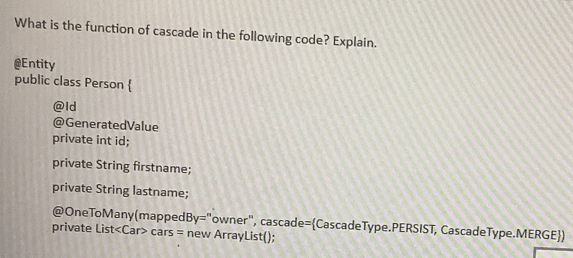
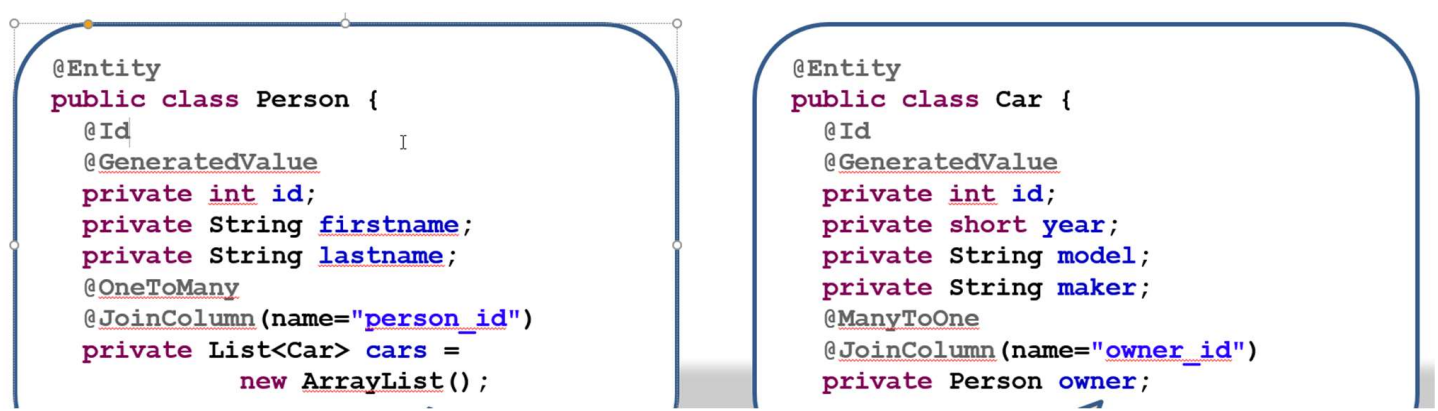
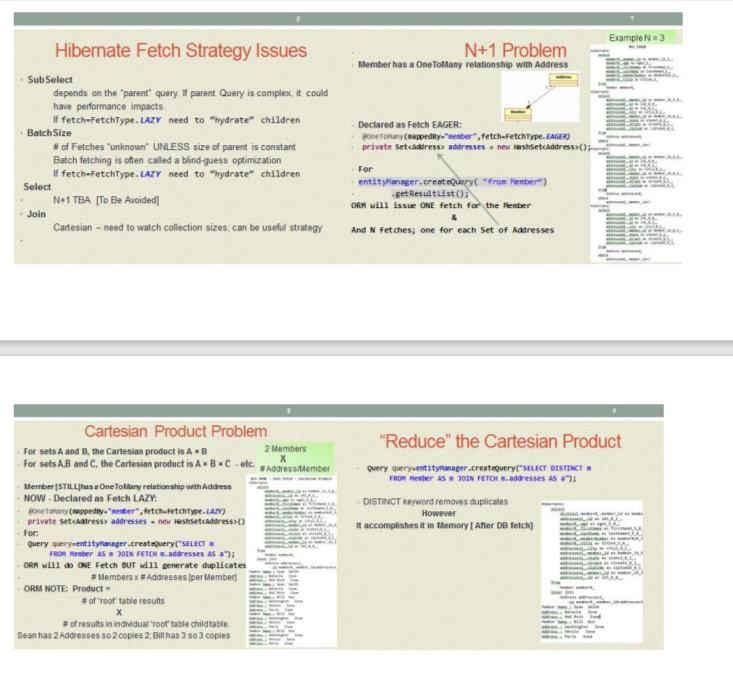
* 
* Solution:
  + The Cascade function commit the association of Person with Cars while saving the Person data the data holding in the Car arrays is also commit. With cascade we do not need to commit the Car separately.
* Question:
  + What would you do if you wanted Hibernate to forcefully push all the cached changes to the underlying DB in the middle of a transaction? IS there a command/method for it?
* Solution:
  + Use the command as: session.flush() to reflect all the changes made in cache in the middle of a transaction.
* List All the phases of Spring lifecycle/initialization?
  + Spring reads the configuration files
  + Spring initializes all the beans and does the constructor dependency injection
  + Spring then does field and proper initialization
  + Spring then executes the @PostConstructor or Init-Method defined in the configuration file
  + Spring will execute other operation
  + Once context is closed or application shut down, spring will call the @PreDestroy or Destroy-Method
* Question
  + In JPA/Hibernate, how can you tell if an entity is mapped using “property” access or “field” access? Can we have a mixed situation?
* Solution:
  + The access type is defined the way we put the @Id annotation in the entity. If @Id is placed before private long Id; then the access mode is going to be the “field” access and if @Id is placed be before getter/setter then access mode is going to be “property” access.
  + Mix access mode is possible on the sub-class entity. By adding an @Access annotation will override the default access type.
* Question:
  + What is the difference between session.get() and session.load()?
* Solution:
  + Session.get() is lazy loading while session.load() is eager loading
* Question:
  + Explain Pessimistic Concurrent?
* Solution:
  + In Perssimistic Concurrent, Hibernate request for the explicit database lock, thus ensuring that other transaction does not alter it. These lock will be released once the Hibernate performs commit operation. The Pessimistic concurrent is aimed at getting rid of un-repeatable read problem that Optimistic concurrent can have. Performance lags in compared to optimistic.
* Question:
  + Explain Optimistic concurrency?
* Solution:
  + Optimistic concurrency assumes that collisions between transactions will rarely occur. For database conflict optimistic concurrency uses the mechanism like maintain version number or maintain time stamp. Locking is generally not done in Optimistic concurrency. If the version number does not match while reading and during commit then StaleObjectStateException is thown so that user can gracefully handle the data. Performance is better as there is no locking mechanism in this type of concurrency.
* Question:
  + List 5 different tiers/layers of Service Oriented Architecture. No explanation necessary.
* Solution:
  + View🡺Controller🡺Service🡺Domain🡺Persistence
* Question:
  + Explain the difference between “field” vs. “property” accesses in Hibernate?
* Solution:

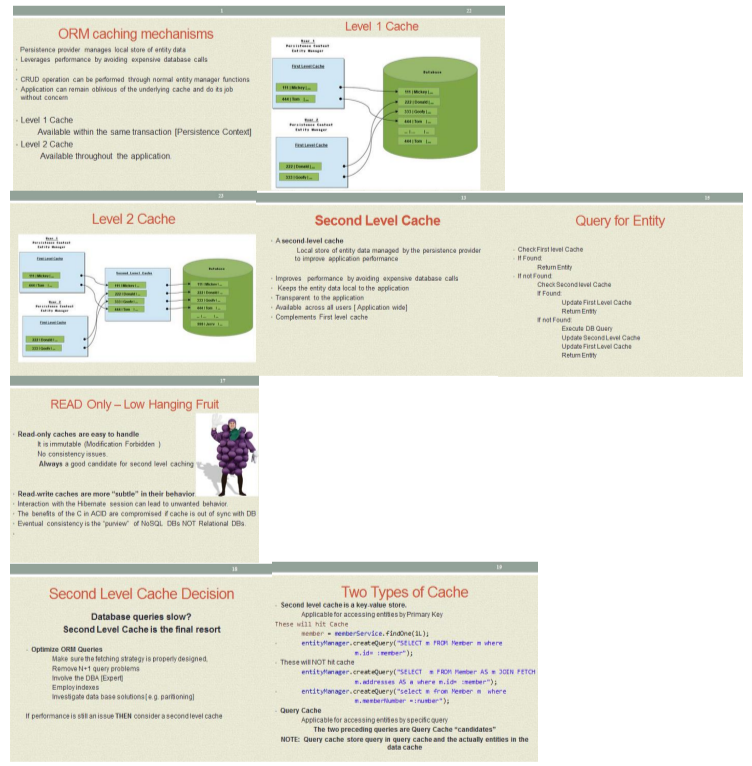
|  |  |
| --- | --- |
| Field Access | Property Access |
| @Access modifier is place at field level | @Access modifier is place at getter or setter level |
| Better encapsulation as Hibernate uses the Java Reflection mechanism for field level access | Access is via getter and setter who is a normal access. |
| No additional set of getters and setters are needed | If in future there is a need for additional validation then additional set of getter and setter to be implemented |
| There is low probability value being tapper while being loaded from database | Probability of value being tapper on getting loaded from Database |

* Question:
  + What are two main advantages of using an ORM (versus using plain database calls)?
* Solution:
  + Performance:
    - Caching
    - Optimized
  + Productive:
    - Few lines of codes are required for persistence
  + Maintainability:
    - ORM maps the business objects to database. All the mapping are done in one place. Less code is needed for maintaining
* Explain the difference between implicit vs. explicit updates in Hibernate?
  + Implicit update: In hibernate any changes made in the entity at the managed state is automatically saved in the database. We do not have to manually update the same.
  + Explicit update: If the entity is in detached state and we want to update the information in that entity then we need to manually write session.update to bring the entity back to managed state for persistence in Database.
* Question:
  + Explain the Single-Table inheritance strategy in JPA?
* Solution:
  + In single-table inheritance strategy, the fields in sub-classes of the hierarchy are all brought down to a single table, the DiscriminatorColumn indicate the table type.
  + Adv:
    - Simple and easy to implement
    - Fast performance
  + Dis:
    - Denormalized and Nullable columns
    - Table may contain the lots of columns
* Question:
  + Name two advantages and two disadvantages of using the Joined-Table inheritance strategy. Provide a little explanation so that I know that you understand it.
* Solution:
  + Adv:
    - Tables are created as per domain views
    - schemas are normalized
  + Dis:
    - Due to join the query performance can be slower than single table
    - Multiple steps will be involved for insert and delete operations
* Question:
  + Under what situations Hibernate flushes the cache and writes all updates to DB? Name 3.
* Solution:
  + Txn.commit()
  + Session.flush()
  + Executing session.createQuery from within the transaction
* Question:
  + Is this the proper way to map a bi-directional mapping in JPA? Explain.



* Solution:
  + No, this is not the proper way. With this Car schema will have two FK columns owner\_id and person\_id which is not acceptable. Best approach is to use MappedBy at OneToMany side and remove joinColumn from there.
* Question:
  + Given the requirements of next questions (DB and Java design), write an HQL query to find out:
  + a) List of students who have taken course with code 544 and have a grade of 3.8 or better
    - Solution:
    - Query query=session.createQuery("select distinct a from Student a join a.grades b join b.course c where c.name=:name and b.grade>=:grade");
    - query.setParameter("name","544");
    - query.setParameter("grade","3.4");
  + List of faculty whose biography is more than 1000 characters
    - Query query=session.createQuery("select distinct a from Faculty a join a.biography b where length(b)>:biography");
    - query.setParameter("biography","1000");
* Question:
  + 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 basis components that make up a spring application?
* Solution:
  + We general make a call to the library as and when needed. However, IOC follows the hollyhood principle that is Do not call us, we will call you. IOC enables us to do programming over the interface. IOC is also know as Dependency Injection where the dependent objects are externally injected to another object without having changes made in the core line of the code.
* Prototype beans are default eagerly loaded 🡺 false
* Singleton beans are default eagerly loaded🡺true
* Prototype beans do not have a “destroy” cycle 🡺true
* Singleton beans do not have a “destroy” cycle🡺false
* Order of events when spring context is loaded?
  + Spring starts reading the config file
  + Spring initializes all singleton beans (constructor arguments will set at this stage)
  + Spring sets all the properties defined for all beans
  + Spring calls the init or @postContruct methods of all beans that have such methods
  + Spring do rest of the work
  + On close, spring will call all the destroy or @preDestroy methods
* Why you want to use constructor injection over field or property injection?
  + There may be situation where you want to initialize the default properties of the class who would be used sub-sequent. In this scenario when need to have constructor injection. Setter will also do the job, however, failing to initialize early and invoking the same may result in null pointer exception
* What is Aspect in AOP?
  + It is the implementation of Advices at the joinPoints.
* Write the point cut execution expression for an around advice that runs around all the public service layer methods.
  + @Around(“expression(public \* edu.miu.cs.cs544.service.\*.\*(..))”)
* Explain the “un-repeatable read” issue and method what isolation level solve it. How?
  + “Un-repeatable read” issue is an issue that occurs when we try to read the same row twice from within the transaction and the result is not consistence due to concurrent update. This issue is basically seen in read committed isolation level.
  + Repeatable Read – isolation level solves this issue. In this level, additional read lock is maintained over previous lock so as to ensure that once the read lock is place then all sub-sequent read will give the same result.
  + Read committed is an isolation level that guarantees that any data read was committed at the moment is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. It makes no promise whatsoever that if the transaction re-issues the read, will find the Same data, data is free to change after it was read.
  + Repeatable read is a higher isolation level, that in addition to the guarantees of the read committed level, it also guarantees that any data read cannot change, if the transaction reads the same data again, it will find the previously read data in place, unchanged, and available to read.
* Question:
  + Explain the session per operation anti-patterns and mention at least two issues with it?
* Solution:
  + Session Per Operation anti-pattern is functional at the persistence level. In this pattern the session is open and closed for each database call in a single thread. Opening and closing the session for each operation is a bulky task and can consume resource and degrade the performance.
  + Issues are:
    - No caching
    - Can have one transaction per operation and hence lack unit of work
    - Code repetition with boiler plates that are mostly copy-paste
    - Improper exception handling as the exception are caught at the lowest level.
* Question:
  + Every interaction with an RDBMS requires a transaction whether a read or write. Without a Transaction Management Capability like Spring’s, DB operation would fail?
* Solution:
  + False, Database has building transaction management capability. Spring TM provides flexibility of managing across many databases independent of underlying database technologies
* Question
  + Spring Transaction Management is based on a logic unit of work?
* Solution
  + True, Spring TM manages multiple DB independent of underlying DB technology
* Question
  + Spring transaction management with JPA requires a Persistence Context
* Solution
  + True, as it establishes the DB connection and maintain a cache for “DB-aware” objects
* Question
  + Spring @Transaction has no built-in metadata for managing any of the DB ACID properties.
* Solution
  + False, @Transaction has an optional parameter for indicating isolation level
* Question
  + Spring Declarative Transaction Management requires little or no application code related to transaction management.
* Solution
  + True, Spring DTM has little impact on application code. Can simple add the annotation @Transaction and spring take care of rest
* Question:
  + 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.
* Solution:
  + Object-Relational Impedance mismatch means there are two different technologies and are two different ways to operate
  + Examples
    - ORM has object which is a instance of class that uniquely identifies it
    - ORM support association
    - ORM support inheritance but DB does not
    - ORM traverse via association to get data while DB used queries and joins
* Question
  + 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?
* Solution:
  + 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



* Explain the concept of ORM caching. Include a discussion of : • First level relate to Persistence Context; Fetch Strategy • Second level o Read-only - read-write o Second-level .vs. query o When do you decide to use a second level cache? Be specific. Give examples. Diagrams are good.
* Question
  + Imagine that you wish to keep last sequential customer number in a DB table and you have an online signup form where customers can signup for services and each receive a new customer number (next number in sequence). Would you use “optimistic concurrency” or “pessimistic concurrency” for this scenario? Why? Justify your answer
* Solution
  + I would use pessimistic concurrency because it ensure a much stronger isolation and is necessary for a case like this in which two
  + 
* Only Managed Beans can be injected in Spring, a POJO or JavaBean cannot
  + False, If the POJO or JavaBean is a Spring Managed bean, they can be injected
* @Autowired works only on interfaces. It cannot work directly on classes.
  + False, It can work on classes. However you lose some of the value, testing; changing implementations