**Enterprise JavaBeans**

Adapted from:
Ruslana Svidzinska
CSE690

---

**What is EJB?**

- An EJB is a specialized, non-visual JavaBean that runs on a server.

- EJB technology supports application development based on a multi-tier, distributed object architecture in which most of an application’s logic is moved from the client to the server.

---

**Server Components**

- Server components are application components that run in an application server.
Java Application Servers

- A Java application server provides an optimized execution environment for server-side Java application components.
- A Java application server delivers a high-performance, highly scalable, robust execution environment specifically suited to support Internet enabled application systems.

WORA

- The Enterprise JavaBeans architecture defines a standard model for Java application servers to support "Write Once, Run Anywhere" (WORA) portability.

Component Portability

EJB technology takes the WORA concept to a new level.
EJB completely portable across any vendor’s EJB compliant application server. The EJB environment automatically maps the component to the underlying vendor-specific infrastructure services.
Overview of EJB Technology

EJB Component Model
- Enterprise JavaBeans component model logically extends the JavaBeans component model to support server component.
- Server components are reusable, prepackaged pieces of application functionality that are designed to run in an application server.
- EJB can be assembled and customized at deployment time using tools provided by an EJB-compliant Java application server.

Simplifying Development.
- The EJB architecture provides an integrated application framework.
- An EJB server automatically manages a number of tricky middleware services on behalf of the application components.
- EJB component-builders can concentrate on writing business logic rather than complex middleware.

Simplifying Development (cont'd)

RESULT:
- Application get developed more quickly
- Code is of better quality.
What EJB Accomplishes

You can take any Java class and with little effort make it a distributed, secure, transactional class.

You can take any data source and make the data source appear to be a collection of Java objects:

- Eliminates distinction between data from a database and any other source.
- All information is accessed through Java objects.
- All SQL is cleanly encapsulated in Java objects.
- True object-oriented programming.
- High reusability.
- Database objects work with the full Java class library.

What EJB Means to Us

- Developers can focus on writing business logic rather than writing low-level infrastructure like data access, concurrency, transactions, threading, etc.
- Reduces development time.
- Reduces complexity.
- Increases quality and reliability.
- The knowledge about EJB is portable among many different products because EJB products are based on a common standard.
- Greater reuse because code is located in shareable, server objects.

So....let’s take a look at Enterprise JavaBeans

- A specification from JavaSoft.
- Enterprise JavaBeans defines a server component model for the development and deployment of Java applications based on a multi-tier, distributed object architecture.
- The Enterprise JavaBeans specification defines:
  - A container model.
  - A definition of the services the container needs to provide to an Enterprise JavaBean, and vice versa.
  - How a container should manage Enterprise JavaBeans.
The EJB architecture specifies the responsibilities and interactions among EJB entities:

- EJB Servers
- EJB Containers
- EJB Beans
- EJB Clients
The EJB Server provides system services and manages resources:
- Process and thread management
- System resources management
- Database connection pooling and caching
- Management API

EJB Container provides a run-time environment for an Enterprise Bean:
- Hosts the Enterprise JavaBeans
- Provides services to Enterprise JavaBeans
  - Naming
  - Life cycle management
  - Persistence (state management)
  - Transaction Management
  - Security
- Likely provided by server vendor

Enterprise JavaBeans:
- A specialized Java class where the real business logic lives
- May be developer-written or tool-generated
- Distributed over a network
- Transactional
- Secure
- Server vendors provide tools that automatically generate distribution, transaction and security behavior.
EJB Clients

- Client access is controlled by the container in which the enterprise Bean is deployed
- Clients locate an Enterprise JavaBean through Java Naming and Directory Interface (JNDI)
- RMI is the standard method for accessing a bean over a network

What's Unique About EJB

Declarative Programming Model

- Mandates a container model where common services are declared, not programmed
- At development and/or deployment time, attributes defining the bean's transaction and security characteristics are specified
- At deployment time, the container introspects the Enterprise JavaBean attributes for the runtime services it requires and wraps the bean with the required functionality
- At runtime, the container intercepts all calls to the object:
  - Provides transactional, threading and security behavior required before the method invocation
  - Invokes the method on the object
  - Cleans up after the call

Understanding EJB Components
**Beans and Applications**

Java Application

- **Java Beans**
- Database Server
- CORBA Server
- Java Server

**Characteristics of Java Beans**

- **Properties**: Both for customization and programmatic use
- **Event-Handling**: Ability for beans to communicate with themselves and with their outside world asynchronously
- **Introspection**: Ability of seeing inside a bean and analyzing what services it provides
- **Customization**: Ability of manipulating the appearance or behavior of a bean while designing the application
- **Persistence**: Capturing the customized state of a bean for later (re) use
- **Security**: Access control based Java security

**Persistence**

- Beans need to support a range of storage behavior for:
  - the use of existing data formats (e.g. OpenDoc)
  - the storage of its internal state
- Beans support two mechanisms:
  - use **Java Serialization Mechanism** which provides an automatic way of storing and restoring the internal state of Java objects
  - use **Java Externalization Mechanism** (extension of above) which gives a class complete control over the writing of its state
- Besides properties (run-time state), beans also need to store customized information (e.g. how to deal with references to other beans).
Events

- Primary communication mechanism for Java Beans
- For plugging together components in application builders
- Some components act as sources and others as listeners of events
- Usually, events propagate state changes from the source object to all its listeners
  - Examples for events are mouse actions and widget updates
  - Listener components need to register to receive events
  - Event notifications are propagated by the event source via implicit method invocation
  - Each Java Bean acts as an event server

Properties

- Named attributes that can affect the appearance or behavior of a bean (e.g., label property)
  - may be exposed in scripting environment (e.g., JavaScript may set label value)
  - may be accessed programmatically by other components
  - may be set during customization of a component
- Properties constitute the state of a bean and may be stored away through Persistence.
- Are always accessed through methods (get/set)
- Properties may be bound or constrained
  - bound properties notify other objects when they are changed (e.g., enable button after input field received value)
  - constrained properties furthermore validate the change

Introspection

- Examine properties, methods, and events that a Java Bean supports (both during design time and run time)
- Used for dynamic invocation
- JavaBeans provides an Introspector Class which analyzes other beans
  - uses reflection and design patterns to infer behavior
  - design patterns infer the usage of specification templates (e.g. names; public void add<EventListenerType>(<EventListenerType> a))
- BeanInfo interface may be used to capture additional bean developer information
Customization

- Ability to customize the appearance and behavior of beans
  - GUI property sheet for viewing Bean design properties
  - Property editor may be used to manipulate them
  - Wizards to lead programmers through the customization process
- Provide default appearance and behavior without customization
- Customization classes may not be required at run-time

Basic Java Beans requirements

- Properties: Named attributes associated with a bean that can be read or manipulated through methods e.g. foreground property
  - Use getXYZ() and setXYZ() methods to access properties
- Methods: Beans export regular methods just as class libraries e.g. paint()
  - Methods may be accessed by other components or environments.
- Events: Provide another mechanism for one component to notify another component that something has happened.
  - Event listener may be associated with event sources.

Internal Architecture of EJB

- Home Interface
  - Object creation, removal, location
  - Finder methods
- Remote Interface
  - Business methods
- Bean
  - Implement business logic
- Container classes
  - Stubs and skeleton
JavaBeans vs Enterprise JavaBeans

- Enterprise JavaBeans is a framework for building and deploying server-side Java components
- JavaBeans is a framework for client-side Java components
- Conceptually related because both are components
- The specifications are different
- The specifications do not build on each other or rely on each other

What's in the EJB Specification

~200 Pages of technical material for EJB vendors

- Goals for the Release
- Roles and Scenarios
- Fundamentals (Scope of EJB)
- Session and Entity Beans
- Transactions, Exceptions, Distribution
- EJB Bean and Container Responsibilities
- API Reference

Enterprise EJB Scenario

Clients
  Browser
  Servlet
  Application

Web Server

EJB Application Server

EJB Container

Shopping Cart

Inventory

Credit Card

EJB Server

Existing Enterprise Middleware

CICS Programs

Databases

SAP Modules
An inside look at the various components of EJB

- Bean Class is written by the developer
- EJBHome and EJBObject interfaces and classes control access to the Bean class
- Deployment Descriptor and MANIFEST describe security and transactional characteristics of the Bean

EJBHome Interface and Class

- Used to get a reference to a bean’s remote interface
- Provides bean creation services
  - myFoo = fooHome.create() instead of myFoo = new foo()
  - Supports multiple signatures to create EJB instances
- Similar to class factory in COM and CORBA
- May be generated by tools that come with an EJB server
- Also manages EJB:
  - querying (Entity Bean)
  - deleting (Entity Bean)

Interface javax.ejb.EJBHome

- Home objects must implement this interface
- Returns a reference to a bean by creating or finding it
- Every bean has a EJBHome interface that provides methods for getting references to one or more beans
  - create methods are used to create new beans
  - there can be many create methods, similar to a class having many constructors
- Provides a bean removal interface also
- The EJBHome Class implementation is provided by the EJB Server Provider
Sample EJBHome Interface

```java
public interface CustomerHome extends EJBHome {
    public abstract Customer create(String id, String name) throws RemoteException, CreateException;
    public abstract Enumeration findByName(String val) throws RemoteException, FinderException;
    public abstract RemoteEnumeration findStateByName(String val) throws RemoteException, FinderException;
    public abstract Customer findByPrimaryKey(CustomerKey pkey) throws RemoteException, FinderException;
    public abstract Customer findByPrimaryKey(CustomerKey pkey, int findSource) throws RemoteException, FinderException;
    public abstract CustomerState findStateByPrimaryKey(CustomerKey pkey, int findSource) throws RemoteException, FinderException;
}
```

Client makes invocation to EJBHome Class via the EJBHome Interface

- Client calls one of the create() methods on the EJBHome Class. The EJBHome class can have multiple create() signatures.

EJBHome Class instantiates the Bean Class

- When the create() method is called in the EJBHome Class, it is responsible for instantiating the Bean class.
**EJBObject Interface and Class**

- Intercepts calls to the EJB Class to add support for:
  - transactions
  - security
  - threading
- EJBObject class has the same methods as the bean and delegates to the bean for actual behavior
- EJBObject class checks security and sets up transaction before delegating method call to the bean
- Clients can never get a reference to a bean’s EJB Class, only the EJBObject interface

**Interface javax.ejb.EJBObject**

```java
public interface javax.ejb.EJBObject extends java.rmi.Remote {
    EJBHome getEJBHome() throws RemoteException;
    Object getPrimaryKey() throws RemoteException;
    boolean isIdentical(EJBObject obj) throws RemoteException;
    void remove() throws RemoteException, RemoveException;
}
```

- Represents a specific bean instance
- Remote objects must implement this interface
- Primary key is an object that represents the primary key for a specific instance of a bean
- The EJBObject Class implementation is provided by the EJB Server Provider

**Sample EJBOBject Interface**

```java
public interface Customer extends EJBObject {
    public abstract String getId() throws RemoteException;
    public abstract String getName() throws RemoteException;
    public abstract void setName(String val) throws RemoteException;
    public abstract boolean getNameNull() throws RemoteException;
    public abstract boolean getIdNull() throws RemoteException;
    public abstract Enumeration getAccounts() throws RemoteException;
    public abstract void addToAccounts(Account relInst) throws RemoteException;
    public abstract void removeFromAccounts(Account relInst) throws RemoteException;
}
```
After the EJBHome Class instantiates the Bean...

- The EJBHome Class will instantiate the EJBObject Class, initializing it with the remote object reference to the Bean Class.
- The Client will now communicate to the EJBObject Class. The EJBObject Class will delegate the call to the Bean.

Client now has reference to the EJBObject Class.
NOT A REFERENCE TO THE BEAN ITSELF!

EJB Object Class

---

Client communicates to the Bean class via the EJBObject Class

- Client never has a direct reference to the Bean.
- The EJBObject Interface is the interface for the Bean, so the EJBObject Class "delegates" the calls to the Bean.
- The EJBObject Class implementation will be generated using vendor tools.

---

EJB Class

- A bean has a single Java class at its core
  - This class is written by a developer if it's a session bean
  - This class is sometimes generated by a tool if it's an entity bean
- Implements application-specific business logic
- Implements one of the following contracts:
  - javax.ejb.EntityBean
  - javax.ejb.SessionBean
- These contracts provide for consistent behavior when activating beans, passivating beans, reading data, writing data
- Every container can expect these methods in every bean
Deployment Descriptor

- Allows you to declare transaction and security attributes, NO PROGRAMMING REQUIRED!
- An EJB Deployment Descriptor describes the classes, interfaces and declarative behavior of an EJB
- Deployment Descriptor format is serialized objects:
  - javax.ejb.deployment.ControlDescriptor
  - javax.ejb.deployment.DeploymentDescriptor
  - javax.ejb.deployment.EntityDescriptor
  - javax.ejb.deployment.SessionDescriptor
- The deployment descriptor is generated by server tools
- The deployment descriptor classes are defined in the EJB specification

Let's look at the value of the Deployment Descriptor

- **Step #1** - Write your Bean implementation.
- **Step #2** - Compile this Java source into Java bytecode
- **Step #3** - Developer is responsible for creating a serialized deployment descriptor for the bean. Most likely using a tool from the vendor. EJB tools will use Reflection API on the compiled EJB Bean to determine:
  - name of Bean class
  - methods
  - parameters
  - return values
- **Step #4** - The EJB Tool will use the above information to generate a Deployment Descriptor file and an editor with which to set transaction and security attributes.

Sample Contents of a Deployment Descriptor

```xml
<PS(TicketDemo,Concert,deployment_descriptor)
Entity{
  VersionNumber
    1.0
  BeanHomeName
    ConcertHome
  ControlDescriptor{
    IsolationLevel
      TRANSACTION_READ_COMMITTED
    Method
      int getId() throws java.rmi.RemoteException
    RunAsMode
      SYSTEM_IDENTITY
    TransactionAttribute
      TX_SUPPORTS
  }
}
```
Enterprise JavaBean Packaging

- Enterprise JavaBeans are comprised of many Java files
- These files are put in a JAR file
  - A JAR file is a ZIP file with a MANIFEST that describes the contents of the file
  - A MANIFEST is a simple text file
    - Name: bank/AccountDeployment.ser
      - enterprise-bean: True
- A JAR file can contain more than one Enterprise JavaBean

Bean Development Process

- Implement the EJB Class
- Specify the remote interface
- Specify the home interface
- Specify security and transactional characteristics using vendor tools (DeploymentDescriptor)
- Use vendor tools to generate supporting code and package components in EJB-jar
- Iterate...

Enterprise Beans:
Session Beans & Entity Beans

- Session Beans vs. Entity Beans
- Bean-Managed Persistence
- Container-Managed Persistence
- EJB Transaction Attributes
Types of EJB

- Session Bean
  - Stateless Session Bean
  - Stateful Session Bean
- Entity Bean
  - Entity Bean with Bean-Managed Persistence
  - Entity Bean with Container-Managed Persistence
- Message Driven Bean

Comparing Session and Entity Beans

**Session Beans**
- Mandatory for EJB 1.0
- Represents a specific client (1 instance per client)
- Short-lived
- Transient
- Can be any Java class
- May be transactional
- Business Logic Beans

**Entity Beans**
- Optional for EJB 1.0
- Represents underlying data object or context (clients share instance)
- Long-lived
- Persistent
- Can be a class that maps to persistent data (e.g., database)
- Always transactional
- Beans which represent data

Session Beans

**Represents Process**
- A transient agent for an individual client that executes on a server (e.g., ShoppingCart)
- Session beans are often a client of multiple entity beans
- Implements javax.ejb.SessionBean interface
- State management types for session EJBs
  - stateful - session bean may maintain state information across method calls
  - stateless - session bean may be used to service multiple clients
  - a stateless session bean can only have a single no-argument create() method in its Home interface
Session Beans - Stateful or Stateless

- A Stateful Session Bean maintains a one-to-one relationship with a Client. It maintains a user "session". Most common example is an e-commerce application with a "shopping cart" unique for each user.
  - Container will automatically "swap out" the Session bean if it is inactive. Here the container calls the ejbPassivate() method to save any private data to some physical storage.
  - When container receives new request, the Container will call the ejbActivate() method to restore the Session Bean.

- A Stateless Session Bean can be accessed by multiple incoming clients and keeps no private data. It does not maintain a unique session with a client.
  - Keeps no persistent data. If it crashes, container simply starts another one and the client transparently connects.
  - All access to the Bean is serialized.

A Typical Session Bean Has

- A Remote Interface defining the business methods a client will call
- A Home Interface for lifecycle management
- An enterprise bean class

Session Bean Remote Interface (From Sun)

```java
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.math.*;

public interface Converter extends EJBObject {
    public BigDecimal dollarToYen(BigDecimal dollars) throws RemoteException;
    public BigDecimal yenToEuro(BigDecimal yen) throws RemoteException;
}
```
The Home Interface

```java
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
public interface ConverterHome extends EJBHome {
    Converter create() throws RemoteException, CreateException;
}
```

The Session Bean Class

```java
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.math.BigDecimal;
public class ConverterBean implements SessionBean {
    BigDecimal yenRate = new BigDecimal("121.6000");
    BigDecimal euroRate = new BigDecimal("0.0077");
    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }
}
```

Life Cycle of Statefull Session Bean

- Does not exist
- From Sun
- Client calls create
- Container calls ejbCreate
- Container calls setSessionContext
- And then calls ejbCreate
- Ready for business methods to be called
- Container calls ejbPassivate before passivating
- Passive
- After activation the container calls ejbActivate
- Client calls remove, container calls ejbRemove
Life Cycle of a Stateless Session Bean (from Sun)

- Does not exist
- setSessionContext
- ejbCreate
- Ready
- ejbRemove

Entity Bean

**Represents Data**
- Implements `javax.ejb.EntityBean` interface
- Maps a data source to a Java class
  - table, view, join or stored procedure in a relational database
  - a set of related records in a database
  - legacy data
- Each instance of an entity bean is one row of data
- Each instance of an entity bean is uniquely identified by a primary key
- An Entity Bean can also have additional methods for business logic, etc.

Defining an Entity Bean

- An Entity Bean must implement the `javax.ejb.EntityBean` interface. In addition, the Entity Bean must implement an `ejbCreate()` method for each `create()` method on the Home Interface.
- When the Home class gets a call to `create()`, it calls the `ejbCreate()` method on the Entity Bean with the corresponding signature.
- `ejbStore()` and `ejbLoad()` used to synchronize Entity Bean data with the database.
- `ejbActivate()` and `ejbPassivate()` used for notification that Entity Bean state is being written to the database or restored from the database.
Since Entity Beans represent Data...

- Each instance of an Entity Bean is uniquely identified by a primary key object
- Primary key can be saved and used later to regain access to the same EJB object

```java
fookey = fooBean.getPrimaryKey();
fooBean = fooHome.findByPrimaryKey(fookey);
```

Entity Bean Operations

- `BeanHome.create()` is equivalent to an SQL `INSERT`
  - Causes a new database record to be inserted into the database:
    ```java
customerBean = customerHome.create("Jane", "Rodgers");
```
- `BeanHome.findSomething()` is equivalent to an SQL `SELECT`
  - Creates a bean instance that represents queried data from a database
  - Can also return an enumeration to represent multiple rows of data
  - There will typically be many `findSomething()` methods:
    ```java
customerBean = customerHome.findByName(firstName, lastName);
```

Entity Bean Operations

- `BeanHome.remove()` and `EJBObject.remove()` are equivalent to an SQL `DELETE`
  - Causes the database record to be deleted from the database
    ```java
customerBean = customerHome.findByName("Jane", "Rodgers");
customerBean.remove();
```
- Committing a change on a found object is equivalent to an SQL `UPDATE`
  ```java
customerBean = customerHome.findByName("Jane", "Rodgers");
customerBean.setEmailAddress("jane.rodgers@yahoo.com");
```
EJB Persistence

- Provides Entity Beans the ability to store and retrieve their state
- Can be implemented by a bean
  - Bean Managed Persistence
  - Can be implemented by a container
  - Container Managed Persistence

Bean Managed Persistence

- The entity bean is responsible for its persistent behavior
- EJB developer must implement database access
  - ejbCreate(...), ejbLoad(), ejbStore(), ejbRemove()
- Not automated, developer manually creates mapping through JDBC calls
- Not as reusable
  - Hard-code database access within class
- Advanced features like connection pooling and caching are difficult to support because of reliance on hand written code

Container Managed Persistence

- The EJB container is responsible for persistence
- The container provides tools that generate code in the EJB class that maps methods in the bean to a result set
  - Can map to a table, view, join or stored procedure in a database
  - Server provides automated mapping to convert relational data to bean instances
- Advanced features like connection pooling and caching are easily supported
- High reuse
EJB Transaction Support

- EJB allows you to either DECLARE transactions or CODE THEM EXPPLICITLY.
  - TO DECLARE: EJB allows you to declare transaction support in the Deployment Descriptor and then have the container take care of the details inside the remote class.
  - Transaction attributes specified at the bean and method level.
- FOR EXPLICIT CONTROL: An EJB Container is required to implement java.transaction.UserTransaction.
  - Provides an interface to a transaction service.
  - Can be used by EJB clients and enterprise beans that are declared to manage their own transactions (TX_BEAN_MANAGED).
- EJB specification does not support nested transactions.

EJB Transaction Attributes

- TX_NOT_SUPPORTED
  - Cannot execute within a transaction.
  - Suspends any existing transaction while method is called.
- TX_SUPPORTS
  - Executes with or without a transaction.
  - Doesn’t suspend existing transaction.
- TX_REQUIRED
  - Executes within a transaction.
  - If no transaction exists, starts a new one and commits it when method completes.
- TX_REQUIRES_NEW
  - Always starts a new transaction and commits it when method completes.
  - Suspends existing transaction.
- TX_BEAN_MANAGED
  - The bean manages its own transaction control.
  - Suspends existing transaction.
- TX_MANDATORY
  - Must execute within a transaction.
  - If no transaction exists, throws the TransactionRequiredException.

Example of EJB Application

- It consists of number of clients accessing session beans and entity beans.
- Each Session bean provides specialized processing on behalf of client.
  e.g. Travel Agent session bean makes travel reservations while Flight Scheduler bean schedules planes to fly on various routes.
- Each Entity Bean represent different type of business entity.
  e.g. Passengers, seats, planes, flights are entity beans.
Example EJB Application

Message Driven Bean

- Handles asynchronous messages
- Normally acts as a JMS message listener
- The message may have originated from an application client, another enterprise bean, a web component or a non-Java application that can generate messages
- Like a stateless session bean but with no interfaces
- All operations within onMessage may be in a transaction context (the message is redelivered on rollback)

From Sun
On The Client

// locate the connection factory and queue

connectionFactory =
(ConnectionFactory) jndiContext.lookup("java:comp/env/jms/MyConnectionFactory");

destination =
(Queue) jndiContext.lookup("java:comp/env/jms/QueueName");

// Next, the client creates the queue connection, session, and sender:

connection = connectionFactory.createConnection();
session = connection.createSession(false,
  Session.AUTO_ACKNOWLEDGE);
messageProducer = session.createProducer(destination);

// Finally, the client sends several messages to the queue:

message = session.createTextMessage();
for (int i = 0; i < NUM_MSGS; i++) {
  message.setText("This is message " + (i + 1));
  System.out.println("Sending message: " +
    message.getText());
  messageProducer.send(message);
}

JMS Message Types

JMS Message Types
Message Type Body Contains

TextMessage
A java.lang.String object (for example, an XML document).

MapMessage
A set of name/value pairs, with names as String objects and values as
primitive types in the Java programming language. The entries can be
accessed sequentially by enumerator or randomly by name. The order
of the entries is undefined.
BytesMessage
A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

StreamMessage
A stream of primitive values in the Java programming language, filled and read sequentially.

ObjectMessage
A Serializable object in the Java programming language.

On the server

```java
public void onMessage(Message inMessage) {
    TextMessage msg = null;
    try {
        if (inMessage instanceof TextMessage) {
            msg = (TextMessage) inMessage;
            System.out.println("MESSAGE BEAN: Message received: " + msg.getText());
        } else {
            System.out.println("Message of wrong type: " + inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        e.printStackTrace();
        mdc.setRollbackOnly();
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

Presentation Layer

- Uses:
  - Servlets
  - Java Server Pages (JSPs)
  - HTML
  - ...
Overview of Servlets

- Are container managed web components
- Replace Common Gateway Interface (CGI) or Active Server Pages (ASP)
- Generate dynamic response to requests from web based clients
- Synchronize multiple concurrent client requests
- Serve as client proxies

Servlet Operation

- Server is Java program that runs as separate thread inside servlet container.
- Servlet container is part of web server
- It interact with web client using response request paradigm

JavaServer Pages (JSP)

- Text based documents describe how to process a request and create a response
- Contains HTML or XML and other JSP elements defined by JSP specification.
- Are installed on web server
- Are web components that sit on top of java servlet mode.
JSP Advantages

- **Performance**
  - Runtime characteristics of servlets
  - Automatic recompilation of modified pages
  - Server side processing

- **Programming**
  - Emphasize use of reusable components
  - Write Once, Run Anywhere properties
  - Extensible through custom tag libraries
  - Provides front end access mechanism to EJBs

Parts of JSP Pages

**Directive**
```java
<%@ page import="java.util.*, MVCApp.Cart, MVCApp.CartItem" %>
```

**Declaration**
```java
<%! Iterator it = null; CartItem ci = null; Vector cpi = null;%>
```

**Raw HTML**
```html
<html><head><title>Shopping Cart</title></head></html>
```

**Action**
```jsp
<jsp:useBean id="Cart" scope="session" class="MVCApp.Cart"/>
```

**Scriplets**
```jsp
Cpi = cart.getCartItems();
while (it.hasNext()) jItem = (CartItem)it.next();
```

**Expression**
```jsp
<td><%= ci.getTitle()%></td>
<td align="right"><%= ci.getQuantity()%></td>
```

**Implicit Objects**
```jsp
<% string action = request.getParameter("action"); %>
```