

SunilOS



Hibernate

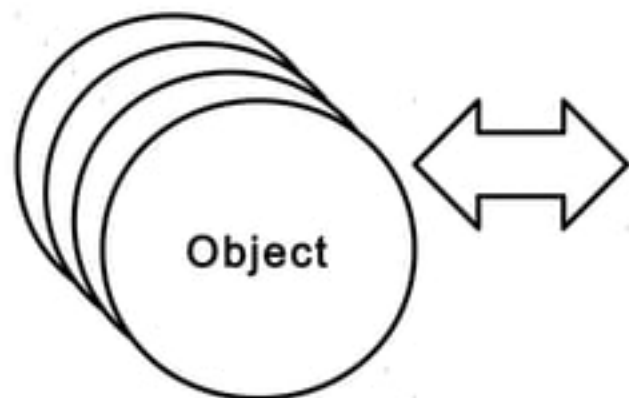
www.sunilos.com

www.raystec.com



Hibernate is ORM

- ❑ Hibernate is an Object to Relational Mapping Tool



ID	NAME	Salary
1	Ram	1000
2	Shyam	1100
3	Vijay	1200
4	Jay	1300

Object to Relational Mapping

❑ USERS

- ID
- FIRST_NAME
- LAST_NAME
- LOGIN_ID
- PASSWORD

```
public class User {  
    private int id = 0;  
    private String userId = null;  
    private String password = null;  
    private String firstName = null;  
    private String lastName = null;  
  
    public User(){} //Def Constructor  
  
    public String getPassword() {  
        return password;  
    }  
    public void setPassword(String password) {  
        this.password = password;  
    }  
    ..Other Setter/Getter
```

- Class will become Table
- Attributes will become columns
- Instances will become Records

POJO (Plain Old Java
Object)
Data Transfer Object
(DTO)
Value Object (VO)

POJO

- ☐ Are persistent classes.
- ☐ Represent domain model.
- ☐ Identifier Property (optional) has PK of a table.
- ☐ Must have default constructor.
- ☐ Accessor methods will have naming convention setName, getName, isName.
- ☐ Properties need not to be public

JDBC How to

```
conn.setAutoCommit(false); //Begin Trn
Statement stmt = conn.createStatement();
int i= stmt.executeUpdate("INSERT into users
(id, first_name,
last_name,address,login_id,password)
VALUES (1,'Sunrays','technologies','South
Tukoganj', 'sunrays','sun')");
conn.commit(); //End Trn
```

XML Mapping

- ☐ Readable metadata
- ☐ POJO / table mappings
- ☐ Attribute/Column mapping
- ☐ Surrogate key generation strategy
- ☐ Collection metadata
- ☐ Fetching strategies
- ☐ POJO and XML file name must be same and will be stored in same folder
- ☐ XMLs should have .hbm.xml extension

User.hbm.xml

```
<hibernate-mapping package="in.co.sunrays.dto">

<class name="User" table="ST_USER">

<id name="id" column="ID">
    <generator class="increment" />
</id>

<property name="lastName" column="LAST_NAME" />
<property name="firstName" column="FIRST_NAME" />
<property name="address" column="STREET" />
<property name="userId" column="LOGIN_ID" />
<property name="password" />

</class>
</hibernate-mapping>
```


Configuration :hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://localhost/test</property>
<property name="connection.username">root</property>
<property name="connection.password">root</property>

<!-- SQL dialect -->
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

Configuration :hibernate.cfg.xml

```
<!-- Echo all executed SQL to stdout -->
```

```
<property name="show_sql">true</property>
```

```
<!-- Drop and re-create the database schema on startup -->
```

```
<property name="hbm2ddl.auto">create</property>
```

```
<mapping resource="in/co/sunrays/dto/User.hbm.xml" />
```

```
<mapping resource="in/co/sunrays/dto/Role.hbm.xml" />
```

```
</session-factory>
```

```
</hibernate-configuration>
```


Configuration

1. Create A Java Project from Eclipse
2. Add Hibernate Capabilities
3. Insert Database (MySQL) Driver into Build Path
4. Create `User.java`
5. Create `User.hbm.xml` in same directory with `User.java`
6. Keep `hibernate.cfg.xml` in the root source folder

TestUser - Add

```
public static void testAdd() throws Exception {  
  
    User pojo = new User ();  
    pojo.setFirstName("Dheeraj");  
    pojo.setLastName("Dubey"); //Prepare POJO  
    ...  
    SessionFactory sessionFactory = new Configuration().configure()  
        .buildSessionFactory(); //Load Factory  
  
    Session s = sessionFactory.openSession(); //Create Session  
    Transaction tx = s.beginTransaction();  
  
    s.save(pojo); //Save POJO  
  
    tx.commit();  
    s.close();  
}
```

TestUser - Update

```
public static void testUpdate() throws Exception {  
  
    User pojo = new User ();  
    pojo.setId (4);  
    pojo.setFirstName("Dheeraj");  
    pojo.setLastName("Upadhyay"); //Prepare POJO  
    ...  
    SessionFactory sessionFactory = new Configuration().configure()  
        .buildSessionFactory(); //Load Factory  
  
    Session s = sessionFactory.openSession(); //Create Session  
    Transaction tx = s.beginTransaction();  
  
    s.update(pojo); //Update POJO  
  
    tx.commit();  
    s.close();  
}
```

TestUser - Delete

```
public static void testDelete() throws Exception {
```

```
User pojo= new User();  
pojo.setId (4);
```

```
...
```

```
SessionFactory sessionFactory = new Configuration().configure()  
.buildSessionFactory(); //Load Factory
```

```
Session s = sessionFactory.openSession(); //Create Session  
Transaction tx = s.beginTransaction();
```

```
s.delete(dto); //Delete POJO
```

```
tx.commit();  
s.close();  
}
```

TestUser - Get

```
public static void testGet() throws Exception {
```

```
SessionFactory sessionFactory =  
new Configuration().configure().buildSessionFactory();  
    //Load Factory
```

```
Session s = sessionFactory.openSession(); //Create Session
```

```
User pojo = (User) s.get(User.class, 4);  
System.out.println(pojo.getId());  
System.out.println(pojo.getFirstName());....
```

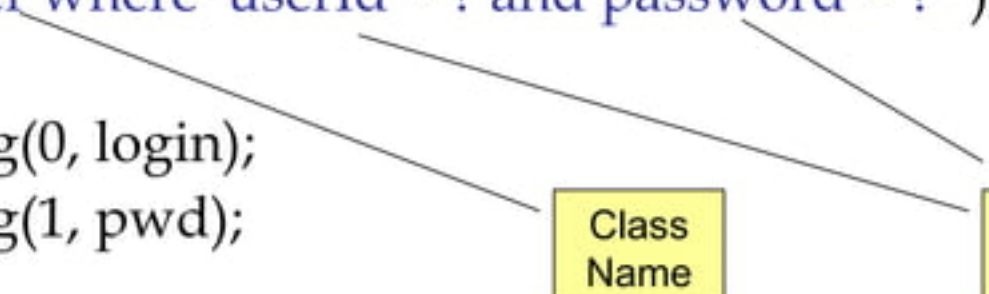
```
s.close();  
}
```

Authenticate User

```
public User authenticate(String login, String pwd) throws  
    UserNotFoundException{  
    SessionFactory sessionFactory =  
        new Configuration().configure().buildSessionFactory();  
    Session s = sessionFactory.openSession();
```

```
    Query q = s.createQuery(  
        "from User where userId = ? and password = ? ");
```

```
    q.setString(0, login);  
    q.setString(1, pwd);
```



Class
Name

Attributes

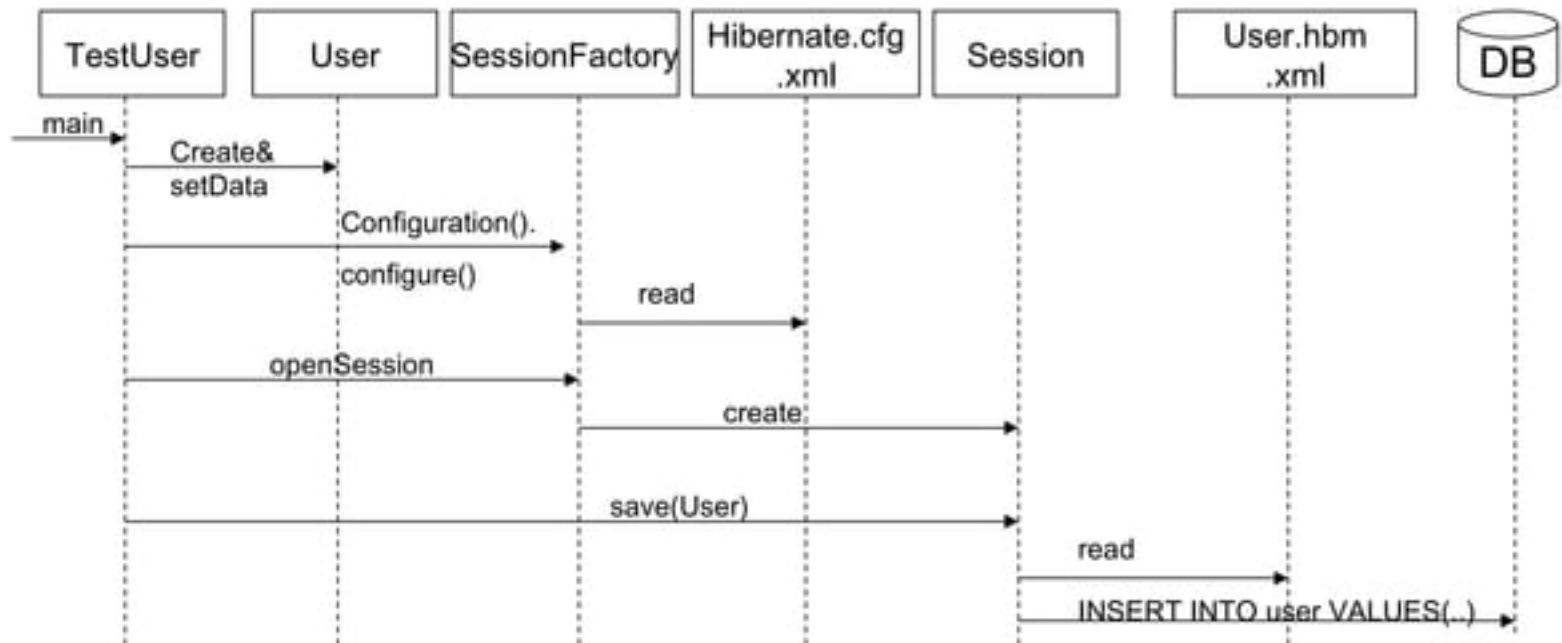
Authenticate User

```
...  
List list = q.list();  
  
if (list.size() == 1) {  
    pojo = (User) list.get(0);  
}else{  
    throw new  
        UserNotFoundException("login.invalid.user");  
}  
s.close();  
return pojo;  
}
```

Get the list of Users

```
public static void testList() throws Exception {  
  
    SessionFactory sessionFactory = new Configuration().configure()  
        .buildSessionFactory();  
  
    Session s = sessionFactory.openSession();  
  
    Query q = s.createQuery("from User");  
  
    List l = q.list();  
    Iterator it = l.iterator();  
  
    User pojo;  
    while (it.hasNext()) {  
        pojo = (User) it.next();  
        System.out.println(pojo.getId());  
        System.out.println(pojo.getFirstName());  
    }  
    s.close();  
}
```

Sequence Diagram



org.hibernate.Session

- ❑ createQuery – SELECT HQL statement
- ❑ save() – INSERT Statement
- ❑ update() – UPDATE Statement
- ❑ saveOrUpdate()
- ❑ delete() - DELETE Statement
- ❑ beginTransaction() – Returns transaction object
- ❑ createSQLQuery() – Create DB native SQL query
- ❑ load(dto.class, pk) - Loads record
 - Throws exception if record does not exist
- ❑ get(dto.class, pk) - Get a record
 - Returns null in case record does not exist
- ❑ flush() - synchronize object state with database

org.hibernate.Session

- ☐ More Interesting Methods
- ☐ persist() – just like save()
- ☐ merge()
- ☐ lock()
- ☐ refresh()
- ☐ evict()
- ☐ replicate()

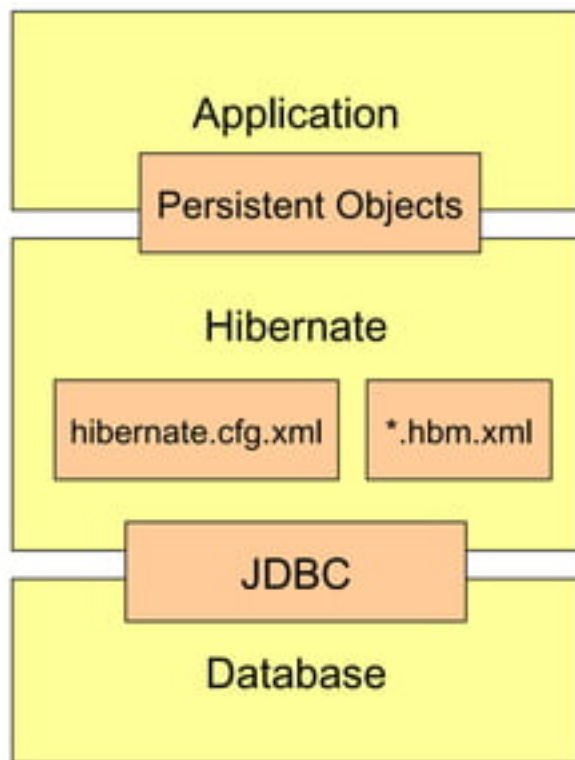
org.hibernate.Session

- ❑ A single-threaded, short-lived object representing a conversation between the application and the persistent store.
- ❑ Wraps a JDBC connection.
- ❑ Factory for Transaction.
- ❑ Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier.
- ❑ Remove Object from First Level Cache
 - `session.evict(User)`

Architecture

Hibernate basically sits between the DB and application

Can map persistent objects to tables



Hibernate

- ❑ Hibernate is an object-relational mapping tool (**ORM**) that allows for persisting Java objects in a relational database.
- ❑ Officially maintained by Jboss, Inc, started in December, **2001**
- ❑ Opensource (LGPL)
- ❑ Current Version is 3.2
- ❑ Will be core of JBoss CMP 2.0 engine

Why Hibernate

▪ Flexibility

- Handles all create-read-update-delete (CRUD) database operations.
- (CRUD) operations using simple API; no SQL
- Generates DDL scripts to create DB schema (tables, constraints, sequences)
- Flexibility to hand-tune SQL and call stored procedures to optimize performance

▪ Database Independent

- Supports over 20 RDBMS; change the database by tweaking configuration files
- Sits between the database and Java code, easy database switch without changing any code

Features

- **Object oriented query language (HQL)**

Portable query language, supports polymorphic queries etc.

- **Native SQL**

You can also still issue native SQL, and also queries by “Criteria” (specified using “parse tree” of Java objects)

- **Transparent persistence**

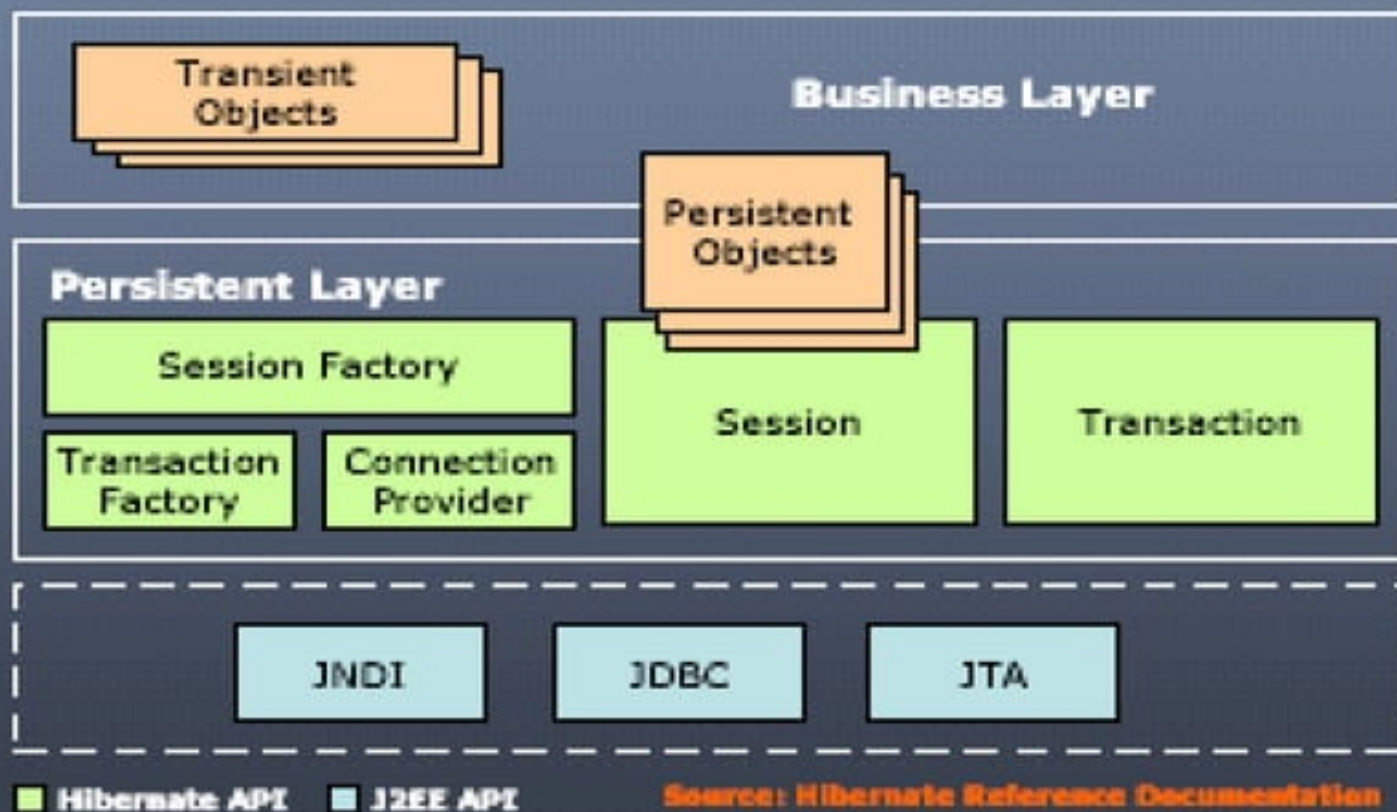
Allows easy saves/delete/retrieve for simple value objects

- **Very high performance**

“In general” due to intelligent (2-level) caching, although in a few cases hand-written SQL might beat it

Architecture

Hibernate Architecture



Hbm.xml Tags

- ☐ Primary Key
- ☐ Composite Key
- ☐ Candidate Key

Composite Key

- ❑ A PK may consists of multiple columns Ex. Dept Code + Employee Code
- ❑ Multiple properties are mapped for PK
- ❑ <composite-id> is used in place of <id>
- ❑ A separate PK class is created with composite attributes
- ❑ PK class must be serializable and override equals() and hashCode() methods to compare two primary keys.

Session Factory

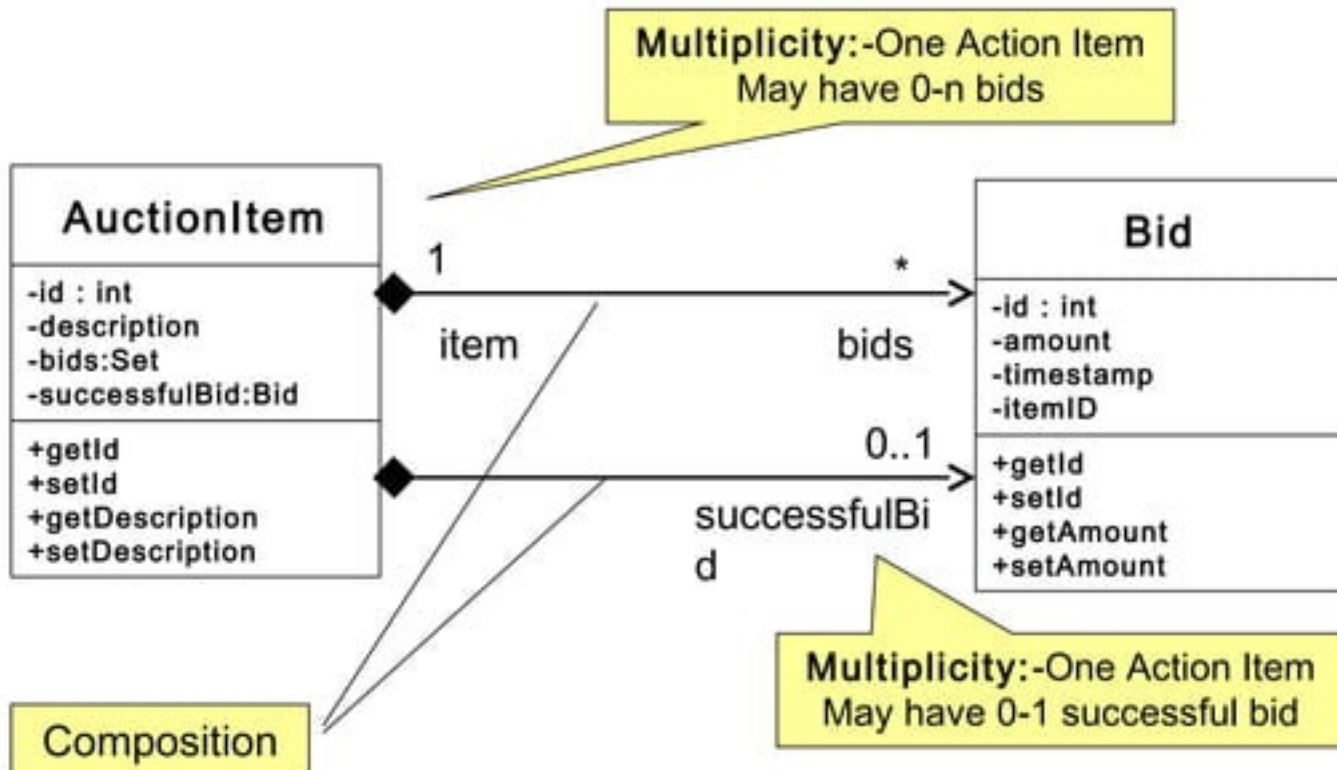
- ☐ SessionFactory is created using config settings in hibernate.cfg.xml
- ☐ Ceate one Session Factory per database
- ☐ Usually application has single SessionFactory
- ☐ It is Thread Safe
- ☐ It is immutable
- ☐ It is Database Connection Pool
- ☐ Optionally holds second-level cache

abc.cfg.xml

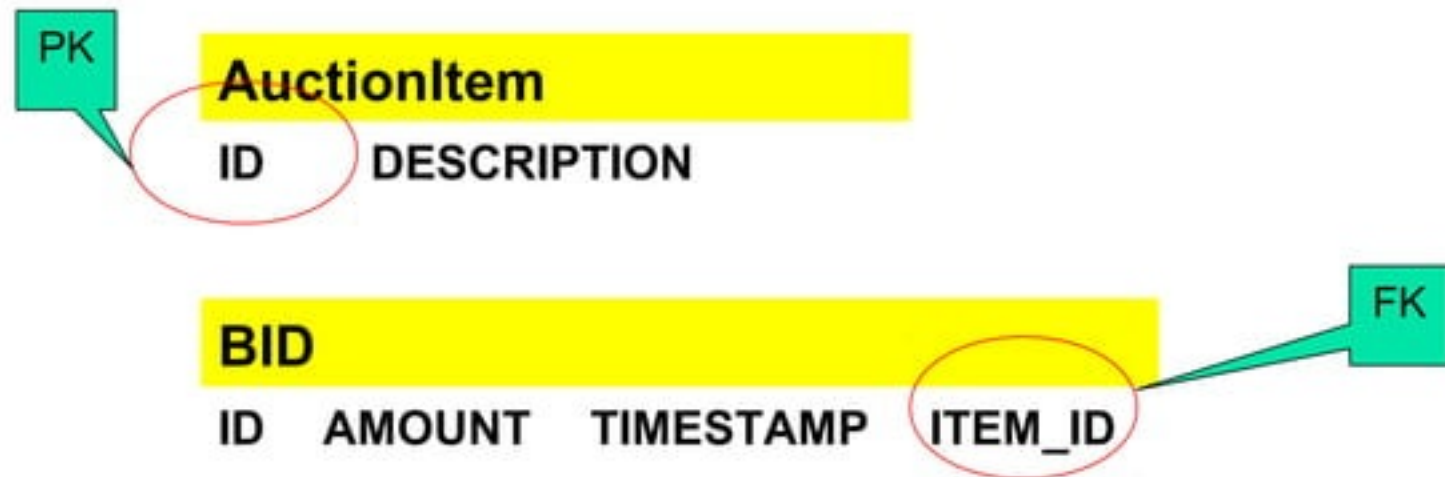
```
SessionFactory sessionFactory = new  
    Configuration().configure()  
.buildSessionFactory(); //Load Factory
```

```
SessionFactory sessionFactory = new  
    Configuration("abc.cfg.xml").configure()  
.buildSessionFactory(); //Load Factory
```

Auction Object Model Association



Auction Table Structure



ActionItem

```
public class AuctionItem{  
    private long id;  
    private Set bids;  
    private String description;  
    private Bid successfulBid = null;  
  
    public Set getBids() {  
        return bids;  
    }  
    public void setBids(Set bids) {  
        this.bids = bids;  
    }  
    .....  
}
```

- ☐ POJO
- ☐ JavaBean
- ☐ Data Transfer Object (DTO)
- ☐ BO (Business Object)
- ☐ Domain Object (DO)

Bid

```
public class Bid{  
    private long id;  
    private int amount;  
    private String timestamp;  
    private long itemId;  
  
    public int getAmount() {  
        return amount;  
    }  
    public void setAmount(int amount) {  
        this.amount = amount;  
  
        .....  
    }  
}
```

AuctionItem.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="in.co.sunrays..dto">

<class name="AuctionItem" table="AuctionItem">
<id name="id" column="ID"></id>
<property name="description" />

<set name="bids" cascade="all" lazy="true">
    <key column="ITEM_ID" />
    <one-to-many class="Bid" />
</set>
</class>
</hibernate-mapping>
```

AuctionItem
-id : int -description -bids:Set -successfulBid:Bid
+getId +setId +getDescription +setDescription

Bid.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-
    3.0.dtd">

<hibernate-mapping package="in.co.sunrays.dto">
<class name="Bid" table="BID">
    <id name="id" column="ID"></id>
    <property name="amount" />
    <property name="timestamp" " />
    <property name="itemId" column="ITEM_ID" />
</class>
</hibernate-mapping>
```

Add Auction

```
AuctionItem auctionItem = new AuctionItem();  
auctionItem.setId(2);  
auctionItem.setDescription("Auction 2");
```

```
Bid bid1 = new Bid();  
bid1.setId(1);  
bid1.setAmount(100);  
bid1.setTimestamp("XXX"); ....
```

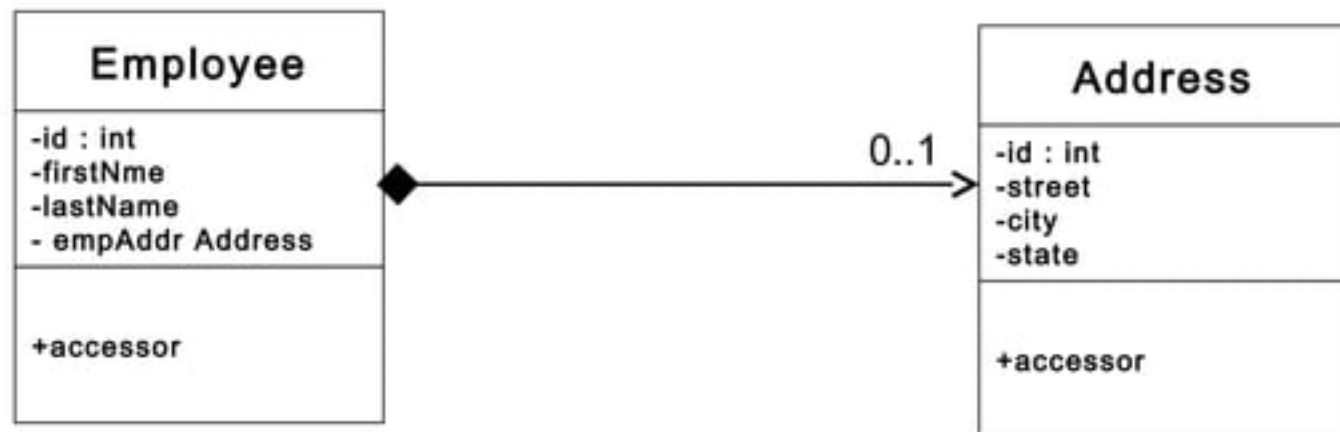
```
Set bids = new HashSet();  
bids.add(bid1);
```

```
auctionItem.setBids(bids);
```

Add Auction (Contd)

```
SessionFactory sessionFactory = new  
    Configuration().configure().buildSessionFactory();  
Session s = sessionFactory.openSession();  
Transaction tx = s.beginTransaction();  
  
s.save(auctionItem);  
  
tx.commit();  
s.close();
```

One to One mapping



Employee.hbm.xml

```
<class name="Employee" table="Employee">
<id name="id" column="ID"></id>
...
<one-to-one name="empAddr" cascade="all" class="Address" />
</class>
```


Dirty Check

- ❑ Retrieve an AuctionItem and change description
- ❑ String newDescription = "XYZ";
- ❑ Session session = sessionFactory.openSession();
- ❑ Transaction tx = s.beginTransaction();
- ❑ AuctionItem item =
 (AuctionItem) session.get(AuctionItem.class, 1);
- ❑ **item.setDescription(newDescription);**
- ❑ tx.commit();
- ❑ session.close();

Transitive Persistence

Retrieve an AuctionItem and create a new persistent Bid

- ☐ Bid bid = new Bid();
- ☐ bid.setId(1);
- ☐ bid.setAmount(bidAmount);

- ☐ Session session = sf.openSession();
- ☐ Transaction tx = session.beginTransaction();

- ☐ AuctionItem item=(AuctionItem) session.get(AuctionItem.class, 1);
- ☐ **Set s = item.getBids();**
- ☐ **s.add(bid);**

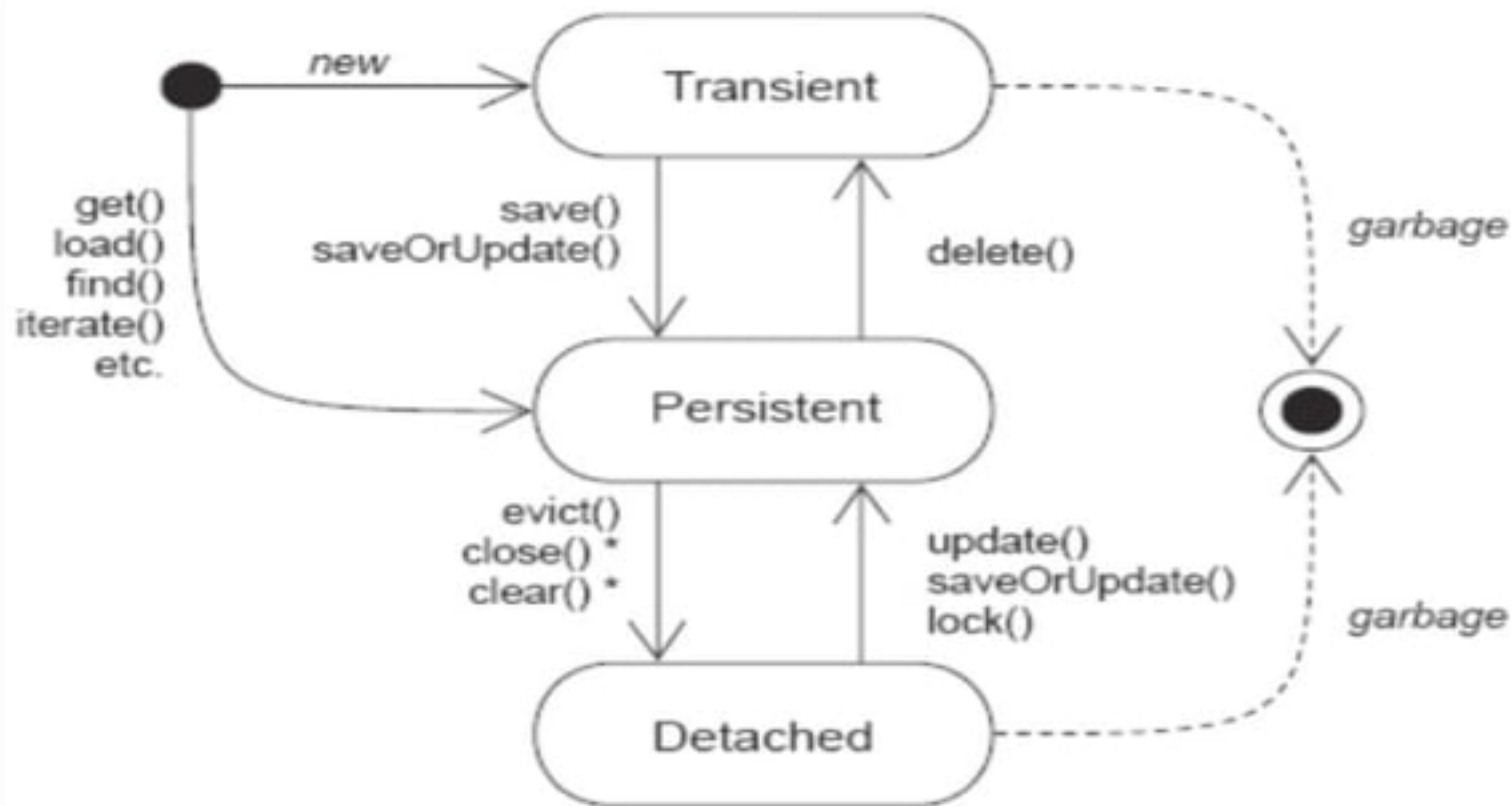
- ☐ tx.commit();
- ☐ session.close();

Detachment

Retrieve an AuctionItem and create a new persistent Bid

- ☐ Session session = sf.openSession();
- ☐ AuctionItem item =(AuctionItem) session.get(AuctionItem.class, 1);
- ☐ session.close();
- ☐ item.setDescription(newDescription);
- ☐ Session session2 = sf.openSession();
- ☐ Transaction tx = session2.beginTransaction();
- ☐ session2.update(item);
- ☐ tx.commit();
- ☐ session2.close

Object States



* affects all instances in a Session

Object States

- ❑ **Transient** : The instance is not, and has never been associated with any Session. It has no persistent identity (primary key value).
- ❑ **Persistent** :The instance is currently associated with a Session. It has a persistent identity (primary key value) and, perhaps, a corresponding row in the database. For a particular Session, Hibernate *guarantees* that persistent identity is equivalent to Java identity (in-memory location of the object).
- ❑ **Detached** :The instance was once associated with a Session, but that Session was closed, or the instance was serialized to another process. It has a persistent identity and, perhaps, a corresponding row in the database.

Optimizing Data Access

❑ Lazy Fetching (On Demand)

- Load on demand
- `<set name="bids" cascade="all" lazy="true">`

❑ Eager (Outer Join) Fetching

- `<set name="bids" cascade="all" fetch="join">`
- Load entire graph once
- Allows a whole graph of objects connected by many-to-one, one-to-many, many-to-many and one-to-one associations to be retrieved in a single SQL SELECT

❑ Batch Fetching

- Hibernate retrieves a batch of entity instances or collections in a single SELECT, by specifying a list of primary keys or foreign keys.

Transparent Lazy Fetching

- ❑ AuctionItem item = (AuctionItem) session.get(AuctionItem.class, 1);
- ❑ SELECT ... FROM AUCTION_ITEM WHERE ID = 1
- ❑ Set bids = item.getBids().;
- ❑ //Iterator iter = bids.iterate();
- ❑ SELECT ... FROM BID WHERE ITEM_ID = ?

Transparent Eager Fetching

- ❑ AuctionItem item = (AuctionItem) session.get(AuctionItem.class, 1);
- ❑ SELECT i.*, b.* FROM AUCTION_ITEM i, BID b WHERE ITEM_ID = 1 AND
- ❑ i.ID = b.ITEM_ID

Exception Handling

- ❑ It throws `HibernateException`
- ❑ `HibernateException` is a unchecked exception
- ❑

```
try {
```
- ❑

```
    transaction = session.beginTransaction();
```
- ❑

```
    session.save(sp);
```
- ❑

```
    transaction.commit();
```
- ❑

```
} catch (HibernateException e) {
```
- ❑

```
    transaction.rollback();
```
- ❑

```
} finally {
```
- ❑

```
    session.close();
```
- ❑

```
}
```

Cascade

- ❑ `<one-to-one name="person" cascade="all"/>`
- ❑ `<one-to-one name="person"
cascade="persist,delete,lock"/>`
- ❑ For each basic operation of the Hibernate session - including `persist()`, `merge()`, `saveOrUpdate()`, `delete()`, `lock()`, `refresh()`, `evict()`, `replicate()` - there is a corresponding cascade style. Respectively, the cascade styles are named **create**, **merge**, **save-update**, **delete**, **lock**, **refresh**, **evict**, **replicate**.
- ❑ Special value : **delete-orphan**

Cascade (Cont.)

- ❑ delete-orphan : applies only to one-to-many associations, and indicates that the delete() operation should be applied to any child object that is removed from the association.
- ❑ Cascade is recommended for <one-to-one> and <one-to-many> but not for <many-to-one> or <many-to-many> association

HQL

- ❑ Q - Get all parts
- ❑ SQL - SELECT * FROM part;
- ❑ Query query = session.createQuery("SELECT * FROM part");
- ❑ Query query = session.createQuery("from Part");

List list = query.list();

Iterator iter = list.iterator();

Part dto;

```
while (iter.hasNext()) {  
    dto =(Part)iter.next();  
    System.out.println(dto.getId() + "\t" + dto.getName()+ "\t\t"  
        +dto.getColor());  
}
```


HQL -Get all grey parts

- ❑ SQL - SELECT * FROM part WHERE color like 'Grey%';
- ❑ Query query = session.createQuery("from Part where color like 'Grey%'");

List list = query.list();

Iterator iter = list.iterator();

Part dto;

```
while (iter.hasNext()) {  
    dto =(Part)iter.next();  
    System.out.println(dto.getId() + "\t" + dto.getName()+ "\t\t"  
        +dto.getColor());  
}
```

HQL – Get some columns

- ❑ Q- Get all parts' ids, names
- ❑ `SELECT id, name FROM part;`
- ❑ `Query query = session.createQuery("select p.id, p.name from Part p");`
- ❑ `List rows = query.list();`

- ❑ `Iterator iter = rows.iterator();`

- ❑ `Object[] columns;`

- ❑ `System.out.println("ID\tName");`
- ❑ **while** (`iter.hasNext()`) {
 - ❑ `columns = (Object[])iter.next();`
 - ❑ `Integer id = (Integer) columns[0];`
 - ❑ `String name = (String) columns[1];`
 - ❑ `System.out.println(id + "\t" + name);`
- ❑ }

HQL-Joins

- ❑ Q- Get parts with their City of Unit
- ❑ `SELECT part.id, name, color, unit.city
FROM part, unit WHERE unit.id =
part.unit_id;`
- ❑ `Query query = session.createQuery("select
p.id, p.name, p.color, u.city from Part p, Unit
u where p.unitId = u.id");`

HQL – Order By

- ❑ Q- Get all grey parts sorted by name
- ❑ SQL – SELECT * FROM part WHERE color like 'Grey%' ORDER BY name;
- ❑ HQL – from Part where color like 'Grey%' order by name

HQL - Aggregate Functions

- ❑ Q- How many parts are there?
- ❑ `SELECT count(*) from part;`
- ❑ `Query query = session.createQuery("select count(*) from Part");`
- ❑ `List rows= query.list();`
- ❑ `Integer row = (Integer)rows.get(0);`

HQL – Group By

- ❑ Q- Get parts' color count
- ❑ SQL – SELECT color,count(*) FROM part GROUP BY color
- ❑ Query query = session.createQuery("select p.color,count(*) from Part p group by p.color");

Criteria

- ❑ Q – Get all parts
- ❑ SQL - SELECT * FROM part;
- ❑ Query query = session.createQuery("from Part");
- ❑ Criteria crit = session.createCriteria(Part.class);
- ❑ List list = crit.list();

Get all grey parts

- ❑ SQL - SELECT * FROM part WHERE color like 'Grey%';
- ❑ Query query = session.createQuery("from Part where color like 'Grey%'");
- ❑ Criteria crit = session.createCriteria(Part.class);
- ❑ crit.add(Restrictions.like("color", "Grey%"));
- ❑ crit.add(Restrictions.eq("unitId", "2"));
- ❑ List list = crit.list();

Criteria – selected attributes

- ❑ Q- Get all parts' ids, names
- ❑ SELECT id, name FROM part;
- ❑ HQL - select p.id, p.name from Part p
- ❑ Criteria crit = session.createCriteria(Part.class);
- ❑ ProjectionList p = Projections.*projectionList*();
- ❑ p.add(Projections.*property*("id"));
- ❑ p.add(Projections.*property*("name"));
- ❑ crit.setProjection(p);
- ❑ List list = crit.list();

Criteria – Aggregate Functions

- ❑ Q- How many parts are there?
- ❑ `SELECT count(id) from part;`
- ❑ `Query query = session.createQuery("select count(id) from Part");`
- ❑ `Criteria crit = session.createCriteria(Part.class);`
 - `ProjectionList projList = Projections.projectionList();`
 - `projList.add(Projections.count("id"));`
 - `projList.add(Projections.rowCount()); //count(*)`
 - `projList.add(Projections.groupProperty("color")); //Group by`
 - `crit.setProjection(projList);`
 - `List results = crit.list();`
- ❑ Other Aggregate Function
 - `Projections.max`
 - `Projections.min`
 - `Projections.avg`

Criteria – AND Condition

- ❑ Criteria crit =
 session.createCriteria(Part.class);
- ❑ crit.add(Restrictions.*like*("color", "Grey%"));
- ❑ crit.add(Restrictions.*eq*("unitId", "2"));
- ❑ Default is AND condition

Criteria – OR Condition

- ❑ Criteria crit = session.createCriteria(Part.class);
- ❑ crit.add(Restrictions.or(
 - Restrictions.like("color", "Grey%"),
 - Restrictions.eq("unitId", "2")
- ❑);

- ❑ crit.add(Restrictions.or(
 - Restrictions.eq("name", "Nut") ,
 - Restrictions.or(
 - Restrictions.like("color", "Grey%"), Restrictions.eq("unitId", "2")
- ❑);

Criteria

❑ Order By

- Criteria crit = session.createCriteria(Part.class);
- crit.add(Restrictions.like("color", "Grey%"));
- Crit.addOrder(Order.asc("name"));
- Crit.addOrder(Order.asc("color"));
- Crit.addOrder(Order.dsc("id"));

❑ Get Max records

- crit.setMaxResults(10);

❑ Get First few Record

- crit.setFirstResult(11)

❑ Create Aliases

- crit.createAlias("Part", "p")

Criteria - Join

- ❑ Applying the restrictions becomes easy in the case of joins as well. For example, the following query
- ❑ **SELECT O.*, P.* FROM ORDERS O, PRODUCT P
WHERE**
- ❑ **O.ORDER_ID=P.ORDER_ID AND P.ID='1111';**
- ❑ Would become
- ❑ **List orders = session.createCriteria(Order.class);**
- ❑ **orders.setFetchMode("products",FetchMode.JOIN);**
- ❑ **orders.add(Restrictions.eq("id","1111"));**
- ❑ **orders.list();**

Criteria – Group By

- ❑ **SELECT COUNT(ID) FROM ORDER HAVING PRICETOTAL>2000 GROUP BY ID**
- ❑ Can be rewritten in Criteria query as follows:
- ❑ **List orders = session.createCriteria(Order.class)
 .setProjection(Projections.projectionList()
 .add(Projections.count("id"))
 .add(Projections.groupProperty("id"))
)
 .list();**

DetachedCriteria

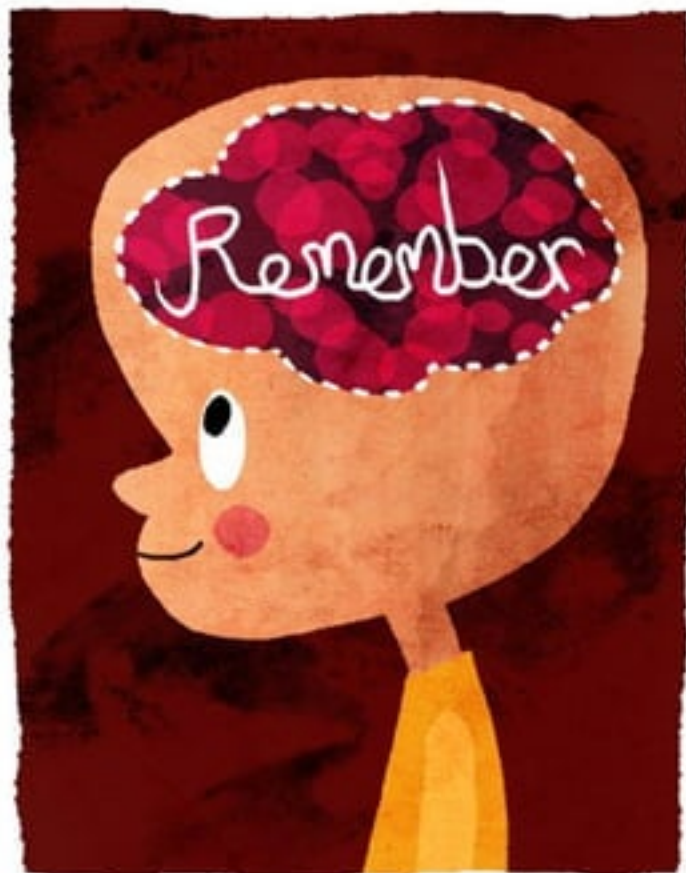
- Some applications need to create criteria queries in "detached mode", where the Hibernate session is not available. This class may be instantiated anywhere, and then a Criteria may be obtained by passing a session to `getExecutableCriteria()`. All methods have the same semantics and behavior as the corresponding methods of the Criteria interface.

DetachedCriteria

- ❑ DetachedCriteria dc =
DetachedCriteria.forClass(Student.class);
- ❑ dc.add(Restrictions.eq("name", "Gavin King"));
- ❑
- ❑ Session s1 = SessionFactory.openSession();
- ❑ dc.getExecutableCriteria(s1);
- ❑ List result = dc.list();
- ❑ s1.close();

- ❑ Session s2 = SessionFactory.openSession();
- ❑ dc.getExecutableCriteria(s2);
- ❑ result = dc.list();
- ❑ s2.close()

Cache



Cache

- ❑ Caching is used to optimize database applications.
- ❑ A cache reduces traffic between application and the database by conserving data already loaded from the database.
- ❑ The application may need to empty (invalidate) the cache from time to time if the database is updated in some way, because it has no way of knowing whether the cache is up to date.

Type of Cache

- ❑ Hibernate has 3 cache Level
 - Query Cache
 - First Level Cache
 - Second Level Cache

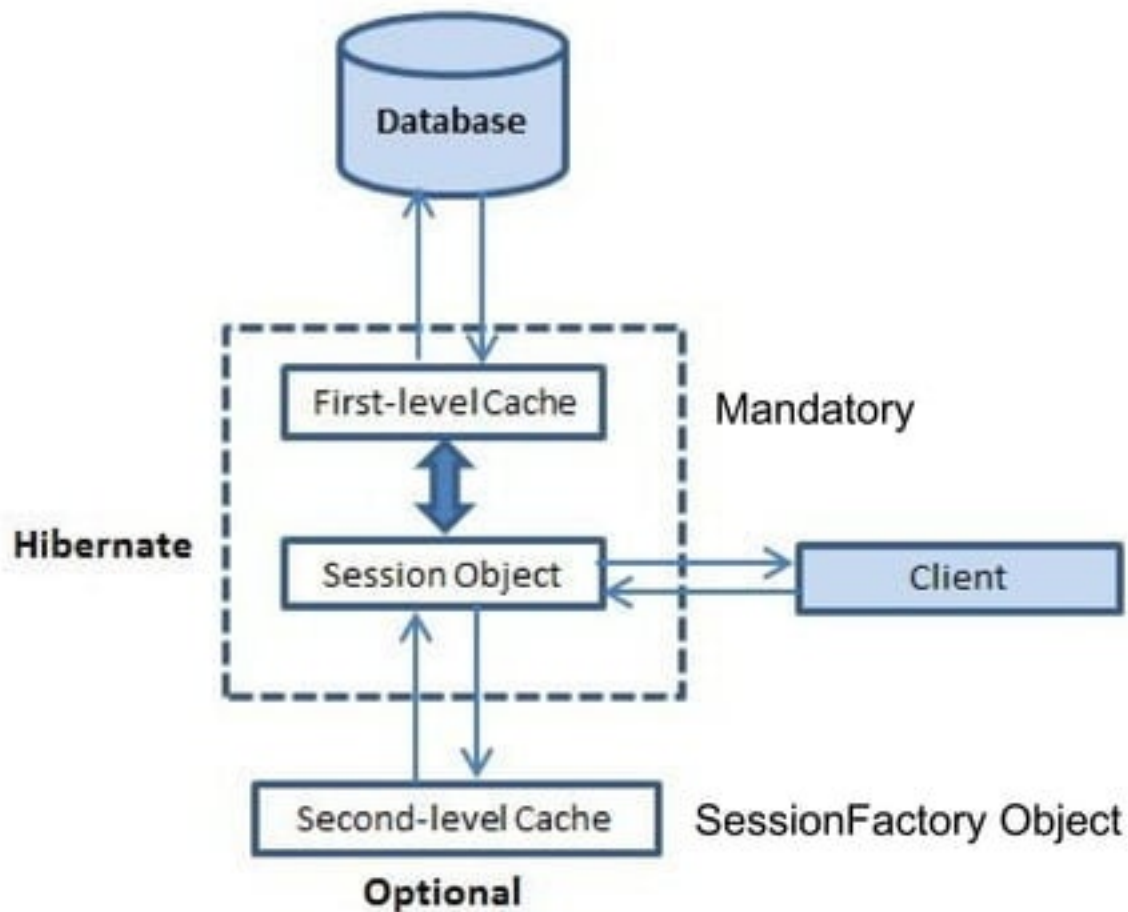
Query Cache

- ❑ Query result sets may be cached. It is useful when same queries are run frequently with the same parameters.
- ❑ To use the query cache enable it from hibernate.cfg.xml by
- ❑ **<property**
name="hibernate.cache.use_query_cache">
- ❑ **true </property>**
- ❑ `query.list();`
- ❑ `query.list(); // Same result will be fetched from Q Cache`

Query Cache (Cont.)

- ❑ The query cache can be forcefully refreshed by
 - ❑ `query.setCacheMode(CacheMode.REFRESH)`.
 - ❑ `query.list()`;

First & Second Level Cache



First Level Cache

- ❑ Associated with Session object.
- ❑ It is a transaction-level cache of persistent data. It is used by default.
- ❑ When an object is pass to `save()`, `update()` or `saveOrUpdate()` and retrieved by `load()`, `get()`, `list()`, `iterate()` or `scroll()`, that object is added to the internal cache of the Session.
- ❑ When `flush()` is subsequently called, the state of that object will be synchronized with the database

Remove from 1st Level

- ❑ An object is removed from first level cache using evict(dto) method.
- ❑ Method clear() can be removed all objects from first level cache.

Second Level Cache

- ❑ It is an optional cache and Stores objects across session.
- ❑ It is configured on a per-class and per-collection basis. It can cache a class or a collection
- ❑ An object is always searched in First-Level cache before locating in Second-Level cache.

Second Level Cache

- Use <class-cache> and <collection-cache> elements in hibernate.cfg.xml.
 - <class-cache class="org.dto.Item" usage="read-write"/>
 - <class-cache class="org.dto.Bid" usage="read-only"/>
 - <collection-cache collection="org.dto.Item.bids" usage="read-write"/>

Enable Second Level Cache

- ❑ By default it is disabled , you can enable it in hibernate.cfg.xml
- ❑ `<property`
name="hibernate.cache.use_second_level_cache">
 o true
- ❑ `</property>`

Named Queries

- ☐ <hibernate-mapping package="in.co.sunrays.dto">
- ☐ <class name="User" table="users">
- ☐ <id name="id" column="ID">
- ☐ <generator class="increment" />
- ☐ </id>
- ☐ <property name="lastName" column="LAST_NAME" />
- ☐ <property name="firstName" column="FIRST_NAME" />
- ☐ <property name="address" column="STREET" />
- ☐ <property name="userId" column="LOGIN_ID" />
- ☐ <property name="password" column="PASSWORD" />
- ☐ </class>
- ☐ <!-- Named Query -->
- ☐ <query name="allUser">
- ☐ <![CDATA[from User]]>
- ☐ </query>
- ☐ </hibernate-mapping>

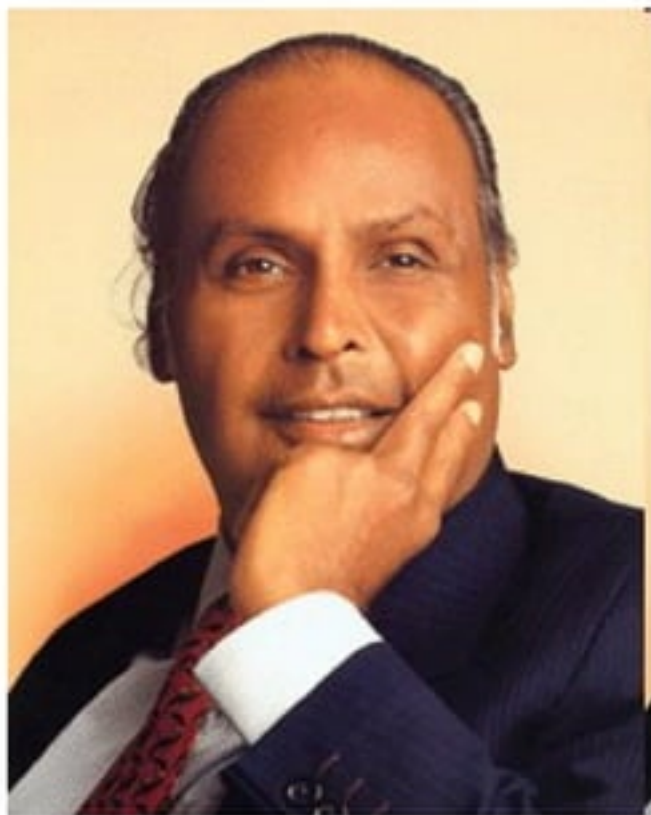
Named Query (Contd)

- ❑ `SessionFactory sessionFactory = new Configuration().configure()`
- ❑ `.buildSessionFactory();`
- ❑ `Session s = sessionFactory.openSession();`

- ❑ `Query q = s.getNamedQuery("allUser");`

- ❑ `List l = q.list();`
- ❑ `Iterator it = l.iterator();`

Inheritance



Inheritance

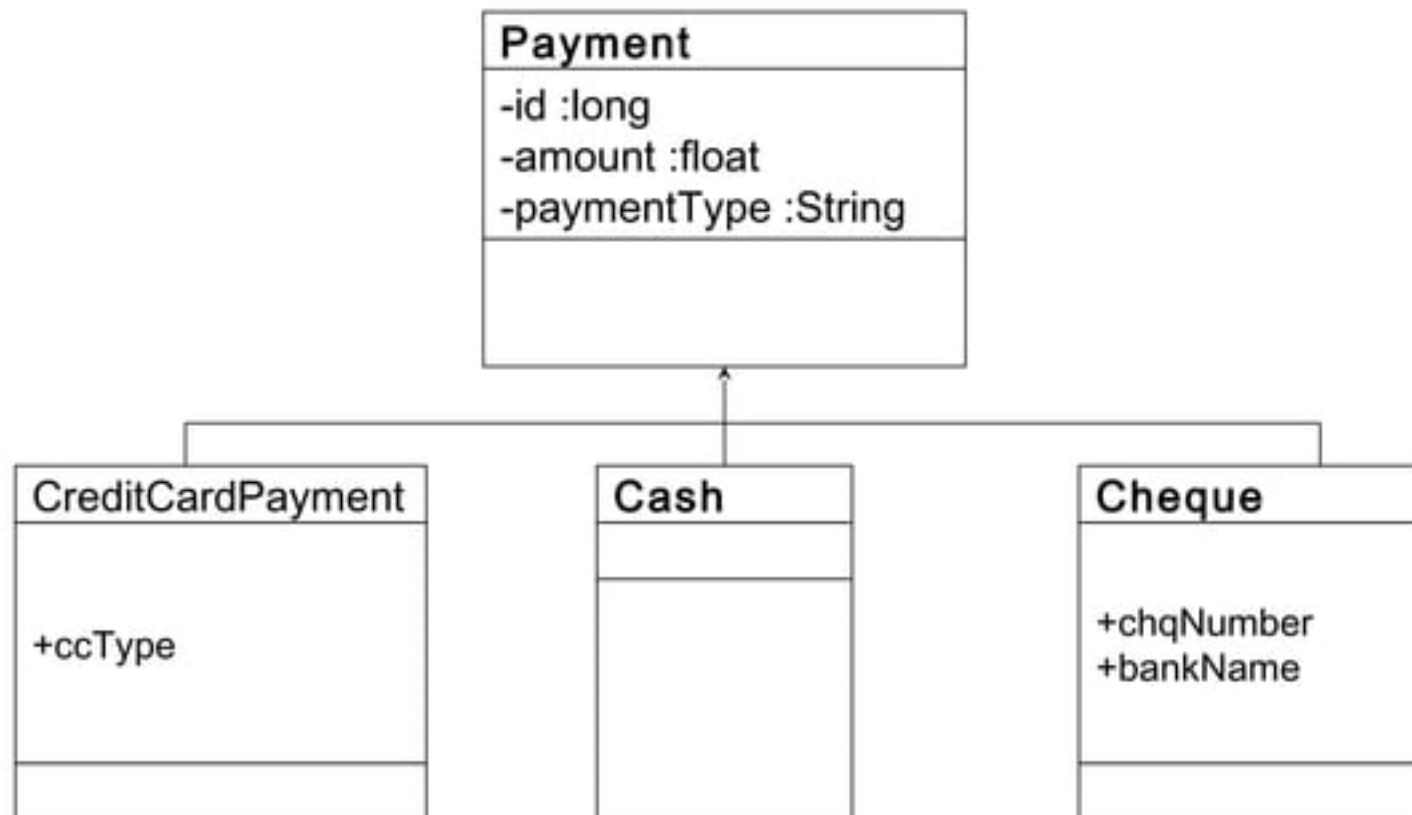


Table per class hierarchy

Payment

- PAYMENT_ID : LONG : PK
- AMOUNT : NUMBER
- PAYMENT_TYPE : VARCHAR
- CC_TYPE : VARCHAR
- CHQ_NUMBER : VARCHAR
- BANK_NAME : VARCHAR

```
SELECT payment_type, sum(amount) FROM Payment  
GROUP BY payment_type
```

Payment.hbm.xml

- ❑ `<class name="Payment" table="PAYMENT">`
- ❑ `<id name="id" type="long" column="PAYMENT_ID">`
- ❑ `<generator class="native"/>`
- ❑ `</id>`
- ❑ `<property name="amount" column="AMOUNT"/>`
- ❑ `<discriminator column="PAYMENT_TYPE" type="string"/>`
- ❑ `...`
- ❑ `<subclass name="CreditCardPayment" discriminator-value="CREDIT">`
- ❑ `<property name="ccType" column="CC_TYPE"/>`
- ❑ `...`
- ❑ `</subclass>`
- ❑ `<subclass name="CashPayment" discriminator-value="CASH">`
- ❑ `</subclass>`

Payment.hbm.xml (Cont)

- ❑ `<subclass name="ChequePayment" discriminator-value="CHEQUE">`
- ❑ `<property name="chqNumber " column=" CHQ_NUMBER"/>`
- ❑ `<property name="bankName " column=" BANK_NAME"/>`
- ❑ `</subclass>`
- ❑ `</class>`

Add Payment

- ☐ CreditCardPayment ccp = new CreditCardPayment();
- ☐ session.save(ccp);

- ☐ ChequePayment cp = new ChequePayment();
- ☐ session.save(cp);

- ☐ CashPayment cp = new CashPayment();
- ☐ session.save(cp);

Get Payment

- ☐ HQL = "from CreditCardPayment"
- ☐ HQL = "from ChequePayment "
- ☐ HQL = "from CashPayment "
- ☐ HQL = "from Payment "

Table per subclass

PAYMENT	CREDIT_PAYMENT
-PAYMENT_ID :LONG : PK -AMOUNT : NUMBER	-PAYMENT_ID :LONG : PK,FK -CC_TYPE : VARCHAR
CASH_PAYMENT	CHEQUE_PAYMENT
-PAYMENT_ID :LONG : PK,FK	-PAYMENT_ID :LONG : PK,FK -CHQ_NUMBER -BANK_NAME

Join Query

❑ **Q: Get all ChequeNumbers**

❑ `select chq_number from cheque_payment`

❑ **Q: Get all Cheque Number and their amounts**

❑ `select chq_number, amount from Payment p,
cheque_payment cp
where p.payment_id = cp.payment_id`

❑ `select amount from Payment, cheque_payment where
payment.payment_id = cheque_payment.payment_id`

❑ `select sum(amount) from Payment, cheque_payment where
payment.payment_id = cheque_payment.payment_id`

Payment.hbm.xml

- ❑ `<class name="Payment" table="PAYMENT">`
- ❑ `<id name="id" type="long" column="PAYMENT_ID">`
- ❑ `<generator class="native"/>`
- ❑ `</id>`
- ❑ `<property name="amount" column="AMOUNT"/>`
- ❑ `...`
- ❑ `<joined-subclass name="CreditCardPayment"`
`table="CREDIT_PAYMENT">`
- ❑ `<key column="PAYMENT_ID"/>`
- ❑ `<property name="creditCardType" column="CCTYPE"/>`
- ❑ `...`
- ❑ `</joined-subclass>`

Payment.hbm.xml

- ❑ `<joined-subclass name="CashPayment" table="CASH_PAYMENT">`
- ❑ `<key column="PAYMENT_ID"/>`
- ❑ `...`
- ❑ `</joined-subclass>`

- ❑ `<joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">`
- ❑ `<key column="PAYMENT_ID"/>`
- ❑ `...`
- ❑ `</joined-subclass>`
- ❑ `</class>`

Table per subclass, using a discriminator

PAYMENT	CREDIT_PAYMENT
-PAYMENT_ID :LONG : PK -PAYMENT_TYPE : VARCHAR -AMOUNT : NUMBER	-PAYMENT_ID :LONG : PK,FK -CC_TYPE : VARCHAR
CASH_PAYMENT	CHEQUE_PAYMENT
-PAYMENT_ID :LONG : PK,FK	-PAYMENT_ID :LONG : PK,FK -CHQ_NUMBER -BANK_NAME

Payment.hbm.xml

- ❑ `<class name="Payment" table="PAYMENT">`
- ❑ `<id name="id" type="long" column="PAYMENT_ID">`
- ❑ `<generator class="native"/>`
- ❑ `</id>`
- ❑ `<discriminator column="PAYMENT_TYPE" type="string"/>`
- ❑ `<property name="amount" column="AMOUNT"/>`
- ❑ `...`
- ❑ `<subclass name="CreditCardPayment" discriminator-value="CREDIT">`
- ❑ `<join table="CREDIT_PAYMENT">`
- ❑ `<key column="PAYMENT_ID"/>`
- ❑ `<property name="creditCardType" column="CCTYPE"/>`
- ❑ `</join>`

Payment.hbm.xml

- ❑ `<subclass name="CashPayment" discriminator-value="CASH">`
- ❑ `<join table="CASH_PAYMENT">`
- ❑ `<key column="PAYMENT_ID"/>`
- ❑ `</subclass>`


Joins

- ☐ Outer Join
- ☐ Inner Join
- ☐ Equ Joins
- ☐ Left Join
- ☐ Right Join

Alternate Technologies

- ☐ JDO – Java Data Object
 - ☐ EJB – CMP (Container Managed Persistence)
 - ☐ JPA (Java Persistence API)
 - ☐ TopLink
 - ☐ iBATIS
-
- ☐ Note : No one replaces JDBC

Assignment – Dealer Mgt

**Bharat Petroleum Corporation Limited**, Manmad Installation, Panewadi
Security Exchange- Tank Lorry Gate

Dealer Entry | Transporter Entry | Lorry Entry | Driver Entry | Cleaner Entry | Reports | Others | Login

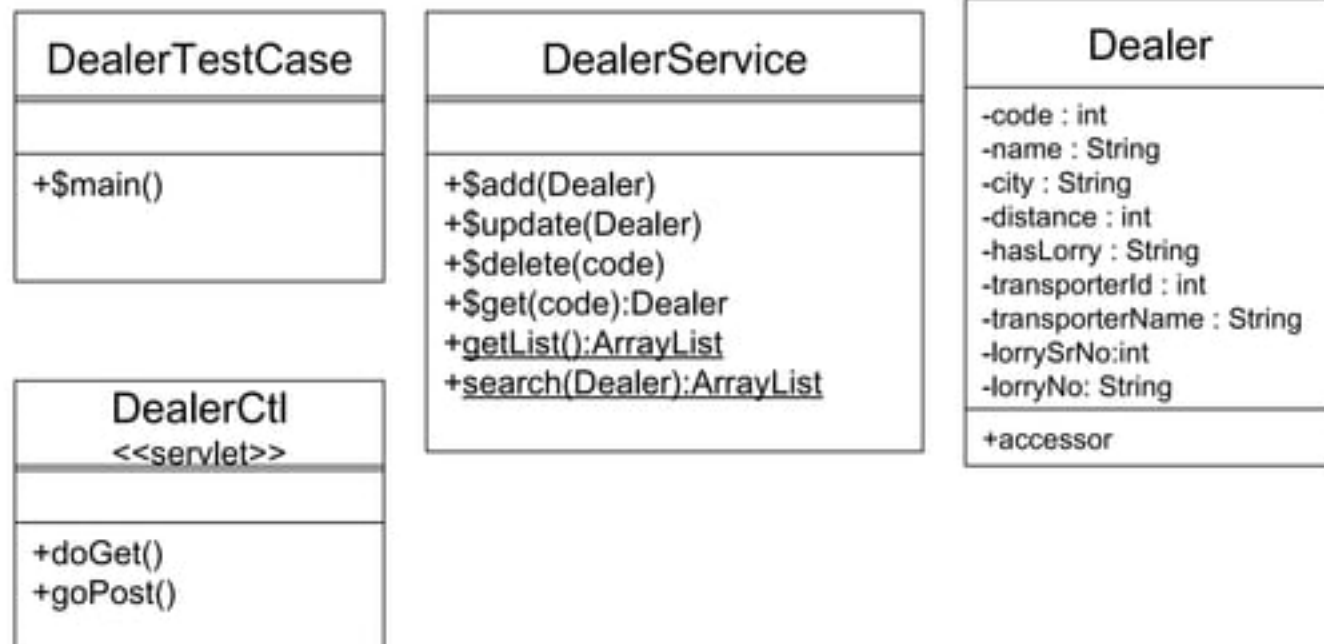
Enter Customer Code for Availability

Customer Details
Customer Code Customer Name
Location (City) Distance (Km)
Has A Lorry ? ☐ Yes ☐ No

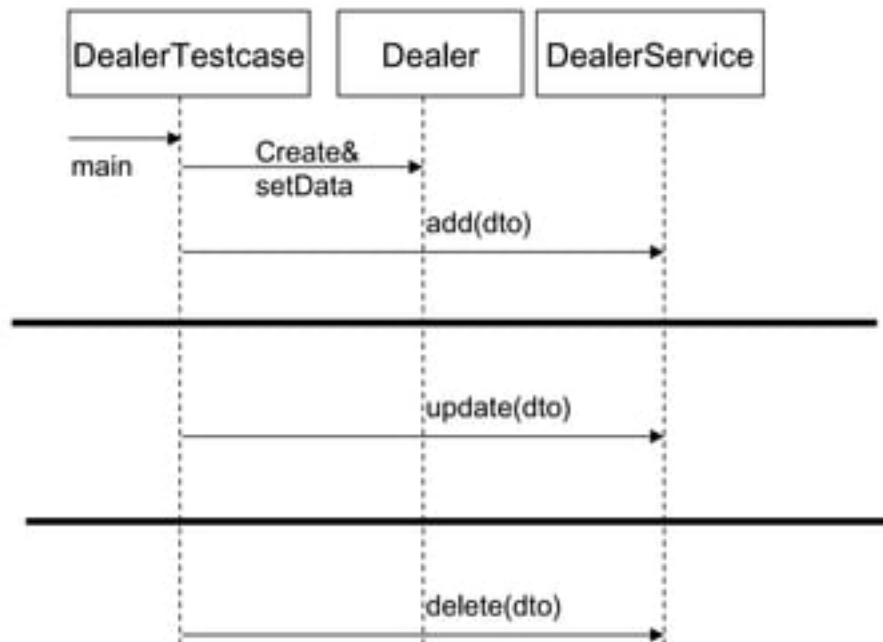
Transporter Details
Transporter Id Transporter Name

Lorry Details
Sr No Lorry No

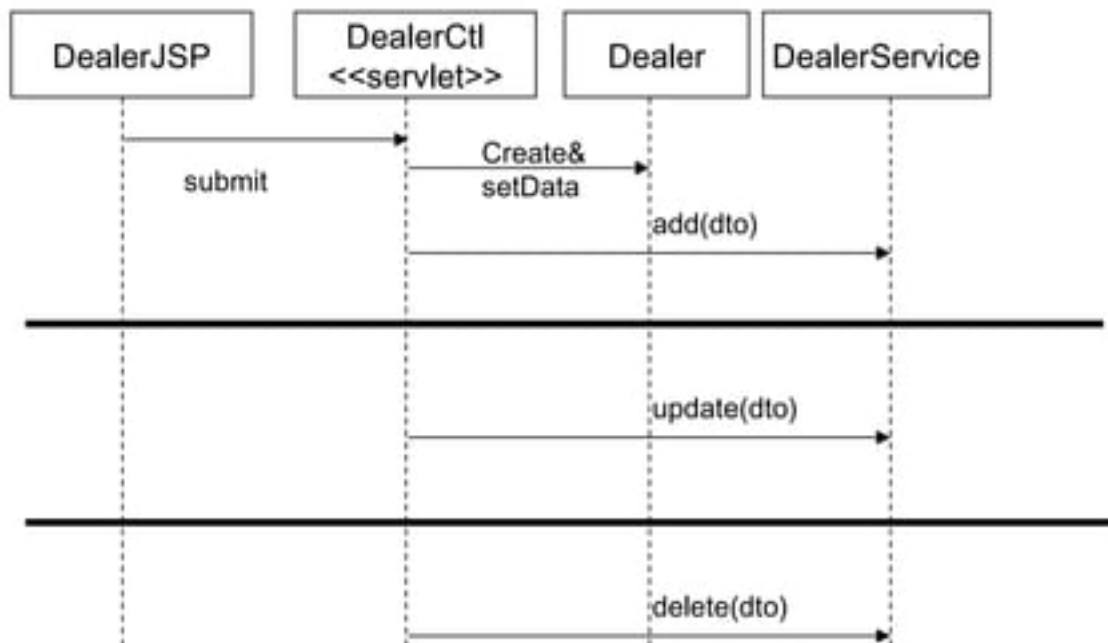
Class Diagram



Sequence Diagram - 1



Sequence Diagram - 2



Disclaimer

- ❑ This is an educational presentation to enhance the skill of computer science students.
- ❑ This presentation is available for free to computer science students.
- ❑ Some internet images from different URLs are used in this presentation to simplify technical examples and correlate examples with the real world.
- ❑ We are grateful to owners of these URLs and pictures.

Thank You!

GET IN TOUCH



www.SunilOS.com