EJB - EXCEPTION HANDLING

EJB are part of enterprise applications which are normally distributed environment based. So apart from normal exceptions that can occur in code, in case of ejb, there can be exception like communication failure, security permissions, server down etc. EJB container considers exceptions in two ways.

- **Application Exception** - If business rule is voilated or exception occurs while executing the business logic.
- **System Exception**- Any exception which is not caused by business logic or business code. RuntimeException, RemoteException are SystemException. For example, error during ejb lookup.

How EJB Container handles exceptions?

When **Application Exception** occurs, ejb container intercepts the exception but returns the same to the client as it is. It does not roll back the transaction unless it is specified in code by EJBContext.setRollBackOnly method. EJB Container does not wrap the exception in case of Application Exception.

When **System Exception** occurs, ejb container intercepts the exception, rollbacks the transaction and start the clean up tasks. It wraps the exception into RemoteException and throws it to the client.

Handling Application Exception

Application exceptions are generally thrown in Session EJB methods as these are the methods responsible to execute business logic. Application exception should be declared in throws clause of business method and should be thrown in case business logic fails.

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   ...

   public List<Book> getBooks() throws NoBookAvailableException {
      List<Book> books =
         entityManager.createQuery("From Books").getResultList();
   if(books.size == 0)
  throw NoBookAvailableException
           ("No Book available in library.");
   return books;
   }
   ...
}
```

Handling System Exception

System exception can occur at any time like naming lookup fails, sql error occurs while fetching data. In such case such exception should be wrapped under EJBException and thrown back to the client.

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   ...

   public List<Book> getBooks() {
      try {
         List<Book> books =
            entityManager.createQuery("From Books").getResultList();
      } catch (CreateException ce){
```

```
            throw (EJBException) new EJBException(ce).initCause(ce);
      } catch (SqlException se){
            throw (EJBException) new EJBException(se).initCause(se);
         }
      return books;
      }
      ...
}
```

At client side, handle the EJBException.

```
public class EJBTester {
   private void testEntityEjb(){
   ...
   try{
      LibraryPersistentBeanRemote libraryBean =
      LibraryPersistentBeanRemote)ctx.lookup("LibraryPersistentBean/remote");

      List<Book> booksList = libraryBean.getBooks();
   } catch(EJBException e) {
      Exception ne = (Exception) e.getCause();
      if(ne.getClass().getName().equals("SqlException")){
      System.out.println("Database error: "+ e.getMessage());
      }
   }
   ...
   }
}
```