



ITSO Poughkeepsie IBM @server z/OS & zSeries 2003 Technical Update



**Redbooks**

## J2EE and its support in WAS V5 on z/OS

Sabine Holl

[sabine\\_holl@at.ibm.com](mailto:sabine_holl@at.ibm.com)

Alex Louwe Kooijmans

[nl53347@nl.ibm.com](mailto:nl53347@nl.ibm.com)

Kevin J. Senior

[kev\\_senior@it.ibm.com](mailto:kev_senior@it.ibm.com)

© 2003 IBM Corporation

ITSO Poughkeepsie IBM @server z/OS & zSeries 2003 Technical Update



## Trademarks



**Redbooks**

eBusiness on zSeries

© 2003 IBM Corporation

## Abstract



This session includes an overview of the J2EE programming model and how it fits in the WebSphere environment.

It starts with a very brief introduction of Java in general and we end with how everything fits in WebSphere. We spend a good portion of time with the workings of Enterprise JavaBeans.

The duration of this session, depending on the audience, varies between 75 and 90 minutes.

## Agenda



- ➡ Java basics
  - The Java 2 programming model
  - A further exploration of EJBs
  - An overview of Web services
  - What's new in WebSphere Application Server V5

## Java is...



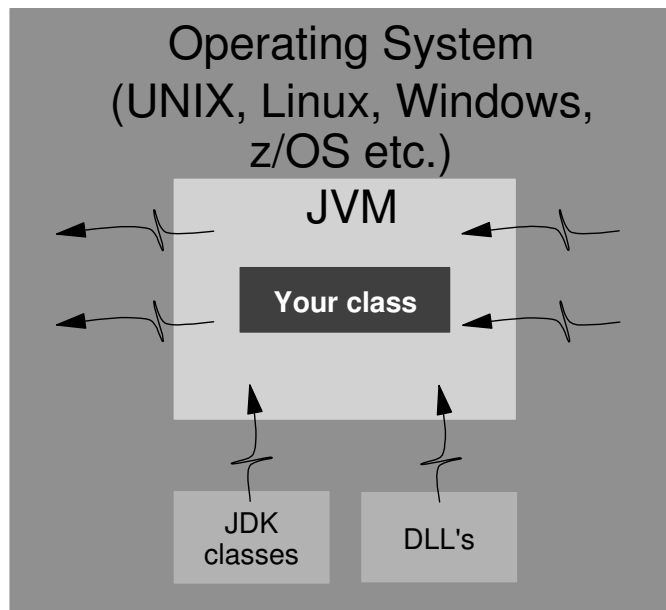
- a language specification
  - ▶ the Java language specification is maintained by SUN
  - ▶ [www.sun.com/java](http://www.sun.com/java)
- a run time environment, called the "Java Virtual Machine"
  - ▶ implemented in Web browsers, (Web) Application Servers and Operating Systems
- a set of tools, for compiling, debugging, documenting etc.
  - ▶ which run from a command line
- a continuously expanding set of APIs, written in 100% pure Java
- a wide choice of application programming frameworks
- a code distribution mechanism
  - ▶ Java archives (.jar files)
- and above all, a philosophy.....
- and it all comes for FREE!

## Some statements regarding Java



- Java does NOT run natively, but ALWAYS requires a JVM
  - ▶ except for some native code compilers that have been developed for Java
- Java programs do NOT get compiled into native code, but into so-called bytecode
- the JVM can ONLY understand bytecode
  - ▶ ...and bytecode is interpreted...
  - ▶ ...which does not mean it is slow and inefficient...
  - ▶ ...as every JVM has a so-called Just-In-Time (JIT) compiler
- this bytecode can be run in any JVM on any platform
  - ▶ ...well, as long as you do not drive platform-specific operations directly from your own Java application code...
  - ▶ ...so, using 100% Pure Java code
- you can do a lot in 100% Pure Java on z/OS
  - ▶ ...and what you cannot do, you should delegate to native (C, C++) code

## The Java Virtual Machine (JVM)



## The Java language is...

- suitable for object-oriented programming
  - ▶ some people program Java, but in a procedural manner
- fairly simple, compared to some other languages
  - ▶ definitely easier to use than C and C++
- highly portable
  - ▶ because of the JVM/bytecode concept
- interpreted
  - ▶ no compilation into native code
- dynamic
  - ▶ memory management
- robust
  - ▶ JVM ensures a certain level of quality
- heavily supported by state-of-the-art development tools
- integrated with many other standards
  - ▶ XML, CORBA, TCP/IP, HTTP, Webservices protocols etc.





## Some more Java terminology

- Java applications are built from "classes"
  - ▶ some developed and others obtained
- a "class" is a collection of data ("fields") and procedures ("methods") that operate on that data
- a class describes the behavior of an "object"
  - ▶ the words "class" and "object" are used many times for the same thing
- a class can have many "instances", each relating to an instance of the corresponding object
- a class can implement one or more "methods", each performing a set of operations on the corresponding object
- "fields" are used to pass values within classes, between instances of classes and between methods of different classes

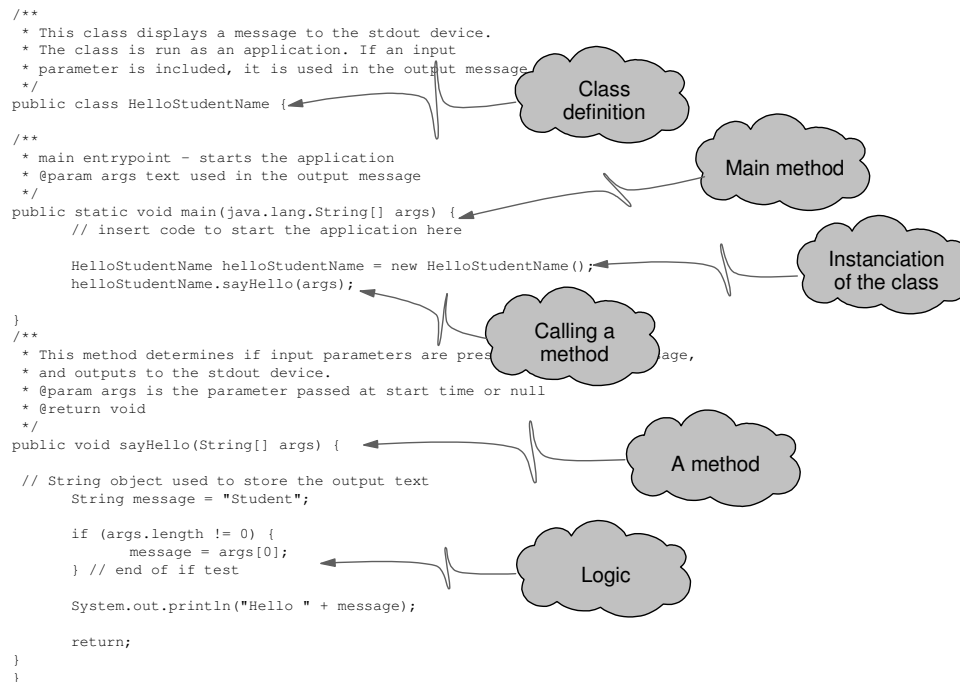


## Some Java terminology (*cont.*)

- a class can be "subclassed", which means that a more specific version of that class is created, "inheriting" the variables and methods from the generic class (= "superclass") and adding specific variables and methods
- each class has at least a "constructor" method, which is responsible for the "instanciation" of a class
- an "interface" is a collection of abstract methods and variables that can be "implemented" by a class
  - ▶ to make access from other objects easier and more transparent
- classes can be grouped together as "packages"
- a "framework" is a collection of Java classes that provides ready-made functionality to the developer



## The HelloStudentName class



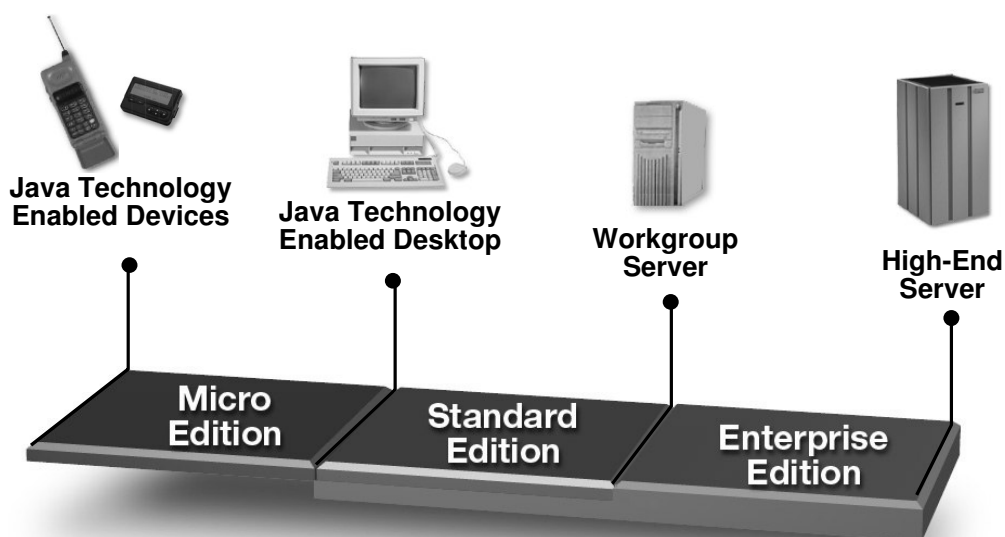
## Java compared with other OO Languages

- Smalltalk
  - ▶ similar object model
    - single-rooted inheritance hierarchy
    - access to objects by reference only
  - ▶ compiled to a *bytecode* (initially interpreted)
  - ▶ dynamic memory allocation and garbage collection
- C++
  - ▶ same syntax for expressions, statements and control flow
  - ▶ similar OO structural syntax (classes, access protection, constructors, method declaration etc.)

## Agenda

- Java basics
- ➔ The Java 2 programming model
  - A further exploration of EJBs
  - An overview of Web services
  - What's new in WebSphere Application Server V5

## A Java environment to run Java

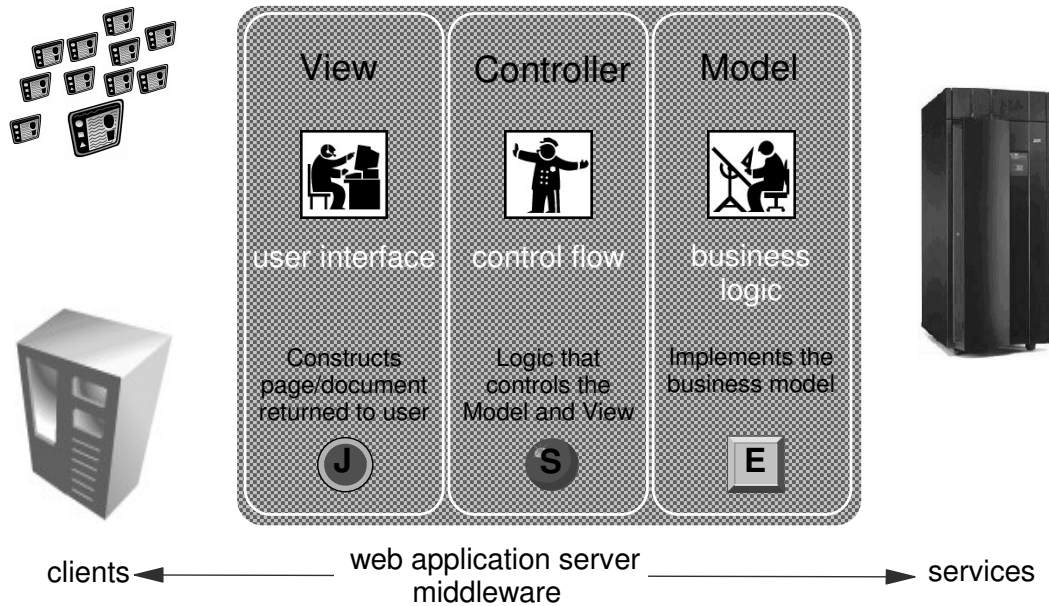




## The MVC programming model - simplified

"A model is an abstract view of reality ... dividing the problem into manageable portions and subsequently applying appropriate technologies to the subsystems."

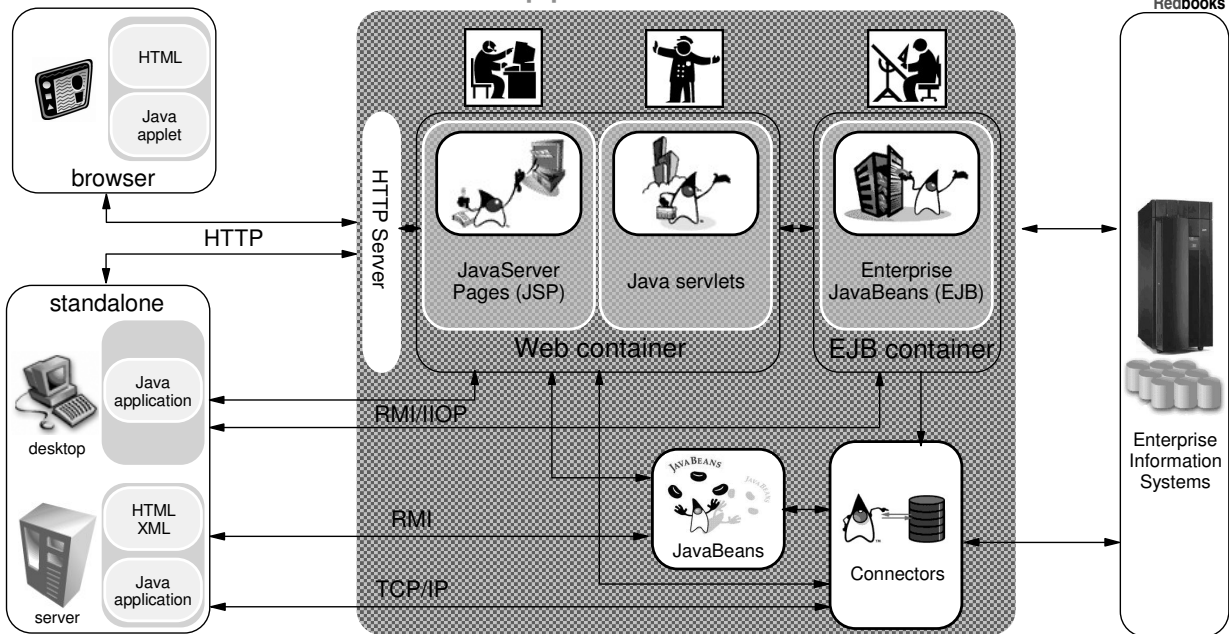
(The J2EE Application Programming Model, Version 1.0 (Beta), Sun Microsystems)



## The MVC model - the role of the components

- Model components encapsulate:
  - ▶ units of work for the business process
  - ▶ logic to access any distributed object servers or back ends
- View components encapsulate:
  - ▶ visible data for the application flow
  - ▶ logic to render the data into the client browser language
- Controller components encapsulate:
  - ▶ user initiated event handlers for the application flow
  - ▶ logic to gather data from model components based on the request
  - ▶ logic to drive the appropriate view components based on the result

## The MVC model in an Application Server context



All containers are optional logical entities ... any container - or all of them - can be bypassed  
 There is no implicit bias favoring one scenario over another  
 There is no suggestion that the scenarios are necessarily exhaustive

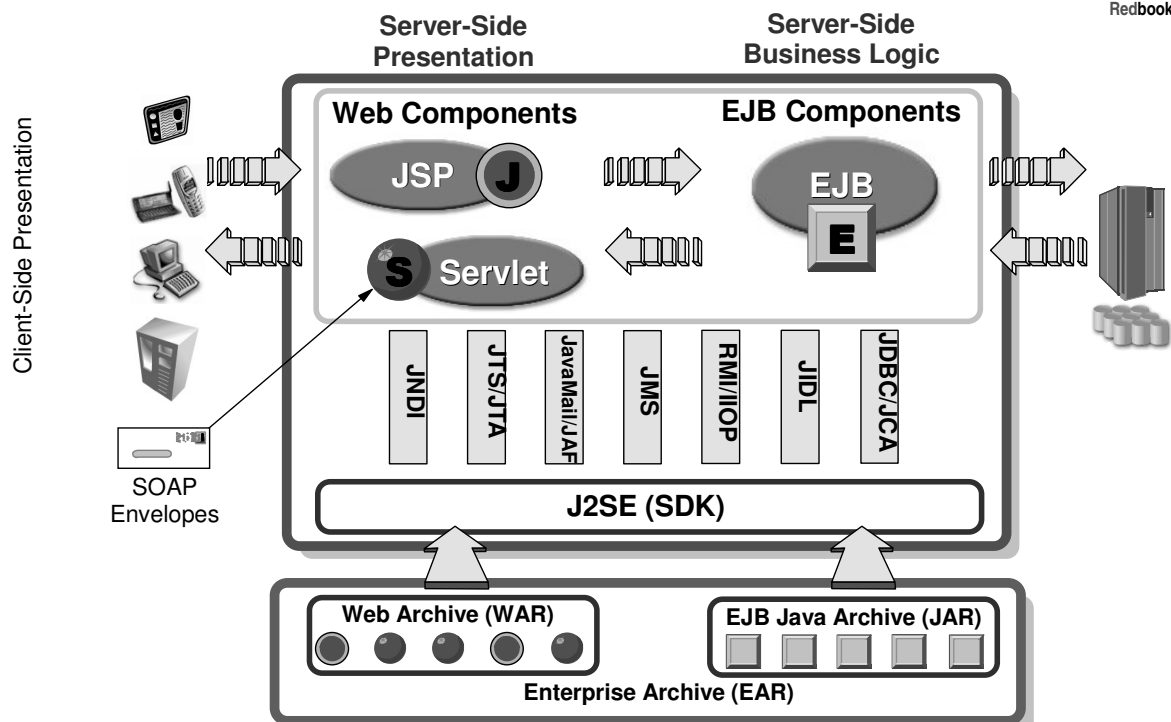
## J2EE frameworks and APIs

- Connectivity
  - ▶ Java Common Connector Architecture (JCA)
  - ▶ Java Message Service (JMS)
  - ▶ Java IDL
  - ▶ RMI-IIOP
- Data Persistence
  - ▶ Java DataBase Connectivity (JDBC)
- Reliability
  - ▶ Java Transaction Service (JTS)
  - ▶ Java Transaction API (JTA)
  - ▶ Enterprise JavaBeans (EJB)
- Front-end
  - ▶ Java servlet API
  - ▶ JavaServer Pages (JSP)
- XML
  - ▶ various APIs
- JavaMail API
- Naming
  - ▶ Java Naming Directory Interface (JNDI)

## A word about the J2EE levels

- The overall J2EE spec. is following a numbering schema, which is not necessarily the same as its subordinate specifications
- It does not match necessarily with JVM, JDK or SDK version numbers either!
- Example:  
The J2EE 1.3 spec. contains EJB 2.0 support and will be supported in the IBM SDKs Version 1.4.
- All middleware vendors have completed a transition phase from J2EE 1.2 to J2EE 1.3, including IBM WebSphere Application Server
- The biggest differences between 1.2 and 1.3 are:
  - ▶ EJB Version 2.0 support
    - CMP persistence restructure
    - Relationships
    - Query language for expressing finders
    - Message-driven beans ...
  - ▶ Java Connector Architecture support
  - ▶ new levels of JSP and servlet

## MVC and J2EE unified





## Types of Java

- Java application
  - ▶ stand-alone application including a "main" method
- Java applet
  - ▶ Browser plugin
- Java servlet
  - ▶ Application Server plugin
- JavaServer Page
  - ▶ Dynamic Web page
- JavaBean
  - ▶ Java "component"
  - ▶ Different types exist, such as access beans and command beans
- Enterprise Bean
  - ▶ Session Bean
  - ▶ Entity Bean
- Java transactions to run in traditional environments
  - ▶ CICS TS 1.3 and up, IMS V7 and up, DB2 V6 and up



## Java applications...

- Execute locally on a computer
- Are started from a (UNIX, Linux, Windows, z/OS etc.) command line or a batch process
- Have platform-dependent capabilities
  - ▶ can access local files and resources directly
  - ▶ can call native C, C++ directly or other languages indirectly
- Are suited for all types of logic including database access, business logic and even GUIs
- Consist of one or multiple class files
- Require a JVM to be started or already active



## Java applets...

- Are always downloaded into the Web browser first
- Can be tested using the JDK appletviewer
- Are written with platform independence in mind
  - ▶ no updates to local files
  - ▶ no calls to non-Java code
- Are not allowed to access local files and resources
  - ▶ unless special security features are used
- Can only communicate with the server (TCP/IP address) from which it was downloaded originally
  - ▶ unless special security features are used
- Are especially meant for intelligent GUIs
- Are not recommended for Internet applications

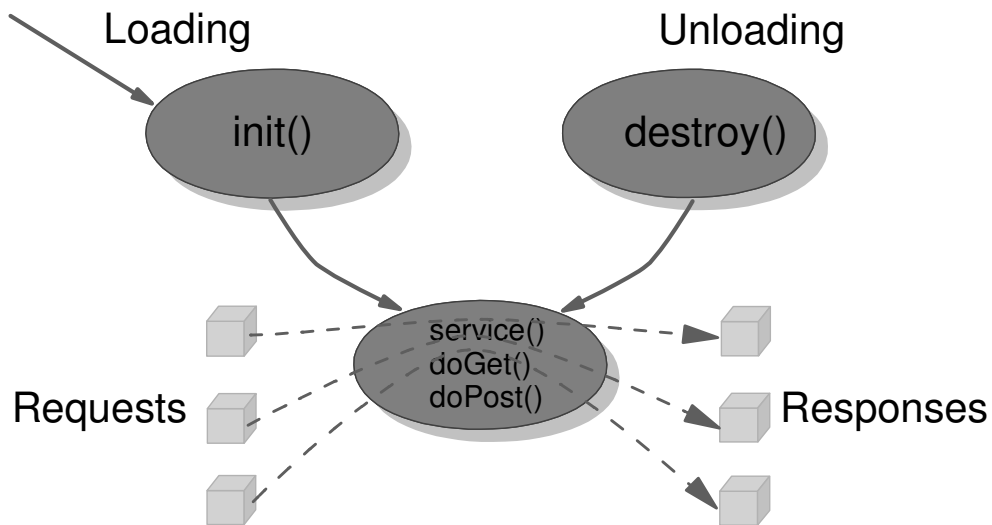


## Java servlets...

- Execute in the JVM of a Web Application Server
- Use a generic set of APIs which are part of the servlet framework (javax.servlet.\*)
  - ▶ the level of this framework is important!
- Usually invoked directly from a browser through the HTTP protocol
  - ▶ `http://myserver.com/myWebApp/MyServlet`
  - ▶ **or, called from another servlet**
- Provide the "controller" layer
  - ▶ use JavaServer Pages for the presentation logic
  - ▶ call Enterprise Beans for executing business logic
- Are portable between Web Application Servers
  - ▶ provided the servlet framework levels are the same!
- Can be built/generated using a variety of Java development tools



## Servlet life cycle



## A typical servlet

```

import packages;

public class <name> extends HttpServlet {

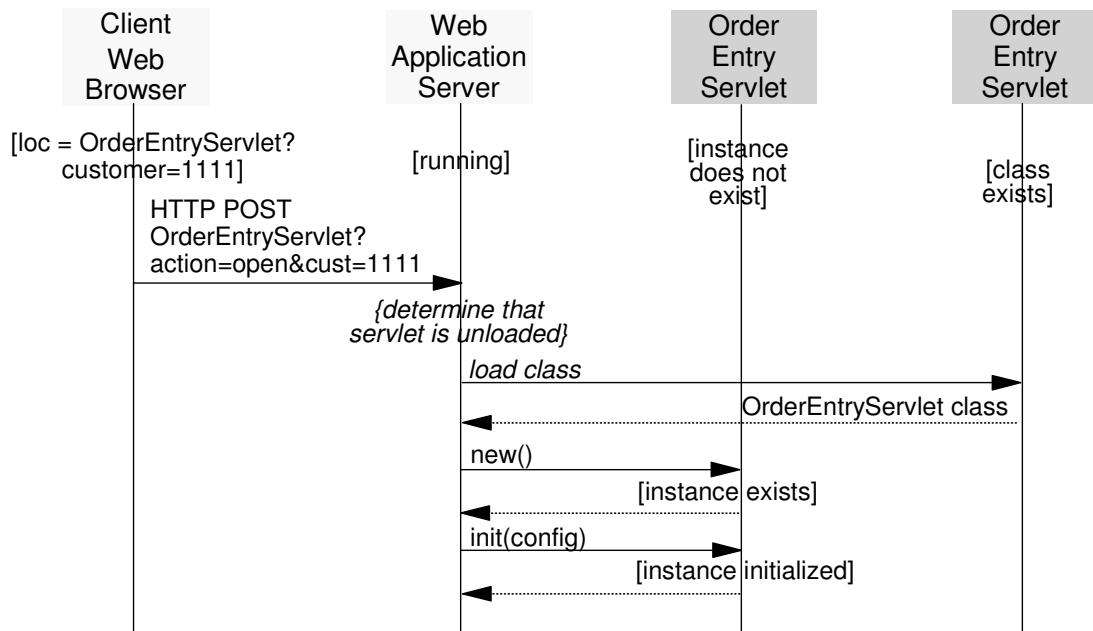
    Initialize variables for servlet

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    javax.servlet.ServletException, java.io.IOException
    {
        PrintWriter out = res.getWriter();
        res.setContentType("text/html");
        Initialize more variables...;
        build html...;
        return;
    }

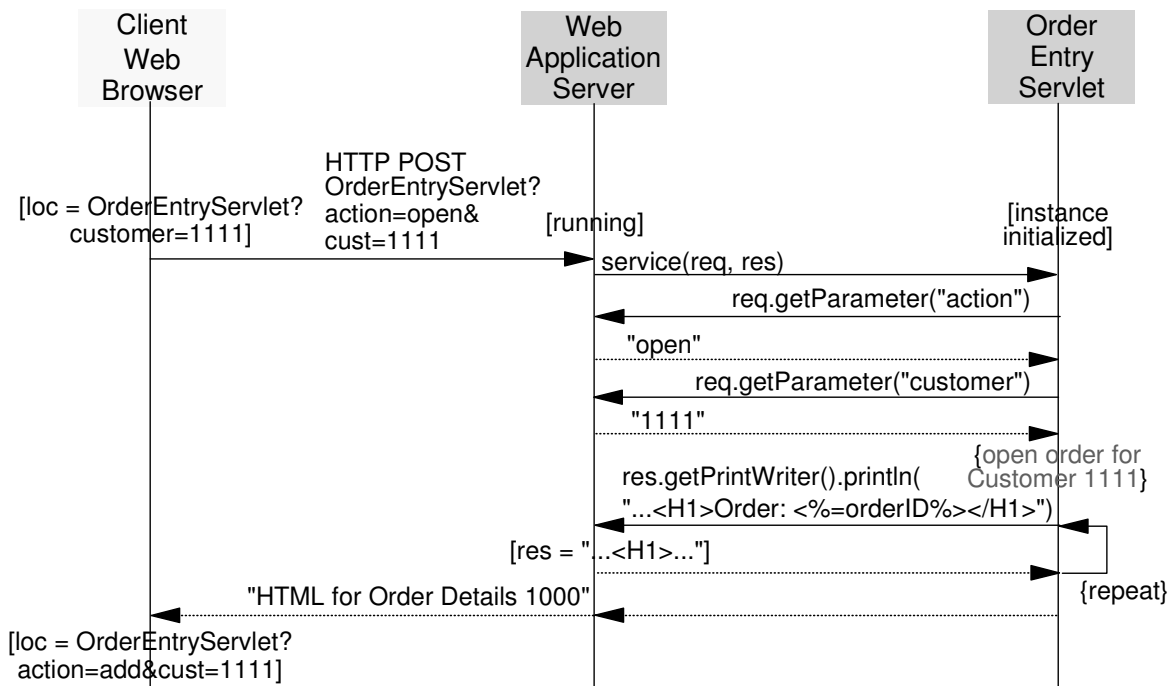
    public void doPost(HttpServletRequest req, HttpServletResponse res) throws
    javax.servlet.ServletException, java.io.IOException
    {
        PrintWriter out = res.getWriter();
        do post processing...;
        build output html...;
    }
}
  
```



## HTTP servlet runtime flow (1)



## HTTP servlet runtime flow (2)



## JavaServer Pages...



- Are developed as dynamic html pages...
  - ▶ ...but upon implementation are converted into servlets
  - ▶ ...and thus run as servlets as well
- Provide the "view" layer, i.e. the presentation
- Contain mostly html statements, but Java code can be imbedded to make the page dynamic
- Specific JSP tags are available too
  - ▶ for instance, to access JavaBeans
- Can be called from a servlet (`callPage` method) or directly from browser (url or html page link)
  - ▶ `http://myserver.com/myWebApp/Myjsp.jsp`
- Are portable between Web Application Servers
  - ▶ provided the JSP levels are the same!

## JavaBeans



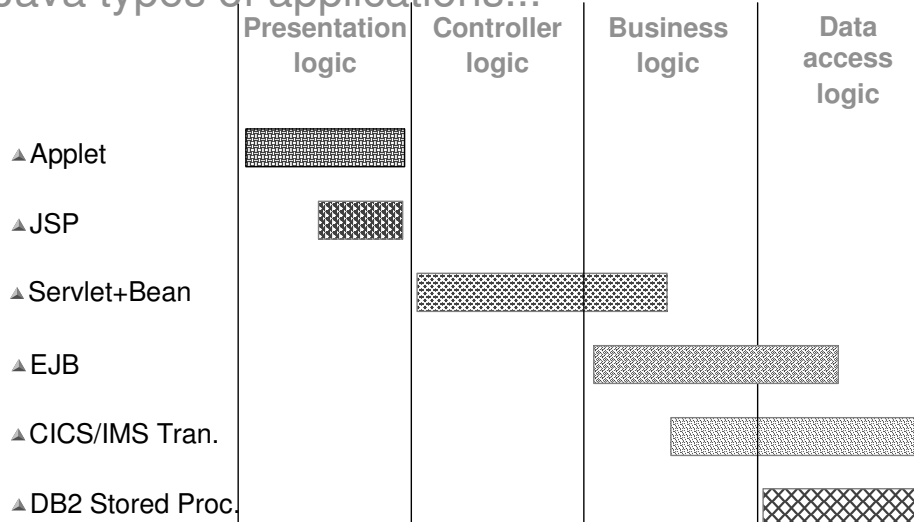
- Can be used in a tool or application via the Bean (public) interface
  - ▶ properties
  - ▶ get and set methods
  - ▶ events
- Are actually also a set of classes
- Types of Beans
  - ▶ Command beans
    - specific business logic task
    - simple, uniform usage pattern
    - hide specific connector interfaces
    - cache information for single round-trip message
  - ▶ Access beans
- Build or buy
- The main difference between a Bean and an ordinary Java class is that a Bean has one single interface which makes it easy and uniform to access

## Enterprise Beans...



- Are server-side Java "components"
- Are implemented in "containers", which are built into an Application Server
- Exhibit enterprise-class type of service
  - ▶ persistence, transaction, naming, messaging, concurrency etc.
  - ▶ QOS is specified using "deployment descriptors"
  - ▶ the container interprets the QOS and propagates the requested service(s) to the application server
- Are built using an EJB-capable tool
- Come in two main flavors:
  - ▶ entity Bean
    - manages business data persistence
  - ▶ session Bean
    - manages sessions
  - ▶ but other types start appearing, such as EMBs (Enterprise Media Beans) and MDBs (Message Driven Beans)

## Java types of applications...



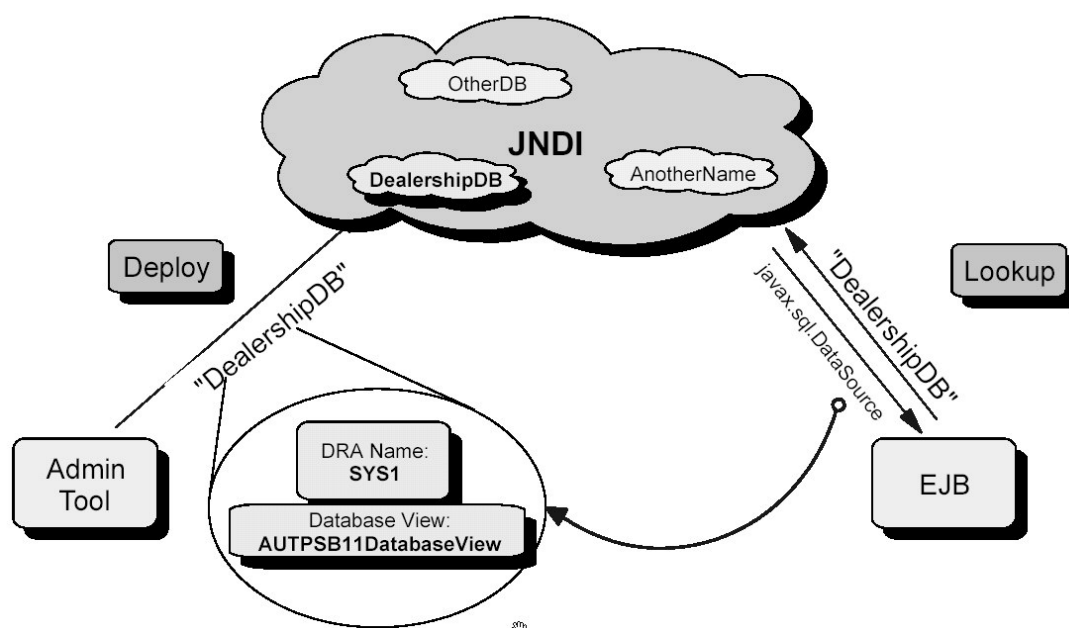
### Notes:

1. An applet can contain intelligent presentation logic, so that network traffic can be reduced significantly. However, the downside of applets is that they are not always compatible with all versions of browsers in the world.
2. Once a JSP is downloaded to browser, the user interface becomes static (html). Eventual intelligence (such as field checking has to be executed on the server.
3. A servlet can theoretically also take care of the presentation (generated html) on one side and the complete business/data access logic on the other side. However, in the latest model the role of the servlet stays limited to controller logic.
4. In EJBs, the goal is to execute the data access logic as much as possible in the container (CMP). However, a persistent EJB will always contain some data access related logic.

## DataSources

- Factory for connections to a physical data source
- Replacement to the *DriverManager* facility
  - ▶ required when running in a managed environment (WebSphere)
- Typically registered with a naming service based on the Java Naming and Directory (JNDI) API
- DataSource objects have properties that can be modified when necessary
  - ▶ code accessing the data source does not need to be changed

## DataSources and JNDI



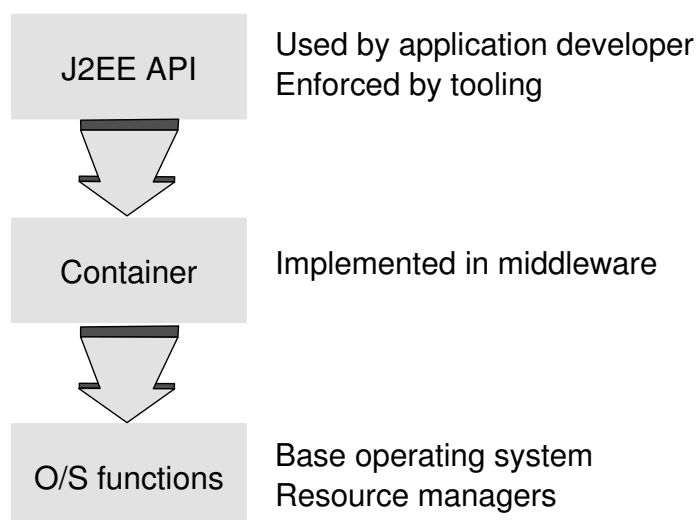


## Agenda

- Java basics
- Java 2 programming model
- ➔ A further exploration of EJBs
- An overview of Web services
- What's new in WebSphere Application Server V5



## J2EE services require middleware and OS

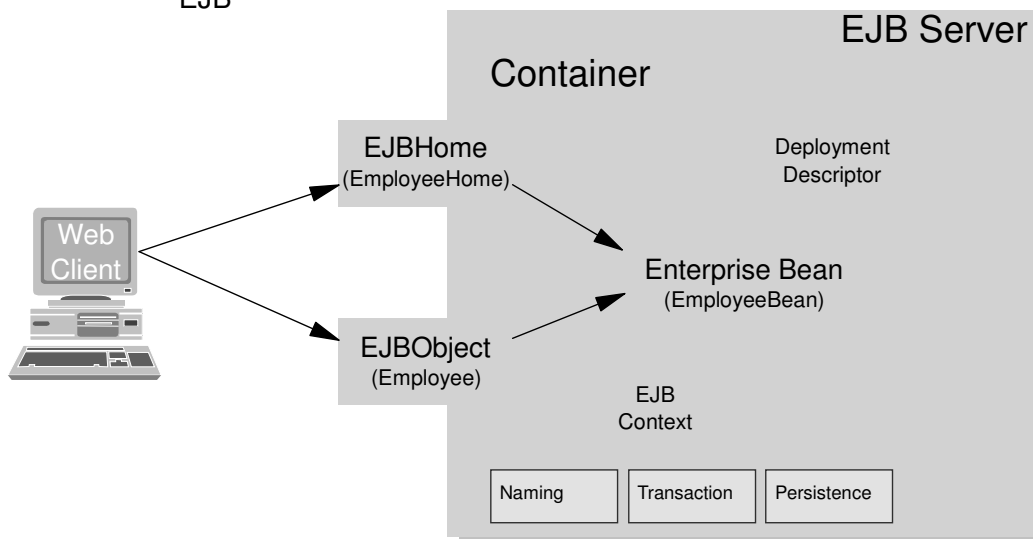


## Role of the container

- Provides an execution environment for J2EE components
  - Web container: servlets and JavaServer Pages
  - EJB container: session and entity Beans
- Provides a buffer between the Java components and the outside world (intercepts all calls)
- Concurrency
  - multiple invocations of Java components at the same time by multiple users
- Access to and pooling of resources
  - connections, threads etc.
- For EJBs:
  - manages remote access to the Bean
  - manages enterprise Bean lifecycle
  - transaction management (includes support for distributed transactions)
  - persistence (entity Beans)
  - security
- It depends on the platform on how the container is implemented

## The EJB runtime model

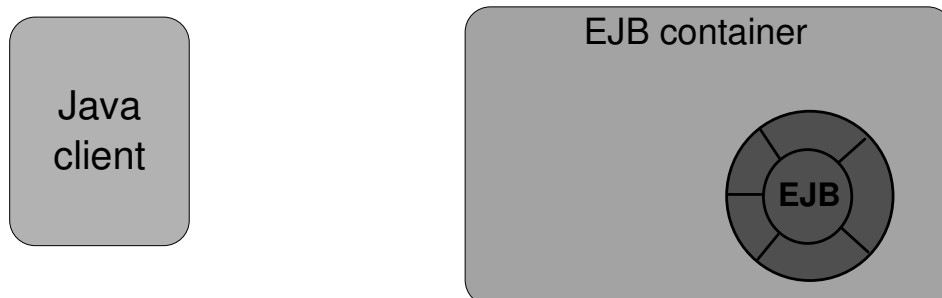
- The "Home" interface provides methods for creating, destroying and locating EJBs
- The "Remote" interface defines the business methods offered by an EJB





## EJB container

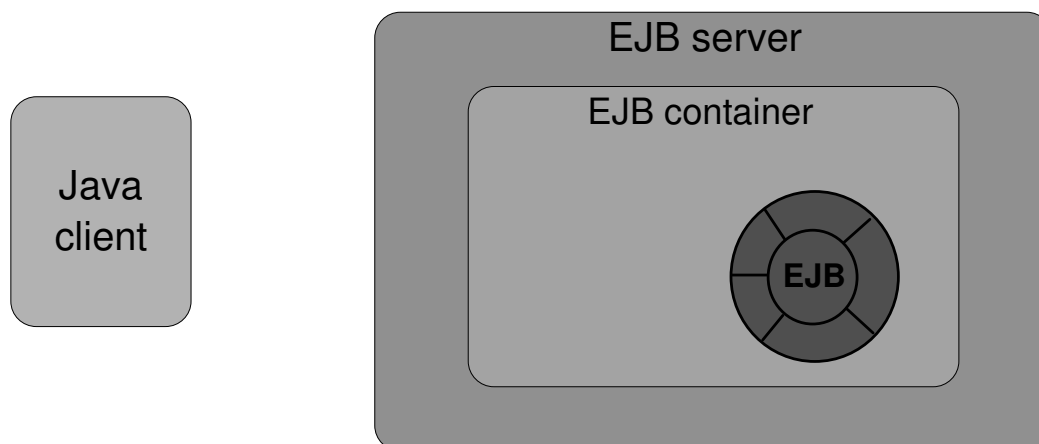
Multiple EJBs can exist in a 'container'



- The container:
  - ▲ manages EJB life cycle
  - ▲ mediates EJB access
  - ▲ manages transactions
  - ▲ imposes security



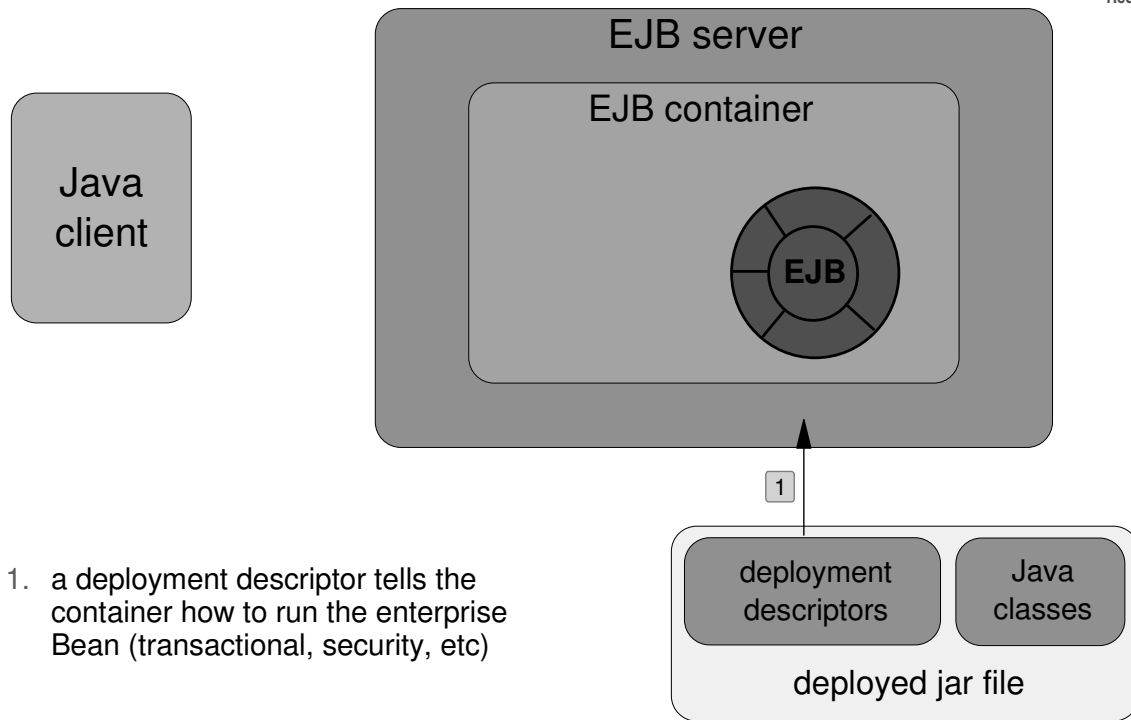
## EJB container is implemented in an EJB server



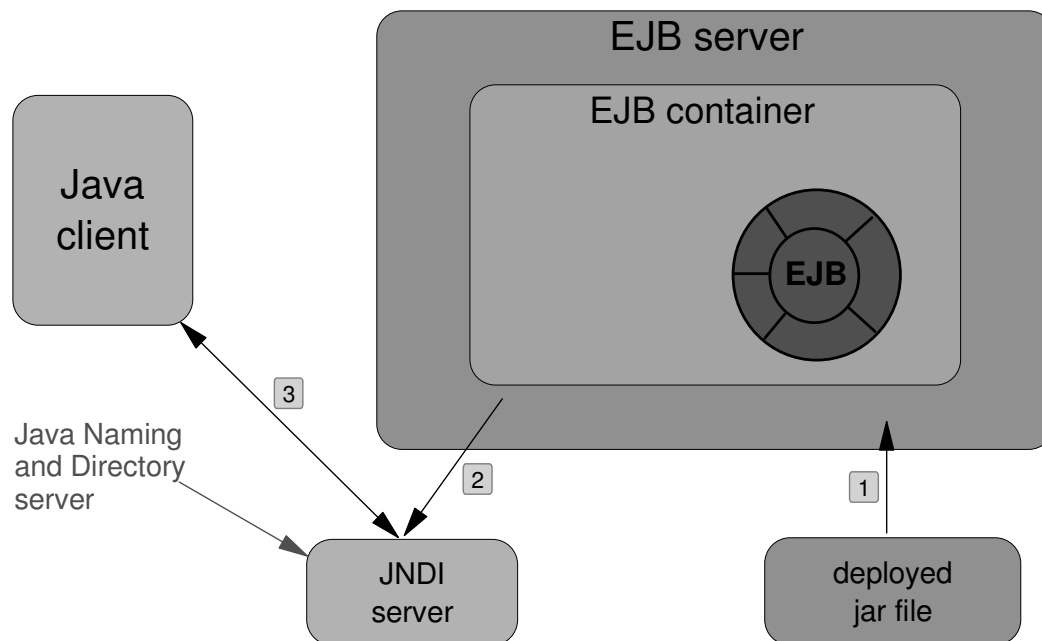
- The EJB specification does *not* define the relationship between the container and the EJB server



## Enterprise JavaBeans - flow (1)

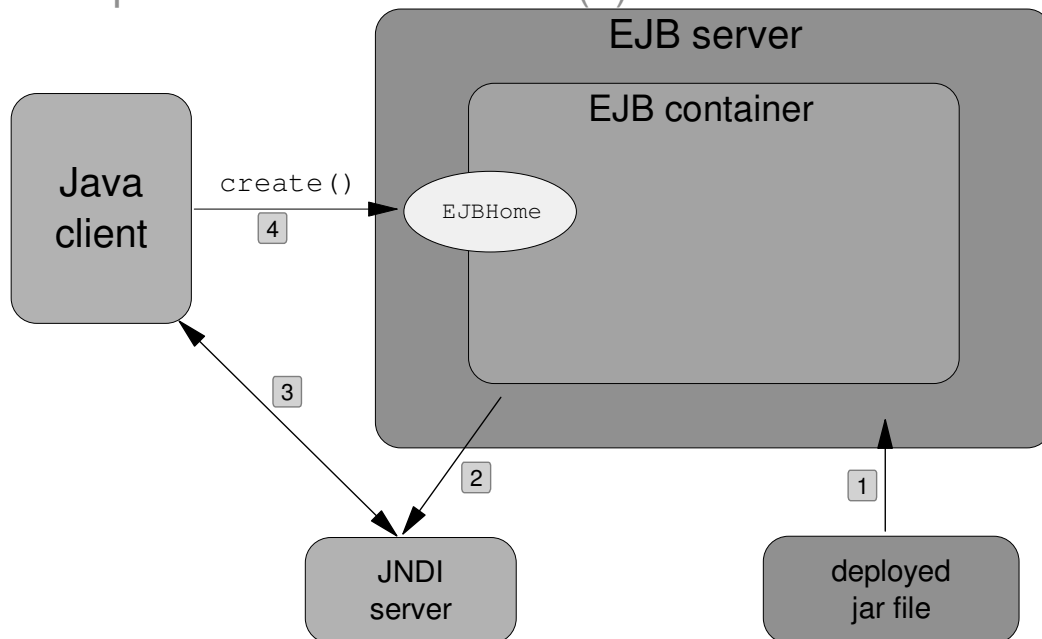


## Enterprise JavaBeans - flow (2)



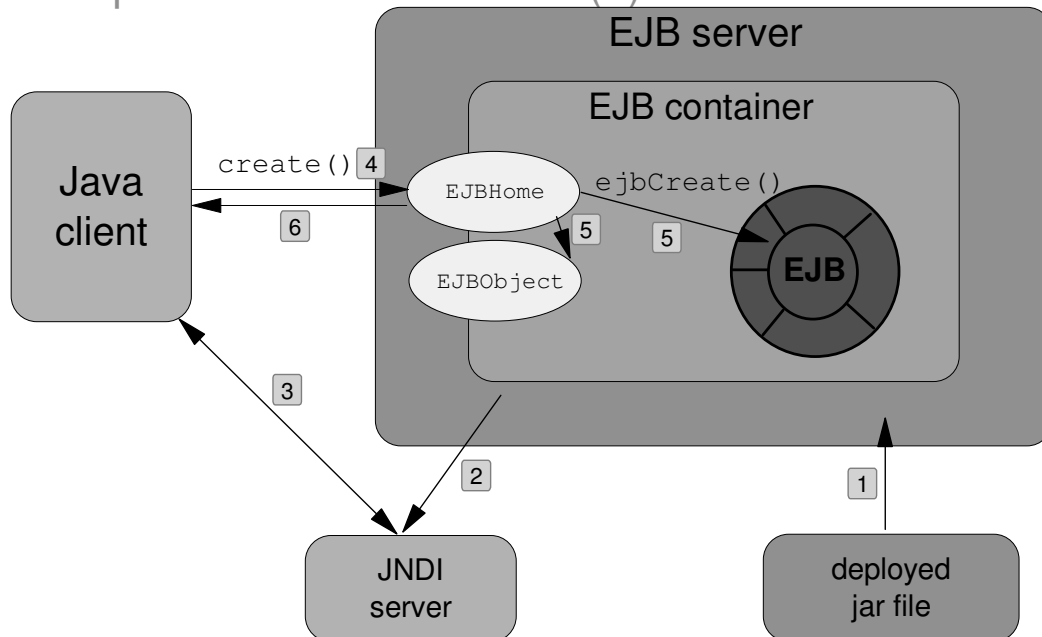
2. enterprise Bean home is 'published' to JNDI server
3. client finds enterprise Bean with a JNDI lookup

## Enterprise JavaBeans - flow (3)



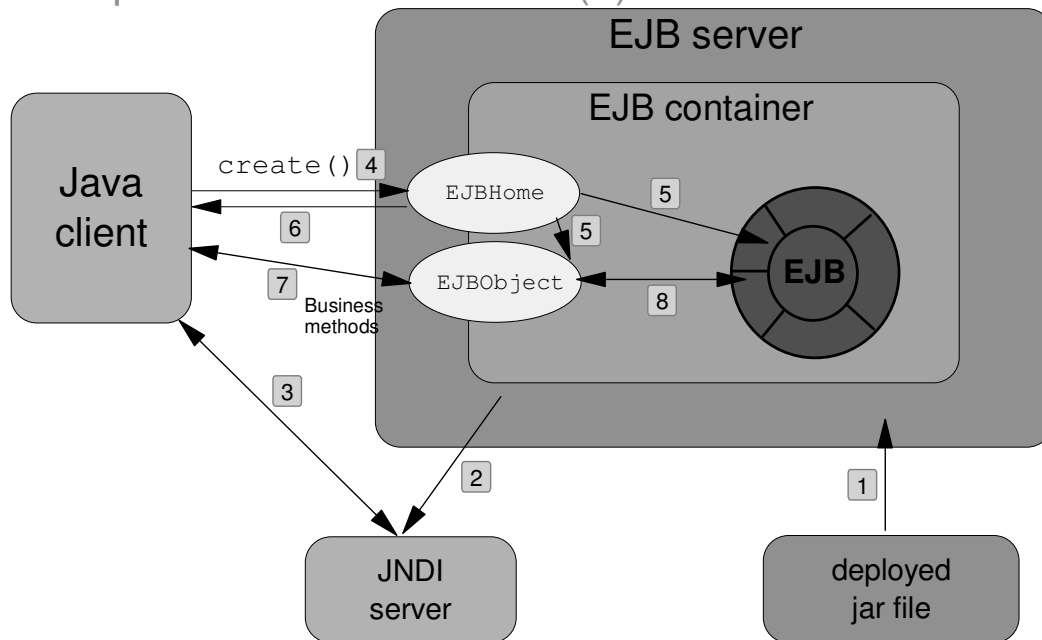
4. a `create()` is requested by the client based on the object info from the JNDI server

## Enterprise JavaBeans - flow (4)



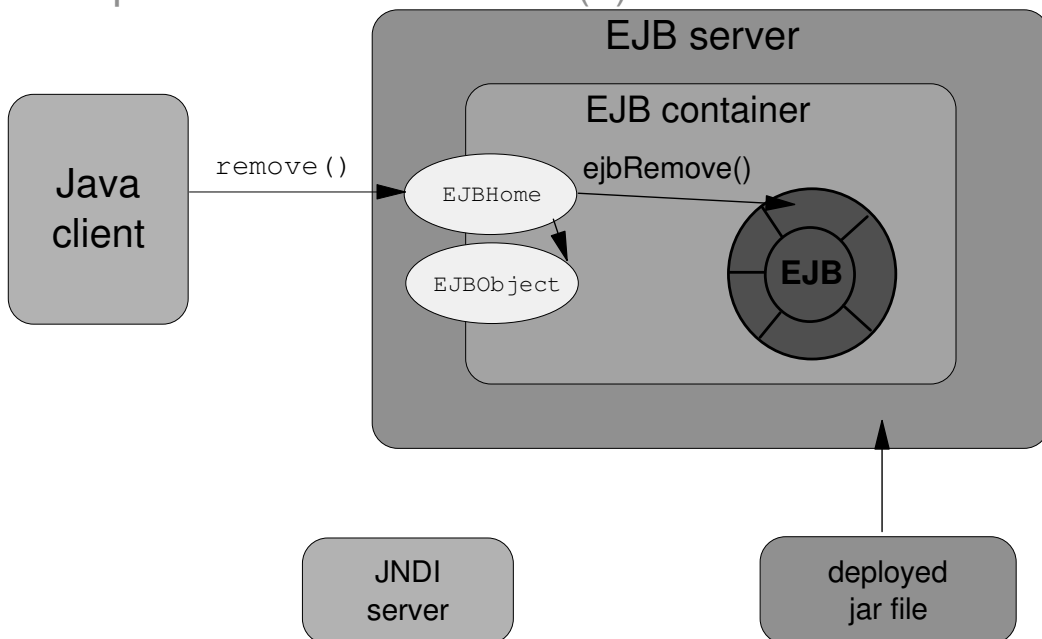
5. the container creates the enterprise bean, drives `ejbCreate()` method on the bean, and
6. exposes its business methods via a 'remote interface' object reference returned to client

## Enterprise JavaBeans - flow (5)



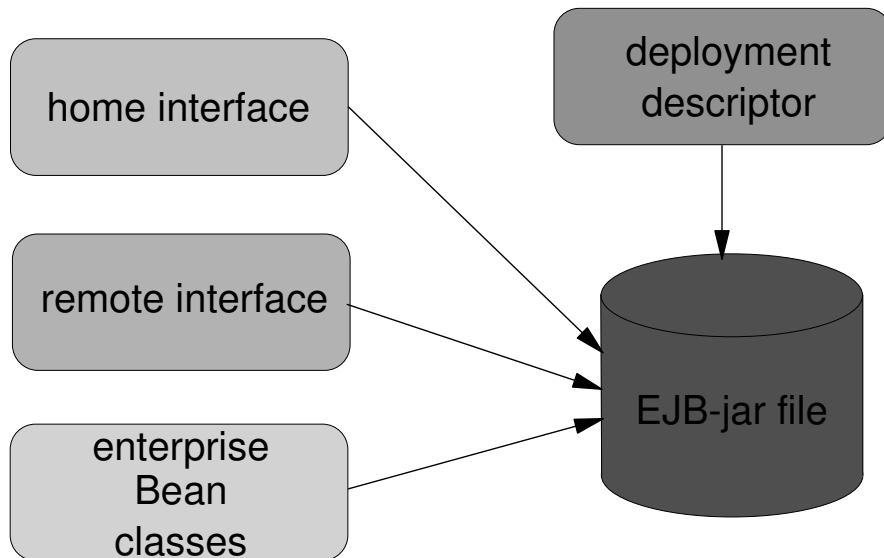
- 7. the client can now drive business methods on the enterprise bean
- 8. the intermediate object is used so that the container can impose transactionality, security, etc.

## Enterprise JavaBeans - flow (6)

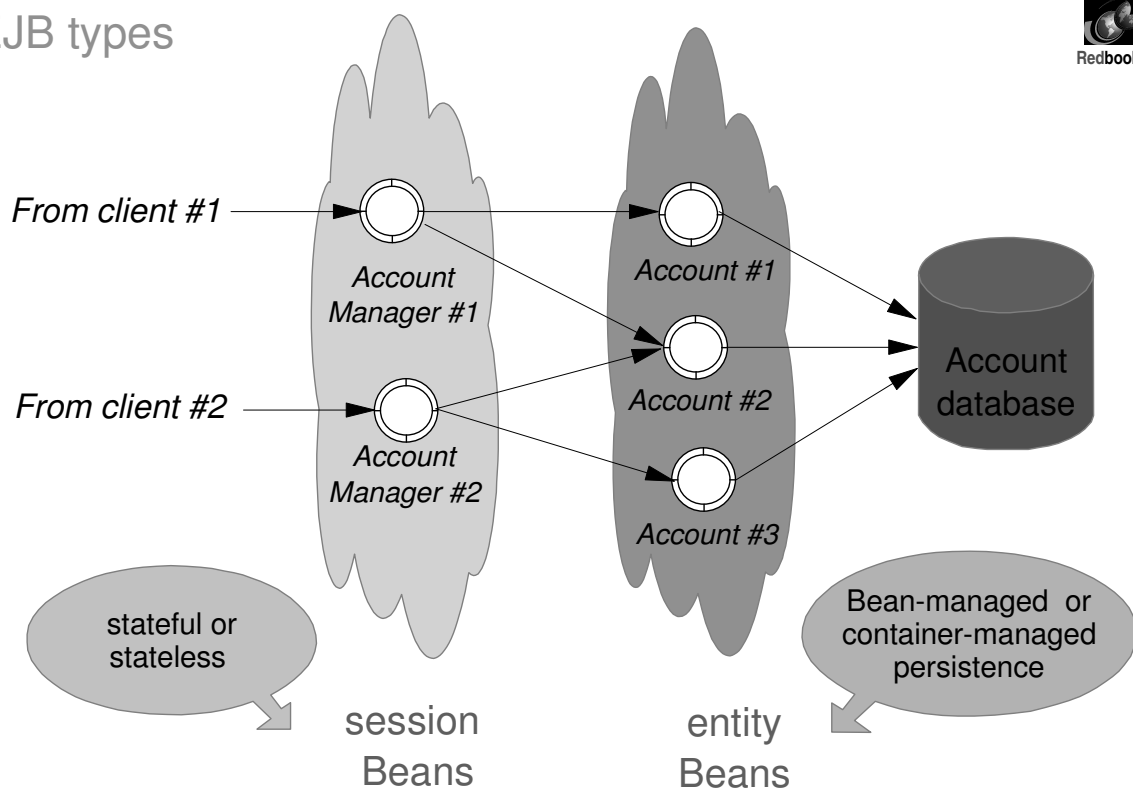


- when the client is done with the enterprise Bean, it can request a `remove()`, the container drives `ejbRemove()` on the enterprise Bean

## What do you need for an Enterprise Bean?



## EJB types





## Entity Bean

- Encapsulates 'permanent' data;
  - ▶ usually represents a row in a database (EJB 1.1 spec.)
- Container Managed Persistence (CMP)
  - ▶ container drives the persistence on behalf of the Bean
- Bean Managed Persistence (BMP)
  - ▶ the Bean drives its persistence itself
- Identified by a primary key
- Backed by a database or back-end application
  - ▶ RDBM via JDBC
  - ▶ transactional EIS via JCA Connector
- Will not run in CICS TS 2.1 (although CICS session Beans can invoke entity beans in other EJS)



## Session Bean

- Tied to the lifetime of a given client session
- Generally short-lived
- Container or Bean managed transactions
  - ▶ Bean managed allows Beans (your code) to determine transaction demarcation
- Stateless
  - ▶ do not maintain conversational state
  - ▶ workload-balanced across replicated servers
  - ▶ can be pooled and reused by any client
  - ▶ example: calculator with no memory
- Stateful
  - ▶ do maintain conversational state
  - ▶ can be passivated
    - in memory if possible, in DB2 if necessary
  - ▶ can move from one server instance to another if necessary
  - ▶ example: shopping cart



## Calling an Enterprise bean is quite complex...

- Obtaining the context to the name server
- Looking up the home of the enterprise bean using the name service context
- Creating an enterprise bean instance from the enterprise bean home, which returns an enterprise bean proxy object
- Accessing the remote methods of the enterprise bean instance through an enterprise bean proxy object using a remote call
  - ▶ quite costly, as each get or set is a remote call



## Access beans make life a lot simpler!

- Can be generated easily using wizards in WSAD
- Are used on the client side of the application
  - ▶ mostly in servlets
  - ▶ JSPs
  - ▶ entity beans calling other entity beans
- Four types:
  - ▶ Java bean wrappers
    - use an entity bean in a client program like an ordinary Java bean
  - ▶ Copy helpers
    - cache (remote) entity bean data, so improve performance
    - client component works with data (get/set) locally
  - ▶ EJB factories (WSAD only)
  - ▶ Data classes (WSAD only)
- Improve performance, as they cache entity bean data on th client side



## Servlets vs. session Beans

### *Why would I use a session Bean and not just a servlet?*

- EJBs can encapsulate calls to other transaction managers within the same transaction context
- EJBs policies applied more granularly
- EJB security
  - ▶ more methods to ACL check
- RMI/IIOP local/remote transparency
  - ▶ backed by naming directory

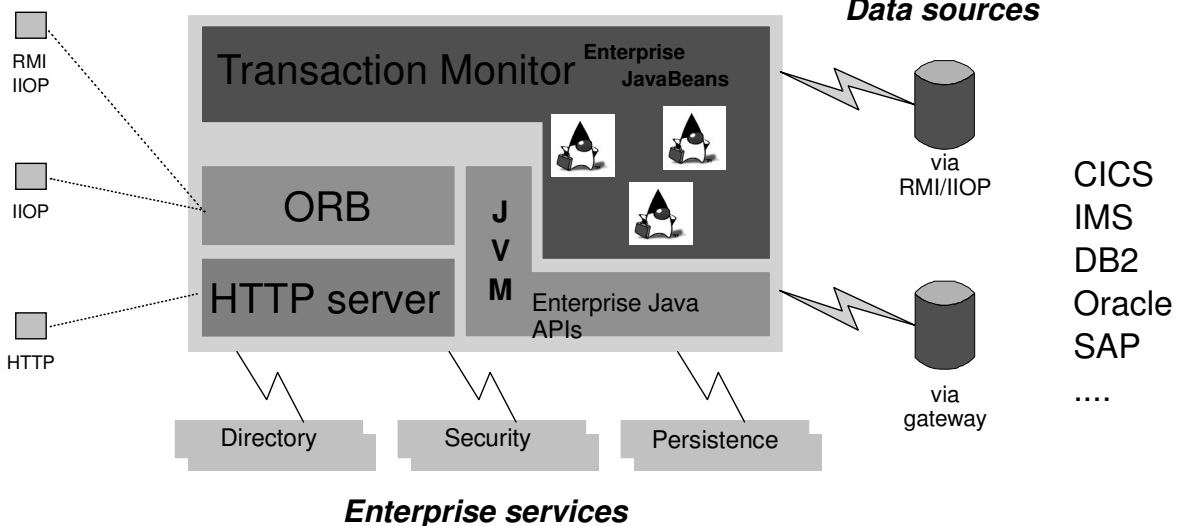


## Where to run EJBs?

- An Enterprise Java Server (EJS) is required to run EJBs and runs those EJBs according to the J2EE framework

**Any client**

**EJB-enabled Application Server**

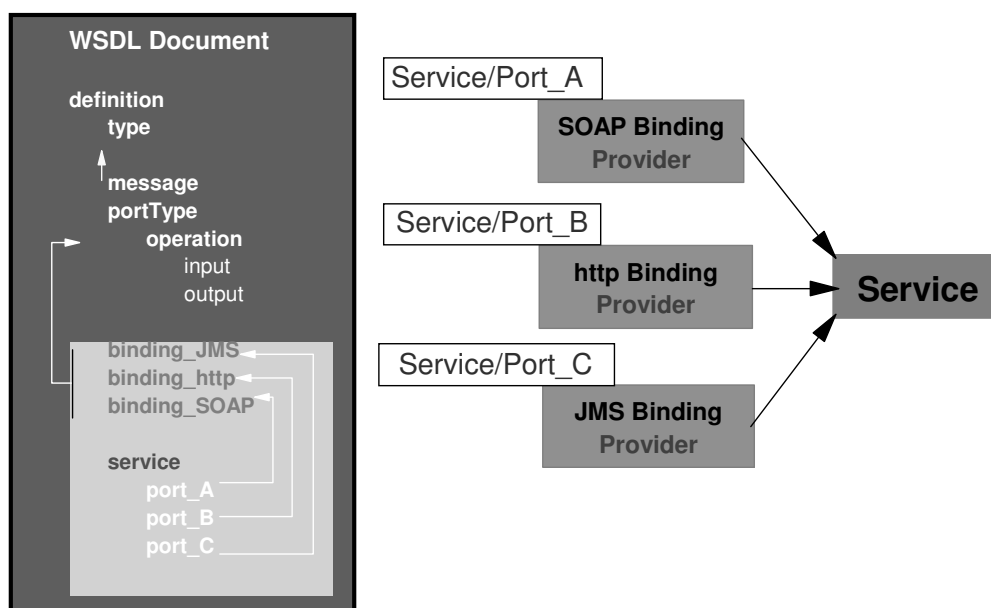


## Agenda

- Java basics
- Java 2 programming model
- A further exploration of EJBs
- ➡ An overview of Web services
- What's new in WebSphere Application Server V5

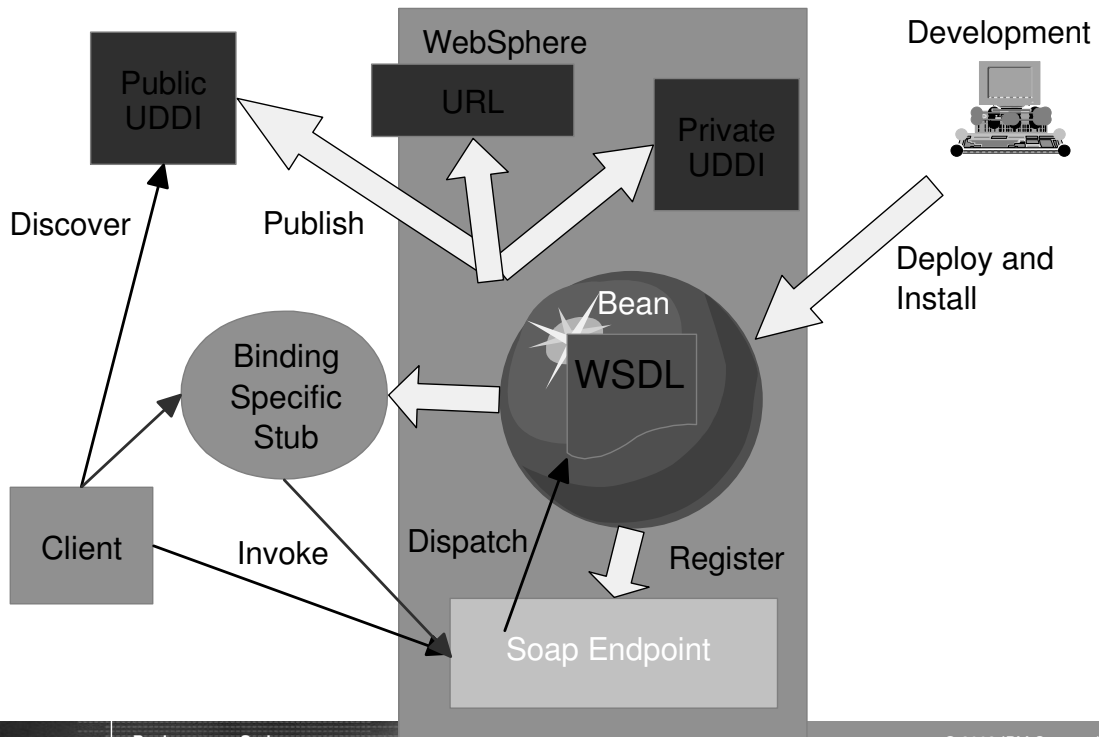
## Web Services - WSDL

### Making the same Service Available via Multiple Bindings





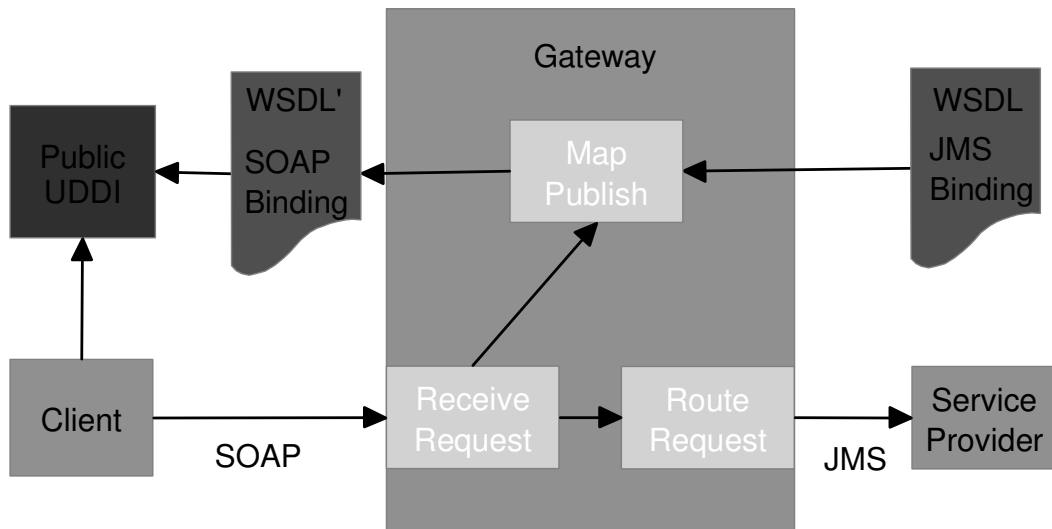
## Web Services in J2EE



eBusiness on zSeries

© 2003 IBM Corporation

## Web Services Gateway



Protocol Conversion

eBusiness on zSeries

© 2003 IBM Corporation



## Agenda

- Java basics
- Java 2 programming model
- A further exploration of EJBs
- An overview of Web services
- ➔ What's new in WebSphere Application Server V5



## The WebSphere V5 Programming Model

- J2EE 1.3 Highlights
  - ▶ EJB 2.0
    - Various improvements, will be addressed in a minute
  - ▶ Servlet 2.3, JSP 1.2, JMS 1.0, JAXP 1.1, JAAS 1.0
  - ▶ JCA 1.0 full compliance
    - Connection manager support for XA-based resource managers
    - Support for non-DB2 databases
- Web Services
  - ▶ Standards compliant (J2EE 1.4)
    - SOAP 2.3, WSDL 1.1, JAX-RPC 1.0, JSR 109
    - AXIS (apache) - SOAP Engine and JAX-RPC reference implementation
  - ▶ UDDI - discovery and publish WSDL, private UDDI directory
  - ▶ Web Services Gateway not supported yet
- All WebSphere family APIs supported at WAS ND level

## The new J2EE 1.3 specs...implemented in WAS V5



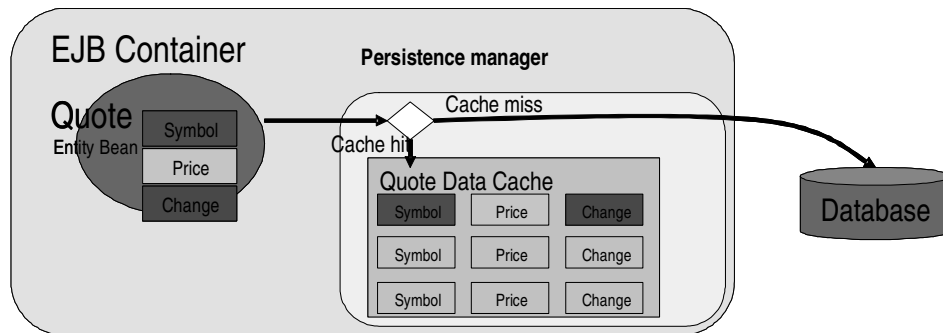
- EJB 2.0
  - ▶ Message-driven Beans
  - ▶ Local Interfaces
  - ▶ CMP and Container Managed Relationships
  - ▶ Many-to-many O/R Mappings
  - ▶ EJB Query Language
  - ▶ EJB Home methods
  - ▶ EJB inheritance
- Servlet 2.3
- JSP 1.2
- J2EE application packaging

## EJB 2.0: Features at a glance



- Local home and local entity interfaces (in memory - no marshaling)
- Message Driven Beans (MDB)
- Container-managed association relationships
- EJB Query Language with WAS extensions (MAX, order by)
- Features for high performance persistence
  - ▶ Read Ahead
    - Pre-loads groups and working-sets of beans in a single datastore operation by following selected bean relationships
  - ▶ Optimistic Concurrency Control
    - Minimizes the amount of time data is actually locked during updates and thus increases overall throughput in heavily-used applications
    - Improve application performance without changing semantic behavior
- CMP Data Cache
  - ▶ Holds the results of finders.
    - Long lifetime caching, for beans that change only infrequently and remain read-only across many transactions or are configured for optimistic transactional control

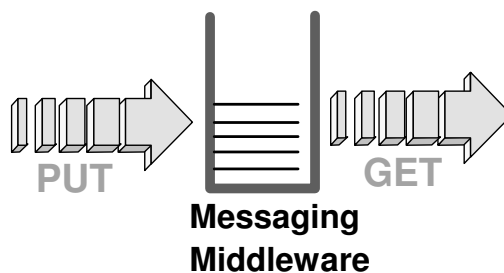
## EJB 2.0: CMP Data Cache



## Messaging

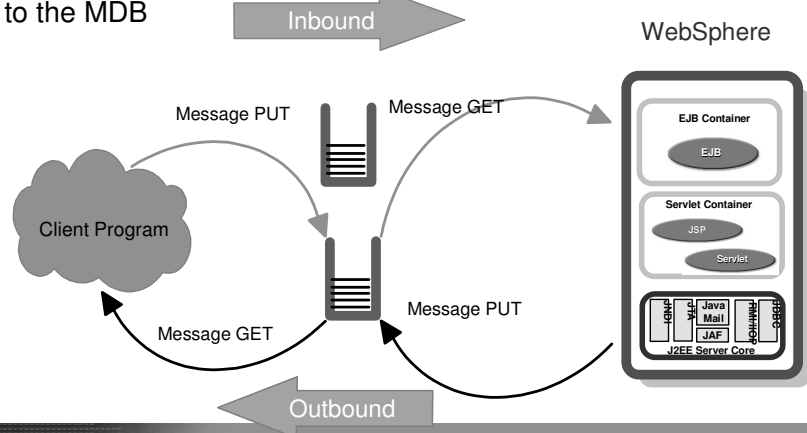


- Why is messaging beneficial?
  - ▶ Loose Coupling
  - ▶ Enables asynchronous communications among applications
    - Application data sent as a message
    - Message consumers may not always be active
      - Messages stored and forwarded to consumers at a later time
  - ▶ Various QoS
    - Guaranteed delivery
    - Transactional
    - Security



## Message-Driven Beans (MDB)

- Message Driven Beans (MDBs)
  - ▶ special Enterprise Beans oriented to processing messages
  - ▶ listen on JMS destinations
  - ▶ modeled after a stateless session bean
  - ▶ developer writes onMessage(java.jms.Message message) method
  - ▶ container creates instances as required
  - ▶ server gets connections to queue or topic, receives the message and delivers it to the MDB

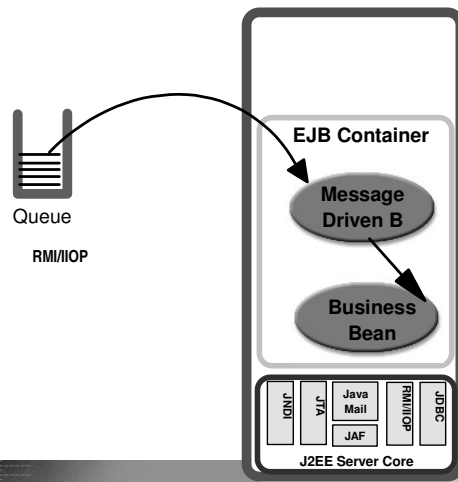


eBusiness on zSeries

© 2003 IBM Corporation

## Benefits of Message-Driven Beans

- Automatic consumption of messages
  - ▶ No **polling** needed in the application code
- Reduce application code
- Synchronous communication between Queue and Listener
- Leave resource management to the container
- WebSphere can now receive 3 styles of requests directly : JMS, IIOP and HTTP



eBusiness on zSeries

© 2003 IBM Corporation



## Message-driven Beans Programming Model

- Basics
  - ▶ Stateless enterprise beans, server side components
  - ▶ Transactional
  - ▶ Point-to-point and Pub/Sub supported
  - ▶ No remote interface, no remote home
    - Container activates MDBs as needed
- Bean Provider responsibilities
  - ▶ Implement *javax.jms.MessageListener* interface
    - *onMessage()* method performs necessary message processing actions
- Application Deployer responsibilities
  - ▶ Associate bean with JMS destinations at deployment
    - Deployment descriptor holds association information



## WebSphere V5 - Integral JMS Message Provider

- J2EE 1.3 requires the messaging engine to be part of the base product
  - ▶ We have chosen to ship a subset of MQSeries and MQ System Integrator (MQSI) to fulfill this requirement
  - ▶ QoS fenced in several areas, one of which is accessibility to Shared Queues
  - ▶ Installation of Integral Provider is optional
  - ▶ Current MQ customers already have a serviceable code base and procedures
  - ▶ Upgrade to full function MQ product is a very small FMID that removes fences
  - ▶ The Integral Provider has very limited configuration capability
- Full JMS QoS will require an MQ license
  - ▶ Support for message sizes greater than 63K is a current MQ issue



## J2EE Version 1.3 messaging requirements

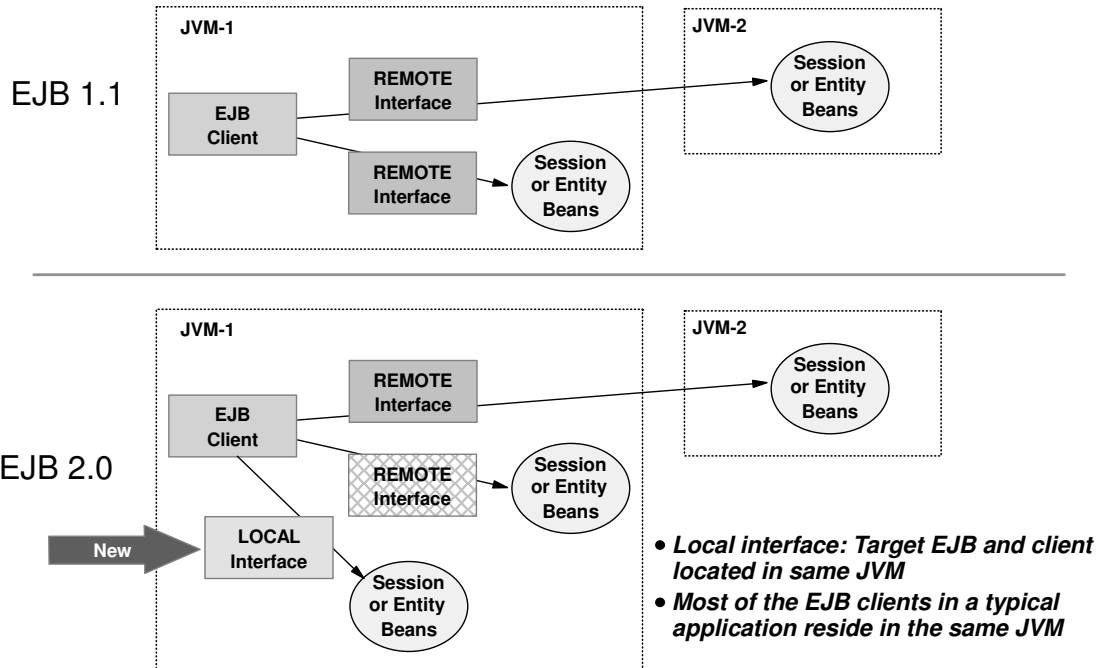
- J2EE Version 1.2 required APIs only
  - ▶ JMS Interface only, no implementation
- J2EE Version 1.3 requires:
  - ▶ JMS Interfaces and Provider (Implementation)
  - ▶ Point to Point & Publish Subscribe
  - ▶ Message Driven Bean (MDB) Support
    - New part of EJB 2.0 spec
    - Application Server can consume JMS messages directly



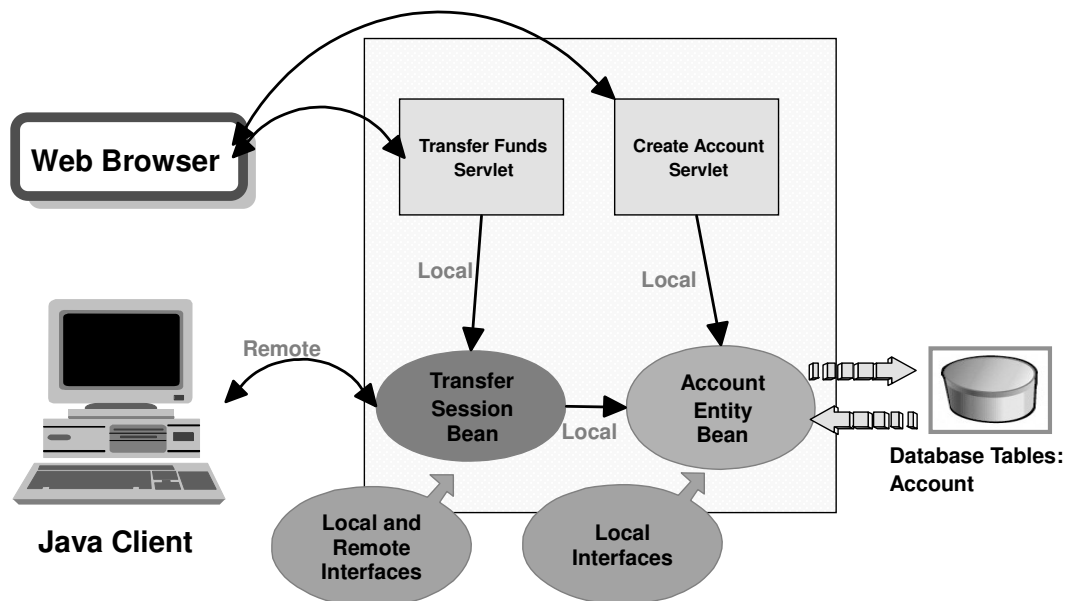
## EJB 2.0: Local interfaces

- Used by clients in the same JVM
- May be more efficient
- Not restricted to serializable types
- Can be cast rather than narrowed
- Arguments passed by reference
  - ▶ must not be stored as part of called bean's state
- Does not involve method permission checks

## EJB 2.0: Local Interfaces

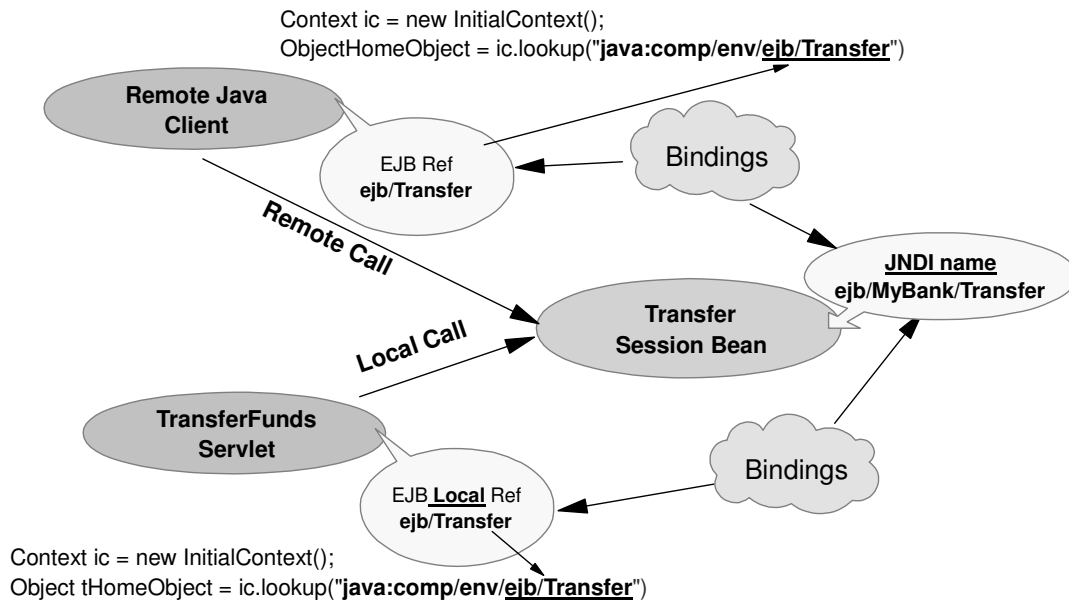


## EJB 2.0: Local Interface Example





## EJB 2.0: Local Interface - Client Calls



## EJB 2.0: Local Interface - Deployment Descriptor

- Local Interface added to EJB Deployment Descriptor (ejb-jar.xml)

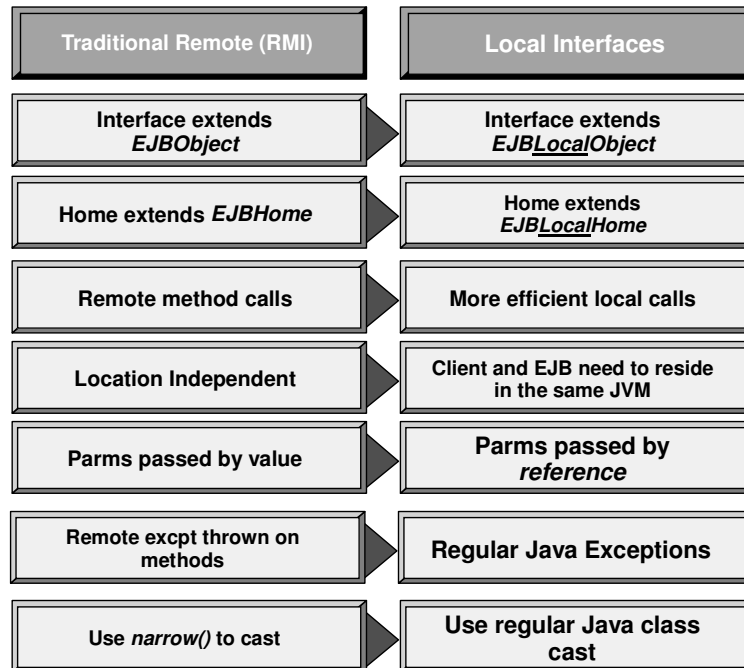
### Sample content of ejb-jar.xml

```

...
<enterprise-beans>
  <entity id="Transfer">
    <display-name>Transfer</display-name>
    <ejb-name>Transfer</ejb-name>
    <home>com.ibm.examples.mybank.ejb.TransferHome</home>
    <remote>com.ibm.examples.mybank.ejb.Transfer</remote>
    <local-home>com.ibm.examples.mybank.ejb.TransferLocalHome</local-home>
    <local>com.ibm.examples.mybank.ejb.TransferLocal</local>
    <ejb-class>com.ibm.examples.mybank.ejb.TransferBean</ejb-class>
    ...
  </entity>
  ...
</enterprise-beans>

```

## EJB 2.0: Local and Remote Interface Comparison

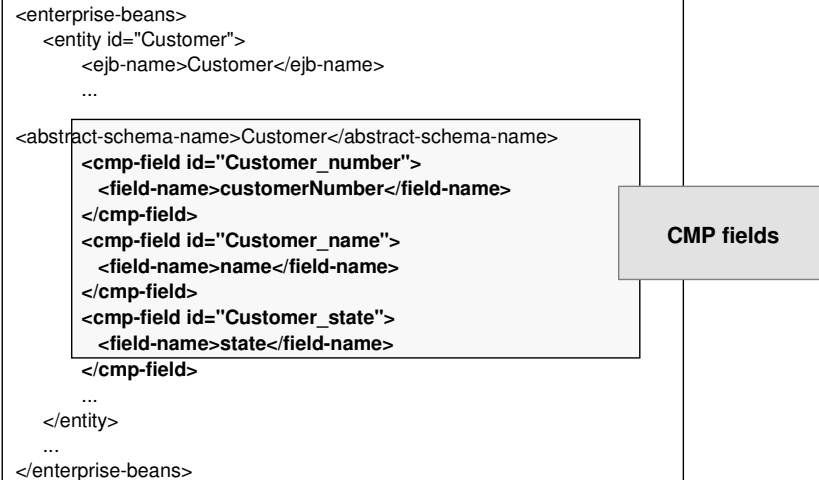


## EJB 2.0: Container Managed Persistence

- EJB 1.1
  - ▶ Persistent data defined by Bean's instance variables
- EJB 2.0
  - ▶ Container supplied Persistent Manager
    - Define more complex relationship
    - Persistence Manager generates the mapping of CMP entities to RDB
- EJB 2.0 CMP bean class
  - ▶ Bean declared as abstract class
  - ▶ No fields in bean class - fields and relationships defined in descriptor
  - ▶ Persistent fields and relationship are accessed using accessor methods (getters and setters)
  - ▶ CMP fields must be Java primitives or serializable
  - ▶ Persistent Manager generates concrete implementation of the abstract bean class
    - Based on the XML deployment descriptor and the bean class

## EJB 2.0 CMP: Deployment Descriptor

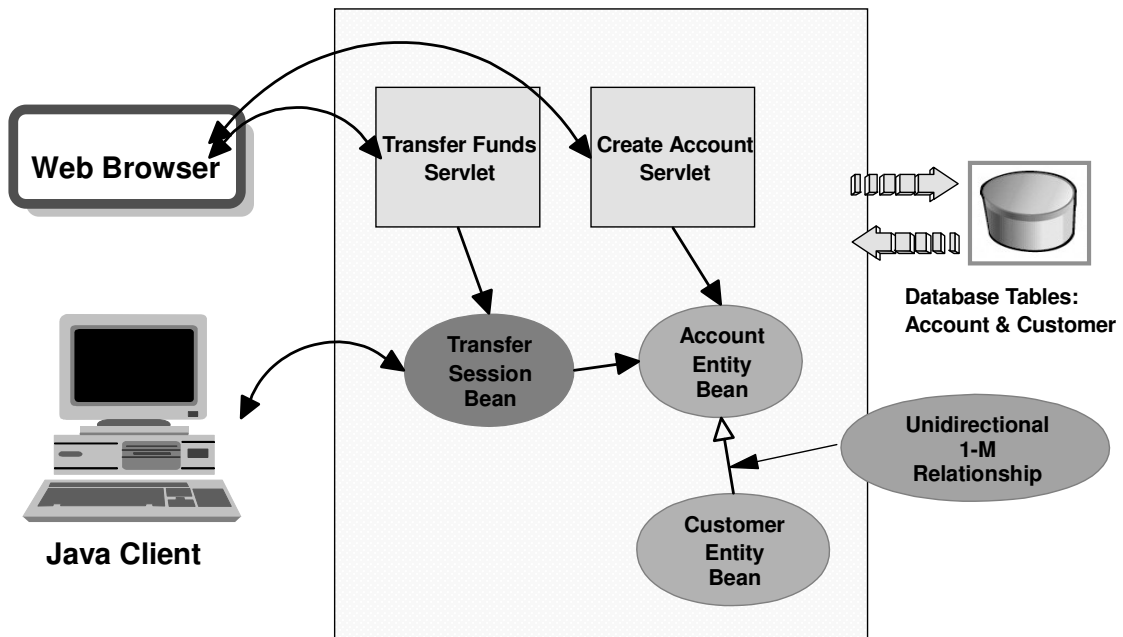
- CMP added to EJB Deployment Descriptor (ejb-jar.xml)



## EJB 2.0: Container Managed Relationships

- Container Managed Relation (CMR)
  - ▶ Allows multiple entity beans to have relationships among themselves
  - ▶ Container manages the relationship (= referential integrity)
  - ▶ One-to-One, One-to-Many and Many-to-Many relationships
  - ▶ Described in Deployment Descriptor
  - ▶ Relationships accessed through setter and getter methods
  - ▶ Members of a CMR must have a local interface
- Benefits:
  - ▶ Allows the developer to create complex relationship and let the container manage those relationships
  - ▶ Container can create more efficient code to interact with the back end systems

## EJB 2.0: CMP and CMR Example



## EJB 2.0: CMR - Deployment Descriptor

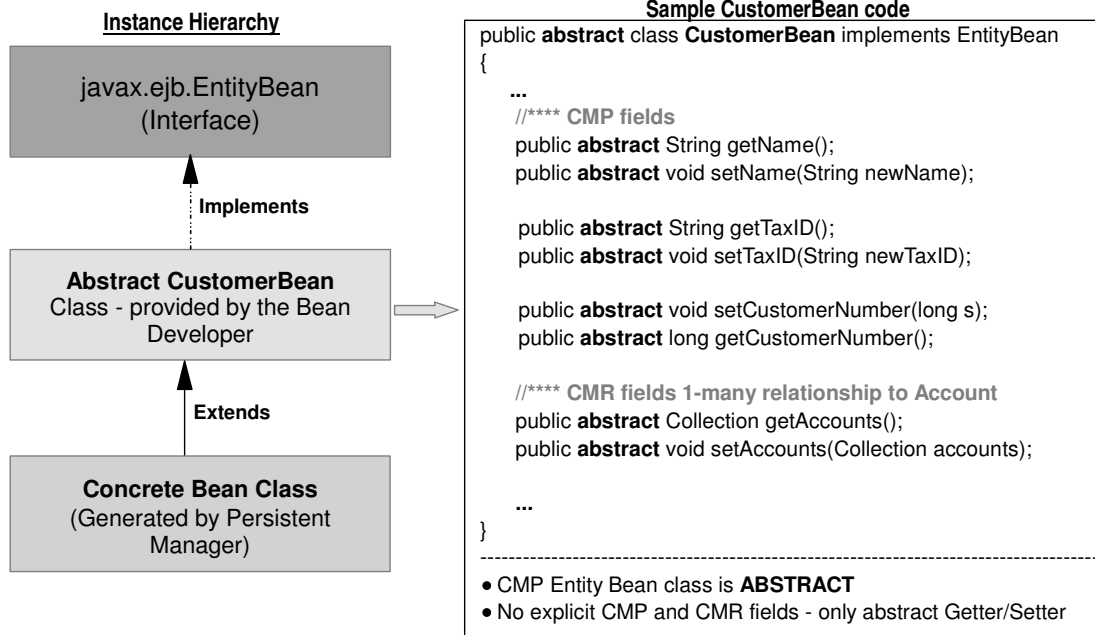
```
<relationships id="Relationships_1">
  <ejb-relation id="EJBRelation_1">
    <ejb-relation-name>CustomerToAccounts</ejb-relation-name>
    <ejb-relationship-role id="EJBRelationshipRole_1">
      <ejb-relationship-role-name>OwnerOfAccounts</ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source id="RoleSource_1">
        <ejb-name>Customer</ejb-name>
      </relationship-role-source>
      <cmr-field id="CMRField_1">
        <cmr-field-name>accounts</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role id="EJBRelationshipRole_2">
      <ejb-relationship-role-name>OwnedAccounts</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <relationship-role-source id="RoleSource_2">
        <ejb-name>Account</ejb-name>
      </relationship-role-source>
    </ejb-relationship-role>
  </ejb-relation>
</relationships>
```

One side of the Relation - "Customer"

Other side of the Relation - "Account"

One-Many Relationship between "Customer" and "Account"  
One Customer can have Many Accounts

## EJB 2.0: CMP/CMR Entity Bean Example



## EJB Query Language (EJB QL)



- EJB QL is a query specification language for the finder and select methods of CMP Entity beans
- Defined in Deployment descriptor
- Based on SQL
  - ▶ Search on the persistent attributes of an EJB and associated bean attributes
- EJB QL is used in Finder methods
  - ▶ Defined in the home interfaces (local or remote)
  - ▶ Returns entity objects (local or remote)
    - There is no need to provide EJB QL for findByPrimaryKey() method
  - ▶ Compiled into SQL at deployment time based on the schema mapping for the bean
- Benefits of EJB QL
  - ▶ EJB QL is independent of the bean's mapping to a RDB
  - ▶ EJB QL is portable

## EJB QL - Deployment Descriptor



- EJB QL specification added to EJB Deployment Descriptor (ejb-jar.xml)

### Example: EJB QL as shown in EJB Deployment Descriptor

```
<query id="Query_1">
  <description>Query to obtain the accounts exceeding a certain balance.</description>
  <query-method id="QueryMethod_1">
    <method-name>ejbSelectAccountsByBalance</method-name>
    <method-params>
      <method-param>float</method-param>
    </method-params>
  </query-method>
  <ejb-ql>
    SELECT a.accountNumber FROM Customer c, IN(c.accounts) a WHERE a.balance > ?1
  </ejb-ql>
</query>
```

## EJB QL - Syntax



- EJB QL query is a string with a SQL-like syntax. It contains:
  - ▶ SELECT clause that specifies the EJB object or cmp field to return
  - ▶ a FROM clause that names the bean collections
  - ▶ an optional WHERE clause that contains search predicates over the collections
  - ▶ Can also contains input parameters that correspond to the arguments of the finder method

- Designates an EJB or a cmp or cmr field
- Supports DISTINCT selections

SELECT <object or ejb field>

- Designates an EJB abstract schema
- In addition, supports navigation to any reachable EJB

FROM <ejb abstract schema, navigational expression>

WHERE <conditions for selection>

- Contains conditional expressions involving cmp/cmr fields
- Supports navigation
- Predicates similar to SQL (including LIKE, IN, BETWEEN, etc.)
- Allows inclusion of substitution parameters



Redbooks

**EJB 2.0 Spec .vs. WebSphere EJB QL**

	<b>EJB 2.0 Spec</b>	<b>WebSphere Query</b>	<b>WS Enterprise Dynamic Query</b>
Bean methods	No	No	Yes
Calendar comparisons	Yes	Yes	Yes
Delimited identifiers	No	Yes	Yes
Dependent Value attributes	No	Yes	Yes
Dependent Value methods	No	No	Yes
Dynamic Query APIs	No	No	Yes
EXISTS predicate	No	Yes	Yes
Inheritance	No	Yes	Yes
Multiple element select clauses	No	No	Yes
Order by	No	Yes	Yes
Scalar functions	Yes	Yes	Yes
Select clause	Required	Optional	Yes
SQL Date/time expressions	No	Yes	Yes
String comparisons	= and <> only	= > <> <	= > <> <
Subqueries, aggregations, group by, and having clauses	No	Yes	Yes



Redbooks

**EJB 2.0: Home methods**

- Not just limited to create and finder methods
- Conceptually similar to 'class' methods
  - ▶ Business logic that is dedicated to the bean class but independent from a particular instance
- Access to the beans attributes is not allowed within the business logic of the home method



## EJB 2.0: inheritance

- The ability for entity beans to inherit from other entity beans
- IBM Extension provided by WSAD V5
- Two different approaches for mapping to data model:
  - ▶ Single Table
    - All beans in inheritance hierarchy map to same table
    - Table not normalized, contains columns for all beans
  - ▶ Root Leaf - Table for each bean in hierarchy
    - Parent table contains columns specific to parent fields and discriminator column
    - Child tables have same key as parent and specific columns
- Home interfaces do not inherit from each other.
- Finder methods on home return instances of correct child beans.



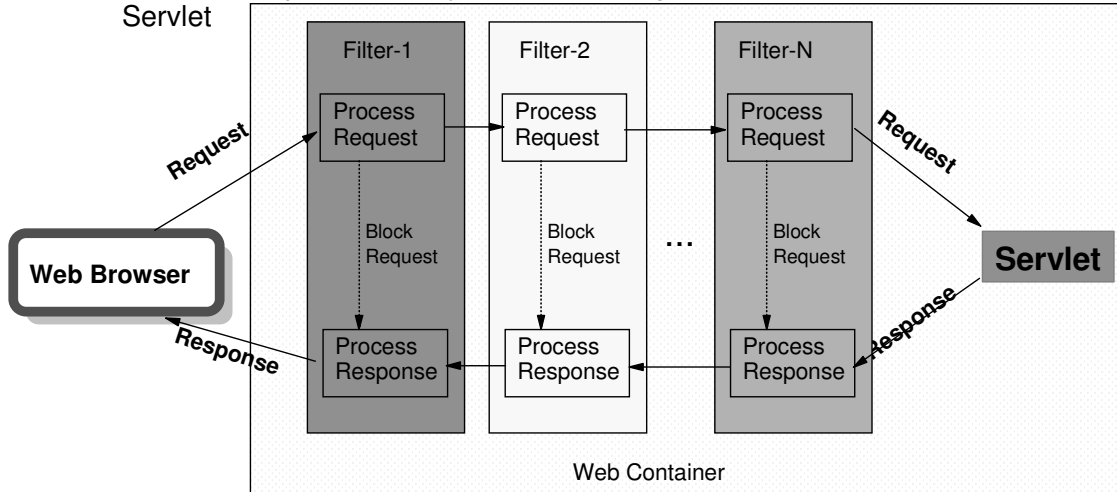
## Servlet 2.3 Changes/Additions

- Servlet Filtering
- Application Lifecycle Listeners and Events
- Enhanced Internationalization Support
- Some API changes



## Servlet 2.3: Filters

- Allows developer to:
  - ▶ Intercept a request before it reaches a servlet
  - ▶ Modify the response after the servlet has processed the request and before the client receives the response
  - ▶ Send the response directly without sending to the next filter or the Servlet



## Servlet 2.3: Filters

- Benefits:
  - ▶ Authentication, Logging and auditing, Encryption
  - ▶ Image conversion, Data compression, Tokenizing filters
  - ▶ Filters that trigger resource access events
  - ▶ XSL/T filters that transform XML content
  - ▶ MIME-type chain filters
  - ▶ Caching filters
- Downside
  - ▶ If any filters fail, the request is not completed



## Servlet 2.3 Filters - Deployment Descriptor

- Filter specification added to Web Deployment Descriptor (web.xml)
  - ▶ Specify filter name, filter class and optional initialization parameters
- For each filter, define the filter mapping
  - ▶ This specifies on which resource(s) to associate the filter
  - ▶ Can be associated with a single Web resource (Servlet, JSP, Static resource), or a group of Web resources (via URI)
- Filters are invoked in the same sequence as defined in the DD

### Example: Filter Definition

```
<filter>
  <filter-name>Image Filter</filter-name>
  <filter-class>com.acme.ImageFilter</filter-class>
</filter>
<filter>
  <filter-name>Logging Filter</filter-name>
  <filter-class>com.sample.LoggingFilter</filter-class>
</filter>
```

### Example: Corresponding Filter Mappings

```
<filter-mapping>
  <filter-name>Image Filter</filter-name>
  <servlet-name>ImageServlet</servlet-name>
</filter-mapping>
<filter-mapping>
  <filter-name>Logging Filter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```



## Servlet 2.3: Application and Event Listener

- Application and Event Listener
  - ▶ Listener objects listen for state changes (lifecycle changes and attribute changes) in the ServletContext and HttpSession objects
  - ▶ Developer provides a list of these listeners (class files) in the Web module (WAR)
  - ▶ Listeners apply to the entire Web module
- Benefits
  - ▶ Allows greater control over interactions with the ServletContext and HttpSession objects
  - ▶ Web developer can monitor the state of the Web application and perform functions
    - Open a database connection for the entire Web application when the ServletContext is created
    - Close the database connection on the shutdown of the ServletContext



## JSP 1.2: Changes/Additions

- XML Views of JSP Pages
  - ▶ New Classes for Tag Library Validation
- New Tag Support for Iteration
- Tag Library Support for Application Lifecycle Events
- Tag Library Lifecycle Improvements



## JSP 1.2: XML Views of JSP Pages

- JSP can be an XML document
  - ▶ Called JSP document
  - ▶ Cannot mix standard syntax and XML syntax in the same jsp file
- Uses the same file extension (.jsp) as a JSP page
- JSP document must have jsp:root as top element
  - ▶ jsp:root cannot appear in a regular JSP page
- Benefits:
  - ▶ Validate JSP document against DTD, XSD
  - ▶ XML tools can manipulate JSP documents
  - ▶ JSP document can be generated from textual representation by applying an XML transformation, such as XSLT
  - ▶ Will become more important as more and more content is authored as XML



## JSP 1.2: XML Syntax Example

```
<html>
<title>positiveTagLib</title>
<body>
<%@ taglib uri="http://java.apache.org/tomcat/examples-taglib" prefix="eg" %>
<%@ taglib uri="/tomcat/taglib" prefix="test" %>
<%@ taglib uri="WEB-INF/tlds/my.tld" prefix="temp" %>
<eg:test toBrowser="true" att1="Working">
Positive Test taglib directive </eg:test>
</body>
</html>
```

Example:  
Standard JSP page

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:eg="http://java.apache.org/tomcat/examples-taglib"
xmlns:test="urn:jsptld:/tomcat/taglib"
xmlns:temp="urn:jsptld:/WEB-INF/tlds/my.tld"
version="1.2">
<jsp:text><![CDATA[<html>
<title>positiveTagLib</title>
<body>

]]</jsp:text>
<eg:test toBrowser="true" att1="Working">
<jsp:text>Positive test taglib directive</jsp:text>
</eg:test>
<jsp:text><![CDATA[
</body>
</html>
]]</jsp:text>
</jsp:root>
```

Example:  
Equivalent JSP  
document



## JSP 1.2: New Tag Support

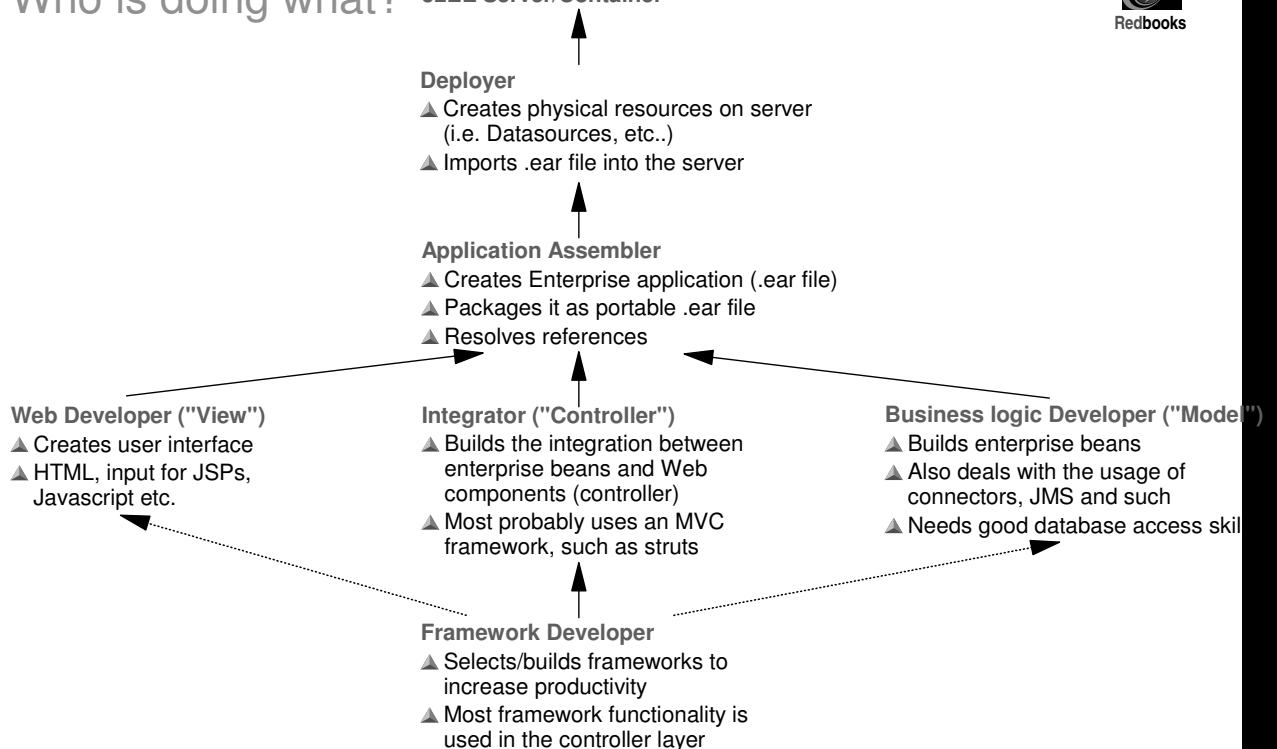
- New Classes for Tag Library Validation
  - ▶ Added in the tag libraries to support validation phase introduced with the support of XML syntax
- Support for Application Lifecycle Events
  - ▶ To support application events support in Servlet 2.3
  - ▶ When processing the web application deployment descriptor at application start time, take specific note of each included directive
- Tag Library Lifecycle Improvements
  - ▶ Add a resetCustomAttributes() method to the Tag interface
    - This will allow the reuse of tag instances in cases where the invocations of the tag do not set the same attributes
- New Tag Support for Iteration
  - ▶ supports iteration without BodyContent
- New TryCatchFinally Interface

## J2EE roles

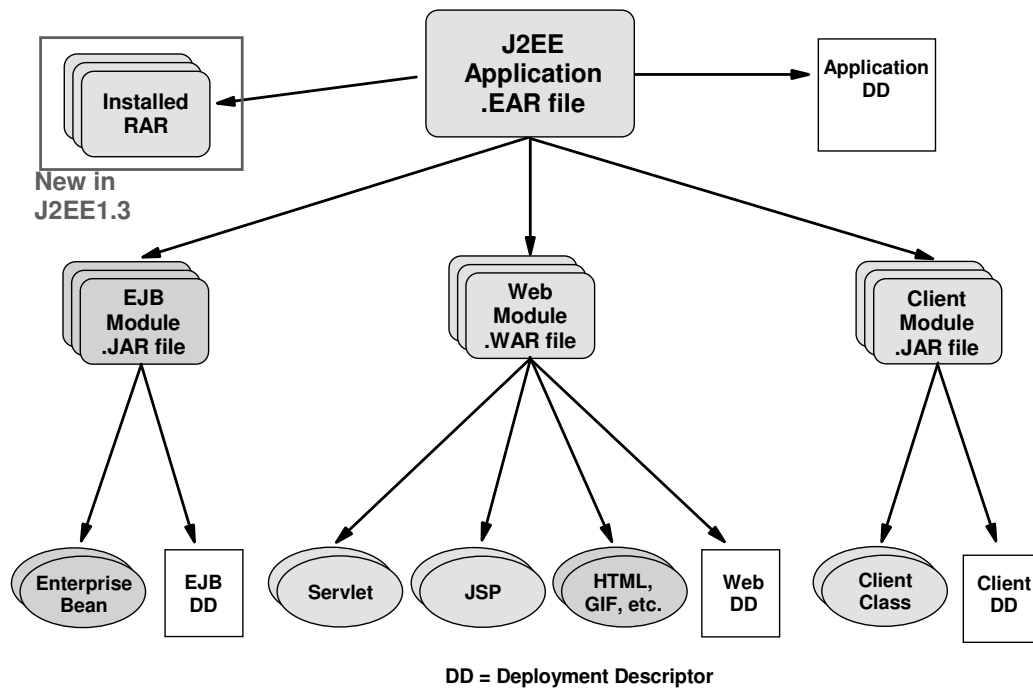


- The development and deployment of J2EE applications requires all kinds of disciplines hardly unified in one person
- We can distinguish between the following roles:
  - ▶ Web developer
  - ▶ Integrator
  - ▶ Business logic developer
  - ▶ Framework developer
  - ▶ Application assembler
  - ▶ Deployer
  - ▶ System administrator
- In smaller projects, one person performs several roles, but in larger projects each person specializes in just one role

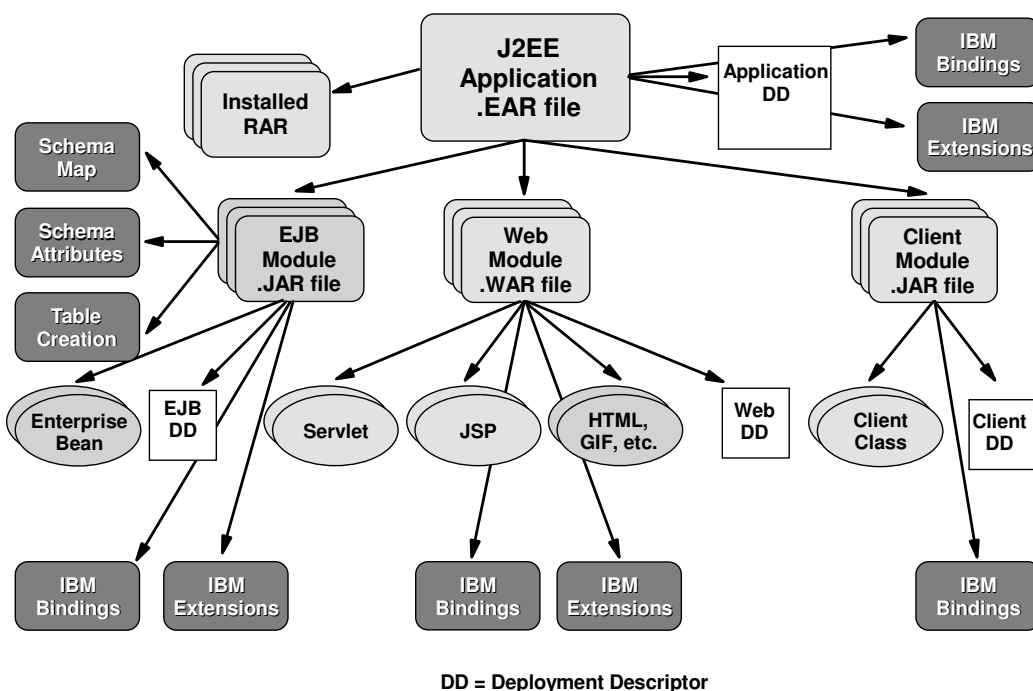
## Who is doing what? J2EE Server/Container



## J2EE 1.3 Enterprise Application Packaging



## WebSphere J2EE 1.3 Application Packaging





## Some notes on the files

- "Applications" are contained in an Enterprise Archive file (.ear file)
- An .ear file is just a .jar file with a specific directory structure and format
  - ▶ can be created using jar command
- An application contains one or more "modules". module types are:
  - ▶ EJBs --> .jar file
    - EJB modules actually contain "components"
  - ▶ Web components --> .war file
- An .ear file contains metadata (i.e. application.xml file) which describes the application content
- J2EE servers receive applications in the form of an .ear file
  - ▶ J2EE servers have containers within them that understand a particular module type
- At this moment, .ear files are not fully binary compatible yet between different environments
  - ▶ reprocessing via the AAT is still necessary
- The content of .ear files can be browsed like a .zip file



## .ear file example

/usr/MyEar

EJB123.jar  
webappABC.war  
myStuff.war

/meta-inf

application.xml  
manifest.mf

```
<?xml version="1.0"
encoding="ISO-8859-1"?>

<application>

  <display-name>MyApp</display-name>

  <module>
    <web>
      <web-uri>webappABC.war</web-uri>
      <context-root> /Payroll </context-root>
    </web>
  </module>

  <module>
    <web>
      <web-uri>myStuff.war </web-uri>
      <context-root> /MyJunk </context-root>
    </web>
  </module>

  <module>
    <ejb>EJB123.jar</ejb>
  </module>

</application>
```