Java Enterprise Edition
The Big Problem

- Enterprise Architecture: Critical, large-scale systems
- Performance
  - Millions of requests per day
- Concurrency
  - Thousands of users
- Transactions
  - Large amounts of data
- Distributed
  - Resource location
  - Communications
  - Security risks

Enterprise Java Beans

- Solve once
- Shared container handles system services
- Components provide application specific services
  - Standard container/component interface
Container Concept

A runtime environment for managing components
- Instantiates and manages the beans
- Provides system-level services
- Transaction management
- Security
- Persistence of the bean
- Remote access to the bean
- Lifecycle management of the bean
- Database-connection pooling
- Instance pooling for the beans

Examples of Container Concept:
- Enterprise Container
  - Containing EJB
- Web Container
  - Containing Servlets
- Applet Container
  - Containing Applets

Client interface

Client
Java Editions

Java Mobile Edition

Optional Packages

Web Profile

JEE

Optional Packages

JSE

Java Virtual Machine

CardVM

Card API

KVM

CLDC

CDC

Personal Profile

MIDP

Web Profile

Uses profiles & configurations to handle device capabilities

New web profile to handle less demanding applications

Java Editions

Optional Packages

Java Virtual Machine
• Java Enterprise Edition – reputation for complexity
  • Complex interfaces that all beans had to implement
  • Complex configuration using XML
  • Complex look-up of components & services
  • Various frameworks sprung up to simplify development
    • E.g. Spring

• Java EE 6 and EJB 3.1
  • Beans are POJOs
  • Configuration using annotations
  • JNDI provides easier look-up
  • Better support for REST
Enterprise Application Layers

- The Client UI
- The presentation logic
  - Interacting with end-users.
- The business logic
- The system services
  - security, caching, logging, transactions
- The data access logic
3-Tier Applications: Remote Object

- Remote Object-based
  - Business logic defined through object interface
  - Client/server protected from changes
  - Communication by SOAP/Web Services, RMI, CORBA
  - Concurrency / transaction control in Server
3-Tier Applications: Browser-based

- Browser-based
  - Business logic and data model handled on server
    - Dynamic HTML generation (Servlet/JSP, ASP)
    - Multiple client types including mobile phones
<table>
<thead>
<tr>
<th>Java Enterprise Edition Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Servlet</td>
</tr>
<tr>
<td>• JSP</td>
</tr>
<tr>
<td>• JSF – Java Server Faces</td>
</tr>
</tbody>
</table>
  • Component-based UI development    |
| • Support for Web services          |
  • Synchronous & asynchronous (polling & callback) |
| • Support for XML                   |
  • XML – Java object conversion     |
| • EJB – Enterprise JavaBeans        |
| • Messaging API (JMS)               |
Enterprise Java Beans

- A reusable, portable component
  - EJB methods encapsulate business logic.
    - e.g. a method to update customer data in a database.
- Developed as POJOs, or Plain Old Java Objects,
  - Communicate with container by metadata annotations
- Runs in a container
  - Developer focuses on the business logic
  - Container handles common complicated issues
    - Transaction support, security, and remote object access
- Key concepts
  - Session Beans, Message Beans, Entity Classes

A Bean
(Java – coffee – bean!)

extra information about something
e.g. the date a picture was taken

Not regarded as Enterprise Beans in EJB 3
What Does The Container Support?

- **Lifecycle**
  - Manage threads, object activation & object destruction.

- **State Management.**
  - Beans do not explicitly save or restore conversational object state between method calls.

- **Security.**
  - Perform all security checking
    - Beans do not explicitly authenticate users or check authorization levels.

Transactions.
- Manage start, enrolment, commitment & rollback of transactions

- **Persistence.**
  - Beans do not explicitly handle retrieving or storing data
Summary

- Java EE provides services at all levels
  - UI / business logic / persistence
  - Common enterprise application services
    • Transactions, persistence, concurrency, security
- Container provides services to components
  • Lifecycle management, etc.
  • Use configuration information to link components in
    • Web container, EJB, applet container
- Enterprise beans capture business logic
  • Prefer POJO: Plain Old Java Objects
Session Beans

- Associated with a client performing actions on the server.
  - E.g. stages in the business process: “handle application"
- Stateful
  - Maintain information across calls during bean lifetime
- Stateless
  - No information retained from one call to the next
  - Typically are not persistent – short life
  - Can participate in transactions.
  - Don't survive server crashes.
- E.g. an EJB serving Web pages & tracking the user's path
  - When the user leaves the site, or after a specified idle time, the session object will be destroyed.

A session bean is like a receptionist. Takes requests, responds to some, passes requests to others.
Stateless session beans

- Use the annotation `@Stateless` on the bean.
- No internal state.
  - Each stateless business method call is independent
  - E.g. calculating taxes or shipping charges.
- Easy for container to manage.
  - Can provide beans from a pool
  - Can share beans across clients
Stateful session bean

• Maintains a conversational state across method calls
  • E.g. online shopping cart holding customer details & purchases

• Transient
  • The state is lost when the session ends or the system crashes

• Management
  • A single bean for a single client – maintain its state
  • After any method annotated @Remove the state is discarded.
  • Once the client finishes interacting with the enterprise bean, and the EJB container removes the enterprise bean, the session for the bean ends and all state data for the bean is discarded.
Entity Classes

- Manages persistent data and identified by a primary key
- Specified by the @Entity annotation.
- Represent persistent data from the database,
  - E.g. an employee record in an employee table.
- Sharable across multiple clients.
- All fields not marked with the @Transient are persistent.
- Create object/relational mappings using annotations.
  - To map a class's empId property to the EMPNO attribute of the Employees table, annotate it:
    - @Table(name="Employees") and @Column(name="EMPNO").
Entity Classes 2

- Represent business logic for a persistent entity
- Like tuples (records) in a relational database
  - Persistent
    - The state exists beyond the life of the application that created it.
  - Allow shared access by multiple clients.
    - Concurrency is handled by the container.
  - Have primary keys
    - Primary-key classes identify an instance of an entity bean.
  - May participate in relationships
    - Interfaces are used to manage relationships between beans.
  - Can participate in transactions
    - Beans can specify the transactional properties for their interaction.
entity Classes and ORM

- Insert, update, select, and remove data
- Bean-Managed Persistence (BMP)
  - The programmer writes code to access the data source.
  - More flexible (but more work)
- Container-Managed Persistence (CMP)
  - Container handles database access for the entity bean.
  - The bean isn’t tied to a specific data source
  - The entity bean can be deployed in different containers and with different data-sources.
    - The link to the data source is set up at configuration time
Java Persistence API

- Entities are POJOs.
  - No special methods.
- Standardized object-relational mapping
  - Annotations or XML descriptors can specify ORM mapping
  - The specification defines default values for them.
- Support for inheritance and polymorphism.
  - Queries are polymorphic: affect ancestor and descendants.
- Support for queries
  - Java Persistence Query Language
  - The native query language of the underlying database.
Java Persistence API 2

- Support for pessimistic locking
  - Locks records to prevent collision
  - Slow if collisions infrequent
- Support for optimistic locking
  - Avoids locks but handles failure due to collision
  - Better improve performance if collisions infrequent
- EntityManager API.
  - A standard EntityManager API to perform Create Read Update Delete (CRUD) operations that involve entities.
Message-Driven Beans

- Allow applications to use asynchronous messages
  - The container handles most of the administration.
  - Two models for delivering messages (queue, topic)
- Annotated `@MessageDriven`.
- Implement the `javax.jms.MessageListener` interface
  - Given messages through its standard callback method
    - `onMessage(javax.jms.Message param)`
      - `param` can be cast to
        - `TextMessage`, `ObjectMessage`, `StreamMessage`, …
Messaging Models

- Queues: Point to point
  - Can have many senders & receivers
  - Message consumed by one bean
- Topics: Publish-Subscribe
  - Can have many senders & receivers
  - Message consumed by all subscribing beans
EJB Advantages

- Beans can be deployed in any vendor’s container
  - A bean can rely on the container to manage lifecycle, persistence, transaction, distribution, and security.
- Transaction semantics are defined declaratively
  - In annotations rather than programmatically.
- Scalability
  - Can distribute beans across multiple servers
- Client-Location Transparency
  - Client programs don’t need to know the location of the EJBs.
- Can change data sources easily
- Reusability
  - Business logic can be written once and reused
Summary

- **Session Beans**
  - Stateless
    - Provide a response to a request
  - Stateful
    - Keep information from one request to next

- **Message Beans**
  - Asynchronous message handling

- **Entity Classes**
  - Each instance corresponds to a row in a table
  - Container looks after persistence
    - Reading and writing to/from disk
• Simplify enterprise Java development with EJB 3.0, Part 1
• Simplify enterprise Java development with EJB 3.0, Part 2
• Oracle resources
  • www.oracle.com/technology/tech/java.ejb30.html
• Java EE5 Tutorial
  • java.sun.com/javaee/5/docs/tutorial/doc/bnaad.html