# Java Persistence API deployment plans



## Java Persistence API

The Java Persistence API is a new programming model under EJB3.0 specification (JSR220) for the management of persistence and object/relational mapping with Java EE and Java SE. With JPA, developers can easily develop java applications that perform operations on relational database management systems using java objects and mapping. In that way, java applications developed using JPA are not only portable across different platforms, but also applications can be easily developed using simple yet powerful programming model provided by JPA. This greatly improves application maintainability against ever changing database world. JPA insulates applications from all the complexity and non-portable boilerplate code involved in database connectivity and operations. Apache geronimo uses OpenJPA for providing Java Persistence API to Java EE applications deployed in the server. Below sections illustrate developing applications using JPA and how to write various deployment descriptors and plans for apache geronimo.

The document is organized as follows.



### ShareAccount sample

This example illustrates developing an enterprise application that uses JPA for persistence. The database used is the embedded derby shipped with apache geronimo. Here, we present a persistence deployment descriptor (`persistence.xml`) that contains database connectivity and other information for the application. The `persistence.xml` is placed under `META-INF/` directory of the application archive. The application contains an ejb module and a web module. Ejb module uses a stateless session bean `ShareHolderBean` that uses JPA to perform database operations on the table `SHAREACCOUNT` in the `ShareDB` derby database. The `SHAREACCOUNT` table contains information about each shareholder along with the information regarding number shares he or she possesses currently in the account. The `ShareHolderBean` has methods that retrieve shareholder information, buy/sell shares of a particular shareholder, close the shareholder account etc. The web application has a servlet that looks up the `ShareHolderBean` and trigger the operations on it. The deployment descriptor information for the ejb module is provided using Java EE annotations in the respective bean classes. However, the persistence deployment descriptor information is provided using `META-INF/persistence.xml` file.

**sample.jpa.ShareAccount.java**

```
package sample.jpa;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.PostLoad;
import javax.persistence.PostUpdate;
import javax.persistence.PrePersist;
import javax.persistence.PreUpdate;
import javax.persistence.Version;
import javax.persistence.Table;

@Entity
@Table(name = "SHAREACCOUNT")
@NamedQuery(name="findThem", query="SELECT a FROM ShareAccount a")
public class ShareAccount implements Serializable {


    @Id
    private int shareAccountNo;
    private String ownerName;
    private int numberOfShares;
    @Version
    private int version;

    public ShareAccount(int shareAccountNo,
                        String ownerName,
                        int numberOfShares) {

      this.shareAccountNo = shareAccountNo;
      this.ownerName = ownerName;
      this.numberOfShares = numberOfShares;
```

```
    }

    public String toString() {
        return "Acc.# " + shareAccountNo +
               ", owner" + ownerName +
               ", numberOfShares: " +
                  numberOfShares + " $";
    }

    @PrePersist
    void prepersist() {
        System.out.println("pre persist!!");
    }

    @PreUpdate
    void preupdate() {
        System.out.println("pre update!!");
    }

    @PostUpdate
    void postupdate() {
        System.out.println("post update!!");
    }

    @PostLoad
    void postload() {
        System.out.println("post load!!");
    }

        public int getShareAccountNo() {
                return shareAccountNo;
        }

        public void setShareAccountNo(int shareAccountNo) {
                this.shareAccountNo = shareAccountNo;
        }

        public String getOwnerName() {
                return ownerName;
        }

        public void setOwnerName(String ownerName) {
                this.ownerName = ownerName;
        }

        public int getNumberOfShares() {
                return numberOfShares;
        }

        public void setNumberOfShares(int numberOfShares) {
                this.numberOfShares = numberOfShares;
        }

        public int getVersion() {
                return version;
        }

        public void setVersion(int version) {
                this.version = version;
        }

}
```

Observe various annotations that represent various aspects of the entity in the class.

`@Entity` : Marks the `ShareAccount` class as an entity that can be persisted entirely in the database.
`@Table(name = "SHAREACCOUNT")` : Designates the database table `SHAREACCOUNT` to the entity. The columns in the database table map to the attributes of `ShareAccount` entity.
`@Id` : This annotation designates primary key to the entity. In this case, it is the `shareAccountNo` attribute of the entity class.
`@NamedQuery(name="findAccounts", query="SELECT a FROM ShareAccount a")` : This annotation declares a named query by name `findAccounts`. This query retrieves all the share accounts from the database. Later in the `ShareHolderBean`, the NamedQuery is executed by using the name `findAccounts`.
`@Version` : The version attribute of the `ShareAccount` entity.

The annotations placed around some of the methods of the `ShareAccount` class are `@PrePersist, @PreUpdate, @PostUpdate and @PostLoad`. These are the callback methods called by persistence container when corresponding database operations are performed on the entity.

**sample.jpa.ShareHolder.java**

```
package sample.jpa;
import java.util.List;

public interface ShareHolder {

    public ShareAccount openShareAccount(int shareHolderNo, String ownerName, int numberOfShares);
    public ShareAccount findShareAccount(int shareHolderNo);
    public ShareAccount close(int shareHolderNo);
    public List<ShareAccount> listAccounts();
    public ShareAccount buyShares(int shareHolderNo, int numberOfShares);
    public int sellShares(int shareHolderNo, int numberOfShares);
    public ShareAccount updateShareAccount(ShareAccount sa);
}
```

The `ShareHolder.java` is the remote interface of the stateless session bean.

**sample.jpa.ShareHolderBean.java**

```
package sample.jpa;
import java.util.List;
import javax.ejb.Stateless;
import javax.ejb.Remote;
import javax.persistence.*;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
@Remote(ShareHolder.class)
public class ShareHolderBean implements ShareHolder {

    @PersistenceContext
    private EntityManager manager;

    public ShareAccount openShareAccount(int shareHolderNo,
                                         String ownerName,
                                         int numberOfShares){

            if (findShareAccount(shareHolderNo) == null){
                    ShareAccount sa = new ShareAccount
                                 (shareHolderNo,
                                  ownerName,
                                  numberOfShares);
                manager.persist(sa);
                return sa;
            }
            return null;
    }

    public ShareAccount findShareAccount(int shareHolderNo){
            ShareAccount sa = manager.find(ShareAccount.class,
                                       shareHolderNo);
```

```
            return sa;
    }

    public ShareAccount close(int shareHolderNo){
            ShareAccount sa = manager.find(ShareAccount.class,
                                    shareHolderNo);
            if (sa != null)manager.remove(sa);
            return sa;

    }

    public List<ShareAccount> listAccounts(){
        Query query = manager.createNamedQuery("findThem");
        return query.getResultList();
    }

    public ShareAccount buyShares(int shareHolderNo,
                                int numberOfShares){
            ShareAccount sa = manager.find(ShareAccount.class,
                                    shareHolderNo);
            if(sa != null){
                    int new_balance = sa.getNumberOfShares()
                                + numberOfShares;
                sa.setNumberOfShares(new_balance);
            }
            return sa;
    }

    public int sellShares(int shareHolderNo,
                          int numberOfShares){
        ShareAccount sa = manager.find(ShareAccount.class,
                                    shareHolderNo);
        if(sa != null){
            if (numberOfShares > sa.getNumberOfShares()){
                    return -2;
            }else{
                    int new_balance = sa.getNumberOfShares()
                                - numberOfShares;
                    sa.setNumberOfShares(new_balance);
                    manager.flush();
                    return numberOfShares;
            }
        }else{
                return -1;
        }
    }

    public ShareAccount updateShareAccount(ShareAccount sa){
            ShareAccount sa1 =
                   findShareAccount(sa.getShareAccountNo());
            if (sa1 != null){
                    manager.merge(sa1);
                    return sa1;
            }
            else return null;

    }


}
```

The `ShareHolderBean.java` is the stateless session bean that uses JPA to perform database operations using `ShareAccount` entity.

**Persistence.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

 <persistence-unit name="PersistencePU">
 <description>ShareHolder</description>
 <provider>
  org.apache.openjpa.persistence.PersistenceProviderImpl
 </provider>
 <class>sample.jpa.ShareAccount</class>
 <properties>
  <property name="openjpa.ConnectionURL"
           value="jdbc:derby:ShareDB" />
  <property name="openjpa.ConnectionDriverName"
           value="org.apache.derby.jdbc.EmbeddedDriver" />
  <property name="ConnectionUserName" value="app" />
  <property name="openjpa.jdbc.SynchronizeMappings" value="false" />
 </properties>
 </persistence-unit>
 <!--
 <jta-data-source>PhoneBookPool</jta-data-source>
 <non-jta-data-source>PhoneBookPool</non-jta-data-source>
 -->
</persistence>
```

The `Persistence.xml` is the persistence deployment descriptor for the ejb module. It provides database connection information and declares entity classes among other things.



The default namespace of the above XML document is http://java.sun.com/xml/ns/persistence. The XML elements that do not have a namespace prefix belong to the default namespace.

**sample.jpa.Test.java**

```
package sample.jpa;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Test extends
  javax.servlet.http.HttpServlet
  implements javax.servlet.Servlet {

  static final long serialVersionUID = 1L;

   public Test() {
        super();
   }

   protected void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException,
                  IOException {
```

```
PrintWriter out = response.getWriter();

try {
 java.util.Properties env =
   new java.util.Properties();

 env.put(Context.INITIAL_CONTEXT_FACTORY
   ,"org.apache.openejb.client.RemoteInitialContextFactory");
 env.put(Context.PROVIDER_URL,
         "ejbd://127.0.0.1:4201");

 Context ctx = new InitialContext(env);
 out.println("Looking up ShareHolderBean");

 ShareHolder shareHolder = (ShareHolder)
        ctx.lookup("ShareHolderBeanRemote");

 out.println("Creating ShareAccount 1,
              Phani, 10");

 ShareAccount sa =
     shareHolder.openShareAccount(1,"phani", 10);
 if(sa == null){
    out.println("account could not be created.");
    out.println("May be, account already
                        exists. Check the database.");

 }
 else {
    out.println("Account is successfully created");
 }
 out.println("Looking up the ShareAccountNumber 1");
 sa = shareHolder.findShareAccount(1);
 out.println("Printing the details of ShareAccountNumber 1");
 printShareAccount(sa,out);
 out.println("");

 out.println("buying shares 100");
 sa = shareHolder.buyShares(1, 100);

 out.println("Printing the details of ShareAccountNumber 1");
 printShareAccount(sa,out);
 out.println("");

 out.println("selling 50 shares of ShareAccountNumber 1");
 int numberShares = shareHolder.sellShares(1, 50);
 if(numberShares == 50){
   out.println("Printing the details of ShareAccountNumber 1");
   sa = shareHolder.findShareAccount(1);
   printShareAccount(sa,out);
 }
 else if(numberShares == -1){
   out.println("ShareAccountNo can not be found");
 }else {
   out.println("The number shares available are less than 50");
 }
 out.println("");

 List<ShareAccount> saList = shareHolder.listAccounts();
 out.println("Printing all the available accounts");
 for (int i = 0; i < saList.size(); i++){
            out.println("*******");
            printShareAccount(saList.get(i),out);
            out.println("*******");
            out.println("");
    }
 out.println("");

 out.println("Setting the ShareAccount 1 with 500 shares
              and updating the database");
 sa = new ShareAccount(1,"phani",0);
```

```
     sa.setNumberOfShares(500);
     shareHolder.updateShareAccount(sa);
     out.println("Printing the details of ShareAccountNumber 1");
     printShareAccount(sa,out);
     out.println("");

     out.println("Closing ShareAccountNumber 1");
     sa = shareHolder.close(1);
     if(sa == null){
        out.println("Account is not found="+1);
     }else{
        out.println("Printing the details of ShareAccountNumber 1");
        printShareAccount(sa,out);
     }

   }
   catch(Exception e){
     e.printStackTrace();
     throw new ServletException(e);
   }
}

 protected void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                  throws ServletException, IOException {

 }
 private void printShareAccount(ShareAccount sa, PrintWriter out){
    if(sa != null){
    out.println("Account Number = "+sa.getShareAccountNo());
    out.println("Owner Name = "+sa.getOwnerName());
    out.println("number of shares "+sa.getNumberOfShares());
    out.println("version="+sa.getVersion());
    }else{
     out.println("ShareAccountNo can not be found");
    }
 }
}
```

**web.xml**

```
<!--
The deployment descriptor of the web client.
-->
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         id="WebApp_ID" version="2.5">

  <display-name>ShareHolderWEB</display-name>
   <servlet>
    <description></description>
    <display-name>Test</display-name>
    <servlet-name>Test</servlet-name>
    <servlet-class>sample.jpa.Test</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/Test</url-pattern>
  </servlet-mapping>
</web-app>
```

**geronimo-web.xml**

```
<!--
The geronimo deployment plan of the web client.
-->

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1"
         xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.2"
         xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
         xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
  <sys:environment>
    <sys:moduleId>
      <sys:groupId>JPA</sys:groupId>
      <sys:artifactId>ShareHolderWEB</sys:artifactId>
      <sys:version>1.0</sys:version>
      <sys:type>car</sys:type>
    </sys:moduleId>
  </sys:environment>
  <context-root>/ShareHolderWEB</context-root>
</web-app>
```

The Servlet client `Test.java`, looks up the `ShareHolderBean` and executes various methods on the `ShareAccount` entity. When the url http://localhost :8080/ShareHolderWEB/Test is hit on a browser window, the following output is displayed.

http://localhost:8080/ShareHolderWEB/Test

```
_____________________________________________
Looking up ShareHolderBean
Creating ShareAccount 1, Phani, 10
Account is successfully created
Looking up the ShareAccountNumber 1
Printing the details of ShareAccountNumber 1
Account Number = 1
Owner Name = phani
number of shares 10
version=1
_____________________________________________

_____________________________________________

buying shares 100
Printing the details of ShareAccountNumber 1
Account Number = 1
Owner Name = phani
number of shares 110
version=2
_____________________________________________

_____________________________________________
selling 50 shares of ShareAccountNumber 1
Printing the details of ShareAccountNumber 1
Account Number = 1
Owner Name = phani
number of shares 60
version=3
_____________________________________________

_____________________________________________
Printing all the available accounts
*******
Account Number = 1
Owner Name = phani
number of shares 60
version=3
*******
_____________________________________________

_____________________________________________
Setting the ShareAccount 1 with 500 shares and updating the database
Printing the details of ShareAccountNumber 1
Account Number = 1
Owner Name = phani
number of shares 500
version=0
_____________________________________________

_____________________________________________
Closing ShareAccountNumber 1
Printing the details of ShareAccountNumber 1
Account Number = 1
Owner Name = null
number of shares 0
version=0
_____________________________________________
```

## Inheritance relationship in entities

Entities can exhibit inheritance relationship among themselves. The entities involved in inheritance relationship can be persisted/updated/retrieved independently. There are several ways of realizing inheritance relationship among entities. There are as follows.

- Single database table per class hierarchy

- Separate database table per subclass
- Single database table per concrete entity class

## Single database table per class hierarchy

In this technique, a single database table is designated for entire entity class hierarchy exhibiting inheritance relationship. For example, if entity C extends entity B which extends entity A, the fields of all the entities are stored in a single table. The entities A, B and C can be persisted/updated/deleted/retrieved independently. This technique requires all the columns except for primary columns and columns of parent most entity A, should be nullable. This is because, suppose take the case when an entity B is being inserted. The entity B will have the values for the fields of A and itself but no values for fields of C. Since entire hierarchy uses a single table, the columns pertaining to entity C must be nullable. The same case arises when inserting entity A.

The below example illustrates the use this technique. The enterprise application has 3 entities and uses a single database table to store these entities. A stateless session bean uses JPA to perform DML operations on these entities. The web client looks up the SLSB trigger the operations.

**Account**

```
package com.sample.jpa;

import java.io.Serializable;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name="Account")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="ACCOUNTTYPE",
 discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("ACCOUNT")
public abstract class Account
       implements Serializable{

 @Id
 private int accountNo;
 private String name;
 private String address;
 private int branchCode;

 public Account(){
  this.accountNo = 0;
  this.name = "DUMMY";
  this.address = "DUMMY";
  this.branchCode = 0;
 }

 public int getAccountNo() {
  return accountNo;
 }
 public void setAccountNo(int accountNo) {
  this.accountNo = accountNo;
 }
 public String getName() {
  return name;
 }
 public void setName(String name) {
  this.name = name;
 }
 public String getAddress() {
  return address;
 }
 public void setAddress(String address) {
  this.address = address;
 }
 public int getBranchCode() {
  return branchCode;
 }
 public void setBranchCode(int branchCode) {
  this.branchCode = branchCode;
 }

}
```

⚠ `@DiscriminatorColumn` annotation designates table column to be used to discriminate rows that correspond to different entities in the class hierarchy. `@DiscriminatorValue` annotation provides value that corresponds to the `Account` entity for the `DiscriminatorColumn`. The value is `Account`. Note that the Account entity class is declared as an abstract class. This is to prevent applications from instantiating Account objects. The intention is, to allow instantiating only sub entities `SavingsAccount` and `CurrentAccount` as given below.

**SavingsAccount**

```
package com.sample.jpa;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name="Account")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorValue("SAVINGSACCOUNT")
public class SavingsAccount extends Account {
 private double interestRate;
 private double minBalance;
 private double balance;
 private String savingsAccountRules;

 public SavingsAccount(){
  super();
  this.interestRate = 0;
  this.minBalance = 0;
  this.balance = 0;
  this.savingsAccountRules = null;
 }

 public double getInterestRate() {
  return interestRate;
 }
 public void setInterestRate(double interestRate) {
  this.interestRate = interestRate;
 }
 public double getMinBalance() {
  return minBalance;
 }
 public void setMinBalance(double minBalance) {
  this.minBalance = minBalance;
 }
 public double getBalance() {
  return balance;
 }
 public void setBalance(double balance) {
  this.balance = balance;
 }
 public String getSavingAccountRules() {
  return savingsAccountRules;
 }
 public void setSavingAccountRules(String savingAccountRules) {
  this.savingsAccountRules = savingAccountRules;
 }
}
```

⚠ `@DiscriminatorValue` annotation provides value that corresponds to the `SavingsAccount` entity for the `DiscriminatorColumn`. The value is `SAVINGSACCOUNT`

**CurrentAccount**

```
package com.sample.jpa;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name="Account")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorValue("CURRENTACCOUNT")
public class CurrentAccount extends Account {
 private double interestRate;
 private double minBalance;
 private double balance;
 private String currentAccountRules;

 public CurrentAccount(){
  super();
  this.interestRate = 0;
  this.minBalance = 0;
  this.balance = 0;
  this.currentAccountRules = null;
 }
 public double getInterestRate() {
  return interestRate;
 }
 public void setInterestRate(double interestRate) {
  this.interestRate = interestRate;
 }
 public double getMinBalance() {
  return minBalance;
 }
 public void setMinBalance(double minBalance) {
  this.minBalance = minBalance;
 }
 public double getBalance() {
  return balance;
 }
 public void setBalance(double balance) {
  this.balance = balance;
 }
 public String getCurrentAccountRules() {
  return currentAccountRules;
 }
 public void setCurrentAccountRules
          (String currentAccountRules) {
  this.currentAccountRules = currentAccountRules;
 }
}
```



`@DiscriminatorValue` annotation provides value that corresponds to the `SavingsAccount` entity for the `DiscriminatorColumn`. The value is `CURRENTACCOUNT`.
The `CurrentAccount` entity has `currentAccountRules` field and `SavingsAccount` has `SavingsAccountRules` field. Both inherit fields from the parent entity `Account`

**AccessAccountStateless**

```
package com.sample.jpa;
import java.util.List;
public interface AccessAccountStateless {
 public SavingsAccount createSavingsAccount
        (int accNo, String name,
         String address, int branchCode,
         double interestRate,
         double minBalance,
         double balance,
         String savingsAccountRules);

 public CurrentAccount createCurrentAccount
       (int accNo, String name,
        String address, int branchCode,
        double interestRate,
        double minBalance,
        double balance,
        String currentAccountRules);
 public List listAccounts();
 public SavingsAccount
        updateSavingsAccountBalance
       (int accNo,
        double newBalance);
 public CurrentAccount
        updateCurrentAccountBalance
       (int accNo,
        double newBalance);
 public void deleteSavingsAccount(int accNo);
 public void deleteCurrentAccount(int accNo);
}
```

**AccessAccountStatelessBean**

```
package com.sample.jpa;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;


@Stateless
@Remote
public class AccessAccountStatelessBean
             implements AccessAccountStateless{
 @PersistenceContext(unitName="SingleTableInheritance")
  EntityManager em;

 public SavingsAccount createSavingsAccount
       (int accNo, String name,
        String address, int branchCode,
        double interestRate,
        double minBalance,
        double balance,
        String savingsAccountRules){

  SavingsAccount sa = new SavingsAccount();
  sa.setAccountNo(accNo);
  sa.setAddress(address);
  sa.setBalance(balance);
  sa.setBranchCode(branchCode);
```

```
 sa.setMinBalance(minBalance);
 sa.setName(name);
 sa.setSavingAccountRules(savingsAccountRules);
 sa.setInterestRate(interestRate);
 em.persist(sa);
 return sa;
}

public CurrentAccount createCurrentAccount
      (int accNo, String name,
       String address, int branchCode,
       double interestRate,
       double minBalance,
       double balance,
       String currentAccountRules){

 CurrentAccount ca = new CurrentAccount();
 ca.setAccountNo(accNo);
 ca.setAddress(address);
 ca.setBalance(balance);
 ca.setCurrentAccountRules(currentAccountRules);
 ca.setInterestRate(interestRate);
 ca.setMinBalance(minBalance);
 ca.setName(name);
 ca.setBranchCode(branchCode);
 em.persist(ca);
 return ca;
}

public List listAccounts(){
 List allList = new ArrayList();
 Query q1 = em.createQuery("SELECT sa FROM SavingsAccount sa");
 List currList1 = q1.getResultList();
 allList.addAll(currList1);
 Query q2 = em.createQuery("SELECT ca FROM CurrentAccount ca");
 List currList2 = q2.getResultList();
 allList.addAll(currList2);
 return allList;
}

public SavingsAccount updateSavingsAccountBalance
        (int accNo, double newBalance){
 SavingsAccount sa =
  em.find(SavingsAccount.class, accNo);
 if (sa == null)
    throw new IllegalStateException
         ("Account number not found");
 Query q = em.createQuery
      ("UPDATE SavingsAccount sa SET
        sa.balance = "+newBalance);
 q.executeUpdate();
 return sa;

}
public CurrentAccount updateCurrentAccountBalance
                     (int accNo, double newBalance){
 CurrentAccount ca = em.find(CurrentAccount.class, accNo);
 if (ca == null) throw new IllegalStateException
                ("Account number not found");
 Query q = em.createQuery("UPDATE CurrentAccount
                       ca SET ca.balance = "+newBalance);
 q.executeUpdate();
 return ca;
}

public void deleteSavingsAccount(int accNo){
 SavingsAccount sa =
  em.find(SavingsAccount.class, accNo);
 if(sa == null)
    throw new
     IllegalStateException("Account number not found");
```

```
  em.remove(sa);
 }
 public void deleteCurrentAccount(int accNo){
  CurrentAccount ca =
   em.find(CurrentAccount.class, accNo);
  if(ca == null)
   throw new IllegalStateException
    ("Account number not found");
  em.remove(ca);
 }
}
```

**com.sample.jpa.Test**

```
package com.sample.jpa;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Test extends
 javax.servlet.http.HttpServlet
 implements javax.servlet.Servlet {

 static final long serialVersionUID = 1L;
 @EJB AccessAccountStateless aas;

 public Test() {
  super();
 }
 protected void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException,
                      IOException {
  PrintWriter out = response.getWriter();
  int accNo = Integer.parseInt(request.getParameter("accNo"));
  String name = request.getParameter("name");

  aas.createCurrentAccount(accNo, name,
      name+"' address", 10, 20, 200, 5000,
      "Current Account!!");

  aas.createSavingsAccount(accNo+1, name,
      name+"' address",        12, 234, 4534, 13323,
      "Savings Account!!");

  out.println("Created a Savings account
               and a Current Account");
  List accList = aas.listAccounts();
  out.println("<table>");
  out.println("<tr>");
  out.println("<th>AccountNumber</th>");
  out.println("<th>Name</th>");
  out.println("<th>Address</th>");
  out.println("<th>Branch Code</th>");
  out.println("<th>Interest Rate</th>");
  out.println("<th>Minimum Balance</th>");
  out.println("<th>Balance</th>");
  out.println("<th>Current Account Rules</th>");
  out.println("<th>Savings Account Rules</th>");
  out.println("</tr>");

  for(int i = 0; i < accList.size(); i++){
   Account a = (Account)accList.get(i);
   if (a instanceof SavingsAccount){
```

```
    SavingsAccount sa = (SavingsAccount)a;
    out.println("<tr>");
    out.println("<td>"+sa.getAccountNo()+"</td>");
    out.println("<td>"+sa.getName()+"</td>");
    out.println("<td>"+sa.getAddress()+"</td>");
    out.println("<td>"+sa.getBranchCode()+"</td>");
    out.println("<td>"+sa.getInterestRate()+"</td>");
    out.println("<td>"+sa.getMinBalance()+"</td>");
    out.println("<td>"+sa.getBalance()+"</td>");
    out.println("<td></td>");
    out.println("<td>"+sa.getSavingAccountRules()+"</td>");
    out.println("</tr>");
   }else if (a instanceof CurrentAccount){
    CurrentAccount ca = (CurrentAccount)a;
    out.println("<tr>");
    out.println("<td>"+ca.getAccountNo()+"</td>");
    out.println("<td>"+ca.getName()+"</td>");
    out.println("<td>"+ca.getAddress()+"</td>");
    out.println("<td>"+ca.getBranchCode()+"</td>");
    out.println("<td>"+ca.getInterestRate()+"</td>");
    out.println("<td>"+ca.getMinBalance()+"</td>");
    out.println("<td>"+ca.getBalance()+"</td>");
    out.println("<td>"+ca.getCurrentAccountRules()+"</td>");
    out.println("<td></td>");
    out.println("</tr>");
   }
  }
  out.println("</table>");
  out.println("Deleting both the accounts "+accNo+" and "+(accNo+1));
  //aas.deleteCurrentAccount(accNo);
  //aas.deleteSavingsAccount(accNo+1);
  //out.println("Accounts successfully deleted..!!");
 }

 protected void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException,
                              IOException {
 }
}
```

The above servlet client only inserts entities in the table `ACCOUNT`. The code that deletes entities is commented out. We can write variety of clients that lookup the EJB and perform operations on the entities.

**Persistence.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
 xmlns="http://java.sun.com/xml/ns/persistence"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 version="1.0"
 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
 http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

 <persistence-unit name="SingleTableInheritance">
 <description>
   Single Table Inheritance example
 </description>
 <provider>
  org.apache.openjpa.persistence.PersistenceProviderImpl
 </provider>
 <class>com.sample.jpa.Account</class>
 <class>com.sample.jpa.SavingsAccount</class>
 <class>com.sample.jpa.CurrentAccount</class>
 <properties>
  <property
   name="openjpa.ConnectionURL"
   value="jdbc:derby:AccountDB" />
  <property
   name="openjpa.ConnectionDriverName"
   value="org.apache.derby.jdbc.EmbeddedDriver" />
   <property name="ConnectionUserName" value="app" />
   <property name="openjpa.jdbc.SynchronizeMappings"
    value="false" />
  </properties>
 </persistence-unit>
 <!--
 <jta-data-source>PhoneBookPool</jta-data-source>
 <non-jta-data-source>PhoneBookPool</non-jta-data-source>
 -->
</persistence>
```

The persistence schema uses `AccountDB` derby database. It declares the entities in the persistence unit as well. The following procedure explains how to deploy/run the sample.

- Create an EAR application that contains an EJB application packaging all entity classes, ejb classes and META-INF/persistence.xml
- Create META-INF/ejb-jar.xml. Since we have used annotations, we do not have to provide any declarations in it.
- Create a WEB application in the EAR and add the above servlet.
- Create a derby database by name `AccountDB` using admin console.
- Create a table by name `ACCOUNT` as following.

  **ACCOUNT table**

  ```
  create table ACCOUNT(ACCOUNTNO integer,
              ACCOUNTTYPE varchar(50),
              NAME varchar(50),
              ADDRESS varchar(225),
              BRANCHCODE integer,
              INTERESTRATE decimal(15,2),
              MINBALANCE decimal(15,2),
              BALANCE decimal(15,2),
              CURRENTACCOUNTRULES varchar(225),
              SAVINGSACCOUNTRULES varchar(225))
  ```

- Run the servlet with the following input
  http://localhost:8080/<web-context>/Test?accNo=1&name=Joe
  http://localhost:8080/<web-context>/Test?accNo=3&name=John

- The following data will be inserted into ACCOUNT table.

**ACCOUNT table**

```
ACCOUNTNO ACCOUNTTYPE NAME ADDRESS BRANCHCODE INTERESTRATE MINBALANCE BALANCE CURRENTACCOUNTRULES
SAVINGSACCOUNTRULES
1 CURRENTACCOUNT Joe Joe's address 10 20.00 200.00 5000.00 Current Account!!
2 SAVINGSACCOUNT Joe Joe's address 12 234.00 4534.00 13323.00  Savings Account!!
3 CURRENTACCOUNT John John's address 10 20.00 200.00 5000.00 Current Account!!
4 SAVINGSACCOUNT John John's address 12 234.00 4534.00 13323.00  Savings Account!!
```

Note that JPA has inserted the value CURRENTACCOUNT in the column ACCOUNTTYPE for CurrentAccount entities and inserted the value SAVINGSACCOUNT for SavingsAccount entites.