Installing a Derby DB

Derby quick installation

- download Derby from http://db.apache.org/derby/ and unzip it
- in a shell, set DERBY_HOME and execute setEmbeddedCP

```
DERBY_HOME=/yourpathto/db-derby-10.11.1.1-bin
DERBY_HOME/bin/setEmbeddedCP
```

- start Derby

```
java –jar DERBY_HOME/lib/derbyrun.jar server start &
```

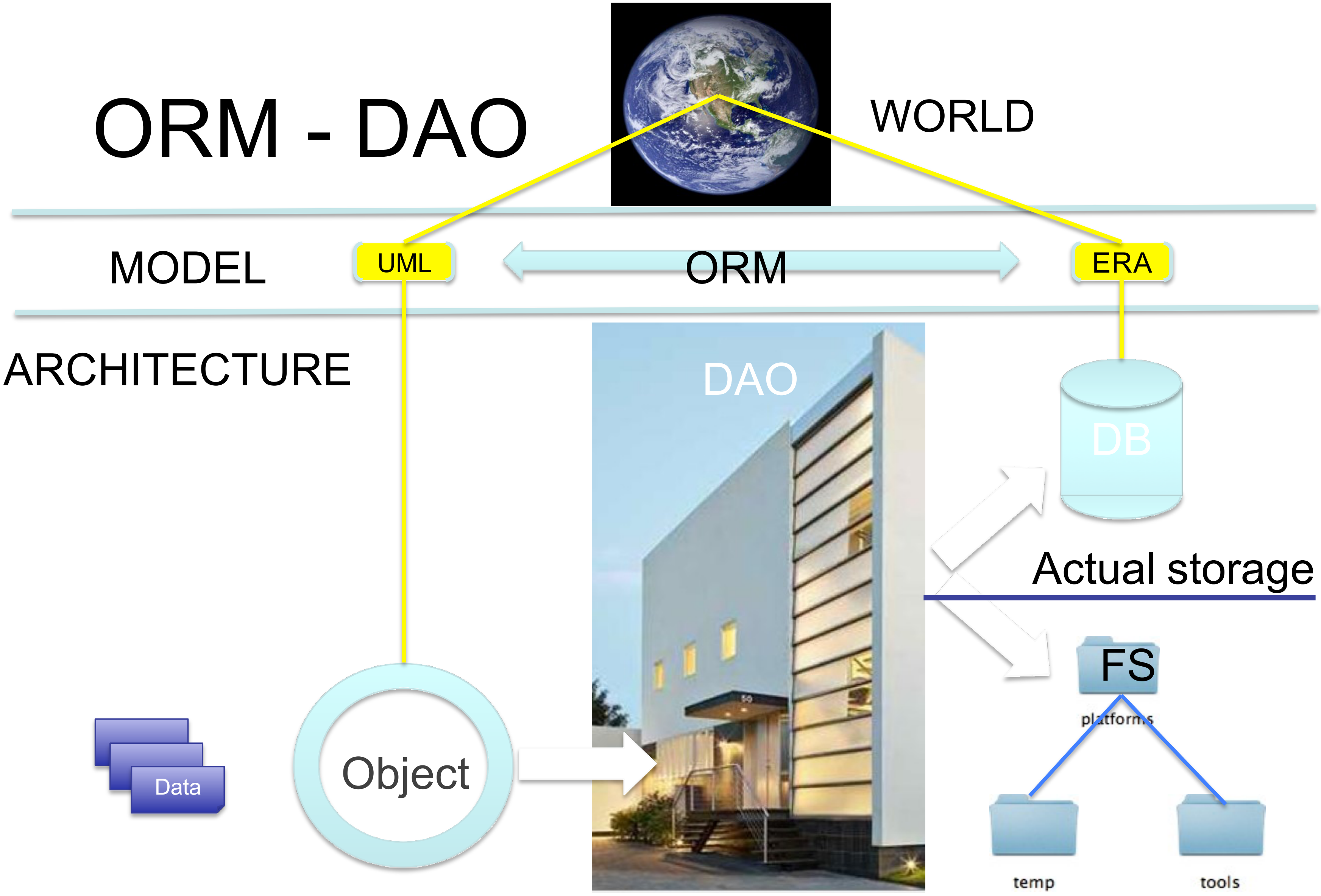
Persistency Demo

download Hibernate from

http://sourceforge.net/projects/hibernate/

unzip it

ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DAO

DB

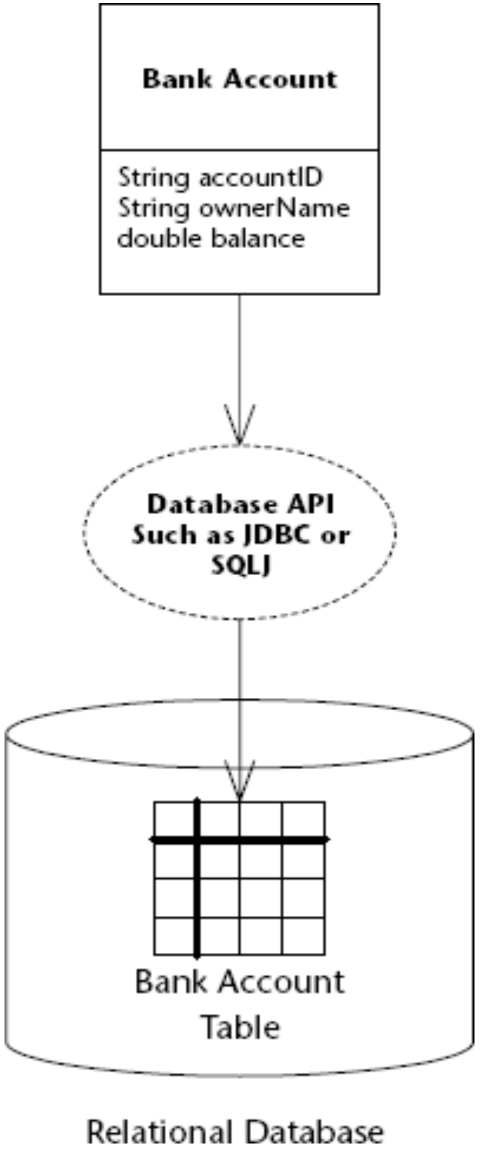
Actual storage

FS

Object

ORM

Object-Relational Mapping is NOT serialization!



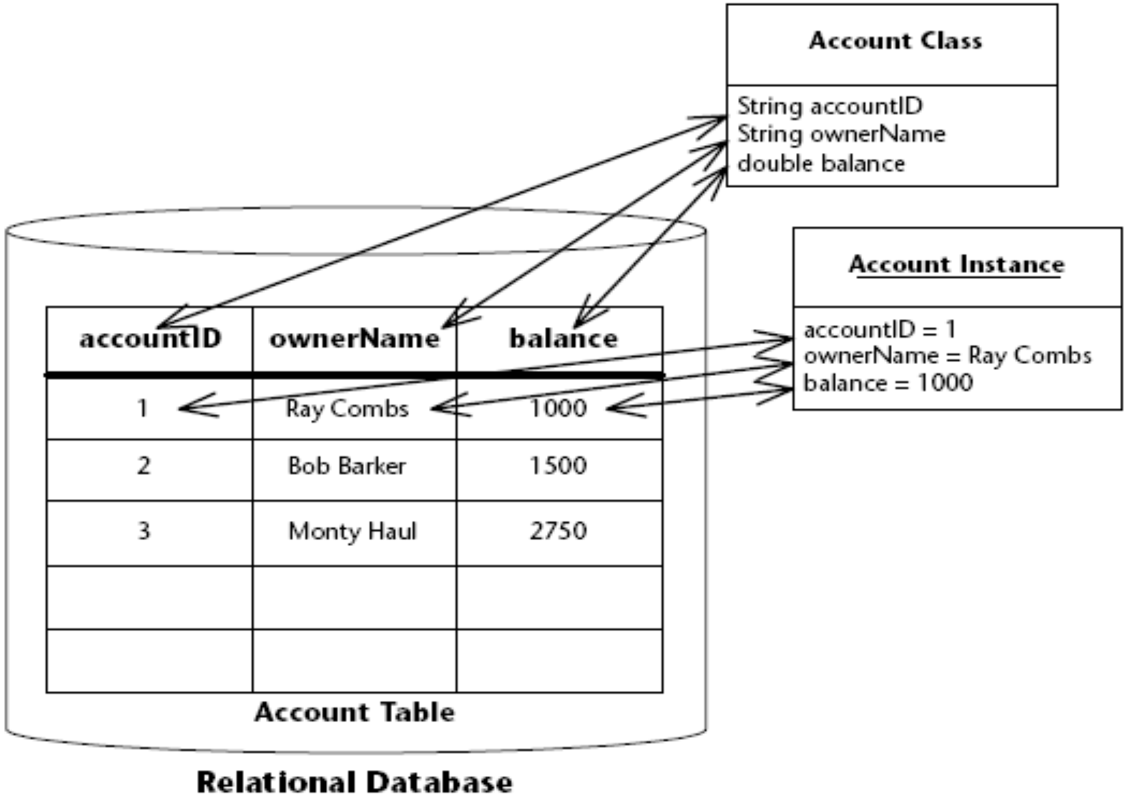
Bank Account

String accountID
String ownerName
double balance

Database API Such as JDBC or SQLJ

Bank Account Table

Relational Database



Account Class

String accountID
String ownerName
double balance

Account Instance

accountID = 1
ownerName = Ray Combs
balance = 1000

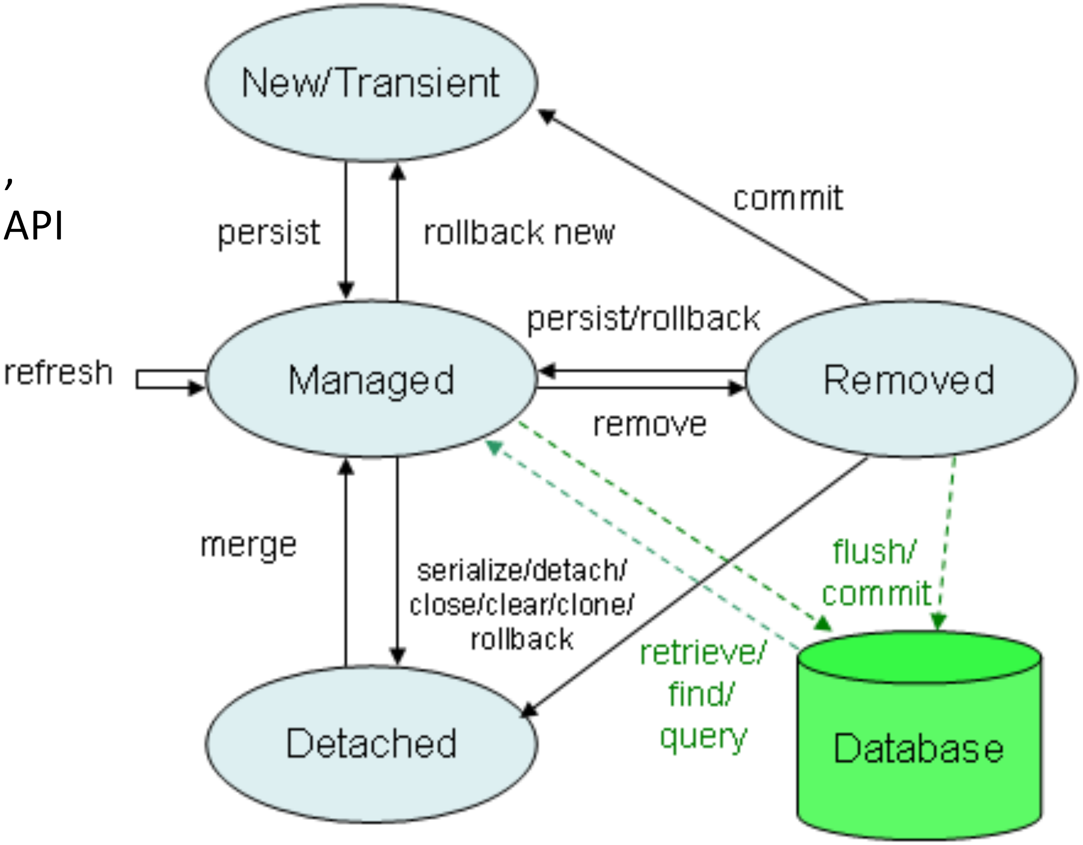
accountID	ownerName	balance
1	Ray Combs	1000
2	Bob Barker	1500
3	Monty Haul	2750

Account Table

Relational Database

Object states in JPA

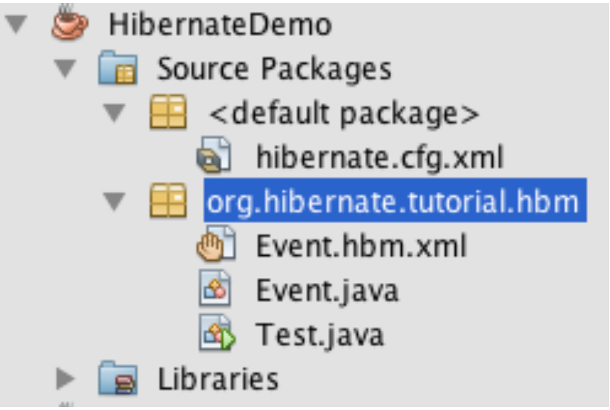
JPA: Java Persistence API,
now Jakarta Persistence API



Demo

Event.java

```
package org.hibernate.tutorial.hbm;
import java.util.Date;
public class Event {
        private Long id;
        private String title;
        private Date date;
        public Event() {
                // this form used by Hibernate
        }
        public Event(String title, Date date) {
                // for application use, to create new events
                this.title = title;
                this.date = date;
        }
        public Long getId() { return id; }
        private void setId(Long id) { this.id = id; }
        public Date getDate() { return date; }
        public void setDate(Date date) { this.date = date; }
        public String getTitle() { return title; }
        public void setTitle(String title) { this.title = title; }
}
```



Event.hbm.xml

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="org.hibernate.tutorial.hbm">
  <class name="Event" table="EVENTS">
    <id name="id" column="EVENT_ID">
      <generator class="increment"/>
    </id>
    <property name="date" type="timestamp" column="EVENT_DATE"/>
    <property name="title"/>
  </class>
</hibernate-mapping>
```

Demo

Test.java

```
package org.hibernate.tutorial.hbm;
import ...;
public class Test {
        private SessionFactory sessionFactory;
        public static void main (String a[]) throws Exception{
            Test x= new Test();
            x.setUp();
            if (a[0].equalsIgnoreCase("R"))  x.read();
            if (a[0].equalsIgnoreCase("W")) x.write();
            x.tearDown();

        }
        protected void setUp() throws Exception {
                // A SessionFactory is set up once for an application
                sessionFactory = new Configuration()
                .configure() // configures settings from hibernate.cfg.xml
                .buildSessionFactory();
        }
        protected void tearDown() throws Exception {
                if ( sessionFactory != null ) { sessionFactory.close();
               }
```

Demo

```
    public void write() {
        // create a couple of events...
        Session session = sessionFactory.openSession();
        session.beginTransaction();
        session.save( new Event( "Our very first event!", new Date() ) );
        session.save( new Event( "A follow up event", new Date() ) );
        session.getTransaction().commit();
        session.close();
     }
     public void read() {
        // now lets pull events from the database and list them
        Session session = sessionFactory.openSession();
        session.beginTransaction();
                List result = session.createQuery( "from Event" ).list();
                for ( Event event : (List<Event>) result ) {
                        System.out.println( "Event (" + event.getDate() +
                        ") : " + event.getTitle() );
                }
        session.getTransaction().commit();
        session.close();
    }
}
```

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</
property>
        <property name="connection.url">jdbc:derby://localhost:1527/
databaseName;create=true</property>
        <property name="connection.username">admin</property>
        <property name="connection.password">admin</property>
        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.DerbyDialect</property>
```

hibernate.cfg.xml

```
    <!-- Disable the second-level cache  -->
    <property name="cache.provider_class">org.hibernate.cache.internal.NoCacheProvider</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>

    <!– To drop and re-create the database schema on startup: use create instead of validate -->
    <property name="hbm2ddl.auto">validate</property>

    <mapping resource="org/hibernate/tutorial/hbm/Event.hbm.xml"/>

  </session-factory>

</hibernate-configuration
```

Note: bindings for other DBs:

http://karussell.wordpress.com/2009/06/09/hibernate-cfg-xml-settings-for-derby-oracle-and-h2/

write

- run:
- set 30, 2014 12:12:44 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
- INFO: HCANN000001: Hibernate Commons Annotations {4.0.5.Final}
- …
- INFO: HHH000037: Columns: [title, event_date, event_id]
- Hibernate: select event0_.EVENT_ID as EVENT_ID1_0_, event0_.EVENT_DATE as EVENT_DA2_0_, event0_.title as title3_0_ from EVENTS event0_
- **INFO: HHH000037: Columns: [title, event_date, event_id]**
- **Hibernate: select max(EVENT_ID) from EVENTS**
- **Hibernate: insert into EVENTS (EVENT_DATE, title, EVENT_ID) values (?, ?, ?)**
- **Hibernate: insert into EVENTS (EVENT_DATE, title, EVENT_ID) values (?, ?, ?)**
- set 30, 2014 12:12:45 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
- INFO: HHH000030: Cleaning up connection pool [jdbc:derby://localhost:1527/databaseName;create=true]
- BUILD SUCCESSFUL (total time: 1 second)

read

```
•   run:
•   set 30, 2014 12:12:44 PM
    org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
•   INFO: HCANN000001: Hibernate Commons Annotations {4.0.5.Final}
•   …
•   INFO: HHH000037: Columns: [title, event_date, event_id]
•   Hibernate: select event0_.EVENT_ID as EVENT_ID1_0_, event0_.EVENT_DATE as
    EVENT_DA2_0_, event0_.title as title3_0_ from EVENTS event0_
•   Event (2014-09-30 12:12:35.622) : Our very first event!
•   Event (2014-09-30 12:12:35.682) : A follow up event
•   set 30, 2014 12:12:45 PM
    org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
•   INFO: HHH000030: Cleaning up connection pool [jdbc:derby://localhost:1527/
    databaseName;create=true]
•   BUILD SUCCESSFUL (total time: 1 second)
```

persistence.xml vs hibernate.cfg.xml

- If you are using Hibernate's proprietary API, you'll need the hibernate.cfg.xml. If you are using JPA i.e. Hibernate EntityManager, you'll need the persistence.xml.
- So you generally don't need both as you use either Hibernate proprietary API or JPA.
- However, if you were using Hibernate Proprietary API and already have a hibernate.cfg.xml (and hbm.xml XML mapping files) but want to start using JPA, you can reuse the existing configuration files by referencing the hibernate.cfg.xml in the persistence.xml in the hibernate.ejb.cfgfile property - and thus have both files

first and second level caching

first level caching: it's associated with the current session and is used to reduce the number of SQL statements within the same transaction. It's on by default.

second level caching: it is used **across sessions**. Hibernate provides a flexible concept to exchange cache providers for the second-level cache. **It's not on by default** but it needs some configuration in your persistence.xml to tell Hibernate to turn on the cache.

The shared-cache-mode element has four possible values:
» **ALL**: Causes all entities and entity related state and data to be cached
» **NONE**: Causes caching to be disabled for the persistence unit and all caching is turned off for all entities
» **ENABLE_SELECTIVE**: Enables the cache and causes entities for which @Cacheable(true) is specified to be cached
» **DISABLE_SELECTIVE**: Enables the cache and causes all entities to be cached except those for which @Cacheable(false) is specified

http://www.mastertheboss.com/jboss-frameworks/hibernate-jpa/hibernate-cache/using-hibernate-second-level-cache-with-jboss-as-5-6-7?showall=

Transactions

```
public void write() {
               // create a couple of events...
               Session session = sessionFactory.openSession();
               session.beginTransaction();
               session.save( new Event( "Our very first event!", new Date() ) );
               session.save( new Event( "A follow up event", new Date() ) );
               session.getTransaction().commit();
               session.close();
        }
```

Introduction to Entities

Entities

- Entities have a client-visible, persistent *identity* (the primary key) that is distinct from their object reference.
- Entities have persistent, client-visible *state*.
- Entities are *not remotely accessible*.
- An entity's *lifetime* may be completely independent of an application's lifetime.
- Entities can be used in both Java EE and J2SE environments

Entities - example

This demo entity represents a Bank Account. The entity is not a remote object and can only be accessed locally by clients. However, it is made serializable so that instances can be passed by value to remote clients for local inspection. Access to persistent state is by direct field access.

```
package examples.entity.intro;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class Account implements Serializable {
    // The account number is the primary key
    @Id
    public int accountNumber;
    public int balance;
    private String ownerName;
    String getOwnerName() {return ownerName;}
    void setOwnerName(String s) {ownerName=s;}

    /** Entities must have a public no-arg constructor */
    public Account() {
        // our own simple primary key generation
        accountNumber = (int) System.nanoTime();
    }
```

Entities - example

```
    public void deposit(int amount) {
      balance += amount;
    }
    public int withdraw(int amount) {
        if (amount > balance) {
          return 0;
        } else {
          balance -= amount;
          return amount;
        }
    }
}
```

The entity can expose **business methods**, such as a method to decrease a bank account balance, to manipulate or access that data. Like a session bean class, an entity class can also declare some standard callback methods or a callback listener class. The persistence provider will call these methods appropriately to manage the entity.

Access to the entity's persistent state is by direct field access. An entity's state can also be accessed using JavaBean-style set and get methods.

The persistence provider can determine which access style is used by looking at how annotations are applied. In the discussed example the @Id annotation is applied to a field (as opposed to annotation applied to a method, so we have field access.

Access to the Entity

```
package examples.entity.intro;
import java.util.List;
import javax.ejb.Stateless;
import javax.ejb.Remote;
import javax.persistence.PersistenceContext;
import javax.persistence.EntityManager;
import javax.persistence.Query;
@Stateless
@Remote(Bank.class)
public class BankBean implements Bank {
    @PersistenceContext
    private EntityManager manager;
    public List<Account> listAccounts() {
        Query query = manager.createQuery ("SELECT a FROM Account a");
        return query.getResultList();
    }
    public Account openAccount(String ownerName) {
        Account account = new Account();
        account.ownerName = ownerName;
        manager.persist(account);
        return account;
    }
```

Access to the Entity

```
    public int getBalance(int accountNumber) {
        Account account = manager.find(Account.class, accountNumber);
        return account.balance;
    }
    public void deposit(int accountNumber, int amount) {
        Account account = manager.find(Account.class, accountNumber);
        account.deposit(amount);
    }
    public int withdraw(int accountNumber, int amount) {
        Account account = manager.find(Account.class, accountNumber);
        return account.withdraw(amount);
    }
    public void close(int accountNumber) {
        Account account = manager.find(Account.class, accountNumber);
        manager.remove(account);
    }
}
```

Persistence.xml

```
<?xml version= 1.0 encoding= UTF-8 ?>
<persistence xmlns= http://java.sun.com/xml/ns/persistence >
    <persistence-unit name= intro />
</persistence>
```

- A persistence unit is defined in a special descriptor file, the persistence.xml file, which is simply added to the META-INF directory of an arbitrary archive, such as an Ejb-jar, .ear, or .war file, or in a plain .jar file.

datasource configuration on Wildfly

http://www.mastertheboss.com/jboss-server/jboss-datasource/how-to-configure-a-datasource-with-jboss-7