

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. [10 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: ___ 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

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: ___ IoC involves a “look up the dependency”[Pull] step before injecting it. Spring has declarative configuration that identifies “where to find” the resource to inject [Push].

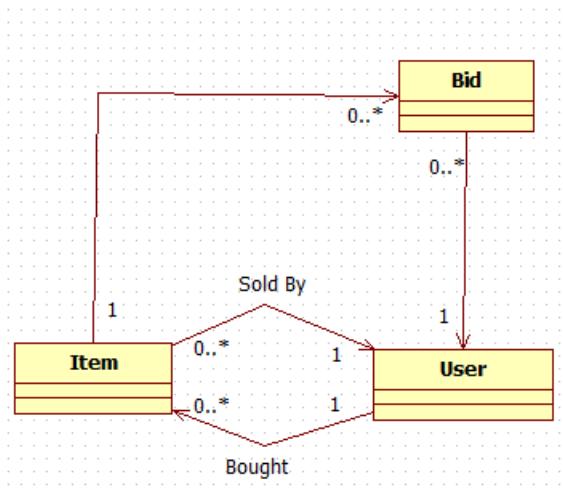
T F A domain object needed in a @Service class is usually a good candidate for Dependency Injection

EXPLAIN: ___ DI is mainly used for “cross” layer transitions, access plumbing resources, etc. NOT for passing business data around...

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: ___ You could inject a DAO[memberDAO in memberService] through XML & inject another DAO[productDAO in productService] through @Autowired. Remember, XML configuration is the final word – if you configure the same bean twice, the XML configuration will take precedence.

2. [15 points] For the following relationships implement a SubSelect that fetches all items with their corresponding collection of bids.



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.

In Item.java

```

public class Item {

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @Fetch(FetchMode.SUBSELECT)
    private List<Bid> bids = new ArrayList<Bid>();
  
```

In ItemServiceImpl.java

```

public List<Item> findBySubSelect() {
    List<Item> items = (List<Item>)this.findAll();
    // hydrate since LAZY load
    items.get(0).getBids().get(0);

    return items;
}
  
```

Subselect does ONE fetch for the Items and ONE fetch for ALL collections. Therefore the number of Items & Bids in the collections does not matter,

It solves the Cartesian issue && the N+1 issue.

If the collection is LAZY loaded, then doing a get all for the “parent” entities [Item as in above] will get the “parent”/Item list. Within the same transaction, a reference to ONE of the Bid collections will fetch ALL the collections.

The Join Fetch, on the other hand will get ALL the Bids AND Items in ONE Select/fetch.

The Join Fetch, however suffers from the Cartesian product issue [ItemsXBids] which means that more than one copy of each Item/Bid pair will be included in the fetch. The number of copies depends on the number of Bids in the collection. So if Item A has 3 bids, 3 copies of Item A will be present.

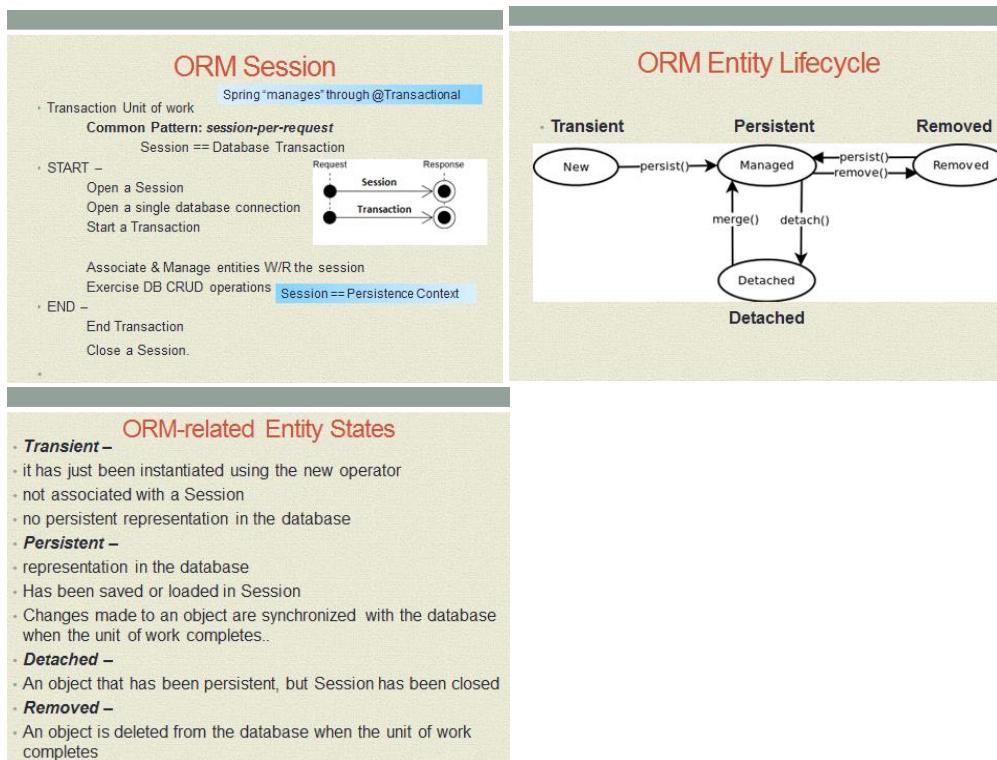
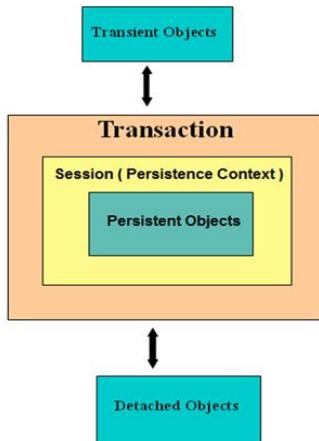
The screenshot shows two slides side-by-side. The left slide, titled 'Hibernate [Default] FETCH Strategy' (slide 4), lists four types of fetching:

- Select fetching:** a second SELECT (per parent N) is used to retrieve the associated collection. `[DEFAULT] [N+1 Fetches]@Fetch(FetchMode.SELECT)`
- Join fetching:** associated collections are retrieved in the same SELECT, using an OUTER JOIN. `[1 Fetch] [EAGER] @Fetch(FetchMode.JOIN)`
- Subselect fetching:** a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. `[2 Fetches]`
- Batch fetching:** Optimization of Select Fetching. Associated collections are fetched according to declared Batch Size(NBatch Size) + 1; `@BatchSize(size=n)`

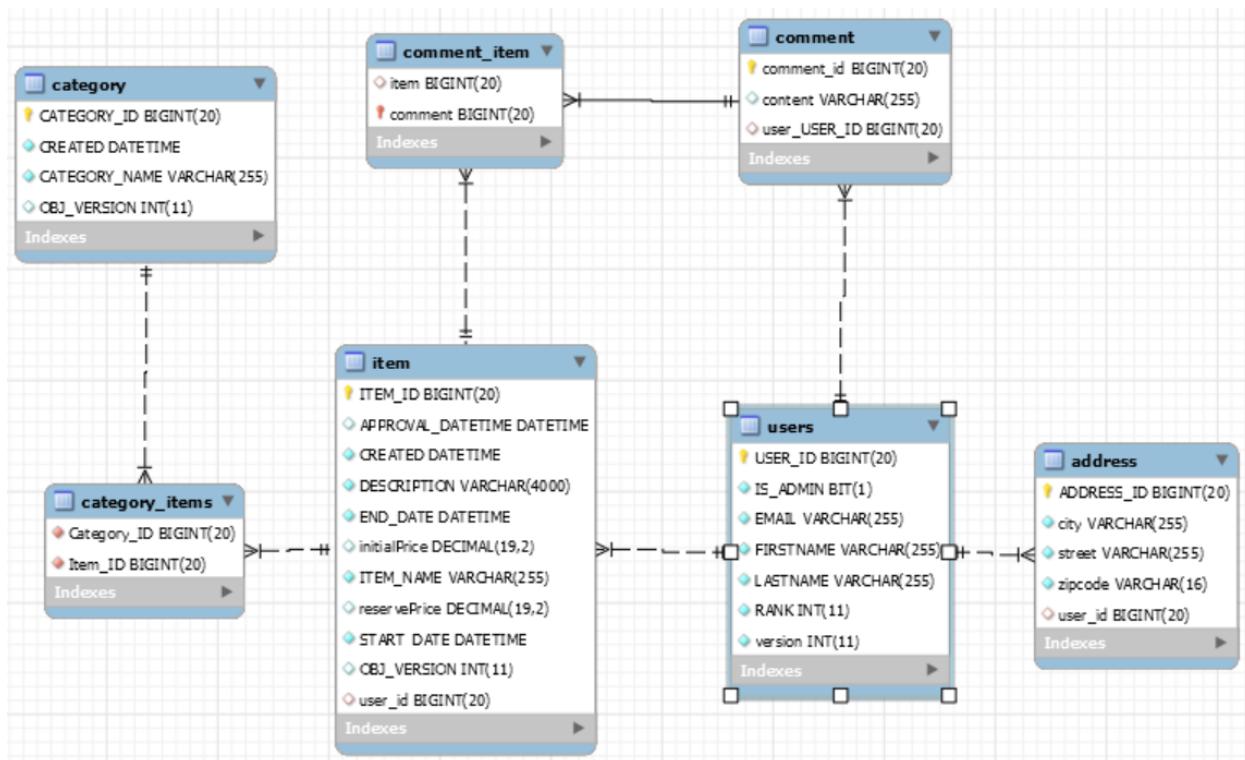
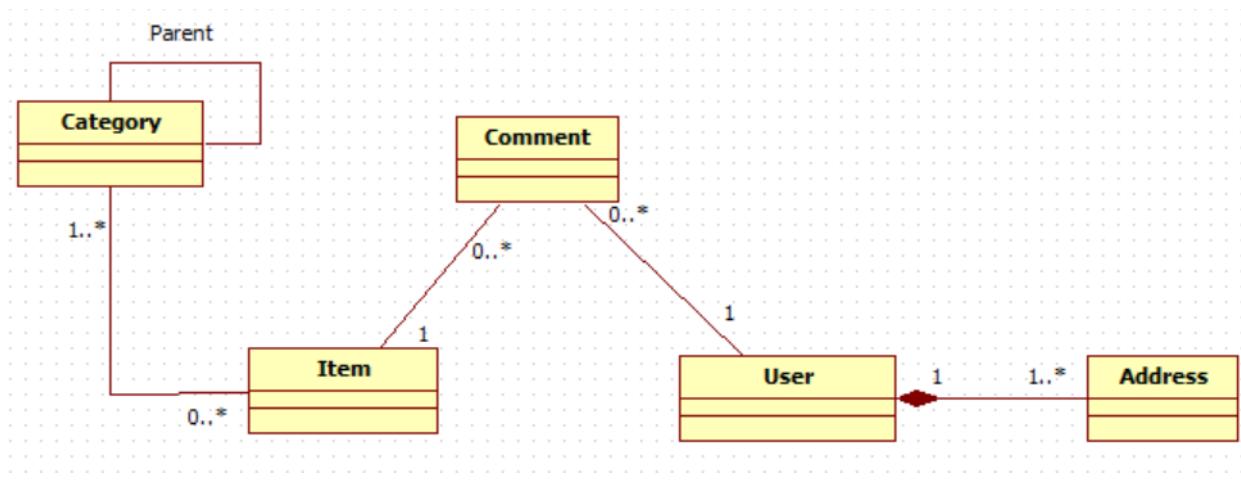
The right slide, titled 'Hibernate Fetch Strategy Issues' (slide 5), lists several issues:

- SubSelect**: depends on the "parent" query. If parent Query is complex, it could have performance impacts.
If `fetch=FetchType.LAZY` need to "hydrate" children
- BatchSize**: # of Fetches "unknown" UNLESS size of parent is constant
Batch fetching is often called a blind-guess optimization
If `fetch=FetchType.LAZY` need to "hydrate" children
- Select**:
 - N+1 TBA [To Be Avoided]
- Join**:
 - Cartesian – need to watch collection sizes; can be useful strategy

3. [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.



4. [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.



```

17 @Entity
18 @Table(name = "USERS")
19 public class User implements Serializable {
20
21     @Id @GeneratedValue(strategy=GenerationType.AUTO)
22     @Column(name = "USER_ID")
23     private Long id = null;
24
25     @Version
26     private int version = 0;
27
28
29     @Column(name = "FIRSTNAME", nullable = false)
30     private String firstName;
31
32     @Column(name = "LASTNAME", nullable = false)
33     private String lastName;
34
35     @Column(name = "EMAIL", nullable = false)
36     private String email;
37
38     @Column(name = "RANK", nullable = false)
39     private int ranking = 0;
40
41     @Column(name = "IS_ADMIN", nullable = false)
42     private boolean admin = false;
43
44
45     @OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.PERSIST, mappedBy="user")
46     List<Comment> comments;
47
48     @OneToMany(mappedBy="user", fetch=FetchType.LAZY, cascade = CascadeType.PERSIST)
49     List<Address> addresses;
50

```

```

15 @Entity
16 public class Comment {
17     @Id
18     @GeneratedValue(strategy=GenerationType.AUTO)
19     @Column(name="comment_id")
20     private long id;
21     @JoinColumn
22     @ManyToOne(fetch=FetchType.EAGER)
23     private User user;
24
25
26     @ManyToOne(fetch=FetchType.EAGER, cascade = {CascadeType.PERSIST, CascadeType.MERGE})
27     @JoinTable ( name="comment_item", joinColumns={@JoinColumn(name="comment")},
28     inverseJoinColumns={ @JoinColumn(name="item") } )
29     private Item item;
30
31     private String content;
32

```

```

17 @Entity
18 @Table(name = "ITEM")
19 public class Item implements Serializable {
20
21     @Id @GeneratedValue
22     @Column(name = "ITEM_ID")
23     private Long id = null;
24
25     @Version
26     @Column(name = "OBJ_VERSION")
27     private int version = 0;
28
29     @Column(name = "ITEM_NAME", length = 255, nullable = false, updatable = false)
30     private String name;
31
32     @Column(name = "DESCRIPTION", length = 4000, nullable = false)
33     private String description = "";
34
35     private BigDecimal reservePrice;
36
37     @ManyToMany(fetch = FetchType.EAGER, cascade= {CascadeType.PERSIST,CascadeType.MERGE}, mappedBy="items")
38     private Set<Category> categories = new HashSet<Category>();
39
40     @OneToMany( mappedBy= "item", fetch=FetchType.EAGER, cascade = CascadeType.PERSIST)
41     List<Comment> comments;

```

```

7  @Entity
8  @Table(
9      name = "CATEGORY")
10 public class Category implements Serializable {
11
12     @Id
13     @GeneratedValue(strategy=GenerationType.AUTO)
14     @Column(name = "CATEGORY_ID")
15     private Long id = null;
16
17     @Version
18     @Column(name = "OBJ_VERSION")
19     private int version = 0;
20
21     @Column(name = "CATEGORY_NAME", length = 255, nullable = false)
22     private String name;
23
24     @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
25     @JoinTable ( name="Category_Items", joinColumns={@JoinColumn(name="Category_ID")},
26     inverseJoinColumns={ @JoinColumn(name="Item_ID")})
27     private List<Item> items = new ArrayList<Item>();
28
29

```

5. [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.

1	14
<h2>Lock Mode for Data Consistency</h2> <ul style="list-style-type: none"> Locking refers to actions taken to prevent data in a relational database from changing between the time it is read and the time that it is used. <h3>Locking strategies</h3> <ul style="list-style-type: none"> Optimistic Lock Concurrent transactions can complete without affecting each other, Transactions do not need to lock the data resources that they affect. Pessimistic Lock Concurrent transactions will conflict with each other Transactions require that data is locked when read and unlocked at commit. 	<h2>Version-Based Optimistic Concurrency[Locking]</h2> <ul style="list-style-type: none"> High-volume systems No Connection maintained to the Database [Detached Objects] Effective in Read-Often Write-Sometimes scenario Uses read committed isolation level <ul style="list-style-type: none"> Guarantees repeatable read isolation level An exception is thrown if a conflict occurs @Version Long version: When entity is updated - version field is incremented.

15	16
<h2>Version-Based Optimistic Concurrency</h2> <ul style="list-style-type: none"> Optimistic concurrency assumes that lost update conflicts generally don't occur <ul style="list-style-type: none"> Keeps version #s so that it knows when they do Guarantees best performance and scalability The default way to deal with concurrency There is no locking anywhere It works well with very long conversations, including those that span multiple transactions First commit wins instead of last commit wins <ul style="list-style-type: none"> An exception is thrown if a conflict would occur <ul style="list-style-type: none"> ObjectOptimisticLockingFailureException 	<h2>Optimistic Locking [Cont.]</h2> <ul style="list-style-type: none"> It works well with long conversations, including those that span multiple transactions LONG CONVERSATION Use Case: <ul style="list-style-type: none"> A multi-step dialog, for example a wizard dialog interacts with the user in several request/response cycles. session-per-request-with-detached-objects <ul style="list-style-type: none"> @Version Locking manages Detached object consistency <pre> sequenceDiagram participant S1 participant S2 participant Service S1->>Service: Request activate Service Note over Service: Detached Object Cache serverUpdate() Service-->>S1: Response deactivate Service S2->>Service: Request activate Service Note over Service: Detached Object Cache serverUpdate() Service-->>S2: Response deactivate Service </pre>

17	
<h2>JPA Optimistic Locking</h2> <p>JPA version-based concurrency control</p> <p>Simply add @Version to an Integer field – JPA takes care of the rest</p> <ul style="list-style-type: none"> @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long id = null; @Version @Column(name = "version") private int version = 0; Hibernate: <ul style="list-style-type: none"> update purchaseOrder set orderNumber=?,version=? where id=? and version=? <div style="border: 1px solid green; padding: 5px; margin-left: 200px;"> <p>@Version property is incremented on every DB write</p> <p>Checked for consistency on every update</p> </div>	

6. [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.

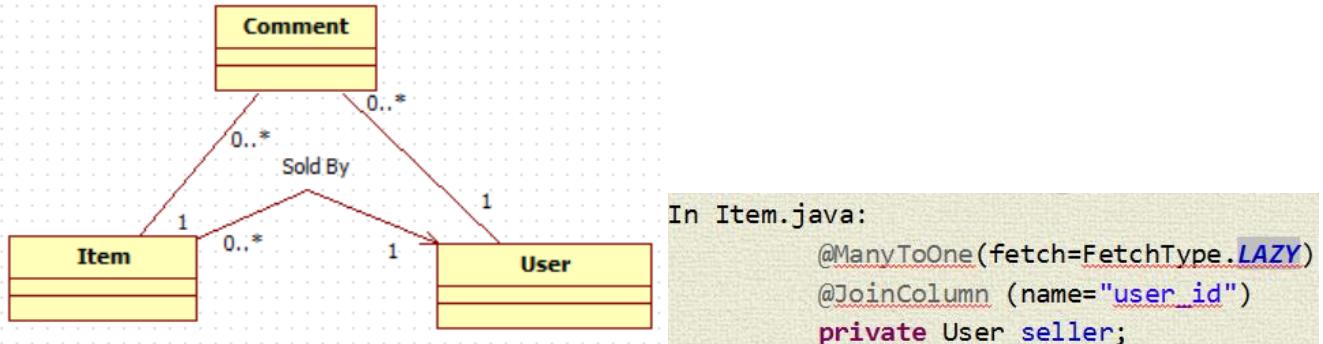
For example:

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]:



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

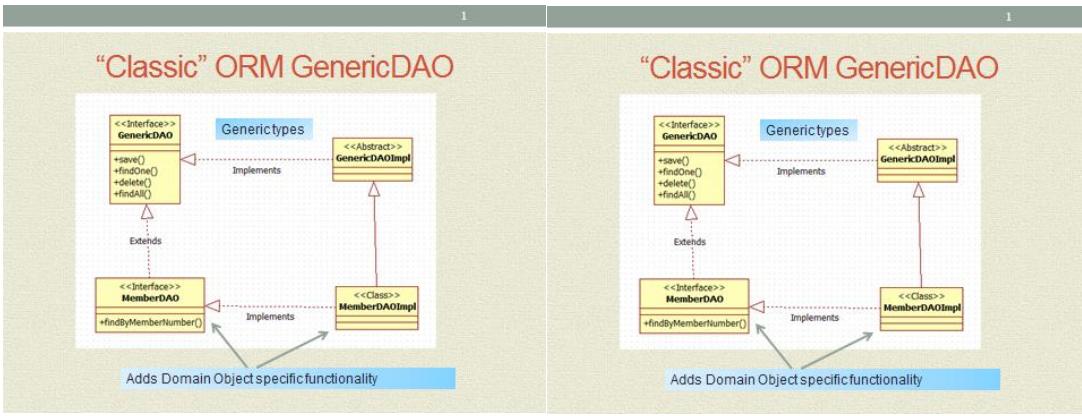
```

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

edu.mum.service.impl.UserService

```

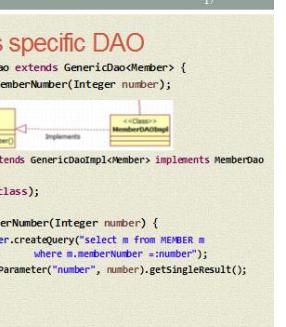
public User findBySoldItemInitialPrice(String email, String itemName, BigDecimal initialPrice) {
    return userDao.findBySoldItemInitialPrice(email, itemName, initialPrice);
}
  
```



16

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 deleted( T entity ) {  
        entityManager.remove( entity );  
    }  
}
```



1

Domain Class specific DAO

```

public interface MemberDao<? extends Member> {
    public Member findByNameMemberNumber(Integer number);
}

public class MemberDaoImpl<? extends Member> implements MemberDao {
    private MemberDAO memberDAO;
    ...
    public Member findByNameMemberNumber(Integer number) {
        ...
        Query query = entityManager.createQuery("select m from MEMBER m
                                                where m.memberNumber = :number");
        ...
        return (Member) query.setParameter("number", number).getSingleResult();
    }
}

```

CS544
Enterprise Architecture
Midterm May 2017

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 Transaction Management:

T F Every interaction with an RDBMS requires a transaction whether a READ or a WRITE. Without a Transaction Management capability like Spring's, DB operations would fail.

EXPLAIN: RDBMS have a built-in transaction capability. The advantage Spring TM provides is capability to manage across multiple DB interaction

T F Spring Transaction Management is based on a logical unit of work.

EXPLAIN: Spans one or MORE DB requests AND “Atomic” across those requests...Either ALL or None

T F Spring Transaction Management with JPA requires a Persistence Context.

EXPLAIN: Persistence Context is needed. It establishes the DB Connection & maintains a cache for “DB-aware” objects.

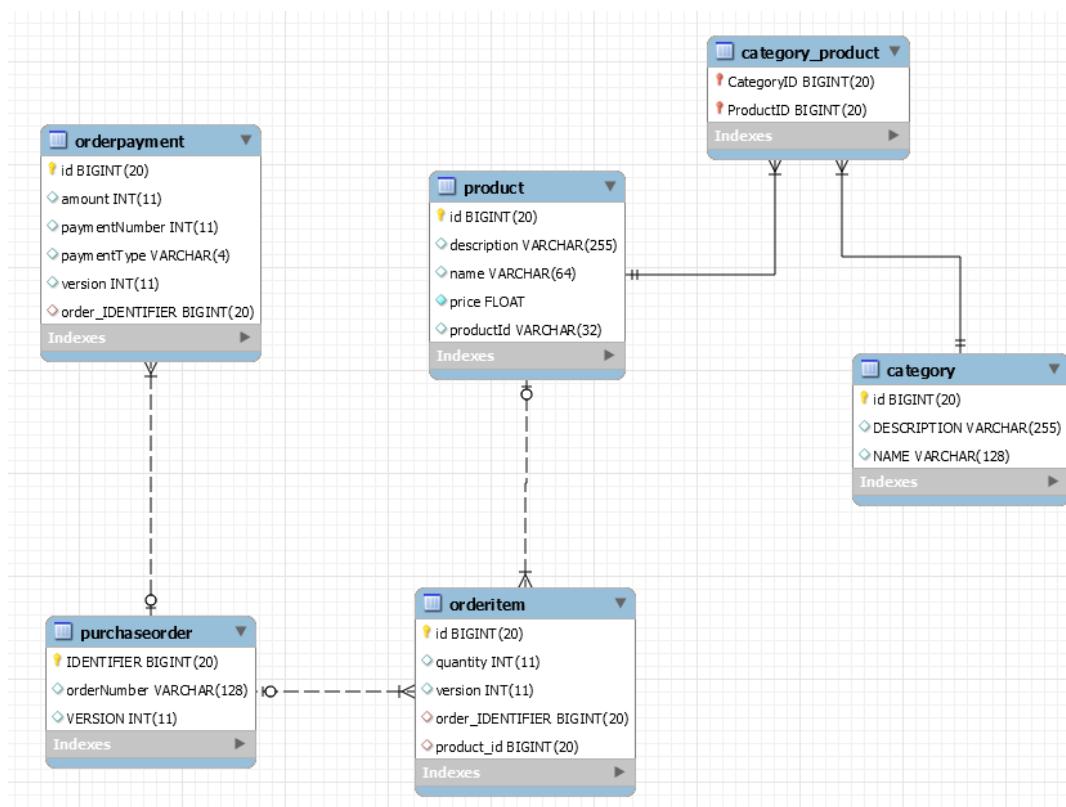
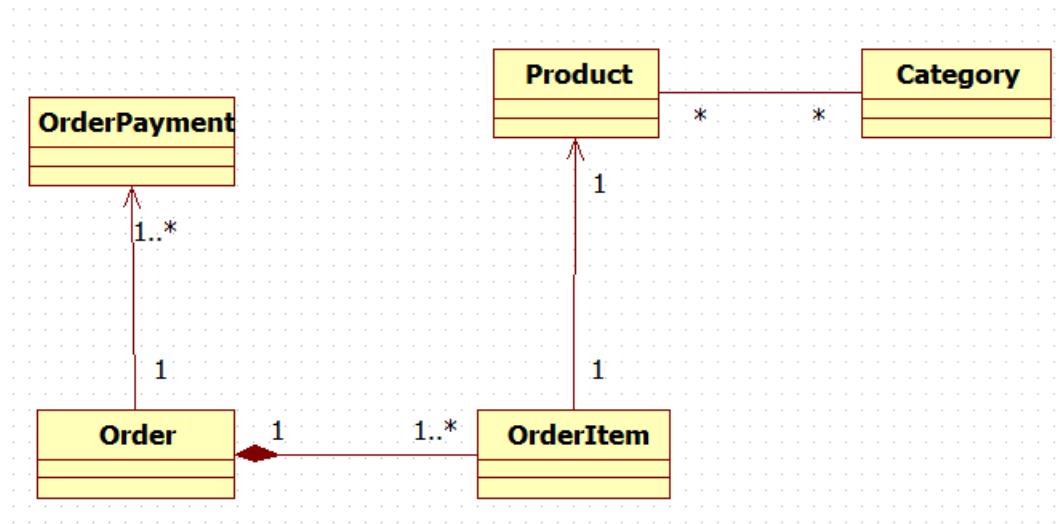
T F Spring @Transaction has no built-in metadata for managing any of the DB ACID properties.

EXPLAIN: @Transactional has an optional parameter for indicating isolation levels. For example -
@Transactional (*isolation=Isolation.READ_COMMITTED*)

T F Spring Declarative Transaction Management requires little or no application code related to transaction management.

EXPLAIN: Spring Declarative TM has little impact on application code...Can simply annotate a class with
@Transactional & the Spring framework takes care of the details.

2. [20 points] Annotate the Domain Objects based on the Domain Model and Entity Relationship Diagram provided. NOTE: All the fields are not listed. Only annotate the fields that are listed.



Product.java

```
20  @Entity
21  public class Product implements Serializable {
22      private static final long serialVersionUID = 5784L;
23
24@     @Id
25     @GeneratedValue(strategy=GenerationType.AUTO)
26     private long id;
27@     @Column(length=64)
28     private String name;
29@     @Column
30     private String description;
31@     @Column(length=32)
32     private String productId;
33@     @Column
34     private float price;
35
36@     @ManyToMany(mappedBy="products",fetch = FetchType.EAGER, cascade = CascadeType.ALL)
37     private Set<Category> categories;
```

Category.java

```
18  @Entity
19  public class Category {
20
21@     @Id
22     @GeneratedValue(strategy=GenerationType.AUTO)
23     private long id;
24
25@     @Column(name="NAME",length=128)
26     String name;
27
28@     @Column(name="DESCRIPTION")
29     String description;
30
31     // If using a List INSTEAD of a SET - less efficient
32@     @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
33     @JoinTable ( name="Category_Product", joinColumns={@JoinColumn(name="CategoryID")}, inverseJoinColumns={@JoinColumn(name="ProductID")})
34     Set<Product> products = new HashSet<Product>();
35
36
```

Order.java

```
21  @Entity
22  @Table(name = "purchaseOrder")
23  public class Order {
24@     @Id
25     @GeneratedValue(strategy = GenerationType.AUTO)
26     @Column(name = "IDENTIFIER", updatable = false, nullable = false)
27     private Long id = null;
28@     @Version
29     @Column(name = "VERSION")
30     private int version = 0;
31
32@     @Column(length=128)
33     private String orderNumber;
34
35     @OneToMany(mappedBy = "order", fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
36     private Set<OrderItem> items = new HashSet<OrderItem>();
37
38     //mappedBy = "order",
39     @OneToOne( fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
40     @JoinColumn(name="order_IDENTIFIER")
41     private Set<OrderPayment> payments = new HashSet<OrderPayment>();
```

OrderItem.java

```
14 @Entity
15 public class OrderItem {
16
17@   @Id
18   @GeneratedValue(strategy = GenerationType.AUTO)
19   @Column(name = "id", updatable = false, nullable = false)
20   private Long id = null;
21@   @Version
22   @Column(name = "version")
23   private int version = 0;
24
25@   @Column
26   private int quantity;
27
28@   @ManyToOne(fetch = FetchType.EAGER)
29   private Order order;
30
31@   @OneToOne(fetch = FetchType.EAGER, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
32   private Product product;
33
```

OrderPayment.java

```
11 @Entity
12 public class OrderPayment {
13
14@   @Id
15   @GeneratedValue(strategy = GenerationType.AUTO)
16   @Column(name = "id", updatable = false, nullable = false)
17   private Long id = null;
18@   @Version
19   @Column(name = "version")
20   private int version = 0;
21
22@   @Column
23   private Integer paymentNumber;
24
25@   @Column(length = 4)
26   private String paymentType;
27
28@   @Column
29   private Integer amount;
30
```

3. [15 points] The reason for an ORM is because object models and relational models do not work very well together. Describe what is known as the Object-Relational Impedance mismatch. Give specific examples of the problems that arise from the mismatch.

2

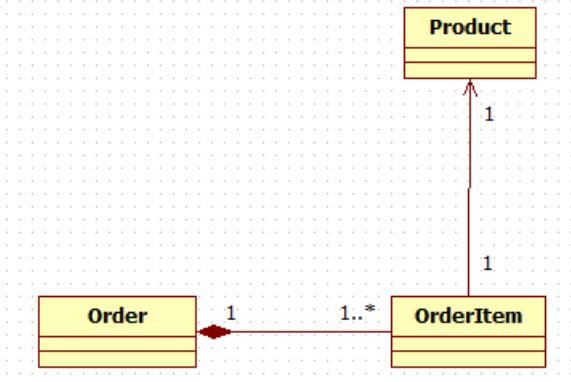
ORM Impedance Mismatch

2 Different Technologies – 2 different ways to operate

EXAMPLE

- **OO traverse objects through relationships**
 - Category category = product.getCategory();
- **RDB join the data rows of tables**
 - SELECT c.* FROM product p,category c where p.category_id = c.id;
- **OTHERS:**
 - Many-to-many relationships
 - Inheritance
 - Collections
 - Identity [Primary Key .vs. a.equals(b)]
 - Foreign Keys
 - Bidirectional ["Set both sides"]
 - Granularity [# of Tables .vs. # of Classes]

4. [15 points] For the following relationships implement a Join fetch of all Orders with their Order Item collection.



What performance problem does the Join fetch address? Give details.

What performance problem does it cause? Give details.

What can be done to “clean up” the data returned by the fetch?

In OrderDaoImpl.java

```
23 public List<Order> findAllJoinFetch() {  
24     Query query = entityManager.createQuery("SELECT o FROM Order AS o JOIN FETCH o.items AS i");  
25     List<Order> orders = query.getResultList();  
26     return orders;  
27 }  
28 }
```

Join Fetch does ONE fetch for ALL collections.

The Join Fetch will get ALL the Orders AND OrderItems in ONE Select/fetch.

It solves the N+1 issue.

However it suffers from the Cartesian product issue.

Hibernate Fetch Strategy Issues

- SubSelect**
 - depends on the “parent” query. If parent Query is complex, it could have performance impacts.
 - If `fetch=FetchType.LAZY` need to “hydrate” children
- BatchSize**
 - # of Fetches “unknown” UNLESS size of parent is constant
 - Batch fetching is often called a blind-guess optimization
 - If `fetch=FetchType.LAZY` need to “hydrate” children
- Select**
 - N+1 TBA [To Be Avoided]
- Join**
 - Cartesian – need to watch collection sizes; can be useful strategy

N+1 Problem

Member has a OneToMany relationship with Address

```

    Hibernate:
    select
        member_.member_id as member_1,
        member_.member_id as member_2,
        member_.member_id as member_3,
        member_.member_id as member_4,
        member_.member_id as member_5,
        member_.member_id as member_6,
        member_.member_id as member_7,
        member_.member_id as member_8,
        member_.member_id as member_9,
        member_.member_id as member_10,
        member_.member_id as member_11,
        member_.member_id as member_12
    from
        Member member_
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    Hibernate:
    select
        address_.member_id as member_1,
        address_.id as address_2,
        address_.city as city_3,
        address_.state as state_4,
        address_.street as street_5,
        address_.zipCode as zipCode_6
    from
        Address address_
    where
        address_.member_id = ?
    
```

Declared as Fetch EAGER:

- `@OneToMany(mappedBy="member", fetch=FetchType.EAGER)`
- `private Set<Address> addresses = new HashSet<Address>();`

For

- `entityManager.createQuery("from Member") .getResultSet();`

ORM will issue ONE fetch for the Member &

And N fetches; one for each Set of Addresses

Cartesian Product Problem

- For sets A and B, the Cartesian product is $A \times B$
 - For sets A,B and C, the Cartesian product is $A \times B \times C - e$
 - Member [STILL] has a OneToMany relationship with Address
 - NOW - Declared as Fetch LAZY:
 - `@OneToMany(mappedBy="member", fetch=FetchType.LAZY)`
 - `private Set<Address> addresses = new HashSet<Address>();`
 - For:
 - Query `query=entityManager.createQuery("SELECT m
FROM Member AS m JOIN FETCH m.addresses AS a");`
 - ORM will do ONE Fetch BUT will generate duplicates
 - # Members x # Addresses [per Member]
 - ORM NOTE: Product -

2 Members
X
#Address/Member

卷之三

inc

Addresses so 2 copies 2; Bill has 3 so 3 copies

“Reduce” the Cartesian Product

- ```
- Query query=entityManager.createQuery("SELECT DISTINCT m
 FROM Member AS m JOIN FETCH m.addresses AS a");
```

- DISTINCT keyword removes duplicates

However

It accomplishes it in Memory [After DB fetch]

```

Hibernate:
select
 *
from
 member
 member._member_id as memb_0_1,
 member._age as age_2_0,
 member._firstname as firstnam_3_0,
 member._lastname as lastnam_4_0,
 member._membernumber as membernum_5_0,
 member._title as title_6_0,
 address,
 address._member_id as member_14_6,
 address._state as state_0_1,
 address._zip as zipcod_0_1,
 address._member_id as member_15_6,
 address._id as id_0_0
from
 Member member,
 inner join
 addres address,
 on member._member_id=address.
Member Name: Sam Smith
Address: 123 Main Street
Address: Red Rock Town
Member Name: Bill Doe
Address: Washington D.C.
Address: Jones
Address: Paris, Iow
Address: Jones

```

## 5. [15 points] Explain the concept of ORM caching. Include a discussion of :

- First level relate to Persistence Context; Fetch Strategy
- Second level
  - Read-only - read-write
  - Second-level .vs. query
  - When do you decide to use a second level cache?

Be specific. Give examples. Diagrams are good.

**ORM caching mechanisms**

Persistence provider manages local store of entity data  
Leverages performance by avoiding expensive database calls

CRUD operation can be performed through normal entity manager functions  
Application can remain oblivious of the underlying cache and do its job without concern

**Level 1 Cache**  
Available within the same transaction [Persistence Context]

**Level 2 Cache**  
Available throughout the application.

**Level 1 Cache**

**Second Level Cache**

**Query for Entity**

```

 - Check First level Cache
 - If Found:
 Return Entity
 - If not Found:
 Check Second level Cache
 If Found:
 Update First Level Cache
 Return Entity
 If not Found:
 Execute DB Query
 Update Second Level Cache
 Update First Level Cache
 Return Entity

```

**READ Only – Low Hanging Fruit**

Read-only caches are easy to handle  
It is immutable (Modification Forbidden )  
No consistency issues.  
Always a good candidate for second level caching

Read-write caches are more "subtle" in their behavior  
Interaction with the Hibernate session can lead to unwanted behavior.  
The benefits of the C in ACID are compromised if cache is out of sync with DB  
Eventual consistency is the "purview" of NoSQL DBs NOT Relational DBs.

**Second Level Cache Decision**

**Database queries slow?**  
**Second Level Cache is the final resort**

**Optimize ORM Queries**  
Make sure the fetching strategy is properly designed,  
Remove N+1 query problems  
Involve the DBA [Expert]  
Employ indexes  
Investigate data base solutions[e.g. partitioning]

If performance is still an issue THEN consider a second level cache

**Two Types of Cache**

- Second level cache is a key-value store.  
Applicable for accessing entities by Primary Key  
These will hit Cache
 

```

member = memberService.findOne(1L);
entityManager.createQuery("SELECT m FROM Member m where
m.id= :member");

```
- These will NOT hit cache
 

```

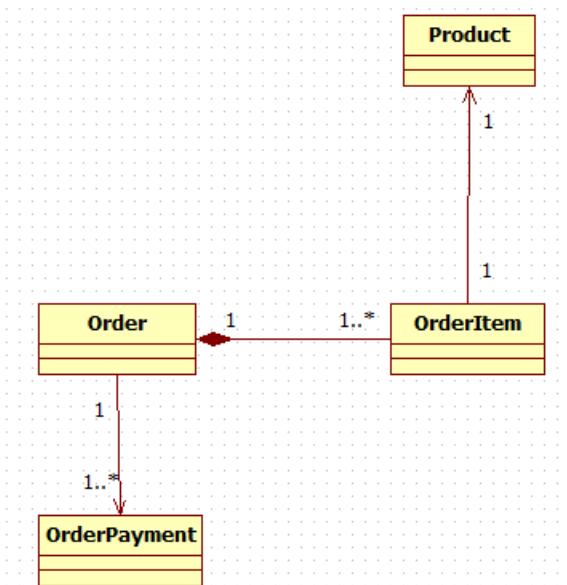
entityManager.createQuery("SELECT m FROM Member AS m JOIN FETCH
m.addresses AS a where m.id= :member");
entityManager.createQuery("select m from Member m where
m.memberNumber =:number");

```
- Query Cache  
Applicable for accessing entities by specific query  
The two preceding queries are Query Cache "candidates"  
NOTE: Query cache store query in query cache and the actually entities in the data cache

6. [15 points] Implement a parameterized JQPL query with this signature:

```
public List<Product> findByAmountRangeAndQuantity(Integer minPayment,
 Integer maxPayment,
 Integer quantity)
```

The query looks up all Product[s] where the Order Item quantity is greater than the supplied quantity and the Order Payment Amount is within the supplied parameters.



The Query should be a parameterized query. Also show the modifications to all classes in order to adhere to the N-Tier architecture convention. Identify the specific packages that each modified class is in.

#### **edu.mum.dao. ProductDao**

```
public List<Product> findByAmountRangeAndQuantity(Integer minPayment, Integer maxPayment, Integer quantity)
```

#### **edu.mum.dao.impl. ProductDaoImpl**

```

52 public List<Product> findByAmountRangeAndQuantity(Integer minPayment, Integer maxPayment, Integer quantity) {

53 Query query = entityManager.createQuery("select p from Product p,OrderItem oi,OrderPayment op where p = p and "

54 + " oi.quantity > :quantity and op member of oi.order.payments " "

55 + " and op.amount > :minPayment and op.amount < :maxPayment") ;

56

57 return (List<Product>) query.setParameter("maxPayment", maxPayment)

58 .setParameter("minPayment", minPayment)

59 .setParameter("quantity", quantity)

60 .getResultList();

```

#### **edu.mum.service. ProductService**

```
public List<Product> findByAmountRangeAndQuantity(Integer minPayment, Integer maxPayment, Integer quantity)
```

#### **edu.mum.service.impl. ProductService Impl**

```
public List<Product> findByAmountRangeAndQuantity(Integer minPayment, Integer maxPayment, Integer quantity) {

 return productDao.findByAmountRangeAndQuantity(minPayment, maxPayment, quantity);

}
```

```
52 public List<Product> findByAmountRangeAndQuantity(Integer minPayment, Integer maxPayment, Integer quantity) {
53 Query query = entityManager.createQuery("select p from Product p,OrderItem oi,OrderPayment op where oi.product =p and "
54 + " oi.quantity > :quantity and op member of oi.order.payments "
55 + " and op.amount > :minPayment and op.amount < :maxPayment") ;
56
57 return (List<Product>) query.setParameter("maxPayment", maxPayment)
58 .setParameter("minPayment", minPayment)
59 .setParameter("quantity", quantity)
60 .getResultList();
61
62 /* + " oi.quantity > :quantity and op in (select opay from oi.order.payments opay "
63 + " where opay.amount > :minPayment and opay.amount < :maxPayment") ;
64 */
65 //This works also
66 // + " oi.quantity > :quantity and op in (select opay from oi.order.payments opay) "
67 // + " and op.amount > :minPayment and op.amount < :maxPayment");
68
```

**CS544**  
**Enterprise Architecture**  
**Midterm December 2016**

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** 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: \_\_\_ If a POJO or JavaBean can be a Spring Managed bean so 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

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

EXPLAIN: \_\_\_ IoC involves a “look up the dependency” step before injecting it. Spring has declarative configuration that identifies “where to find” the resource to inject.

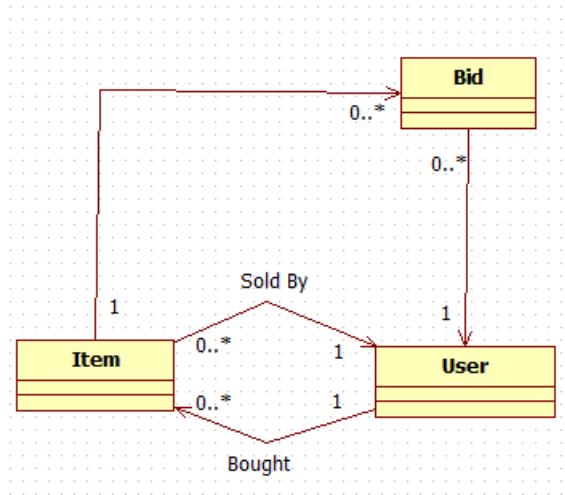
**T F** A domain object needed in a @Service class is usually a good candidate for Dependency Injection

EXPLAIN: \_\_\_ DI is mainly used for “cross” layer transitions, access plumbing resources, etc. NOT for passing business data around...

**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, then you should use it, exclusively for all beans.

EXPLAIN: \_\_\_ You could inject a DAO[ memberDAO in memberService] through XML & inject another DAO[productDAO in productService] through @Autowired. Remember, XML configuration is the final word – if you configure the same bean twice, the XML configuration will take precedence.

2. [15 points] For the following relationships implement a SubSelect that fetches all items with their corresponding collection of bids.



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

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.

## In Item.java

```

public class Item {
 @OneToOne(mappedBy = "item", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
 @Fetch(FetchMode.SUBSELECT)
 private List<Bid> bids = new ArrayList<Bid>();

```

## In ItemServiceImpl.java

```

public List<Item> findBySubSelect() {
 List<Item> items = (List<Item>)this.findAll();
 // hydrate since LAZY load
 items.get(0).getBids().get(0);

 return items;
}

```

Subselect does ONE fetch for the Items and ONE fetch for ALL collections. Therefore the number of Items & Bids in the collections does not matter,

It solves the Cartesian issue && the N+1 issue.

If the collection is LAZY loaded, then doing a get all for the “parent” entities [Item as in above] will get the “parent”/Item list. Within the same transaction, a reference to ONE of the Bid collections will fetch ALL the collections.

The Join Fetch, on the other hand will get ALL the Bids AND Items in ONE Select/fetch.

The Join Fetch, however suffers from the Cartesian product issue [ItemsXBids] which means that more than one copy of each Item/Bid pair will be included in the fetch. The number of copies depends on the number of Bids in the collection. So if Item A has 3 bids, 3 copies of Item A will be present.

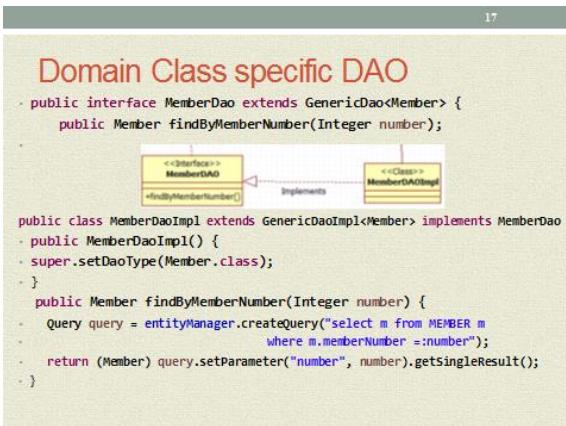
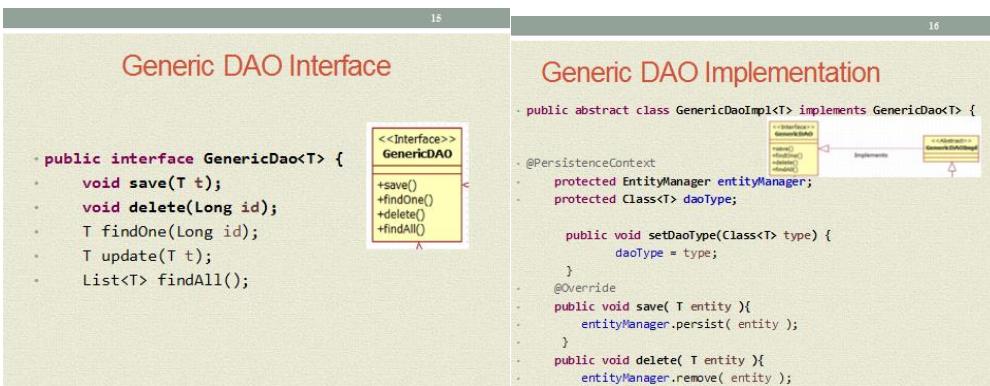
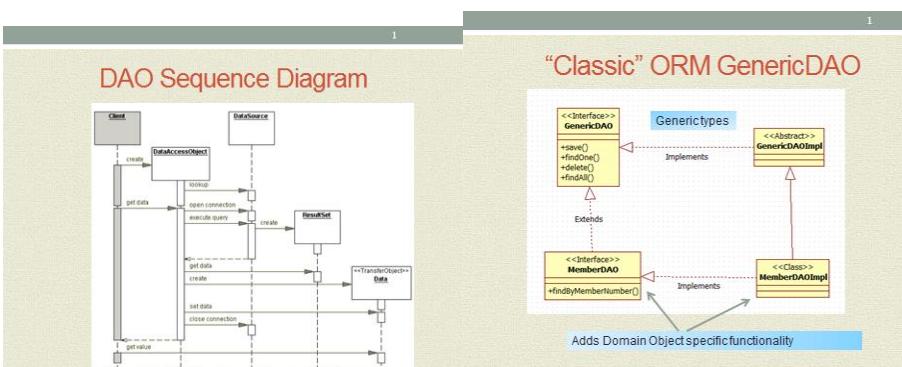
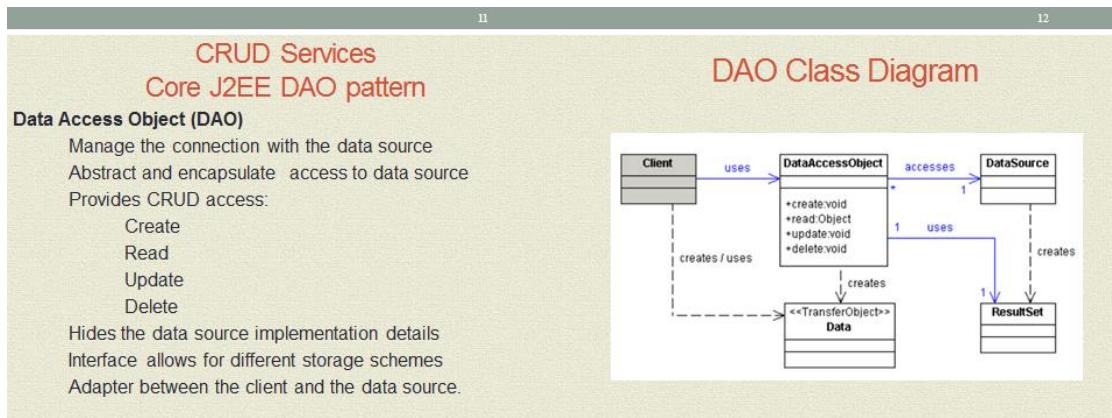
The screenshot shows two slides side-by-side. The left slide, titled 'Hibernate [Default] FETCH Strategy' (slide 4), lists four types of fetching:

- Select fetching:** a second SELECT (per parent N) is used to retrieve the associated collection. `[DEFAULT] [N+1 Fetches]@Fetch(FetchMode.SELECT)`
- Join fetching:** associated collections are retrieved in the same SELECT, using an OUTER JOIN. `[ 1 Fetch] [EAGER] @Fetch(FetchMode.JOIN)`
- Subselect fetching:** a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. `[2 Fetches]`
- Batch fetching:** Optimization of Select Fetching. Associated collections are fetched according to declared Batch Size(NBatch Size) + 1; `@BatchSize(size=n)`

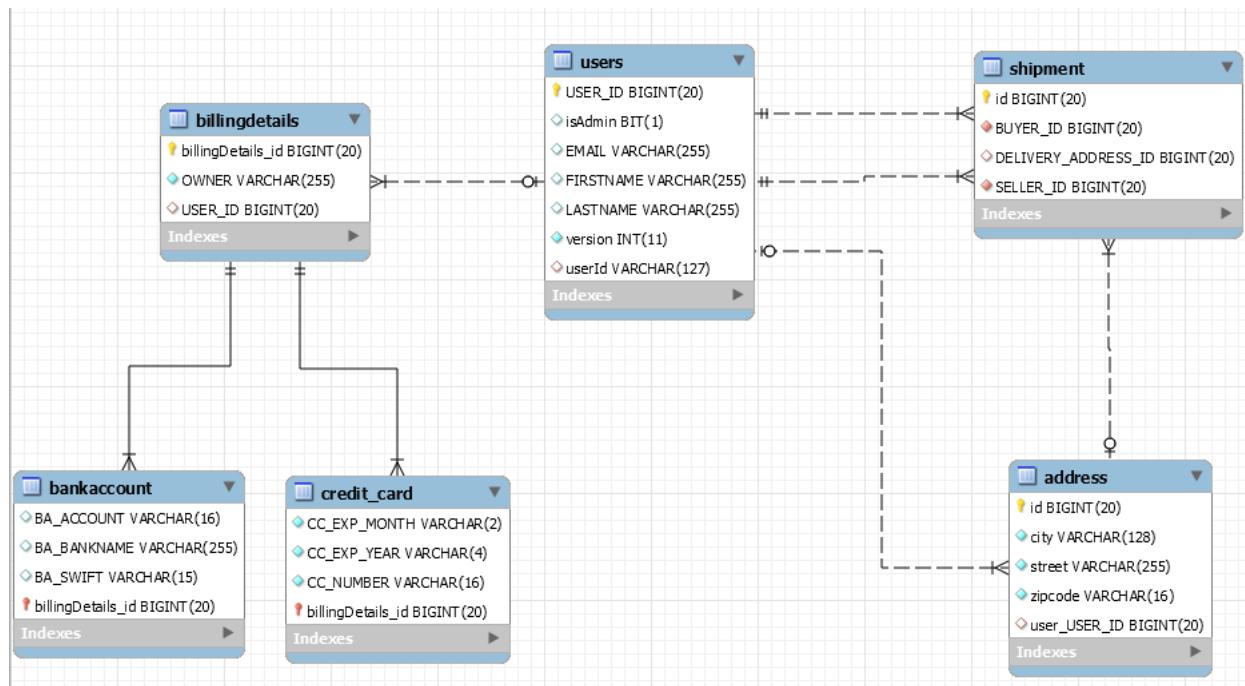
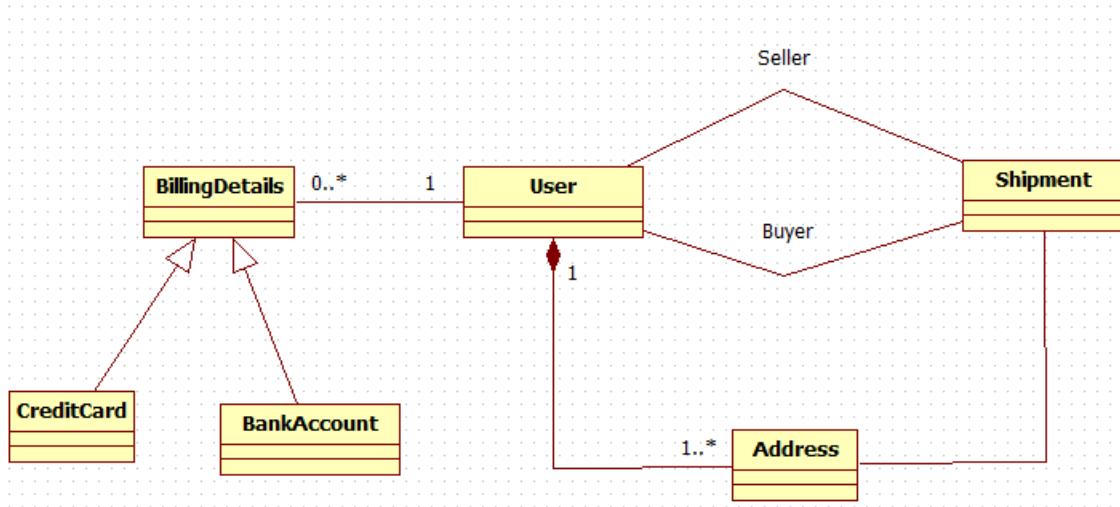
The right slide, titled 'Hibernate Fetch Strategy Issues' (slide 5), lists several issues:

- SubSelect**: depends on the "parent" query. If parent Query is complex, it could have performance impacts.  
If `fetch=FetchType.LAZY` need to "hydrate" children
- BatchSize**: # of Fetches "unknown" UNLESS size of parent is constant  
Batch fetching is often called a blind-guess optimization  
If `fetch=FetchType.LAZY` need to "hydrate" children
- Select**:
  - N+1 TBA [To Be Avoided]
- Join**:
  - Cartesian – need to watch collection sizes; can be useful strategy

3. [15 points] The Core J2EE DAO pattern is fundamental to a well-organized ORM application. Explain the pattern, what is for, how it works. Include in the explanation, the sequence of interactions between the user, a DAO and the database. Include an explanation [& UML diagram] of the Generic DAO design. Be specific, give examples.



4. [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.



```

22 @Entity
23 @Table(name = "USERS")
24 public class User implements Serializable {
25
26 @Id @GeneratedValue(strategy=GenerationType.AUTO)
27 @Column(name = "USER_ID")
28 private Long id = null;
29
30 @Column(name = "FIRSTNAME")
31 private String firstName;
32
33 @Column(name = "LASTNAME")
34 private String lastName;
35
36 @Column(name = "EMAIL")
37 private String email;
38
39 @Column(name = "isAdmin")
40 private boolean admin = false;
41
42 @OneToOne(fetch=FetchType.EAGER, cascade = CascadeType.ALL)
43 @JoinColumn(name="userId")
44 private UserCredentials userCredentials;
45
46 @OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.PERSIST, mappedBy="user")
47 private Set<Address> addresses = new HashSet<Address>();
48
49 @OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.PERSIST, mappedBy="buyer")
50 private Set<Shipment> buyShipments = new HashSet<Shipment>();
51
52 @OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.PERSIST, mappedBy="seller")
53 private Set<Shipment> sellShipments = new HashSet<Shipment>();
54
55 @OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.PERSIST, mappedBy="user")
56 private Set<BillingDetails> billingDetails = new HashSet<BillingDetails>();
57
58 }

6 @Entity
7 @Table(name = "SHIPMENT")
8 public class Shipment {
9
10 @Id @GeneratedValue
11 private Long id = null;
12
13 @ManyToOne(fetch = FetchType.EAGER)
14 @JoinColumn(name="DELIVERY_ADDRESS_ID")
15 private Address deliveryAddress;
16
17 @ManyToOne(fetch = FetchType.EAGER)
18 @JoinColumn(name="BUYER_ID")
19 private User buyer;
20
21 @ManyToOne(fetch = FetchType.EAGER)
22 @JoinColumn(name="SELLER_ID")
23 private User seller;

```

```

25 @Entity
26 public class Address implements Serializable {
27
28 @Id
29 @GeneratedValue(strategy=GenerationType.AUTO)
30 private Long id = null;
31
32 @Column(length = 255)
33 private String street;
34
35 @Column(length = 16)
36 private String zipcode;
37
38 @Column(length = 128)
39 private String city;
40
41 @ManyToOne(fetch=FetchType.LAZY)
42 private User user;
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2296
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
23
```

5. [10 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.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Lock Mode for Data Consistency</b></p> <ul style="list-style-type: none"> <li>Locking refers to actions taken to prevent data in a relational database from changing between the time it is read and the time that it is used.</li> </ul> <p><b>Locking strategies</b></p> <ul style="list-style-type: none"> <li>Optimistic Lock           <ul style="list-style-type: none"> <li>Concurrent transactions can complete without affecting each other,</li> <li>Transactions do not need to lock the data resources that they affect.</li> </ul> </li> <li>Pessimistic Lock           <ul style="list-style-type: none"> <li>Concurrent transactions will conflict with each other</li> <li>Transactions require that data is locked when read and unlocked at commit.</li> </ul> </li> </ul> | <p><b>Version-Based Optimistic Concurrency[Locking]</b></p> <ul style="list-style-type: none"> <li>High-volume systems</li> <li>No Connection maintained to the Database [Detached Objects]</li> <li>Effective in <b>Read-Often Write-Sometimes</b> scenario</li> <li>Uses <b>read committed isolation level</b> <ul style="list-style-type: none"> <li>Guarantees <b>repeatable read isolation level</b></li> <li>An exception is thrown if a conflict occurs</li> </ul> </li> <li><b>@Version Long version:</b></li> <li>When entity is updated - version field is incremented.</li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Version-Based Optimistic Concurrency</b></p> <ul style="list-style-type: none"> <li>Optimistic concurrency assumes that lost update conflicts generally don't occur           <ul style="list-style-type: none"> <li>Keeps version #s so that it knows when they do</li> <li>Guarantees best performance and scalability</li> <li>The default way to deal with concurrency</li> <li>There is no locking anywhere</li> <li>It works well with very long conversations, including those that span multiple transactions</li> </ul> </li> <li>First commit wins instead of last commit wins           <ul style="list-style-type: none"> <li>An exception is thrown if a conflict would occur               <ul style="list-style-type: none"> <li>ObjectOptimisticLockingFailureException</li> </ul> </li> </ul> </li> </ul> | <p><b>Optimistic Locking [ Cont.]</b></p> <ul style="list-style-type: none"> <li>It works well with long conversations, including those that span multiple transactions</li> <li><b>LONG CONVERSATION Use Case:</b> <ul style="list-style-type: none"> <li>A multi-step dialog, for example a wizard dialog interacts with the user in several request/response cycles.</li> </ul> </li> <li><b>session-per-request-with-detached-objects</b> <ul style="list-style-type: none"> <li>@Version Locking manages Detached object consistency</li> </ul> </li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <p><b>JPA Optimistic Locking</b><br/>JPA version-based concurrency control</p> <p>Simply add <b>@Version</b> to an Integer field – JPA takes care of the rest</p> <ul style="list-style-type: none"> <li>@Id</li> <li>@GeneratedValue(strategy = GenerationType.AUTO)</li> <li>private Long id = null;</li> <li>@Version</li> <li>@Column(name = "version")</li> <li>private int version = 0;</li> <li><b>Hibernate:</b> <ul style="list-style-type: none"> <li>update purchaseOrder</li> <li>set orderNumber=? ,version=?</li> <li>where id=? and version=?</li> </ul> </li> </ul> | <p><b>@Version</b> property is incremented on every DB write</p> <p>Checked for consistency on every update</p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|

6. [15 points] Implement a JQPL query that looks up a User by email who bought an Item with a shipping address that has a specific zip code. [Reference UML in problem #4]  
For instance:

**Find User who has an email address of JohnDoe@mail.com who bought an Item that was shipped to a zip code equal to 52556**

**OR**

**Find User who has an email address of JBean@post.com who bought an Item that was shipped to a zip code equal to 12345.**

The Query should be a parameterized query. Also identify all the classes in the specific packages that need to be modified to adhere to the N-Tier architecture convention.

## ANSWER:

**edu.mum.dao.** UserDao

```
public User findByBoughtItemShippedZip(String email, String zipCode);
```

**edu.mum.dao.impl.** UserDaoImpl

```
public User findByBoughtItemShippedZip(String email, String zipCode) {

 Query query = entityManager.createQuery("select u from User u, Shipment s
 where u.email = :email and s.buyer = u "
 + " and s.deliveryAddress.zipcode = :zipCode");
 return (User) query.setParameter("email", email)
 .setParameter("zipCode", zipCode).getSingleResult();
}
```

**edu.mum.service.** UserService

```
public User findByBoughtItemShippedZip(String email, String zipCode);
```

**edu.mum.service.impl.** UserServiceImpl

```
public User findByBoughtItemShippedZip(String email, String zipCode) {
 return userDao.findByBoughtItemShippedZip(email, zipCode);
}
```

**CS544**  
**Enterprise Architecture**  
**Midterm July 2017**

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 ORM technologies:

**T F** An example of impedance mismatch is the fact that a RDB puts information in rows and an OO language puts information in Objects

**EXPLAIN:** \_\_\_ A RDB consists of individual columns which represent the fields/properties of an object.  
This requires some way to BIND the DB data to the Object & make sure that the data types match.

**T F** A good use case for using an ORM is complex interactions between entities

**EXPLAIN:** \_\_\_ This is a major advantage of an ORM. The ORM “automatically” maps these relationships, reducing boiler plate code.

**T F** The value of a good ORM is that it automatically takes care of all the issues relating to a RDB

**EXPLAIN:** \_\_\_ It is NOT possible for an ORM to do everything. It covers mapping and CRUD services. However there are situations where “manual” intervention is necessary

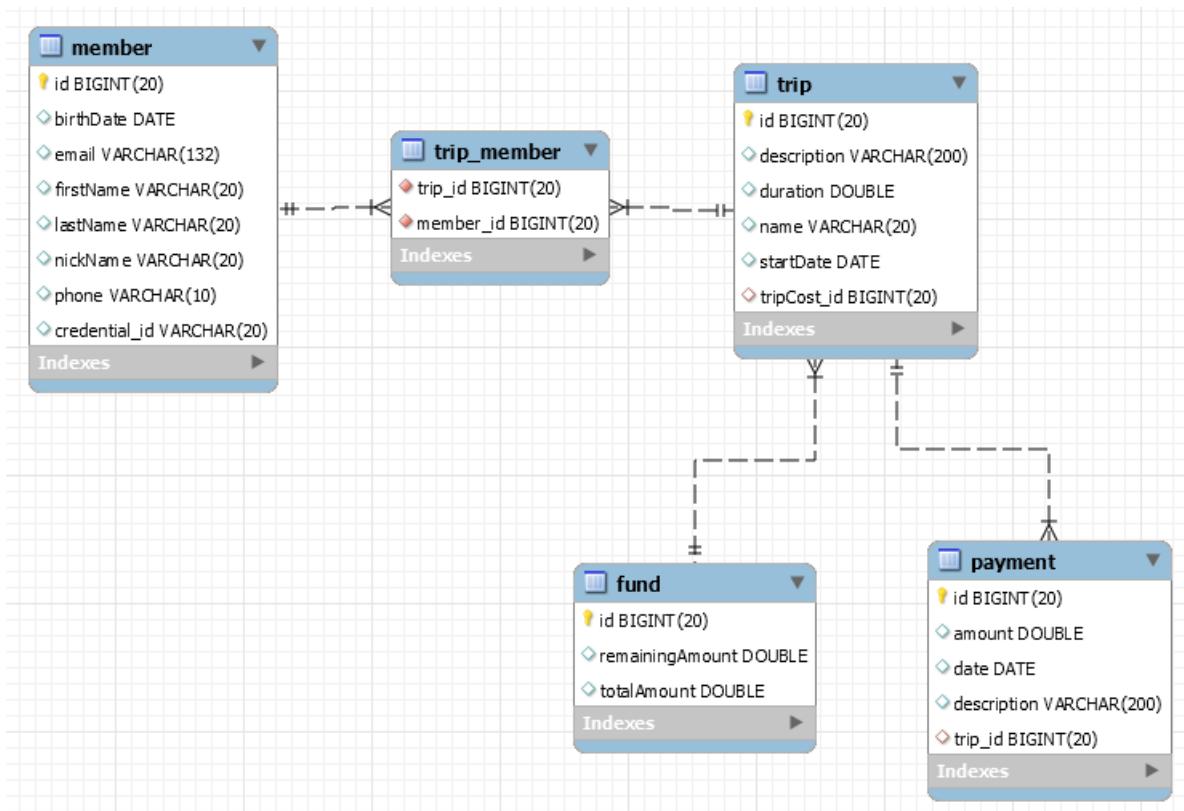
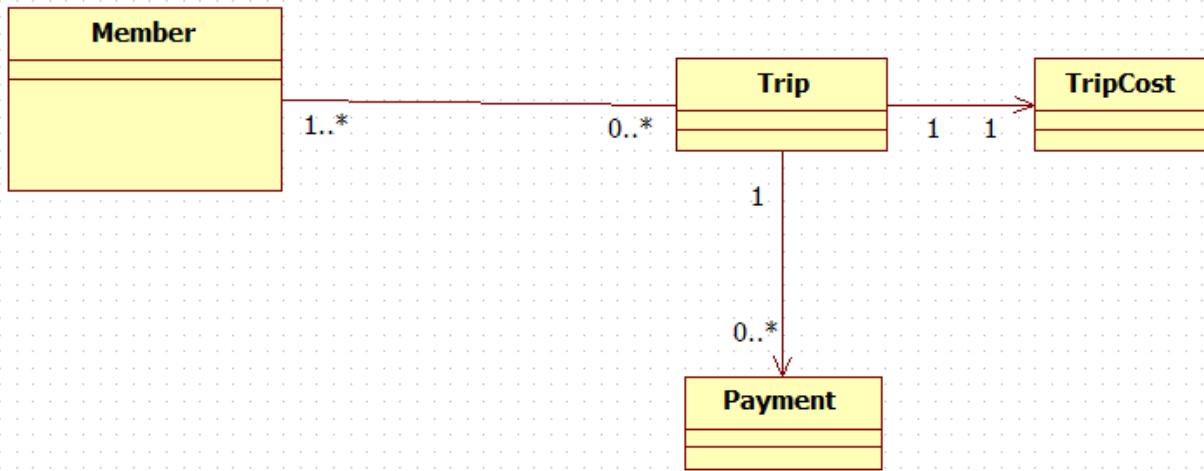
**T F** JPA is an industry standard for ORMs and has made Hibernate obsolete.

**EXPLAIN:** \_\_\_ JPA is an industry standard but does not replace Hibernate. In fact, Hibernate implements JPA As JPA is “only” an API.

**T F** Native SQL Queries are supported by a good ORM solution and are recommended as the first choice way to access entity relationships.

**EXPLAIN:** \_\_\_ Native queries are fallback mechanism to be used when queries are complex and cannot be adequately implemented in JPQL.

2. [20 points] Annotate the Domain Objects based on the Domain Model and Entity Relationship Diagram provided. NOTE: All the fields are not listed. Only annotate the fields that are listed.



## Trip.java

```
30 @Entity
31 public class Trip {
32
33 @Id
34 @GeneratedValue(strategy = GenerationType.AUTO)
35 private Long id;
36
37 @Column(length = 20)
38 private String name;
39
40 @Column(length = 200)
41 private String description;
42
43 @Column
44 private Double duration;
45
46 @Temporal(TemporalType.DATE)
47 private Date startDate;
48
49 @Transient
50 private Date endDate;
51
52 @ManyToMany(cascade = { CascadeType.PERSIST, CascadeType.MERGE })
53 @JoinTable(name = "trip_member", joinColumns = { @JoinColumn(name = "trip_id") }, inverseJoinColumns = {
54 @JoinColumn(name = "member_id") })
55 List<Member> members = new ArrayList<>();
56
57 @OneToOne(fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REMOVE })
58 @JoinColumn(name="fund_id")
59 Fund fund;
60
61 @OneToMany(fetch = FetchType.EAGER, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
62 @JoinColumn(name="trip_id")
63 List<Payment> payments = new ArrayList<>();
```

## TripCost.java

```
15 @Entity(name="TripCost")
16 public class Fund {
17
18 @Id
19 @GeneratedValue(strategy = GenerationType.AUTO)
20 private long id;
21
22 private Double remainingAmount;
23
24 private Double totalAmount;
```

## Payment.java

```
21 @Entity
22 public class Payment {
23
24 @Id
25 @GeneratedValue(strategy = GenerationType.AUTO)
26 private long id;
27
28 @Column(length = 200)
29 private String description;
30
31 @Column
32 private Double amount;
33
34 @Temporal(TemporalType.DATE)
35 private Date date;
```

## Member.java

```
27 @Entity
28 public class Member {
29
30 @Id
31 @GeneratedValue(strategy = GenerationType.AUTO)
32 private Long id;
33
34 @Column(length = 20)
35 private String firstName;
36
37 @Column(length = 20)
38 private String lastName;
39
40 @Column(length = 20)
41 private String nickName;
42
43 @Transient
44 private Gender gender;
45
46 @Column(length = 132)
47 private String email;
48
49 @Column(length = 10)
50 private String phone;
51
52 @Temporal(TemporalType.DATE)
53 private Date birthDate;
54
55 @ManyToMany(cascade= {CascadeType.PERSIST,CascadeType.MERGE}, mappedBy="members")
56 List<Trip> trips;
```

3. [15 points] Transaction management is an important part of RDBMS oriented enterprise applications. Spring provides core functionality to assist in transaction management. Describe the Spring transaction functionality, how it is implemented, how it facilitates ORM Transaction management. Include an explanation on how it supports the RDBMS ACID properties of Consistency and Isolation. Be specific. Give Examples.

ANSWER:

| <h3>ORM – RDB Interactions</h3> <pre> graph TD     PC[Persistence Context] --&gt; T[Transaction]     T --&gt; PO[Persistent Objects]     PO --&gt; DO[Detached Objects]     subgraph Transaction [ ]         direction TB         T         PO     end     subgraph PersistenceContext [Persistence Context]         direction TB         PC         PO     end     subgraph DetachedObjects [Detached Objects]         direction TB         DO     end </pre> <p><b>Persistence Context</b></p> <ul style="list-style-type: none"> <li>Establishes DB connection</li> <li>Holds DB-aware objects</li> </ul> <p><b>Transaction</b></p> <ul style="list-style-type: none"> <li>Logical unit of work</li> <li>One or more DB queries</li> <li>Atomic [ All or None ]</li> </ul> <p><b>Detached Objects</b></p> | <p>21</p> <h3>ORM Persistence Context [Hibernate: session]</h3> <p>Transaction Unit c Spring “manages” through @Transactional</p> <p>Common Pattern: <b>session-per-request</b></p> <p>Persistence Context == Database Transaction</p> <pre> @Service @Transactional public class ProductServiceImpl implements ProductService {     Open PersistenceContext/Start Transaction when method is called     public Product getProductById(Long productID) {         return productDao.findOne(productID);     }     End Transaction/ Close PersistenceContext when method is exited } </pre> <p>END –</p> <p>End Transaction</p> <p>Close a Persistence Context</p>                                                                                                                      |                                      |                               |           |                 |                           |          |         |                                      |         |                   |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|-------------------------------|-----------|-----------------|---------------------------|----------|---------|--------------------------------------|---------|-------------------|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>Spring ORM Support</h3> <p><i>Comprehensive transaction support is among the most compelling reasons to use the Spring Framework.</i></p> <ul style="list-style-type: none"> <li>Integration with Hibernate, Java Persistence API (JPA)...</li> <li><b>Hibernate Support</b> <ul style="list-style-type: none"> <li>First-class integration support through IoC/DI</li> <li>Easier testing</li> <li>Resource management</li> <li>Integrated transaction management</li> </ul> </li> </ul> <p><a href="#">Spring Framework Data Access</a></p>                                                                                                                                                                                                                                                            | <p>21</p> <h3>Hibernate Spring Managed Transactions</h3> <p>..On Service Layer</p> <pre> @Service @Transactional → Starts Transaction public class VehicleServiceImpl implements edu.mum.service.VehicleService {     ...     public void save( Vehicle vehicle ) {         vehicleDao.save(vehicle);     }      public abstract class GenericDaoImpl&lt;T&gt; implements GenericDao&lt;T&gt; {         ...         protected Session getSession() {             return sessionFactory.getCurrentSession();         }         public void save( T entity ) {             this.getSession().save(entity);         }     } } </pre> <p>Reduction in code*: manage transaction open/close session</p> <p>* Compared to Hibernate Solo SEE <a href="#">HibernateTransactions DEMO</a></p> |                                      |                               |           |                 |                           |          |         |                                      |         |                   |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <h3>Declarative Transaction Attributes</h3> <table border="1"> <thead> <tr> <th>Propagation</th> <th>enum: Propagation</th> <th>Optional propagation setting.</th> </tr> </thead> <tbody> <tr> <td>Isolation</td> <td>enum: Isolation</td> <td>Optional isolation level.</td> </tr> <tr> <td>ReadOnly</td> <td>boolean</td> <td>Read/write vs. read-only transaction</td> </tr> <tr> <td>TimeOut</td> <td>Integer [seconds]</td> <td>Max Transaction time</td> </tr> </tbody> </table> <pre> @Service @Transactional(propagation=Propagation.REQUIRES_NEW,                isolation=Isolation.READ_COMMITTED) public class ReadCommittedServiceImpl     implements edu.mum.service.ReadCommittedService { } </pre>                                                                                           | Propagation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | enum: Propagation                    | Optional propagation setting. | Isolation | enum: Isolation | Optional isolation level. | ReadOnly | boolean | Read/write vs. read-only transaction | TimeOut | Integer [seconds] | Max Transaction time | <h3>ACID Database properties</h3> <p>Set of properties that guarantee that database transactions are processed reliably.</p> <p>Atomicity and Durability are strict properties, i.e., Black or White, All or None</p> <p><b>ATOMIC:</b> The transaction is considered a single unit, either the entire transaction completes, or the entire transaction fails.</p> <p><b>CONSISTENT:</b> A transaction transforms the database from one consistent state to another consistent state</p> <p><b>ISOLATED:</b> Data inside a transaction can not be changed by another concurrent processes until the transaction has been committed</p> <p><b>DURABLE:</b> Once committed, the changes made by a transaction are persistent</p> <p>Consistency and Isolation are “configurable” &amp; Interdependent</p> |
| Propagation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | enum: Propagation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Optional propagation setting.        |                               |           |                 |                           |          |         |                                      |         |                   |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Isolation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | enum: Isolation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Optional isolation level.            |                               |           |                 |                           |          |         |                                      |         |                   |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| ReadOnly                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | boolean                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Read/write vs. read-only transaction |                               |           |                 |                           |          |         |                                      |         |                   |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| TimeOut                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Integer [seconds]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Max Transaction time                 |                               |           |                 |                           |          |         |                                      |         |                   |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

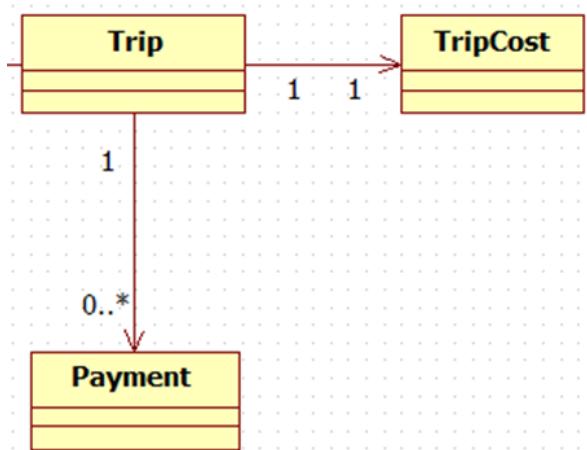
## Isolation in a Relational Database

- The challenge is to maximize concurrent transactions, while maintaining consistency
  - The shorter the lock acquisition interval, the more requests a database can process.
- Isolation Levels:**
- SERIALIZABLE (NO dirty, non-repeatable OR phantom reads)
  - REPEATABLE READ (NO dirty OR non-repeatable reads)
    - READ COMMITTED (NO dirty reads)
    - READ UNCOMMITTED (ANYTHING Goes)
- Most databases default to READ COMMITTED

## Isolation Types

|                  | dirty reads | non-repeatable reads | phantom reads |
|------------------|-------------|----------------------|---------------|
| READ_UNCOMMITTED | yes         | yes                  | yes           |
| READ_COMMITTED   | no          | yes                  | yes           |
| REPEATABLE_READ  | no          | no                   | yes           |
| SERIALIZABLE     | no          | no                   | no            |

4. [15 points] For the following relationships implement a Batch fetch of all Trips with their Payments collection. Assume the Payment collection is fetch LAZILY.



What performance problem[s] does the batch fetch address?

How does it work? – Explain the “algorithm” based on a universe of 20 Trips each with a collection of 5-10 Payments.

One fetch for ALL the Trips PLUS N Payment collection fetches where N is based on batch Size & # of Trips.

For example, 20 Trips with batch size = 2 results in 10 collection fetches.

For example, 20 Trips with batch size = 3 results in 7 collection fetches. [6 fetches of 3 PLUS 1 fetch of 2].

For example, 20 Trips with batch size = 4 results in 5 collection fetches, etc...

## In TripServiceImpl.java

```

52 public List<Trip> findAllBatch() {
53 List<Trip> trips = (List<Trip>)this.findAll();
54 // hydrate - need to access ALL since we don't know batch Size
55 // e.g. if size =2 AND there are 20 trips we need to hydrate trips #1 & #3 & #5 ... & #19
56 for (Trip trip : trips)
57 if (!trip.getPayments().isEmpty()) trip.getPayments().get(0);
58
59 return trips;
60 }
61
62 }
```

## In Trip.java

```

61 @OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
62 @JoinColumn(name="trip_id")
63 @org.hibernate.annotations.Fetch(org.hibernate.annotations.FetchMode.SELECT)
64 @org.hibernate.annotations.BatchSize(size = 2)
65 List<Payment> payments = new ArrayList<>();
```

Solves N+1.. AND Cartesian Product BUT # of collection fetches is “unknown”...

## Hibernate Fetch Strategy Issues

### SubSelect

depends on the "parent" query. If parent Query is complex, it could have performance impacts.

If fetch=FetchType.LAZY need to "hydrate" children

### BatchSize

# of Fetches "unknown" UNLESS size of parent is constant

Batch fetching is often called a blind-guess optimization

If fetch=FetchType.LAZY need to "hydrate" children

### Select

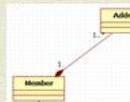
N+1 TBA [To Be Avoided]

### Join

Cartesian – need to watch collection sizes; can be useful strategy

## N+1 Problem

- Member has a OneToMany relationship with Address



- Declared as Fetch EAGER:

@OneToMany(mappedBy="member", fetch=FetchType.EAGER)

private Set<Address> addresses = new HashSet<Address>();

- For

entityManager.createQuery("from Member")

.getResultSet();

ORM will issue ONE fetch for the Member

&

And N fetches; one for each Set of Addresses

### Example N = 3

```
NetBeans
Hibernate:
select
 member0_.member_id as member_14_3_,
 addresses0_.id as id1_0_,
 addresses0_.member_id as member_id2_0_,
 addresses0_.city as city2_0_,
 addresses0_.state as state2_0_,
 addresses0_.street as street2_0_,
 addresses0_.zipCode as zipCode2_0_
from
 member member0_
inner join
 address addresses0_
 on member0_.member_id = addresses0_.member_id
where
 member0_.member_id = member_14_3_;
```

## Cartesian Product Problem

- For sets A and B, the Cartesian product is  $A \times B$
- For sets A,B and C, the Cartesian product is  $A \times B \times C$  - etc.

Member[STILL]has a OneToMany relationship with Address

NOW - Declared as Fetch LAZY:

@OneToMany(mappedBy="member", fetch=FetchType.LAZY)

private Set<Address> addresses = new HashSet<Address>();

For:

Query query=entityManager.createQuery("SELECT m  
FROM Member AS m JOIN FETCH m.addresses AS a");

ORM will do ONE Fetch BUT will generate duplicates

# Members x # Addresses [per Member]

ORM NOTE: Product =

# of "root" table results

X

# of results in individual "root" table childtable.

Sean has 2 Addresses so 2 copies 2; Bill has 3 so 3 copies

## "Reduce" the Cartesian Product

Query query=entityManager.createQuery("SELECT DISTINCT m  
FROM Member AS m JOIN FETCH m.addresses AS a");

- DISTINCT keyword removes duplicates

However

It accomplishes it in Memory [ After DB fetch]

```
Hibernate:
select
 distinct member0_.member_id as member_id,
 addresses0_.id as id1_0_1,
 member0_.age as age2_0_1,
 member0_.firstName as firstName3_0_1,
 member0_.lastName as lastName4_0_1,
 member0_.memberNumber as memberNumber5_0_1,
 member0_.title as title5_0_1,
 addresses0_.city as city6_0_1,
 addresses0_.state as state6_0_1,
 addresses0_.street as street6_0_1,
 addresses0_.zipCode as zipCode6_0_1,
 addresses0_.id as id1_0_...
```

```
Hibernate:
inner join
 Address addresses0_
 on member0_.member_id=addresses0_.member_id
Member Name: Sean Smith
Address: Batavia Iowa
Member Name: Peat Moss
Member Name: Bill Due
Address: Mexico Iowa
Address: Peat Moss
Member Name: Bill Due
Address: Washington Iowa
Member Name: Peat Moss
Address: Paris Iowa
Address: Mexico Iowa
Address: Paris Iowa
```

## Batch Size Fetch

Declared as Fetch LAZY:

@OneToMany(mappedBy="member", fetch=FetchType.LAZY)

@Fetch(FetchMode.SELECT)

@BatchSize(size = 3)

private Set<Address> addresses = new HashSet<Address>();

Need to "Hydrate" collections:

List<Member> members = (List<Member>)this.findAll()

for (Member member : members)

if (!member.getAddresses().isEmpty())

member.getAddresses().get(0);

In example, ORM will do ONE Collection Fetch

[based on batch size = # parents]

### NOTE:

In example member = 3; BatchSize = 3; 1 Collection fetch

If member = 3; BatchSize = 2; 2 Collection fetches

If member = 3; BatchSize = 1; 3 Collection fetches

5. [15 points] IoC and DI are part of the Spring Core Technologies. Explain in detail what they are and how they work. Explain it in terms of the “Essence of a Spring Application” and the basic components that make up a Spring Application.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2 style="color: red;">Spring Core Technologies</h2> <ul style="list-style-type: none"> <li>- IoC ***<br/>Inversion of Control Container</li> <li>- AOP ***<br/>Aspect-Oriented Programming</li> <li>- SpEL **<br/>Spring Expression Language that supports querying and manipulating an object graph at runtime.</li> <li>- Resource **<br/>Common API that abstracts the type of underlying resource such as a URL, file or class path resource.</li> </ul> | <p style="text-align: right;">★★★</p> <h2 style="color: red;">Inversion of Control [IOC]</h2> <p><b>Objects do not create other objects that they depend on.</b></p> <ul style="list-style-type: none"> <li>- Promotes loose coupling between classes and subsystems</li> <li>- Adds potential flexibility to a codebase for future changes.</li> <li>- Classes are easier to unit test in isolation.</li> <li>- Enable better code reuse.</li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

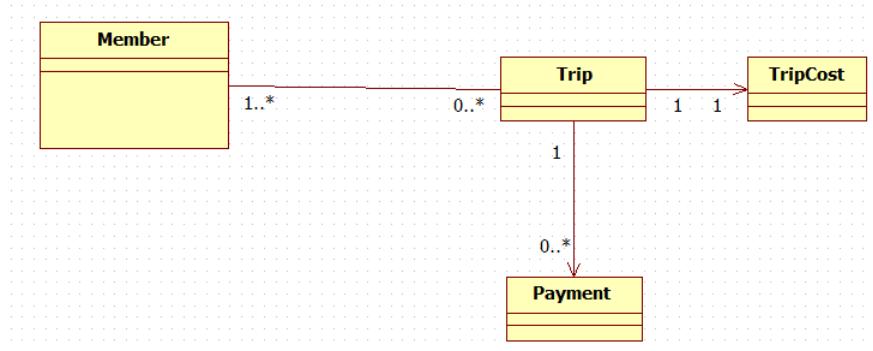
|                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2 style="color: red;">Spring Core – IoC Container</h2> <h3 style="color: black;">The Essence of a Spring Application</h3> <pre> graph TD     BO[Business Object POJO Classes] --&gt; IOC[Spring IOC Container]     CM[Configuration Metadata] --&gt; IOC     IOC --&gt; FCA[Fully Configured Application for use]   </pre> | <h2 style="color: red;">Spring Core - CORE</h2> <p>The <b>HEART</b> of the Spring Framework is the <b>Spring Inversion Of Control [IOC] **</b> Container</p> <p>The IOC container is used to Manage &amp; Configure <b>Plain Old Java Objects [POJO]</b> Through <b>Interfaces</b></p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2 style="color: red;">JavaBeans .vs. POJO .vs. Spring Bean</h2> <p><b>JavaBean</b></p> <p>Adhere to Sun's JavaBeans specification<br/><a href="#">Spring Documentation</a><br/>Implement "Bean" is used interchangeably with POJO instance<br/>Both mean object instance created from a Java class.<br/>public no arguments constructor</p> <p><b>POJO</b></p> <p>'Fancy' way to describe ordinary Java Objects<br/><a href="#">Spring Documentation</a><br/>D "Component" is used interchangeably with POJO class<br/>Both mean a Java class from which an object instance is created<br/>S.</p> <p><b>Spring Bean</b></p> <p>Spring managed - configured, instantiated and injected</p> <p><b>A Java object can be a JavaBean, a POJO and a Spring bean all at the same time.</b></p> | <h2 style="color: red;">Inversion of Control [IOC]</h2> <p><b>"Hollywood Principle: Don't call us, we'll call you".</b></p> <p>The terms IOC and Dependency Injection [DI] are often used interchangeably.</p> <p>IOC actually refers to:</p> <ul style="list-style-type: none"> <li>Lookup involves a "pull" of the dependency from a resource: e.g. Registry Lookup</li> <li>Dependency Lookup &amp;</li> <li>Dependency Injection</li> <li>For injection, IOC container "pushes" the dependency...</li> </ul> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

6. [15 points] Implement a parameterized JQPL query with this signature:

```
public Member findByEmailAndTotalCost(String email,Double amount)
```

The query looks up all Members[s] by email that has a Trip associated with it that has a Trip Cost greater than the supplied amount value. Refer to Problem #2 for field names.



The Query should be a parameterized query. Also show the modifications to all classes in order to adhere to the N-Tier architecture convention. Identify the specific packages that each modified class is in.

**edu.mum.dao.MemberDao**

```
public Member findByEmailAndTotalCost(String email,Double amount);
```

**edu.mum.dao.impl.MemberDaoImpl**

```
18 public Member findByEmailAndTotalCost(String email,Double amount) {
19
20 Query query = entityManager.createQuery("select m from Member m,Trip t " +
21 " where m.email =:email and t member of m.trips "
22 "+ " and t.fund.totalAmount > :amount");
23 return (Member) query.setParameter("email", email)
24 .setParameter("amount", amount).getSingleResult();
25 }
26
```

**edu.mum.service.MemberService**

```
public Member findByEmailAndTotalCost(String email,Double amount);
```

**edu.mum.service.impl.MemberServiceImpl**

```
public Member findByEmailAndTotalCost(String email,Double amount) {
 return memberDao.findByEmailAndTotalCost(email, amount);
}
```

**CS544**  
**Enterprise Architecture**  
**Midterm July 2017**

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 ORM technologies:

**T F** An example of impedance mismatch is the fact that a RDB puts information in rows and an OO language puts information in Objects

**EXPLAIN:** \_\_\_ A RDB consists of individual columns which represent the fields/properties of an object.  
This requires some way to BIND the DB data to the Object & make sure that the data types match.

**T F** A good use case for using an ORM is complex interactions between entities

**EXPLAIN:** \_\_\_ This is a major advantage of an ORM. A RDB entity-entity relationship uses Foreign keys. The ORM “automatically” maps these relationships, reducing boiler plate code.

**T F** The value of a good ORM is that it automatically takes care of all the issues relating to a RDB

**EXPLAIN:** \_\_\_ It is NOT possible for an ORM to do everything. It covers mapping and CRUD services. However there are situations where “manual” intervention is necessary [e.g. custom SQL queries]

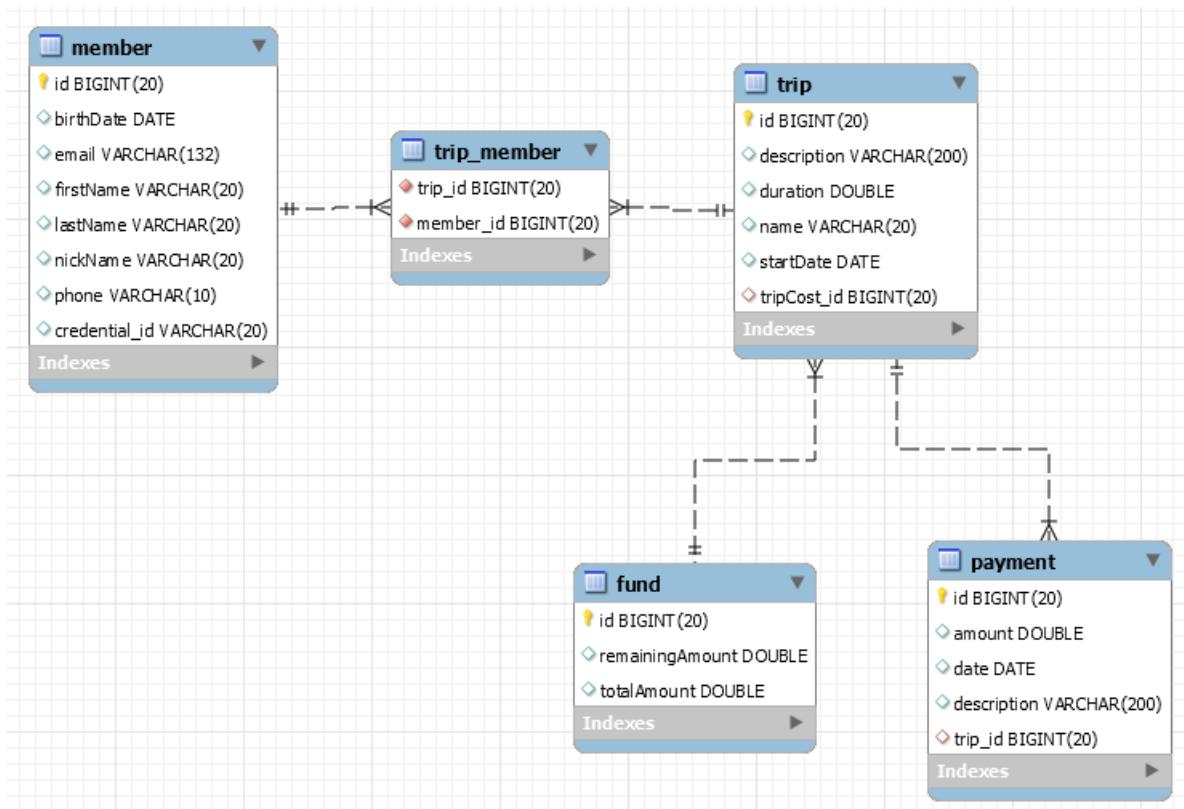
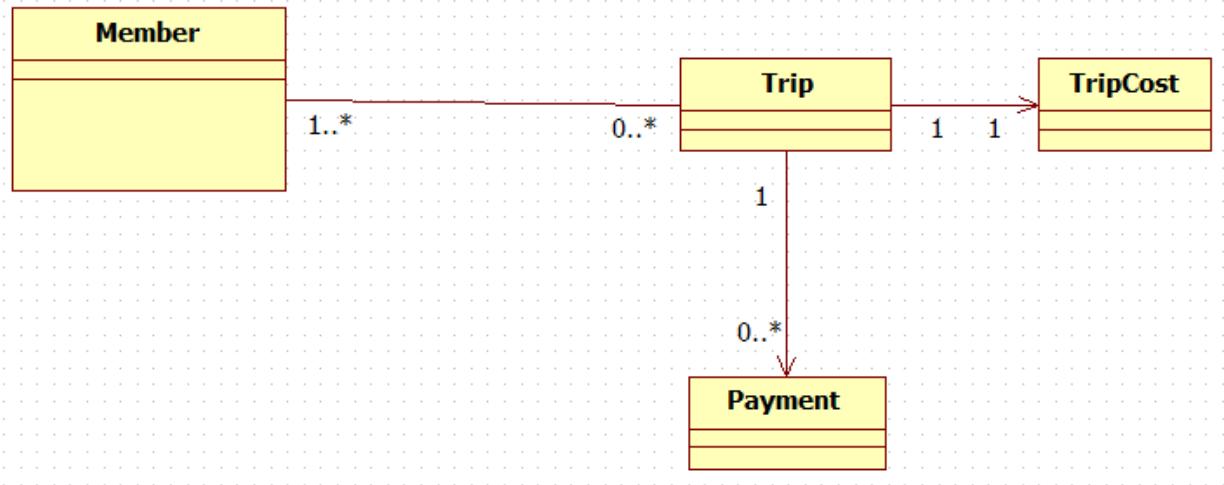
**T F** JPA is an industry standard for ORMs and has made Hibernate obsolete.

**EXPLAIN:** \_\_\_ JPA is an industry standard but does not replace Hibernate. In fact, Hibernate implements JPA As JPA is “only” an API.

**T F** Native SQL Queries are supported by a good ORM solution and are recommended as the first choice way to access entity relationships.

**EXPLAIN:** \_\_\_ Native queries are fallback mechanism to be used when queries are complex and cannot be adequately implemented in JPQL.

2. [20 points] Annotate the Domain Objects based on the Domain Model and Entity Relationship Diagram provided. NOTE: All the fields are not listed. Only annotate the fields that are listed.



## Trip.java

```
30 @Entity
31 public class Trip {
32
33 @Id
34 @GeneratedValue(strategy = GenerationType.AUTO)
35 private Long id;
36
37 @Column(length = 20)
38 private String name;
39
40 @Column(length = 200)
41 private String description;
42
43 @Column
44 private Double duration;
45
46 @Temporal(TemporalType.DATE)
47 private Date startDate;
48
49 @Transient
50 private Date endDate;
51
52 @ManyToMany(cascade = { CascadeType.PERSIST, CascadeType.MERGE })
53 @JoinTable(name = "trip_member", joinColumns = { @JoinColumn(name = "trip_id") }, inverseJoinColumns = {
54 @JoinColumn(name = "member_id") })
55 List<Member> members = new ArrayList<>();
56
57 @OneToOne(fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REMOVE })
58 @JoinColumn(name="tripCost_id")
59 TripCost tripCost;
60
61 @OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
62 @JoinColumn(name="trip_id")
63 @org.hibernate.annotations.Fetch(org.hibernate.annotations.FetchMode.SELECT)
64 @org.hibernate.annotations.BatchSize(size = 1)
65 List<Payment> payments = new ArrayList<>();
66
```

## TripCost.java

```
16 @Entity(name="Fund")
17 public class TripCost {
18
19 @Id
20 @GeneratedValue(strategy = GenerationType.AUTO)
21 private long id;
22
23 @Column
24 private Double remainingAmount;
25
26 @Column
27 private Double totalAmount;
28
```

## Payment.java

```
21 @Entity
22 public class Payment {
23
24 @Id
25 @GeneratedValue(strategy = GenerationType.AUTO)
26 private long id;
27
28 @Column(length = 200)
29 private String description;
30
31 @Column
32 private Double amount;
33
34 @Temporal(TemporalType.DATE)
35 private Date date;
```

## Member.java

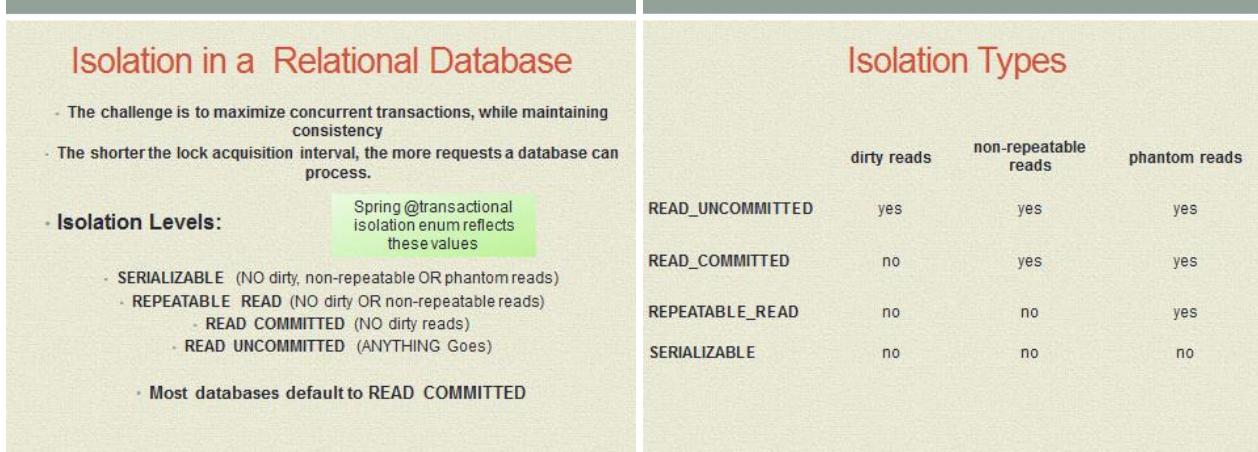
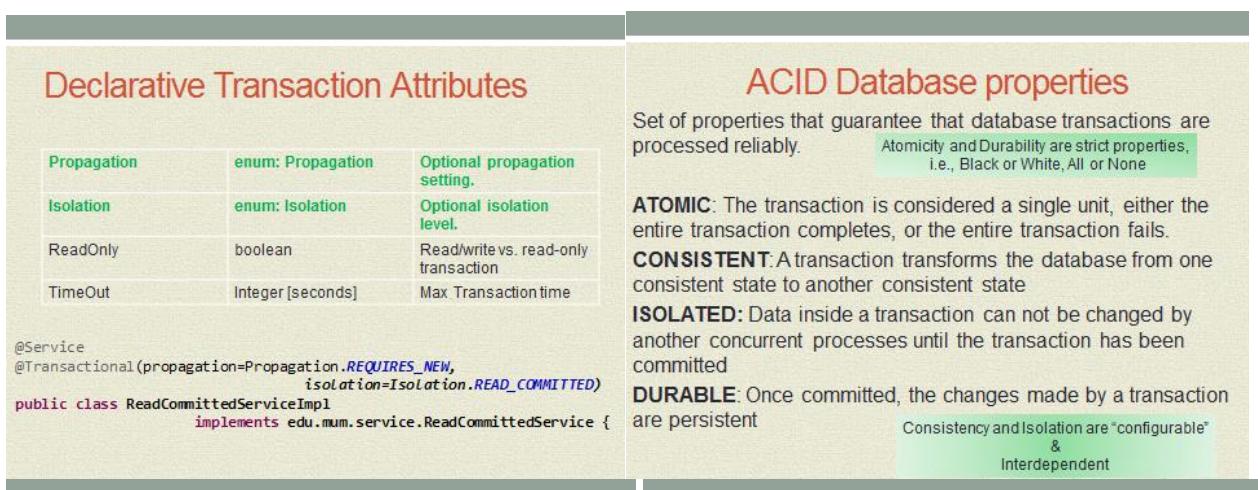
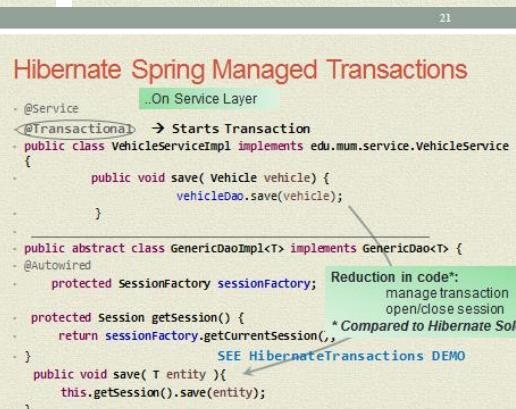
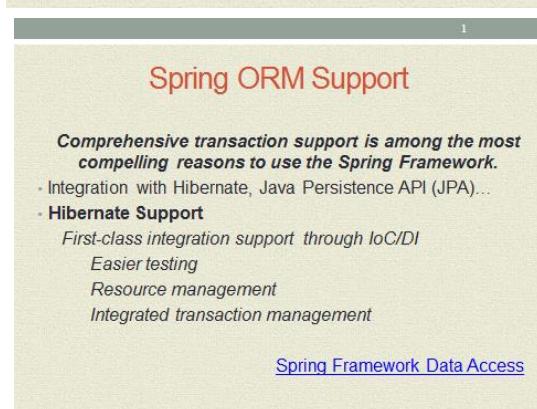
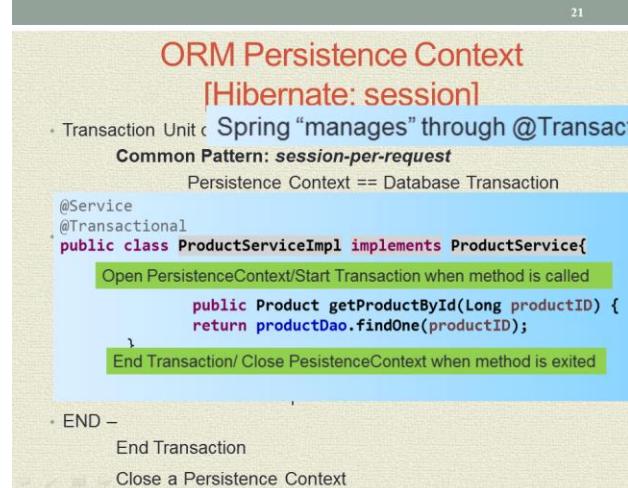
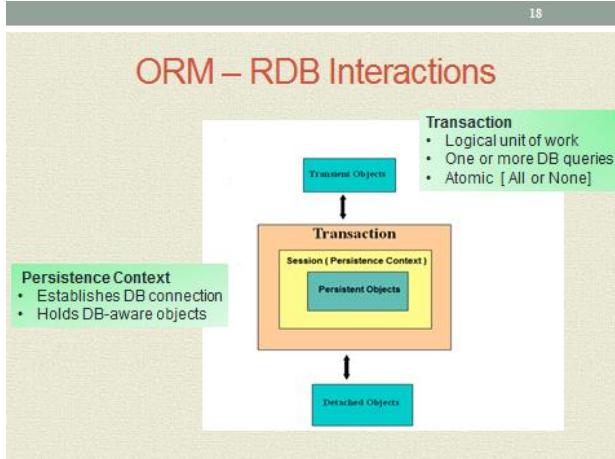
```

27 @Entity
28 public class Member {
29
30 @Id
31 @GeneratedValue(strategy = GenerationType.AUTO)
32 private Long id;
33
34 @Column(length = 20)
35 private String firstName;
36
37 @Column(length = 20)
38 private String lastName;
39
40 @Column(length = 20)
41 private String nickName;
42
43 @Transient
44 private Gender gender;
45
46 @Column(length = 132)
47 private String email;
48
49 @Column(length = 10)
50 private String phone;
51
52 @Temporal(TemporalType.DATE)
53 private Date birthDate;
54
55 @ManyToMany(cascade= {CascadeType.PERSIST,CascadeType.MERGE}, mappedBy="members")
56 List<Trip> trips;

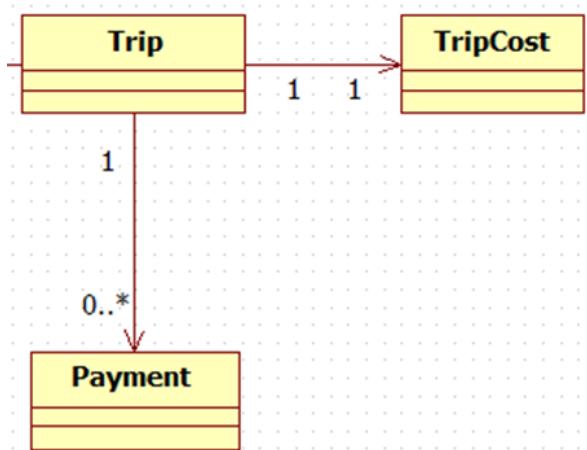
```

3. [15 points] Transaction management is an important part of RDBMS oriented enterprise applications. Spring provides core functionality to assist in transaction management. Describe the Spring transaction functionality, how it is implemented, how it facilitates ORM Transaction management. Include an explanation on how it supports the RDBMS ACID properties of Consistency and Isolation. Be specific. Give Examples.

ANSWER:



4. [15 points] For the following relationships implement a Batch fetch of all Trips with their Payments collection. Assume the Payment collection is fetch LAZILY.



What performance problem[s] does the batch fetch address?

How does it work? – Explain the “algorithm” based on a universe of 20 Trips each with a collection of 5-10 Payments.

One fetch for ALL the Trips PLUS N Payment collection fetches where N is based on batch Size & # of Trips.

For example, 20 Trips with batch size = 2 results in 10 collection fetches.

For example, 20 Trips with batch size = 3 results in 7 collection fetches. [6 fetches of 3 PLUS 1 fetch of 2].

For example, 20 Trips with batch size = 4 results in 5 collection fetches, etc...

## In TripServiceImpl.java

```

52 public List<Trip> findAllBatch() {
53 List<Trip> trips = (List<Trip>)this.findAll();
54 // hydrate - need to access ALL since we don't know batch Size
55 // e.g. if size =2 AND there are 20 trips we need to hydrate trips #1 & #3 & #5 ... & #19
56 for (Trip trip : trips)
57 if (!trip.getPayments().isEmpty()) trip.getPayments().get(0);
58
59 return trips;
60 }
61
62
63
64
65

```

## In Trip.java

```

61 @OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.PERSIST, CascadeType.MERGE })
62 @JoinColumn(name="trip_id")
63 @org.hibernate.annotations.Fetch(org.hibernate.annotations.FetchMode.SELECT)
64 @org.hibernate.annotations.BatchSize(size = 2)
65 List<Payment> payments = new ArrayList<>();

```

Solves N+1.. AND Cartesian Product BUT # of collection fetches is “unknown”...



5. [15 points] IoC and DI are part of the Spring Core Technologies. Explain in detail what they are and how they work. Explain it in terms of the “Essence of a Spring Application” and the basic components that make up a Spring Application.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| <h2 style="color: red;">Spring Core Technologies</h2> <ul style="list-style-type: none"> <li>- IoC ***<br/>Inversion of Control Container</li> <li>- AOP ***<br/>Aspect-Oriented Programming</li> <li>- SpEL **<br/>Spring Expression Language that supports querying and manipulating an object graph at runtime.</li> <li>- Resource **<br/>Common API that abstracts the type of underlying resource such as a URL, file or class path resource.</li> </ul> | <span style="font-size: 2em;">★★★</span> |
| <h2 style="color: red;">Inversion of Control [IOC]</h2> <p><b>Objects do not create other objects that they depend on.</b></p> <ul style="list-style-type: none"> <li>- Promotes loose coupling between classes and subsystems</li> <li>- Adds potential flexibility to a codebase for future changes.</li> <li>- Classes are easier to unit test in isolation.</li> <li>- Enable better code reuse.</li> </ul>                                                |                                          |

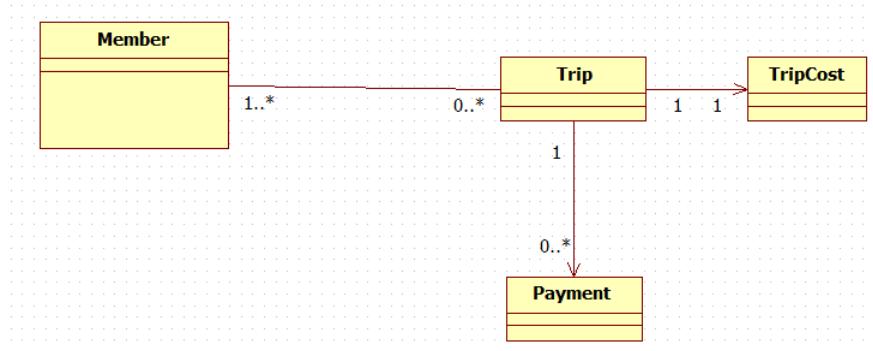
|                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2 style="color: red;">Spring Core – IoC Container</h2> <h3 style="color: black;">The Essence of a Spring Application</h3> <pre> graph TD     BO[Business Object POJO Classes] --&gt; IOC[Spring IOC Container]     CM[Configuration Metadata] --&gt; IOC     IOC --&gt; FCA[Fully Configured Application for use]   </pre> | <h2 style="color: red;">Spring Core - CORE</h2> <p>The <b>HEART</b> of the Spring Framework is the <b>Spring Inversion Of Control [IOC] **</b> Container</p> <p>The IOC container is used to Manage &amp; Configure <b>Plain Old Java Objects [POJO]</b> Through <b>Interfaces</b></p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2 style="color: red;">JavaBeans .vs. POJO .vs. Spring Bean</h2> <p>JavaBean<br/>Adhere to Sun's JavaBeans specification<br/><a href="#">Spring Documentation</a><br/>Implement 'Bean' is used interchangeably with POJO instance<br/>Both mean object instance created from a Java class.<br/>public no arguments constructor</p> <p>Reusable Java classes for visual application composition</p> <p>POJO<br/>'Fancy' way to describe ordinary Java Objects<br/><a href="#">Spring Documentation</a><br/>Component is used interchangeably with POJO class<br/>Both mean a Java class from which an object instance is created<br/>S... S</p> <p>Spring Bean<br/>Spring managed - configured, instantiated and injected</p> <p><b>A Java object can be a JavaBean, a POJO and a Spring bean all at the same time.</b></p> | <h2 style="color: red;">Inversion of Control [IOC]</h2> <p><b>"Hollywood Principle: Don't call us, we'll call you".</b></p> <p>The terms IOC and Dependency Injection [DI] are often used interchangeably.</p> <p>IOC actually refers to:<br/>Lookup involves a "pull" of the dependency from a resource: e.g. Registry Lookup<br/>Dependency Lookup &amp;<br/>Dependency Injection<br/>For injection, IOC container "pushes" the dependency...</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

6. [15 points] Implement a parameterized JQPL query with this signature:

```
public Member findByEmailAndTotalCost(String email,Double amount)
```

The query looks up all Members[s] by email that has a Trip associated with it that has a Trip Cost greater than the supplied amount value. Refer to Problem #2 for field names.



The Query should be a parameterized query. Also show the modifications to all classes in order to adhere to the N-Tier architecture convention. Identify the specific packages that each modified class is in.

#### **edu.mum.dao.MemberDao**

```
public Member findByEmailAndTotalCost(String email,Double amount);
```

#### **edu.mum.dao.impl. MemberDaoImpl**

```

19
public Member findByEmailAndTotalCost(String email,Double amount) {
20
21 Query query = entityManager.createQuery("select m from Member m,Trip t " +
22 " where m.email =:email and t member of m.trips " +
23 " + and t.tripCost.totalAmount > :amount");
24
25 Member member = (Member) query.setParameter("email", email)
26 .setParameter("amount", amount).getSingleResult();
27 return member;
28 }
29
30

```

#### **edu.mum.service. MemberService**

```
public Member findByEmailAndTotalCost(String email,Double amount);
```

#### **edu.mum.service.impl.MemberServiceImpl**

```
public Member findByEmailAndTotalCost(String email,Double amount) {
 return memberDao.findByEmailAndTotalCost(email, amount);
}
```