Part III

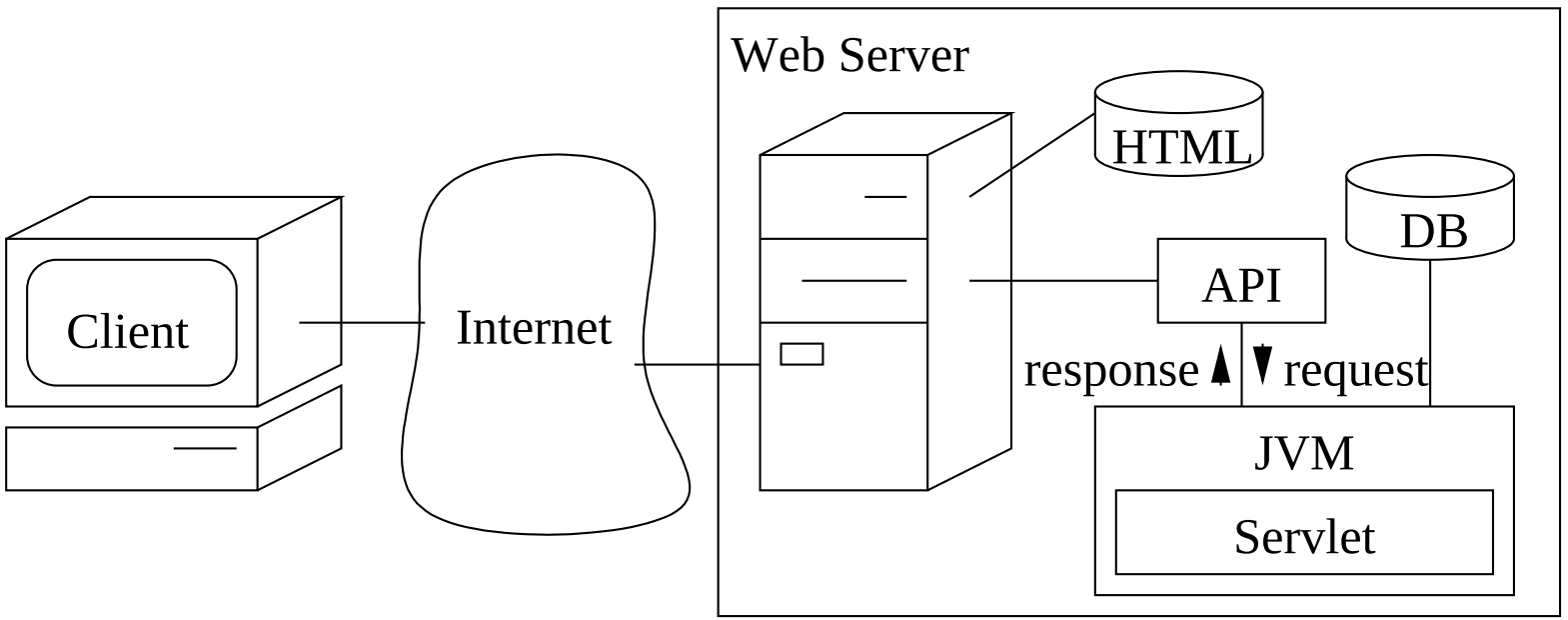
Java Platform, Enterprise Edition (Java EE)

Java Platform, Enterprise Edition (Java EE)

- (Java Servlets, JSP, and) JSF in web access layer
- Enterprise JavaBeans 3.1 for implementing business logic

1. Java Servlets

- executed by Java runtime environment, which is integrated into the web server (e.g. Apache)
- enable the dynamic generation of HTML pages
- for each request: new thread in JVM



Java Servlets (continued)

- servlets are (e.g.) collected in a special directory
- the server is configured, such that an access to a corresponding file causes it to be executed as a servlet
- request to a servlet via HTTP, e.g.:
  http://servername/servlets/Welcome?name=Bob+King
- parameter passing and result comfortably via request and response objects (as arguments of `doGet`) rather than via environment variables or standard input

Example: Java Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Welcome
             extends HttpServlet{
 public void doGet(
            HttpServletRequest req,
            HttpServletResponse res)
     throws ServletException,
            IOException{
   String name =
      req.getParameter("name");
   if ((name==null) ||
       (name.length()==0))
     name = "Unknown";
res.setContentType("text/html");
PrintWriter out =res.getWriter();
out.println("<HTML>\n
             <HEAD>\n
             <TITLE>Hello
             </TITLE>\n
             </HEAD>");
out.println("<BODY>\n
   <H1>Hello "+name+".</H1>");
out.println("</BODY>\n
             </HTML>");
out.close();}
}
```

2. JavaServer Pages (JSP)

- disadvantages of servlets: for web pages mainly consisting of static HTML and little dynamically computed parts, servlets mainly consist of output statements
- then better: logic (Java) included in HTML page
- thus: JSP
- JSP code internally transformed into a servlet
- (own) tag libraries allow to separate web design and logic

JSP Example 1: Hello

```
<HTML>
<HEAD><TITLE>Hello</TITLE>
<BODY>
<H2>JSP Example</H2>
<% if (request.getParameter("name") == null)
     out.println("Hello!");
   else out.println("Hello " +
              request.getParameter("name")+"!");
%>
<it>Welcome!
</it>
</BODY></HTML>
```

JSP Example 2: Square

```
<HTML>
<HEAD><TITLE>Square</TITLE>
<BODY>
<%@ include file = "/head.html" %>
<H2>JSP Example</H2>
<%@ page session="false" %>
<%@ page errorpage = "/error.jsp" %>
<% String firstname = request.getParameter("firstname");
   String lastname = request.getParameter("lastname");
   int z = Integer.parseInt(request.getParameter("number"));
   if ((firstname == null) || (lastname == null))
     throw new Exception("Please enter your name!");
   else out.println("Hello "+firstname+" "+lastname+"!"); %>
The square of <%= z %> is
<font color=red><%= square(z) %></font>.
<%@ include file = "/foot.html" %>
</BODY></HTML>
<%! private int square(int x){return x*x;} %>
```

Predefined Variables

- available in the Java code contained in a JSP page
- HttpServletRequest request
- HttpServletResponse response
- javax.servlet.jsp.JspWriter out
- HttpSession session
- ServletContext application
- javax.servlet.jsp.PageContext pageContext

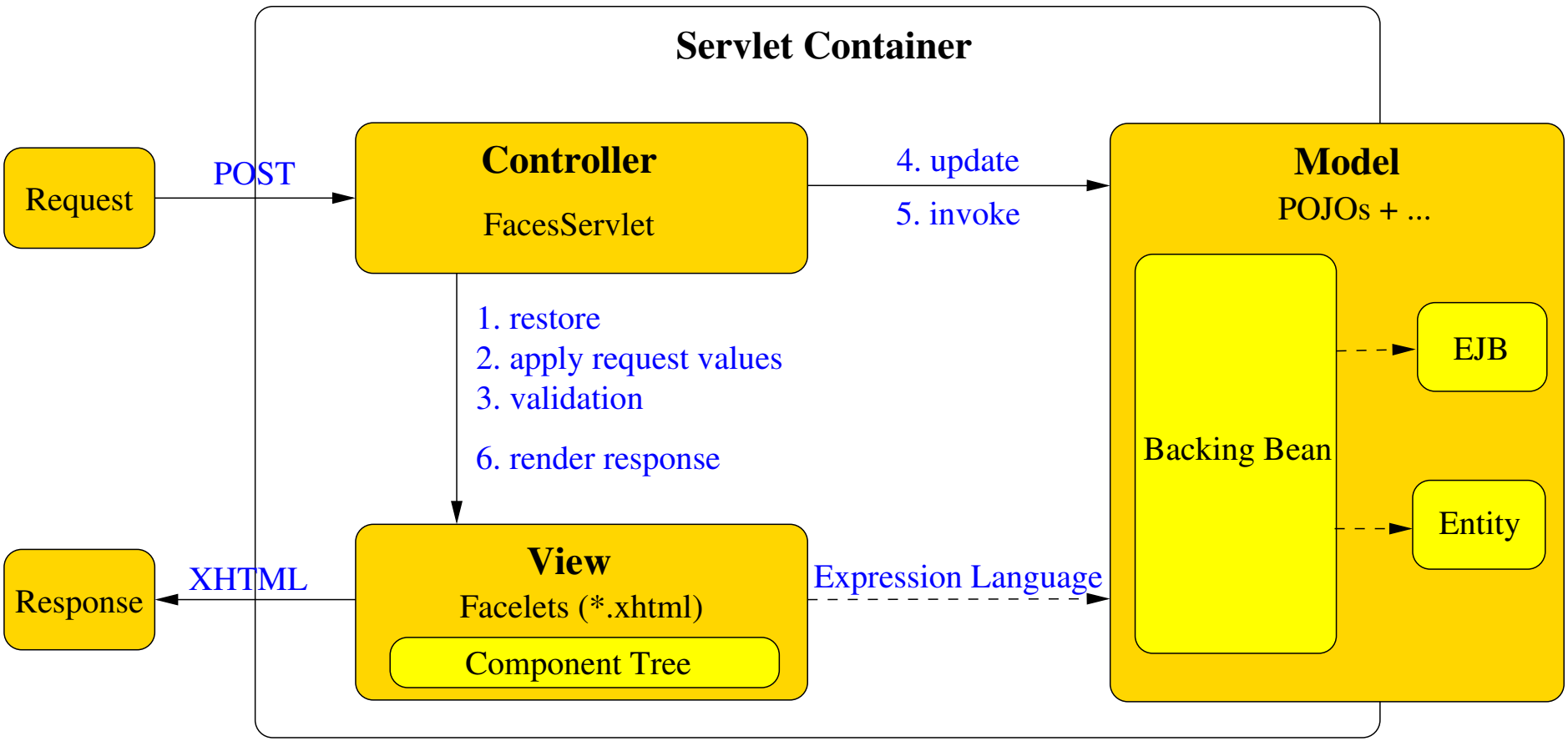
JSP Features

<% *c* %>	Java code *c* will be executed
<%= *e* %>	Java expression *e* is evaluated; the result is inserted as a String
<%! *d* %>	Java declaration *d* is inserted before `doGet` in the generated servlet
<%@ *d* %>	directive *d* will be executed when transforming JSP to a servlet; e.g. switch off cookies, determine error page, insert file
<jsp:useBean . . . > </jsp:useBean>	insert JavaBean ($\neq$ EJB)
<jsp:include page ="/my.jsp" > </jsp:include>	insert file at execution time
<jsp:forward page="login.jsp"> </jsp:forward>	redirect to other page

3. JavaServer Faces

- preferred framework for graphical user interfaces of web applications
- based on Java servlets and JSP technology
- MVC architecture
- separation of layout and programming aspects
- UI described by XHTML documents using predefined tags
- variables in XHTML document connected to attributes of backing bean
- overview of tags: http://download.oracle.com/javaee/6/tutorial/doc/bnarf.html

Overview JSF



Example: XHTML Description of Web Page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f=http://java.sun.com/jsf/core>
<h:head><title>New book</title></h:head>
<h:body><h:form>
  Title: <h:inputText id="title" value="#{createBook.book.title}"/>
         <h:message for="title"/><br/>
  Author: <h:inputText id="author" value="#{createBook.book.author}"/>
                                                          <br/>
  ISBN: <h:inputText id="isbn" value="#{createBook.book.isbn}">
          <f:ajax render="isbn-message" /> </h:inputText>
        <h:message id="isbn-message" for="isbn"/><br/>
        <h:commandButton value="Submit" action="#{createBook.submit}"/>
</h:form></h:body>
</html>
```

Example: Backing Bean

```
... imports

@ManagedBean
public class CreateBook{
  @EJB
  protected BookService bookService;

  protected Book book = new Book();

  public Book getBook() {return book;}

  public String submit(){
    // Action
    try{ bookService.createBook(book); }
      catch(EJBException e){ /* ... */ }
      // Navigation
      return "listBooks.xhtml";}
}
```

4. Enterprise JavaBeans

- Java-based middleware for distributed OO applications
- components (beans) are provided in EJB container
  (on application server)
- application servers: e.g. IBM WebSphere, GlassFish, JBoss ...
- the container offers frequently needed services for ISs
- advantage: basic services “for free”
- disadvantage: overhead; no OS calls (due to transactions)

Services provided by the Container

- managing and searching beans (name service based on JNDI)
- transactions (based on JTS and JTA)
- persistence
- accessing remote objects (based on RMI/IIOP or JMS)
- resource management (instance pooling, loading and unloading of of beans)
- authorization and authentication (based on JAAS)

Where are EJBs used?

- typical 4-Layer-Architecture:
  - client tier (HTML+JavaScript)
  - web tier (JSF, Servlets (,JSP))
  - business tier (EJB)
  - EIS tier (DB)

Kinds of Beans

- Entity (Bean):
  - encapsulates persistent data
  - accessed via session bean (or other entity)
- Session Bean:
  - implements business logic (e.g. use case)
  - not persistent
  - accessed from e.g. servlet or Java client (possibly via RMI)
  - variants: stateless and stateful
- Message-driven Bean:
  - processes asynchronous message
  - see chapter on message-oriented middleware for details

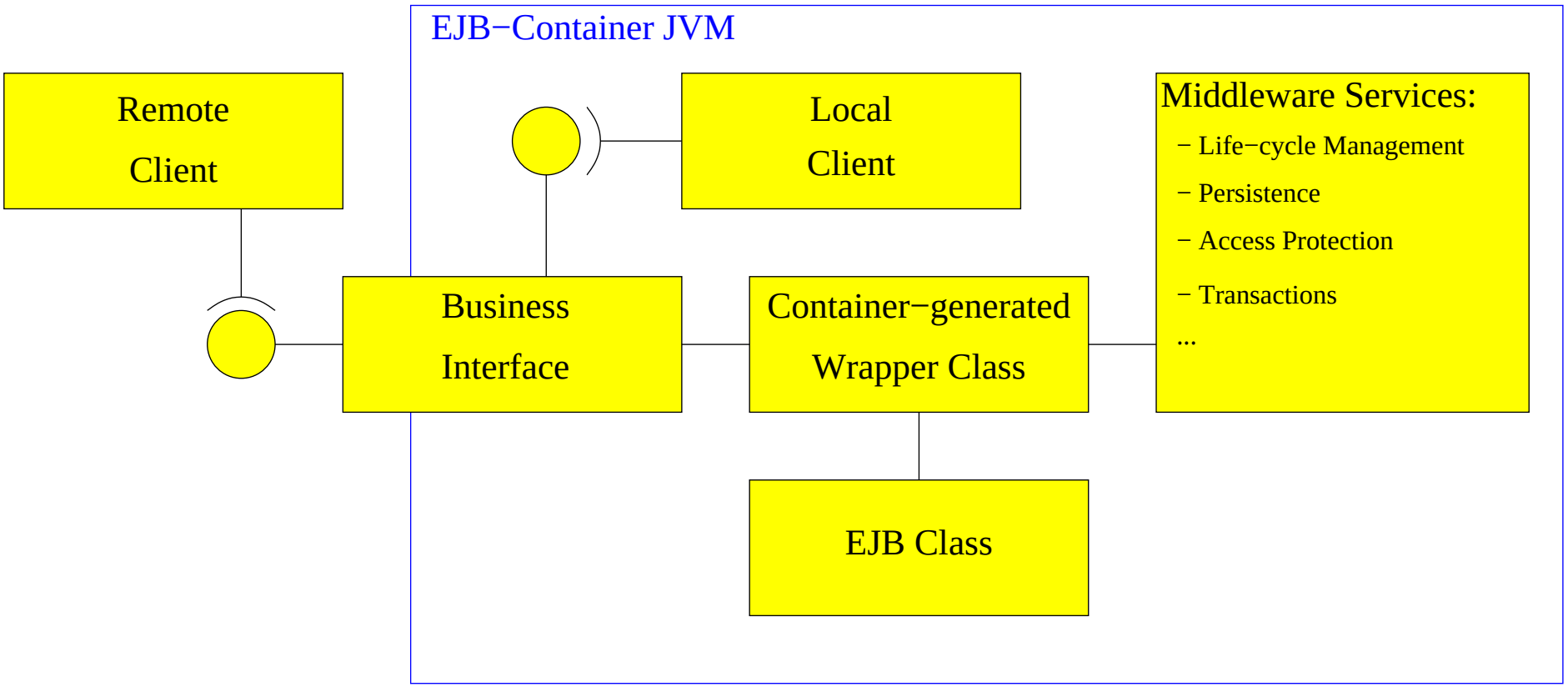
Variants of Session Beans

- Variant 1: stateless
  - efficiently and jointly used by several clients (successively, $\rightarrow$ pool)
  - state (local variables) available during method call
  - no continuous assignment to a client across a session
  - each request during a session is typically processed by a different bean
- Variant 2: stateful
  - exclusive for one client
  - state is maintained during a session (consisting of several requests)
  - state lost, if system crashes

Set-up of Enterprise JavaBeans

- Remote Interface (or Local Interface, if only locally accessed) for “business” methods (for session beans only)
- management operations (until EJB 2.X in Home Interface) now transparent (e.g. generation, deletion, activation, passivation)
- Bean Class:
  - implements the business methods
  - from EJB 3.0 on: “POJO” (plain old Java object)
- optional deployment descriptor:
  - XML document for the configuration of a bean w.r.t. persistence, associations (“relations”), transactions, primary key, . . .
  - from EJB 3.0 on typically replaced by annotations in bean class
- all together assembled in .jar-archive (packaging)

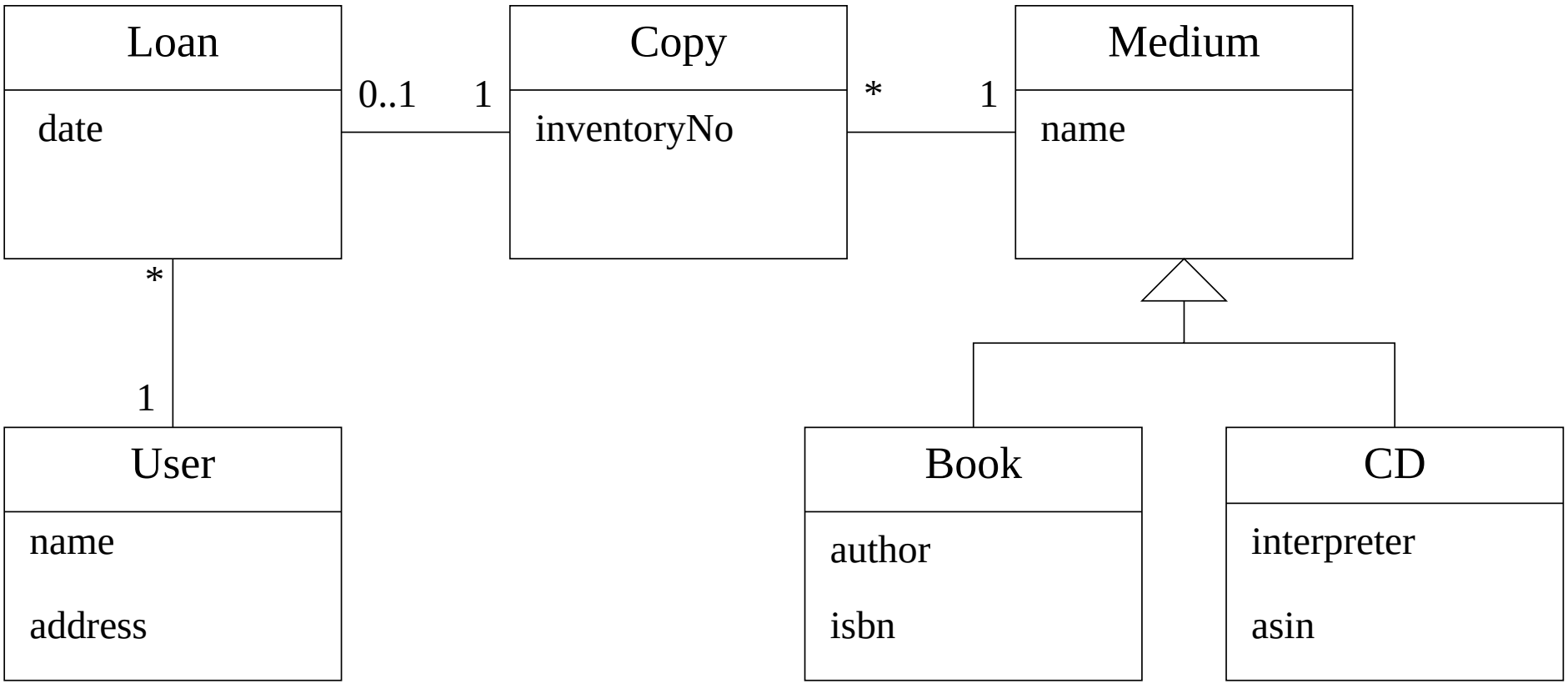
EJB 3.1 Programming Model



Deployment

- a bean is configured according to its annotations (or its deployment descriptor) and provided in a container
- auxiliary classes for integrating a bean class are automatically provided
- alternatively: comfortable, container-managed persistence or more flexible bean-managed persistence (with explicit JDBC calls)
- alternatively: container- or bean-managed transactions

Example Application: Library



Entity Class User

```
package library;

import java.util.ArrayList;  import java.util.Collection;  import javax.persistence.*;


@Entity
public class User implements java.io.Serializable {
   protected int uid;      /** primary key */
   protected String name;
   protected String address;
   protected Collection<Loan> loans = new ArrayList<Loan>();

   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   public int getUid(){return uid;}
   public void setUid(int id){uid = id;}

   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;}
```

Entity Class User (continued)

```
    @OneToMany(cascade = CascadeType.ALL, mappedBy="user")
    public Collection<Loan> getLoans(){return  loans;}
    public void setLoans(Collection<Loan> coll){loans = coll;}
    public void addLoan(Loan loan){loans.add(loan);}
    public void removeLoan(Loan loan){loans.remove(loan);}
}
```

Entity Class Loan

```
...
@Entity
public class Loan implements java.io.Serializable {
   protected int  id;        protected Date date;
   protected User user;      protected Copy copy;

   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   public int getId(){return id;}
   public void setId(int id){this.id = id;}

   public Date getDate(){return date;}
   public void setDate(Date d){date = d;}

   @ManyToOne
   @JoinColumn(name = "user")
   public User getUser(){return user;}
   public void setUser(User u){user = u;  u.addLoan(this);}

   @OneToOne
   @JoinColumn(name = "copy")
   public Copy getCopy(){return copy;}
   public void setCopy(Copy c){copy = c;  c.setLoan(this);}
}
```

Entity Class Copy

```
...
@Entity
public  class Copy implements java.io.Serializable{
   protected int inventoryNo;
   protected Medium medium;
   protected Loan loan;

   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   public int getInventoryNo(){return inventoryNo;}
   public void setInventoryNo(int no){inventoryNo = no;}

   @ManyToOne
   @JoinColumn(name = "medium")
   public Medium getMedium(){return medium;}
   public void setMedium(Medium m){medium = m; m.addCopy(this);}

   @OneToOne(cascade = CascadeType.ALL, mappedBy="copy")
   public Loan getLoan(){return loan;}
   public void setLoan(Loan lo){loan = lo;}
}
```

Entity Class Medium

```
...
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public abstract class Medium implements java.io.Serializable{
   protected int id;                    protected String name;
   protected Collection<Copy> copies;

   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   public int getId(){return id;}
   public void setId(int id){this.id = id;}

   public String getName(){return name;}
   public void setName(String n){name = n;}

   @OneToMany(cascade = CascadeType.ALL, mappedBy="medium")
   public Collection<Copy> getCopies(){return copies;}
   public void setCopies(Collection<Copy> coll){copies = coll;}
   public void addCopy(Copy c){copies.add(c);}
   public void removeCopy(Copy c){copies.remove(c);}
}
```

Entity Class Book

```
...
@Entity
public class Book extends Medium implements java.io.Serializable{
   @Pattern(regexp="[0-9X]*",message="only digits or X allowed")
   Size(min=10, message="at least 10 characters required")
   Column(unique=true)
   protected String isbn;
   protected String author;

   public String getISBN(){return isbn;}
   public void setISBN(String no){isbn = no;}

   public String getAuthor(){return author;}
   public void setAuthor(String a){author = a;}
}
```

class CD analogously

Example: Session Bean: Remote Interface

```
...
@Remote
public interface UserFacade {
  public void createUser(String name, String address)
                                                  throws Exception;
}
```

Example: Session Bean Class

```
...
@PermitAll
@Stateless
public class UserManagement implements UserFacade{
  @PersistenceContext
  private EntityManager em;

  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  public void createUser(String name, String address) throws Exception{
     Query q = em.createQuery("SELECT COUNT(*) FROM User u
                               WHERE u.name = :n");
     q.setParameter("n",name);
     if (((Long) q.getSingleResult()).intValue() == 0){
        User user = new User();
        user.setName(name);
        user.setAddress(address);
        em.persist(user);}
     else throw new Exception("Name previously used!");
  }
}
```

Annotations

- annotations determine the kind of bean: entity (@Entity) or session bean (@Stateless,@Stateful)

Annotations for Entities:

- @Id determines the primary key
- @Inheritance determines, how a class hierarchy is mapped to a relational database (SINGLE_TABLE, TABLE_PER_CLASS, JOINED)

  (Details later!)

Annotations for Session-Beans

- annotations for controlling the access to classes and methods: @PermitAll, @DenyAll (only for methods), @RolesAllowed("Role"), @RunAs("Role")
- @TransactionManagement(BEAN) replaces the (preset) container-managed transaction handling to a bean-managed one
- @TransactionAttribute allows to determine, how a method is integrated into a transaction (options: MANDATORY, NOT_SUPPORTED, REQUIRED (default!), REQUIRES_NEW, SUPPORTS)

Associations

- for ?:1-relationships, the considered class gets an attribute of the type of the neighborclass (and corresponding getters and setters)
- for ?:N-relationships, the class gets an attribute of type Collection<NeighborClass> (e.g. ArrayList)
- the multiplicity (@OneToOne, @OneToMany, @ManyToOne, @ManyToMany) is annotated before the getter of the corresponding attribute
- in case of a bidirectional association, one of both classes is responsible for the consistency
- for ?:N-relationships, the/a “N-site” shall be responsible for it
- the responsible site uses friend methods of the other site in order to ensure consistency

Parameters of Multiplicity Annotations

- parameter cascade specifies, whether the neighbor objects shall be updated, (persistently) changed, and/or deleted together with the considered object
  (@ALL, @MERGE, @REMOVE, @PERSIST, @REFRESH)
- for @ManyToMany, a table with a database mapping is used
- the annotation @JoinTable allows to fix details of the mapping table

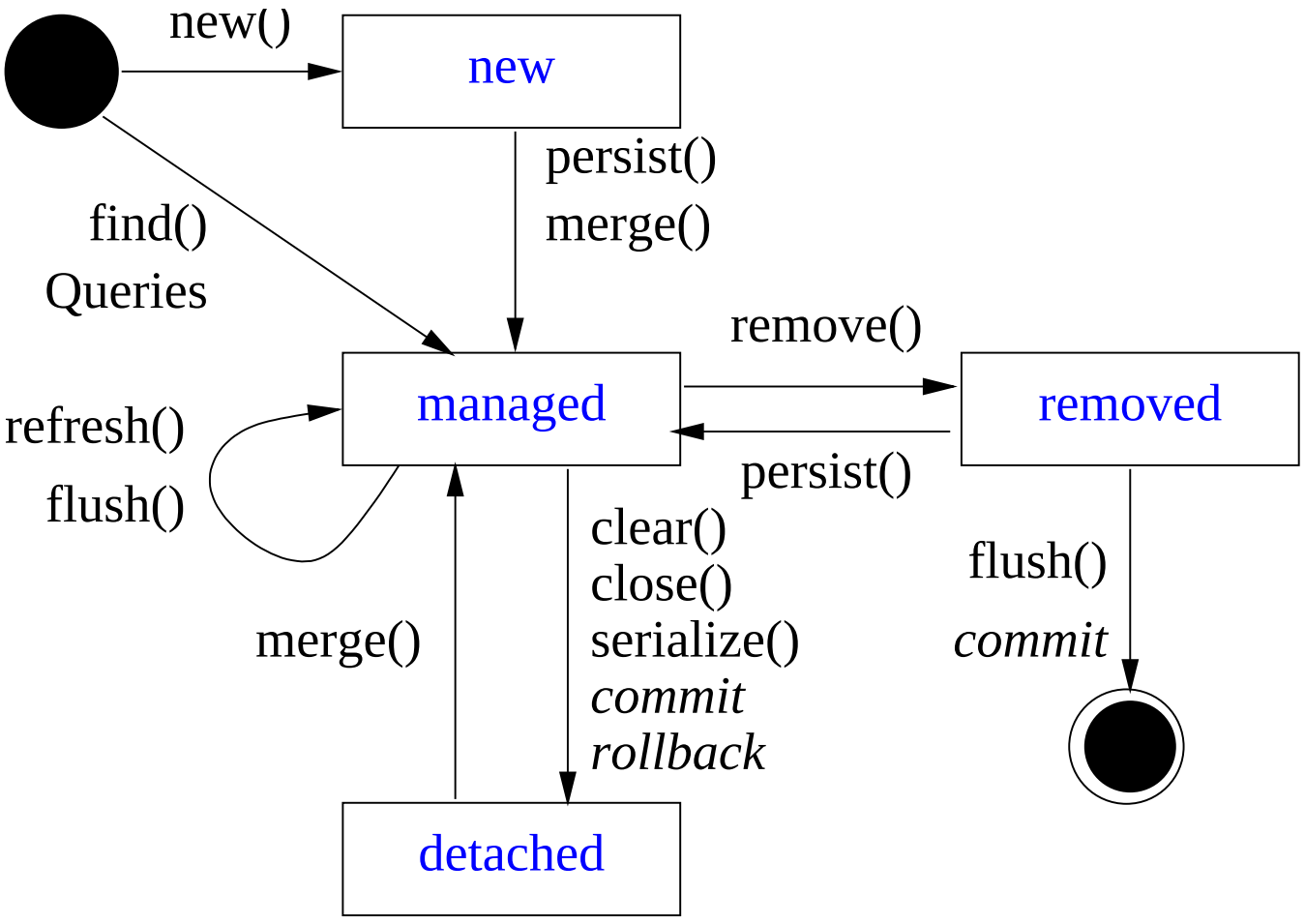
Parameters of Multiplicity Annotations (continued)

- the parameter fetch specifies, whether the neighbor objects shall be loaded from the DB together with the considered object (default for ?:1: EAGER, default for ?:N: LAZY)
- the parameter optional determines, whether null values are allowed
- the non-responsible site of an association uses parameter mappedBy to specify the foreign key of the opposite site

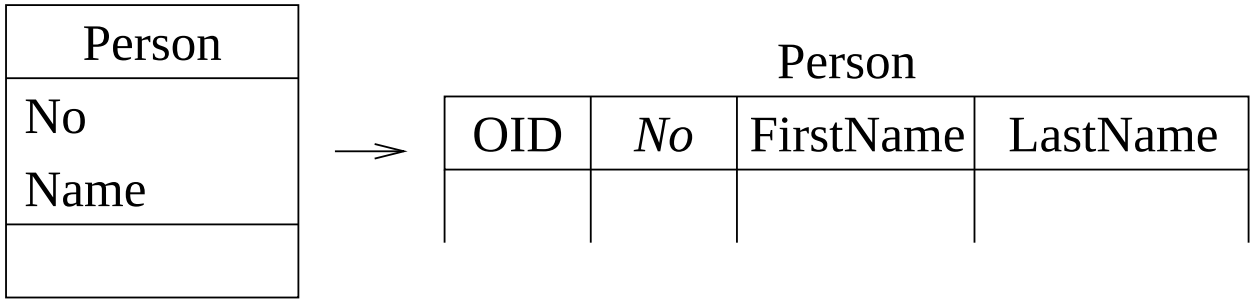
Features of Entities

- container-managed entities are not directly accessible “from outside” (from other JVM)
- however, they can be delivered as a result to clients
- this causes them to be “detached”, i.e. the container does not manage them any longer
- special result classes (data transfer objects) are no longer needed from EJB 3.0 on
- entities have to be serializable for transmission
- after transmission, entities loose their connection to lazily loaded neighbor objects
- entities and their attributes mus not be `final`

Life Cycle of an Entity



Transforming Classes to Relations

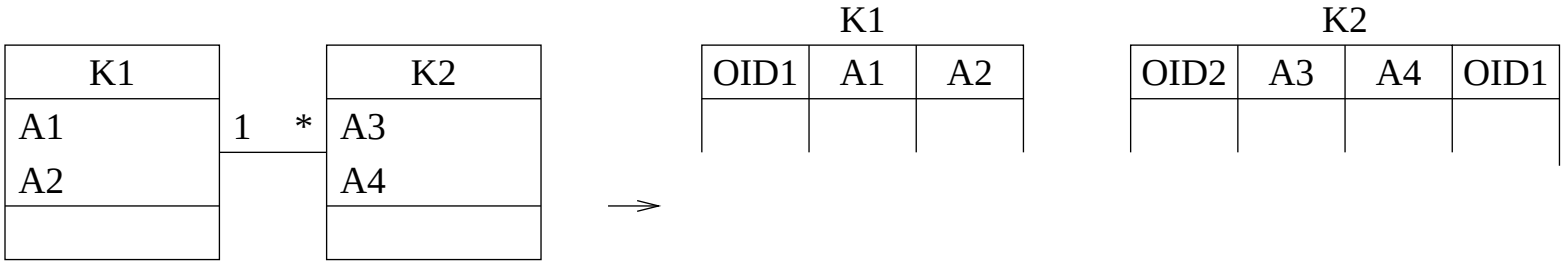



```
create table Person
( OID        number(8) not null,
  No         number(8) not null,
  FirstName  char(20),
  LastName   char(20) not null,
  primary key (No) );
create secondary index PersonIndex
  on Person(LastName);
```

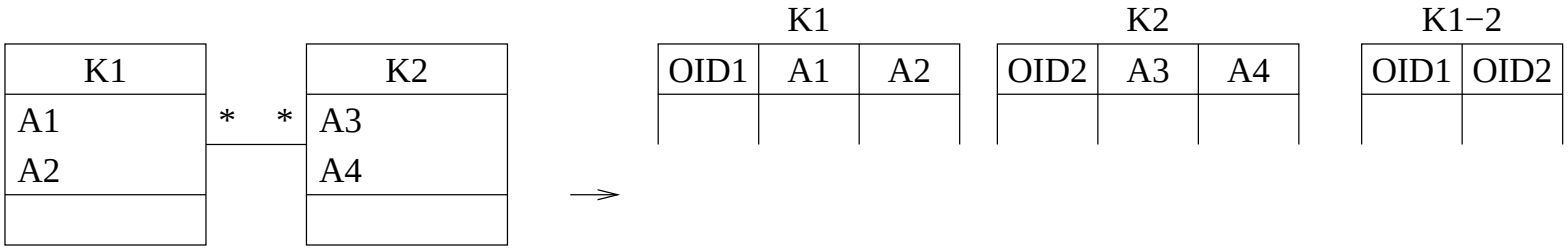
- mapping of attribute type to SQL type; possibly via embedded relation (@Embeddable,@Embedded)
- (possibly) add OID column
- fix primary key (@Id)
- fix optional attributes
- optional: create 2^{nd} index

Handling Associations

- for 1 : *m*-association/composition ($m \geq 1$): foreign key



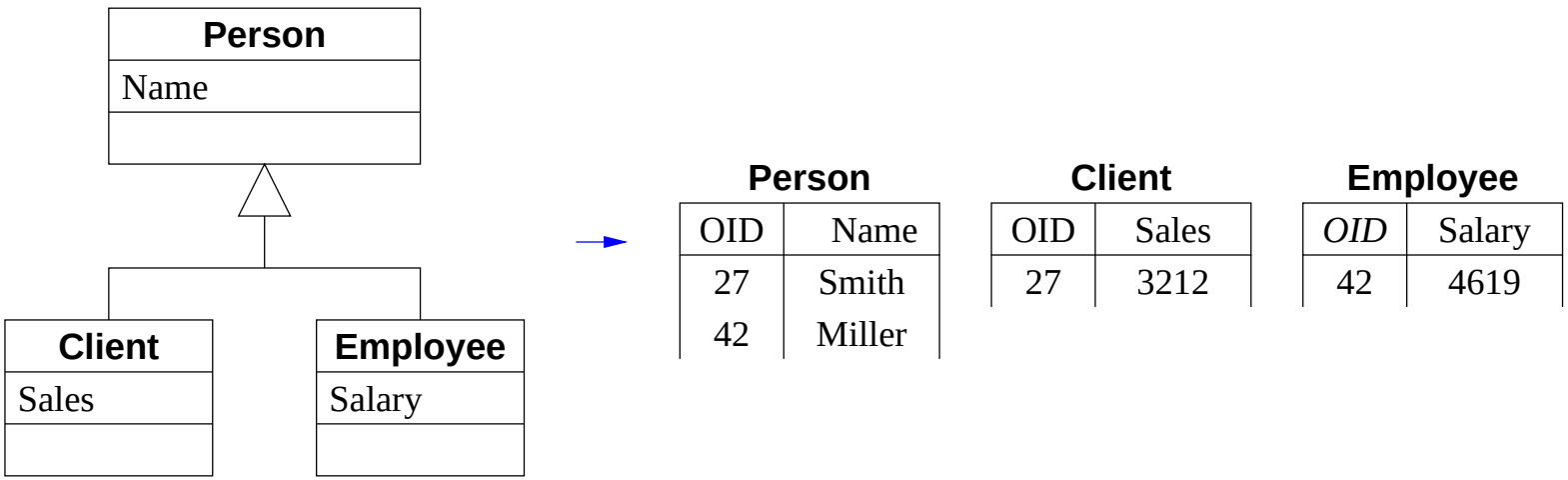
- for *n* : *m*-association/composition ($n, m > 1$): associative table



Transforming Inheritance

a) own table per class

- all tables contain primary key
- small schemata, but scattered information (overhead for $\bowtie$)
- 3NF
- JavaEE annotation @Inheritance(strategy = InheritanceType.JOINED)



Transforming Inheritance (2)

b) completed tables

Client

OID	Name	Sales
27	Smith	3212

Employee

OID	Name	Salary
42	Miller	4619

- for each concrete class, there is an own table with all attributes (including inherited ones)
- all superclass attributes are moved to the subclass tables
- overhead due to unions when accessing the superclass
- reasonable, if few superclass attributes
- 3NF
- annotation `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`

Transforming Inheritance (3)

c) one table for whole class hierarchy

Person

OID	Name	Group	Sales	Salary
27	Smith	Client	3212	NULL
42	Miller	Employee	NULL	4619

- move all subclass attributes to superclass table
- reasonale, if few subclasses with few attributes
- null values for missing attribute values
- violates 3NF
- JavaEE annotation `@Inheritance(strategy = InheritanceType.SINGLE_TABLE)`