Hibernate

What is Hibernate ?
0. Complete solution to the problem of managing persistence in Java.
1. ORM tool.(Object Relational Mapping) used mainly in data access layer or DAO layer.
2. Provides automatic & transperent persistence.
3. JPA(Java Persistence API) implementor
JPA vs Hibernate
JPA ---standard part of J2EE specification --vendor --J2EE / Jakarta EE (sun/oracle/Eclipse)
Implementation classes -- JAR ---hibenrate core JARs(implementor of JPA)

Provides automatic & transparent persistence framework to store & retrieve data from database.
Open Source Java based framework founded by Gavin King in 2001, hosted on hibernate.org
Currently hosted on sourceforge.net
Java Persistence API (JPA) compliant
Current version Hibernate 5.x / 6.x
Other popular ORM Frameworks :: EclipseLink,iBATIS,Kodo etc.

Why Hibernate ? (refer to readme)
1. open source n light weight
2. supports cache (L1 , L2 , query cache) : faster performance
3. auto table creation.
4. simplifies join queries
5. 100 % DB independent (HQL/JPQL ---Hibernate : DB dialect -- converts DB independent queries in DB specific syntax)
Hibernate 5.x onwards : no need to specifiy DB dialect property in config file (hibernate.cfg.xml : run time classpath)
6. Hibernate developer doesn't have to go to DB level , DB ,table ,cols , rows sql set up the db conn , prepare stmts (st/pst/cst) exec queries : process RST : convert it into pojo / collection of POJOs All of above will be automated by Hibernate
7. JDBC : fixed db conn.(new separate conn per call to DriverManager.getConnection)

Hibernate creates :internal connection pool => collection of DB connections
when -> hibernate framework booting time at the time of creation of singleton SessionFactory(SF)
at the time confgure() -- hibernate.cfg.xml(location : by default run time class path) is parsed :
DB config details -- drvr class , db url , user name , pwd
hibernate.connection.pool_size= 10 (max size)

In DAO layer : When you invoke , open session n begin tx : db conn is pooled out -- wrapped in Session instance n reted to caller.
try
CRUD work (save/get/JPQL/update/delete...)
end of try --commit

catch --RunTimeExc --rollback
finally : session .close ---pooled out db cn simply rets to the pool : so that the same conn can be REUSED for some other request.

8.Solves the important issue of Impedance mismatch in DBMS
Object world (java objs in heap , inheritance , association , polymorphism) -----
RDBMS (table , row cols ,E-R,FKs,join tables...)
9. Exception translation mechanism
   Hibernate translates checked SQL excs --un checked hibernate excs (org.hibernate.HibernateException) : so that prog is not forced to handle the same.

->> Object oriented query language, where table names are replaced by POJO class names & column names are replaced by POJO property names, in case sensitive manner.

->>>Hibernate's   Automatic Dirty checking
The process of automatically updating the database with the changes to the persistent object when the session is flushed(@ commit) is known as automatic dirty checking.

An object(POJO) enters persistent state when any one of the following happens:
When the code invokes session.save, session.persist or session.saveorUpdate or session.merge
OR
When the code invokes session.load or session.get
OR
Result of JPQL

Any changes to a persistent object are automatically saved to the database when the session in flushed.
Flushing is the process of synchronizing the underlying database with the objects in the session's L1 cache.
Even though there is a session.flush method available but you generally don’t need to invoke it explicitly.
A session gets flushed when the transaction is commited.

Some basics :
1. Hibernate uses runtime reflection to determine the persistent properties of a class.

2. The objects to be persisted(called as POJO or Entity) are defined in a mapping document or marked with annotations.
Either these HBM XML docs or annotations serves to describe the persistent fields and associations, as well as any subclasses or proxies of the persistent object.

3. The mapping documents or annotations are compiled at application startup time

and provide the framework with necessary information for a persistent class.
4. What is Hibernate config.?
An instance of Hib Configuration allows the application to specify properties and mapping documents to be used at the frmwork start-up.
The Configuration : initialization-time object.

5. SessionFactory is created from the compiled collection of mapping documents .
The SessionFactory provides the mechanism for managing persistent classes, the Session interface.

6. A web application or Java SE apllication will create a single Configuration, build a single instance of SessionFactory and then instantiate multiple Sessions in threads servicing client requests.
SessionFactory : immutable and does not reflect any changes done later to the Configuration.

7. The Session class provides the interface between the persistent data store and the application.
The Session interface wraps a JDBC connection, which can be user-managed or controlled by Hibernate.

-> What is Session? (org.hibernate.Sesion : i/f)
Represents a wrapper around pooled out jdbc connection.
Session object is persistance manager for the hibernate application
Session object is the abstraction of hibernate engine for the Hibernate application
Session object provides methods to perform CRUD operations
Session just represents a thin wrapper around pooled out DB connection.
Session is associated implicitely with L1 cache (having same scope as the session lifetime) , referred as Persistence context.

Example of CRUD

save()	-	Inserting the record
get() / load()	-	Retrieveing the record
update()	-	Updating the record
delete()	-	Deleting the record

-> What is SessionFactory?(org.hibernate.SessionFactory : i/f)
It is a provider(factory) of session objects.
We use sessionfactory object to create session object(via openSession or getCurrentSession).
It is singleton(1 instance per DB / application) ,immutable,inherently thrd safe.
It is a heavy weight object, therefore it has to be created only once in the beginning for an application & that too at the very beginning.
It is associtated with L2 cache(must be explicitely enabled)

-> What is Configuration Object ?(org.hibernate.cfg.Configuration)
Configuration object is used to create the SessionFactory object.
Object Oriented Representation of Hibernate configuration file and mapping file is nothing but Configuration object.
When we call configure() method on configuration object ,hibernate configuration file(hibernate.cfg.xml placed in run time classpath) and mapping files are loaded in the memory.

Why connection pooling?
Java applications should use connection pools because :
    Acquiring a new connection is too expensive
    Maintaining many idle connections is expensive
    Creating prepared statements is expensive

Hibernate provides basic or primitive connection pool -- useful only for classroom testing.
Replace it by 3rd party vendor supplied connection pools(eg Apache or C3P0 or hikari in spring boot) for production grade applications.

Managing an Entity Instances Life Cycle

You manage entity instances(or POJOs) by invoking operations on the entity/POJO using EntityManager/Session instance.

Entity instances are in one of four states (2 imp aspects of it : its asso. with the hibernate session & sync of its state with the underlying DB)

States : new or transient , managed or persistent, detached, removed.

New entity instances have no persistent identity and are not yet associated with a hib. session (transient)

Managed entity instances have a persistent identity and are associated with a hib. session.(persistent : via save() or saveOrUpdate()) Changes to DB will be done when tx is commited.

Detached entity instances have a persistent identity and are not currently associated with a persistence context/Hib session.

Removed entity instances have a persistent identity, are associated with a persistent context and are scheduled for removal from the data store.(removed via session.delete(obj))

Introduction to Hibernate Caching

While working with Hibernate web applications we will face so many problems in its performance due to database traffic. That too when the database traffic is very heavy . Actually hibernate is well used just because of its high performance only. So some techniques are necessary to maintain its performance.

Caching is the best technique to solve this problem.

The performance of Hibernate web applications is improved using caching by optimizing the database applications.

The cache actually stores the data already loaded from the database, so that the traffic between our application and the database will be reduced when the application want to access that data again.
At maximum the application will work with the data in the cache only. Whenever some another data is needed, the database will be accessed. Because the time needed to access the database is more when compared with the time needed to access the cache. So obviously the access time and traffic will be reduced between the application and the database.
Here the cache stores only the data related to current running application. In order to do that, the cache must be cleared time to time whenever the applications are changing.

POJO/Entity Life cycle

1.Transient State
An object is said to be in transient state if it is not associated with the session,and has no matching record in the database table.

2.Persistent State
An object is said to be in persistent state if it is associated with session object (L1 cache) and will result into a matching record in the databse table.(i.e upon commit)

```
session.save(account);tx.commit();
        or
Account account=session.get(Account.class,102);
        OR via HQL/JPQL
```

When the POJO is in persistent state it will be in synchronization with the matching record in DB i.e if we make any changes to the state of persistent POJO it will be reflected in the database.(after commiting tx) -- i.e automatic dirty checking will be performed(resulting in insert/update/delete)

3.Detached state
Object is not associated with session but has matching record in the database table.

If we make any changes to the state of detached object it will NOT be reflected in the database.

```
session.clear();
session.evict(Object);
session.close();
```

Note :
By calling update method on session object it will go from detached state to persistent state.
By calling delete method on session object it will go from persistenet state to transient state.

When the object is in detached state record is present in the table but object is not in sync with database, therefore update() method can be called
to update the record in the table

-> Which exceptions update method can raise?
1.StaleStateException -- If u are trying to update a record (using session.update(ref)), whose id doesn't exist.
  i.e update can't transition from transient --->persistent
  It can only transition from detached --->persistent.
eg -- update_book.jsp -- supply updated details + id which doesn't exists on db.

2. NonUniqueObjectException -- If there is already persistence instance with same id in session.
eg -- UpdateContactAddress.java

Natural Key Vs Surrogate Key

If u have User reg system -- then u have a business rule that --- user email must be distinct. So if u want to make this as a prim key --then user will have to supply this during regsitration.
This is called as natural key. Since its value will be user supplied , u cant tell hibernate to generate it for u---i.e cant use @GeneratedValue at all.

Where as -- if u say I will reserve user id only for mapping purposes(similar to serial no), it need not come from user at all & can definitely use hib. to auto generate it for u---this is ur surrogate key & can then use @GeneratedValue.

Hibernate API

0. SessionFactory API
public Session openSession() throws HibernateExc
opens new session from SF,which has to be explicitely closed by prog.

public Session getCurrentSession() throws HibernateExc
Opens new session , if one doesn't exist , otherwise continues with the exisitng

one.
Gets automatically closed upon Tx boundary or thread over(since current session is bound to current thread --mentioned in hibernate.cfg.xml property ---current_session_context_class ---thread)

---

1. public void persist(Object transientRef)
if u give some non-null id (existing or non-existing) while calling persist(ref) --gives exc
org.hibernate.PersistentObjectException: detached entity passed to persist:
why its taken as detached ? ---non null id.

---

2. public Serializable save(Object ref)
save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exc.
Ignores ur passed id & creates its own id & inserts a row.

---

3. saveOrUpdate
public void saveOrUpdate(Object ref)
--either inserts/updates or throws exc.
null id -- fires insert (works as save)
non-null BUT existing id -- fires update (works as update)
non-null BUT non existing id -- throws StaleStateException --to indicate that we are trying to delete or update a row that does not exist.

3.5 merge
public Object merge(Object ref)
I/P -- either transient or detached POJO ref.
O/P --Rets PERSISTENT POJO ref.

null id -- fires insert (works as save)
non-null BUT existing id -- fires update (select , update)
non-null BUT non existing id -- no exc thrown --Ignores ur passed id & creates its own id & inserts a row.(select,insert)

---

4. get vs load
& LazyInitilalizationException.

---

5. update
Session API
public void update(Object object)
Update the persistent instance with the identifier of the given detached instance.
I/P --detached POJO containing updated state.
Same POJO becomes persistent.

Exception associated :
1. org.hibernate.TransientObjectException: The given object has a null identifier:
i.e while calling update if u give null id. (transient ----X ---persistent via update)
2. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.

3.
org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session
---
6. public Object merge(Object ref)
Can Transition from transient -->persistent & detached --->persistent.
Regarding Hibernate merge
1. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().
2. API of Session
Object merge(Object object)
3.
Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
3.If there is no persistent instance currently associated with the session, it will be loaded.
4.Return the persistent instance. If the given instance is unsaved, save a copy of and return it as a newly persistent instance. The given instance does not become associated with the session.
5. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

7.public void evict(Object persistentPojoRef)
It detaches a particular persistent object
 from the session level cache(L1 cache)
(Remove this instance from the session cache. Changes to the instance will not be synchronized with the database.)
---
8. void clear()
When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.
But Databse Connection is not returned to connection pool.
(Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)
---
9. void close()
When close() is called on session object all
the persistent objects associated with the session object become detached(l1 cache is cleared) and also closes the Database Connection.
---
10. void flush()
When the object is in persistent state , whatever changes we made to the object state will be reflected in the databse only at the end of transaction.
BUT If we want to reflect the changes before the end of transaction
(i.e before commiting the transaction)
 call the flush method.
(Flushing is the process of synchronizing the underlying DB state with persistable state of session cache)
---
11. boolean contains(Object ref)

The method indicates whether the object is
associated with session or not.(i.e is it a part of l1 cache ?)

---

12. void refresh(Object ref) -- ref --persistent or detached
This method is used to get the latest data from database and make
corresponding modifications to the persistent object state.
(Re-reads the state of the given instance from the underlying database

---

ENTITY TYPE VS VALUE TYPE

Entity Types :
1. If an object has its own database identity (primary key value) then it’s type is Entity Type.
2. An entity has its own lifecycle. It may exist independently of any other entity.
3. An object reference to an entity instance is persisted as a reference in the database (a foreign key value).
eg : College is an Entity Type. It has it’s own database identity (It has primary key).

Value Types :
1. If an object don’t have its own database identity (no primary key value) then it’s type is Value Type.
2. Value Type object belongs to an Entity Type Object.
3. It’s embedded in the owning entity and it represents the table column in the database.
4. The lifespan of a value type instance is bounded by the lifespan of the owning entity instance.

Different types of Value Types

Basic, Composite, Collection Value Types :
1. Basic Value Types :
Basic value types are : they map a single database value (column) to a single, non-aggregated Java type.
Hibernate provides a number of built-in basic types.
String, Character, Boolean, Integer, Long, Byte, … etc.

---

2. Composite Value Types :
In JPA composite types also called Embedded Types. Hibernate traditionally called them Components.
2.1 Composite Value type looks like exactly an Entity, but does not own lifecycle and identifier.

Annotations Used
1. @Embeddable :
Defines a class whose instances are stored as an intrinsic part of an owning entity and share the identity of the entity. Each of the persistent properties or fields of the embedded object is mapped to the database table for the entity. It doesn't have own identifier.

eg : Address is eg of Embeddable
Student HAS-A Address(eg of Composition --i.e Address can't exist w/o its owning Entity i.e Student)
College HAS-A Address (eg of Composition --i.e Address can't exist w/o its owning Entity i.e College)
BUT Student will have its own copy of Address & so will College(i.e Value Types don't support shared reference)

2. @Embedded :
Specifies a persistent field or property of an entity whose value is an instance of an embeddable class. The embeddable class must be annotated as Embeddable.
eg : Address is embedded in College and User Objects.

3. @AttributesOverride :
Used to override the mapping of a Basic (whether explicit or default) property or field or Id property or field.

In Database tables observe the column names. Student table having STREET_ADDRESS column and College table having STREET column. These two columns should map with same Address field streetAddress. @AttributeOverride gives solution for this.
To override multiple column names for the same field use @AtributeOverrides annotation.
eg : In Student class :

```
@Embedded
 @AttributeOverride(name="streetAddress", column=@Column(name="STREET_ADDRESS"))
private Address address;
```

where , name --POJO property name in Address class

---

3. Collection Value Types :
Hibernate allows to persist collections.
But Collection value Types can be either collection of Basic value types, Composite types and custom types.
eg :
Collection mapping means mapping group of values to the single field or property. But we can’t store list of values in single table column in database. It has to be done in a separate table.

eg : Collection of embeddables

```
@ElementCollection
        @CollectionTable(name="CONTACT_ADDRESS",
joinColumns=@JoinColumn(name="USER_ID"))
        @AttributeOverride(name="streetAddress",
column=@Column(name="STREET_ADDRESS"))
        private List<ContactAddress> address;
```

eg : collection of basic type

```
        @ElementCollection
        @CollectionTable(name="Contacts", joinColumns=@JoinColumn(name="ID"))
        @Column(name="CONTACT_NO")
        private Collection<String> contacts;
```


Advanced Hibernate

Relationship between Entity n Entity
(Inheritance , Association : HAS-A)

Types of associations
one-to-one
one-to-many
many-to-one
many-to-many

Objective --Using one-to-many & many-to-one association between entities

eg : Course 1 <---->* Student
Different type of relationships between entities
One To One
One To Many
Many To One
Many To Many

---

1. One To Many bi directional relationship between the entities
JPA Annotations : @OneToMany & @ManyToOne

eg : Course 1 <----->* Student
Table Relationship
courses Table columns : id,title, start_date , end_date , fees , capacity
students Table columns : id,name,email + Foreign Key(FK) : course_id

Since courses table has a OneToMany relationship with the students table , a single course row can be referenced by multiple student rows.

The course_id column in the students table , maps this relationship via a foreign key that references the primary key of the courses table.

Since you can't insert a student record , w/o course record
parent-side (@OneToMany) : course
child-side (@ManyToOne) : student

The @ManyToOne association is responsible for synchronizing the foreign key column with the Persistence Context (the First Level Cache).

As a thumb rule (for perfomance benefits) : DO NOT use uni directional @OneToMany associations

Owning side of the association
The side having the join column in its table is called the owning side or the owner

of the relationship.

Non owning (inverse side)
The side that does not have the join column is called the non owning or inverse side.

Entities involved :
Course Entity
Student Entity

Description
Course : one , parent , inverse
Student : many , child , owning side (FK)

Best Practices to code a bidirectional @OneToMany association
eg : Course 1 <----->* Student
Entity Relationships
Course POJO properties : id,title, startDate , endDate , fees +
@OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL,orphanRemoval=true)
private List<Student> students=new ArrayList<>();
Note : Always init collection to empty one , to avoid null pointer excpetion

Student POJO properties : id,name,email +
@ManyToOne
@JoinColumn(name="course_id")
private Course selectedCourse

Detailed explanation
1. Add Suitable mapping annotations : @OneToMany & @ManyToOne
otherwise JPA / Hibernate throws MappingException
---
2. Add mappedBy attribute in the inverse side of the association

What is mappedBy & when it's mandatory?
Mandatory only in case of bi-dir associations
It's attribute of the @OneToMany / @ManyToMany / @OneToOne annotation.

What will happen if you don't add this attribute , in case of one-to-many
 Additional table (un necessary for the relationship mapping) gets created

It MUST appear in the inverse side of the association.
What should be value of mappedBy ?
name of the association property as it appears in the owning side.

eg : In Course POJO : inverse side
@OneToMany(mappedBy="selectedCourse")
public List<Student> getStudents() {..}
---
3. Use @JoinColumn to Specify the Join Column Name (FK column)
Use it to override hibernate's default naming strategy for column names.

---
4. Cascade from Parent-Side to Child-Side
If you don't add cascade option : what will happen ?
eg : When try to save Course object, with multiple students, insert query gets fired only on courses table.
Reason -- default cascade type = none
Solution --Add suitable cascade type & observe.
eg : @OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL)
public List<Student> getStudents(){...}
---
5. What will happen if simply add student reference into the list?
eg :
eg : Course newCourse=new Course(....);
newCourse.getStudents().add(newStudent1);
newCourse.getStudents().add(newStudent2);
newCourse.getStudents().add(newStudent3);
session.persist(newCourse);

Ans : 1 record will be inserted into courses table. Thanks to cascade option , 3 records will be inserted into students table. BUT value of FK will be null.
Why : No linking from child ----> parent &

Which is the best way to establish bi-dir linking (As per THE founder of Hibernate : Gavin King)
Add helper methods in the parent side of the POJO
eg : In Course POJO
```
public void addStudent(Student s)
{
  students.add(s);
  s.setSelectedCourse(this);
}

For removing bi dir link
public void removeStudent(Student s)
{
  students.remove(s);
  s.setSelectedCourse(null);
}
```
Above approach is recommended to keep both sides of the association in sync.
---
6. Set orphanRemoval on the Parent-Side
Setting orphanRemoval on the parent-side guarantees the removal of children without references.
It is good for cleaning up dependent objects that should not exist without a reference from an owner object.

eg : Cancel Student admission

If you don't add cascade option : problem observed
When try to save Course object, with multiple students, insert query gets fired

only on courses table.
Reason -- def cascade type = none
Solution --Add suitable cascade type & observe.
eg : @OneToMany(mappedBy="selectedCourse",
cascade=CascadeType.ALL)
public List<Student> getStudents(){...}

Problem associated with one to many
org.hibernate.LazyInitializationException
Trigger : GetCourseDetails : while accessing the Student details

WHY ?

Hibernate follows default fetching policies for different types of associations
one-to-one : EAGER
one-to-many : LAZY
many-to-one : EAGER
many-to-many : LAZY

one-to-many : LAZY
Meaning : If you try to fetch details of one side(eg : Course) , will it fetch auto details of many side ?
NO (i.e select query will be fired only on courses table)
Why ? : for performance

When will hibernate throw LazyInitializationException ?
Any time you are trying to access un-fetched data from DB , in a detached manner(outside the session scope)
cases : one-to-many
many-many
session's load

un fetched data : i.e student list in Course obj : represented by : proxy (substitution) : collection of proxies
proxy => un fetched data from DB

Solutions
1. Change the fetching policy of hibernate for one-to-many to : EAGER
eg :
```
@OneToMany(mappedBy = "selectedCourse",cascade =
CascadeType.ALL,fetch=FetchType.EAGER)
        private List<Student> students=new ArrayList<>();
```

Is it recommneded soln : NO (since even if you just want to access one side details , hib will fire query on many side) --will lead to worst performance.

2.
```
@OneToMany(mappedBy = "selectedCourse",cascade = CascadeType.ALL)
        private List<Student> students=new ArrayList<>();
```
Solution : Access the size of the collection within session scope : soln will be

applied in DAO layer

Dis Adv : Hibernate fires multiple queries to get the complete details

3. How to fetch the complete details , in a single join query ?
Using "join fetch" keyword in JPQL
String jpql = "select c from Course c join fetch c.students where c.title=:ti";

Another trigger for lazy init exception
: Session's API
load.