Java EE Resources

- Deitel Textbook Chapters 29, 30, (and 31)
- Java EE Tutorial
  - http://docs.oracle.com/javaee/7/index.html
- Lynda.com
  - http://www.lynda.com/Java-tutorials/Java-EE-Essentials-Enterprise-JavaBeans/170059-2.html
  - http://www.lynda.com/Java-tutorials/Java-EE-Essentials-Servlets-JavaServer-Faces/124399-2.html

Java EE

- JEE addresses the need for distributed, transactional, and portable applications
- Provides the speed, security, and reliability of server-side technology
- Java EE platform is developed through the Java Community Process (the JCP), which is responsible for all Java technologies
- XML deployment descriptors are now optional:
- Developers enter the information as an **annotation** directly into a Java source file, and the Java EE server will configure the component at deployment and runtime
- Dependency injection can be used in EJB containers, web containers, and application clients.
- Dependency injection allows the Java EE container to automatically insert references to other required components or resources, using annotations.

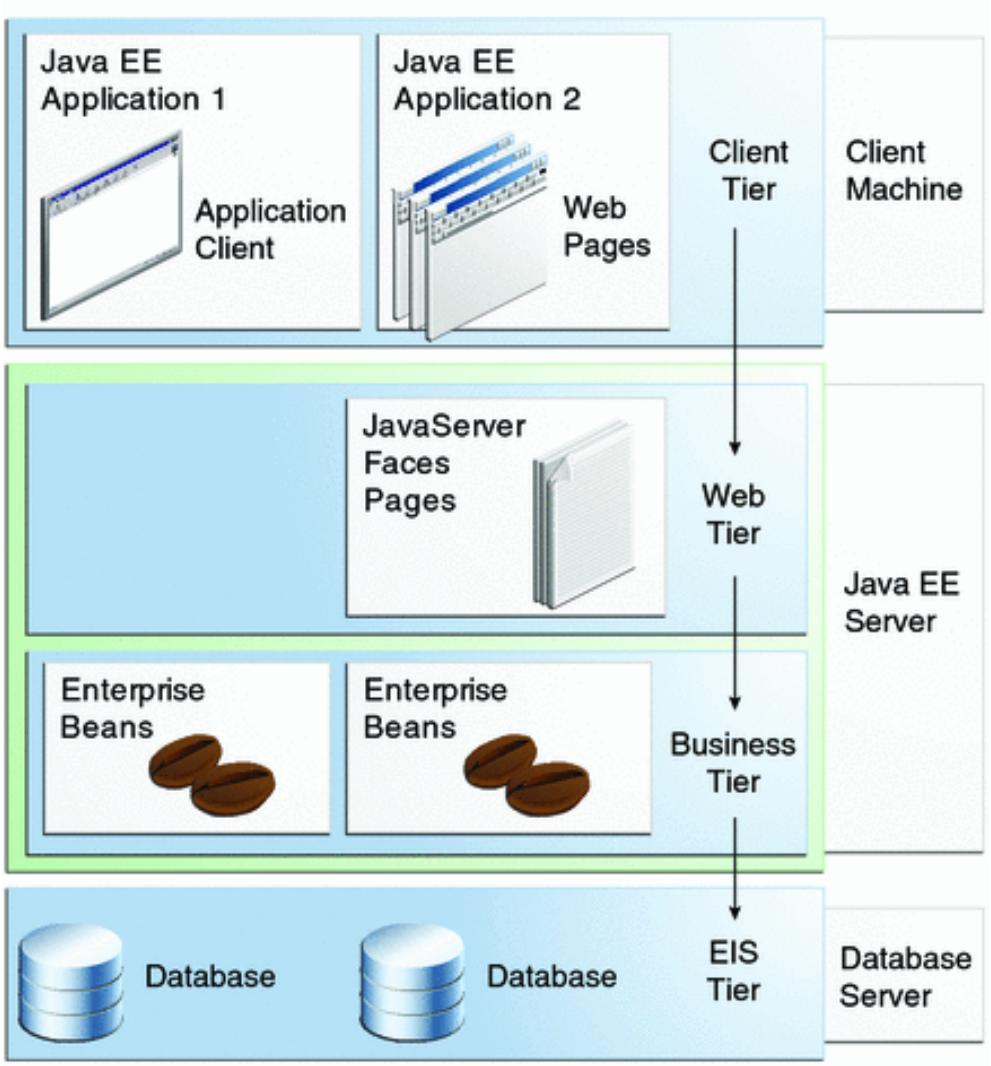
Java EE programming model

- Java programming language
- the Java virtual machine
- portability, security, and developer productivity these provide, form the basis of the application model
- Enterprise applications are inherently complex
- accessing data from a variety of sources and distributing applications to a variety of clients
- To better control and manage these applications, the business functions to support these various users are conducted in the middle tier
- The middle tier represents an environment that is closely controlled by an enterprise's information technology department.
- dedicated server hardware and has access to the full services of the enterprise

Distributed Multitiered Applications

- Application logic is divided into components according to function
- applications are generally considered to be three-tiered applications because they are distributed over three locations:
  - client machines,
  - the Java EE server machine,
  - and the database or legacy machines at the back end.

Multitiered Applications



Security

- Java EE security environment enables security constraints to be defined at deployment time
- Java EE platform makes applications portable to a wide variety of security implementations by shielding application developers from the complexity of implementing security features
- provides standard declarative access control rules that are defined by the developer and interpreted when the application is deployed on the server
- same application works in a variety of security environments without changing the source code

Java EE Components

- A **Java EE component** is a self-contained functional software unit
- A Component is assembled into a Java EE application with its related classes and files and it communicates with other components
- Application clients and applets are components that run on the client.
- Web components that run on the server:
  - Java Servlet
  - JavaServer Faces
  - JavaServer Pages (JSP)
- Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server

Java EE versus Java SE

- difference between Java EE components and “standard” Java classes is that EE components are
  - assembled into a Java EE application
  - verified to be well formed and in compliance with the Java EE specification
  - deployed to production (IT dept can focus on production)
  - run and managed by the Java EE Application Server
- Application Server containers manage the lifecycle of JEE components

Java EE Clients

- **Web Clients** (two parts)**:**
  - Dynamic web pages containing various types of markup language (HTML, XML, etc), which are generated by web components running in the web tier
  - A web browser, which renders the pages received from the server
- A web client is sometimes called a **thin client**
- With thin clients, heavyweight operations are usually handled by enterprise beans executing on the Java EE server, ie
  - query databases
  - execute complex business rules
  - connect to legacy applications
  - On the server side (Java EE), these can leverage
    - Security
    - Speed
    - Services
    - Reliability

Java EE Clients

- **Application Clients**
  - These run on a client machine and provide a way for users to handle tasks that require a richer user interface than can be provided by a markup language
  - typically have a GUI based on JavaFX, Swing or the Abstract Window Toolkit (AWT) API (Or on the other extreme, Command line interface)
  - directly access enterprise beans running in the business tier
  - can open an HTTP connection to establish communication with a servlet running in the web tier
  - This way, over HTTP, clients on other platforms (Android) or written in languages other than Java can interact with Java EE servers (loose coupling)

Applets

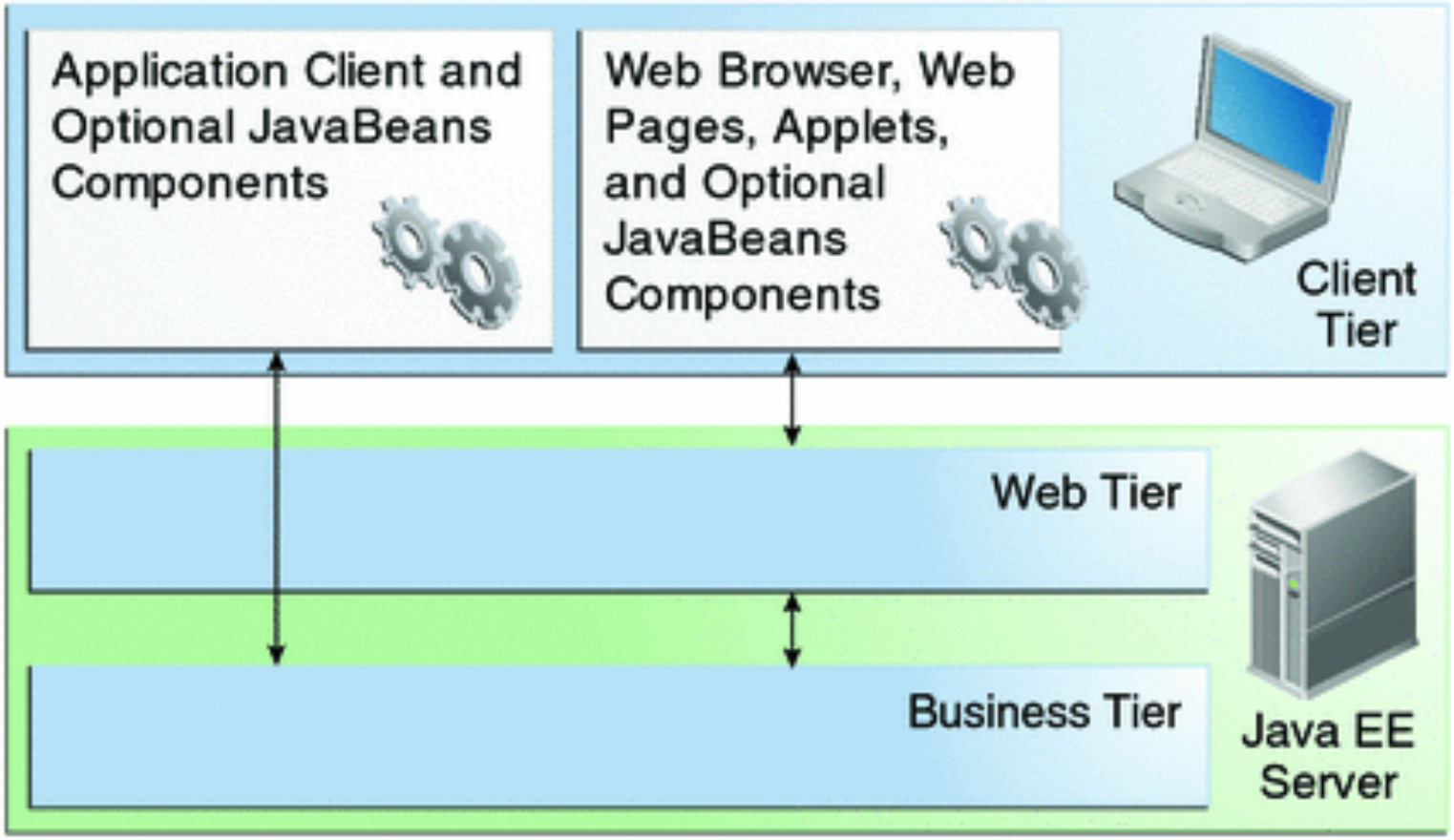
- an **applet** is a small client application that executes in the Java virtual machine installed in the web browser
- Big disadvantages of applets
  - Client systems need the Java Plug-in
  - Security policy file
- Web components are the preferred API for web clients
  - No applet disadvantages
  - Clean separation (partitioning) of web page design from application programming (Java) -- two different groups of creators: design vs programmer

JavaBeans Components

- A JavaBean is a Plain Old Java Object (POJO), but with certain characteristics
- JavaBeans conform to naming and design conventions:
  - all attributes are private
  - there are get and set methods (adhering to naming conventions) for accessing all attributes
  - an attribute with such getter and setter methods is called a **property**
  - so now we may use "property" two ways: 1. generally as a synonym for "attribute", or 2. as an attribute together with getter and setter methods adhering to JavaBeans naming conventions
- Enterprise JavaBeans are a specific kind of JavaBean, a component of JEE applications:
  - Message-driven Beans
  - Session Beans
    - stateless
    - stateful

Java EE Server Communications

Client communicates directly with business tier, or if running in a browser, then through web pages or servlets in web tier



Java EE Web Components

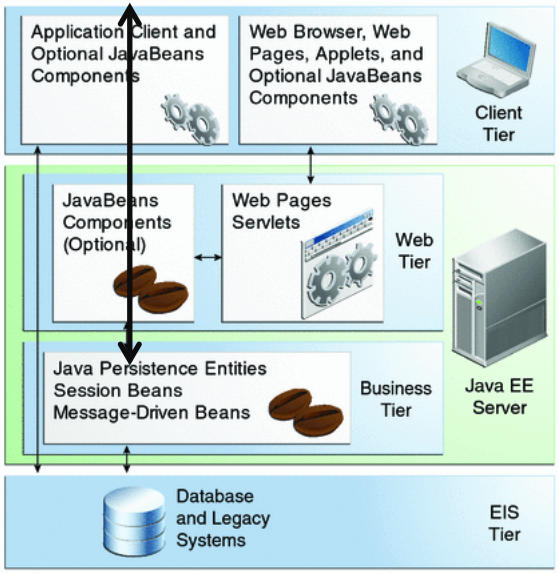
- These are either servlets or web pages created using JavaServer Faces technology and/or JSP technology (JSP pages)
- Servlets: java classes that dynamically process requests and construct responses
- JSP pages are text-based documents that execute as servlets but allow a more natural approach to creating static content
- JavaServer Faces technology builds on servlets and JSP and provides a user interface component framework for web apps
- Static HTML pages, applets, and server-side utility classes are not considered web components

Business Components

- Business code is logic that meets the needs of a business domain:
  - Banking
  - Retail
  - Finance
  - Factory production
  - etc
- Business code runs in enterprise beans running in either the business or web tier

Business and EIS Tiers

An enterprise bean receives data from client programs, processes it, and sends it to the enterprise information tier for storage



Enterprise Information System Tier

- Enterprise infrastructure systems:
  - Enterprise resource planning (ERP)
  - Mainframe transaction processing
  - Database systems
  - Legacy information systems

Implementations of JEE Application Server Specification

- Glassfish opensource
- Oracle Glassfish
- IBM WebSphere
- Apache Geronimo
- Fujitsu Interstage Application Server

- Redhat JBoss

- Apache Tomcat (web container)

- Implement the same spec, but differ in extra features (like connectors, clustering, fault tolerance, high availability, security, etc), installed size, memory footprint, startup time, etc.

Java EE Containers

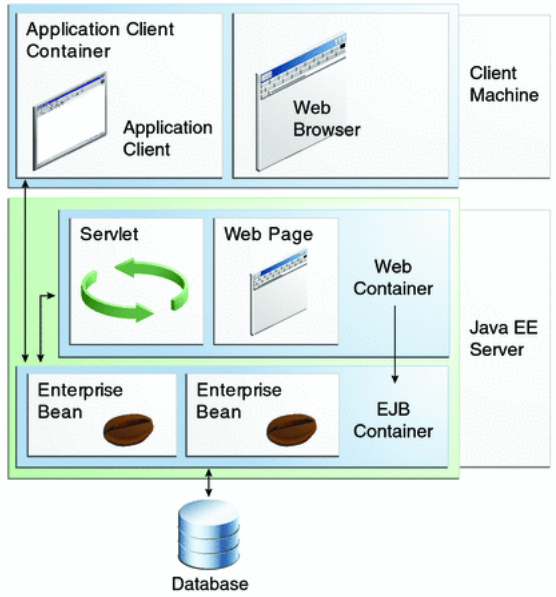
- thin-client multitiered applications are hard to write:
  - Transaction and state management
  - Multithreading
  - Resource pooling
  - Complex low-level details…
- the Java EE server provides underlying services in the form of a container for every component type
- Containers form the interface between the component and the low-level platform specific functionality
- Each web, enterprise bean, or application client component must be assembled into a Java EE module and deployed in its container

Java EE Containers

- Container settings customize the underlying support by the JavaEE server:
- **Security**: Java EE security model lets you configure a web component or enterprise bean so that system resources are accessed only by authorized users.
- **Transaction Management**: lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
- **Java Naming and Directory Interface (JNDI) lookup services**: provide a unified interface to multiple naming and directory services in the enterprise so that application components can access these services.
- **Remote connectivity** model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.

Container Types

Java EE server: The runtime portion of a Java EE product, provides EJB and web containers.
Enterprise JavaBeans (EJB) container: Manages the execution of enterprise beans, which, with their container, run on the Java EE server.
Web container: Manages the execution of web pages, servlets, and some EJB components for Java EE applications. Web components and their container run on the Java EE server.
Application client container: Manages the execution of application client components, which, with their container run, on the client.
Applet container: Manages the execution of applets, and consists of a web browser and Java Plug-in running on the client together.



Packaging Java EE Applications

- A Java EE application is delivered in a
  - Java Archive (JAR) file
  - Web Archive (WAR) file (jar file with .war extension)
  - Enterprise Archive (EAR) file (jar file with .ear extension)
- Deployment Descriptor (optional): a declarative (XML) document, describes the deployment settings of an app, module or component
- Java EE Server reads the deployment descriptor at runtime
- Two types:
  - Java EE deployment descriptor: configure deployment settings
  - Runtime deployment descriptor: configure Java EE implementation-specific parameters
- Example, GlassFish deployment descriptor contains
  - Context root of a web application
  - GlassFish Server implementation-specific parameters e.g. caching directives

Enterprise JavaBeans

- Enterprise Bean is a body of code having fields and methods to implement modules of business logic
- Two types
  - **Session Bean**: usually represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone.
  - **Message-driven Bean**: combines features of a session bean and a message listener, allowing a business component to receive messages asynchronously. Commonly, these are Java Message Service (JMS) messages.

Java Servlet Technology

- Usually not needed explicitly in JEE programming. Like RMI, Servlets are behind the scenes to implement JSF, JSP)
- Lets you define HTTP-specific classes
- Extends the capabilities of servers that host applications
- Accessed by way of request-response programming model (loose coupling)
- Can respond to any type of request, but commonly used to extend the applications hosted by web servers

JavaServer Faces Technology

- A user interface framework for building web applications
- Deitel and Deitel, Chapters 29/30
- Main Components:
  - A GUI component framework
  - A flexible model for rendering components in different kinds of HTML or different markup languages and technologies. A Renderer object generates the markup to render the component, and converts the data stored in a model object to types that can be represented in a view
- Features supporting GUI components:
  - Input validation
  - Event Handling
  - Data conversion between model objects and GUI components
  - Managed model object creation
  - Page navigation configuration
  - Expression Language (EL)

JavaServer Pages Technology

- JSP has been replaced by JSF, which is based on JSP, which is based on Servlets
- JavaServer Pages (JSP) technology lets you put snippets of servlet code directly into a text-based document. A JSP page is a text-based document that contains two types of text:
  - Static data, which can be expressed in any text-based format such as HTML or XML
  - JSP elements, which determine how the page constructs dynamic content

Java Persistence API (JPA)

- The Java Persistence API is a Java standards-based solution for persistence.
- Hibernate implements the JPA (the default implementation when you install JEE is called EclipseLink)
- Persistence uses an object/relational mapping approach to bridge the gap between an object-oriented model and a relational database.
- The Java Persistence API can also be used in Java SE applications, outside of the Java EE environment
- Java Persistence consists of the following areas:
  - The Java Persistence API
  - The query language
  - Object/relational mapping metadata

Java Transaction API

- JTA provides a standard interface for demarcating transactions
- The Java EE architecture provides a default auto commit to handle transaction commits and rollbacks
- An **auto commit** means that any other applications that are viewing data will see the updated data after each database read or write operation.
- When an application performs two separate database access operations that depend on each other, developers use the JTA API to demarcate where the entire transaction, including both operations, begins, rolls back, and commits

Java API for RESTful Web Services

- Representational State Transfer (REST) architectural style
- The JAX-RS API was introduced in Java EE 6 (now at JEE 8)

Named Beans

- lightweight container-managed objects (POJOs)
- The @Named (javax.inject.Named) annotation in a class, along with a scope annotation, automatically registers that class as a resource with the JavaServer Faces implementation

- Example ShoppingCart class:

```
@Named("cart")
@SessionScoped
public class ShoppingCart ... { ... }
```

Named bean scopes

- Application (javax.enterprise.context.ApplicationScoped): Application scope persists across all users' interactions with a web application.
- Session (javax.enterprise.context.SessionScoped): Session scope persists across multiple HTTP requests in a web application.
- Request (javax.enterprise.context.RequestScoped): Request scope persists during a single HTTP request in a web application.

GlassFish (http://glassfish.java.net/docs/index.html)

- The GlassFish Server is a compliant implementation of the Java EE 7 platform
- (supports all the aforementioned APIs, and more that weren't mentioned)
- Glassfish includes:
  - Administration Console: stop, manage users, resources, applications
  - asadmin: command-line admin utility equivalent of Console
  - appclient: launches the application client container and invokes an app in an application client jar file
  - capture-schema: command-line tool to extract schema information from a database
  - package-appclient : command-line tool to package application client container
  - Java DB database (Derby)
  - ..more

Glassfish Application Server

- Application Server (as-)
- as-install: e.g. C:\glassfish4\glassfish
  e.g. C:\Program Files\glassfish4\glassfish
- in config files: ${com.sun.aas.installRoot}
- Whenever you see "as-install" in commands on these slides, replace it with your own as-install (e.g. C:\glassfish4\glassfish)
  - as-install-parent in that example would be C:\glassfish4
- Starting/Stopping, Commandline
  - start: as-install/bin/asadmin start-domain [domain1]
  - stop: as-install/bin/asadmin stop-domain [domain1]
  - list: as-install/bin/asadmin list-domains

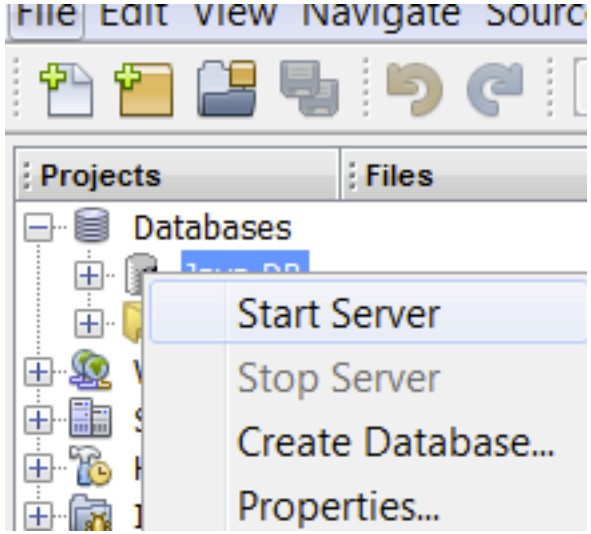
JEE and Databases

- We can set up a MySQL database as a datasource for Glassfish:
  - http://computingat40s.wordpress.com/how-to-setup-a-jdbc-connection-in-glassfish/
- Glassfish also comes with a built-in database engine called Derby

Derby

- Starting/Stopping the database:
  - as-install/bin/asadmin start-database --dbhome as-install-parent/javadb
    - example: asadmin start-database –dbhome C:\glasfish4\javadb
  - as-install/bin/asadmin stop-database

- Where is the database?
- Properties of Netbeans->Services->Databases->Java DB



Glassfish

- Admin Console of running server:
  - http://localhost:4848
- Deploying war files:
  - as-install/bin/asadmin deploy war-name (war-name=path of hello.war)
  - as-install/bin/asadmin list-applications
  - as-install/bin/asadmin undeploy appname (e.g. appname = hello)
- To Deploy automatically to a domain (e.g. domain1):
  - as-install/domains/domain1/autodeploy
  - any applications placed in the autodeploy directory will be deployed
- Launching the graphical update tool:
  - as-install-parent/bin/updatetool

Java EE Annotations

- What are Annotations?
- Java EE 5 and later support annotations in your source code, so that you can embed
  - resources, dependencies, services, and life-cycle notifications

    in your source code, without having to maintain these artifacts elsewhere in an xml file.
- An annotation is a modifier or Metadata tag that provides additional data to Java
  - classes, interfaces, constructors, methods, fields, parameters, and local variables.
- Annotations
  - Replace XML descriptors for most purposes
  - Remove the need for marker interfaces (like java.rmi.Remote)
  - Allow application settings to be visible in the component they affect

Finding out about Annotations

- The JEE Tutorial examples use and explain the annotations we'll need.
- For more depth -- JSRs: Java Specification Requests
- JSR 342: Java™ Platform, Enterprise Edition 7 (Java EE 7) Specification athttp://jcp.org/en/jsr/detail?id=342

- JSR 250: Common annotations
- JSR-220: EJB 3.0 annotations
- JSR-181: Web Services annotations
- JSR-220: Java Persistence API annotations
- JSR-222: JaxB annotations (Java arch for XML binding)
- JSR-224: WS2 annotations (XML-based web services)

Servlets

- Servlets are server-side programming technology for dynamic content
- A Servlet extends the capability of the server in the request-response programming model
- javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets
- All servlets must implement the Servlet interface, which defines lifecycle methods

Servlet Lifecycle

- Servlet Lifecycle is controlled by the container in which the servlet has been deployed
- When a request is mapped to a servlet, the container performs the following:
  - If an instance of the servlet does not exist, the web container
    1. Loads the servlet class.
    2. Creates an instance of the servlet class.
    3. Initializes the servlet instance by calling the init method.
  - Invokes the service method, passing request and response objects.
- If it needs to remove the servlet, the container finalizes the servlet by calling the servlet's destroy method

Creating and Initializing a Servlet

- Use the @WebServlet annotation to define a servlet component in a web application
- The annotated servlet must specify at least one URL pattern
- This is done by using the urlPatterns or value attribute on the annotation
- Classes annotated with @WebServlet must extend the javax.servlet.http.HttpServlet class
- Example (mood):

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
@WebServlet("/report")
public class MoodServlet extends HttpServlet {
   …
```

Creating and Initializing (cont'd)

- The web container initializes a servlet after loading and instantiating the servlet class and before delivering requests from clients
- To customize this process, you can either override the init method of the Servlet interface or specify the initParams attribute of the @WebServlet annotation (see Mood filter, Programming Filters Slide below) to
  - read persistent configuration data
  - Initialize resources
  - Any other one-time activities
- The initParams attribute contains a @WebInitParam annotation
- If it cannot complete its initialization process, a servlet throws an UnavailableException

Maintaining Client State

- Many applications require that a series of requests from a client be associated with one another (e.g. state of shopping cart)
- Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions
- Sessions are represented by an HttpSession object
- You access a session by calling the getSession method of a request object

The Mood Example

- The example shows how to develop a simple application by using the @WebServlet, @WebFilter, and @WebListener annotations to create a servlet, a listener, and a filter.
- The mood example application is comprised of three components:
  - mood.web.MoodServlet
  - mood.web.TimeOfDayFilter
  - mood.web.SimpleServletListener
- Operation:
  - Request comes in
  - TimeOfDayFilter sets a mood attribute in the request
  - MoodServlet then processes the request
  - All the while, SimpleServletListener is responding to events by logging to the server's log

MoodServlet

- MoodServlet, the presentation layer of the application, displays Duke's mood in a graphic, based on the time of day.
- The @WebServlet annotation specifies the URL pattern:

```
@WebServlet("/report")
public class MoodServlet extends HttpServlet { ...
```

TimeOfDayFilter

- TimeOfDayFilter sets an initialization parameter indicating that Duke is awake:

```
@WebFilter(filterName = "TimeOfDayFilter",
 urlPatterns = {"/*"},
 initParams = {
        @WebInitParam(name = "mood", value = "awake")})
public class TimeOfDayFilter implements Filter { ...
```

- The filter calls the doFilter method, which contains a switch statement that sets Duke's mood based on the current time.
- SimpleServletListener logs changes in the servlet's lifecycle. The log entries appear in the server log.

Hello1 example

- Example of JavaServer Faces technology
- index.html is the default landing page for a Facelets application
- For this application, the page uses simple tag markup to display a form with a graphic image, a header, a text field, and two command buttons
- Notice the Submit commandButton element specifies the action as response, meaning that when the button is clicked, the response.xhtml page is displayed
- Hello object is a named bean
- The web.xml file contains several elements that are required for a Facelets application. All these are created automatically when you use NetBeans

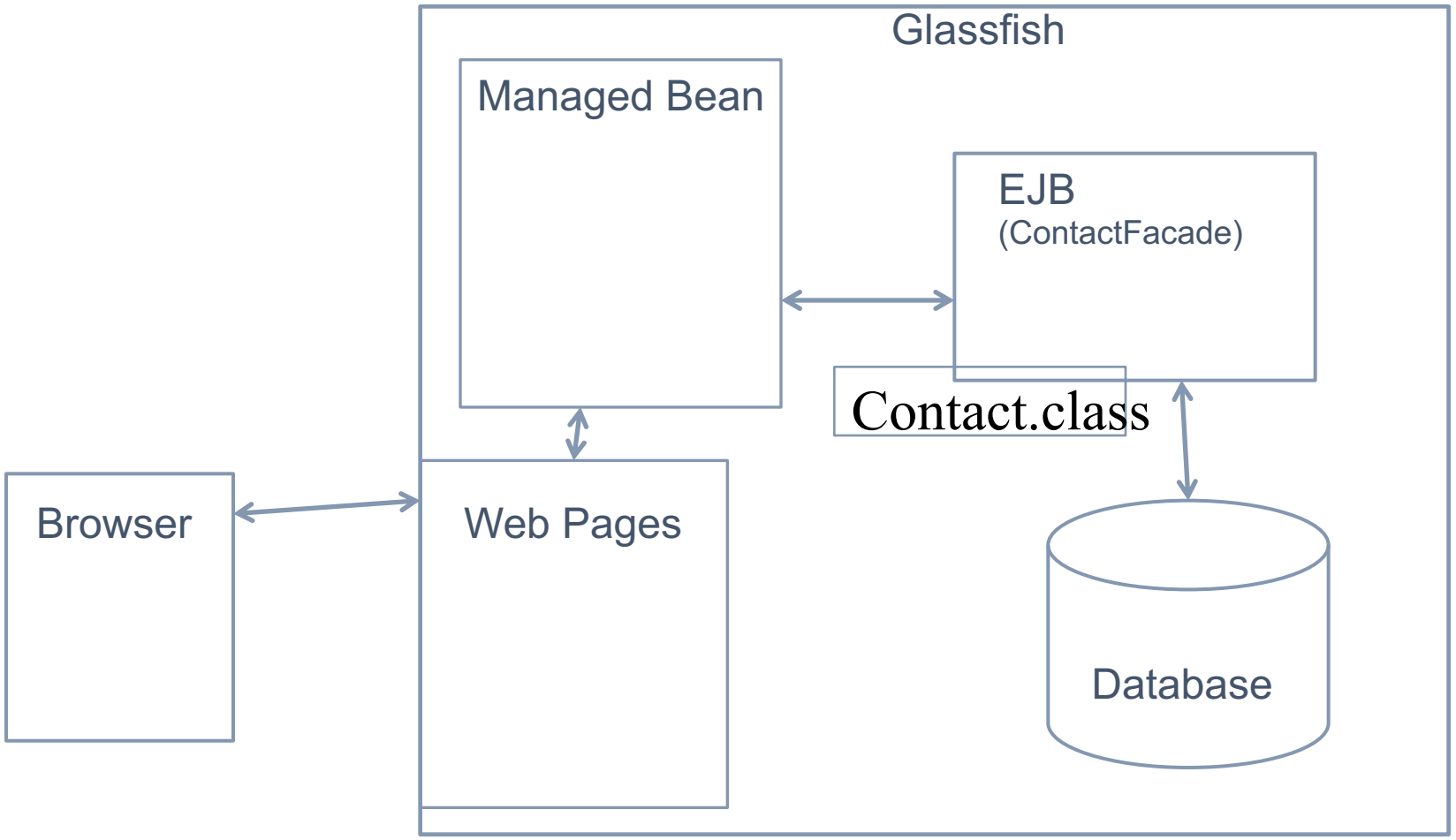
Hello2 example

- The hello2 application behaves almost identically to the hello1 application, but it is implemented using Java Servlet technology instead of JavaServer Faces technology
- This servlet overrides the doGet method, implementing the GET method of HTTP

Simple Persistence Example

- We will study the addressbook JSF example, and refer to it when we create our own application
- To create this kind of application from scratch:
- http://netbeans.org/kb/docs/javaee/javaee-gettingstarted.html
- Simple Java EE 7 web application that contains an EJB 3.1 stateless session bean façade for an entity class (just like address-book)
- use the IDE to generate the entity class and session bean
- The code generated uses queries defined in the Criteria API part of Java Persistence API 2.0
- create a named managed bean that accesses the session façade and
- create a presentation layer that uses the Facelets view framework as specified in JSF 2.0

Addressbook application



AddressBook mechanisms

- Entity Class: Contact.java
  - contains annotations for ORM
  - Contact table created in Derby database engine
- Stateless Session Bean: ContactFacade.java
  - database session, not user's session
  - interacts with and queries the database
- Named Managed Bean: ContactController.java
  - session scoped (the user's session)
  - connects the presentation layer (JSF) with the EJB
  - Obtains an instance of ContactFacade through dependency injection
    - @EJB private ContactFacade ejbFacade;
  - Example: prepareEdit and edit methods
    - when the user clicks the "Edit" button, it calls the prepareEdit function, which gets the data through the ContactFacade and forwards to the Edit view, a JSF page
    - user enters data and clicks Submit, EL calls the update method, saving updated data through the ContactFacade

AddressBook Cont'd

- JSF presentation layer: xhtml pages that contain Expression Language expressions used to communicate with the Named bean
- Expression Language
  - #{bean.method}
  - EL example:

  #{customer.name}

  When reading, this would be the result of the **getName** method of the named bean called "customer"

  When writing, this would call **setName** method of the bean called "customer"
  - https://docs.oracle.com/javaee/7/tutorial/jsf-el.htm#GJDDD
- Uses JSF tag libraries:
  - xmlns:h="http://xmlns.jcp.org/jsf/html"
  - xmlns:f="http://xmlns.jcp.org/jsf/core">

Derby

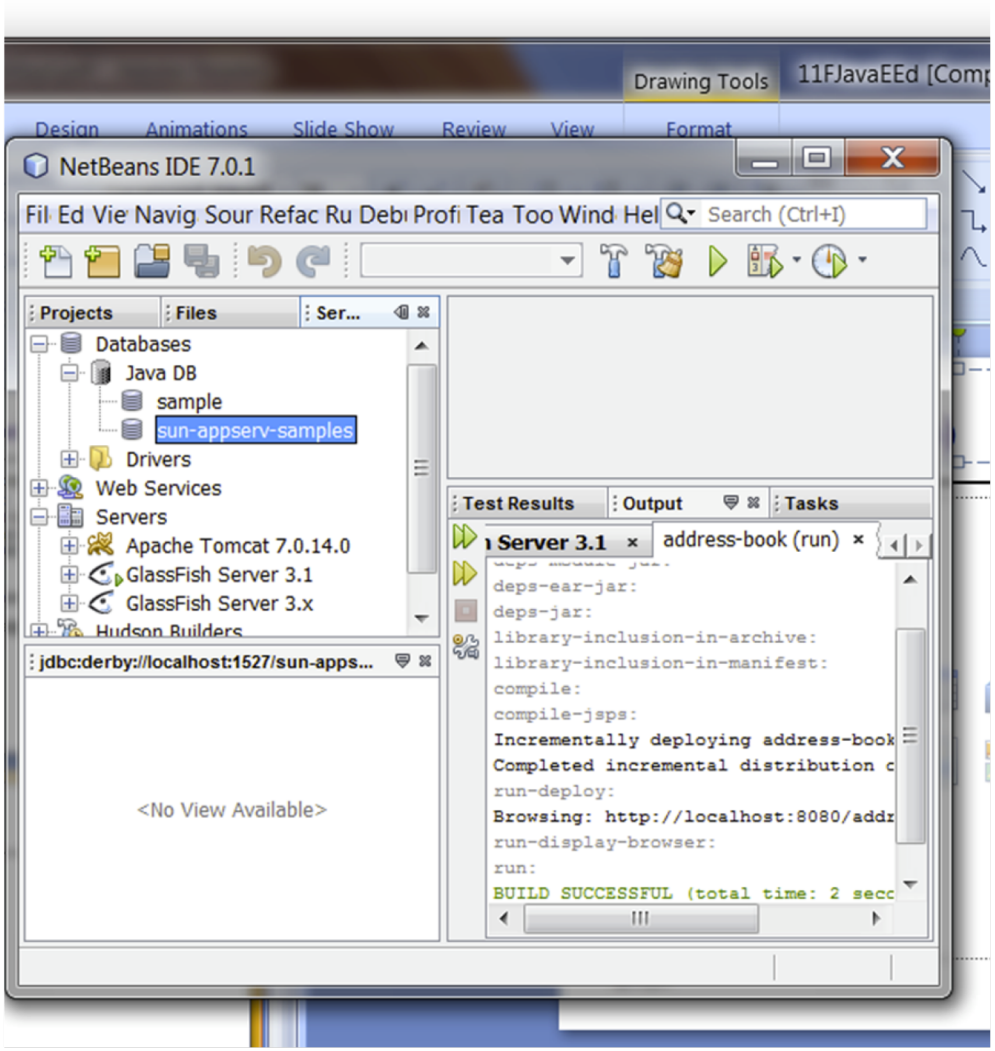
- creating a database
  - rightclick on JavaDB in Netbeans

  or
  - ;create=true; in connect statement in commandline ij

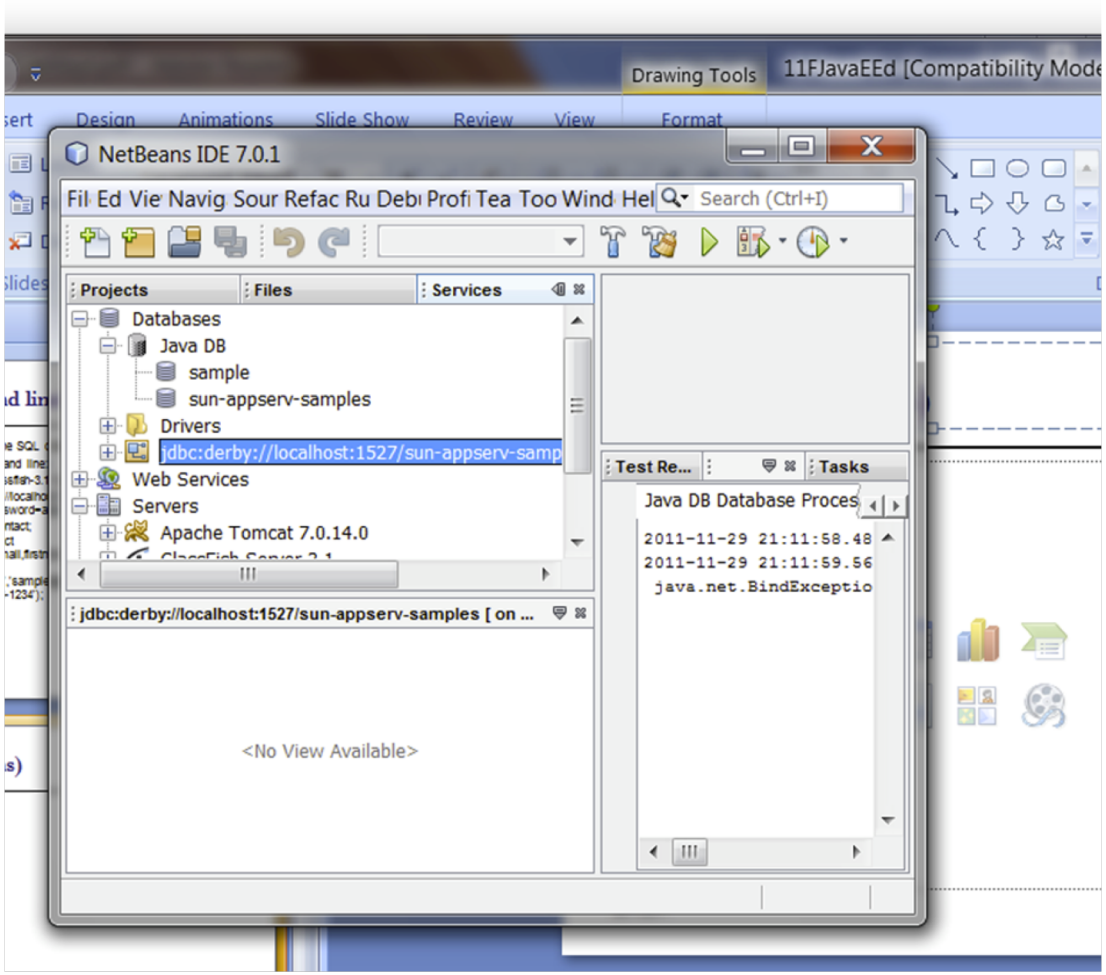
Derby (command line)

- ij is Derby's command line SQL client
- run from windows command line:
  - cd c:\program files\glassfish-3.1.1\javadb\bin\ij
  - ij> connect 'jdbc:derby://localhost:1527/sun-appserv-samples;user=app;password=app';
  - ij> select * from app.contact;
  - ij> insert into app.contact (id,Birthcity,birthday,email,firstname,homephone,lastname,mobilephone) values (7,'Ottawa','10/10/2000','sample@algonquincollege.com','Bubbie','123-123-1234','Lu','123-123-1234');

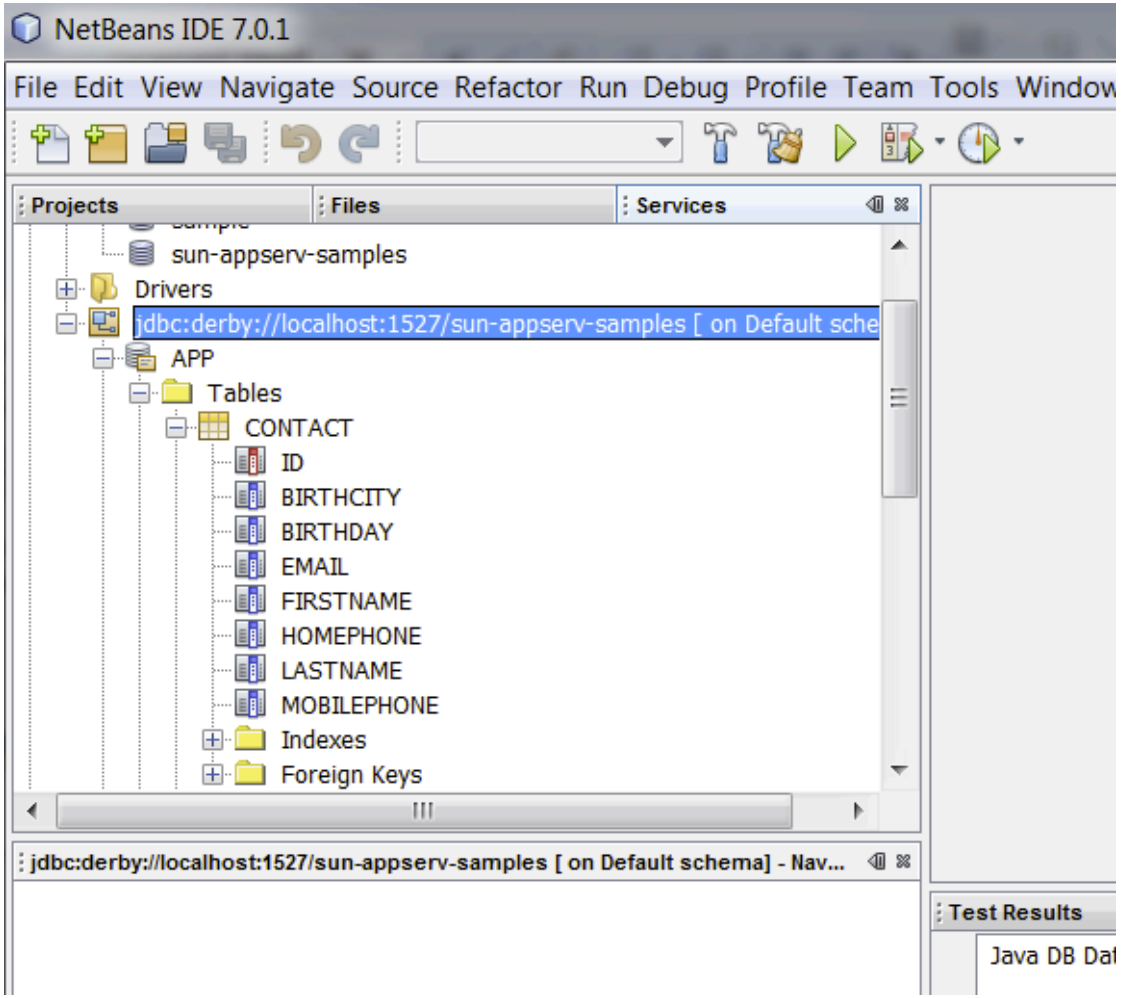
Derby (Netbeans)



Derby (Netbeans)



Derby (Netbeans)



JPA (Java Persistence API)

- Both Hibernate (a popular ORM framework) and EclipseLink implement the JPA -- annotations supported can be different, with different options
- Hibernate can be used without JEE: The overall creation and management of persistent objects is basically the same (annotate the Entity class(es), carry out transactions)
- In both we annotate the entity class (examples: Employee, Contact)

without JEE versus with JEE

- Hibernate without JEE:
  - load configuration (**hibernate.cfg.xml**), create session factory (once)
  - (as needed) **getCurrentSession**, **beginTransaction**, do work, **commit**
- with JEE:
  - **persistence.xml** specifies a **persistence-unit** which specifies a **jta-data-source**
  - Netbeans creates the ***Entity*Facade** for us where ***Entity*** is the name of the entity class (example, **ContactFacade**)
  - **@PersistenceContext** annotation on the **EntityManager** in the ***Entity*Facade** ties the **EntityManager** to the **persistence-unit**
  - The bean that deals with persistent objects (example **ContactController**) obtains a reference to the **ContactFacade** through Dependency Injection: **@EJB private ContactFacade ejbFacade;**

What's this *Entity*Facade?

- The **ContactFacade** is the DAO (Data Access Object) for the **Contact** Entity (a **Contact** is a persistent object)
- A DAO is the go-to object for doing CRUD operations on an Entity
  - Create
  - Read
  - Update
  - Delete

- Sessions and Transactions of the **EntityManager** in the **ContactFacade** are handled for us by the **JTA** (see persistence.xml):

```
<persistence-unit name="address-bookPU" transaction-type="JTA">
   <jta-data-source>MySQLDB</jta-data-source>
```

User to Database (address-book app)

1. User's Browser submits the form to Glassfish when the user clicks **save**
2. JSF pages are implemented with JSP and Servlets behind the scenes
3. EL (Expression Language) expressions use the getters and setters on the properties of the Named Bean

   example from Create.xhtml

```
<td><h:inputText id="firstName"
        value="#{contactController.selected.firstName}"
        title="#{bundle.CreateContactTitle_firstName}" />
</td>
```

   to display the value, on the ContactController object, call getSelected(), then on that, call getFirstName()

   to update the value upon submission, call getSelected(), then on that, call setFirstName()

   the bundle.CreateContactTitle_firstName is the string that should label the field on the User's webpage (this is Internationalization support – the string can be English, French, Chinese, etc)

User to Database (cont'd)

4. EL expression on save button:

   <h:commandLink action="#{contactController.create}"

   value="#{bundle.CreateContactSaveLink}" />

   contactController object create() method gets called

5. **create()** calls **getFacade().create(current);**
6. The Facade's create method calls **getEntityManager().persist(entity);**

   analogous to **session.save(entity)** with Hibernate

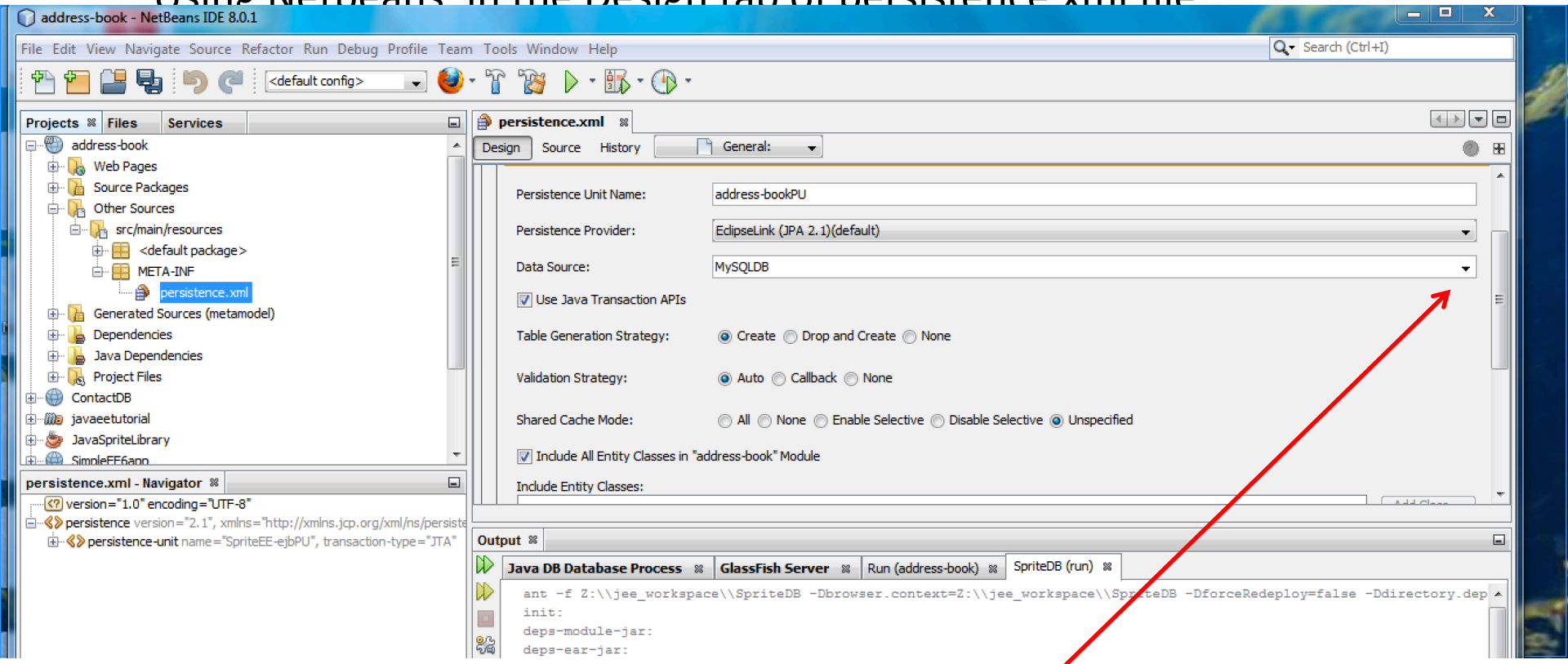
5. JTA carries out the transaction to update the database

For you **TODO**:

look at the AddressBook application and trace through this process yourself (follow the method calls)

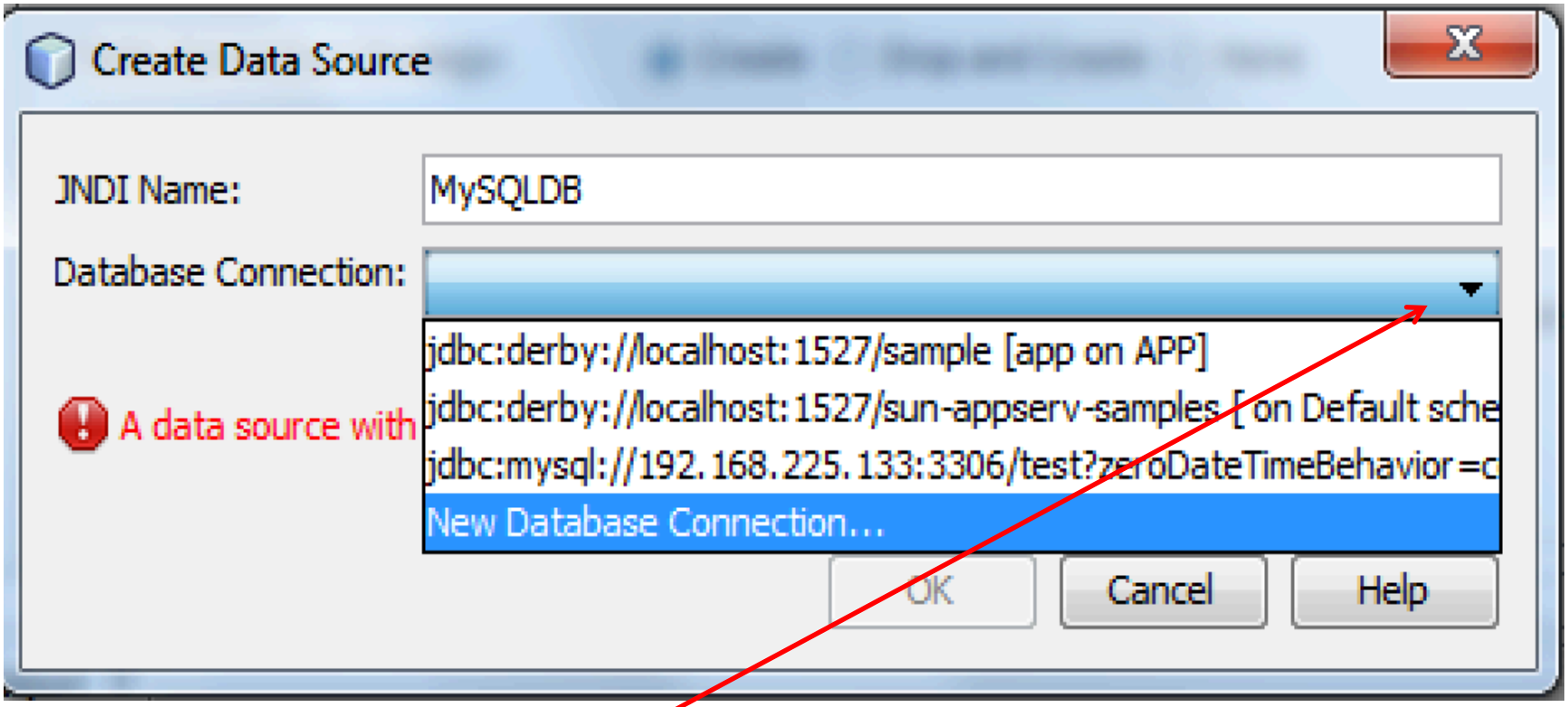
Adding a MySQL datasource

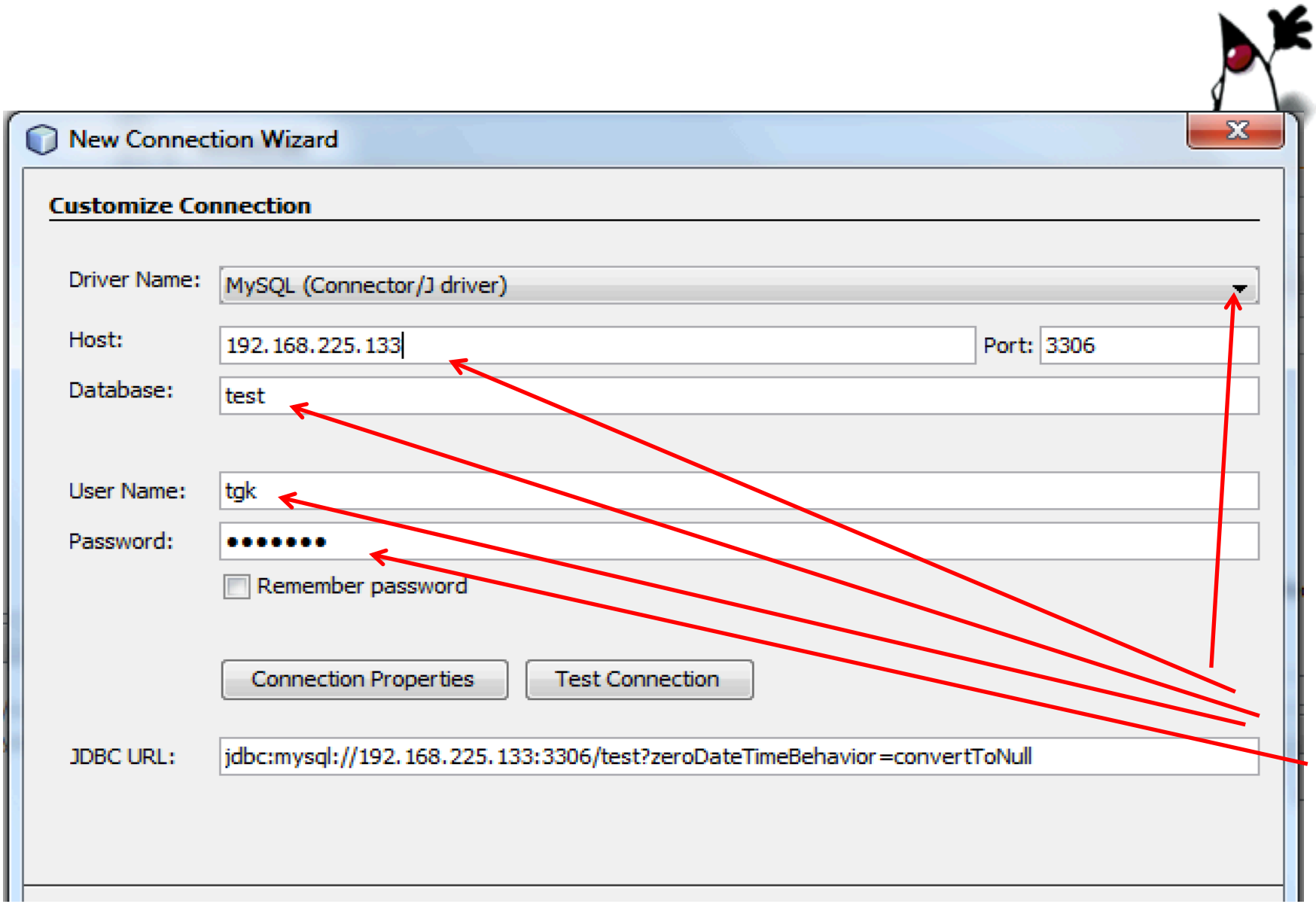
- Using Netbeans, in the Design tab of persistence.xml file



Adding a datasource (MySQL)

Select "New Data Source..." to get this window:




New Connection Wizard
Customize Connection
Driver Name:
MySQL (Connector/J driver)
Host:
192.168.225.133
Port:
3306
Database:
test
User Name:
tgk
Password:
Remember password
Connection Properties
Test Connection
JDBC URL:
jdbc:mysql://192.168.225.133:3306/test?zeroDateTimeBehavior=convertToNull