

компьютерного
Центр[®]
(ОБУЧЕНИЯ)
«СПЕЦИАЛИСТ»
при МГТУ им. Н.Э.Баумана

Creating Session Beans

Objectives

After completing this lesson, you should be able to:

- Describe session beans
- Create stateless and stateful session beans by using annotations
- Understand the passivation and activation of stateful session beans
- Use interceptor methods and classes

What Is a Session Bean?

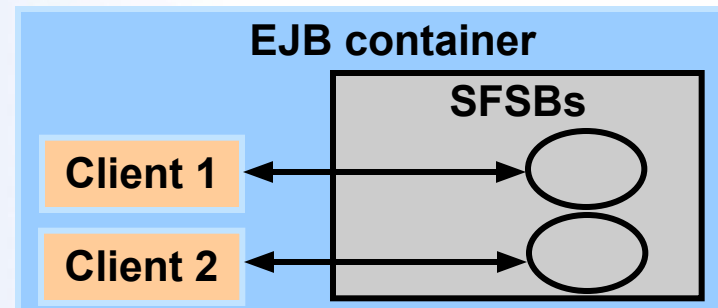
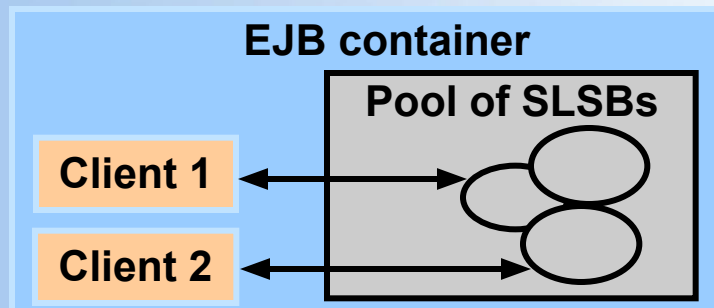
A session bean is a type of Enterprise JavaBean (EJB) that:

- Implements a business process
- Represents a client/server interaction
- Has a short lifespan
- Lives in memory rather than in persistent storage
- Is used to create a session facade

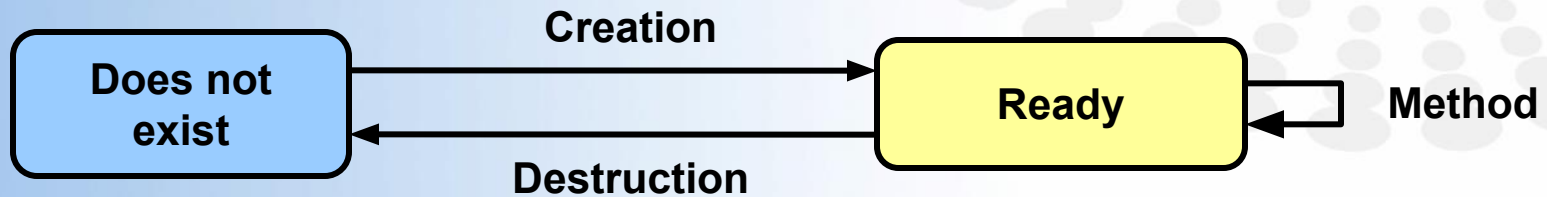
Stateless Versus Stateful Session Beans

There are two types of session beans:

- Stateless session bean (SLSB)
 - Conversation is contained in a single method call.
 - Business process does not maintain client state.
- Stateful session bean (SFSB)
 - Conversation may invoke many methods.
 - Business processes can span multiple method requests, which may require maintaining state.



Life Cycle of a Stateless Session Bean



Creating a Stateless Session Bean

To create a stateless session bean:

1. Define the stateless session bean.
2. Define the local and remote interfaces (as needed).

Define the Stateless Session Bean

```
// HelloWorldBean.java
package helloworld.ejb
import javax.ejb.Stateless;
@Stateless(name="HelloWorld")
public class HelloWorldBean implements HelloWorld
{
    public void sayHello()
    {
        System.out.println("Hello World!");
    }
}
```

1

2

4

3

5

Create the Remote and Local Interfaces

```
// HelloWorld.java
package helloworld.ejb 1
import javax.ejb.Remote; 2
@Remote 3
public interface HelloWorld { 4
    public void sayHello();
}
```

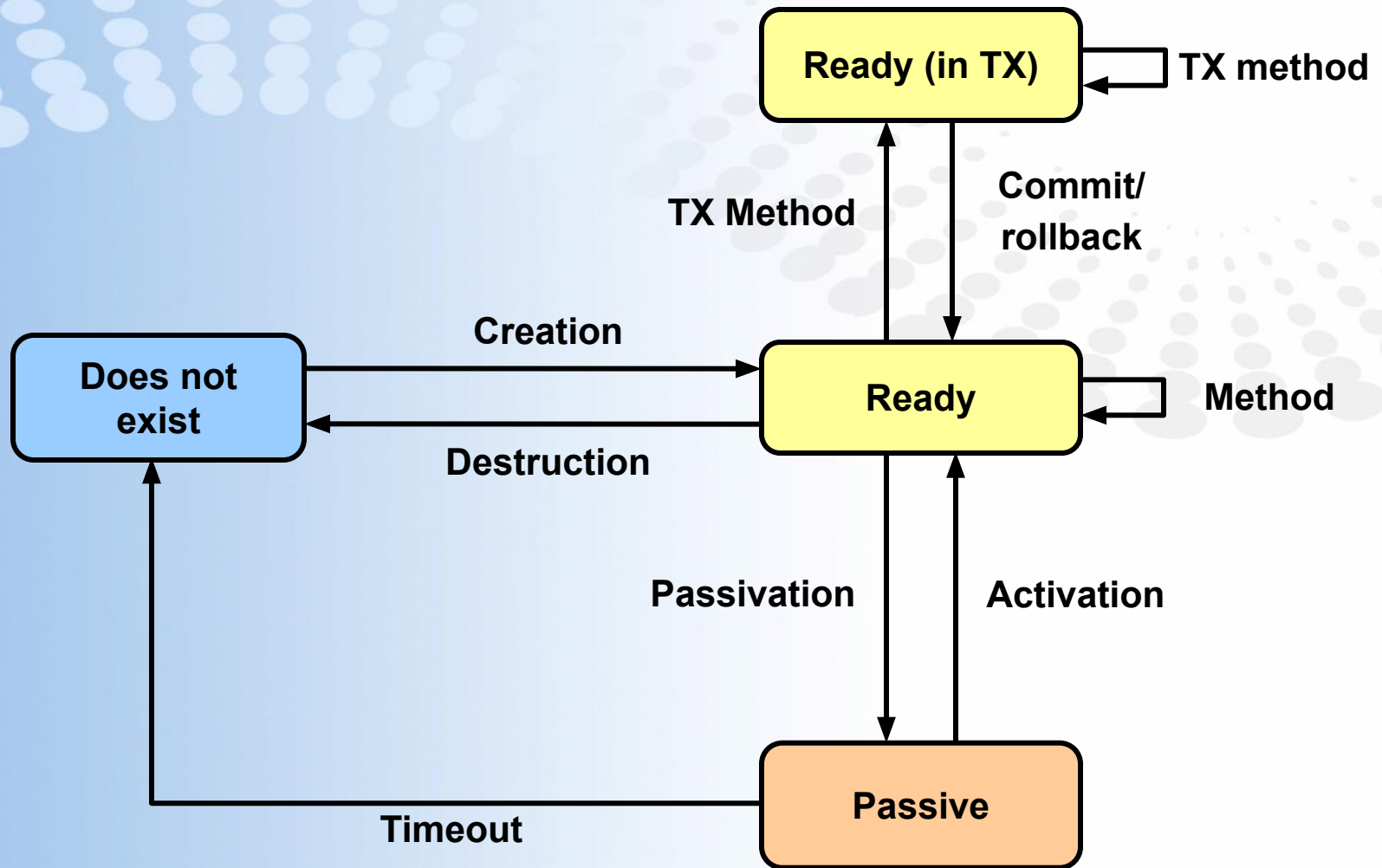
```
// HelloWorldLocal.java
package helloworld.ejb 1
import javax.ejb.Local; 2
@Local 3
public interface HelloWorldLocal { 4
    public void sayHello();
}
```


Create a Test Client for the SLSB

```
// HelloWorldClient.java
import helloworld.ejb;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class HelloWorldClient {
    public static void main(String [] args)
        throws NamingException {
        try {
            final Context context = new InitialContext();
            HelloWorld helloWorld =
                (HelloWorld) context.lookup("HelloWorld");
            helloWorld.sayHello( );
        } catch (Exception ex) { ex.printStackTrace(); }
    };
}
```

Life Cycle of a Stateful Session Bean





Passivation and Activation Concepts

Passivation and activation are stages in a session bean's life cycle controlled by the EJB container:

- Passivation
 - Serializes the bean state to secondary storage
 - Removes the instance from memory
- Activation
 - Restores the serialized bean's state from secondary storage
 - Creates a new bean instance or uses a bean from the pool (initialized with the restored state)

Creating a Stateful Session Bean

To create a stateful session bean:

1. Define the stateful session bean.
2. Define the local and remote interfaces (as needed).

Define the Stateful Session Bean

```
// CartBean.java
package cart.ejb 1
import javax.ejb.Stateful; 2
...
@Stateful(name="Cart") 3
public class CartBean implements Cart {
    private ArrayList items;

    @PostConstruct 4
    public void initialize() { items = new ArrayList(); }
    public void addItem(String item) { items.add(item); }
    public void removeItem(String item) {
        items.remove(item); }
    public Collection getItems() { return items; }
    @Remove 5
    public void dumpCart() {System.out.println("BYE!");};
}
```

Create the Remote and Local Interfaces

```
// Cart.java
package cart.ejb
import javax.ejb.Remote;
...
@Remote
public interface Cart {
    public void addItem(String item);
    public void removeItem(String item);
    public Collection getItems();
}
```

```
// CartLocal.java
package cart.ejb
import javax.ejb.Local;
...
@Local
public interface CartLocal { ...
```

Create a Test Client for the SFSB

```
// CartClient.java
import ...
public class CartClient {
    public static void main(String[] args) throws
        Exception {
        Context context = new InitialContext();
        Cart cart = (Cart) context.lookup("Cart");
        cart.addItem("Item1");
        cart.addItem("Item2");
        Collection items = cart.getItems();
        for (Iterator i = items.iterator(); i.hasNext();) {
            String item = (String) i.next();
            System.out.println("  " + item);
        }
        cart.dumpCart();
    }
}
```


Interceptor Methods and Classes

EJB 3.0 introduces the ability to create custom interceptor methods and classes that are called before invoking the methods they intercept. Interceptors:

- Are available for only session beans (stateless and stateful) and message-driven beans
- Provide more granular control of a bean's method invocation flow
- Can be used to implement custom transaction or security processes instead of having those services provided by the EJB container
- Are a new feature whose implementation details are not fully defined and are subject to change

Interceptor Method

```
import javax.ejb.Stateless; 1  
import javax.ejb.AroundInvoke;  
import javax.ejb.InvocationContext;  
  
@Stateless  
public class HelloWorldBean implements HelloWorld  
{  
    @AroundInvoke 2  
    public Object myInterceptor(InvocationContext ctx) 3  
        throws Exception {  
        System.out.println("Ahem...");  
        return ctx.proceed(); 4  
    }  
  
    public void sayHello()  
    { System.out.println("Hello World!"); }  
}
```

Interceptor Classes

External interceptor classes can be created to abstract the behavior of interceptors and to define multiple interceptors for a bean.

```
// Bean Class
@Stateless
@Interceptor(CheckUserInterceptor.class)
@Interceptor(LogActivity.class)
public class HelloWorldBean implements HelloWorld
{... }
```

```
// Interceptor Class
public class CheckUserInterceptor {
    @AroundInvoke
    public Object checkId(InvocationContext ctx) {...}
}
```

Summary

In this lesson, you should have learned how to:

- Describe session beans
- Create stateless and stateful session beans by using annotations
- Understand the passivation and activation of stateful session beans
- Use interceptor methods and classes

Practice 5 Overview: Creating Session Beans

- This practice covers the following topics:
 - Using JDeveloper to generate `ServiceRequestFacade` as a stateless session facade for entities
 - Creating a Java client application for testing the session facade functionality