Preparation for Sun Certified Business Component Developer Exam, Java EE 5 (CX-310-091)

By Henry Naftulin (SCBCD, SCWCD, SCJD, SCJP)

Note: These notes are not enough for one to pass the exam – these are only review notes. The information here is not guaranteed to be right, I tried to make it correct, but I cannot guarantee it. What helped me pass the exam, might not help other people. This document is still somewhat disorganized. Please e-mail me with suggestions or corrections: henryn73@gmail.com

Transactions

EJB has to support distributed transactions by supporting two-phase commit. One can have bean managed transactions and container managed transactions. Bean managed transactions uses javax.transaction,UserTransaction interface. The UserTransaction interface could be obtained by @Resource annotation like @Resource UserTransaction ut; JNDI lookup like JNDI lookup intialConetx.lookup("java:comp/userTransaction") or by using EJBContext like ejbContext.getUserTransaction(). In CMT it will cause IllegalStateException.
Only one transaction could be supported by EJB 3.0 at a time, no nested transactions. If an new transaction has to be started while there is another transaction in place, the transaction that is in place will be suspended and new one started.
UserTransaction { begin(), commit(), rollback(), setRollbackOnly(), getStatus(), setTransactionTimeout(int seconds)}
A stateless session bean and MDB must commit transaction before method call returns or method time out call back returns. A statefull session bean does not have to commit transaction before method call returns.
An enterprise bean with BMT must not use getRollbackOnly() or setRollbackOnly() method of the EJBContext. It can use UserTransaction.getSatus() or can call UserTransaction.setRollbackOnly().
If a stateless bean starts but not completes transaction log is written for sys admin, transaction is rolled back, instance of the bean is discarded, EJBException is thrown.
If CMT is used, bean must not use the following methods of Connection: commit, setAutoCommit, rollback; and following JMS session: commit, rollback. It must not attempt to get UserTransaction interface. By default beans uses CMT with attribute Required (@TransactionManagement() for example @TransactionManagement(TransactionManagementType.BEAN).) Transaction is associated with method of bean interface, message listener method of MDB, timeout callback, stateless session bean web service end point.
MDB: Required and Not_Supported. For EJB **timeout** callbacks Required, Requires New, Not Supported. If you have **session synchronization** interface: required, required new, mandatory. **All are**: Not Supported, Required, Requires New, Supports, Mandatory, Never.
Session Synchronization { afterBegin, beforeCompetion, afterCompletion(completed)} only for container managed statefull session beans..

CMT Bean can use EJBContext.getRollbackOnly(). If there is an exception and you want to roll back transaction and set EJBContext.setRollbackOnly- rollback first, then throw a application exception the user. Alternatively mark your application exception as rollback=true. IllegalStateException if the method is Supports (and no exception is present), Not Supported, Never. CMT marking transactions for rollback context.setRollBackOnly – will throw exception if no transaction is in progress. Context.getRollbackOnly() tells you whether transaction is marked for rollback – only call that if there is existing transaction otherwise you will get an exception.
Statefull session bean cannot be passivated while in transaction; passivation is used to reduce memory footprint – serialization.

Persistent Entity Operations

Entity fields can only be protected, private or default visibility, not public (since they should not be accessed directly by clients).
Entity Manager API is used to create and remove persistent entity instances, to find entities by primary key and to query over entities. Persistent Unit – set of classes which are mapped to the same DB. Entity Manager is divided into following functional areas: Transaction Association, Entity Transaction (getTransaction() (JTA)) Entity lifecycle management.
States New, Managed, Detached, Removed.

	New	**Managed**	**Detached**	**Removed**
Persist	Managed	Ignored	IllegalArg	Managed
Remove	Ignored	Removed	IllegalArg	Ignored
Refresh	Ignored	Refreshed	IllegalArg	Ignored
Merge	Managed	Ignored	Managed	IllegalArg

Lock – for reading or writing.
Entity Identity Mangement: find(class, id), getReference(class, id) – does not necessarily go to db, might return hollow element and then when goes to db may return EntityNotFoundException; contains(item) – true if it is within the persistent context. If item is removed – false.
Cache management: flush() – writes changed items to db; getFlushMode, setFlushMode – Commit – only flush when committing, therefore queries might not take latest results that are not been committed; Auto – most results are committed; clear() – end persistent context and all entities will be detached.
Query Factory: createQuery(), createNamedQuery() (throws Illegal Arg), createNamedQuery(sql,class or ResultMap). Closing – isOpen(), close() – all entities become detached, query instances become invalid – in the managed environment, don't close!
Synchronization with DB does not refresh the managed entities unless refresh called. Entity Manager and Query setFlushMode() control synchronization to DB, no transaction – no flushing. Detached Entities: transaction commit, rollback; clearing persistent context or closing entity manager, serialization => only hydrated state can be accessed safely (including eager joining via Join Fetch). If field of detached object marked lazy – it should be ignored with respect to operation by merging operation by EJB subsystem – version must be checked.

Primary Key must be public and if property access is used it must have public or protected properties.
Entity, Embedable and Mapped Super-classes need to be denoted as managed persistent classes.

Entity listeners: can have zero or more listener. Entity listeners are stateless. The lifecycle methods can throw unchecked/ runtime exceptions. They are invoked in security and transaction context of calling component. The call takes no arguments and one of the annotations (@PrePresist, @PostPersist, @PreRemove, @PostRemove, @PreUpdate, @PostUpdate, @PostLoad) If you have listener class implement the method it has signature void method(object) where object is entity that it is listening to. Use mostly for cross cutting concerns - @EntityListener on Entity. Any access but not static or final. If there are multiple listeners order is: default listeners, super-class before sub-class for listener and if on classes – in the last starting with most generic one.
Optimistic locking – OptimisticLockException transaction rolls back, the exception might not be thrown before flush or commit.
Associations @OneToOne (in which for bi-directional association attribute mapped by is optional), @OneToMany (usually with mappedby), @ManyToOne (usually the owner of relationships). @XtoY – X refers to the class, Y refers to the variable. For example Item { @OneToMany (mapped=”item”) protected Set<Bid> bids; } Bid { @ManyToOne protected Item item; } The @XtoY annotation could be used for both fields and on getters, no default.

Element	@OneToOne	@OneToMany	@ManyToOne	@ManyToMany
targetEntity (what is the entity class, redundant)	Yes	Yes	Yes	Yes
Cascade	Yes	Yes	Yes	Yes
Fetch	EAGER	Lazy	Eager	Lazy
Optional (related object must always be present)	Yes (true)	No	Yes (true)	no
MappedBy	Yes	Yes	No	Yes

@Table(name=’’, uniqueConstraints=@UniqueConstraint(columnNames={‘’,’’}))
@Column(name,unique, nullable, insertable, updatable)
@Enumerated(EnumType.Ordinal /default/|String) – to save enumerated values
@SecondayTable(name,PKJoinColumn=@PrimaryKeyJoinColimn(name=’a_id’))
@Lob – for binary objects, goes with @Basic(fetch=FetchType.Lazy)
@Temporal(TemportyaType.Date|Time|Timesamp /all caps/) e.g. @Temporal(DATE)
@Embadable – also might be used with @AttributeOverrides({@AtributeOverride()})
@GeneratedValue(strategy=GeneratedType.Auto|Identity) goes with Id for pk generation
@OneToMany – has to have mappedBy=’’

Persistent Unit and Context

Two types: if controlled by JTA (JTA entity manager either application or container managed) or Entity Manager resource local entity manager- application managed. **Entity transaction interface** for resource local entity manager { begin(), commit(), rollback(), setRollbackOnly, getRollbackOnly, isActive() }. Unless you are using extended persistence context commit or rollback transaction ends persistence context. @PersistenceContext EntityManager em; - when transaction completes all entities become detached. Extended Entity manager only for statefull session beans, which is closed by @Remove marked method. Entity Manager never checks two instances representing same managed object. After entity manager is closed only getTransaction and isOpen don't throw IllegalStateException. For bean manager entity manager you need to em.joinTransaction() and only needed when Entity Manager is created from the factory. Otherwise it is not necessary. If no JTA is involved – illegal to call joinTransaction.

Persistent Unit is defined by persistence .xml in META-INF directory. Persistence Unit must have name. Persistence.xml can have more than one persistence unit. Persistence unit has managed classes, orm mapping of these classes (if no provider is specified – portable across providers. JTA is default and container provides data source. EJB-Jar, war, application client jar and ear can define persistence unit. EAR unit is visible at all levels but could be overwritten. # is used to reference specific persisitance unit lib/jar#myPersistanceUnit. All instances of Persistence Exception (except NoResults and NonUniqueResult) mark transaction for rollback. Rollback eception is thrown when em.commit() fails.

JPQL

Select From Where Group by Having Order. @Entity by name. Join – Select o from Order o Join o.items i. Count returns long, min/max same as they apply to, avg – double, sum Long or BigInteger or BigDecimal. [Inner] Join, Left [Outer] Join; Join Fetch – syntax to explicitly fetch via select. One can have collection in the Fetch clause of the select. Select o from Order o Join o.lineItems l where l.x = "y" ⇔ Select o from Order o, in (o.lineItems) l where l.x = "y"

Like = % any # of chars, _ 1 char. Collection operations IS[NOT] EMPTY, [NOT] MEMBER [OF]; select emp from Employee emp where emp.salary > ALL {or ANY, SOME} (select e.salary from …) Functions: concat, substring, trim [leading | following | both] [character from] string – like trim both j from x.title. lower, upper, locate – position of the string 0 if not found and 1 fist element, length, abs, mod, size, current_date, current_time_stamp

Sub-queries can be used in Where and Having clauses

Named queries @NamedQuery(name="fun", query="select …") em.createNamedQuery("fun"). @NamedQueries({@NamedQuery(), @NamedQuery()})

Dynamic em.createQuery("select.. a =:q").setParameter("q", 1).setMaxResults(40). SQL queries is only one class is populated the you can have syntax like em.createNativeQuery("select ..", com.my.ZClass.class); otherwise a SQL ResultMapping must be used em.createNativeQuery("select", "mappingName"); @SqlResultMapping(name="mappingName", enteties=@EntityRestult(entityClass=com.my.ZClass.class,

fields(@FieldResult(name=”id”, column=”order_id”),@FieldResult(name=”a”, colmn=”order_a”))/
Entity Manager: createQuery(jqlstring), createNamedQuery(name), createNativeQuery(name), createNativeQuery(sql, class), createNativeQuery(sql, mappedName).
Query: getResultList(), getSingleResult(), executeUpdate(), setMaxResults(), setFirstResult() – where to start, setParameter(int | stringName, object), setParameter(int | stringName, date | calendar, TemporalType), setFlushMode()

Exceptions

Clients should recover from application exceptions. Application exception is either a checked exception or a Runtime exception annotated as Application exception. Application exception cannot be a subclass of RemoteException. Ejb.CreateException, EBJ.RemoveException and ejb.FinderException are application exceptions. Application exceptions do not roll back transactions unless @ApplicationException(rollback=true). By default all checked exceptions are applications exceptions, runtime exceptions have to be annotated or be marked as such in the deployment descriptor <application-exceptions><exception-class>..</></> One can overwrite the transaction rollback in the descriptor. You can query rollback (for CMB) using EJBContext.getRollbackOnly() and change it by EJBContext.setRollbackOnly(). System exception is RemoteException, and all runtime exceptions that are not application exceptions. If they are thrown container must log it, throw EJBException (if web service then RemoteException) which will rollback the transaction, and throw away the instance of the bean. All managed resources are reclaimed. @PreDestroy might not be called. When clent receives EJBException or RemoteException client should either discontinue the transaction (set it for rollback) or continue one. If client continues transaction it should check whether it is not marked for rollback (CMT EJBContext.getRollbaclOnly, BMT – UserTransaction.getSatus) The reason for it is because communication subsystem on client side may not able to send the request – and so transaction might still be in-tact.
CreateException, FinderExcpetion (and subclasses) and RemoveException are ejb checked application exceptions.

Client	**Session Bean**	**Exception Type**	**Exception thrown to client**
Client in transaction/Client in no transaction	Bean in clients transaction Bean in it’s own transaction Bean in no transaction Bean runs locally or remotely	Application exception	Same application exception
Client in transaction	Bean in clients transaction, locally	System exception	EJBTransaction Rollback exception
Client in transaction	Bean in clients transaction, remotely	System exception	Transaction Rollback exception

Client in transaction	Bean in it's own transaction or Bean in no transaction Locally or Remotely	System exception	EjbException
Client in no transaction	Bean not in transaction, Bean in it's own transaction; Locally or Remote	System exception	Same application exception

Security

If bean provider or Application assembler do not assign security roles – deployer will have to. If no role assigned – all can call the bean or method. Bean provider and Application Assembler define only logical roles – they don't know the execution environment. Bean provider App Assembler <assemble-descriptor><security-role><description></><role-name>manager</></> -- security-role-ref. Bean Provider is responsible for putting @DeclaredRoles("") on the bean or <security-role-ref><role-name>… on the declarations. One cant test it by calling EJBContext.isCallerInRole(). Application assembler can put in <role-link> in the deployment descriptor. Responsibilities: optionally bean provider or application assembler could define security – per method security annotations or deployment descriptor. (Otherwise deployer neds to know all the about the application). It is logical security view of the app. Deployer has to ensure that application is secure after deployment. Deployer assigns principals or principal groups to the application security environment. EJB Context provides 2 call isCalllerInRole("rolename") getCallerPrincipal() / designed to get caller identity/ RolesAllowed, PermitAll, DenyAll, RunAs – RunAs applies to sessions bean as a whole and principal for that is assigned by deployer.

Messaging

Point-to-Point or Publish-Subscribe. @Resource(name="jms/QueueConFactory") private ConnectionFactory cf; @Resource(name="jms/shipQueue") private Destination destination; to create a sender we need to get session.createProducer(destination) and send message on producer. Connections are thread safe, but not Sessions. On transactional sessions the messages are send when session is closed or session.commit is called.

MDB provide multithreading, simplified message code and automatic message consumption upon start of the container. MDB must directly or indirectly (@MessageDriven(messageListenerInterface="class"), or deployment descriptor) implement message listener interface. It must be concrete, not a subclass of another MDB, public. Must not throw rmi.RemoteException. @MessageDriven – could be used just like that or like @MessageDriven(name="myMDB", activationConfig=(@ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Queue"),

@ActivationConfigProperty(propertyName=”destinationName”, propertyValue=”jms/shipQueue”)).
@Resource has to use tye element if it is used at the class level. Also DI is either done by knowing the the name like context for EJB @Resouce EJBContext context; by reference in deployment descriptor <resource-ref-><res-ref-name>abc/efd</><res-type>com..T</></> and @Resource(name=”abc/efd”) T myVar; or by environment entry of primitive wrapper types (but not Calendar or Date) <env-entry><env-entry-name>abc</><env-entry-type>java.lang.Boolean</><env-entry-value>true</></>

Interceptor

@Interceptor(A.class[,B.class]) on a method or on the class. Interceptor class has @AroundAdvice method with InvocationContext parameter. You can specify interceptors at class level, method level or even default interceptor at deployment descriptor level. You can exclude default interceptor and class interceptor. InvocationContext{ getTarget(), getMethod(), getParameters(), setParameters(), Map getContextData() /communication between interceptors/, proceed()}. AroundInvoke pattern Object method(invocationContext) throws Exception; Event lifecycle callbacks can be interceptors – these call backs have InvocationContext as parameter, though lifecycle interceptors can only throw runtime exceptions.

Beans can have multiple @PostConstruct and @PreDestoy methods – void method() make sure that the method does not throw any checked Exception. Statefull bean can have multiple @Remove methods – and should have at least one @Remove.

Timer Services

Either EJBContext.getTimerServices() or @Resource TimerService timerService. Could only be used in Stateless Session Beans and MDBs. Has to have @Timeout method or it can be specified in deployment descriptor as timeout. The method takes timer as argument: @Timeout [anything] void method(Timer t). To create time user the following: timerService.createTimer(Date|durationMilis, serializableInfo) or createTimer(Date|durationsMilis, intervalMilis, serializableInfo). If bean implements java.ejb.TimedObject – ejbTimeout method is timeout. Timer { cancel(), long getTimeRemaining(), Date getNextTimeout(), getHandle(), getInfo()} – all transactional. TimeHandle – serializable object to get info abut timer. @Timeout method could be transactional (Requred, Requires_new). Timers survive EJB crashes.
You cannot use timers in Statefull Session Beans.

Migration 2.1 to 3.0

Ejb-jar needs to be version 3.0 (or not specified version at all) – otherwise the container will ignore the EJB bean annotations, for example if version is 2.1. Invoking EJB 2 from EJB3: you inject the home interface into the EJB 3 bean, call home.create() and then use business method of the bean; note that if you are using remote interface a narrowing on the home interface is needed. If callig EJB2 CMP entity from EJB 3 do the same – inject home interface and then call home.create(dto). Using EJB3 from EJB2: remember that

there is no injecting in EJB2 so you need to lookup references. 1. you need to have ejb-ref and ejb-local-ref in ejb-jar.xml to enable references to EJB3 bean. 2. Then you need to do JNDI lookup like this BizInterface biz = (BizInterface) context.lookup("jave:comp/env/ejb/BizInterface).

Methods that can be called:

	getEjbHome getEjbLocalHome JDNI lookup	getBusinessObject	getCallerPrincipal, getCallerInRole	getRollBackOnly setRollBackOnly	Timer	EM, EMF,RM, EJB
Constructor	-	-	-	-	-	-
Dependency injection Set session context	Y (CMP/BMP)	-	-	-	-	-
PostConstruct PreDestroy PrePassivate PostActivate	Y(CMP/BMP)	Y(CMP/BMP)	Y(CMP/BMP)	-	-	Y(CMP/BMP)
Bean method	Y	Y	Y	Y	Y	Y
afterBegin beforeCompletion	Y(only CMP)	Y(only CMP)	Y(only CMP)	Y(only CMP)	Y(only CMP)	Y(only CMP)
afterCompletion	Y(only CMP)	Y(only CMP)	Y(only CMP)	-	-	-

Environment entry values have to be java.lang. object classes: String, Float, Integer, etc.