Annotations in Java, JBoss and our first JEE program

Annotations in Java

Annotations in Java

- They have been introduced as part of Java 5 (inspired by XDoclet, Qdocs, JavaDoc)
- An annotation is an information written within the code
  - It can be associated to:
    - A class
    - A class member
    - A parameter
    - A variable
    - A package
  - Information contained in an annotation are available at:
    - Programming time
    - Compile time
    - Execution time (through reflection)
  - They are directed to:
    - The IDE
    - The compiler
    - The virtual machine
    - The application

Annotations in Java (2)

- When sources are compiled, the annotations are stored as metadata within the class
- The JVM (or another program) can use them to understand how to interact and use the instances of a class
- Syntax:
  - @Annotation(parameter)

Annotation: advantages and drawbacks

- Advantages
  - Declarative programming within a procedural programming language
- Drawbacks
  - Performance
  - Missing standards for metadata
    - Which ones, how, when?

Anotations in EJB3

- Provide information used by the container to understand how to manage the EJB lifecycle
  - Including the interaction with primary services (security, transactions, naming, …)

- Let us see a simple example…

Our first program
Titan Cruises

Titan Cruises

- Titan Cruises is a cruises line
- It incorporates various business
  - Management of cabins (similar to hotel rooms management)
  - Catering management
  - Management of entertainment opportunities
  - Interaction with turistic operators

- We focus on how to manage the reservation of a cabin (in a very simple and partial way)

Our first entity and session beans

- I step:
  - Creation of the Cabin entity bean
    - It contains the data and the logic to manage a single cabin on the boat

- II step:
  - Creation of TravelAgent session bean
    - It contains the logic to manage the creation and search of cabins

Cabin.java

```
package com.titan.domain;

import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.Column;
import javax.persistence.Id;

@Entity
@Table(name="CABIN")
public class Cabin implements
    java.io.Serializable {
    private int id;
    private String name;
    private int deckLevel;
    private int shipId;
    private int bedCount;
```

```
    @Id
@Column(name="ID")
public int getId()  {
            return id;
}
public void setId(int pk)  {
            id = pk;
}

@Column(name="NAME")
public String getName()  {
            return name;
}
public void setName(String str) {
            name = str;
}
```

Cabin.java

```
   @Column(name="DECK_LEVEL")
public int getDeckLevel() {
                return deckLevel;
    }
public void setDeckLevel(int level) {
                deckLevel = level;
}

@Column(name="SHIP_ID")
public int getShipId() {
                return shipId;
}
public void setShipId(int sid) {
                shipId = sid;
}
```

```
   @Column(name="BED_COUNT")
public int getBedCount() {
                return bedCount;
}
public void setBedCount(int bed) {
                bedCount = bed;
}
}
```

Entity Beans annotations

- @javax.persistence.Entity
  - Used to state that for the class an O/R mapping is defined
- @javax.persistence.Table
  - Indicates the name of the table related to the class
- @javax.persistence.Column
  - Indicates the name of the table column to which the property refers (it is defined next to the getter method)
- @javax.persistence.Id
  - Indicates that a property is related to the primary key of the table
- @javax.persistence.GeneratedValue
  - Indicates that the container (o the DBMS) will have to generate the corresponding value automatically

TravelAgentRemote.java

```java
package com.titan.travelagent;

import javax.ejb.Remote;
import com.titan.domain.Cabin;

@Remote
public interface TravelAgentRemote
{
    public void createCabin(Cabin cabin);
    public Cabin findCabin(int pKey);
}
```

TravelAgentBean.java

```
package com.titan.travelagent;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import com.titan.domain.Cabin;
import org.jboss.annotation.ejb.RemoteBinding;

@Stateless
@RemoteBinding(jndiBinding="TravelAgentRemote")
public class TravelAgentBean implements TravelAgentRemote {
  @PersistenceContext(unitName="titan") private EntityManager manager;

    public void createCabin(Cabin cabin) {
        manager.persist(cabin);
    }

    public Cabin findCabin(int pKey) {
        return manager.find(Cabin.class, pKey);
    }
}
```

See next slide

persistence.xml

- Used to configure the name of the EntityManager and to associate it to a DBMS

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>
  <persistence-unit name="titan">
    <jta-data-source>java:/DefaultDS</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    </properties>
  </persistence-unit>
</persistence>
```

Session beans annotations

- @javax.ejb.Remote:
  - The EJB will be used remotely
- @javax.ejb.Stateless
  - It is a stateless session bean
- @javax.ejb.RemoteBinding
  - Indicates the name for the EJB that will be stored within JNDI (if this annotation is not used, a default name is assigned)
- @javax.persistence.PersistenceContext
  - Used to inject in the code the EntityManager to be used
    - @PersistentContext(unitName=“nome”) private EntityManager manager;

Client.java

```
package com.titan.clients;
import com.titan.travelagent.TravelAgentRemote;
import com.titan.domain.Cabin;
import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class Client
{
      public static Context getInitialContext()
       throws javax.naming.NamingException {
       return new javax.naming.InitialContext();
   }
   public static void main(String [] args){
       try {
           Context jndiContext = getInitialContext();
           Object ref = jndiContext.lookup("TravelAgentRemote");
           TravelAgentRemote dao = (TravelAgentRemote)
           PortableRemoteObject.narrow(ref,TravelAgentRemote.class);
```

Client.java

```
Cabin cabin_1 = new Cabin();
cabin_1.setId(1);
cabin_1.setName("Master Suite");
cabin_1.setDeckLevel(1);
cabin_1.setShipId(1);
cabin_1.setBedCount(3);

dao.createCabin(cabin_1);

Cabin cabin_2 = new Cabin();
cabin_2.setId(2);
cabin_2.setName("Junior Suite");
cabin_2.setDeckLevel(1);
cabin_2.setShipId(1);
cabin_2.setBedCount(3);

dao.createCabin(cabin_2);
```

Client.java

```
System.out.println("Looking for cabin number 1...");
cabin_1 = dao.findCabin(1);
System.out.println(cabin_1.getName());
System.out.println(cabin_1.getDeckLevel());
System.out.println(cabin_1.getShipId());
System.out.println(cabin_1.getBedCount());

System.out.println("Looking for cabin number 2...");
cabin_2 = dao.findCabin(2);
System.out.println(cabin_2.getName());
System.out.println(cabin_2.getDeckLevel());
System.out.println(cabin_2.getShipId());
System.out.println(cabin_2.getBedCount());
}

catch (javax.naming.NamingException ne){
    ne.printStackTrace();
}
}
}
```

Jboss

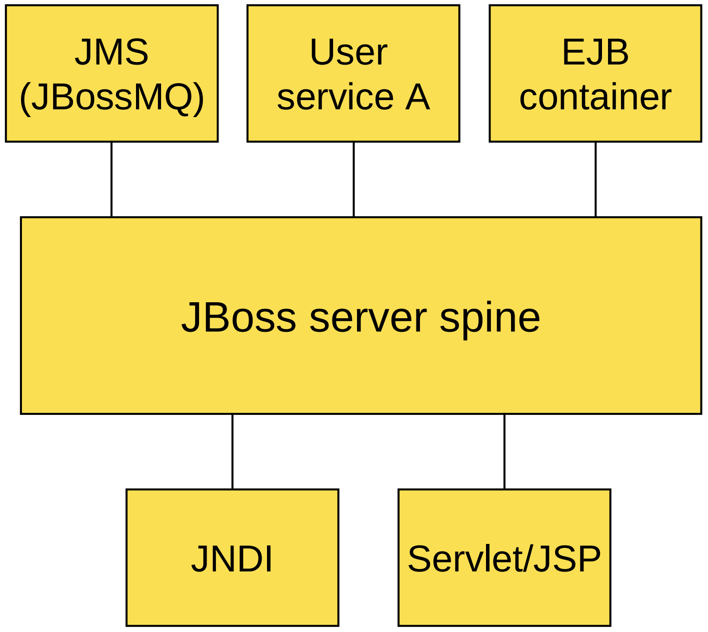
JBoss

- Jboss is an open souce application server
- Derives from an international collaboration
- Currently is the first one in terms of distribution
  - More than 6 millions of download in the last 4 years

- Implements all JEE services
  - EJB
  - Java Persistence
  - Java Messaging Service (JMS)
  - Java Transaction Service/Java Transaction API (JTS/JTA)
  - Servlet e JSP
  - Java Naming and Directory Interface (JNDI)

Microkernel architecture of JBoss

- Based on JMX (Java Management eXtensions)
- Is modular and can be configured by selecting the needed modules (the MBeans)



Main characteristics (1)

- Hot deployment/redeployment
  - Continuously control the content of the deploy directory
    - If a new .jar is copied there, installs and executes the corresponding components
    - If a .jar is updated, terminates the execution of the old components and repeats the installation and execution for the new components
  - Useful when deployment is a critical operation to be executed quickly

Main characteristics (2)

- The communication protocols managers (invokers) are decoupled by the services and the applications
  - Currently JBoss supports invokers for
    - Fast socket
    - IIOP
    - SOAP
    - JMS

Installation

- Requirement
  - Java SE 5 JDK
- Download and install:
  - http://labs.jboss.com/portal/jbossas/download
  - “Run Installer”
  - Select the option EJB3!
- By default the server is installed at the port 8080.

Main directories

- bin
  - Scripts to run and terminate JBoss
    - In Windows run.bat
- client
  - Jar and configuration files to compile client programs
    - jbossall-client.jar
- docs
  - Contains examples of configuration files for JCA (Java Connector Architecture), for instance to support various DBMSs

Main directories (2)

- lib
  - jar files needed for the Jboss microkernel
  - Never add files in this directory!
- server
  - Every subdirectory corresponds to a different configuration of the server
    - In Windows run.bat -c <config-name>

Configurations

- In the “server” by default there are three configurations
  - Minimal:
    - Logging, JNDI, URL, deployment scanner (to detect new deploys)
    - No web containers, no EJB nor JMS
  - Default:
    - Standard services
    - No JAXR, IIOP or clustering services
  - All:
    - everything

Default configuration

- conf
  - jboss-service.xml: Specifies which MBean belong to the configuration
  - jacorb.properties: configuration of JBoss IIOP
  - jbossmq-state.xml: JBossMQ configuration
  - log4j.xml: logging configuration
  - login-config.xml: security configuration
- data
  - Persistent data (HyperSonic)

Default configuration (2)

- deploy
  - Contains hot-deployable (jar, war, ear) applications and services
- lib
  - Jars needed to configure the server
- log
  - Contains log files

build.xml

```
<?xml version="1.0"?>

<!-- JBoss build file   -->


<project name="JBoss" default="ejbjar" basedir=".">

  <property environment="env"/>
  <property name="src.dir" value="${basedir}/src/main"/>
  <property name="src.resources" value="${basedir}/src/resources"/>
  <property name="jboss.home" value="${env.JBOSS_HOME}"/>
  <property name="build.dir" value="${basedir}/build"/>
  <property name="build.classes.dir" value="${build.dir}/classes"/>
```

build.xml

```
<!-- Build classpath -->

 <path id="classpath">
       <fileset dir="${jboss.home}/server/default/lib">
           <include name="*.jar"/>
       </fileset>
       <fileset dir="${jboss.home}/server/default/deploy/ejb3.deployer">
           <include name="*.jar"/>
       </fileset>
       <fileset dir="${jboss.home}/server/default/deploy/jboss-aop-jdk50.deployer">
           <include name="*.jar"/>
       </fileset>
       <fileset dir="${jboss.home}/lib">
           <include name="*.jar"/>
       </fileset>
      <pathelement location="${build.classes.dir}"/>
      <!-- So that we can get jndi.properties for InitialContext and log4j.xml file -->
      <pathelement location="${basedir}/client-config"/>
 </path>

 <property name="build.classpath" refid="classpath"/>
```

build.xml

```
  <!-- Prepares the build directory        -->

  <target name="prepare" >
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${build.classes.dir}"/>
  </target>

  <!-- Compiles the source code          -->

<target name="compile" depends="prepare">
    <javac srcdir="${src.dir}"
           destdir="${build.classes.dir}"
           debug="on"
           deprecation="on"
           optimize="off"
           includes="**">
            <classpath refid="classpath"/>
    </javac>
  </target>
```

build.xml

```
<target name="ejbjar" depends="compile">
   <jar jarfile="build/titan.jar">
     <fileset dir="${build.classes.dir}">
        <include name="com/titan/domain/*.class"/>
        <include name="com/titan/travelagent/*.class"/>
     </fileset>
     <fileset dir="${src.resources}/">
         <include name="META-INF/persistence.xml"/>
     </fileset>
    </jar>
    <copy file="build/titan.jar" todir="${jboss.home}/server/default/deploy"/>
 </target>

 <target name="run.client" depends="ejbjar">
   <java classname="com.titan.clients.Client" fork="yes" dir=".">
     <classpath refid="classpath"/>
   </java>
 </target>
```

build.xml

```
  <!-- Cleans up generated stuff        -->

<target name="clean.db">
    <delete dir="${jboss.home}/server/default/data/hypersonic"/>
  </target>

  <target name="clean">
    <delete dir="${build.dir}"/>
    <delete file="${jboss.home}/server/default/deploy/titan.jar"/>
  </target>

</project>
```