

Session: 5



Singleton Session Beans

For Aptech Centre Use Only



Objectives



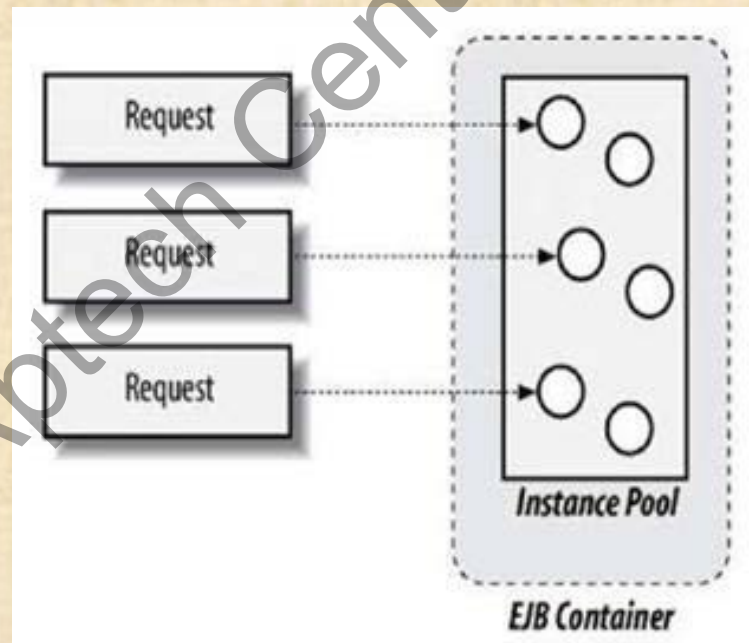
- ☐ Explain the need of Singleton Session bean
- ☐ Describe the characteristics of Singleton Session bean
- ☐ Describe the various stages in the lifecycle of Singleton Session bean
- ☐ Explain Singleton Session bean specification and its initialization strategies
- ☐ Describe how to achieve concurrency access to Singleton Session bean
- ☐ Explain container-managed concurrent access to Singleton Session bean
- ☐ Explain bean-managed concurrent access to Singleton Session bean
- ☐ Explain the mechanism to configure access time out in concurrency
- ☐ Explain how to implement Singleton Session bean in an enterprise application



Introduction 1-2



- ❑ In both, the Stateless Session bean and Stateful Session bean model:
 - Only single request can access the bean instance at any time. This means these beans are thread-safe, as each request is represented by the invocation of a single thread.
- ❑ Following figure demonstrates how session beans are accessed from a pool of bean instances:



Introduction 2-2



- ☐ There are some situations when a single shared instances is used by all the clients.
- ☐ This lead to the concurrence access to the bean instance.

EJB 3.1 specification have introduced Singleton Session bean which provides concurrent access to the clients.



Singleton Session Bean 1-2



❑ A Singleton Session bean:

- Is one which gets instantiated only once for every application.
- Exists during the entire lifecycle of an application.
- One or more clients can simultaneously access the same bean instance at the same time.

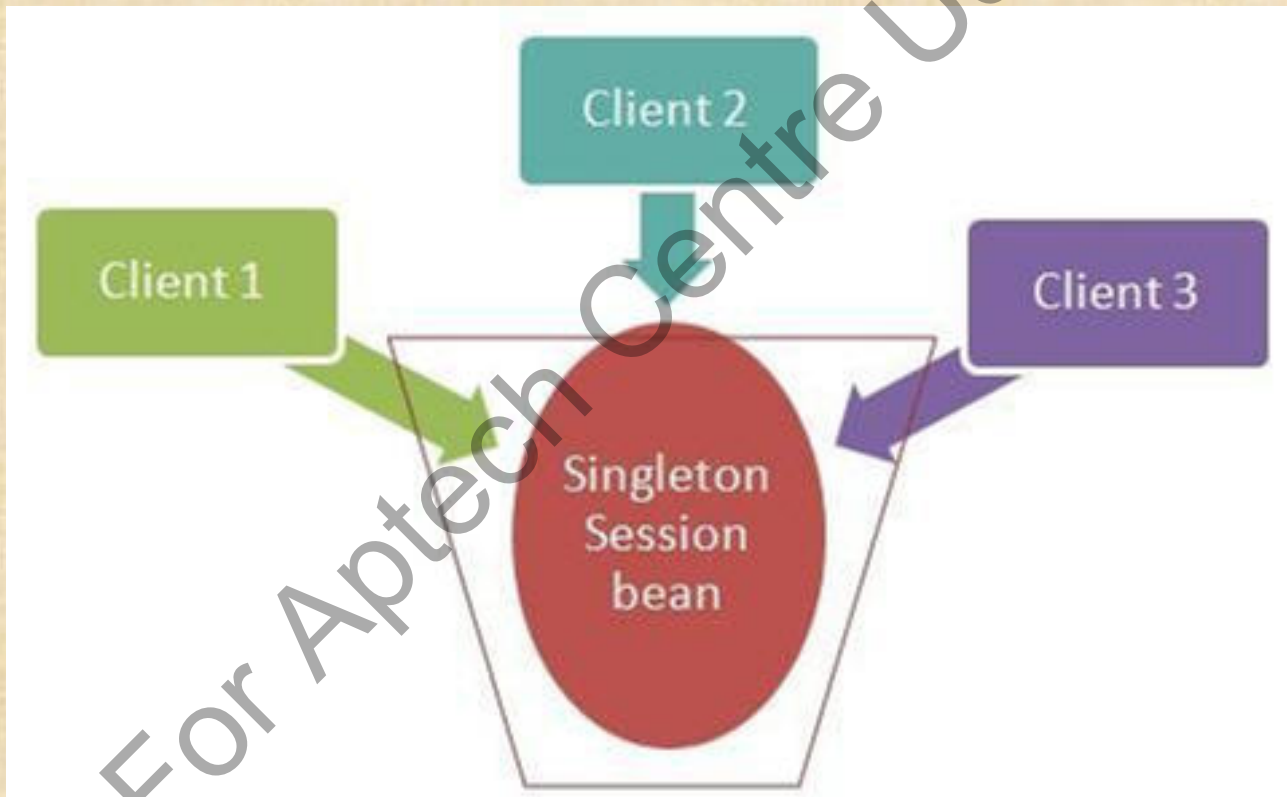
For Aptech Centre Use Only



Singleton Session Bean 2-2



- ❑ Following figure demonstrates how a single bean instance can be accessed by multiple clients:



Characteristics of a Singleton Session Bean



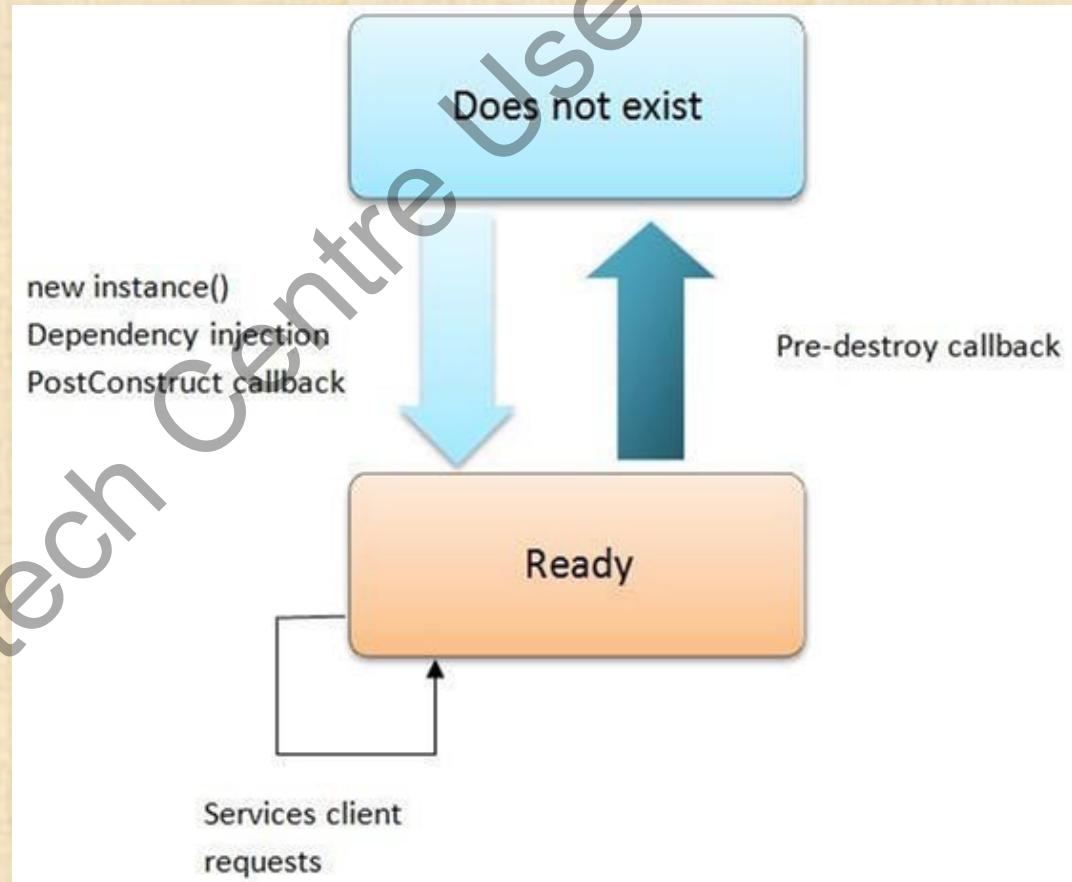
- ☐ It is shared by all requests.
- ☐ It is similar to Stateless Session bean.
- ☐ Like Stateless Session beans, the conversational state of Singleton Session beans cannot be persisted onto permanent storage.
- ☐ It's instance cannot be passivated and must be thread-safe.
- ☐ It has less memory foot print as compared to other session beans.
- ☐ Container is responsible for deciding when to initialize a Singleton bean instance.



Lifecycle of a Singleton Session Bean 1-3



- ❑ Following figure demonstrates the lifecycle of a Singleton Session bean:



Lifecycle of a Singleton Session Bean 2-3



❑ 'Does not Exist' State

- The bean remains in this state until it has been instantiated by the container.

❑ 'Ready' State

- After the container instantiates the bean instance, it moves to the 'Ready' state.
- Bean instance is ready to accept client requests, once it is instantiated and injected with required resources.

❑ Once the session bean is instantiated all the required resources for the bean class are acquired through dependency injection:

- **PostConstruct** callback methods are invoked by the container over the bean instance.
- When the application shut downs, the **PreDestroy** callback method is invoked by the application container, before removing the bean instance from the container.

For Apteck Centre Use Only



Lifecycle of a Singleton Session Bean 3-3



- ❑ Annotations supported by Singleton Session bean are:
 - `@javax.ejb.Startup` - A Singleton Session bean can be explicitly started by the container during the application startup by invoking a method annotated with `@Startup`.
 - `@javax.ejb.Singleton` - Is used to mark class declaration as a Singleton Session bean class.

For Aptech Centre Use Only



Singleton Session Bean Specification 1-3



- ❑ A Singleton Session bean comprises a bean class and one or more optional business interfaces.

Bean class

It is a standard Java class which needs to be marked with `@Singleton` annotation.

Business interface

The business interface can be a local or remote. Singleton Session bean also supports no-interface local view for the clients deployed on the application server.



Singleton Session Bean Specification 2-3



- ❑ Following code snippet demonstrates the creation of Singleton Session bean:

```
import javax.ejb.Singleton;
import javax.ejb.Startup;

@Singleton(name = "ItemCount")
public class ShoppingItemCount {
    private int counter = 0;

    // Increment number of shopper counter
    public void incrementCounter() {
        shopperCounter++;
    }
}
```



Singleton Session Bean Specification 3-3



```
// Return number of shoppers
public int getCounter() {
    return Counter;
}
// Reset counter
public void initializeCounter() {
    shopperCounter = 0;
}
}
```

- ❑ The code annotates the `ShoppingItemCount` class with `@Singleton` annotation. The `name` attribute of the annotation defines the `ejb-name` of the bean by which it is referenced by other beans in the container.



Initialize Singleton Session Bean



- ❑ By default, EJB container is responsible for initializing the Singleton Session bean instance.
- ❑ The bean developer can also configure when to initialize the Singleton Session bean instance.
- ❑ If the bean is annotated with `@Startup`, then the bean is initialized at application startup.
- ❑ This initialization is also known as **eager initialization**, where the container initializes the Singleton Session bean as soon as the application startup.
- ❑ Following code snippet demonstrates the usage of `@Startup` annotation:

```
@Startup
@Singleton
public class ShoppingItemCount {
    . . .
}
```

- Instructs the container to configure and initialize the Singleton Session bean during the startup process of the application.



Startup Initialization Dependencies 1-2



Multiple Singleton Session beans components in an application can have explicit initialization ordering dependencies upon each other.

The container must define the sequence of initialization to address these dependencies.

`javax.ejb.DependsOn` annotation is used to ensure that a specific Singleton Session bean is initialized, before other Singleton Session beans dependent on it are initialized by the container.

The `value` attribute of `DependsOn` attribute holds the Singleton Session bean class name.

Based on this annotation, the container ensures that the dependency is satisfied, before the dependent beans are initialized.



Startup Initialization Dependencies 2-2



- ❑ Following code snippet demonstrates the usage of DependsOn annotation:

```
// Class definition of ShoppingItemPoints
@Singleton
@Startup
@DependsOn("ItemCount")
```

```
public class ShoppingItemPoints {
    private final int points;
    public void AddPoints() {
        // Business logic for adding points
    }
}
```

- The ShoppingItemPoints class has been annotated with the @DependsOn annotation indicating that the instantiation of the bean object is dependent on the ShoppingItemCount object.



LifeCycle Callback Methods 1-2



- ❑ Callback events handle the construction and destruction of a Singleton Session bean.
- ❑ Callback events are mapped to the following events:

PostConstruct

- It is denoted by `@PostConstruct` annotation and is fired after a bean instance is instantiated by the container, and before the invocation of the first business method defined in the class.

PreDestroy

- It is denoted by `@PreDestroy` annotation and is fired when the application is shutting down.



LifeCycle Callback Methods 2-2



- ❑ Following code snippet demonstrates the PostConstruct and PreDestroy interceptor methods:

```
...  
@Singleton(name = "ItemCount")  
@Startup  
public class ShoppingItemCount {  
...  
@PostConstruct  
public void applicationStartup() {  
    System.out.println("ApplicationStartup - Initializing the  
    counter variable to zero.");  
    Counter = 0;  
}  
  
@PreDestroy  
public void applicationShutdown() {  
    System.out.println("ApplicationShutdown Happening");  
}  
}
```

- The code handles the application startup and application shutdown in the callback interceptor methods.

Concurrency in Singleton Session Bean 1-2



- ❑ Every application has only one instance of a Singleton Session bean.
- ❑ When multiple clients access the same instance of the Singleton Session bean, then this results into **concurrency**.
- ❑ There are two ways by which the concurrent access to a singleton session bean can be controlled. These are as followed:
 - **Container-managed concurrent** - As the name suggests, the container is responsible for managing the concurrent access to the Singleton Session bean data or methods. This is the default concurrency management type.
 - **Bean-managed concurrent** – This container provides full access of the bean to the bean developer. Even the synchronization of bean state is managed by the bean developer.

For Aptech Certified Use Only



Concurrency in Singleton Session Bean 2-2



- ❑ The concurrency management method used for the current enterprise bean can be defined through the annotation `javax.ejb.ConcurrencyManagement`.
- ❑ The type of concurrency management can be specified through a `type` attribute whose value can be either set to `javax.ejb.ConcurrencyManagementType.CONTAINER` or `javax.ejb.ConcurrencyManagementType.BEAN`.

For Aptech Centre Use Only



Container Managed Concurrency 1-3



- ❑ Associates each business method of the Singleton Session bean with a lock.
- ❑ Annotations are used to define the type of the lock to be acquired.
- ❑ `javax.ejb.Lock` is used to specify the required locks.
 - It has an attribute `javax.ejb.LockType`, which accepts the lock type as `READ` or `WRITE` for the bean methods.
 - `READ` lock provides shared access and `WRITE` lock provides exclusive access.
 - For example, marking the business method with `@Lock (LockType.READ)` grants shared access to the bean method.

For Aptech Centre Use Only



Container Managed Concurrency 2-3



- ❑ Following code snippet demonstrates container-managed concurrency applied to the Singleton Session bean:

```
. . .
@Singleton(name = "WebsiteVisitCount")
@Startup
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)

public class WebsitevisitCount{
. . . .
private int Counter;

// Increment number of visitors
@Lock(LockType.WRITE)
public void incrementCounter() {
    Counter++;
}
```

- The `@ConcurrencyManagement` annotation specifies that the concurrency type is set to `CONTAINER`.
- The `@Lock(LockType.WRITE)` acquires the `WRITE` lock which blocks the access of the method for all other clients, while one client is accessing it.



Container Managed Concurrency 3-3



```
// Return number of visitors  
@Lock (LockType.READ)  
public int getVisitorCount() {  
    return Counter;  
    . . .  
}
```

- The `@Lock (LockType.READ)` annotation is specified at the method-level to the `getVisitorCount()` which returns the number of users visited to the site.

For Aptech Centre Use Only



Bean Managed Concurrency



- ❑ Bean managed concurrency is specified through the annotation `ConcurrentManagementType.BEAN`.
- ❑ It is the responsibility of the bean developer to synchronize the state of the Singleton Session bean to avoid synchronization errors occurring due to the concurrent access.
- ❑ Bean developer uses `synchronized` and `volatile` primitive types for synchronization.

For Aptech Centre Use Only



Concurrent Access Timeout 1-2



- ❑ When a client has an exclusive lock, other clients are blocked from accessing the bean.
- ❑ A client cannot hold the lock for unlimited time.
- ❑ `@AccessTimeout` annotation can be used to specify the maximum amount of time for which a client can be blocked.
- ❑ `AccessTimeout` annotation has two attributes `value` and `timeUnit`.
 - The default time unit specified in the `value` attribute is in milliseconds.
 - The developer can change it to one of the constants provided in the `java.util.concurrent.TimeUnit`.
 - If the `AccessTimeout` value specified is -1, then it indicates that the client will block indefinitely until it gains access to the bean.

For Aptech Centre Use Only



Concurrent Access Timeout 2-2



- ❑ Following code snippet demonstrates the configuration of access time value for the Singleton Session bean:

```
// Increment number of visitors
@Lock(LockType.WRITE)
@AccessTimeout(value=120000)
public void incrementCounter() {
    Counter++;
}
```

- The code specifies the access time out for the `incrementCounter()` method to 120000 milliseconds.



Developing a Singleton Session Bean 1-6



- ❑ Following figure demonstrates creating a Singleton Session bean:

A screenshot of the 'New Session Bean' wizard dialog box. The 'Steps' pane on the left shows '1. Choose File Type' and '2. Name and Location'. The 'Name and Location' pane contains the following fields: 'EJB Name' with the value 'NewSessionBean', 'Project' with 'CounterExample-war', 'Location' with 'Source Packages', and 'Package' with 'Beans'. Under 'Session Type', there are three radio buttons: 'Stateless', 'Stateful', and 'Singleton'. The 'Singleton' option is selected and highlighted with a blue rectangular box. Below this, the 'Create Interface' section has a 'Local' checkbox which is unchecked. At the bottom of the dialog are five buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), 'Cancel', and 'Help'. A large, semi-transparent watermark 'For Aptech Centre Use Only' is overlaid diagonally across the entire dialog box.

A Singleton Session bean can be created by choosing the option **Singleton**.

Developing a Singleton Session Bean 2-6



- ❑ Following code snippet demonstrates a Singleton Session bean to add a variable to the bean:

```
package Beans;
import javax.ejb.ConcurrencyManagement;
import static javax.ejb.ConcurrencyManagementType.CONTAINER;
import javax.ejb.LocalBean;
import javax.ejb.Lock;
import static javax.ejb.LockType.WRITE;
import javax.ejb.Singleton;
@Singleton
@ConcurrencyManagement(CONTAINER)
@LocalBean
public class CountBean {
    static int visits=0;
    @Lock(WRITE)
    public int incrementCounter() {
        return ++visits;
    }
}
```

Developing a Singleton Session Bean 3-6



- ❑ Following code snippet demonstrates how a client can access a Singleton Session bean:

```
. . .  
public class Module5Client1{  
public Module5Client1() {  
    CountBean C = new CountBean();  
    System.out.println("Counter value seen by Client 1  
"+C.incrementCounter());  
}  
}
```

For Apteck Centre Use Only



Developing a Singleton Session Bean 4-6



- ❑ Following code demonstrates how a second client accesses the Singleton Session bean:

```
. . .  
public class Module5Client2{  
  
    public Module5Client2() {  
        CountBean C = new CountBean();  
        System.out.println("Counter value seen by  
Client 2 "+C.incrementCounter());  
    }  
}
```

- Earlier code snippet and the given code snippet are two Java classes that invoke the Singleton Session bean in the constructor of the class.
- Whenever an object of the client class is invoked, the Singleton Session bean is accessed.



Developing a Singleton Session Bean 5-6



- ❑ Following code snippet demonstrates multiple clients accessing a Singleton Session bean:

```
package Beans;  
public class ClientRequest {  
    public static void main(String[] args) {  
        Module5Client1 m1 = new Module5Client1();  
        Module5Client2 m3 = new Module5Client2();  
        Module5Client1 m2 = new Module5Client1();  
        Module5Client2 m4 = new Module5Client2();  
    }  
}
```

- The code creates four instances of clients which are equivalent to client requests to access the Singleton Session bean.



Developing a Singleton Session Bean 6-6



- ❑ Following figure shows the execution of multiple client requests by the Singleton Session bean:

A screenshot of an IDE's output window. The window has a title bar with 'Output' and a close button. Below the title bar, there are three tabs: 'Java DB Database Process', 'GlassFish Server', and 'CounterExample-'. The 'CounterExample-' tab is selected. The output text shows a sequence of four log messages: 'run:', 'Counter value seen by Client 1 11', 'Counter value seen by Client 2 12', 'Counter value seen by Client 1 13', and 'Counter value seen by Client 2 14'. The final line is 'BUILD SUCCESSFUL (total time: 1 second)' in green text. A large, diagonal watermark 'For Aptech Centre Use Only' is overlaid across the entire image.

```
run:
Counter value seen by Client 1 11
Counter value seen by Client 2 12
Counter value seen by Client 1 13
Counter value seen by Client 2 14
BUILD SUCCESSFUL (total time: 1 second)
```



Summary



- ❑ A Singleton Session bean is instantiated only once in the application lifecycle and retained throughout the application lifecycle.
- ❑ Singleton Session bean cannot store the conversational state nor can it be passivated.
- ❑ Singleton Session bean has `@PreDestroy` and `@PostConstruct` callback methods.
- ❑ When there are multiple Singleton beans in the application and they are dependent on each other, then the EJB container should initialize them in right order.
- ❑ `@DependsOn` annotation is used to define the dependencies among the Singleton beans.
- ❑ Multiple clients can access the Singleton Session bean in the application; this concurrent access has to be managed.
- ❑ Container-managed and Bean-managed concurrent access are the two variants of concurrency management for Singleton Session beans.
- ❑ The client requests cannot be blocked indefinitely, therefore an access time out value is associated with each blocked request.

