Introduction to Java EE (J2EE)
Session Objectives

- Understanding the value propositions of J2EE
- Getting a big picture of J2EE architecture and platform
- Getting high-level exposure of APIs and Technologies that constitute J2EE
  - You don't have to understand the details
- Understanding why J2EE is a great platform for development and deployment of web services
Agenda

- What is J2EE?
- Evolution of Enterprise Application Development Frameworks
- Why J2EE?
- J2EE Platform Architecture
- J2EE APIs and Technologies
- Standard Impl (J2EE 1.4), Compatibility Test Suite (CTS)
- BluePrints
- J2EE and Web Services
- How to get started
What is J2EE?
Enterprise Computing

Challenges
Portability
Diverse Environments
Time-to-market
Core Competence
Assembly
Integration

Key Technologies
J2SE™
J2EE™
JMS
Servlet
JSP
Connector
XML
Data Binding
XSLT

Products
App Servers
Web Servers
Components
Databases
Object to DB tools

Legacy Systems
Databases
TP Monitors
EIS Systems
What Is the J2EE?

- Open and standard based platform for
- developing, deploying and managing
- n-tier, Web-enabled, server-centric, and component-based enterprise applications
The Java™ Platform

Java Technology Enabled Devices
Java Technology Enabled Desktop
Workgroup Server
High-End Server

Micro Edition
Standard Edition
Enterprise Edition
The Java™ Platform

Java 2 Platform Micro Edition (J2ME™)

Java 2 Enterprise Edition (J2EE)
Java 2 Standard Edition (J2SE)

Optimal Packages

Personal Basis Profile
Personal Profile
Foundation Profile
MIDP
CDC
CLDC

JVM
KVM
CardVM

Java Card APIs
What Makes Up J2EE?

- API and Technology specifications
- Development and Deployment Platform
- Standard and production-quality implementation
- Compatibility Test Suite (CTS)
- J2EE brand
- J2EE Blueprints
- Sample codes
Evolution of Enterprise Application Frameworks
Evolution of Enterprise Application Framework

• Single tier
• Two tier
• Three tier
  – RPC based
  – Remote object based
• Three tier (HTML browser and Web server)
• Proprietary application server
• Standard application server
About Enterprise Applications

• Things that make up an enterprise application
  - Presentation logic
  - Business logic
  - Data access logic (and data model)
  - System services

• The evolution of enterprise application framework reflects
  - How flexibly you want to make changes
  - Where the system services are coming from
Single Tier (Mainframe-based)

- **Dumb terminals** are directly connected to mainframe
- Centralized model (as opposed distributed model)
- Presentation, business logic, and data access are intertwined in one monolithic mainframe application
Single-Tier: Pros & Cons

• Pros:
  – No client side management is required
  – Data consistency is easy to achieve

• Cons:
  – Functionality (presentation, data model, business logic) intertwined, difficult for updates and maintenance and code reuse
### Two-Tier

- **Fat clients talking to back end database**
  - SQL queries sent, raw data returned
- **Presentation, Business logic and Data Model processing logic in client application**
Two-Tier

• Pro:
  – DB product independence (compared to single-tier model)

• Cons:
  – Presentation, data model, business logic are intertwined (at client side), difficult for updates and maintenance
  – Data Model is “tightly coupled” to every client: If DB Schema changes, all clients break
  – Updates have to be deployed to all clients making System maintenance nightmare
  – DB connection for every client, thus difficult to scale
  – Raw data transferred to client for processing causes high network traffic
Three-Tier (RPC based)

- Thinner client: business & data model separated from presentation
  - Business logic and data access logic reside in middle tier server while client handles presentation
- Middle tier server is now required to handle system services
  - Concurrency control, threading, transaction, security, persistence, multiplexing, performance, etc.
Three-tier (RPC based): Pros & Cons

• Pro:
  – Business logic can change more flexibly than 2-tier model
    • Most business logic reside in the middle-tier server

• Cons:
  – Complexity is introduced in the middle-tier server
  – Client and middle-tier server is more tightly-coupled (than the three-tier object based model)
  – Code is not really reusable (compared to object model based)
Three-Tier (Remote Object based)

- Business logic and data model captured in objects
  - Business logic and data model are now described in “abstraction” (interface language)
- Object models used: CORBA, RMI, DCOM
  - Interface language in CORBA is IDL
  - Interface language in RMI is Java interface
Three-tier (Remote Object based): Pros & Cons

• Pro:
  – More loosely coupled than RPC model
  – Code could be more reusable

• Cons:
  – Complexity in the middle-tier still need to be addressed
Three-Tier (Web Server)

- Browser handles presentation logic
- Browser talks Web server via HTTP protocol
- Business logic and data model are handled by “dynamic contents generation” technologies (CGI, Servlet/JSP, ASP)
Three-tier (Web Server based): Pros & Cons

• Pro:
  - Ubiquitous client types
  - Zero client management
  - Support various client devices
    • J2ME-enabled cell-phones

• Cons:
  - Complexity in the middle-tier still need to be addressed
Trends

- Moving from single-tier or two-tier to multi-tier architecture
- Moving from monolithic model to object-based application model
- Moving from application-based client to HTML-based client
Single-tier vs. Multi-tier

**Single tier**
- No separation among presentation, business logic, database
- Hard to maintain

**Multi-tier**
- Separation among presentation, business logic, database
- More flexible to change, i.e. presentation can change without affecting other tiers
## Monolithic vs. Object-based

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Object-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 1 Binary file</td>
<td>• Pluggable parts</td>
</tr>
<tr>
<td>• Recompiled, relinked, redeployed every time there is a change</td>
<td>• Reusable</td>
</tr>
<tr>
<td></td>
<td>• Enables better design</td>
</tr>
<tr>
<td></td>
<td>• Easier update</td>
</tr>
<tr>
<td></td>
<td>• Implementation can be separated from interface</td>
</tr>
<tr>
<td></td>
<td>• Only interface is published</td>
</tr>
</tbody>
</table>
Outstanding Issues & Solution

- Complexity at the middle tier server still remains
- Duplicate system services still need to be provided for the majority of enterprise applications
  - Concurrency control, Transactions
  - Load-balancing, Security
  - Resource management, Connection pooling
- How to solve this problem?
  - Commonly shared container that handles the above system services
  - Proprietary versus Open-standard based
Proprietary Solution

- Use "component and container" model
  - Components captures business logic
  - Container provides system services
- The contract between components and container is defined in a well-defined but with proprietary manner
- Problem of proprietary solution: Vendor lock-in
- Example: Tuxedo, .NET
Open and Standard Solution

• Use "component and container" model in which container provides system services in a well-defined and as industry standard

• J2EE is that standard that also provides portability of code because it is based on Java technology and standard-based Java programming APIs
Why J2EE?
Platform Value to Developers

• Can use any J2EE implementation for development and deployment
  – Use production-quality standard implementation which is free for development/deployment
  – Use high-end commercial J2EE products for scalability and fault-tolerance

• Vast amount of J2EE community resources
  – Many J2EE related books, articles, tutorials, quality code you can use, best practice guidelines, design patterns etc.

• Can use off-the-shelf 3rd-party business components
Platform Value to Vendors

- Vendors work together on specifications and then compete in implementations
  - In the areas of Scalability, Performance, Reliability, Availability, Management and development tools, and so on
- Freedom to innovate while maintaining the portability of applications
- Do not have create/maintain their own proprietary APIs
Platform Value to Business Customers

- Application portability
- Many implementation choices are possible based on various requirements
  - Price (free to high-end), scalability (single CPU to clustered model), reliability, performance, tools, and more
  - Best of breed of applications and platforms
- Large developer pool
J2EE APIs & Technologies
J2EE 1.4 APIs and Technologies

- J2SE 1.4 (improved)
- JAX-RPC (new)
- Web Service for J2EE
- J2EE Management
- J2EE Deployment
- JMX 1.1
- JMS 1.1
- JTA 1.0
- Servlet 2.4
- JSP 2.0
- EJB 2.1
- JAXR
- Connector 1.5
- JACC
- JAXP 1.2
- JavaMail 1.3
- JAF 1.0
Java EE 5

- JAX-WS 2.0 & JSR 181
- Java Persistence
- EJB 3.0
- JAXB 2.0
- JavaServer Faces 1.2 – new to Platform
- JSP 2.1 – Unification w/ JSF 1.2
- StAX – Pull Parser – new to Platform
Servlet & JSP (JavaServer Pages)
What is a Servlet?

- Java™ objects which extend the functionality of a HTTP server
- Dynamic contents generation
- Better alternative to CGI, NSAPI, ISAPI, etc.
  - Efficient
  - Platform and server independent
  - Session management
  - Java-based
Servlet vs. CGI

Servlet Based Webserver

CGI Based Webserver

Request CGI1

Request CGI2

Request CGI1

Request Servlet1

Request Servlet2

Request Servlet1

Child for CGI1

Child for CGI2

Child for CGI1

Servlet1

Servlet2

JVM
What is JSP Technology?

- Enables separation of business logic from presentation
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as Java Beans or custom tags
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology
EJB
(Enterprise Java Beans)
What is EJB Technology?

- A server-side component technology
- Easy development and deployment of Java technology-based application that are:
  - Transactional, distributed, multi-tier, portable, scalable, secure, …
Why EJB Technology?

- Leverages the benefits of **component-model** on the server side
- Separates **business logic** from system code
  - Container provides system services
- Provides framework for **portable components**
  - Over different J2EE-compliant servers
  - Over different operational environments
- Enables **deployment-time configuration**
  - Deployment descriptor
EJB Architecture
Enterprise JavaBeans

Synchronous communication
- Session Bean
  - Stateless
  - Stateful
- Entity Bean
  - Bean managed Persistence (BMP)
  - Container managed Persistence (CMP)

Asynchronous communication
- Message-Driven Bean
JMS
(Java Message Service)
Java Message Service (JMS)

- Messaging systems (MOM) provide
  - De-coupled communication
  - Asynchronous communication
  - Plays a role of centralized post office
- Benefits of Messaging systems
  - Flexible, Reliable, Scalable communication systems
- Point-to-Point, Publish and Subscribe
- JMS defines standard Java APIs to messaging systems
Connector Architecture
Connector Architecture

- Defines standard API for integrating J2EE technology with EIS systems
  - CICS, SAP, PeopleSoft, etc.

- Before Connector architecture, each App server has to provide an proprietary adaptor for each EIS system
  - $m$ (# of App servers) x $n$ (# of EIS's) Adaptors

- With Connector architecture, same adaptor works with all J2EE compliant containers
  - 1 (common to all App servers) x $n$ (# of EIS's) Adaptors
m x n Problem Before Connector Architecture
JAAS (Part of J2SE 1.4) 
(Java Authentication & Authorization Service)
JAAS: Authentication

• Pluggable authentication framework
  – Userid/password
  – Smartcard
  – Kerberos
  – Biometric

• Application portability regardless of authentication schemes underneath
  – JAAS provides authentication scheme independent API
  – Authentication schemes are specified Login configuration file, which will be read by JAAS
JAAS Pluggable Authentication

Applications

LoginContext API

LoginModule SPI

Configuration

Kerberos

Smart Card

Biometric
JAAS: Authorization

• Without JAAS, Java platform security are based on
  – Where the code originated
  – Who signed the code
• The JAAS API augments this with
  – Who’s running the code
• User-based authorization is now possible
Other J2EE APIs & Technologies
JNDI

- Java Naming and Directory Interface
- Utilized by J2EE applications to locate resources and objects in **portable** fashion
  - Applications use symbolic names to find object references to resources via JNDI
  - The symbolic names and object references have to be configured by system administrator when the application is deployed.
JDBC

- Provides standard Java programming API to relational database
  - Uses SQL
- Vendors provide JDBC compliant driver which can be invoked via standard Java programming API
J2EE Management (JSR-77)

- Management applications should be able to **discover** and **interpret** the managed data of any J2EE platform
- Single management platform can manage multiple J2EE servers from different vendors
- Management protocol specifications ensure a **uniform view** by SNMP and WBEM management stations
- Leverages JMX
J2EE Deployment (JSR-88) - J2EE 1.4

Tools
- IDEs
- Vendor Deploy Tools
- Management Tools

Standard Deployment API (Universal Remote)

J2EE Platforms
JMX

A single technology for the J2EE platform

- JMX API into the J2EE 1.4 platform
- Dynamic Deployment
- JMX defacto
JACC (Java Authorization Contract for Containers) - J2EE 1.4

- Defines contract between J2EE containers and authorization policy modules
  - Provider configuration subcontract
  - Policy configuration subcontract
  - Policy enforcement subcontract

- Enable application servers to integrate with enterprise user registries and authorization policy infrastructure
J2EE is an End-to-End Architecture
The J2EE Platform Architecture

- B2B Applications
- B2C Applications
- Web Services
- Wireless Applications
- Application Server
- Existing Applications
- Enterprise Information Systems
J2EE is End-to-End Solution

Client Tier

Client

Client

Client

Client

HTML/XML

Web Server
JSP, Servlets

J2EE Application Server

Enterprise JavaBeans™

Enterprise Information Systems (EIS):
Relational Database,
Legacy Applications,
ERP Systems

Other Services:
JNDI, JMS,
JavaMail™

Middle Tier

Enterprise JavaBeans™

Firewall

Client

Client

Client

Client

J2EE is End-to-End Solution
N-tier J2EE Architecture

Web Tier

EJB Tier
J2EE
Component & Container Architecture
J2EE Containers & Components
Containers and Components

**Containers**
- Handle
- Concurrency
- Security
- Availability
- Scalability
- Persistence
- Transaction
- Life-cycle management
- Management

**Components**
- Handle
- Presentation
- Business Logic
Containers & Components

• Containers do their work invisibly
  – No complicated APIs
  – They control by interposition

• Containers implement J2EE
  – Look the same to components
  – Vendors making the containers have great freedom to innovate
J2EE Application Development & Deployment Life Cycle
J2EE Application Development Lifecycle

- Write and compile component code
  - Servlet, JSP, EJB
- Write deployment descriptors for components
  - From Java EE 5, you can use annotations
- Assemble components into ready-to-deployable package
- Deploy the package on a server
Life-cycle Illustration

Creation

Created by Component Developer

Assembly

Assembled and Augmented by Application Assembler

Deployment

Processed by Deployer

Deploy

J2EE Container

Enterprise Components

J2EE Modules → J2EE APP

J2EE APP → Deploy

J2EE Modules
J2EE Development Roles

- Component provider
  - Bean provider
- Application assembler
- Deployer
- Platform provider
  - Container provider
- Tools provider
- System administrator
The Deployment Descriptor

- Gives the container instructions on how to manage and control behaviors of the J2EE components
  - Transaction
  - Security
  - Persistence
- Allows **declarative** customization (as opposed to programming customization)
  - XML file
- Enables **portability** of code
J2EE Application
Anatomies
Possible J2EE Application Anatomies
J2EE Application Anatomies

- 4-tier J2EE applications
  - HTML client, JSP/Servlets, EJB, JDBC/Connector
- 3-tier J2EE applications
  - HTML client, JSP/Servlets, JDBC
- 3-tier J2EE applications
  - EJB standalone applications, EJB, JDBC/Connector
- B2B Enterprise applications
  - J2EE platform to J2EE platform through the exchange of JMS or XML-based messages
Which One to Use?

• Depends on several factors
  – Requirements of applications
  – Availability of EJB tier
  – Availability of developer resource
J2EE 1.4
Standard Implementation, Compatibility Suite, Brand
Standard Implementation

- Under J2EE 1.4 SDK, it is Sun Java Application Server Platform Edition 8
- Production-quality J2EE 1.4 compliant app server
- Free to develop and free to deploy
- Seamless upgrade path to Sun Java Application Server Enterprise Edition
Compatibility Test Suite (CTS)

- Ultimate Java™ technology mission:
  - Write Once, Run Anywhere™
  - My Java-based application runs on any compatible Java virtual machines
  - My J2EE based technology-based application will run on any J2EE based Compatible platforms
J2EE Application Verification Kit (J2EE AVK)

• How can I test my J2EE application portability?
  • Obtain the J2EE RI 1.3.1 and the J2EE Application Verification Kit (J2EE AVK)

• Self verification of application
  – Static verification
  – Dynamic verification

• Obtain the tests results, verify that all criteria are met
Compatible Products for the J2EE Platform (Brand)

ATG
Bea Systems
Borland
Computer Associates
Fujitsu
Hitachi
HP
IBM
IONA

iPlanet
Macromedia
NEC
Oracle
Pramati
SilverStream
Sybase
Talarian
Trifork
The J2EE Platform “Ecosystem,” Application Servers and…

- **Tools**
  - Modeling, Performance, Testing, etc.

- **Enterprise Integration:** Connectors, Java Message Service (JMS) API, XML

- **Components**
- **Frameworks**
- **Applications**
Major Investment in Compatibility by the Industry

• Sun has spent scores of engineer years developing tests
• Licensees have spent scores of engineer years passing the tests
• Testing investment on top of specification investment, implementation investment, business investments
• In total, tens of millions of dollars invested in J2EE platform compatibility by the industry
J2EE Blueprint & Pet Store Application
J2EE Blueprint

• Best practice guidelines, design patterns and design principles
  – MVC pattern

• Covers all tiers
  – Client tier
  – Web tier
  – Business logic (EJB) tier
  – Database access tier

• Sample codes come with J2EE 1.4 SDK
  – Java Pet Store, Adventure builder
Why J2EE for Web Services?
Why J2EE for Web Services?

- Web services is just one of many service delivery channels of J2EE
  - No architectural change is required
  - Existing J2EE components can be easily exposed as Web services

- Many benefits of J2EE are preserved for Web services
  - Portability, Scalability, Reliability
  - No single-vendor lock-in
Web Services Model Over J2EE

J2EE Server JSP™ / JavaServlet API / EJB™

XML
JAX-RPC
JAXM
JAXB
JAXP

Rich Clients

MIDP Devices

Browsers

Services

Rich Clients

MIDP Devices

Browsers

XML/SOAP

XML/SOAP

XML/SOAP

XML/SOAP

XML/SOAP

DBMS

Existing Apps

JDBC

JMS Connectors

HTML/XML
Where Are We Now?

- **Java APIs** for Web Services are being developed very rapidly
  - Web services support on WUST (WSDL, UDDI, SOAP) ready now
  - Next layer Web services work in progress
- **Tools** are available now for exposing existing J2EE components as Web services
- J2EE community has defined overall framework for Web Services (J2EE 1.4, Web services for J2EE)
Design Goals J2EE 1.4 Web Services Framework

- **Portability** of Web services component
  - Over different vendor platform
  - Over different operational environment
- **Leveraging existing J2EE programming models** for service implementation
- **Easy** to program and deploy
  - High-level Java APIs
  - Use existing deployment model
J2EE 1.4 Web Services Framework

- J2EE 1.4 (JSR 151)
- Web services for J2EE (JSR 109)
- JAX-RPC (JSR 101)
- JAXR (Java API for XML Registries)
- SAAJ (SOAP with Attachments API for Java)
- EJB 2.1
How to Get Started
Step 1: For Beginners and Intermediate J2EE Programmers

- Follow along with this course
- Start using J2EE IDE of your choice
- Try open source IDE's
  - NetBeans IDE 5.0 (netbeans.org)
    - Excellent out of the box J2EE support
    - We will use NetBeans IDE 5.0 or NetBeans IDE 5.5 as our default IDE in this course
    - Lots of tutorials
  - Eclipse
Step 2: Next Step (For Advanced J2EE Programmers)

- Learn practical open-source solutions
  - Spring framework (for light-weight framework)
  - Hibernate (for O/R mapping)
  - JDO (for transparent persistence)
  - Struts, WebWork, Tapestry (for Web-tier frameworks)
  - JUnit (for unit testing)
  - Log4j (for logging)
  - Many more
Step3: Next Step (For Advanced J2EE Programmers)

- There is no shortage of quality J2EE online resources
  - java.sun.com/j2ee
  - www.theserverside.com
  - www.javapassion.com/j2ee/J2EEresources.html#J2EEResourceSites
Summary & Resources
Summary

• J2EE is the platform of choice for development and deployment of n-tier, web-based, transactional, component-based enterprise applications
• J2EE is standard-based architecture
• J2EE is all about community
• J2EE evolves according to the needs of the industry
Resources

• J2EE Home page
  – java.sun.com/j2ee

• J2EE 1.4 SDK
  – java.sun.com/j2ee/1.4/download.html#appserv

• J2EE 1.4 Tutorial
  – java.sun.com/j2ee/1.4/download.html#appserv

• J2EE Blueprints
  – java.sun.com/blueprints/enterprise/index.html
NetBeans Resources

• NetBeans IDE Homesite
  – http://www.netbeans.org

• NetBeans IDE Tutorials/Articles Master index
Passion!