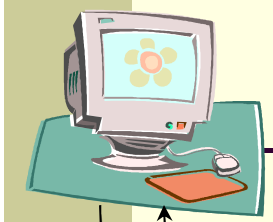
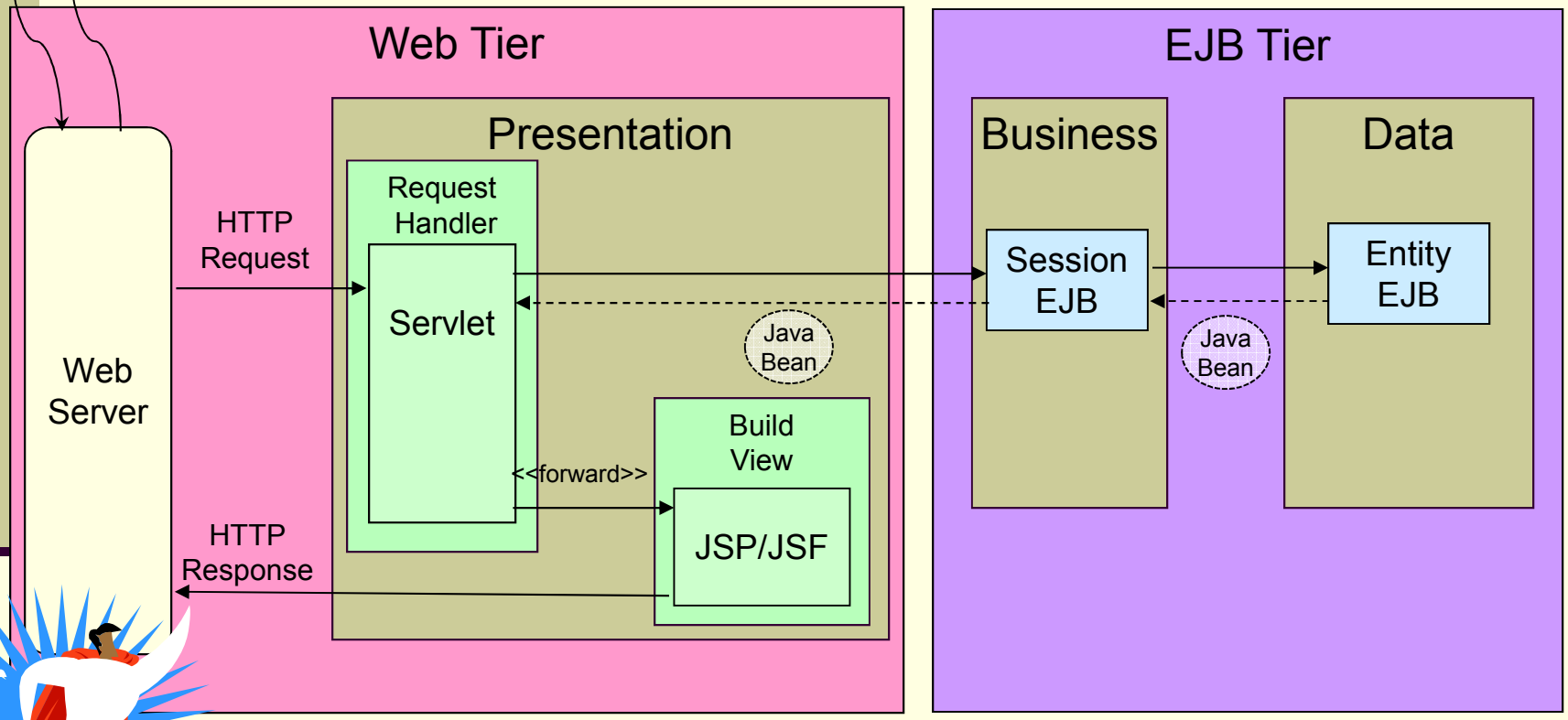
Enterprise Java Beans

R. Kent Jackson
jacksonk@byui.edu

Model 2 with JEE EJB's



Model Two Architecture



EJB Benefits

- Flexibility, Scalability and Adaptability
- Design based on best practices (design patterns)
- Facilitates communication (common framework)
- Reusability
- Saves development time/robust solution
  - Security
  - Transaction management
  - Concurrency control
  - Resource management
  - Persistence
  - Platform agnostic
  - Error handling

When should I Use JEE Model 2?

- Multiple apps share common data repository
- Must support transactions
- You need fine grained security (functional level)
- High Scalability
- High Availability
- Must support multiple clients

Java Beans vs. Enterprise Java Beans

Java Beans

- General purpose component
- A single class
- Executed and used anywhere

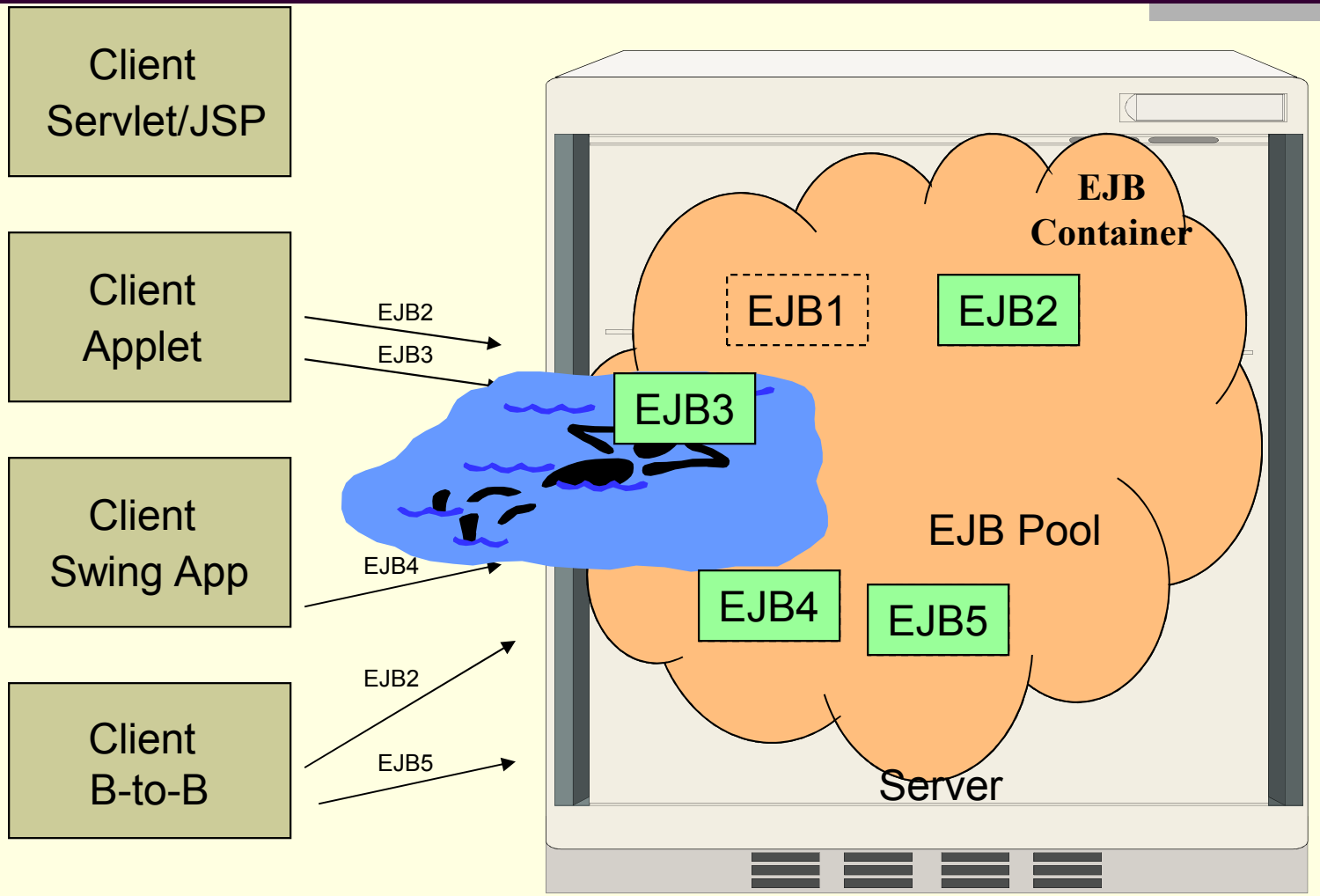
Enterprise Java Beans

- Highly specialized business logic components
- Collection of classes based on patterns
- Executed only in an enterprise container

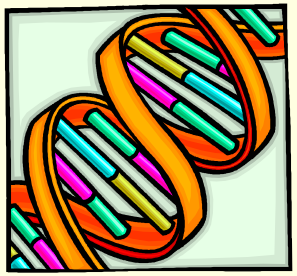
Three types of EJBs

- Session beans – control/business logic and transaction management (Control)
- Entity beans – to access data objects (database, file system, etc.) (Model)
- Message beans – to communicate with other applications

Managing resources in EJB Pool



EJB Entity Beans



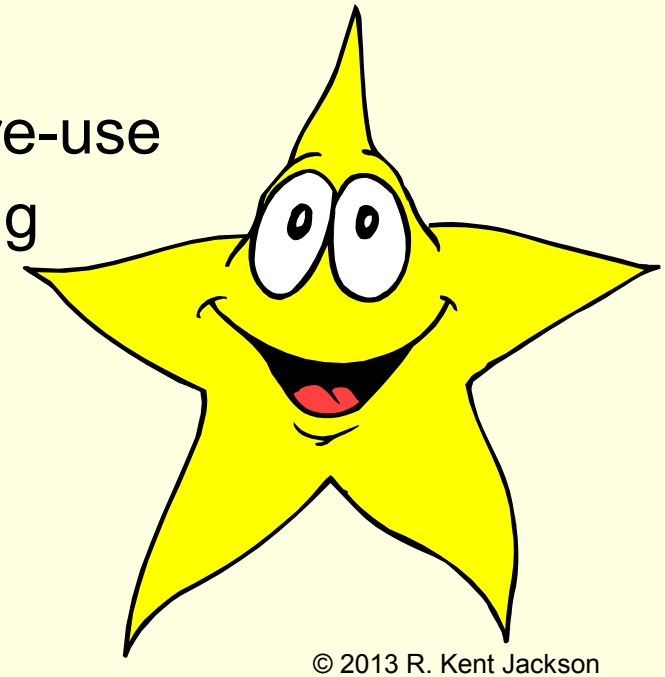
"Modeling your data"

What are Entity Beans?

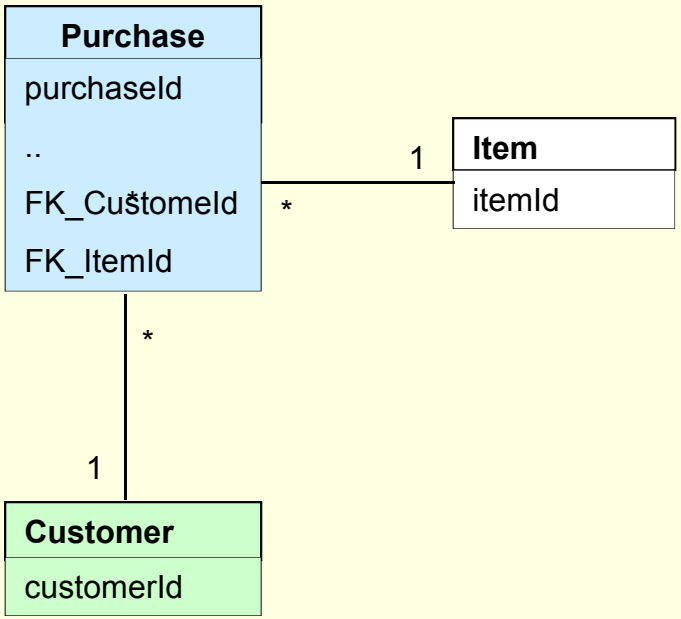
```
@Entity
public class Item implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @SequenceGenerator(name=”ItemGen", sequenceName="SEQITEM")
   @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="ItemGen")

   private BigDecimal itemId;
   private String description;
   private String price;

   @OneToMany(mappedBy = ”item")
   private Collection<Purchase> purchases;

   public Item() {
   }

   public BigDecimal getItemId() {
      return itemId;
   }

   public void setAddressId(BigDecimal itemId) {
      itemId= itemId;
   }

   public String getDescription() {
      return description;
   }
   …
}
```

Advantages

- Simpler to use than JDBC or ADO
- Automatic persistence management
- Share data in memory
- Automatic synchronization to the database
- Greater scalability, portability, maintainability, reliability, code re-use
- Automatic transaction processing



Relationships View



Use JDBC To Access Data

```
public class getSubTotal(long customerId) {

   Class.forName("org.gjt.mm.mysql.Driver"); // load jdbc driver classes

    String url = "jdbc:mysql://localhost/northwind";
    connection = DriverManager.getConnection(url, "USERNAME", ”PASSWORD");

    statement = connection.createStatement();
    String sql = "SELECT C.*, O.*, I.* " +
                 "FROM Customer C INNER JOIN Purchases O  INNER JOIN Items I " +
                 "ON C.customerId = O.FK_CustomerId " +
                 "ON O.FK_ItemId = I.ItemId";

    ResultSet resultSet = statement.executeQuery(sql);

    while( resultSet.next() ) {
        double price= resultSet.getDouble( “PRICE" );
        double quantity = resultSet.getDouble( “QUANTITY" );
        double total += price * quantity;
    }
}
```

Object Oriented View


Your Company Name

INVOICE

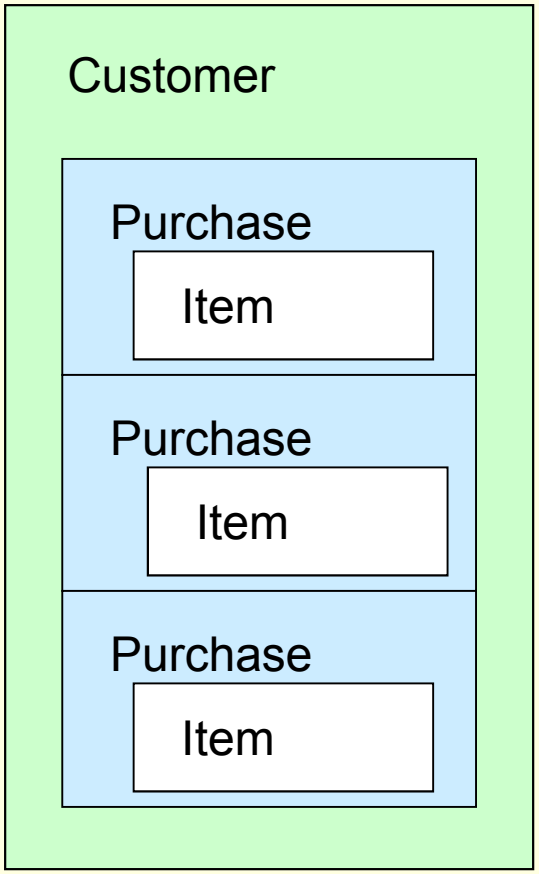
P.O. #	Sales Rep. Name	Ship Date	Ship Via	Terms	Due Date
0200012005	Sales1	11/17/2006	UPS	Net 7	

Product ID	Description	Quantity	Unit Price	Line Total
P1003	Motorola E815	10	420.00	4,200.00
P1000	Nokia 3220	12	199.99	2,399.88
P1004	Nokia 6230	5	200.00	1,000.00
P1002	Motorola V3 Razr Black	10	800.00	8,000.00

	SUBTOTAL	13,417.54
	PST 6.60%	885.74
	GST 3.20%	397.30
	SHIPPING & HANDLING	-
	TOTAL	14,622.94
	PAID	-
	TOTAL DUE	14,622.94

Notes:

THANK YOU FOR YOUR BUSINESS!



Entity EJBs

```
public double totalPurchases (long customerId) {
   double total = 0;
   Customer customer = entityManager.find(Customer.class, customerId);

  Collection<Purchase> purchases = customer.getPurchases();
  for (Purchase purchase in purchases) {
     total += purchase.getItem().getPrice() * purchase.getQuantity();
  }

  return total;
}
```

Insert object into Datastore

```
public void addBranch() {
      // create a branch java bean
      Branch branch = new Branch();
      branch.setName("Bank of Nauvoo");
      branch.setPhone("203-356-1426");

      // inserts a branch into the database
      entityManager.persist(branch);
}
```

Update Object in Datastore

```
public void renameBranch(String branchid, String newName) {
     // get branch by its Primary Key from datastore
     Branch branch = (Branch) entityManager.find(Branch.class, branchid);

     // update the branch
     branch.setBranchname(newName);
     entityManager.merge(branch);
}
```

Delete Object from Datastore

```
public void deleteBranch(String branchid) {
     // get branch by its Primary Key from datastore
     Branch branch = (Branch) entityManager.find(Branch.class, branchid);

     // Delete the branch
     entityManager.remove(branch);
}
```

Query Object/s from Datastore

```
public Collection<Branch> getBranches(String name){

   // Define query prepared statement
   String ejbql = "SELECT b FROM Branch b WHERE b.branchname LIKE :branchname";

   // Create query object
   Query query = entityManager.createQuery(ejbql);

   // Substitute value to search for in prepared statement
   query.setParameter("branchname", searchValue);

   // Execute query to get list of branches
   List<Branch>branches = query.getResultList();

return branches;
}
```

Entity Bean query language (ejb-ql)

select_clause from_clause [where_clause]

Simple Example:

```
SELECT j
FROM Job AS j
WHERE j.jobType='FN'
```

Entity Bean query language (ejb-ql)

Inner Joins of Entities

Complex Example:

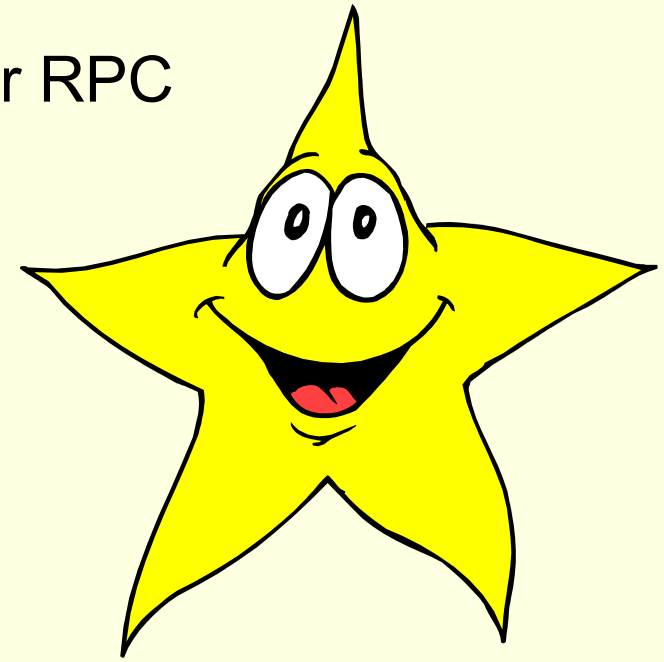
```
SELECT  i
FROM Customer AS c
INNER JOIN c.purchases AS o
INNER JOIN o.items AS i
WHERE i.price > 20.00
AND c.customerID = :customerId
```

EJB Session Beans

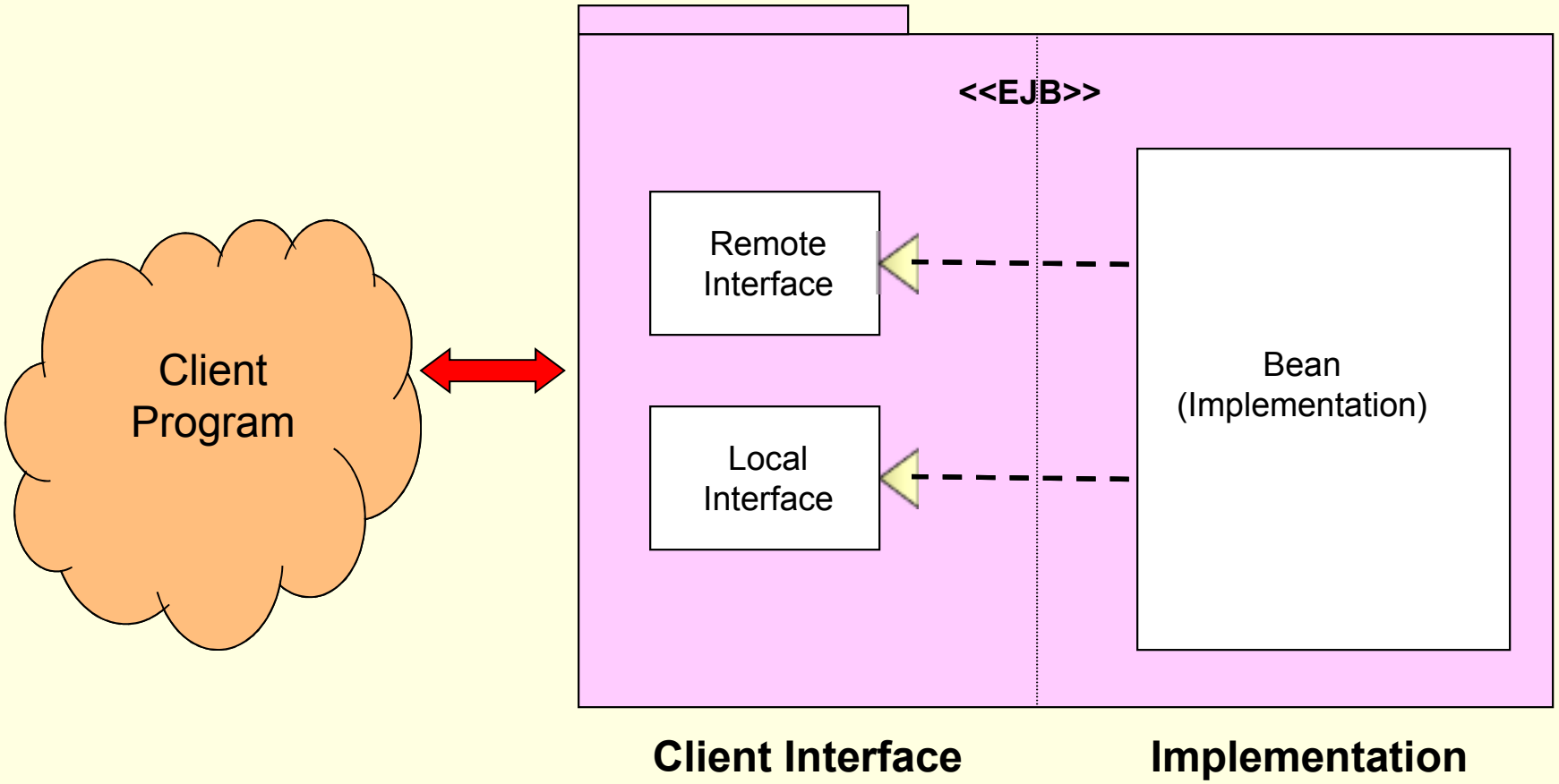
"Controlling your application"
(the business logic)

Advantages

- Automatic Transaction management
- Modularity
- Resource management
- Function level security
- Simple programming interface for RPC



Clients View of Session EJB



Session EJB Interfaces

```
@Remote
public interface BusinessFunctionsRemote {

    public Person login(String username, String password);
    public Collection<AtomMessage> getMessages(long personId);

}
```

AND/OR

```
@Local
public interface BusinessFunctionsLocal {

    public Person login(String username, String password);
    public Collection<AtomMessage> getMessages(long personId);

}
```

Bean Implementation Class

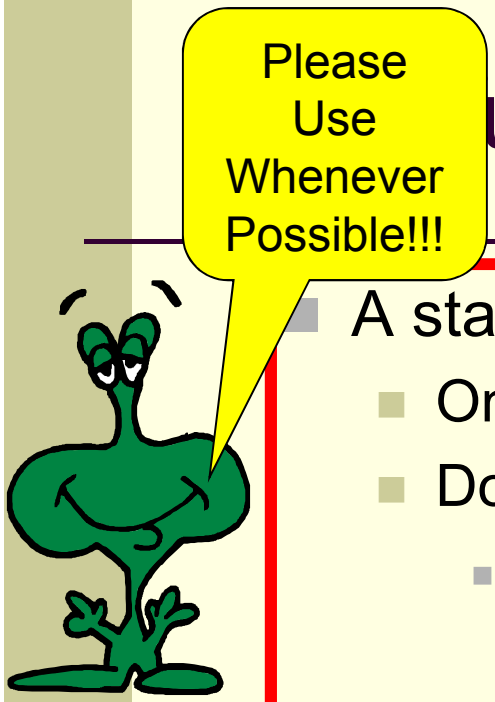
```
@Stateless
public class BusinessFunctions implements BusinessFunctionsRemote {

    @PersistenceContext
    EntityManager entityManager;

    @Override
    public Person login(String username, String password) {

        String ejbql = "SELECT p "
                + "FROM Person AS p "
                + "WHERE p.userName = :username "
                + "AND p.password = :password";

        Query query = entityManager.createQuery(ejbql);
        query.setParameter("username", username);
        query.setParameter("password", password);
        Person person = (Person) query.getSingleResult();

        return person;

    }

    @Override
    public Collection<AtomMessage> getMessages(long personId) {
        Collection<AtomMessage> messages;

        String ejbql = "SELECT k "
                + "FROM GroupPerson AS gp "
                + "INNER JOIN gp.groupId AS g "
```

Remote vs. Local Session Bean

- Remote Session EJB
  - Anywhere on the network
  - Pass by value

- Local EJB are faster but not as flexible, scalable and adaptable
  - On same server as Session Beans
  - No network traffic
  - Pass by reference

ul vs. Stateless Transactions



- A stateless transactions is
  - One call to server
  - Does not the save transaction state data
    - No memory is allocated in the pool for each session with a client
    - EJB is never passivated or activated
- Stateful transaction
  - Used for multi-step transactions
  - Saves state data for each client session
    - Memory is allocated in pool for each client session
    - EJB is passivated and activated

Guidelines for use

- Coarse-Grained better than Finely-Grained
- Use stateless whenever possible
- Use the remote interface for flexibility

Where is my session EJB?

- On remote server
- Use directory server to locate
- JNDI (Java Naming Directory Interface)

  “SAMS/authors/Martin Bond”

Outline for using remote EJBs

1. Get initial JNDI naming context
2. Use JNDI to lookup and find EJB
3. Call business function

Calling business methods

```
try
{
    InitialContext initialContext = new InitialContext(); // get jndi context

    // lookup and get remote interface for session bean
    BusinessRulesRemote businessRulesRemote =
            (BusinessRulesRemote) initialContext.lookup("java:global/classes/BusinessFunctions");

    // call any business methods defined in the interface
    Person person = businessRulesRemote.login(username, password);

}
catch (NamingException ne) {
    throw new MyAppException( “Session EJB not found, jndiname=” + jndiName));
}

catch (Exception e) {
    throw new MyAppException(ee.getMessage());
}
```

Calling business methods

```
try
{
    InitialContext initialContext = new InitialContext(); // get jndi context

    // lookup and get remote interface for session bean
    BusinessRulesRemote businessRulesRemote =
            (BusinessRulesRemote) initialContext.lookup("java:global/classes/BusinessFunctions");

    // call any business methods defined in the interface
    Person person = businessRulesRemote.login(username, password);

}
catch (NamingException ne) {
    throw new MyAppException( “Session EJB not found, jndiname=” + jndiName));
}

catch (Exception e) {
    throw new MyAppException(ee.getMessage());
}
```

Calling business methods

```
try
{
    InitialContext initialContext = new InitialContext(); // get jndi context

    // lookup and get remote interface for session bean
    BusinessRulesRemote businessRulesRemote =
            (BusinessRulesRemote) initialContext.lookup("java:global/classes/BusinessFunctions");

    // call any business methods defined in the interface
    Person person = businessRulesRemote.login(username, password);

}
catch (NamingException ne) {
    throw new MyAppException( “Session EJB not found, jndiname=” + jndiName));
}

catch (Exception e) {
    throw new MyAppException(ee.getMessage());
}
```

Calling business methods

```
try
{
    InitialContext initialContext = new InitialContext(); // get jndi context

    // lookup and get remote interface for session bean
    BusinessRulesRemote businessRulesRemote =
            (BusinessRulesRemote) initialContext.lookup("java:global/classes/BusinessFunctions");

    // call any business methods defined in the interface
    Person person = businessRulesRemote.login(username, password);

}
catch (NamingException ne) {
    throw new MyAppException( "Session EJB not found, jndiname=" + jndiName));
}

catch (Exception e) {
    throw new MyAppException(ee.getMessage());
}
```

Demonstration