involves a discovery step concerning the dependency.

EXPLAIN: \_\_\_\_\_.

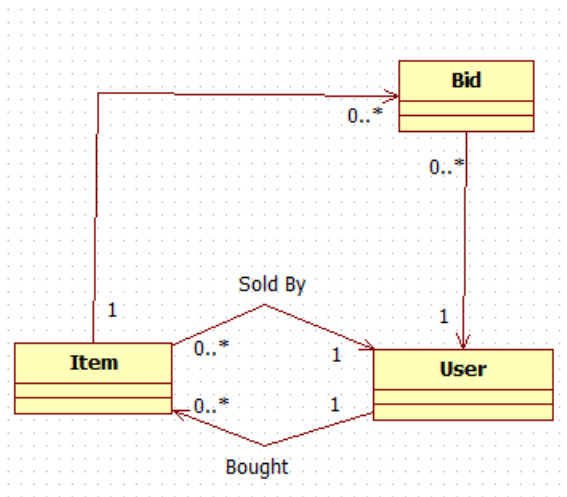
T F    A domain object needed in a @Service class is usually a good candidate for Dependency Injection

EXPLAIN: \_\_\_\_\_.

T F    In Spring, DI can be done through either XML or through Annotations. They are mutually exclusive. That means, if you use XML for DI for one bean, they you should use it, exclusively for all beans.

EXPLAIN: \_\_\_\_\_.

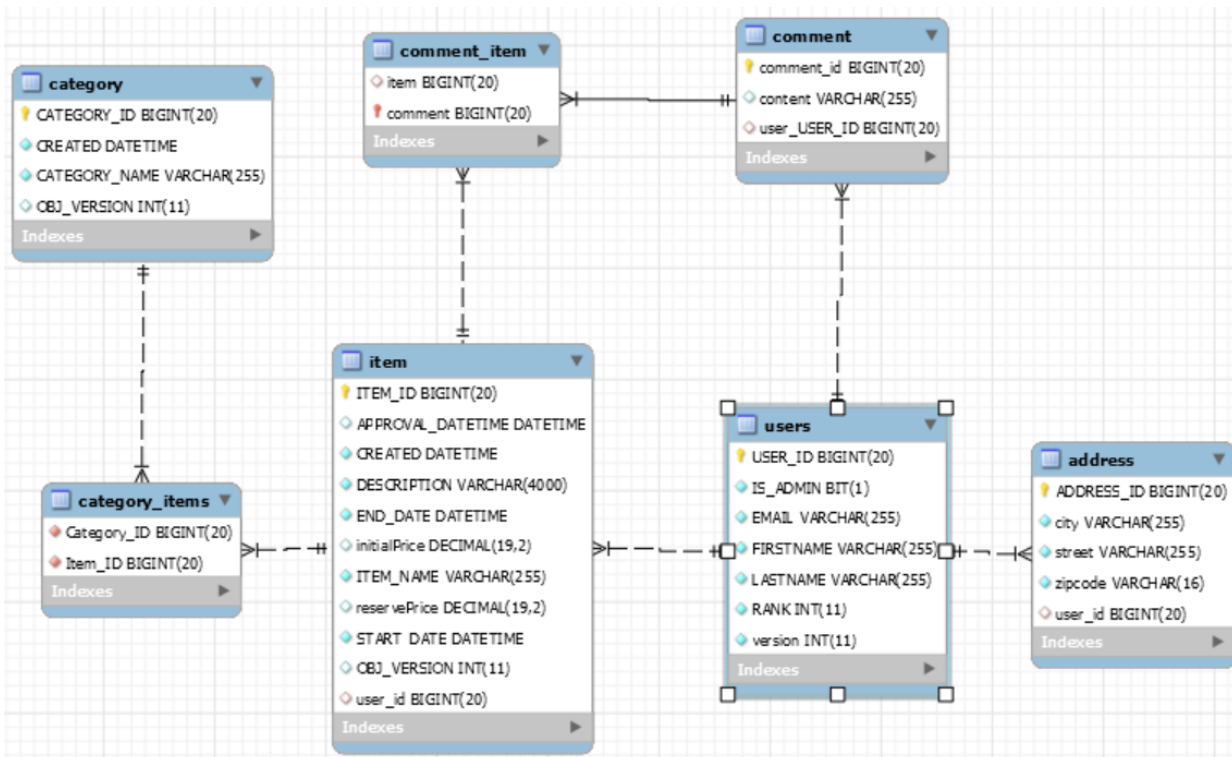
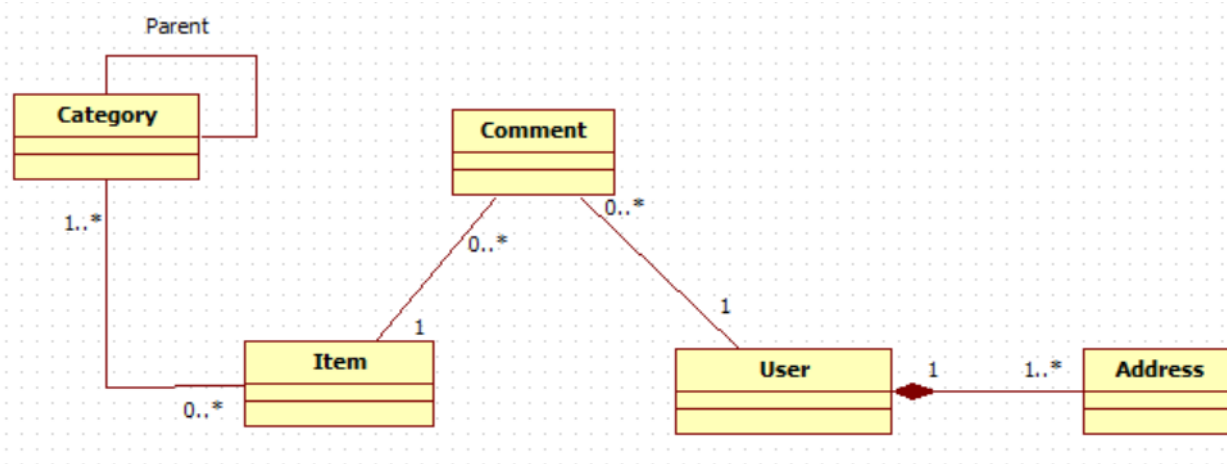
2. [15 points] For the following relationships implement a SubSelect that fetches all items with their corresponding collection of bids. The bid collection should be set as Lazy loaded.



What performance problem[s] does the SubSelect fetch address? What are its issues?

How does it work? – Explain the “algorithm” based on a universe of 10 Items each with a collection of 5-10 Bids. Compare it to Join Fetch.

3. [20 points] Annotate the Domain Objects based on the Domain Model and Entity Relationship Diagram provided. NOTE: All the Domain Objects are not listed. All the fields are not listed. Only annotate the objects and fields that are listed.



```
public class User {  
  
    private Long id = null;  
  
    private int version = 0;  
  
    private String firstName;  
  
    private String lastName;  
  
    private String email;  
  
    private int ranking = 0;  
  
    private boolean admin = false;  
  
    List<Comment> comments;  
  
    List<Address> addresses;
```

```
public class Comment {  
  
    private Long id = null;  
  
    private User user;  
  
    private Item item;  
  
    private String content;
```

```
public class Item {  
  
    private String name;  
  
    private String description;  
  
    private BigDecimal reservePrice;  
  
    private Set<Category> categories ;  
  
    List<Comment> comments;
```

```
public class Category {  
  
    private Long id = null;  
  
    private int version = 0;  
  
    private String name;  
  
    private List<Item> items;
```

4. [15 points] Explain the concept of locking. Include the definition of the two strategies covered in class. Give the details of Version-Based Optimistic Concurrency [Locking], how it relates to isolation levels, how it is implemented in JPA.

5. [15 points] Implement a JQPL **parameterized** query that looks up a User **by email** who is selling a specific Item with an initial price greater than a specified dollar value.

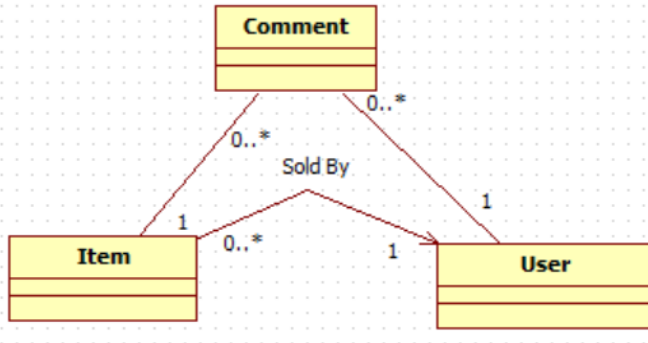
For example:

*Find John Smith who is selling an Item, “cardboard box” with an initial price greater than 70.00.*

Another example:

*Find Will Henry who is selling an Item named “Pencil Set” with an initial price greater than 100.00.*

**Item – User relationship [See relevant class properties in previous problem]:**



In Item.java:

```
@ManyToOne(fetch=FetchType.LAZY)
@JoinColumn (name="user_id")
private User seller;
```

Remember the Query is a **parameterized query**. Also identify all the classes in the specific packages that need to be modified to implement the query in accordance with the N-Tier architecture convention. Describe the “pattern” that exists at the persistence layer.

**ANSWER:**