Aprameyah
Technologies Pvt. Ltd.

Hibernate Essentials

Aprameyah
Technologies Pvt. Ltd.

About Me

- Maruthi R Janardhan
  - Been doing java since jdk 1.2
  - Been with IBM, ANZ, HCL-HP, my own startup Leviossa..
  - Total 16 years programming C, C++, Java, Ruby, Perl, Python, PHP, etc

Aprameyah
Technologies Pvt. Ltd.

ORM

- Object Relational Mapping
  - Does all the plumbing work
  - Acts as a layer of abstraction over JDBC and keeps the code queries database independent (ideally)
  - Allows for a lot of easy programming fragments
  - Special patterns allow loading only whats needed

Aprameyah Technologies Pvt. Ltd.

Comparison

```
Statement statement =
getCon().createStatement();
ResultSet rs =
statement.executeQuery("select * from
users");
List<User> users = new
ArrayList<User>();
while(rs.next()){
    User u = new User();
    u.setId(rs.getInt("id"));
    u.setAge(rs.getInt("age"));
u.setEmailId(rs.getString(“email_id"));
u.setJoinDate(rs.getDate(“join_date"));
    u.setName(rs.getString(“name"));
u.setPassword(rs.getString(“password”))
;
u.setState(rs.getString("state"));
       users.add(u);
}
return users;
```

```
session.createQuery("select u from
User u").list();
```

Aprameyah Technologies Pvt. Ltd.

Standard SQL

Derby

```
CREATE TABLE users
(
    id INTEGER NOT NULL GENERATED
    ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
    name varchar(100),
    email_id varchar(100) ,
    password varchar(100) ,
    join_date timestamp ,
    age int  ,
    state varchar(2) ,
    CONSTRAINT primary_key_users PRIMARY KEY (id)
)
```

MySql

```
CREATE TABLE `users` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `email` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `password` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `updated_at` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `remember_token` varchar(100) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `users_email_unique` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Aprameyah
Technologies Pvt. Ltd.

Hibernate Dialects

```
<property
name=“hibernate.dialect">org.hibernate.dialect.DerbyDialect</
property>

OR

<property
name="hibernate.dialect">org.hibernate.dialect.MySqlDialect</
property>
```

Aprameyah
Technologies Pvt. Ltd.

Loading Relationships

```
List<Users> userList = session.createQuery("select u from User
u").list();
for (Users u : userList) {
   Set<Orders> orders = u.getOrders();
}
```

Aprameyah
Technologies Pvt. Ltd.

Configuring Session Factory

```
<hibernate-configuration>
 <session-factory>
  <property
name="hibernate.connection.driver_class">org.apache.derby.jdbc.ClientDri
ver</property>
  <property name="hibernate.connection.password">app</property>
  <property name="hibernate.connection.url">jdbc:derby://localhost:
1527//Users/maruthir/Documents/Training/workspace/CRUD/WebContent/WEB-
INF/mydb</property>
  <property name="hibernate.connection.username">APP</property>
  <property
name="hibernate.dialect">org.hibernate.dialect.DerbyDialect</property>
  <property name="hibernate.show_sql">true</property>
  <mapping resource="com/mydomain/model/Users.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

Aprameyah
Technologies Pvt. Ltd.

Mapping Objects

```
<hibernate-mapping>
   <class name="com.mydomain.model.Users" table="USERS" schema="APP">
      <id name="id" type="int">
         <column name="ID" />
         <generator class="native" />
      </id>
      <property name="name">
         <column name="NAME"/>
      </property>
      <property name="emailId">
         <column name="EMAIL_ID"/>
      </property>
</hibernate-mapping>
```

Aprameyah
Technologies Pvt. Ltd.

Loading SessionFactory And Querying

```
Configuration configuration = new
Configuration().configure("hibernate.cfg.xml");
        ServiceRegistry serviceRegistry
            = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();
SessionFactory sessionFactory =
configuration.buildSessionFactory(serviceRegistry);

Session session = sessionFactory.openSession();
List<Users> userList = session.createQuery("select u from User
u”).list();
System.out.println(userList);
session.close();
```

Aprameyah
Technologies Pvt. Ltd.

Annotation Based Config

- Instead of mapping hbm.xml, the configuration can be done with annotations too

- Mapping resource change in hibernate.cfg.xml:

```
<mapping class="com.mydomain.model.Users"/>
```

- Mapping on entity

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

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;
    @Column(name = "EMAIL_ID", nullable = true, length=100)
    private String emailId;
    @OneToMany
    private Set<Orders> orders;
```

Aprameyah
Technologies Pvt. Ltd.

Saving Data

- Create a new User object, set all fields except the ID

- call session.save()

- Wrap it in a transaction

```
Transaction tx= session.beginTransaction();
tx.commit();
```

Updating Data

- Create an User object
- Set all fields including the ID field
- call session.update() wrapped in a transaction

Aprameyah
Technologies Pvt. Ltd.

An Experiment

- Open a session and start a transaction
- Load all users and print them
- Now create an user object and set all fields including id for updating
- call session.update
- Commit transaction

Aprameyah
Technologies Pvt. Ltd.

Session Cache

- Session acts like a first level cache
- Session.evict removes an object from session cache
- Session.clear removes all objects from session
- Create a new object, save it with Session.save, then modify the object and commit the transaction

Aprameyah
Technologies Pvt. Ltd.

More About Session

- Create a new object, save it with Session.save, then modify the object, evict the object using session.evict() and commit the transaction

- Create a new object, save it with Session.save, then modify the object, flush using session.flush(), evict the object using session.evict() and commit the transaction

Problems With Session Cache

- Session cache can cause memory issues in long running operations with heavy data.
- Flush and clear the session often in long running operations

Aprameyah
Technologies Pvt. Ltd.

Session Behaviour

- Open two sessions, start two transactions, read an user object using session.load(User.class,id) in both sessions
- Modify age field in both objects
- Commit both transactions
- Re-run the same program

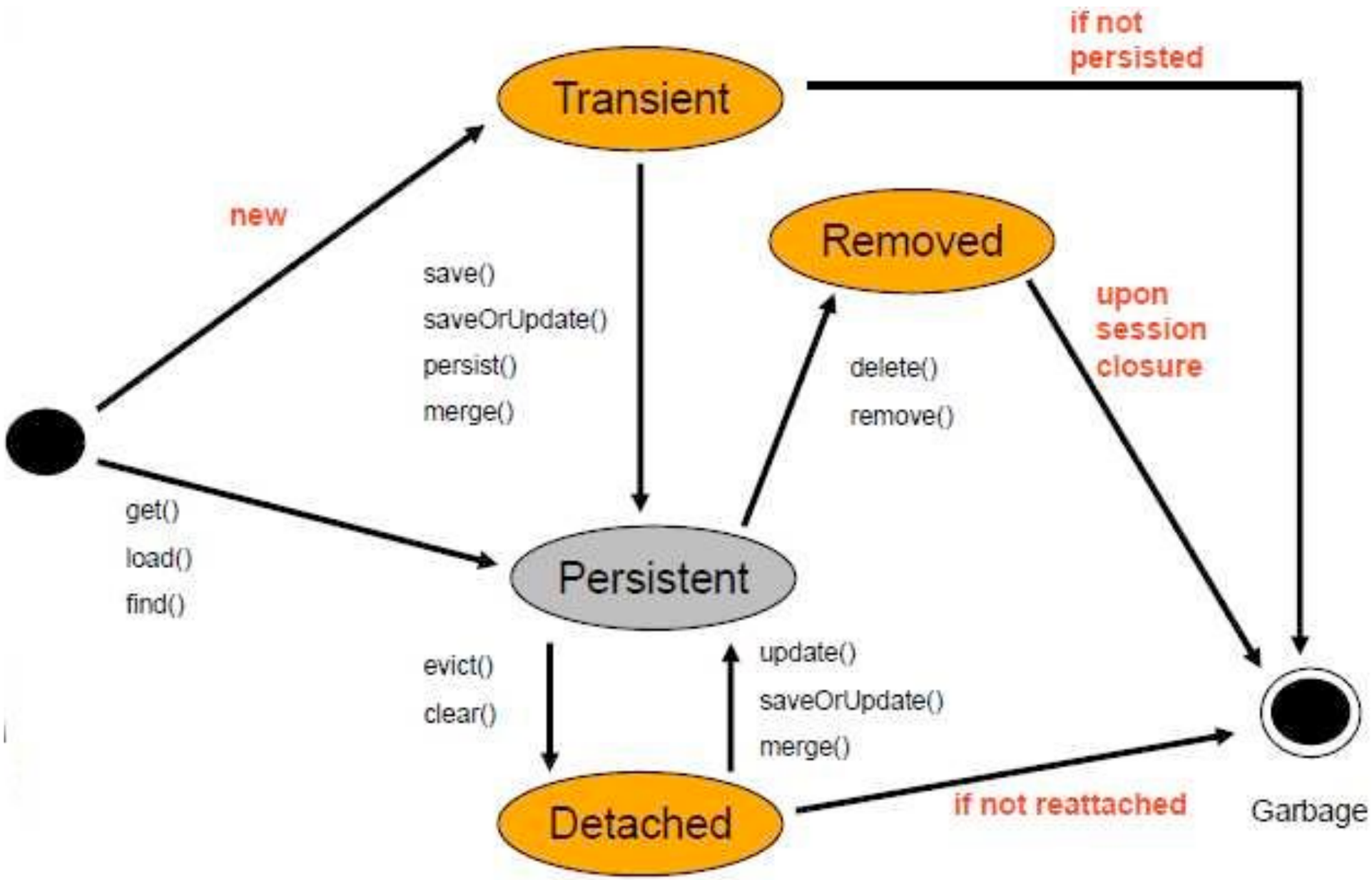
Session Cache

- Session keeps a copy of every loaded object in its original form and knows when an object is modified by means of comparing with existing objects at the time of commit.

- If the object has not changed, there is no update done

Aprameyah
Technologies Pvt. Ltd.

Bean Life Cycle



Aprameyah
Technologies Pvt. Ltd.

Transaction Isolation

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	Permitted	Permitted	Permitted
READ COMMITTED	--	Permitted	Permitted
REPEATABLE READ	--	--	Permitted
SERIALIZABLE	--	--	--

- <property name="hibernate.connection.isolation">2</property>
- 1: READ UNCOMMITTED
- 2: READ COMMITTED
- 4: REPEATABLE READ
- 8: SERIALIZABLE

Aprameyah
Technologies Pvt. Ltd.

Isolation Behaviour 1

- Open two sessions, start two transactions, read an user object using session.load(User.class,id) in both sessions
- Modify age field in both objects
- Commit both transactions
- Repeat the program with isolation setting 8,4,2

```
<property name="hibernate.connection.isolation">8</property>
```

Aprameyah
Technologies Pvt. Ltd.

Isolation Behaviour 2

- Open two sessions, start two transactions, select all users using

```
List<Users> userList = session.createQuery("select u from User
u”).list();
```

- Create a new user object and call session.save in any one session
- Commit both transactions
- Repeat the program with isolation setting 8,4,2

```
<property name="hibernate.connection.isolation">8</property>
```

Aprameyah
Technologies Pvt. Ltd.

Optimistic Locking

- Use a lower isolation level but prevent objects from being overwritten in other transactions
- Map the PRODUCTS table to Product entity
- Map the version column after id column in hbm like this: <version column="version" name="version" type="int" insert="false" />
- Repeat the isolation behaviour 1 program with isolation setting removed

Aprameyah
Technologies Pvt. Ltd.

Mapping Relationships

- One to Many
- Many to Many
- Many to One
- One to One

Aprameyah
Technologies Pvt. Ltd.

One to Many

- Map the order table's fields to an Order entity with a hbm.xml

- Define a "Set<Order> orders" property in User entity

- In User.hbm.xml mention this relationship

```
<set name="orders">
   <key column="order_id" not-null="true" />
   <one-to-many class="com.mydomain.model.Orders" />
</set>
```

Aprameyah
Technologies Pvt. Ltd.

Many to One

- Define a property “private User user” in Order class

- Define the Many to One relationship

```
<many-to-one name="user" column="user_id"
                class="com.mydomain.model.Users" not-null="true"/>
```

Aprameyah
Technologies Pvt. Ltd.

Many to Many

- Map the “Products” table to Product entity in hbm.xml
- In Order entity create a “Set<Product> products” property
- Map the many to many relationship in orders.hbm.xml using the intermediate table

```
<set name="products" table="PRODUCT_ORDER_LNK">
    <key column="order_id" />
    <many-to-many column="product_id" class="com.mydomain.model.Products" />
</set>
```

- Map the reverse relationship from Product to order aswell

Aprameyah
Technologies Pvt. Ltd.

One to One

- Map the Billing_info table to BillingInfo entity in hbm.xml
- In Order create a “BillingInfo billingInfo” property
- Then provide the mapping in order.hbm.xml

```
<one-to-one name="billingInfo" class="com.mydomain.model.BillingInfo"
        cascade="all" foreign-key="order_id"></one-to-one>
```

Transform the WebApp to Hibernate

- A WebApp implemented with JDBC is shared (HibernateWebApp). Transform that to use hibernate.

- Create a class HibernateUtil that loads the SessionFactory from configuration.

- Bring in User<—>Order relationship and in User::getOrderCount, return orders.size();

- Change getAllUsers method to use hibernate

Aprameyah Technologies Pvt. Ltd.

Basic HibernateUtil

```
public class HibernateUtil {

   private static SessionFactory sesFac = null;
   static{
       Configuration configuration = new
Configuration().configure("hibernate.cfg.xml");
        ServiceRegistry serviceRegistry
            = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();
       sesFac = configuration.buildSessionFactory(serviceRegistry);
   }

   public static Session getSession(){
      return sesFac.openSession();
   }
}
```

Aprameyah
Technologies Pvt. Ltd.

Lazy Initialisation

- Works on one to many, many to many and many to one relationships

- Javassist (Java programming assistant) is a Java library providing a means to manipulate the Java bytecode of an application.

  - Support for structural reflection, i.e. the ability to change the implementation of a class at run time.

Aprameyah
Technologies Pvt. Ltd.

Open Session-in-view

- Design Pattern for solving lazy loading issues in Web apps
- A filter opens a session and sets it in a thread-local
- DAO uses the session from thread-local

Aprameyah Technologies Pvt. Ltd.

Filter For This Pattern

```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws
IOException, ServletException {
    Transaction tx=null;
    HttpServletRequest request = (HttpServletRequest) req;
    try {
        //Begin and Commit Transaction
        Session ses = HibernateUtil.currentSession();
        tx = ses.beginTransaction();
        chain.doFilter(req, res);
        tx.commit();
    } catch (Exception e) {
        tx.rollback();
        throw new ServletException(e);
    } finally {
        HibernateUtil.closeSession();
    }
}
```

```
<filter>
    <filter-name>HibernateSessionFilter</filter-name>
    <filter-class>com.mydomain.HibernateSessionFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>HibernateSessionFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Hibernate Interceptors

- Interceptors in hibernate are fired for various events in the session and session factory

- There can be an interceptor for the entire session factory or one for each session.

- Create a class that extends EmptyInterceptor and override onSave() method. Update the join date of a user whenever an user is saved

- Also prevent users from saving string data with leading/trailing spaces

- Override the onFlushDirty method to update the modified date each time an user is updated