# Introduction to Java EE 5 and EJB 3.0

Mike Keith
michael.keith@oracle.com

**http://otn.oracle.com/ejb3**

## About Me

- Co-spec Lead of EJB 3.0 (JSR 220)
- Java EE 5 (JSR 244) expert group member
- Co-author “Pro EJB 3: Java Persistence API”
- Persistence/Container Architect for Oracle
- 15+ years experience in distributed, server-side and persistence implementations
- Presenter at numerous conferences and events

## About You

- How many people have already used Java EE 5 and/or EJB 3.0?
- How many people can’t because they are stuck on older versions of the JDK?
- How many people are using non-standard products with similar features?

## About Java EE 5

- Developed by JSR 244 expert group with Bill Shannon as spec lead

Mandate:

- To make enterprise Java programming much easier than it previously was
- Reduce the steep learning curve for new developers coming to Java EE
- Maintain high level of integration of all subcomponents within the  umbrella spec

## Reference Implementation

- "Glassfish" project on java.net
  - Also included in Java EE 5 SDK
- Sun and Oracle partnership
  - Sun Application Server + Oracle persistence
- All open source (under CDDL license)
  - Anyone can download/use source code or binary code in development or production

## Major Features

- Simplified EJB programming model
- Use of annotations for metadata
- Extensive defaults for common cases
- Dependency injection of resources
- Simplified web services support
- New Java Persistence API
- New Java ServerFaces component
- Fully compatible with J2EE 1.4

## Simplified EJB Components

- POJO development model
- Fewer programming constraints
- Vastly reduced metadata requirements
- Fewer interfaces and programming artifacts
- Component testability outside the server
- Easier access to resources and other components (through dependency injection)

## EJB 2.1 Remote Interface

```
package com.acme;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface AccountProcessor extends EJBObject {
   public void deposit(int accountId, Float amount)
      throws RemoteException;
   public Boolean withdraw(int accountId, Float amount)
      throws RemoteException;
   public Float getBalance(int accountId)
      throws RemoteException;
}
```

## EJB 2.1 Remote Home

```
package com.acme;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface AccountProcessorHome extends EJBHome {
   public AccountProcessor create()
      throws CreateException, RemoteException;
}
```

## EJB 2.1 Bean Class

```
package com.acme;
import javax.ejb.*;

public class AccountProcessorBean
                             implements SessionBean {
    SessionContext ctx;
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbSetSessionContext(SessionContext sc){
       this.ctx = sc;
    }
```

## EJB 2.1 Bean Class (cont’d)

```
   public void deposit(int accountId,Float amount) {
      // Do stuff
   }
   public Boolean withdraw(int accountId,Float amount) {
      // Do stuff
   }
   public Float getBalance(int accountId) {
      // Do stuff
   }
}
```

# EJB 2.1 Deployment Descriptor

```
<session>
   <ejb-name>AccountProcessorBean</ejb-name>
   <home> AccountProcessorHome</home>
   <remote>AccountProcessor</remote>
   <ejb-class>com.acme.AccountProcessorBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <resource-ref>
      <res-ref-name>jdbc/accountDB</res-ref-name>
      <res-ref-type>javax.sql.DataSource</res-ref-type>
      <res-auth>Container</res-auth>
   </resource-ref>
</session>
...
<assembly-descriptor>
   ...
</assembly-descriptor>
```

## EJB 3.0 Business Interface

```
package com.acme;
import javax.ejb.Remote;


@Remote
public interface AccountProcessor {

   public void deposit(int accountId,Float amount);
   public Boolean withdraw(int accountId,Float amount);
   public Float getBalance(int accountId);
}
```

## EJB 3.0 Bean Class

```
package com.acme;
import javax.ejb.Stateless;

@Stateless
public class AccountProcessorBean
                implements AccountProcessor {

   public void deposit(int accountId,Float amount) {
      // Do stuff
   }

   public Boolean withdraw(int accountId,Float amount) {
      // Do stuff
   }
   public Float getBalance(int accountId) {
      // Do stuff
   }
}
```

## Where Did it All Go?

- Home interface no longer needed
  - Bean creation happens automatically
- Very little metadata required
  - Configuration values are defaulted
  - Need only specify exceptions to defaults
- Life cycle methods only used when needed
- Extends/implements contraints loosened
- Runtime exceptions for cleaner user code

# EJB 3.0 Stateful Session Bean

```
package com.acme;
import javax.ejb.*;

@Stateful
public class AccountProcessorBean
                implements AccountProcessor {
   Account account;


   public void initalize(Account acct) { ... }
   public void deposit(Float amount) { ... }
   public Boolean withdraw(Float amount) { ... }
   public Float getBalance() { ... }
   @Remove
   public void completeProcessing() { ... }
}
```

## Dependency Injection (DI)

- Instance of “Inversion of Control” (IoC)
- Program defines dependencies and relies upon the container to supply them
- Annotations are useful for specifying DI because they are co-located with the code
  - May use XML if preferred
- Set of resources that may be injected is defined by the Java EE specification

# DI Support

- *@Resource*
  - Connection factories, env entries, UserTransaction, EJBContext, etc.
- *@EJB*
  - EJB business (and home) interfaces
- *@PersistenceContext, @PersistenceUnit*
  - EntityManager and EntityManagerFactory
- *@WebServiceRef*
  - Web service references

## Field Injection

```
@Stateless
public class AccountProcessorBean
                                implements AccountProcessor {

   @Resource
   protected SessionContext ctx;

   public Boolean withdraw(int accountId,Float amount) {
      if (!ctx.getCallerPrincipal().getName()
                                      .equals("MikeKeith"))
         throw new WithdrawalException(
           “All your money has been donated to Mike");

      ...

   }
   ...
}
```

## Setter Injection

```
@Stateless
public class AccountProcessorBean
                        implements AccountProcessor {
   protected EntityManager em;

   public EntityManager getEntityManager() {
      return this.em;
   }
   @PersistenceContext(unitName="Accounts")
   public void setEntityManager(EntityManager em) {
      this.em = em;
   }
   public Boolean withdraw(int accountId,Float amount) {
      Account acct = em.find(Account.class, accountId);
      ...
   }
   ...
}
```

## The Manual Approach

```
@Stateless
@Resource(name="jdbc/accountDataSource")
public class AccountProcessorBean
                                implements AccountProcessor {
    @Resource
    protected SessionContext ctx;

    public void deposit(int accountId, Float amount) {
        DataSource ds = (DataSource)
                ctx.lookup("jdbc/accountDataSource");
        Connection conn = ds.getConnection();
        ...
    }
    ...
}
```

# Invoking an EJB

```
@Stateless
public class AuditServiceBean implements AuditService {
    ...
}


@Stateless
public class AccountProcessorBean
                        implements AccountProcessor {
    @EJB
    AuditService audit;

    public void deposit(int accountId, Float amount) {
        ...
        audit.deposit(accountId,amount);
    }
    ...
}
```

## Callback Methods

- Callbacks occur at a given life cycle state
- Only receive notification for events that apply to the component type
  - SLSB – PostConstruct, PreDestroy
  - MDB – PostConstruct, PreDestroy
  - SFSB – PostConstruct, PreDestroy, PrePassivate, PostActivate
- Can handle events in the component class or in a separate interceptor class

## Callback Methods

```
@Stateful
public class AccountProcessorBean
                                implements AccountProcessor {

   Account account;
   AuditSession audit;

   @PostConstruct @PostActivate
   private void initAuditSession() {
      audit = AuditSessionFactory.createSession();
   }
   @PreDestroy @PrePassivate
   private void cleanUpAuditSession() {
      audit.close();
   }
   ...
}
```

## Interceptors

- Interceptors provide interposition points across business method execution
- Similar to AOP ***around*** advice
- Can chain any number of interceptors together
- Control over business method execution:
  - Manipulate arguments and results
  - Pass context data to other interceptors
  - Veto the operation

## Interceptor Scoping

- Default interceptors
  - Apply to all business methods of all EJB components in the ejb-jar
  - Specified in deployment descriptor (no place for application-level annotations)
- Class-level interceptors
  - Apply to all business methods of bean class
- Method-level interceptors
  - Apply to specific business method only

## Interceptor Class

```
public class Profiler {

  @AroundInvoke
  public Object profile(
           InvocationContext inv) throws Exception {

    long start = System.currentTimeMillis();
    try { return inv.proceed(); }
    finally {
      long elapsed = System.currentTimeMillis() - start;
      Method m = inv.getMethod();
      Logger.getLogger("AcctLog").info(m.toString() +
                           ": " + elapsed + " millis");
    }
  }
}
```

## Adding an Interceptor

```
@Stateless
@Interceptors(Profiler.class)
public class AccountProcessorBean
                    implements AccountProcessor { ... }
```

or…

```
@Stateless
public class AccountProcessorBean
                    implements AccountProcessor {

   @Interceptors(Profiler.class)
   public Boolean withdraw(int accountId,Float amount) {
      ...
   }
   ...
}
```

## IDE Support

- Free IDE's:
  - **Sun NetBeans (5.5)**
    - Java EE 5 and EJB 3.0 support
    - **http://community.java.net/netbeans**
  - **Oracle JDeveloper (10.1.3.1)**
    - Java EE 5 and EJB 3.0 support
    - **http://otn.oracle.com/jdev**
- For purchase:
  - **JetBrains IntelliJ Idea (6.0)**
    - Java EE 5 and EJB 3.0 support
    - **http://www.jetbrains.com/idea/index.html**

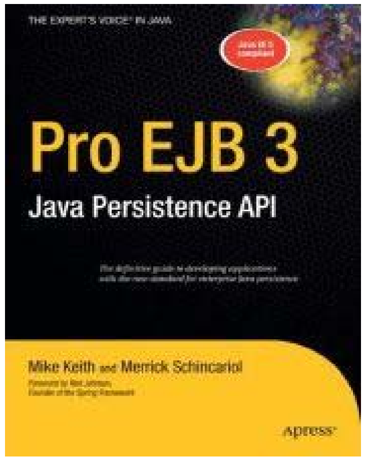
## Summary

- ✓ Java EE 5 and EJB 3.0 are not only much easier to use but also more powerful
- ✓ Programming model requires fewer artifacts and is less constrained
- ✓ Leverages dependency injection pattern for simple and effective resource access
- ✓ Minimal metadata is required, with the choice of using either annotations or XML
- ✓ Flexible callback and interceptor mechanisms

# Links and Resources

- Glassfish and Java EE 5 resources
  **http://glassfish.dev.java.net/**
- EJB 3.0 white papers, tutorials and examples
  **http://otn.oracle.com/ejb3**
- Pro EJB 3: Java Persistence API

  Mike Keith & Merrick Schincariol
  (Apress)


THE EXPERT'S VOICE® IN JAVA
Pro EJB 3
Java Persistence API
Mike Keith and Merrick Schincariol
Apress®