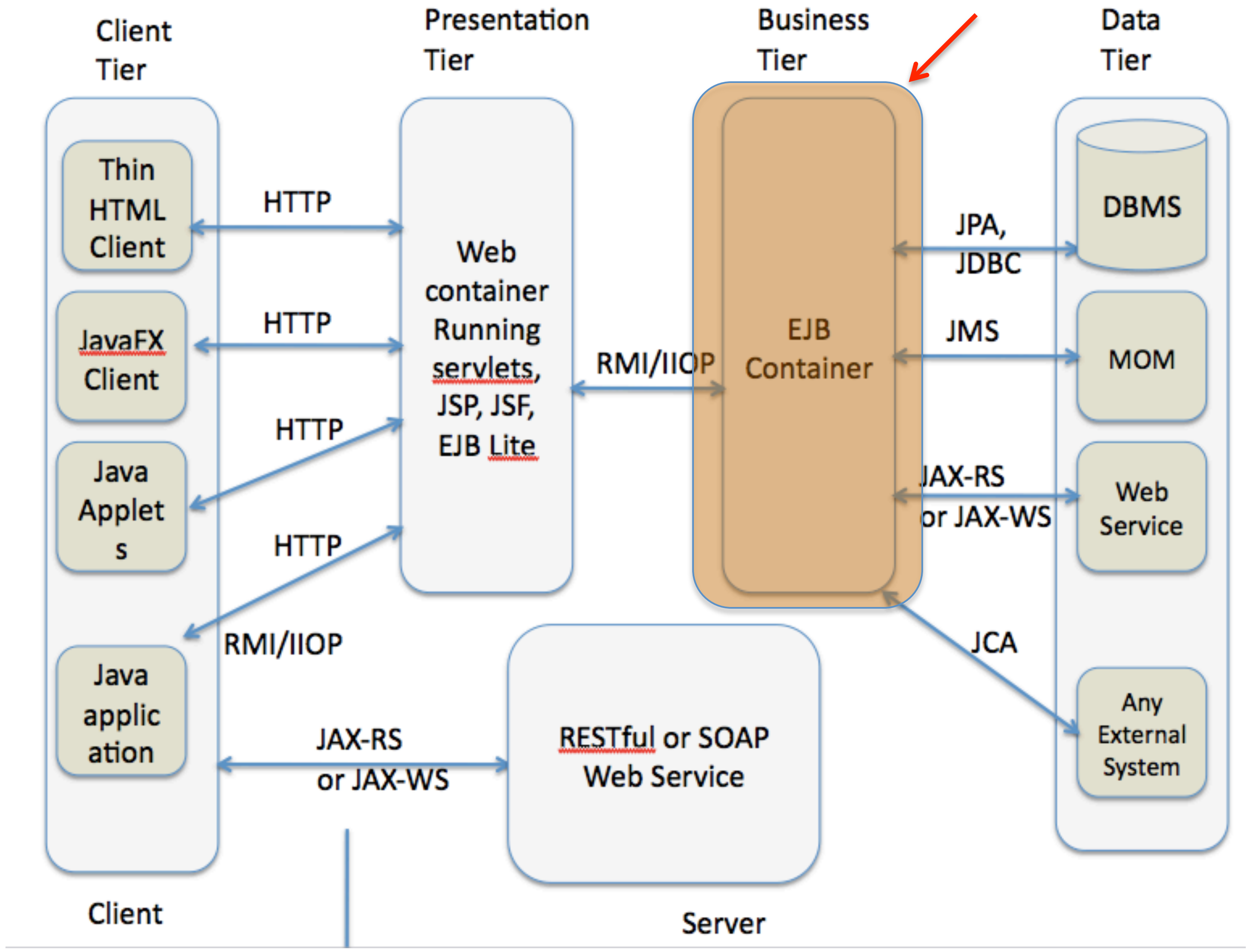
Java Programming Unit 17

Enterprise Java Beans.

Brief Overview of JPA

Enterprise Java Beans (EJB)



Major Benefits of EJB Containers

- No need to manually program multi-threading
- Availability of pooled resources (e.g. DB and JMS connection pools)
- Support of transactions (including the distributed ones)
- Support of Message-Driven Beans

Types of EJB

1. Session Beans (the business logic goes here)

2. Message-Driven Beans (consumers of messages from MOM)

Session Beans

Stateless session beans contain business logic, but don’t support state. An EJB container allocates any available in the pool bean to process client’s request.

Stateful beans contains both the business logic and the state. The EJB container allocates **specific instance** of the bean to the client and store the state between subsequent method invocations.

A Singleton session bean guarantees that there is only one instance of such a bean in the container. It’s like a global repository, where one bean can put some data to be used by another bean.

Packaging EJBs

- EAR – Enterprise Archive (.ear file)
- WAR – Web Archive (.war file)
- JAR – Java Archive (.jar file)

JAR can go inside WAR,WAR can go inside EAR

HelloWorld Session EJB

```
@Stateless
public class HelloWorldBean {

    public String sayHello(){
        return "Hello 
World!";
    }
}
```

```
@LocalBean
@Stateless
public class HelloWorldBean {

  public String sayHello(){
       return "Hello World!";
    }
}
```

`@LocalBean` - the clients of this bean run in the same JVM.

An EJB class can implements a business interface, which can be annotated as `@Local` or `@Remote`. Remote EJB must implement remote interface.

Servlet as a Client of EJB

```
@WebServlet("/HelloWorldServlet")
public class HelloWorldServlet extends HttpServlet {

~~//Context ctx = new InitialContext();~~
~~//HelloWorldBean myBean = (HelloWorldBean)~~
~~               ctx.lookup("java:global/Lesson32/HelloWorldBean");~~

  @EJB HelloWorldBean myBean;   // resource injection

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

      PrintWriter out = response.getWriter();
      out.println(myBean.sayHello());
  }
}
```

Walkthrough 1 (start)

- Download the code from Lesson 32 from the textbook site. Unzip it into a folder.
- In Eclipse, create new Dynamic Web Project called ServletEJB.
- Copy the content of the unzipped src folder into the Java Resources/src in Eclipse.
- Deploy the project to GlassFish 4 in Eclipse (right-click, menu Add and Remove)
- Start GlassFish and note the lines “Portable JNDI names” in server console.
- Run HelloWorldServlet – there is no output and no error messages.

Walkthrough 1 (end)

- Fix the bug in line 46 (replace `e.getStackTrace()` with `e.printStackTrace()`).
- Restart the server and re-run the servlet. You'll see the stack trace output in the server console – there is a wrong JNDI lookup string.
- Replace *lesson32* with *ServletEJB* in the lookup string (line 40).
- Restart the server if needed. Re-run the servlet – it print the output now.
- Comment out two lines that do JNDI lookup and add the class variable with resource injection instead:

  ```
  @EJB HelloWorldBean myBean;
  ```
- Run the program – the output is the same.

Stateful Session Beans

The client gets the same instance of the EJB for each method call:

```
MyShoppingCart myCart = (MyShoppingCart)
                       ctx.lookup("java:global/OnlineStore/MyShoppingCart");

// The client is browsing the catalog and finds the first item to buy
…
myCart.addItem(myFirstItem);

// The client continue browsing the catalog and finds the second item to buy
…
myCart.addItem(mySecondItem);

// The client is ready to check out
…
myCart.placeOrder();
```

To complete the shopping process and release the stateful bean for other clients call one of the bean's `MyShoppingCart` methods that's marked with `@Remove` (in this case annotate `placeOrder()`).

Transactions: CMT and BMT

- Container-managed transactions. No need to call commit or rollback in the application code.

- Bean-managed transactions. App code uses the `UserTransaction` interface, which has methods `begin()`, `commit()` and `rollback()`.

Methods' Transaction Attributes

- `Required` (default for CMT beans)
- `RequiresNew`
- `Mandatory`
- `NotSupported`
- `Supports`
- `Never`

Message-Driven Beans (MDB)

- The MDB's goal is to retrieve messages from queues and topics via JMS API.
- **The clients never need to access MDB directly.**
- The client needs to drop a message in a queue or publish it to a topic, and the MDB(s) listening to these destinations will get invoked.
- MDBs are stateless – they do not retain state from any specific client.
- No need to create `JMSContext`, `JMSConsumer`, and call `setMessageListener()`.

To become an MDB a class must:

- Have `@MessageDriven` annotation
- Have no-argument constructor
- Be public and non-abstract

Message-Driven Beans

MDB implements `MessageListener` interface.

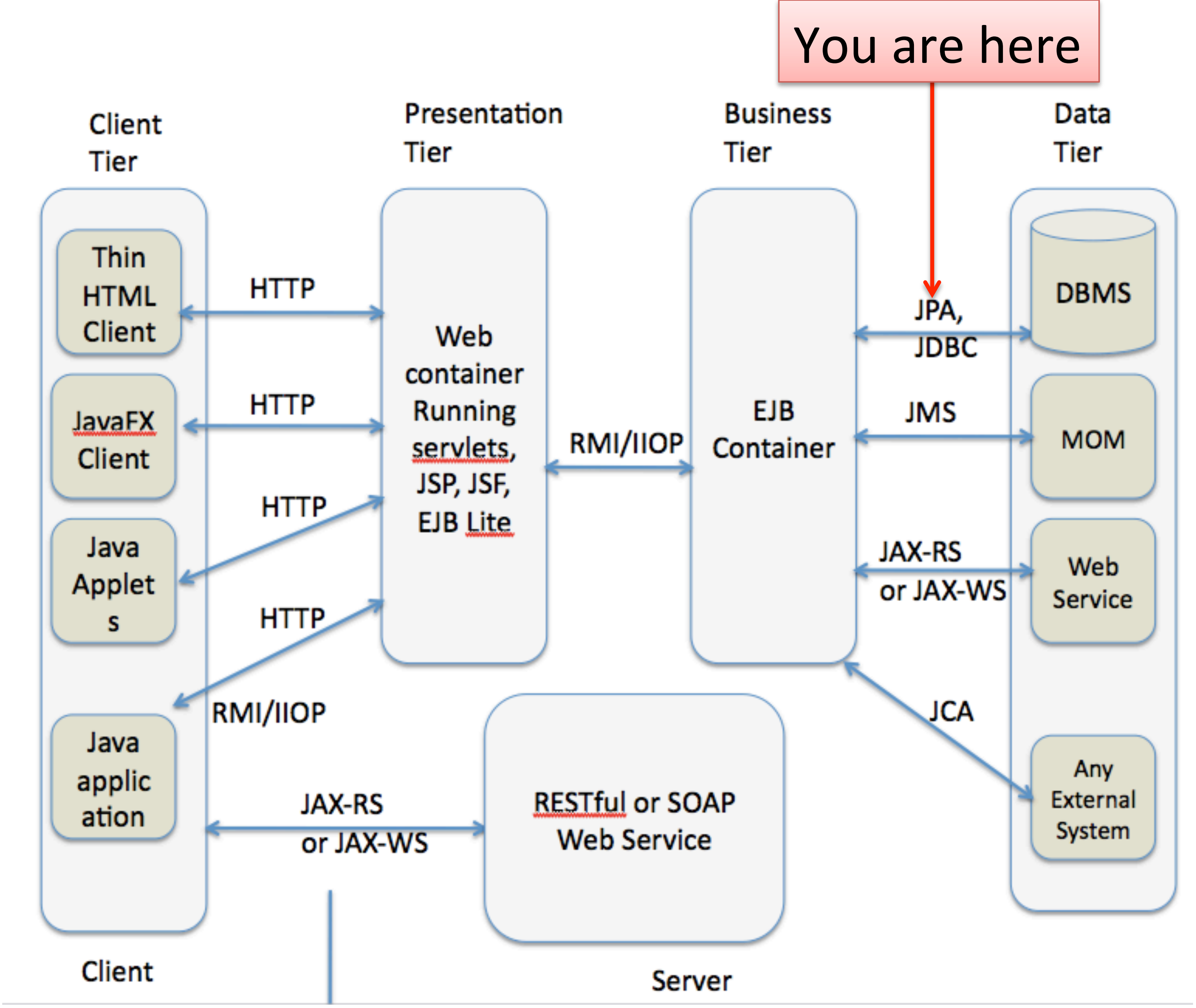
```
@MessageDriven(mappedName="jms/testQueue", activationConfig =  {
        @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                  propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType",
                                  propertyValue = "javax.jms.Queue")
    })
public class MyMessageBean implements MessageListener {

// security support, transaction rollback
@Resource MessageDrivenContext ctx;

  // A no-argument  constructor is required
  public MyMessageBean() {}

  public void onMessage(Message message){
     try{
     // The business logic is implemented here.
     // …
     catch(JMSException e){
        System.out.println(e.toString());
        ctx.setRollbackOnly();
     }
  }
}
```

Java Persistence API (JPA)

- JPA defines a standard way of mapping the Java classes to relational database tables. JPA often uses Object-Relational Mapping (ORM) with RDBMS.

- GlassFish comes with JPA implementation called EclipseLink, which supports not only relational DBMS, but the NoSQL data storage too.

- Many years ago EJBs had *entity beans.* Now they are replaced with frameworks that implement JPA.


You are here
Client Tier
Presentation Tier
Business Tier
Data Tier
Thin HTML Client
JavaFX Client
Java Applets
Java application
HTTP
HTTP
HTTP
HTTP
RMI/IIOP
Web container Running servlets, JSP, JSF, EJB Lite
RMI/IIOP
EJB Container
JPA, JDBC
JMS
JAX-RS or JAX-WS
JCA
DBMS
MOM
Web Service
Any External System
JAX-RS or JAX-WS
RESTful or SOAP Web Service
Client
Server

Three Ways of Working With RDBMS

With JPA you can map Java classes to database tables and perform Create Retrieve Update Delete (CRUD) operations using one of the following:

- Java Persistence Query Language (JPQL)

- Persistence Criteria API

- SQL language.

Entity Classes

A Java bean that's marked with `@Entity` is called an *entity*.

Each entity instance corresponds to a row in a database table.

If you start with an empty database, JPA tools can create database entities based on Java entities.

You can also map Java entities to the existing database tables.

```
@Entity
public class Employee{

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private Long id;

  @NotNull
  @Size(max=10)
  public String firstName;

  @NotNull
  @Size(min=2, max=20)
  public String lastName;

  @Column(name="boss_name")
  public String managerName;

  @OneToMany (mappedBy = "employee")
  public List<Address> addresses = new ArrayList<Address>();

}
```

Not every Java class that corresponds to some data in the database has to be an entity.

You can have embeddable classes that define a group of properties that belong to an entity.

Let's say a company gives to each employee a smart phone that identified by a phone number and the model.

You can create a Java class to represent this device and mark it with `@Embeddable`

```
@Embeddable
public class SmartPhone implements Serializable{

  @Size(max=10)
  public String phoneNumber;

  public String model;
}
```

```
@Entity
public class Employee{

 @Id
 @GeneratedValue(strategy=GenerationType.IDENTITY)

 @NotNull
 public String firstName;

…
 @Embedded
 public SmartPhone companyPhone;

}
```

JPQL

JPQL is a SQL-like query language.

But if SQL operates on DBMS tables, stored procedures, JPQL manipulates with Java objects and their attributes.

```
SELECT e.managerName,
FROM Employee AS e
WHERE e.lastName='Smith'

SELECT e.firstName, e.lastName
FROM Employee AS e
WHERE e.companyPhone.model='iPhone'

SELECT e FROM Employee AS e

SELECT DISTINCT e
FROM Employee AS e JOIN e.addresses as a
WHERE a.city = 'New York'
```

Entity Manager

EntityManager executes all your JPA requests to read from or to write into a database.

Each instance of EntityManager is associated with a set of entities. Such a set is called persistence unit.

```
@PersistenceContext EntityManager em;  //injection of Entity Manager

Employee employee = em.find(Employee.class, 1234); // find an employee with id=1234

@Resource UserTransaction userTransaction;
…
Employee newEmployee = new Employee();
newEmployee.firstName=”Mary”;
newEmployee.lastName=”Thompson”;
…
  userTransaction.begin();
  em.persist(newEmployee);
  em.remove(oldEmployee);

 userTransaction.commit();
```

Queries with Entity Manager

```
EntityManager em;
List employees;
…
employees = em.createQuery(
“SELECT e.managerName FROM Employee AS e
WHERE e.firstName=’Mary’ AND e.lastName=’Thompson’”).getResultList();
```

```
EntityManager em;
List employees;

String fName = “Mary”;
String lName = “Thompson”;
…
employees = em.createQuery(“SELECT e.managerName FROM Employee AS e
WHERE    e.firstName= :fname AND lastName= :lname”)
   .setParameter(“lname”, lastName)
   .setParameter(“fname”, firstName)
   .getResultList();
```

Homework

Complete the assignment from Lesson 32 from textbook.

Additional Reading

- The EJB tutorial form Oracle: http://bit.ly/1ntZfdO
- Transactions in EJBs: http://bit.ly/1pWpLjh
- Packaging EJBs: http://bit.ly/1uLloY8
- Java EE Web Profile: http://bit.ly/1hAG2YB
- JPA implementation in EclipseLink: http://bit.ly/1hK5cEq