CS544
Enterprise Architecture
Exam 1 Sample

Name_____

Student ID _____

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

1. [10 points] **Circle w**hich of the following is TRUE/FALSE concerning Spring Inversion of Control/Dependency Injection:

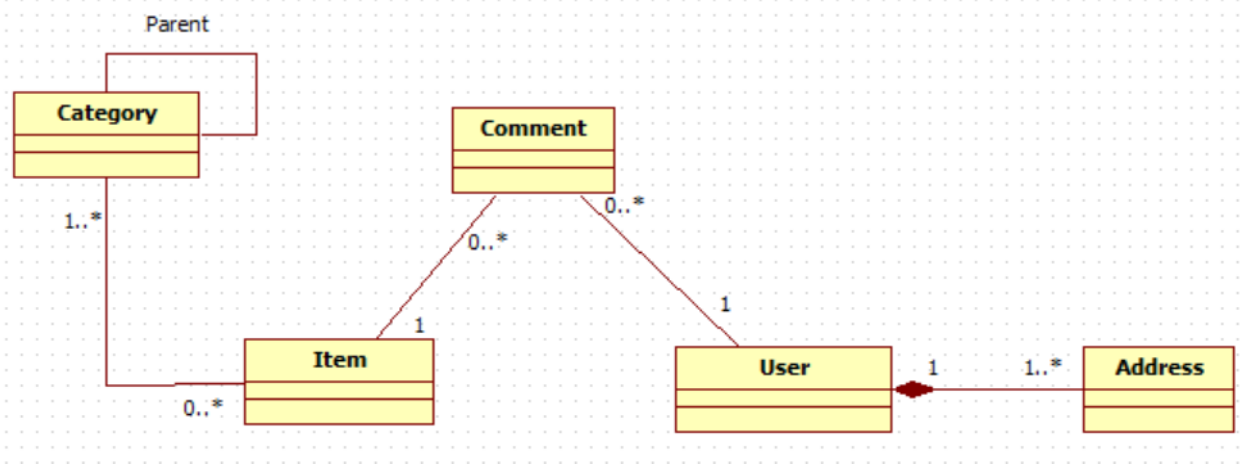
   T **F** Only Managed Beans can be injected in Spring, a POJO or JavaBean cannot.

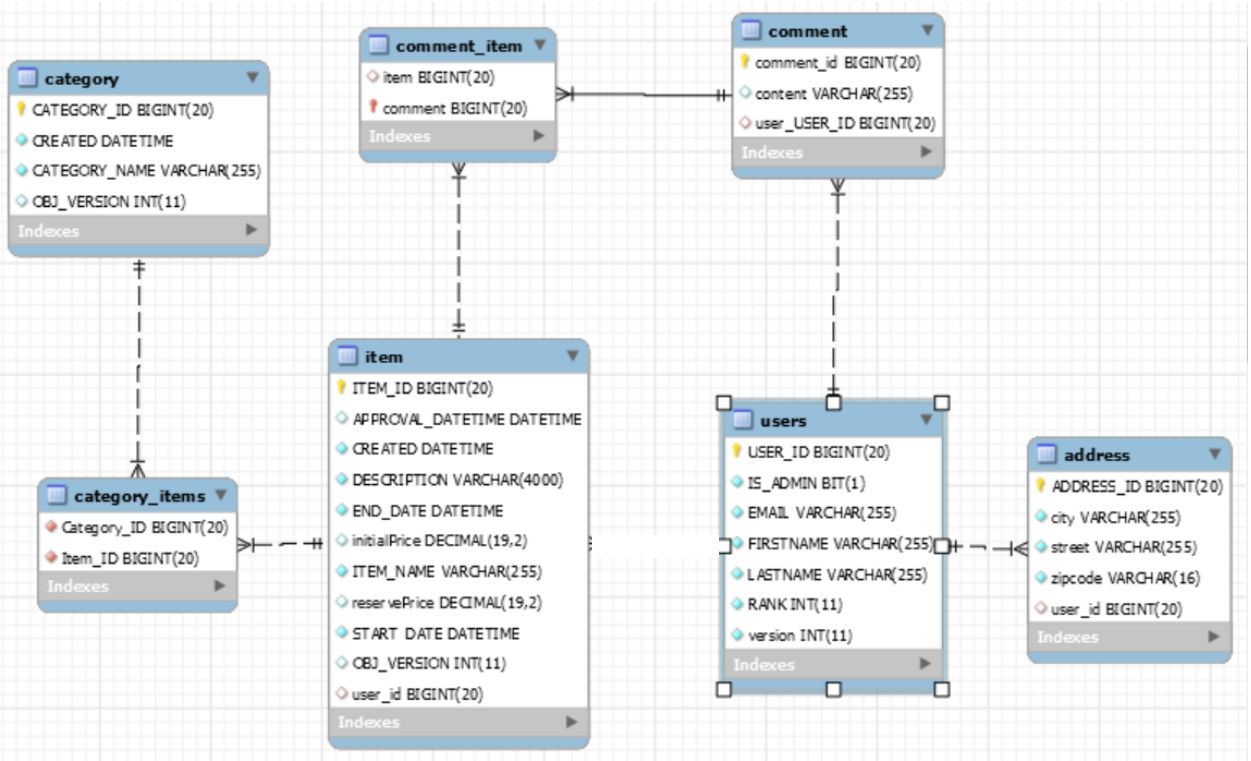
   EXPLAIN:____If the POJO or JavaBean is a Spring Managed bean, they can be injected.

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

   EXPLAIN:__ It can work on classes. However you lose some of the value, testing; changing implementations

2. [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.





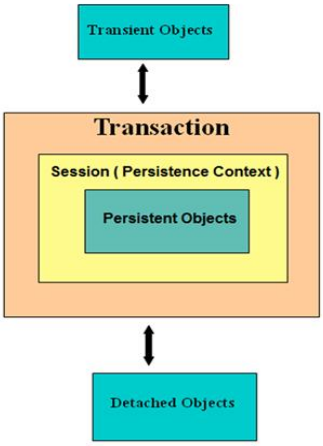
```java
@Entity
@Table(name = "USERS")
public class User implements Serializable  {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "USER_ID")
    private Long id = null;

    @Version
    private int version = 0;


     @Column(name = "FIRSTNAME", nullable = false)
    private String firstName;

    @Column(name = "LASTNAME", nullable = false)
    private String lastName;

    @Column(name = "EMAIL", nullable = false)
    private String email;

    @Column(name = "RANK", nullable = false)
    private int ranking = 0;

    @Column(name = "IS_ADMIN", nullable = false)
    private boolean admin = false;


    @OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.PERSIST, mappedBy="user")
     List<Comment> comments;

    @OneToMany(mappedBy="user", fetch=FetchType.LAZY, cascade = CascadeType.PERSIST)
    List<Address> addresses;

```

```java
@Entity
public class Comment {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="comment_id")
    private long id;
     @JoinColumn
    @ManyToOne(fetch=FetchType.EAGER)
    private User user;


    @ManyToOne(fetch=FetchType.EAGER, cascade = {CascadeType.PERSIST, CascadeType.MERGE})
    @JoinTable ( name="comment_item", joinColumns={@JoinColumn(name="comment")},
    inverseJoinColumns={ @JoinColumn(name="item")} )
    private Item item;

    private String content;

```

```java
@Entity
@Table(name = "ITEM")
 public class Item implements Serializable {

     @Id @GeneratedValue
    @Column(name = "ITEM_ID")
    private Long id = null;

    @Version
    @Column(name = "OBJ_VERSION")
    private int version = 0;

    @Column(name = "ITEM_NAME", length = 255, nullable = false, updatable = false)
    private String name;

    @Column(name = "DESCRIPTION", length = 4000, nullable = false)
    private String description = "";

    private BigDecimal reservePrice;

    @ManyToMany(fetch = FetchType.EAGER, cascade= {CascadeType.PERSIST,CascadeType.MERGE}, mappedBy="items")
    private Set<Category> categories = new HashSet<Category>();

    @OneToMany( mappedBy= "item", fetch=FetchType.EAGER, cascade = CascadeType.PERSIST)
    List<Comment> comments;
```

```java
 @Entity
@Table(
   name = "CATEGORY")
public class Category implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "CATEGORY_ID")
    private Long id = null;

    @Version
    @Column(name = "OBJ_VERSION")
    private int version = 0;

    @Column(name = "CATEGORY_NAME", length = 255, nullable = false)
    private String name;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable ( name="Category_Items", joinColumns={@JoinColumn(name="Category_ID")},
    inverseJoinColumns={ @JoinColumn(name="Item_ID")} )
    private List<Item> items = new ArrayList<Item>();
```

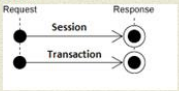
1. [15 points] In JPA, the Persistence Context plays an important role in the implementation of an ORM. Explain, by example the Entity lifecycle as it pertains to the following drawing.



ORM Session

Spring "manages" through @Transactional

- Transaction Unit of work
  - **Common Pattern: *session-per-request***
  - Session == Database Transaction
- START –
  - Open a Session
  - Open a single database connection
  - Start a Transaction



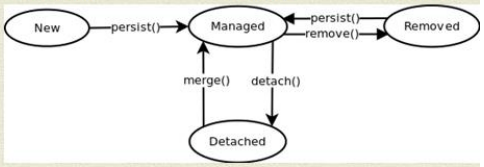
  - Associate & Manage entities W/R the session
  - Exercise DB CRUD operations

Session == Persistence Context

- END –
  - End Transaction
  - Close a Session.

ORM Entity Lifecycle

- **Transient** **Persistent** **Removed**



Detached

ORM-related Entity States

- ***Transient –***
- it has just been instantiated using the new operator
- not associated with a Session
- no persistent representation in the database
- ***Persistent –***
- representation in the database
- Has been saved or loaded in Session
- Changes made to an object are synchronized with the database when the unit of work completes..
- ***Detached –***
- An object that has been persistent, but Session has been closed
- ***Removed –***
- An object is deleted from the database when the unit of work completes

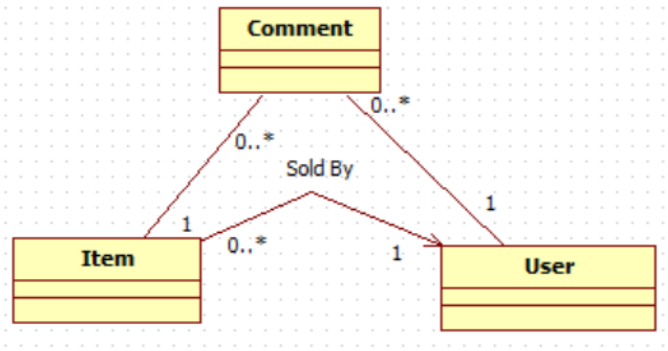
2. [15 points] Implement a JQPL **parameterized** query that looks up a User **by email** who is selling a specific Item with an initial price greater than a specified dollar value.
   For example:

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:

edu.mum.dao. UserDao

public User findBySoldItemInitialPrice(String email,String itemName, `BigDecimal initialPrice`);

edu.mum.dao.impl. UserDaoImpl

```
public User findBySoldItemInitialPrice(String email, String boughtItem, BigDecimal initialPrice) {
  Query query=entityManager.createQuery("select u from User u, Item i where i.seller.email=:email"
                          + "and i.name = :boughtItem and i.initialPrice > :initialPrice");
  return (User) query.setParameter("boughtItem", boughtItem).
       setParameter("email", email). setParameter("initialPrice", initialPrice).getSingleResult();

}
```

edu.mum.service. UserService

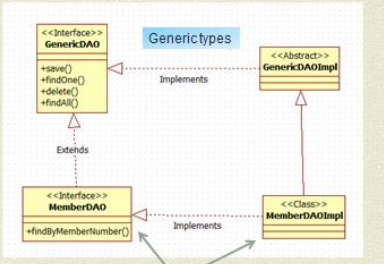
public User findBySoldItemInitialPrice(String email,String itemName, `BigDecimal initialPrice`);

edu.mum.service.impl. UserServiceImpl

public User findBySoldItemInitialPrice(String email,String itemName, `BigDecimal initialPrice`) {
return userDao.findBySoldItemInitialPrice(email,itemName, `BigDecimal initialPrice`);

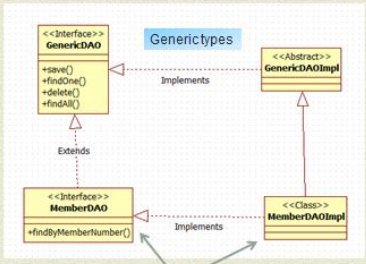
}

"Classic" ORM GenericDAO



1

"Classic" ORM GenericDAO



Generic DAO Implementation

```
public abstract class GenericDaoImpl<T> implements GenericDao<T> {

@PersistenceContext
    protected EntityManager entityManager;
    protected Class<T> daoType;

     public void setDaoType(Class<T> type) {
            daoType = type;
     }
    @Override
    public void save( T entity ){
        entityManager.persist( entity );
     }
    public void delete( T entity ){
        entityManager.remove( entity );
```


Domain Class specific DAO

```
public interface MemberDao extends GenericDao<Member> {
    public Member findByMemberNumber(Integer number);

public class MemberDaoImpl extends GenericDaoImpl<Member> implements MemberDao
public MemberDaoImpl() {
super.setDaoType(Member.class);
}
 public Member findByMemberNumber(Integer number) {
   Query query = entityManager.createQuery("select m from MEMBER m
                                where m.memberNumber =:number");
  return (Member) query.setParameter("number", number).getSingleResult();
}
```