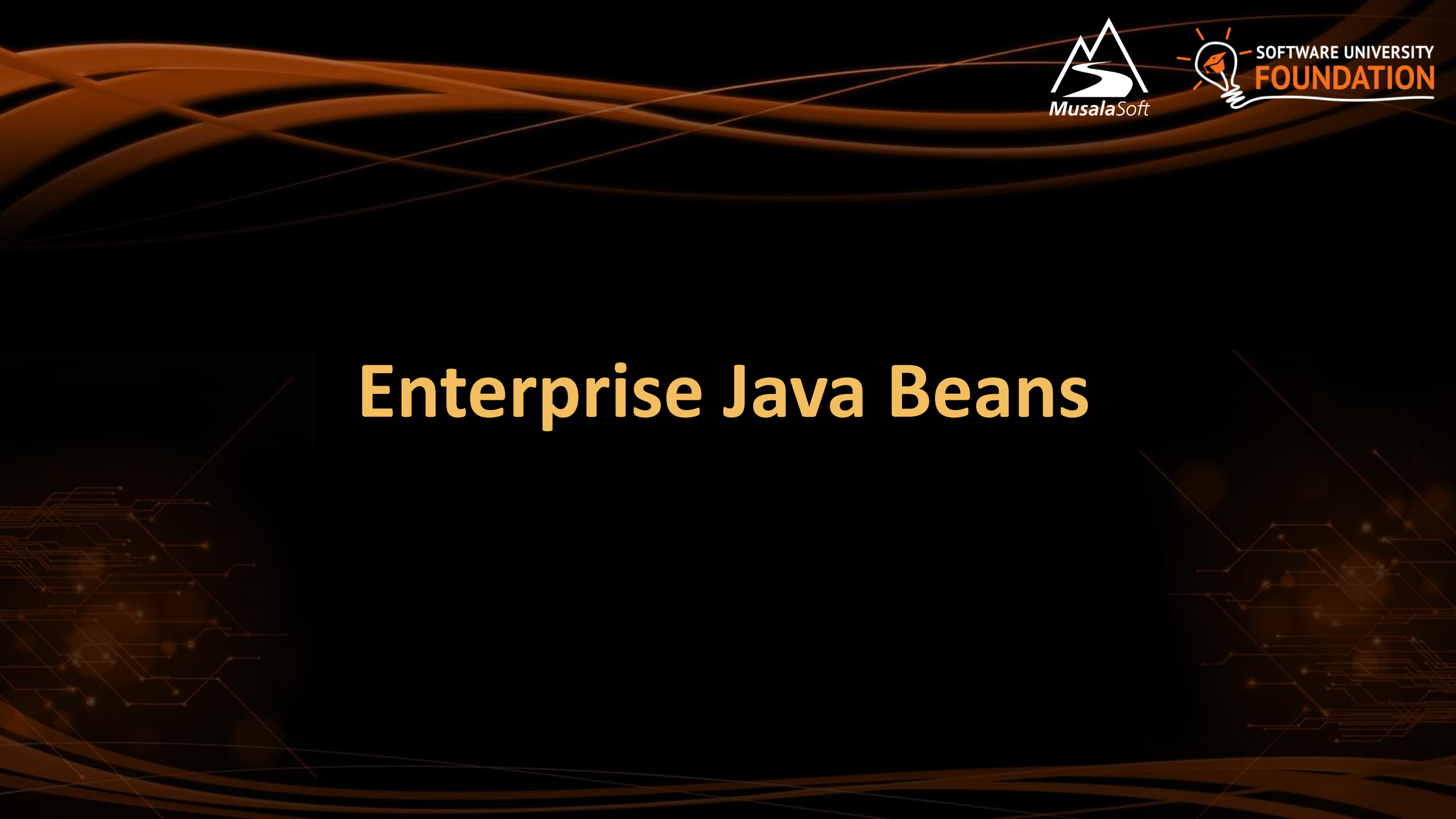


Enterprise Java Beans

Content

- Dependency Injection
- Enterprise Java Beans

Dependency Injection

Inversion of Control (IoC)

- Inversion of control (IoC) describes a design in which custom-written portions of a computer program receive the flow of control from a generic, reusable library.
- A software architecture with this design inverts control as compared to traditional procedural programming:
  - in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks
  - with inversion of control, it is the reusable code that calls into the custom, or task-specific, code

IoC Principles

- There are two principles of IoC:
  - Main classes aggregating other classes should not depend on the direct implementation of the aggregated classes.
  - Abstraction should not depend on details, details should depend on abstraction.

Dependency Injection (DI)

- The ability to inject components into an application in a typesafe way, including the ability to choose at deployment time which implementation of a particular interface to inject

- Since the Java EE 5 platform was introduced, annotations have made it possible to inject resources and some other kinds of objects into container-managed objects.

DI Usage

- The following kinds of objects can be injected:
  - (Almost) any Java class
  - Session beans
  - Java EE resources: data sources, Java Message Service topics, queues, connection factories, etc.
  - Persistence contexts (JPA EntityManager objects)
  - Producer fields
  - Objects returned by producer methods
  - Web service references
  - Remote enterprise bean references

Enterprise Java Beans

Enterprise Java Beans (EJB)

- EJB is an essential part of a Java EE platform.
- Java EE platform have component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.
- EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability and high performance.
- An EJB application can be deployed on any of the application server compliant with Java EE 1.3 standard specification.

EJB Benefits

- Simplified development of large scale enterprise level application.

- Application Server/ EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling and so on. Developer has to focus only on business logic of the application.

- EJB container manages life cycle of ejb instances thus developer needs not to worry about when to create/delete ejb objects.

EJB Types

- Session Bean – stores data of a particular user for a single session. Session bean gets destroyed as soon as user session terminates.
  - Stateful – keeps the state between calls
  - Stateless – independent calls (doesn't keep the state)
  - Singleton – single instance
- Entity Bean – represents persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.
- Message Driven Beans – used in context of Java Messaging Service (JMS). They can consumes JMS messages from external entities and act accordingly (allows asynchronous operations).

EJB Callbacks

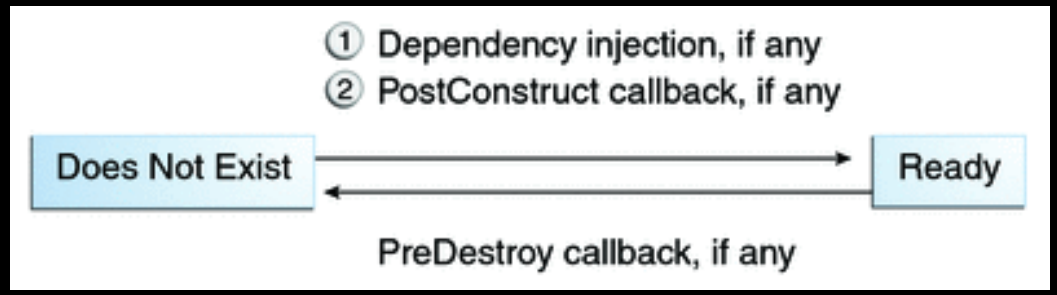
- Callback is a mechanism by which life cycle of an enterprise bean can be intercepted.
- EJB 3.0 specification has specified callbacks for which callback handler methods are to be created.
- EJB container calls these callbacks.
- We can define callback methods in the ejb class itself or in a separate class.
- EJB 3.0 has provided many annotations for callbacks

Stateless Beans

- A stateless session bean is a type of enterprise bean which is normally used to do independent operations.
- A stateless session bean as per its name does not have any associated client state, but it may preserve its instance state.
- EJB container normally creates a pool of few stateless bean's objects and use these objects to process client's request.
- Because of pool, instance variable values are not guaranteed to be same across lookups/method calls.

Stateless Beans Life Cycle

- Because a stateless session bean is never passivated, its lifecycle has only two stages: nonexistent and ready for the invocation of business methods.



- The EJB container typically creates and maintains a pool of stateless session beans, beginning the stateless session bean's lifecycle.

Stateless Beans Callbacks

- The container performs any dependency injection and then invokes the method annotated @PostConstruct, if it exists. The bean is now ready to have its business methods invoked by a client.

- At the end of the lifecycle, the EJB container calls the method annotated @PreDestroy, if it exists. The bean's instance is then ready for garbage collection.

Stateless Beans Programming

- Following are the steps required to create a stateless ejb:
  - Create a remote/local interface exposing the business methods. This interface will be used by the ejb client application.
  - Use @Local annotation if ejb client is in same environment where ejb session bean is to be deployed.
  - Use @Remote annotation if ejb client is in different environment where ejb session bean is to be deployed.
  - Create a stateless session bean implementing the above interface.
  - Use @Stateless annotation to signify it a stateless bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

Stateless Beans Example

- Local interface:

```
import javax.ejb.LocalBean;

@Local
public interface LocalBeanExample {
  //add business method declarations
}
```

- Stateless EJB

```
import javax.ejb.Stateless;

@Stateless
public class LocalBeanExampleImpl implements LocalBeanExample {
  //implement business method
}
```

Stateless Enterprise Java Beans

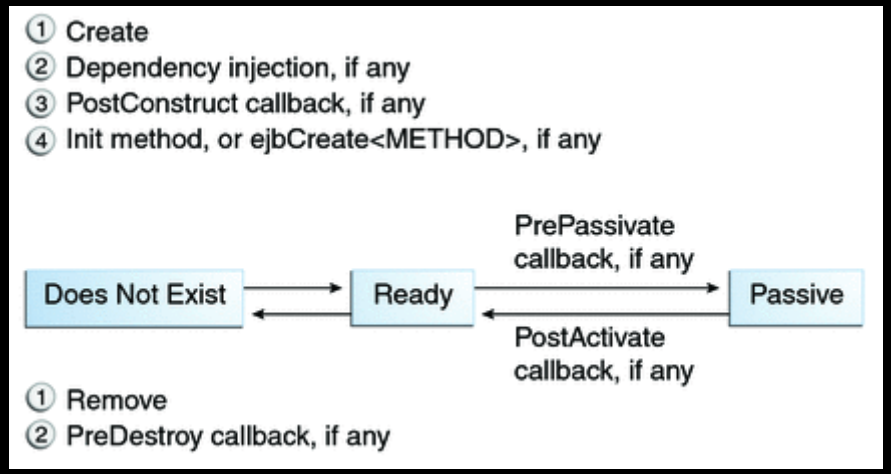
Demo

Stateful Beans

- A stateful session bean is a type of enterprise bean which preserve the conversational state with client.
- A stateful session bean as per its name keeps associated client state in its instance variables.
- EJB container creates a separate stateful session bean to process client's each request.
- As soon as request scope is over, stateful session bean is destroyed.

Stateful Beans Life Cycle

- The client initiates the lifecycle by obtaining a reference to a stateful session bean.



- The bean is now ready to have its business methods invoked by the client.

Stateful Beans Callbacks

- The container performs any dependency injection and then invokes the method annotated with @PostConstruct, if any.
- While in the ready stage, the EJB container may decide to deactivate, or passivate, the bean by moving it from memory to secondary storage. The EJB container invokes the method annotated @PrePassivate, if any, immediately before passivating it.
- If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean, calls the method annotated @PostActivate, if any, and then moves it to the ready stage.
- At the end of the lifecycle, the client invokes a method annotated @Remove, and the EJB container calls the method annotated @PreDestroy, if any. The bean's instance is then ready for garbage collection.

Stateful Beans Programming

- Following are the steps required to create a stateful ejb.
  - Create a remote/local interface exposing the business methods.
  - This interface will be used by the ejb client application.
  - Use @Local annotation if ejb client is in same environment where ejb session bean is to be deployed.
  - Use @Remote annotation if ejb client is in different environment where ejb session bean is to be deployed.
  - Create a stateful session bean implementing the above interface.
  - Use @Stateful annotation to signify it a stateful bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

Stateful Beans Example

- Local interface:

```
import javax.ejb.LocalBean;

@Local
public interface LocalBeanExample {
  //add business method declarations
}
```

- Stateless EJB

```
import javax.ejb.Stateful;

@Stateful
public class LocalBeanExampleImpl implements LocalBeanExample {
  //implement business method
}
```

Stateful Enterprise Java Beans

Demo

Singleton Beans

- A singleton session bean is instantiated once per application and exists for the lifecycle of the application.
- Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans.
- Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.
- Applications that use a singleton session bean may specify that the singleton should be instantiated upon application startup, which allows the singleton to perform initialization tasks for the application.

Singleton Beans Usage

- Singleton session beans are appropriate in the following circumstances.
  - State needs to be shared across the application.
  - A single enterprise bean needs to be accessed by multiple threads concurrently.
  - The application needs an enterprise bean to perform tasks upon application startup and shutdown.
  - The bean implements a web service.

Singleton Beans Example


```
import javax.annotation.PostConstruct;
import javax.ejb.LocalBean;
import javax.ejb.Singleton;
import javax.ejb.Startup;

@Startup
@Singleton
@LocalBean
public class SingletonEjb {

    @PostConstruct
    private void initialize() {

    }
}
```

Singleton Enterprise Java Beans

Demo