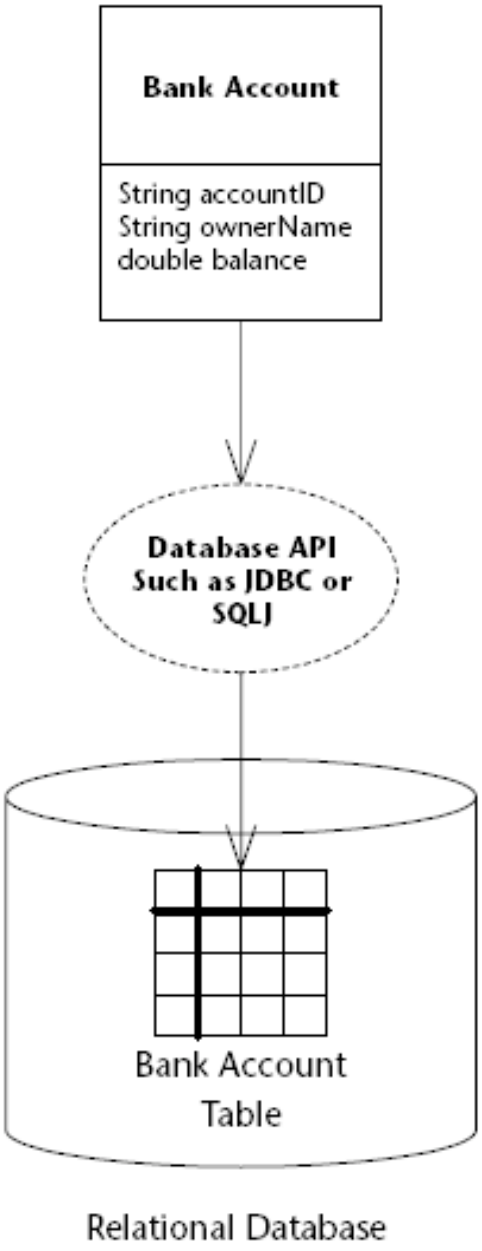
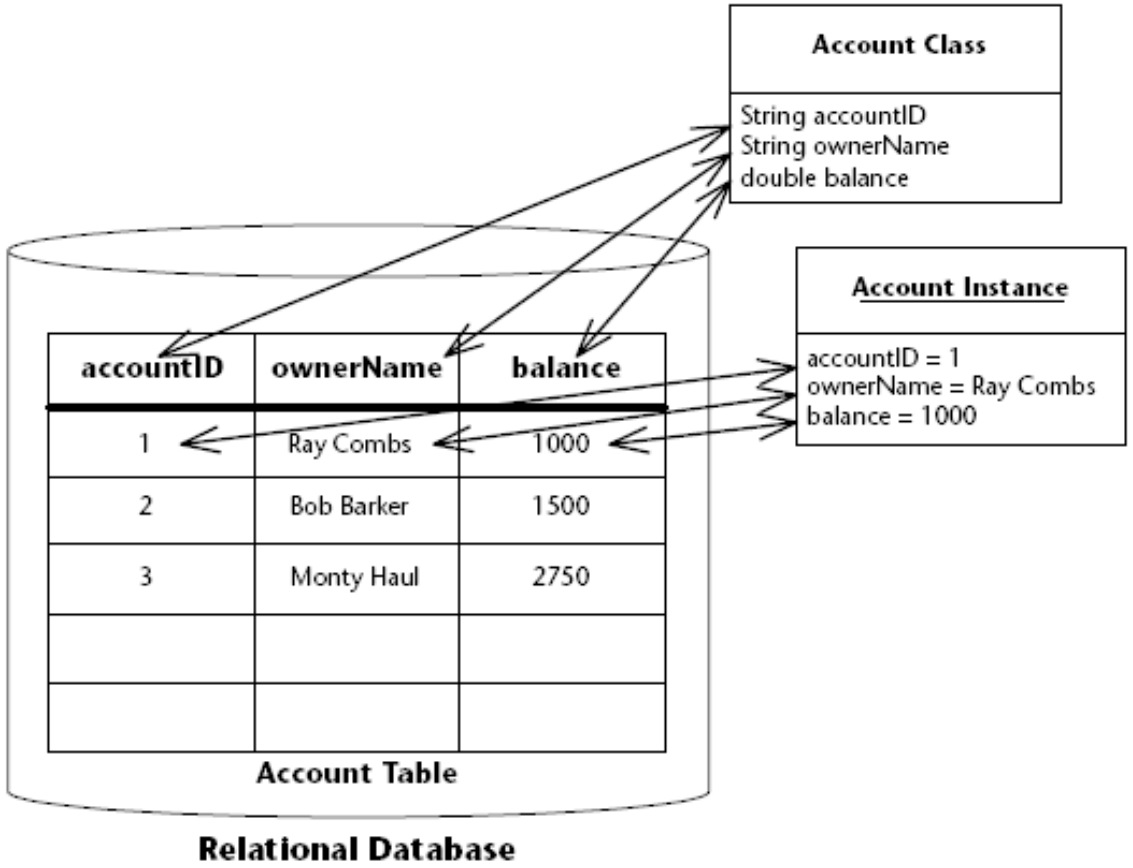
ORM

Object-Relational Mapping is NOT serialization!

You can perform queries on each field!





The Sun *Java Data Objects* (JDO) specification, defines portable APIs to a persistence layer that is conceptually neutral to the database technology used to support it. It can thus be implemented by vendors of relational and object-oriented databases.

The new Java Persistence specification finally defines a standardized objec-trelational mapping and requires compliant products to implement it. There is now a broad industry consensus on a portable programming model for persistent Java objects.

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 Source 6.1, the @Id annotation is applied to a field, 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.