301AA - Advanced Programming

Lecturer: Andrea Corradini
andrea@di.unipi.it
http://pages.di.unipi.it/corradini/

AP-08-EE: Java EE & Enterprise Java Beans
Overview

- Java EE
- Multi-tier applications
- Components in Java EE:
  - Servlets / JSP (Java Server Pages)
  - EJB (Enterprise Java Beans)

Java Distributions

• Java Card
  – allows Java-based applications (applets) to be run securely on smart cards
• Java SE (Standard Edition)
• Java EE (Enterprise Edition)
  – Suite of specifications for application servers
  – Around 20 implementations available
  – Reference implementation: Oracle Glassfish
• Java ME (Micro Edition)
  – embedded and mobile devices, e.g. micro-controllers, sensors, gateways, mobile phones, personal digital assistants (PDAs), TV set-top boxes, printers...
• JavaFX
  – software platform for creating and delivering desktop applications, as well as rich Internet applications (RIAs) that can run across a wide variety of devices.
Java EE

• Realizes a "standard" platform for the development, execution and management of enterprise applications:
  – **Multi-tier** → structured into "levels"
  – **Web-enabled** → accessible through the Web
  – **Server-centric** → executed in a specific server environment
  – **Component-based** → consisting of sw components running on one or more distributed server instances

• It is based on the Java SE platform to which it adds specifications and tools (API) ad hoc

• Shares the benefits of Java SE applications:
  – One standard specification vs. many implementations
  – Implementations available for most host systems
  – Portability, ease of development, reuse, security, etc.
Multi-tier Architecture

• “Abstract” architectural model for enterprise applications
  – independent of technological choices (language, platform, etc.)
• Functionalities of the application are divided into 3 isolated "levels" (Tiers):
  – **Client Tier** → executes requests to the Middle-tier
  – **Middle Tier** → manages requests from clients and processes application data
  – **Data Tier** → keeps data in permanent storage structures
• Java EE is a particular implementation of the model that focuses on the Middle Tier → **Java EE Application Server**
Java EE: Client Tier

• The **Client Tier** includes the client applications that "use" the enterprise application by communicating with the Java EE Application Server.

• Clients are usually running on hosts other than the one hosting the server.

• Two types of client applications:
  – **Web Client** → is a web browser that makes requests via HTTP to the Web Tier.
  – **B2B Client** → one or more applications that make requests to the Business Tier through SOAP / Web Services or Java RMI.
Java EE: Web Tier

• The **Web Tier** consists of components that manage the interactions between the Web clients and the **Business Tier**

• Main functions:
  – dynamic generation (“on-the-fly”) of content for different clients
  – collection of input data that users send via the Web client interface
  – generation of output based on the Business Tier components
  – control of navigation flow on the client
  – maintaining status for a user session
  – basic application logic and temporary storage of information within Java components (e.g. JavaBeans)
## Java EE: Web Tier

<table>
<thead>
<tr>
<th>Technology</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Servlets</td>
<td>Java classes that process HTTP requests and dynamically generate responses (HTML)</td>
</tr>
<tr>
<td>JavaServer Faces (JSF)</td>
<td>Design framework for Web application user interface</td>
</tr>
<tr>
<td>JavaServer Faces Facelets</td>
<td>Particular JavaServer Faces applications that use XHTML pages instead of JSP</td>
</tr>
<tr>
<td>Expression Language</td>
<td>Set of standard tags used in JSP and Facelets to refer to Java EE components</td>
</tr>
<tr>
<td>JavaServer Pages (JSP)</td>
<td>Text documents compiled and transformed into Servlets to add dynamic content to HTML pages</td>
</tr>
<tr>
<td>JavaServer Pages Standard Tag Library</td>
<td>Tag library that collects features common to JSP pages</td>
</tr>
<tr>
<td>JavaBeans Components</td>
<td>Java objects for temporary storage of the contents of an application</td>
</tr>
</tbody>
</table>
Java EE: Business Tier

- The **Business Tier** consists of components that provide the *business logic* of the application

<table>
<thead>
<tr>
<th>Technology</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise JavaBeans (EJB)</td>
<td>Components managed by the Application Server that encapsulate the main functionalities of the application</td>
</tr>
<tr>
<td>JAX-RS RESTful Web Services</td>
<td>API for creating REST Web Services (via HTTP GET and POST)</td>
</tr>
<tr>
<td>JAX-WS Web Service Endpoints</td>
<td>API for creating and consuming Web Services XML / SOAP</td>
</tr>
<tr>
<td>Java Persistence API Entities</td>
<td>API for mapping data contained in persistent storage systems and corresponding Java objects</td>
</tr>
<tr>
<td>Java EE Managed Beans</td>
<td>Essentially EJBs that do not require security / transactional requirements</td>
</tr>
</tbody>
</table>
Java EE: Data Tier

• The **Data Tier** refers to the various “data sources” from which the application can draw and includes:
  – Relational Database Management Systems (MySQL, Oracle, etc.)
  – Enterprise Resource Planning Systems (SAP)
  – Mainframes (IBM AS / 400)

• The data sources
  – are located on hosts other than the one on which the Java EE Application Server is running
  – are accessed by the Business Tier components
## Java EE: Data Tier

<table>
<thead>
<tr>
<th>Technology</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Database Connectivity API (JDBC)</td>
<td>Low level API for accessing and retrieving data stored on permanent media. Typically used to execute SQL queries to a particular RDBMS</td>
</tr>
<tr>
<td>Java Persistence API (JPA)</td>
<td>API for handling persistent storage of Java objects exploiting a Relational Database</td>
</tr>
<tr>
<td>Java EE Connector Architecture (JCA)</td>
<td>API for connecting application servers and enterprise information systems (EIS, legacy systems),</td>
</tr>
<tr>
<td>Java Transaction API (JTA)</td>
<td>API for defining and managing transactions between multiple and distributed data sources</td>
</tr>
</tbody>
</table>
Java EE Application Servers

• Server that implements the **Java EE** platform
• Hosts the **Middle Tier** components of a multi-tiered enterprise application
• Provides the **standard services** specified by Java EE to these components in the form of a **container**:
  – concurrency management, scalability
  – security
  – persistence, transactions
  – life cycle management of software components
• “Famous” Java EE servers: **GlassFish** (open-source reference implementation: Oracle till 2017, now Eclipse Foundation), **JBoss AS** (Red Hat), **WebLogic** (Oracle-BEA), **WebSphere** (IBM), etc.
Java EE Containers

• Interface between an application component and the low-level features provided by the platform to support that component

• The functionalities of a container are specified by the platform

• One type of container for each type of component

• Java EE Server provides the various containers with a homogeneous environment in which the functioning of each component of the application is guaranteed
Web Container

• Interface between web components and the web server
• A web component can be a Servlet, a JSF or JSP page
• Manages the component’s life cycle
• Dispatches requests to the various components of the application
• Provides interfaces to “contextual data” (e.g. information on the current request)
Application Client Container

• Interface *(gateway)* between Java EE client *applications* and the Java EE *server*

• The clients are particular Java SE applications that use the Java EE server components

• Running on client machines (generally different from the Java EE server)
EJB Container

• Interface between Enterprise JavaBeans that implement the business logic of the application and the Java EE server
• Running on the machine that hosts the Java EE server
• Manages the execution of the EJB components of the application
Life Cycle of Java EE Applications

• Development / deployment cycle
  – Static content design (HTML, CSS, etc.)
  – Dynamic content development (Servlets, JSPs, EJBs, etc.)
  – Deployment descriptors (web.xml, application.xml, ejb-jar.xml, etc.)
  – Packaging (JAR, WAR, EAR, etc.)
  – Deployment packages (JAR, WAR, EAR, etc.) on Java EE server (e.g., JBoss AS)
  – Management of Java EE applications running on the server
Life Cycle of Java EE Applications
Deployment Descriptors

• Files that contain the “instructions” for a given container on how to use and manage the Java EE components
  – Safety
  – Transactions
  – Persistence
• Customizable (XML-based)
• They guarantee the portability of the components
Java components for WEB applications

Client Tier

Web Client

B2B Client

Middle Tier

Web Tier

Business Tier

Connector/Messaging Tier

Data Access Tier

Java EE Application Server

Data Tier

Legacy Tier
Dynamic web contents

- A dynamic web page may vary its content according to the parameters provided by the client at the time of the request
- A dynamic web page may show dynamic contents
- Client- vs. Server-side scripting: where is the code running?
- Client-side scripting: Dynamic content is generated by code running on the client
  - Main client-side scripting language: JavaScript
  - Java components, now obsolete: Applets
Server-side scripting

- The dynamism concerns more than a single web page
- Dynamic content is generated by code running on the (web) server side
- It manages user sessions and controls the application flow
- HTML form, parameters in the request URL, type of browser used, etc.
- Main server-side languages: Perl, PHP, Java, ASP
- Server-side extensions: CGI, JSP, ASP.NET
Server-side web technologies

Web Browser -> Web [Application] Server

- CGI
- Cold Fusion
- PHP
- ASP
- Servlet JSP
Java Servlet/Java Server Pages (JSP)

- A **servlet** is a Java Component designed to handle a web request
- Each servlet must implement the interface `javax.servlet.Servlet`
  - It specifies life cycle methods
- Server request are handled by independent threads in the JVM
- A Java Server Page (JSP) is an HTML page with Java scripts
  - Its compilation causes the execution of a servlet
CGI vs. Servlet/JSP

- CGI: Each request is processed by a separate process.
- Servlet/JSP: Requests are handled by a thread pool in the Servlet Engine.
Servlet and Web Applications

- The Servlet Container provides "low level" services needed for the Servlets and JSPs lifecycle:
  - HTTP connection management, sessions, threading, security, resource management, monitoring, deployment, etc.
Java Servlet: Life Cycle

• In response to a request from the client, the container:
  – Check that the Servlet has already been loaded
    • If it is not, it will load the corresponding class and generate an instance
    • Initialize the newly created instance by invoking the init method on it
  – Invoke the service method corresponding to the Servlet instance passing the objects representing the request and the response as arguments
• Servlet removal from the container is achieved by calling the destroy method
Loading/Instantiation
Class.forName().newInstance()

Initialize
init()

Servicing
service()

Destroy
destroy()

Unload/Garbage Collector
finalize()

1 request = 1 thread

Incoming client requests for the Servlet S
Enterprise Java Beens 3.2 (2013)

• Server-side components that implement the “business logic” of an application
• EJBs cooperate within a Java EE server
• The possible “clients” of the EJBs are:
  – Web Tier Components (local or remote to the server)
  – Remote clients (eg Java RMI)
  – Web Service Client (HTTP / SOAP)
• The idea of EJBs is to move all the application logic out of the Web level, into a specially dedicated layer
EJB 3.2 components

• Two main components defined by the standard:
  – **Session Beans** → perform the application logic, manage transactions and access control
  – **Message-Driven Beans** → perform actions (asynchronously) in response to events (e.g., receiving a JMS message)

• Each of the two types of components has its own specific life cycle

• The behavior of each EJB component is specified through the use of metadata:
  – Code annotations
  – XML descriptors
EJB 3.2 components (note)

• Before EJB 3.0, the so-called Entity Beans were also considered EJB components
  – Since they too were managed by the EJB container
• Entity Beans represent the tables of a relational DB
• Starting from EJB 3.0 the Entity Beans are no longer managed by the EJB container but use services implemented by an appropriate interface (JPA, Java Persistence API)
The EJB Container
The EJB Container

- The EJB instances are running within the EJB container. The container is a runtime environment that controls an EJB component instance and provides all necessary management services for its whole lifetime.
  - **Transaction management**: ensuring transaction properties of multiple distributed transaction executions.
  - **Persistence management**: ensuring a persistent state of an entity bean, which is backed up by database.
  - **Life cycle management**: ensuring the EJB component state transitions in its life cycle.
The EJB Container (cont.)

• The EJB container provides an interface for the EJB component to the outside world. All access requests to the EJB component and responses from the EJB component must get through the EJB container.

• The EJB container isolates EJB component from direct access by its clients.

• The container will intercept the invocation from clients to ensure the persistence, properties of transaction, security of client operations on EJB.

• The EJB container is in charge of generating an EJB home object, which helps to locate, create, and remove the EJB component object.
EJB: Advantages

• Simplify the development of large distributed enterprise applications
  – EJBs developers need to focus only on the logic and leave the rest to the EJB container
  – UI development is simplified given the separation between interface and logic
  – Portability of EJBs on other Java EE servers is guaranteed (as long as they implement at least the same specifications)
EJB: Disadvantages

• Until the EJB 2.1 (2003) specification, EJB applications were difficult to develop, deploy and test
  – Many file descriptors for configurations
  – Automatic unit testing impossible
  – Much heavier than the competing Spring framework

• Several features changed since EJB 3.0
  – Annotations instead of interface implementation
  – Adoption of Dependency Injection
  – Asynchronous communication with EJB’s using futures
EJB: Session Beans

- Reusable components that contain the application logic
- Clients interact with Session Beans either locally or remotely
- Access is via invocation of EJB public methods
- Two types of Session Beans:
  - **Stateless** Session Beans
  - **Stateful** Session Beans
EJB: Stateless Session Beans

• Performs a task on behalf of a client and does NOT maintain its status
  – Or rather, it maintains it only for the duration of the invocation of the method
• It ends when the method call is returned
• Does not keep information in secondary memory (disk)
• They can be "shared" and are the most common EJBs
• Examples: sending emails, converting currency, ...
EJB: Stateful Session Beans

- Maintain client status information through multiple calls to EJB methods
- Release the state once the client requests it, or the bean "ends"
- You may need to keep the status on secondary memory
- Usually represents temporary entities such as the “shopping cart”
EJB: Message Driven Beans

• EJBs that process application messages in asynchronous mode
• Implemented as JMS listeners
• Similar to Stateless Session Beans
  – They do not maintain client status information
• A single MDB can serve messages from multiple clients
• The MDBs do not expose interfaces such as Session Beans
Session Beans: Interfaces

• The Session Beans must define the interfaces with which they can be invoked by the clients
• The interfaces allow you to isolate the underlying implementation (facilitating any future changes)
• Three possible interfaces:
  – Remote Interface
  – Local Interface
  – Web Service Interface
Session Beans: Remote Interface

• Clients may be running on different JVMs than EJBs

• Used by:
  – web components (eg Servlet)
  – other EJBs
  – application client

• “Position” of the EJB is transparent to the client

• The Bean can be distributed → scalability
Session Beans: Local Interface

- Clients must be running **on the same JVM** as compared to EJBs
- Used by:
  - web components (eg Servlet)
  - other EJBs
- “Position” of the EJB must be that of the client
- Strong coupling between client and EJB
Session Beans: Web Service Interface

- Only for "Stateless" Session Beans
- Clients are running on a different JVM than the EJBs
- Used by Web service clients through the protocols provided by the specifications (SOAP / WSDL over HTTP)
- Clients can be implemented in any programming language
Stateless Session Beans: Life Cycle

• Instances created by the EJB container
• Kept in a "pool" of "ready" instances
• Given a method call from a client:
  – The EJB container assigns to the call one of the ready bean instances
  – Once the method is executed, the instance returns to the pool
Stateful Session Beans: Life Cycle

- The client starts a session
- The default bean constructor is invoked
- Resources are "injected" (if present)
- The bean method annotated with the @PostConstruct tag is executed
- From this point on, the bean remains in the cache to perform other client requests
Message-Driven Beans: Life Cycle

- The bean receives a message
- The EJB container searches for an instance of the bean available in the pool
- If available, the instance is used
- Once the execution of the onMessage() method is completed, the instance returns to the pool
- Similar to the Stateless Session Bean
Java Persistence API (JPA)

- It allows to automatically store data contained in Java objects on relational DBs
- Object-Relational Mapping (ORM)
- Applications can manage relational DB tables as "normal" Java objects
- The Java classes (entities) correspond 1:1 to the relational tables defined in the DB
JPA: Advantages

• It has its own SQL-like syntax for static and dynamic queries
  – Java Persistence Query Language (JPQL)
  – Portability compared to various DBs
• It prevents the developer from writing “low level” JDBC / SQL queries
• Provides caching services and performance optimization transparently