EJB 3.X

Rajeev Gupta

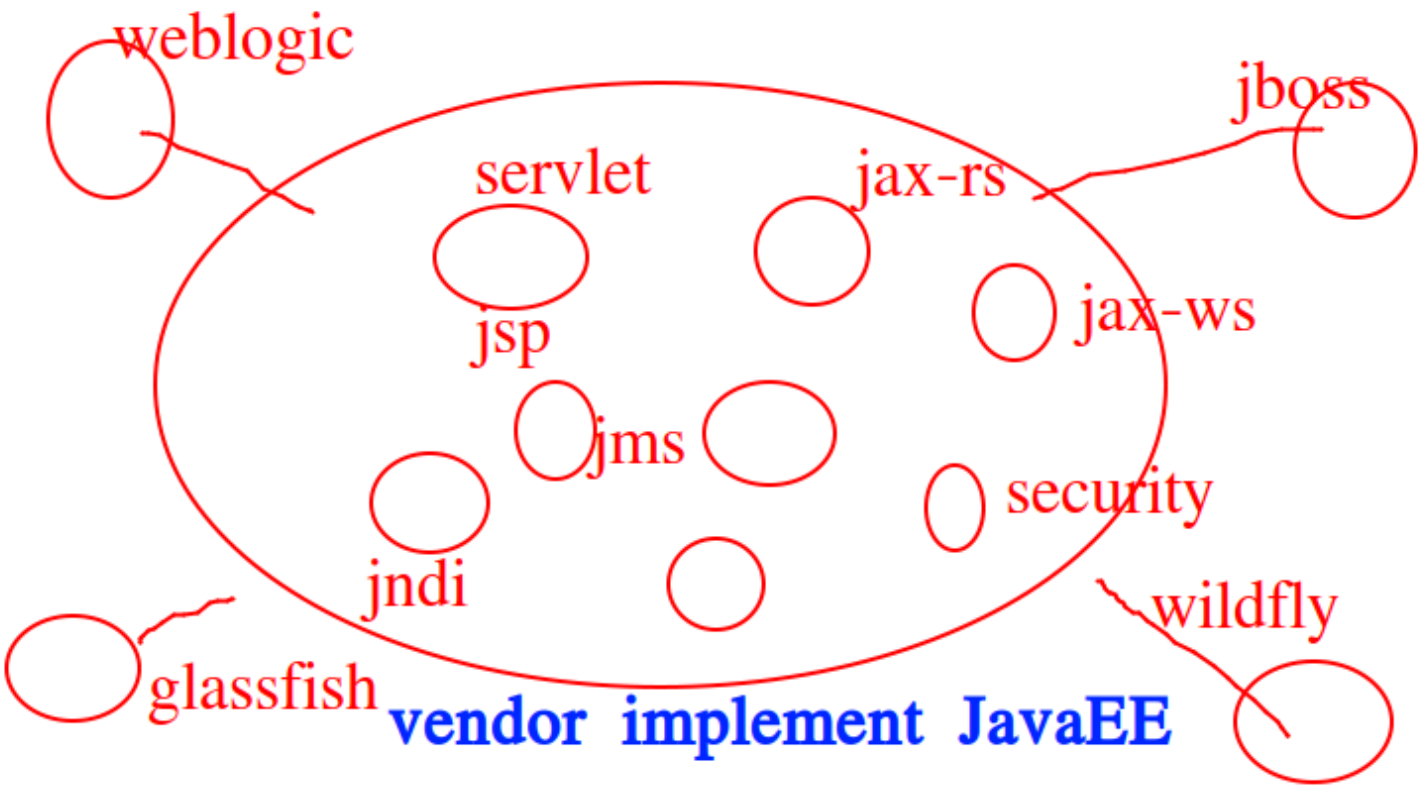
Agenda

- Introduction to EJB
- Stateless Session bean, Bean life cycle
- Stateful session bean, Life cycle
- Singleton Bean, life cycle
- Accessing EJB to client
- References and Injections
- EJB JPA integration, CRUD application
- Transactions
- Exception handling
- JAX-WS
- JAX-RS

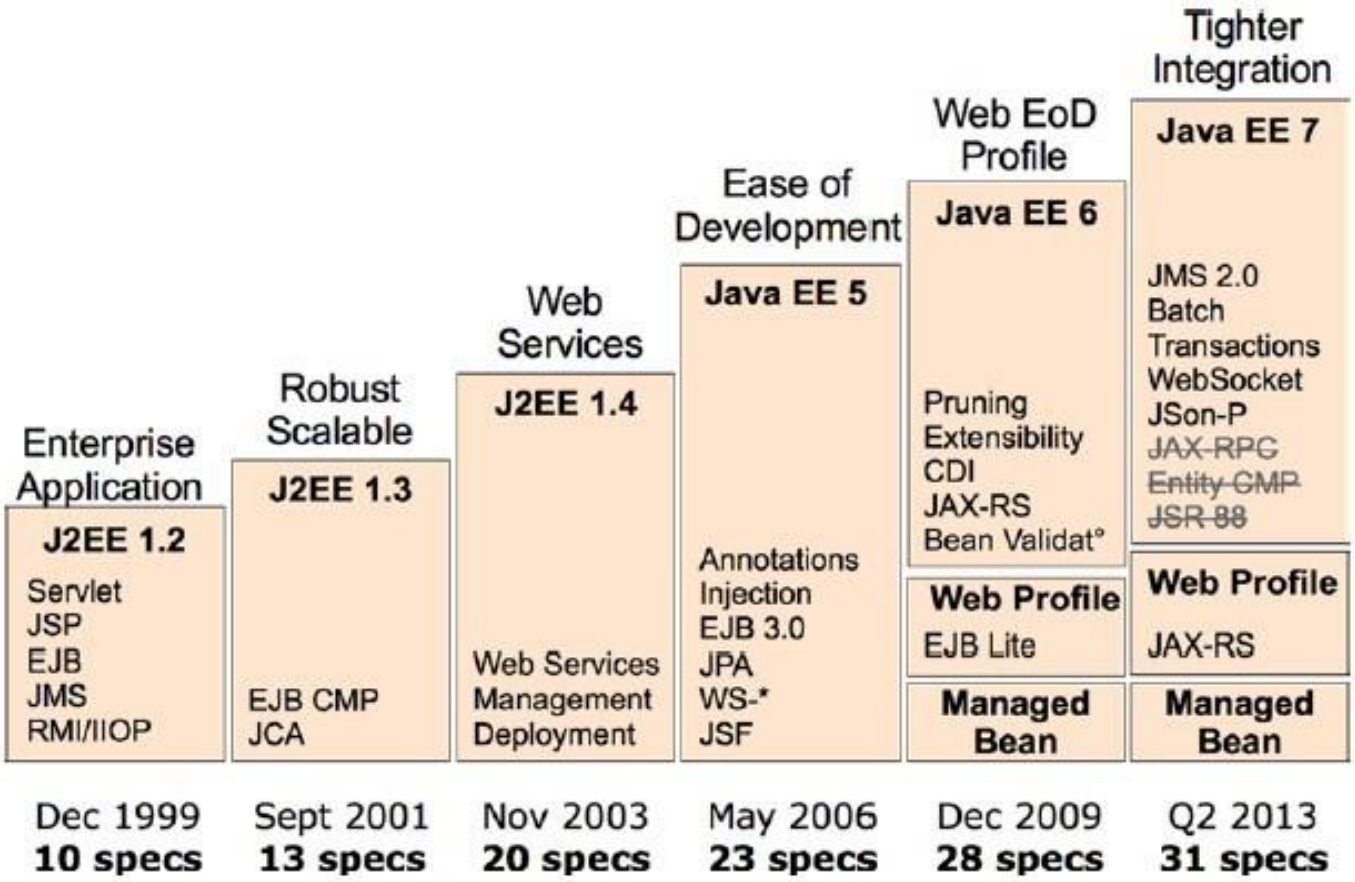
Introduction to EJB

What is Java EE?

Java EE is group of Specification JSR on top of J2SE for
creating dynamic distribtued application



History of Java EE



Java EE layered architecture

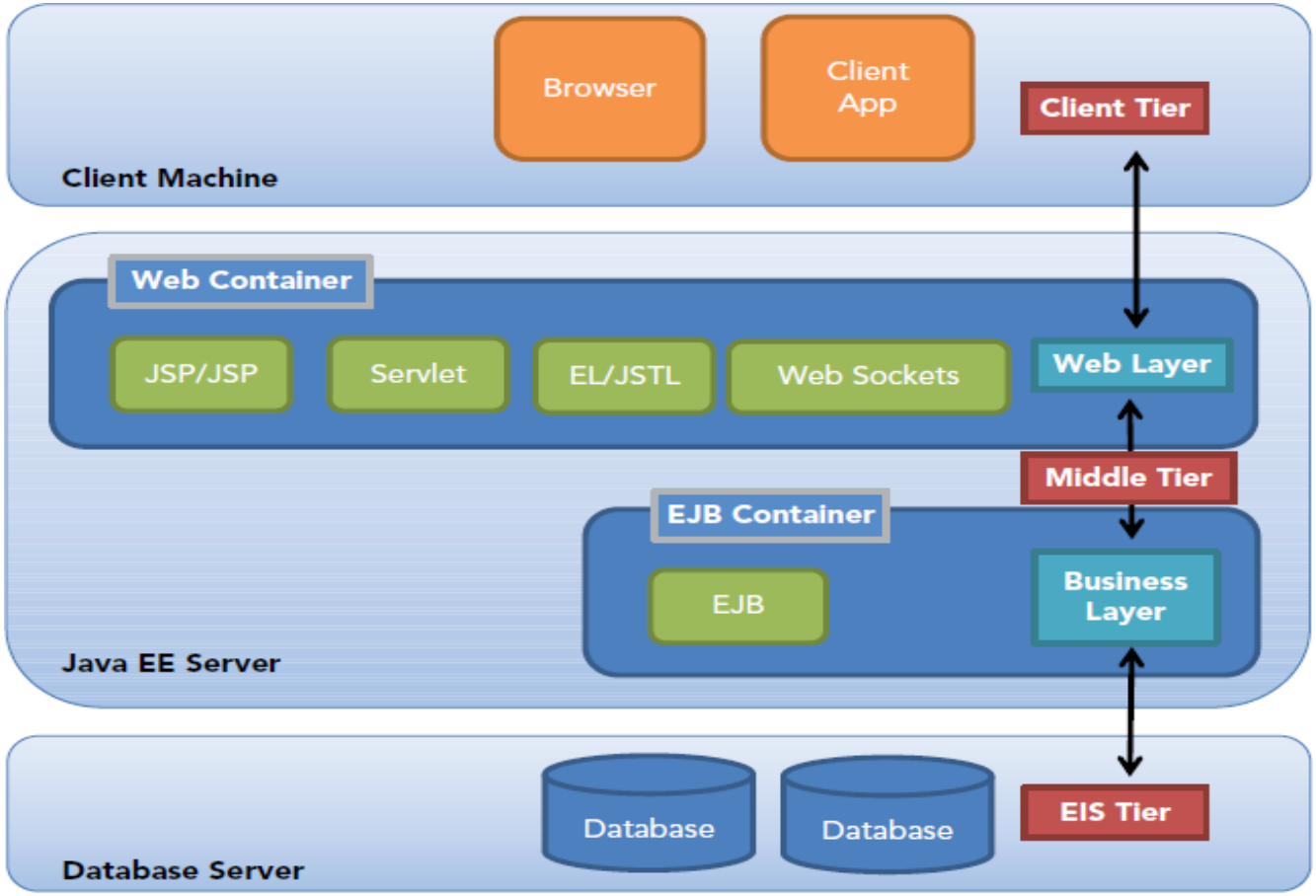


FIGURE 2-1: Multitier architecture showing the interaction between tiers

What is EJB ?

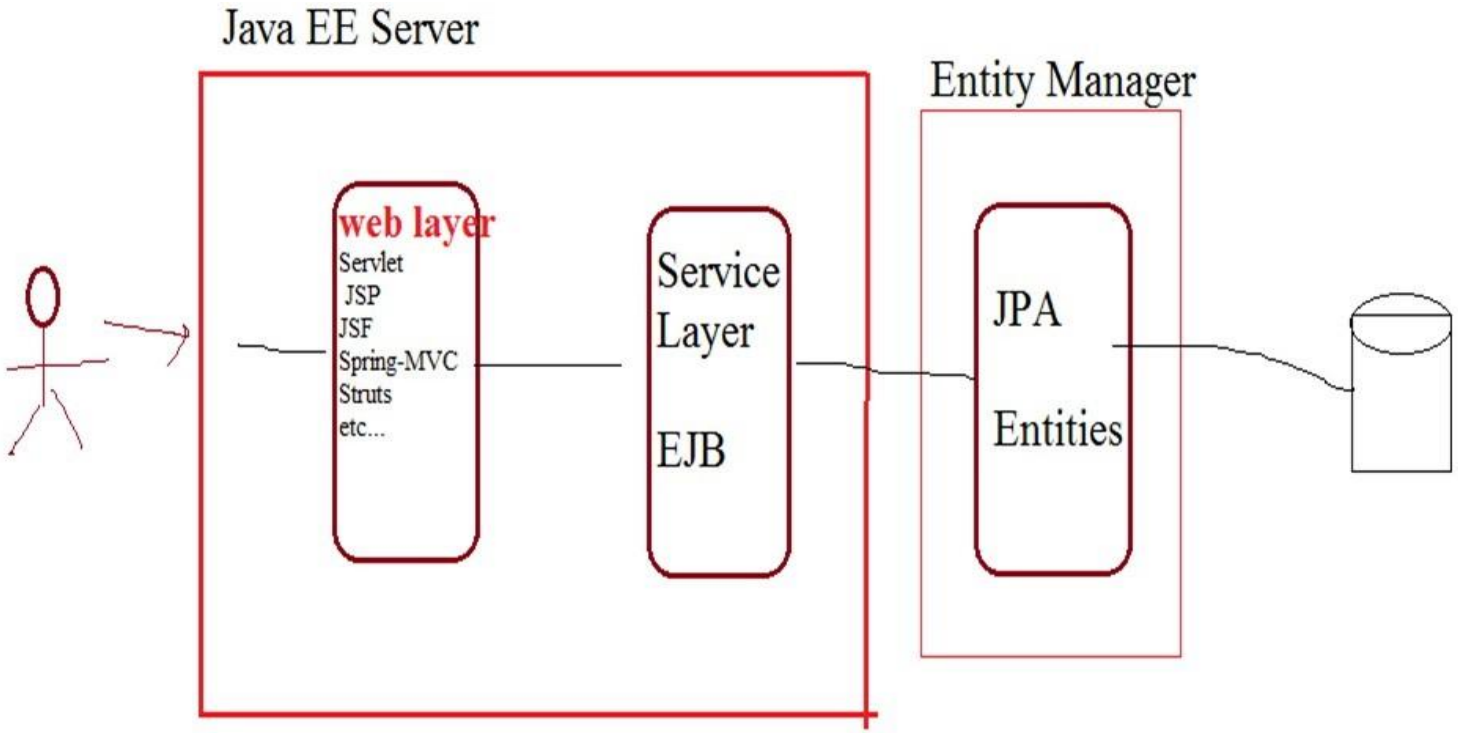
- is platform for building ***portable, reusable***, and ***scalable business applications*** using the Java
- is Java components (classes) that executes in a specialized runtime environment called the ***EJB container***

- EJB is component based framework for Building and deploying distributed applications

Benefits of Enterprise Beans

- EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems
- Enterprise beans are portable components, the application assembler can build new applications from existing beans

Three tier architecture With EJB



Web layer<=======>Service layer<=======>Persistance Layer <======>DBMS

Where EJB fits: Service layer

- Service layer benefits?

  # Transactions

  #Security

  #Persistence connection pooling

  #Remotablility

  # Interceptors

  #Timer

  #JMS

  ==>Without EJB overhead on the programmer to write all above services

  ==>EJB container is deployed on application server such as Jboss, Weblogic, glassfish etc.

Types of EJB

```
                EJB
                |
        ------------------------------------------------
        |                       |                       |
        Seession                Entity  X               Message Driven
        Bean                    bean                    Bean
        |                       |
   ------------------      ------------------
   |                |      |                |
SFSB             SLSB    CMP              BMP
```

- SFSB: Stateful session bean
- SLSB: Stateless session bean

Entity beans

- classified based on who manages transaction mgt
  - CMP: Container managed persistence
  - BMP: Bean (programmer) managed persistence

- Concept of Entity Bean is replaced by Entity in JEE 5
  - What is Entity?
    - POJO annotated with annotation JPA
    - Persisted, transactional
    - Entity life cycle is maintained by EntityManager not by app server container

Stateless Session bean, Bean life cycle

Session bean

- Performs specific business-logic operations
- Managed by EJB Container
- Available for a client session
- Is removed when the EJB container crashes
- Can be transaction-aware
- Can be invoked either locally or remotely using Java RMI

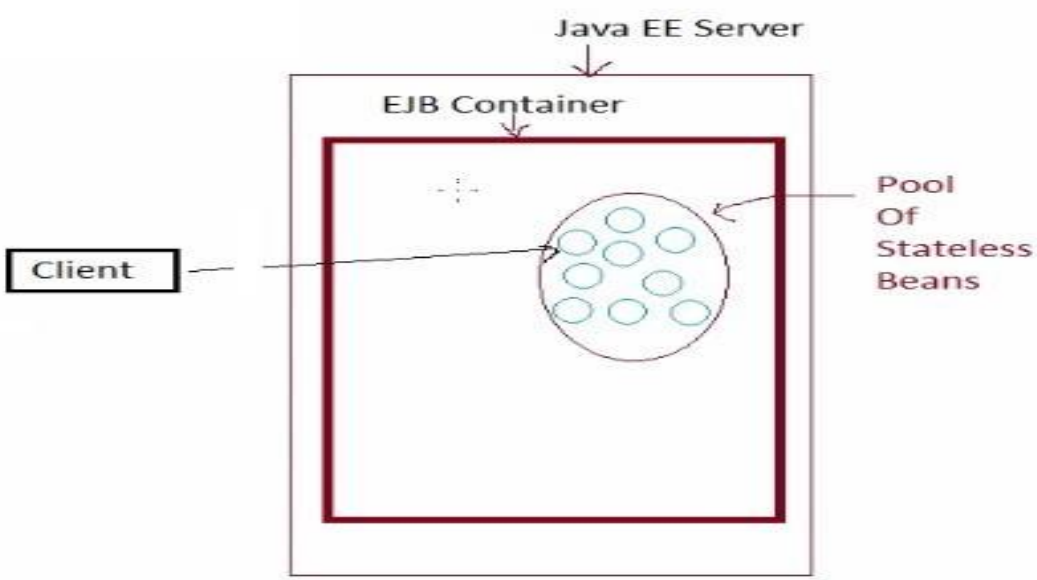
Types of Session Beans

- Stateless Session Beans
- Stateful Session Beans
- Singleton Session Beans

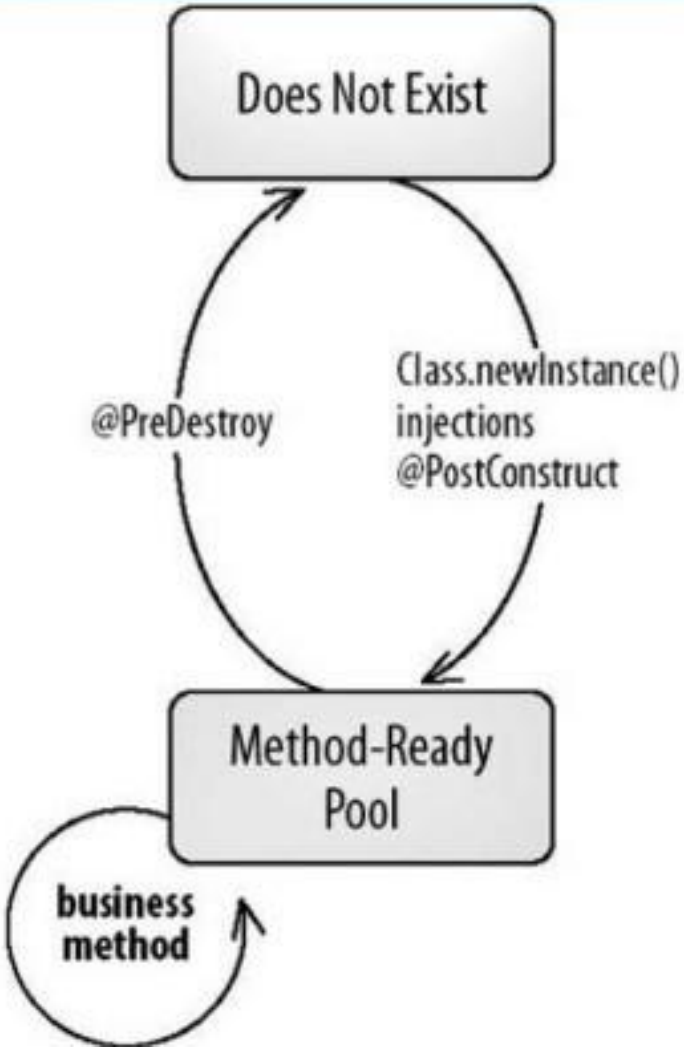
Stateless SLSB

- A session bean that does not maintain conversational state.
- Used for reusable business service that are not connected to any specific client.

SLSB: container manages the pool of beans



Stateless Bean Lifecycle and Callbacks




```
@Stateless
public class CalculationBean {

    @PostConstruct
    void construct() {
    }

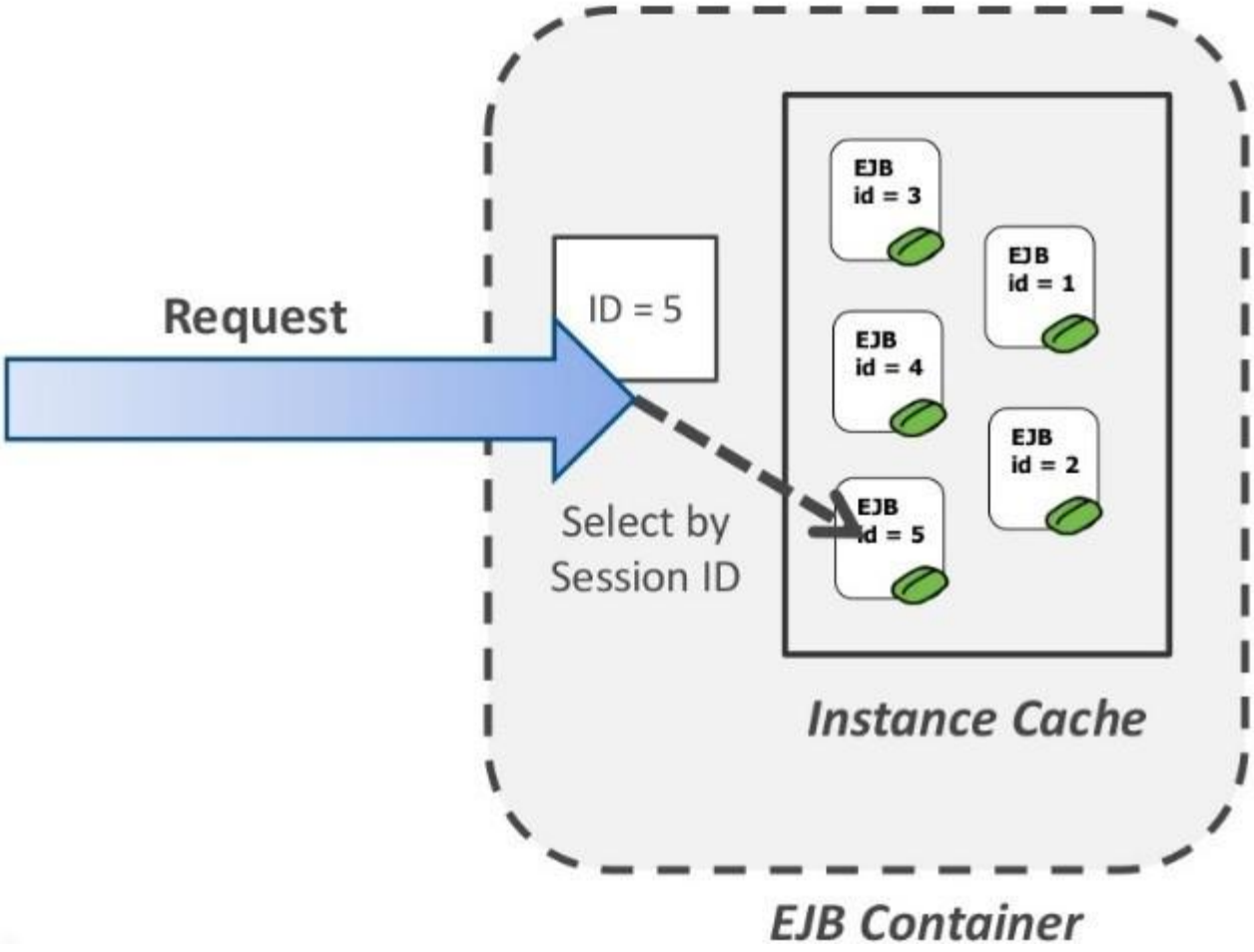
    @PreDestroy
    void destroy() {
    }

    public int sum(int a, int b) {
        return a + b;
    }
}
```

SLSB Life cycle

1. The life cycle of a stateless or singleton session bean starts when a client requests a reference to the bean (using either dependency injection or JNDI lookup). In the case of a singleton, it can also start when the container is bootstrapped (using the @Startup annotation). The container creates a new session bean instance.
2. If the newly created instance uses dependency injection through annotations (`@Inject`, `@Resource`, `@EJB`, `@PersistenceContext`, etc.) or deployment descriptors, the container injects all the needed resources.
3. If the instance has a method annotated with `@PostContruct`, the container invokes it.
4. The bean instance processes the call invoked by the client and stays in ready mode to process future calls. Stateless beans stay in ready mode until the container frees some space in the pool. Singletons stay in ready mode until the container is shut down.
5. The container does not need the instance any more. It invokes the method annotated with `@PreDestroy`, if any, and ends the life of the bean instance.

Stateless session bean: Pool



Stateless session bean use cases

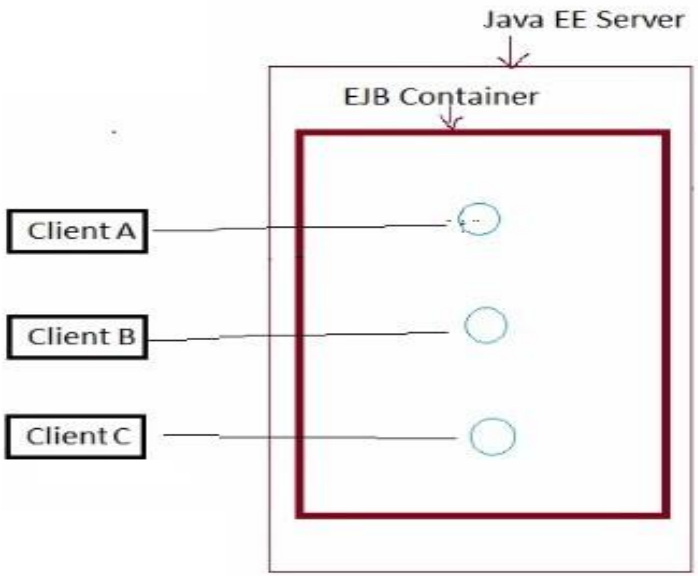
- Use cases of Stateless session bean
  - The bean's state has no data for a specific client.
  - In a single method invocation, the bean performs a generic task for all clients.
  - Need to implement bean as web service (SFSB can not deployed as WS)
- Features of Stateless session bean
  - Container manages pools of SLSB
  - One of bean form pool randomly picked for a client request, on second request from the same client another instance can be picked up hence We should not use instance variable to save state of bean in SLSB
  - fit for Singular business cases

Stateful Session bean, Bean life cycle

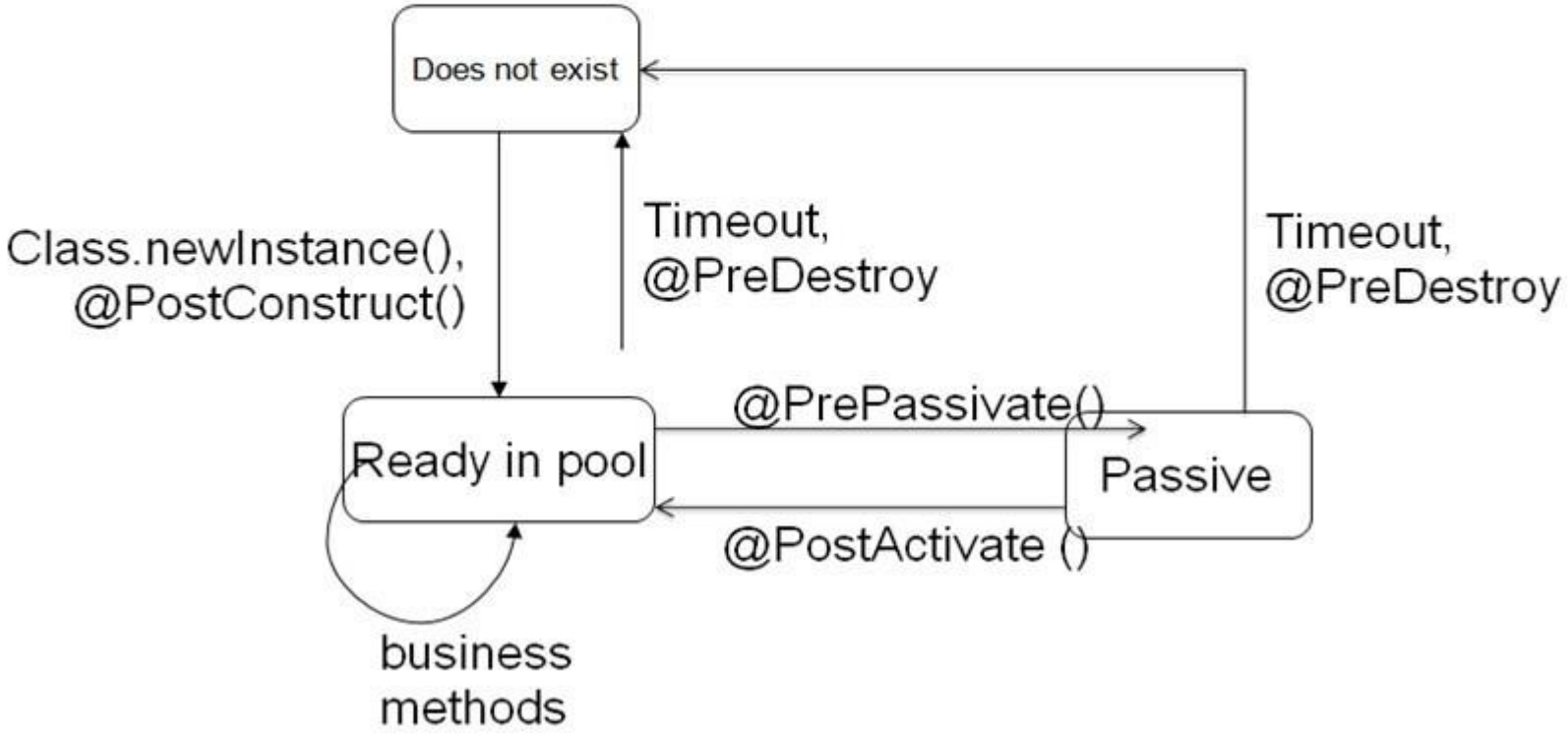
Stateful Session Bean SFSB

- A session bean that does maintain conversational state.
- Used for conversational sessions with a single client (for the duration of its lifetime) that maintain state, such as instance variable values or transactional state.

SFSB: Container do not maintain pool of Beans



Stateful EJB Session Bean Lifecycle



Life cycle methods in SFSB

1. The life cycle of a stateful bean starts when a client requests a reference to the bean (either using dependency injection or JNDI lookup). The container creates a new session bean instance and stores it in memory.
2. If the newly created instance uses dependency injection through annotations (`@Inject`, `@Resource`, `@EJB`, `@PersistenceContext`, etc.) or deployment descriptors, the container injects all the needed resources.
3. If the instance has a method annotated with `@PostContruct`, the container invokes it.
4. The bean executes the requested call and stays in memory, waiting for subsequent client requests.
5. If the client remains idle for a period of time, the container invokes the method annotated with `@PrePassivate`, if any, and passivates the bean instance into a permanent storage.
6. If the client invokes a passivated bean, the container activates it back to memory and invokes the method annotated with `@PostActivate`, if any.
7. If the client does not invoke a passivated bean instance for the session timeout period, the container destroys it.
8. Alternatively to step 7, if the client calls a method annotated by `@Remove`, the container then invokes the method annotated with `@PreDestroy`, if any, and ends the life of the bean instance.

Stateful Session Bean lifecycle Example

```
@Stateful
public class CalculationBean implements CalculationRemote {
    private int result = 0;
    @PostConstruct
    void construct(){}
    @PreDestroy
    void destroy(){}
    @PostActivate
    void activate(){}
    @PrePassivate
    void passivate(){}

    @Remove
    public void remove(){}

    public void add(int a) {
        result += a;
    }

    public int getResult() {
        return result;
    }
}
```

features of SFSB

- Client oriented
- No pool of instance Bean created before client invoke One instance per client
- If client call one method on Stateful bean, second time he can call another method on the same bean and can use state of instance variable
- **State of instance variable is preserved.**

SLSB vs. SFSB

- Stateless session beans are usually pooled, and maintaining pool is container responsibility.
- Stateful session bean although maintain by container but it is client responsibilities to call a specific method to remove /destroy that bean
- Instance variable in SFSB represent conversational state for a particular client
- we need to manually use @Remove to signals end of the session in SFSB

Rule for Life cycle methods

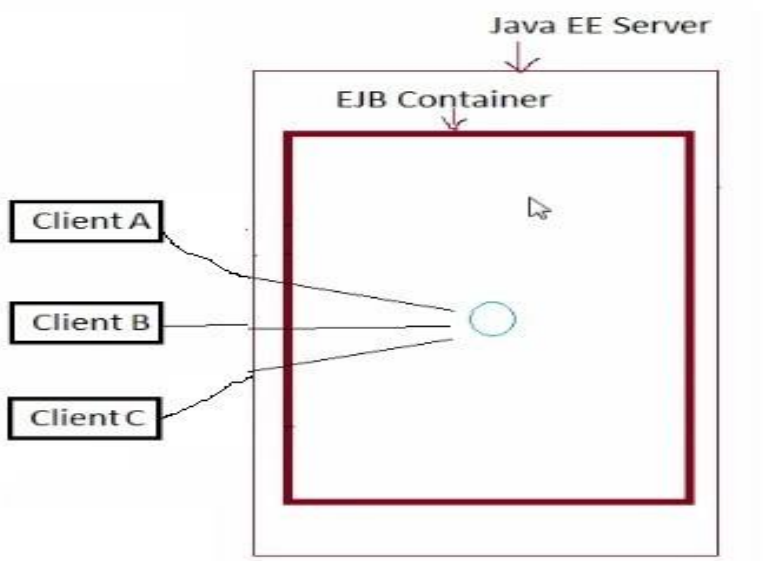
- @PostConstruct and @PreDestroy @PostActivate and @PrePassivate call back methods
- these methods:
  1. Must return null and take no arguments
  2. should not throw Checked Exception.can have any access modifiers public, private etc
  3. May be final
  4. Container may passivate a SFSB instance only if it is not in transaction

Singleton Session bean, Bean life cycle

Singleton Session Bean

- One bean instance per app server
- Two/more client can access same bean at a time
- concurrency issue
- Can be used as main for testing application , start-up, creating schema etc.

Singleton Session Bean: one instance per app server



Singleton Session Bean

- Instantiated once per application and exists for the lifecycle of the application
- Only one bean per application (jvm)
- Does not maintain state across server crashes/shutdowns
- Supports concurrency requests
- Can implement a web service
- Perform tasks upon application startup and shutdown
- State shared across the application

Where Singleton session bean fits?

- Singleton bean fits when:
  - State needs to be shared across the application

- A single enterprise bean needs to be accessed by multiple threads concurrently

- application needs an enterprise bean to perform tasks upon application start-up and shutdown

Singleton Session Bean. Concurrency Management

Singleton session beans are designed for *concurrent access*

@javax.ejb.ConcurrencyManagement :

- **container-managed concurrency**
  javax.ejb.ConcurrencyManagementType.CONTAINER - default

- **bean-managed concurrency**
  javax.ejb.ConcurrencyManagementType.BEAN

Container-Managed concurrency. Locking types

Access locking types:
javax.ejb.Lock,
javax.ejb.LockType

- **@Lock(LockType.READ)** - for read-only operations
- **@Lock(LockType.WRITE)** - default; for exclusive access to the bean instance

javax.ejb.AccessTimeout - annotation is used to specify the number of **TimeUnits** before an access timeout occurs

Container-Managed Singleton Example

```
@Singleton
@ConcurrencyManagement(CONTAINER)
@AccessTimeout(value=20, unit = TimeUnit.SECONDS)
public class CalculationBean implements CalculationLocal {

    private Long result;

    @PostConstruct
    void construct() {
        result = 0L;
    }

    @Lock(LockType.WRITE)
    public void add(int a) {
        result += a;
    }

    @Lock(LockType.READ)
    @AccessTimeout(value=3600)
    public Long getResult() {
        return result;
    }
}
```

Initializing Singleton Session Beans

Eager Initialization:

@javax.ejb.Startup

@javax.ejb.DependsOn

```
@Startup
@Singleton
@DependsOn({"PrimaryBean",
"SecondaryBean"})
public class CalculationBean
        implements CalcLocal, CalcRemote {

    private Long result;

    @PostConstruct
    void construct() {
        // any bean initialization ...
        result = 0L;
    }
```

Summary: Session Bean Types

Stateless	Stateful	Singleton
Has no client association	Each client has its own bean instance	Instantiated once per application
Has no state between calls	Stores state between client calls	Each client obtains single state
Pooled in memory	May be passivated to disk, cached	-
Client couldn't manage lifecycle	Removable by client	Client couldn't manage lifecycle
Does not support concurrency	Does not support concurrency	Supports concurrency
Implements WS	Couldn't implement WS	Implements WS

When to use what?

Stateful session beans

```
        # The bean's state represents the interaction between the bean and a specific client.
        # To hold information about the client across method invocations
```

Stateless session bean

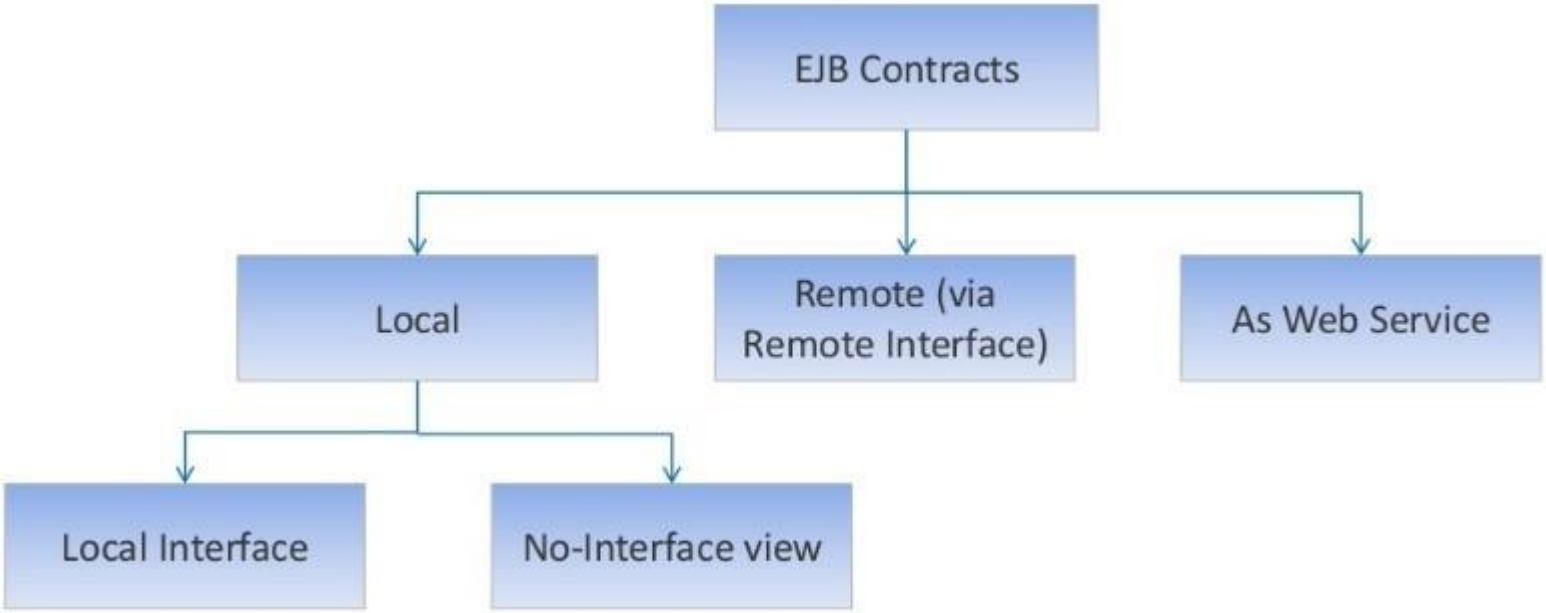
```
        # has no data for a specific client
        # performs a generic task for all clients
        # implements a web service
```

Singleton session

```
        # state needs to be shared across the application
        # To be accessed by multiple threads concurrently.
        # To perform tasks upon application startup and shutdown
        # implements a web service
```

Accessing EJB to client

Session bean contracts with clients



Local Interface Contract

- Run in the same JVM as the EJB
- Client: **Web component** or another **enterprise bean**
- Supports **pass-by-reference** semantics
- The **local business interface** defines the bean's business methods

Local Bean Declaration

To build an enterprise bean that allows local access:

```
@Local public interface CalculationLocal { … }

@Stateless
public class CalculationBean implements CalculationLocal { … }
```

or

```
public interface CalculationLocal { … }

@Stateless
@Local(CalculationLocal.class)
public class CalculationBean implements CalculationLocal { … }
```

Remote Contract

- Client can run on a different JVM
- It can be a **web component**, an **application client**, or **another EJB**
- Supports **pass-by-value** semantics
- Must implement a business interface
- Requires creating a stub and a skeleton, which will cost cycles
- Expensive taking time to transfer the object to the client and create copies (Serializable)

Remote Bean Declaration

To create an enterprise bean that allows remote access:

- ```
  @Remote
  public interface CalculationRemote {…}

  @Stateless
  public class CalculationBean
              implements CalculationRemote {…}
  ```

- Or decorate the bean class with @Remote:

  ```
  public interface CalculationRemote {…}

  @Remote(CalculationRemote.class)
  public class CalculationBean
                  implements CalculationRemote {…}
  ```
  ```

Local Contracts. No-Interface view

- **No-interface view** == local view
- Public methods available for client access
- The client and the target bean must be packaged in the same application (EAR)
-
```
@Stateless
public class CalculationBean { … }
```

Using Session Beans in Clients

Client obtains Session Bean instances through:

- **JNDI lockup** (using the Java Naming and Directory Interface syntax to find the enterprise bean instance)
- **dependency injection** (using annotations)

Portable JNDI Syntax

Three JNDI namespaces are used for portable JNDI lookups:

- **java:global** – remote beans using JNDI

  java:global[/application name]/module name/enterprise bean name[!interface name]

- **java:module -** local enterprise beans within the same module (JAR)
  java:module/enterprise bean name[!interface name]

- **java:app** - local enterprise beans packaged within the same application (EAR)
  java:app[/module name]/enterprise bean name[!interface name]

Lookup Example

```
Properties props = new Properties();

props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.enterprise.naming.SerialInitContextFactory");
 …

InitialContext initialContext = new InitialContext(props);



CalculationLocal bean = (CalculationLocal) initialContext
.lookup("java:app/myapp-
ejb/CalculationBean!net.myapp.service.ejb.CalculationLocal");
```

Accessing Beans Using JNDI

- **No-Interface view:**

```
CalculationBean  bean = (CalculationBean)
initialContext.lookup("java:module/CalculationBean");
```

- **Local:**

```
CalculationLocal  bean = (CalculationLocal)
initialContext.lookup("java:module/CalculationBean!
CalculationLocal");
```

- **Remote:**

```
CalculationRemote  bean = (CalculationRemote)
initialContext.lookup("java:global/myapp/myapp-
ejb/CalculationBean!CalculationRemote");
```

Summary: client contracts

Remote	Local	No Interface View (Local)
Accessed by Interface @Remote	Accessed by Interface @Local	Public methods available for client access
Could be accessed from another JVM	Accessed from the same JVM	Accessed from the same EAR
Supports **pass-by-value** semantics	Supports **pass-by-reference** semantics	Supports **pass-by-reference** semantics
JNDI Lookup or DI (from the same JVM only)	JNDI Lookup or DI	JNDI Lookup or DI

Accessing Beans Using Dependency Injection

- **No-Interface view:**

```
@EJB
private CalculationBean bean;
```

- **Local:**

```
@EJB
private CalculationLocal bean;
```

- **Remote:**

```
@EJB
private CalculationRemote bean;
```

References and Injections

Entity Manager Reference

- Entity Manager can be registered in the JNDI Context of an EJB
- EJB container has full control over the lifecycle of the underlying persistence context of the Entity Manager
- **@javax.persistence.PersistenceContext** can be used on bean class's setter methods or member fields or directly on the class

- The @PersistenceContext annotation can also be placed on a setter method or member field :

```
@Stateless
public class CalculationBean implements CalculationLocal
{
    @PersistenceContext(unitName="MyDB")
    private EntityManager em;
}
```

Resource Reference

- JNDI Context can be used to look up external resources
- the external resources are mapped into a name within the JNDI Context by using annotations or DD xml content
- External Resources:

  javax.sql.DataSource

  javax.jms : Queue, Topic, Connection Factories

  javax.mail.Session

  java.net.URL

  java.lang: String, Character, Byte, Short, Integer, Long, Boolean, Double, Float, Class, and all Enum types.

  Javax.transaction: UserTransaction, TransactionSynchronizationRegistry

  CORBA ORB references

  JPA PersistenceUnit and PersistenceContext

```
@Resource(name="jms/QueueConnectionFactory")
private javax.jms.QueueConnectionFactory
queueConnectionFactory;

@Resource(name="jms/someQueue")
private javax.jms.Queue queue;

@Resource(
        name="jdbc/PostgresDB",
        type=javax.sql.DataSource.class,
        shareable = true,
        authenticationType =
AuthenticationType.APPLICATION)
private javax.sql.DataSource dataSource;
```

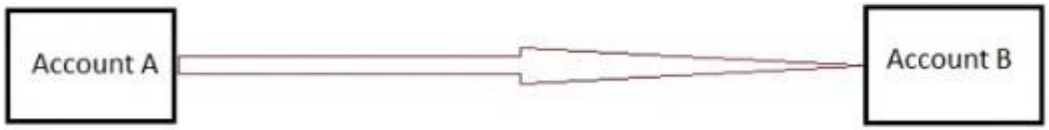
Transaction management

What is the need of transaction management?

Transaction

The set of operations treated as a single unit, and all of the operations must succeed or none of them can succeed

Aim: To transfer 100 $ from Account A to Account B



1) Subtract 100 $ from Account A

2)Add 100 $ to Account B

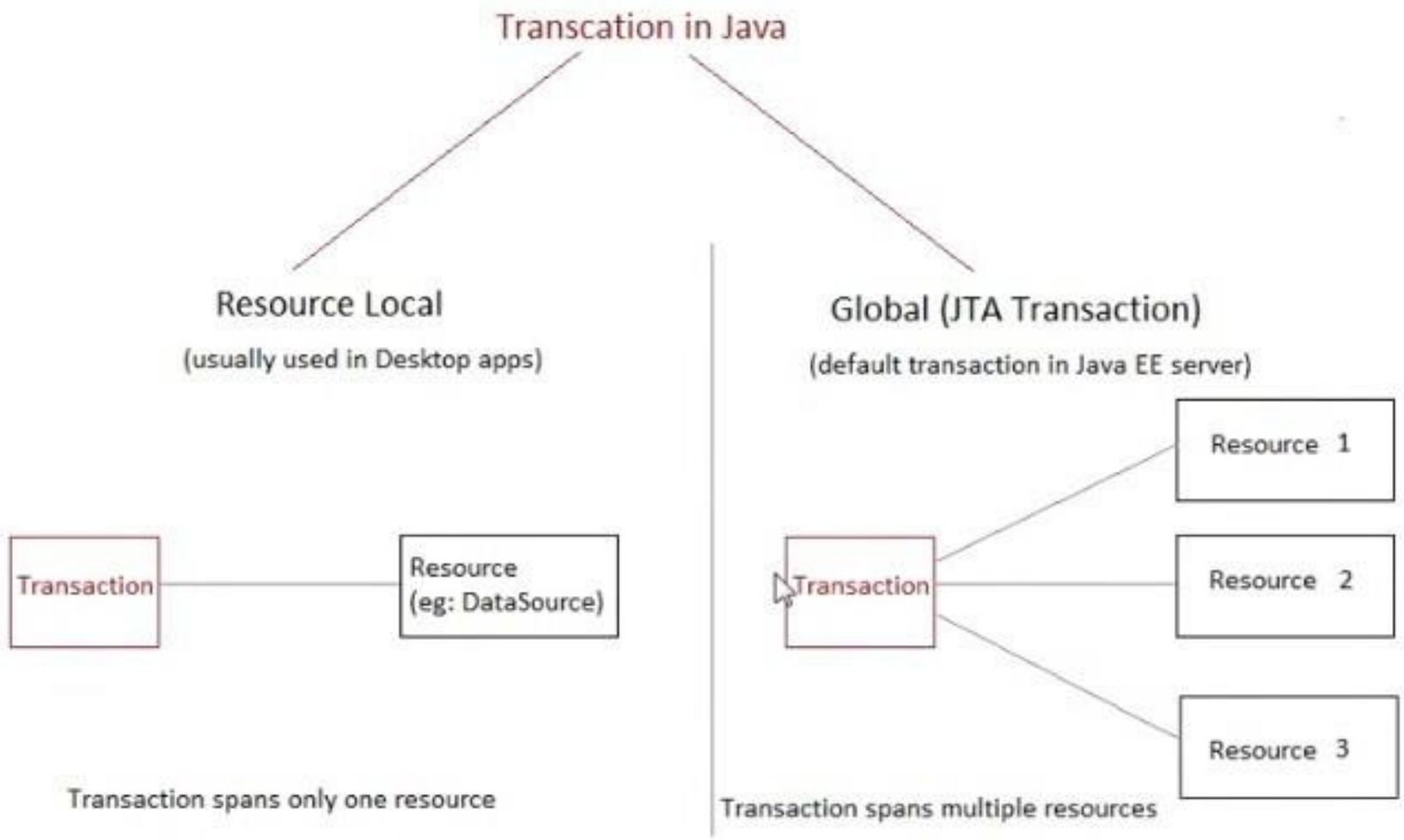
ACID Properties

- Atomicity: Either all the operations in a transaction are successful or none of them is. The success of every individual operation is tied to the success of the entire group.

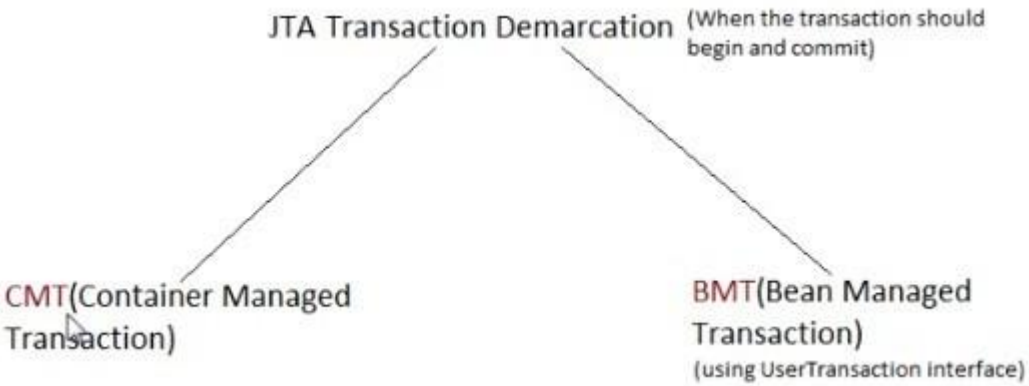
- Consistency: The resulting state at the end of the transaction adheres to a set of rules that define acceptability of the data. The data in the entire system is legal or valid with respect to the rest of the data in the system.

- Isolation: Changes made within a transaction are visible only to the transaction that is making the changes. Once a transaction commits the changes, they are atomically visible to other transactions.

- Durability: The changes made within a transaction endure beyond the completion of the transaction.


Transcation in Java
Resource Local
(usually used in Desktop apps)
Transaction
Resource
(eg: DataSource)
Transaction spans only one resource
Global (JTA Transaction)
(default transaction in Java EE server)
Resource 1
Transaction
Resource 2
Resource 3
Transaction spans multiple resources

JTA transaction Types



CMT: default transction mode

```
 */
@Stateless
@LocalBean
@TransactionManagement(TransactionManagementType.CONTAINER)
public class TransBean {

    @PersistenceContext(unitName = "com.mycompany_MyApp_ejb_1.0-SNAPSHOTPU")
    private EntityManager em;

    public void saveAnimal() {
        Animal a = new Animal();
        a.setTotalNo(5);
        a.setType("Monitor lizard");
        em.persist(a);

    }
}
```

```
@Singleton
@Startup
public class MyTester {

  @EJB
  TransBean myBean;

   @PostConstruct
   public void myMain(){

      myBean.saveAnimal();

   }
```

BMT: We must start transaction

```
@Stateless
@LocalBean
@TransactionManagement(TransactionManagementType.BEAN)
public class TransBean {

    @PersistenceContext(unitName = "com.mycompany_MyApp_ejb_1.0-SNAPSHOTPU"
    private EntityManager em;

    public void saveAnimal() {
        Animal a = new Animal();
        a.setTotalNo(5);
        a.setType("Monitor lizard");
        em.persist(a);

    }
}
```

```
lassFish Server 3+   MyApp
        at com.sun.ejb.containers.AbstractSingletonContainer.createSingletonEJB(Abstract
        ... 39 more
aused by: javax.persistence.TransactionRequiredException
```

Using the begin, commit, and rollback methods

```
@Stateful
@TransactionManagement(TransactionManagementType.BEAN)
public class CalculationBean implements CalculationLocal {
    @Resource
    UserTransaction ut;

    public void calculate(…) throws SystemException {
        try {
            ut.begin(); // Transaction starts here
            //... transaction operation 1, 2, 3
            ut.commit(); // Transaction finishes here
        } catch (Exception e) {
            ut.rollback(); // Oops: roll back
        }
    }
}
```

Type of transaction attributes

```
@Resource
private Use
@Persistenc
private Ent

@TransactionAttribute()
public void
    try {
        tx.
        Ani
        a.s
        a.s
        em.
        tx.
    } catch
    }
```

```
TransactionAttributeType.MANDATORY                    TransactionAttributeType
TransactionAttributeType.NEVER                        TransactionAttributeType
TransactionAttributeType.NOT_SUPPORTED                TransactionAttributeType
TransactionAttributeType.REQUIRED                     TransactionAttributeType
TransactionAttributeType.REQUIRES_NEW                 TransactionAttributeType
TransactionAttributeType.SUPPORTS                     TransactionAttributeType
value = javax.ejb.TransactionAttributeType.REQUIRED   TransactionAttributeType
false
true
AbstractMethodError
Animal
Appendable
ArithmeticException
ArrayIndexOutOfBoundsException
ArrayStoreException
```

ssFish Server 3+

CMT transaction attributes

CMT Transaction Attributes

CLIENT Client in a Transaction Client without a transaction	EJB marked with REQUIRED EJB uses that transaction EJB creates a new transaction
CLIENT Client in a Transaction Client without a transaction	EJB marked with REQUIRES_NEW EJB creates a new transaction EJB creates a new transaction
CLIENT Client in a Transaction Client without a transaction	EJB marked with MANDATORY EJB uses that transaction EJB throws a exception
CLIENT Client in a Transaction Client without a transaction	EJB marked with NEVER EJB throws a exception EJB without a transaction
CLIENT Client in a Transaction Client without a transaction	EJB marked with SUPPORTS EJB uses that transaction EJB without a transaction
CLIENT Client in a Transaction Client without a transaction	EJB marked with NOt_SUPPORTED EJB without a transaction EJB without a transaction

Transaction management

Transaction Scoped EntityManager

Transaction Scoped Persistence Context

```
@Stateless
public class TransactioScoped {
   @PersistenceContext(unitName="somePU")
   EntityManager em;

   public void someMethod(){
       Animal a = new Animal();
       a.setType("Peacock");
       a.setTotalNo(4);

       em.persist(a);
   }

}
```

Persistence Context lasts as long as Transaction is active

Extended Entity Manager

Extended Persistence Context

```
@Stateful
public class ExtendedPersistenceContext {
   @PersistenceContext(unitName="somePU",
           type= PersistenceContextType.EXTENDED)
   EntityManager em;

   Animal animal;

   public void someMethod(){
       Animal a = new Animal();
       a.setType("Peacock");
       a.setTotalNo(4);

       em.persist(a);
       animal = a;
   }

    public void someOtherMethod(){
      animal.setTotalNo(3);
   }

    @Remove
    public void finish(){
      animal.setTotalNo(3);
   }
}
```

Persistence Context last as long as Bean is alive

Application Managed EntityManager

Application Managed Persistence Context

```
public class MyJPAApp {

    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MyJPAAppPU");
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        Member member = new Member();
        member.setName("john");
        member.setAge(25);
        member.setGender("M");
        member.setSalary(25000);
        em.persist(member);
        em.getTransaction().commit();

        em.close();
        emf.close();
    }
}
```

Persistence Context extends from creation of entity manager to its closing

Persistance context and managed entities

Persistence Context and Managed Entities

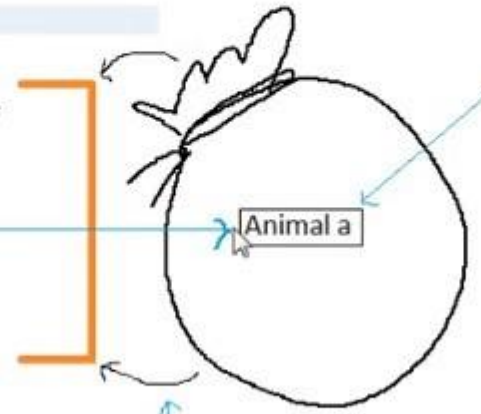
```
@Stateless
public class MyBean                    {
   @PersistenceContext(unitName="somePU")
   EntityManager em;

   public void someMethod(){
       Animal a = new Animal();
       a.setType("Peacock");
       a.setTotalNo(4);

       em.persist(a);


       a.setTotalNo(3);
   }

}
```



Because of EntityManager operation, Animal a goes into the Persistence Context and becomes 'managed'. All changes made to managed entites are persisted

This PersistenceContext lasts as long as same transaction is alive

What is persistance context?

Persistence Context

1) Persistence Context is a set of managed entities.

2) as long as a entity remains in a Persistence context, changes made to it are persisted.

Transaction scoped EntityManager

3)Persistence Context latches on to a transaction. As long as a transaction is alive, the persistence context is alive. When transaction ends, persistence context gets destroyed and all entites in it become unmanaged or detached.

Scope of persistance context

```
@Stateful
public class ExtendedPersistenceContext {
   @PersistenceContext(unitName="somePU")

   EntityManager em;

   Animal animal;

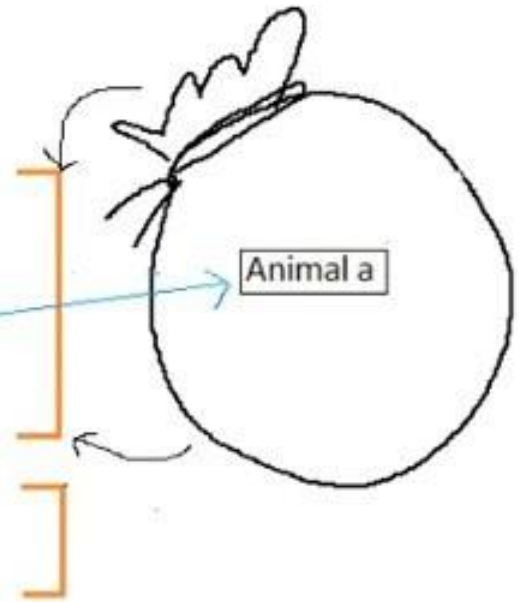
   public void someMethod(){
       Animal a = new Animal();
       a.setType("Peacock");
       a.setTotalNo(4);

       em.persist(a);
       animal = a;
   }

    public void someOtherMethod(){
      animal.setTotalNo(3);
   }

    @Remove
    public void finish(){

   }
}
```



Extended persistance context

```
@Stateful
public class ExtendedPersistenceContext {
   @PersistenceContext(unitName="somePU",
           type= PersistenceContextType.EXTENDED)
   EntityManager em;

   Animal animal;

   public void someMethod(){
       Animal a = new Animal();
       a.setType("Peacock");
       a.setTotalNo(4);

       em.persist(a);
       animal = a;
   }

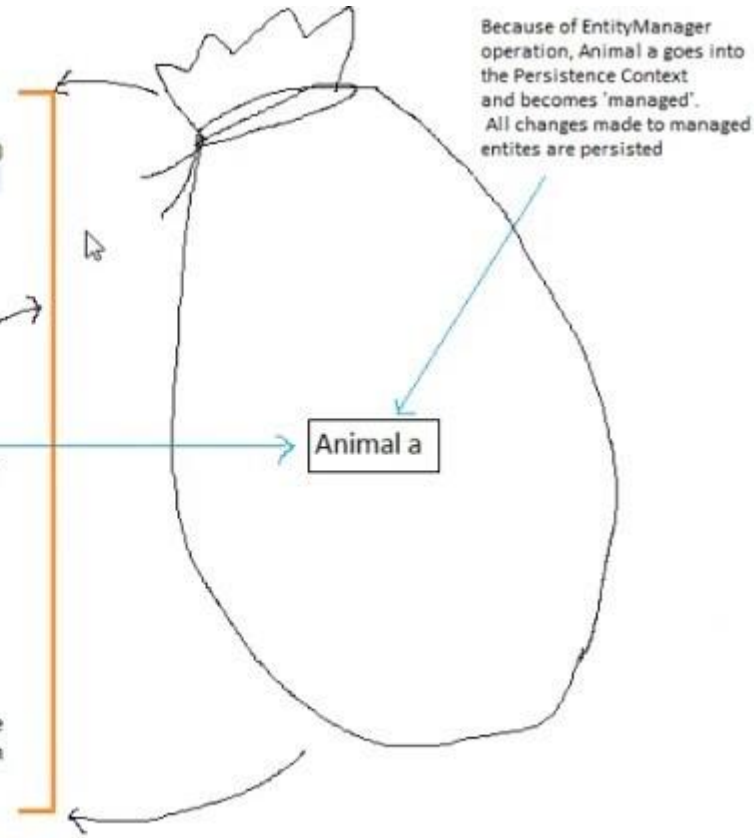
    public void someOtherMethod(){
      animal.setTotalNo(3);
   }

    @Remove
    public void finish(){

   }
}
```

Persistence Context last as long as Bean is alive

Change saved since PersistenceContext is scoped to the lifetime of the bean and not the transaction



Because of EntityManager operation, Animal a goes into the Persistence Context and becomes 'managed'. All changes made to managed entites are persisted

Handling exceptions

Handling System Exceptions

- EJB Container throws it when encounters an internal application failure
- Application throws it to abort business process
- Subclasses: EJBException, EJBTransactionRolledBackexception
- Always cause a transaction to roll back

- The container handles automatically :
    1. Roll back the transaction
    2. Log exception
    3. Discard EJB instance

Handling System Exceptions

- If transaction is started, a system exception (thrown by the enterprise bean method) will be caught by the container and rethrown as an **EJBTransactionRolledbackException**
- If the client did not propagate a transaction to the EJB, the system exception will be caught and rethrown as an **EJBException**

- Always delivered directly to the client without being repackaged as an EJBException type
- By default, does not cause a transaction to roll back
- To force application Exception to transaction roll back **@javax.ejb.ApplicationException**

```
@ApplicationException(rollback = true)
public class MyException extends Exception
{
    ...
}
```

- Can be used with checked and unchecked exceptions, exception will not be wrapped into EJBException