CS/INFO 330
Java EE Technology

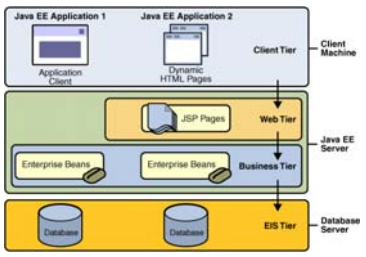
Mirek Riedewald
mirek@cs.cornell.edu

(Based on Sun's Java EE Tutorial)

Resources

- Java EE tutorial
  - http://java.sun.com/javaee/5/docs/tutorial/doc/
- Java EE SDK API
  - http://java.sun.com/javaee/5/docs/api/

Java EE Application Model



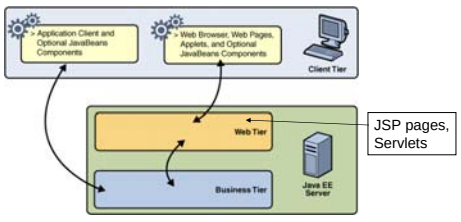
Java EE Components

- Client
  - Application client, applets
- Server
  - Web components: Java Servlet, JavaServer Faces, JavaServer Pages (JSP)
  - Business components: Enterprise JavaBeans (EJB)
- Written in Java, compiled like other Java code, but:
  - Assembled into a Java EE application
  - Verified to be well-formed and Java EE spec compliant
  - Deployed to production: run and managed by Java EE server

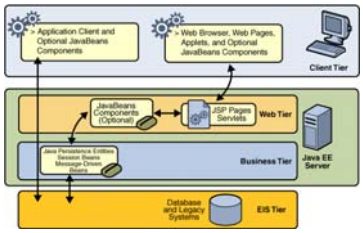
Java EE Web Client

- Dynamic web pages containing markup
  - HTML, XML
  - Generated by web components running in the web tier
- Web browser
  - Renders pages received from the server
- Thin client: offloads heavyweight operations like DB access and execution of business logic to enterprise beans executing on the server

Client-Server Communication



Enterprise Information System



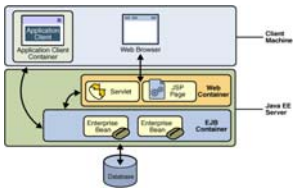
Java EE Containers

- Organize business logic into reusable components
- Server provides services in form of a container for every component type

- Container = interface between component and low-level functionality
  - Before component can be executed, it must be assembled into a Java EE module and deployed into its container
  - Can specify container settings for each component for assembly
    - Security model
    - Transaction model
    - JNDI lookup services
    - Remote connectivity model

Container Types

- Java EE server: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.
- Enterprise JavaBeans (EJB) container: Manages the execution of enterprise beans for Java EE applications.
- Web container: Manages the execution of JSP page and servlet components for Java EE applications.
- Application client container: Manages the execution of application client components.
- Applet container: Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

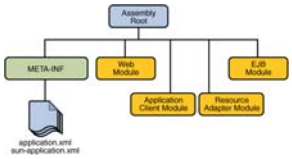


Web Services

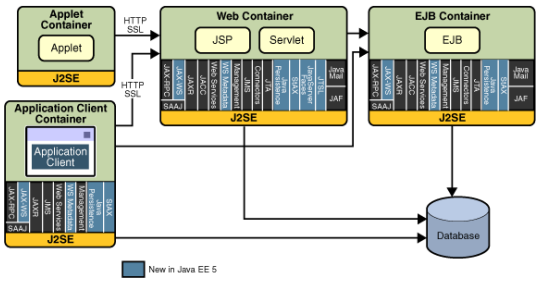
- Java EE XML APIs
  - No low-level programming needed
- SOAP protocol for transporting client requests and web service responses
- WSDL format for describing network services
- UDDI and ebXML to publish information about products and web services

Packaging Applications

- Java EE application is delivered in an Enterprise Archive (EAR) file (similar to JAR)
  - Deployment descriptor: deployment settings of application, module, or component
  - EJB modules: class files for enterprise beans and EJB deployment descriptor
  - Web modules: servlet class files, JSP files, supporting class files, GIF and HTML files, and web application deployment descriptor
  - Application client modules: class files and application client deployment descriptor
  - Resource adapter modules: all Java interfaces, classes, native libraries, and other documentation, along with the resource adapter deployment descriptor



Java EE 5 APIs



EJB

- Implements modules of business logic
  - Used alone or with other EJBs
- Session bean
  - Represents transient conversation with client
  - When client finishes executing, session bean and its data are gone
- Message-driven bean
  - Receive messages, typically Java Message Service (JMS) messages, asynchronously
- (Persistent) entity beans are replaced by Java persistence API entities

Other APIs

- Java Servlet
  - Extends capabilities of servers that use the request-response programming model
- JavaServer Pages (JSP)
  - Lets you add servlet code directly into a text-based document
- JavaServer Faces
  - User interface framework
  - GUI components, rendering components
- Java Message Service API (JMS)
  - Messaging standard for loosely coupled, reliable, and asynchronous message exchange
- Java Transaction API
  - Standard interface for demarcating transactions

More APIs

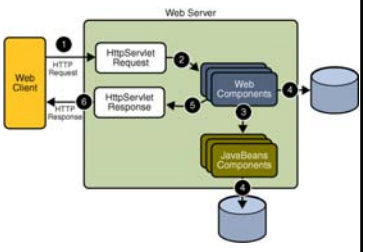
- JavaMail API
  - For sending email notifications
- JavaBeans Activation Framework (JAF)
  - Used by JavaMail
- Java API for XML Processing (JAXP)
  - Parse and transform XML documents: DOM, SAX, XSLT
- Java API for XML Web Services (JAX-WS)
- Java Architecture for XML binding (JAXB)
- Java API for XML Registries (JAXR)
  - For resource discovery

Even More APIs

- Java Database Connectivity API (JDBC)
  - Invoke SQL commands from Java programs
  - Used in enterprise bean when session bean has access to the database or directly from servlet or JSP page
- Java Persistence API
- Java Naming and Directory Interface (JNDI)
  - Enables access to various naming and directory services
- Java Authentication and Authorization Service (JAAS)

Web Applications

- Client sends HTTP request
- Web server that implements Java Servlet and JavaServer Pages technology converts request into an HTTPServletRequest object
- Object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content
- Web component can then generate an HTTPServletResponse or pass request to another web component
- Eventually a web component generates a HTTPServletResponse object, which is converted to an HTTP response and returned to the client

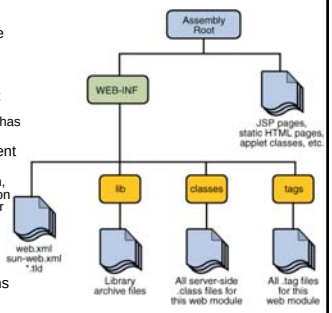


Web Application Life Cycle

1. Develop the web component code
2. Develop the web application deployment descriptor
3. Compile the web application components and helper classes referenced by the components
4. Optionally package the application into a deployable unit
5. Deploy the application into a web container
6. Access a URL that references the web application

Web Modules

- Smallest deployable and usable unit of web resources
- web.xml: Web application deployment descriptor
  - Not needed if module does not contain any servlets, filter, or listener components (i.e., only has JSP pages and static files)
- sun-web.xml: runtime deployment descriptor
  - Context root of web application, mapping of names of application resources to Application Server resources
- Can create application-specific subdirectories
- Portable: Can be deployed into any web container that conforms to the Java Servlet spec



Java Servlets

- Generate dynamic content on server side
- Main advantages over CGI: portability, scalability
- Definition: Servlet = Java programming language class
  - Extends capabilities of servers for request-response type applications
  - Can respond to any type of request, but commonly used to extend applications hosted by web servers
    - Defines HTTP-specific servlet classes
    - javax.servlet, javax.servlet.http
- Must implement Servlet interface
- HttpServlet class provides methods for handling HTTP-specific services

Servlet Examples

- Look at example servlets in Java EE 5 tutorial; they show
  - Handling of HTTP GET requests
  - Construction of responses
  - Tracking of session information

Servlet Life Cycle

- Controlled by container in which servlet has been deployed
- When request is mapped to a servlet, container performs several steps
  - Initialize servlet if it does not exist yet
    - Loads servlet class and creates class instance
    - Initializes servlet instance by calling init()
  - Invoke service() method, passing request and response objects
- Container removes servlet by calling its destroy() method

Handling Life Cycle Events

- You can monitor and react to servlet life cycle events by defining listener objects, e.g.:
  - Initialization, destruction: javax.servlet.ServletContextListener and javax.servlet.ServletContextEvent
  - Servlet request started: javax.servlet.ServletRequestListener and javax.servlet.ServletRequestEvent
- Specified in listener element of deployment descriptor (web.xml file)

Servlet Errors

- Web container generates default page when servlet exception occurs
- Can return more specific error messages
  - See bookstore example in tutorial

Information Sharing

- Option 1: Through scope objects
  - Web context (javax.servlet.ServletContext): accessible from web components within a web context
  - Session (javax.servlet.http.HttpSession): accessible from web components handling a request that belongs to a session
  - Request (javax.servlet.ServletRequest): accessible from web components handling the request
  - Page (javax.servlet.jsp.JspContext): accessible from JSP page that creates the object
- Challenge: might have to handle concurrent access to shared resources
  - Use proper synchronization techniques for multi-threaded programs

Example: Session Scope Object

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
  // Get the user's session and shopping cart
  HttpSession session = request.getSession(true);
  ResourceBundle messages = (ResourceBundle)
    session.getAttribute("messages");

  ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");

  // If the user has no cart, create a new one
  if (cart == null) {
    cart = new ShoppingCart();
    session.setAttribute("cart", cart);
}
```

Access session attribute value

Set session attribute value

Example: Web Context Scope Object

- After loading/instantiation of servlet class, web container initializes the servlet
  - During initialization, servlet can read persistent configuration data, initialize resources, etc.
  - Example: initialize variable that points to the DB access object created by the web context listener

```
public class CatalogServlet extends HttpServlet {
  private BookDBAO bookDB;

  public void init() throws ServletException {
    bookDB = (BookDBAO) getServletContext().getAttribute("bookDB");
    if (bookDB == null) throw new UnavailableException("Couldn't get database.");
  }
}
```

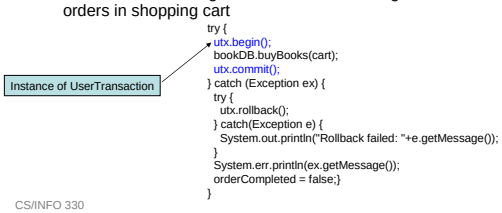
Synchronization Example

- Hit counter or order counter from bookstore example:

```
public class Counter {
  private int counter;
  public Counter() {
    counter = 0;
  }
  public synchronized int getCounter() {
    return counter;
  }
  public synchronized int setCounter(int c) {
    counter = c;
    return counter;
  }
  public synchronized int incCounter() {
    return(++counter);
  }
}
```


Information Sharing (contd.)

- Option 2: Through database access
  - For shared data that is persistent between web application invocations
  - DB access through Java Persistence API
  - Use transactions, e.g., to ensure all-or-nothing execution of orders in shopping cart




```
try {
utx.begin();
 bookDB.buyBooks(cart);
 utx.commit();
} catch (Exception ex) {
 try {
   utx.rollback();
 } catch(Exception e) {
   System.out.println("Rollback failed: "+e.getMessage());
 }
 System.err.println(ex.getMessage());
 orderCompleted = false;}
}
```


Servlet Service Methods

- Handles the service request
  - GenericServlet.service(ServletRequest, ServletResponse) method
  - HttpServlet.do*XYZ*(ServletRequest, ServletResponse), where *XYZ* is Get, Post, or other HTTP request
- General pattern
  - Extract information from request
  - Access external resources, e.g., database
  - Populate response based on that information

Extracting Request Information

- Methods in ServletRequest to get *XYZ*: get*XYZ*
  - E.g., identifier of book that customer wants to purchase is included as parameter to the request: String bookId = request.getParameter("bookId");
- HttpServletRequest contains HTTP request object
  - Specific functionality: getCookies, getMethod (e.g., GET or POST), getQueryString (from URL), getSession

Constructing Responses

- Implements ServletResponse interface
  - Retrieve output stream for sending data to client (getWriter, getOutputStream methods)
  - Indicate content type (setContentType method)
  - Indicate whether to buffer output (setBufferSize method)
  - Set localization information, e.g., locale and character encoding
- HttpServletResponse additional functionality
  - HTTP header fields, cookies can be added

Example Response Code

```
public class BookDetailsServlet extends HttpServlet {
  ...
  public void doGet (HttpServletRequest request,
HttpServletResponse response) throws
ServletException, IOException {
      ...
    // set headers before accessing the Writer
    response.setContentType("text/html");
    response.setBufferSize(8192);
    PrintWriter out = response.getWriter();

    // then write the response
    out.println("<html>" + "<head><title>+
messages.getString("TitleBookDescription") +
</title></head>");

    // Get the dispatcher; it gets the banner to the user
    RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/banner");
    if (dispatcher != null)
      dispatcher.include(request, response);

    // Get the identifier of the book to display
    String bookId = request.getParameter("bookId");
    if (bookId != null) {
      // and the information about the book
      try {
        Book bd = bookDB.getBook(bookId);
            ...
        // Print the information obtained
        out.println("<h2>" + bd.getTitle() + "</h2>" +
            ...
      } catch (BookNotFoundException ex) {
        response.resetBuffer();
        throw new ServletException(ex);
      }
    }
    out.println("</body></html>");
    out.close();
  }
}
```

Filtering Requests and Responses

- Filter can transform header and content
  - Does not create response, but modifies it
  - Can add attribute to request or insert data (e.g., a visitor counter) in response
- Main tasks:
  - Query the request and act accordingly
  - Block request-response pair from passing any further
  - Modify request headers and data (customized request)
  - Modify response headers and data (customized response)
  - Interact with external resources
- Examples: authentication, logging, image conversion, data compression, encryption, tokenizing streams, XML transformations

Programming Filters

- Filter, FilterChain, FilterConfig interfaces in javax.servlet package
- Use doFilter method for filter functionality
- Usual init() and destroy() methods
  - Can pass initialization parameters to the filter through the FilterConfig object
- Look at Java EE tutorial examples

Maintaining Client state

- Example: shopping cart
- API for managing sessions
  - HttpSession object
- Use session attributes

```
public class CashierServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Get the user's session and shopping cart
      HttpSession session = request.getSession();
      ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");
       ...
      // Determine the total price of the user's books
      double total = cart.getTotal();
```

Session Management

- HTTP client cannot terminate session
  - Each session has associated timeout
  - Timeout can be set in web.xml file
  - “Last” servlet needs to invalidate session

```
public class ReceiptServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      // Get the user's session and shopping cart
      HttpSession session = request.getSession();
      // Payment received -- invalidate the session
      session.invalidate();
      ...
```


Session Tracking

- Pass identifier between client and server
  - Cookie or as part of URL
- ID in URL when cookies disabled: call response’s encodeURL method
  - Example from doGet method of ShowCartServlet:

```
out.println("<p>   <p><strong><a href=\"" +
   response.encodeURL(request.getContextPath() + "/bookcatalog") +
      "\">" + messages.getString("ContinueShopping") +
      "</a>      " + "<a href=\"" +
   response.encodeURL(request.getContextPath() + "/bookcashier") +
      "\">" + messages.getString("Checkout") +
      "</a>      " + "<a href=\"" +
    response.encodeURL(request.getContextPath() + "/bookshowcart?Clear=clear") +
      "\">" + messages.getString("ClearCart") + "</a></strong>");
```


Finalizing a Servlet

- Container calls destroy() method
  - Release all resources
  - Save persistent state
- Make sure any threads still handling client requests have completed before shutdown
  - Might have to use counter for number of running service methods for a servlet
  - Notify long-running methods to shut down
  - Make long-running methods behave “politely”
    - Need to check field that notifies them about shutdown
- Look at tutorial for examples

Enterprise Bean

- Server side component that encapsulates business logic
- Runs in EJB container
  - Container provides system-level services like transaction management and security authorization
- Forms the core of transactional Java EE applications
- Bean types:
  - Session bean
  - Message-driven bean

Session Bean

- Represents single client inside the Application Server
  - Client invokes session bean's methods
  - Not shared between clients
  - Not persistent
- Stateful session bean
  - State retained for duration of client-bean session
- Stateless session bean
  - No client-specific state retained after bean method finishes

Message-Driven Bean

- For asynchronous message processing
- Acts as JMS (Java Message Service) message listener
  - Messages from any Java EE component (app client, another enterprise bean, web component), JMS app, or even non Java-EE system
- Client components accesses bean through sending of message to destination for which bean is MessageListener
  - Think of triggers in a database system

Defining Client Access to Beans

- Only for session beans…
- Remote client
  - @Remote annotation of business interface and bean class
  - Typically used for beans accessed by app client and to allow distributed processing
- Local Client
  - Optional @Local annotation of business interface and bean class
  - Typically used for tightly coupled beans
    - E.g., bean for processing sales orders that calls bean that emails confirmation message
- If in doubt, choose remote

Other Bean Info in Tutorial

- Contents
- Naming conventions: important, because bean consists of multiple parts
- Different life cycles for different bean types
- Check out examples in tutorial
  - Stateless session bean for currency conversion (uses JSP)
  - Stateful session bean for cart, accessed by remote client
  - Stateless session bean that sets a timer
  - Message-driven bean (read this if you plan to use asynchronous messaging functionality)

Java Persistence API

- Entity typically represents table in DBMS
  - Entity instance = row in table
  - Has mapping of persistent fields and properties of entity to relational data in the DBMS
- See tutorial for possible data types for storing persistent state
- Method signatures for single-valued properties:
  - Type getProperty()
  - void setProperty(Type type)
- Method signatures for collection-valued fields and properties, e.g., if customer has property that contains a set of phone numbers:
  - Set<PhoneNumber> getPhoneNumbers() {}
  - void setPhoneNumbers(Set<PhoneNumber>) {}

Enforcing Constraints

- Can specify properties like in relational database
  - Primary key
  - Multiplicity in entity relationships
    - One-to-one, one-to-many, etc.
  - Referential integrity
    - Cascading deletion
  - Entity inheritance
- Good news: Can automatically create entity classes from existing database

Managing Entities

- Managed by entity manager
  - Instance of javax.persistence.EntityManager
- EntityManager instance is associated with persistence context
  - Set of managed entity instances in particular data store
  - EntityManager API supports database query and update functionality

More Info About Java Persistence

- Persistence unit is configured by descriptor file persistence.xml
- DataSource object represents real data source
  - Created in application server
- Look at bookstore example in Chapter 25
  - Book class for accessing book table

Book Class

```
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="WEB_BOOKSTORE_BOOKS")

public class Book implements Serializable {

   private String bookId;
   private String title;

   public Book() { }

   public Book(String bookId, String title, ...) {
      this.bookId = bookId;
      this.title = title;
      ...
   }

   @Id
   public String getBookId() {
      return this.bookId;
   }

   public String getTitle() {
      return this.title;
   }
   ...

   public void setBookId(String id) {
      this.bookId=id;
   }

   public void setTitle(String title) {
      this.title=title;
   }
   ...
}
```

Declare it as entity

Name of database table

Primary key

Accessing An Entity Manager

- Need to obtain EntityManager instance to access database
  - Use @PersistenceUnit annotation to inject EntityManagerFactory into object
  - Can obtain EntityManager instance from factory
  - Can only be used with classes that are managed by Java EE compliant container
    - Cannot inject into web container
      - Can do it by injection into object managed by the container, in particular servlet and ServletContextListener objects

Example for Web Tier

```
public final class ContextListener implements ServletContextListener {
...
@PersistenceUnit
private EntityManagerFactory emf;

public void contextInitialized(ServletContexEvent event) {
   context = event.getServletContext();
   ...
   try {
      BookDBAO bookDB = new BookDBAO(emf);
      context.setAttribute("bookDB", bookDB);
   } catch (Exception ex) {
      System.out.println("Couldn't create bookstore database bean: " + ex.getMessage());
   }}}
```

BookDBAO object can then obtain EntityManager from the factory:

```
private EntityManager em;

public BookDBAO (EntityManagerFactory emf) throws Exception {
   em = emf.getEntityManager();
   ...
}
```

Access From Managed Beans

- Managed bean allows resource injection
  - Do not need EntityManagerFactory
  - Thread safety not an issue here (but would be when working with servlets and listeners, therefore they need to inject thread-safe factory)
  - Simpler code, e.g., BookDBAO object as managed bean:

```
import javax.ejb.*;
import javax.persistence.*;
import javax.transaction.NotSupportedException;
public class BookDBAO {

   @PersistenceContext
   private EntityManager em;
...
```


Database Queries

```
public List getBooks() throws BooksNotFoundException {
   try {
      return em.createQuery(
         "SELECT bd FROM Book bd ORDER BY bd.bookId").getResultList();
   } catch(Exception ex){
      throw new BooksNotFoundException("Could not get books: " + ex.getMessage());
   }
}
```

```
public Book getBook(String bookId) throws BookNotFoundException {
    Book requestedBook = em.find(Book.class, bookId);
    if (requestedBook == null) {
       throw new BookNotFoundException("Couldn't find book: " + bookId);
    }
    return requestedBook;
}
```


Database Updates

```
public void buyBooks(ShoppingCart cart) throws
OrderException{
   Collection items = cart.getItems();
   Iterator i = items.iterator();
   try {
      while (i.hasNext()) {
         ShoppingCartItem sci =
(ShoppingCartItem)i.next();
         Book bd = (Book)sci.getItem();
         String id = bd.getBookId();
         int quantity = sci.getQuantity();
         buyBook(id, quantity);
      }
   } catch (Exception ex) {
      throw new OrderException("Commit failed: "
         + ex.getMessage());
   }
}
```

```
public void buyBook(String bookId, int quantity)
    throws OrderException {
   try {
      Book requestedBook = em.find(Book.class,
bookId);
      if (requestedBook != null) {
         int inventory = requestedBook.getInventory();
         if ((inventory - quantity) >= 0) {
            int newInventory = inventory - quantity;
            requestedBook.setInventory(newInventory);
         } else{
            throw new OrderException("Not enough of "
               + bookId + " in stock to complete
order.");
         }
      }
   } catch (Exception ex) {
      throw new OrderException("Couldn't purchase
book: "
         + bookId + ex.getMessage());
   }
}
```

Database Update (contd.)

- Ensure all book purchases in cart are treated as single transaction
  - Inject UserTransaction resource into Dispatcher servlet

```
@Resource
UserTransaction utx;
...
try {
   utx.begin();
   bookDBAO.buyBooks(cart);
   utx.commit();
} catch (Exception ex) {
   try {
      utx.rollback();
   } catch (Exception exe) {
      System.out.println("Rollback failed: "+exe.getMessage());
}
...
```


Persistence In EJB Tier

- Check out the tutorial, chapter 26
- Will get back to this after programming assignments