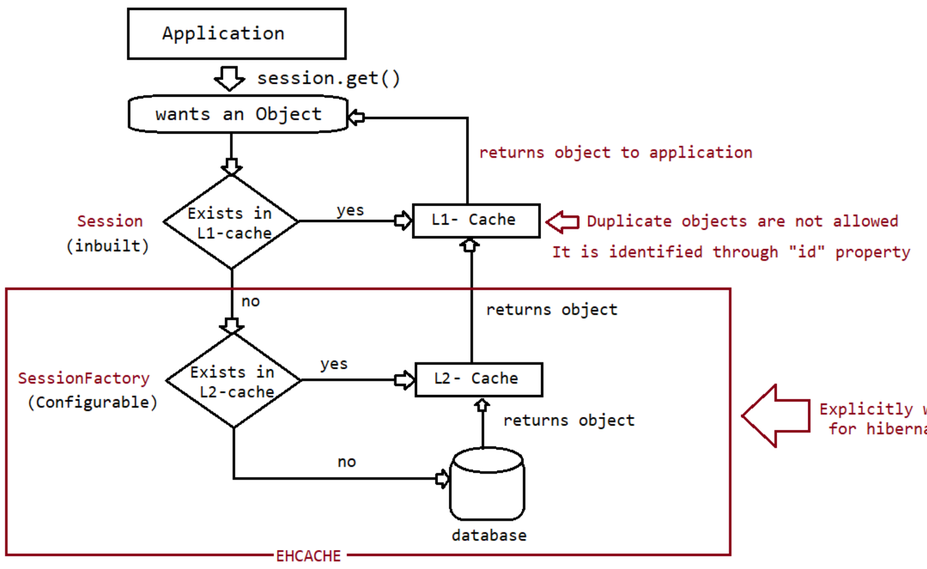
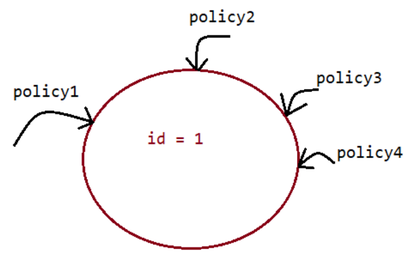
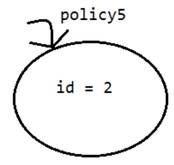
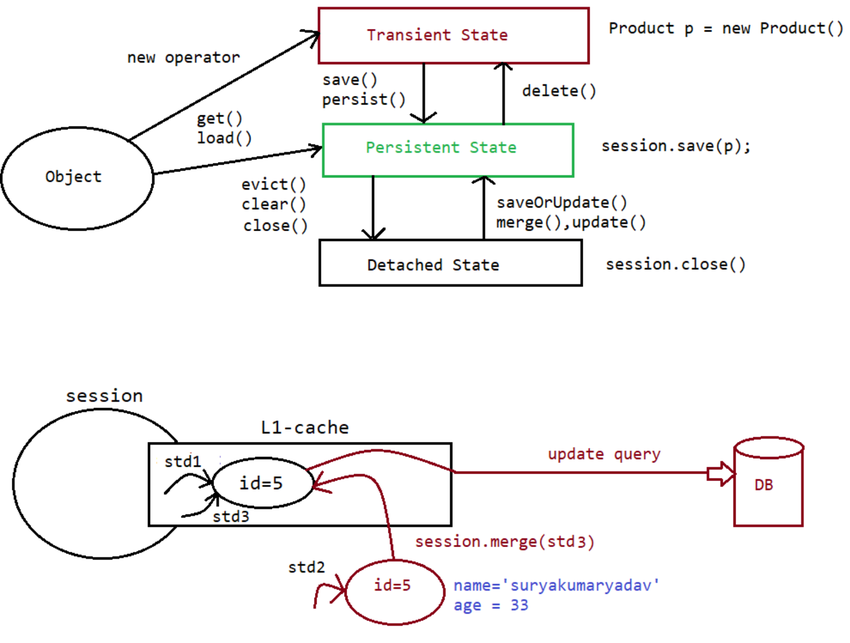






Explicitly we need to configure this cache for hibernate otherwise it won't be available.

Entity Object Life Cycle



Tommo and day after tommo(09/03/2023) => holiday
Extra class :: 11/03/2023(saturday) ==> Timings 7.30pm to 11.00pm
==========

Object Time Stamping
==========
=> It allows to keep track of Object is saved and Object is lastly updated.
usecase: Keep track of when the back account is opened and lastly updated.

=> We use 2 annotations like
1. @CreationTimeStamp => To hold object/record creation date and time.
2. @UpdateTimeStamp => To hold when the object is lastly modified /updated.

```
@CreationTimeStamp
LocalDateTime openingDate;

@UpdatetimeStamp
LocalDateTime lastUpdate;
```

=> In annotation driven environment we can apply both Versioning and TimeStamp feature.

refer:: HB-16-HibernateTimeStampingApp

Caching in hibernate
==========
=> It is most important feature/benefit of hibernate.
=> It reduces the no of trips b/w application and database, it improves the performance of the application.
=> Hibernate maintains cache at 2 levels, so it improves the peformance.
=> When an application wants the data from the database, then hibernate looks for the object at the L1-cache,if not then
it looks for the object at L2-cache and if not then it would go for "database".

Methods associated with cache are
a. session evict(object) => To remove particular object from cache.
b. session.clear() => To clean the cache or to remove all objects from cache.
c. session.contains(object) => To check particular object exists in the cache.

L1Cache(inbuilt cache)
This cache is associated with session object, we can't disable it as it is inbuilt present in Session object.

refer:: HB-17-HibernateCachingApp

L2Cache(configurable cache)
It is associated with SessionFactory object,so it is called as "GlobalCache".
It is configurable cache which we can enable/disable.
We have multiple providers supporting L2Cache
a. EH_cache(It support inmemory and diskcaching)
b. swarmcache
c. oscache

L2Cache Concurency strategies
==========
1. read-only : caching will work for read only operation
2. nonstrict-read-write : caching will work for read and write but only at a time.
3. read-write : caching will work for read and write simultaneously.

4. transactional : caching will work for transaction.

      In Annotation driven environment, keep the annotation at the top most class level

```
            @Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
```

hibernate.cfg.xml

```
  <!-- For each cache -->
  <property name="cache.use_second_level_cache">true</property>
  <property name="net.sf.ehcache.configurationResourceName">ehcache.xml</property>
  <property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
  <property name="hibernate.cache.use_query_cache">true</property>
```

ehcache.xml

```
<ehcache>
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
        maxElementsInMemory="100"
        eternal="false"
        timeToIdleSeconds="10"
        timeToLiveSeconds="30"
        overflowToDisk="true"
     />
</ehcache>
```

refer:: HB-18-HibernateCaching-L2

Entity Object Life Cycle diagram/3 states of Entity Object

1. Transient State
2. Persistent State/Attached State
3. Detached State

Transient state

 Here the object of the Entity not bound or linked with Session object.
 Here the object don't have id value and it doesn't hold /represent any Db table data.

```
            eg: Product p = null;
                Product p = new Product();
```

Persistent State

 Here the object is linked with Session.its placed in L1 cache of Session Object.
 Contains id value and it represents the record in the db table having synchornization.
 All persistence operations takes place by bringing Object into this state.
            eg: save(),saveOrUpdate(),load(),get(),persist()

Detached State

 Previously persistent ,but not now.
 Contains id value, but does not represents the record in the db table and it does not maintain synchornization also.

session.close(),session.clear(),session.evict()

```
eg: Product p =new Product();//p-> Transient state
            session.save(p);//p -> persistent state
          transaction.commit()
            session.close()//Detached state
```

What is the difference b/w session.saveOrUpdate() and session.merge()?

saveOrUpdate()

=> First select operation(based on id),if it is succesfull then it performs update operation

otherwise it would perform insert operation.

merge()

1. Version1=> (same as saveOrUpdate)

=> First select operation(based on id), if it exists, then it performs update operation

otherwise it would perform insertion operation.

2. Version2 => (upon we performing the loading explicitly,if we want to perform update operation)

refer:: HB-19-MergeOperationApp

Connection Pooling in Hibernate

=> SessionFactory object holds jdbc connection pool having set of readymade jdbc connection objects and uses them in creation of hibernate objects.

=> By default hibernate app uses hibernate built in jdbc connection pool which is not suitable for production environment,becoz of performance issues.

hibernate.cfg.xml

```
<!-- Connection provider to work with hikaricp -->
<property
name="connection.provider_class">org.hibernate.hikaricp.internal.HikariCPConnection
Provider</property>

 <!-- HikariCP settings -->
<property name="hikari.connectionTimeout">50000</property>
<property name="hibernate.hikari.minimumIdle">10</property>
<property name="hibernate.hikari.maximumPoolSize">20</property>
<property name="hibernate.hikari.idleTimeout">30000</property>
```

refer: HB-20-ConnectionPool