Session: 2

Enterprise JavaBeans

Objectives

- Describe the principles of component-based development
- Define Enterprise JavaBeans
- List the characteristics of enterprise JavaBeans
- Describe the evolution of enterprise JavaBeans
- Explain the features of EJB 3.0
- Explain the different types of Enterprise JavaBeans
- Explain the JNDI service on Java EE platform
- Explain JNDI APIs
- Describe the various roles involved in EJB application development
- Explain the various steps involved in developing and packaging an enterprise application
- Describe tools used for developing enterprise application

Introduction 1-2

- Enterprise applications are developed to fulfil the needs of domains such as banking, finance, and so on involving:
  - Large numbers of users
  - Complex requirements

- Enterprise application uses Enterprise JavaBeans (EJBs) as business components on the middleware tier of the multitiered application.

Introduction 2-2

- The implementation of EJBs on the middleware tier is:
  - Laid by the EJB specification provided by Sun Microsystems
  - Based on **component-based** development framework.

Principles of Component-Based Development 1-2

Component-Based Development

- They have an interface defined so that other application components can access it.
- They have a well defined lifecycle mechanism.
- They are configurable.
- They have a third-party integration scheme.
- Components can be assembled with other program blocks to build a complete application.
- Components are portable and reusable.
- Components can function independently or when assembled with other components.

Principles of Component-Based Development 2-2

- Advantages of component-based applications are:
  - **Reduction of cost and time in building large complicated systems:**
    - As there is a reuse of components.
  - **Better quality of software is ensured:**
    - As the components are tested and stabilized through multiple iterations of testing.
  - **A developer creating an application component does not require knowledge of the entire application:**
    - Since, the components are independently developed.

Enterprise JavaBeans (EJBs) 1-2

- EJBs are server-side components of an enterprise application.
- EJB technology provides a platform for developing portable, reusable, and scalable business applications.
- EJB components are deployed into a container which provides generic services to these components.
- Containers services include such as security, persistence transaction management, and so on.

Enterprise JavaBeans (EJBs) 2-2

- There are two ways of looking at EJBs:
  - **EJB are specifications**
    - Lay out rules and standards on how you should code your EJBs.

  - **EJBs are Java interfaces**
    - Are exposed to the EJB clients to access the implemented bean code.

Characteristics of Enterprise Java Beans

- Following are the characteristics of Enterprise Java Beans:

Implements business logic of the application

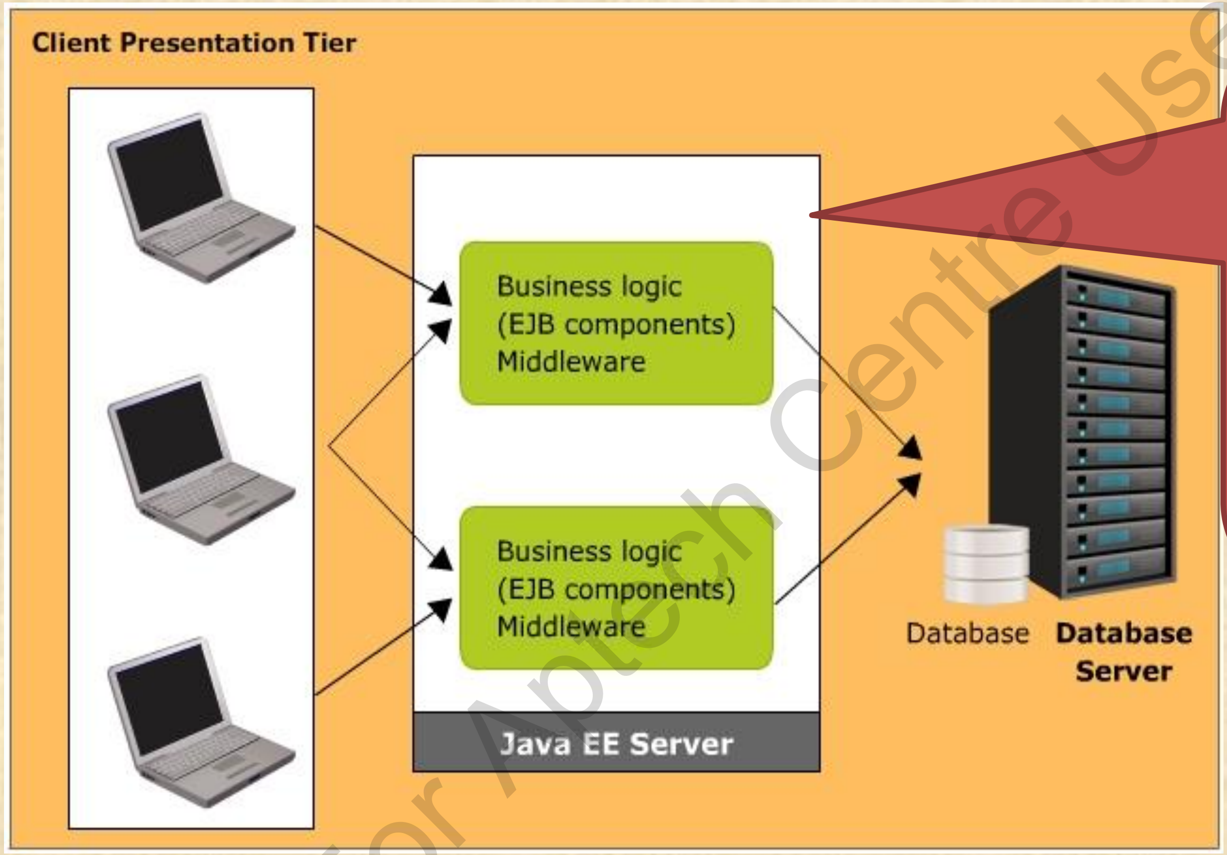
Deployed in the EJB container on the Java-enabled application server

Configured through deployment descriptors which defines the configurations of the bean

Communicates with the application clients through interfaces

EJB as a Component 1-2

- Following figure shows different components in an enterprise application:

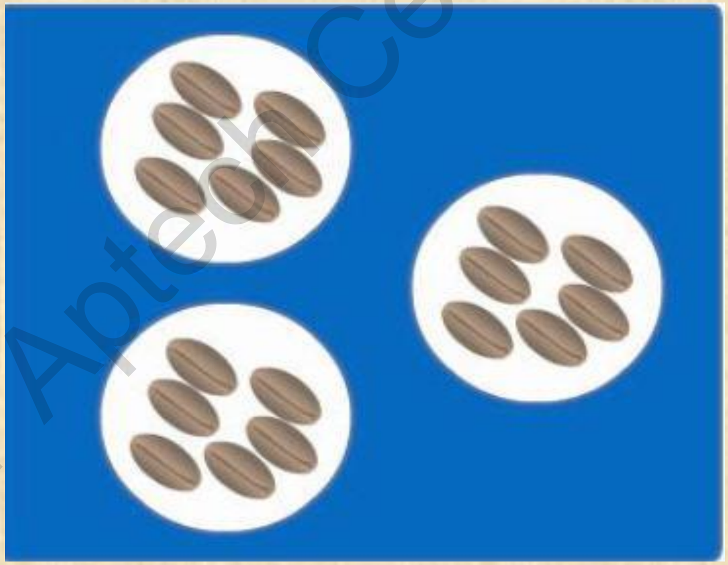

Client Presentation Tier
Business logic (EJB components) Middleware
Business logic (EJB components) Middleware
Java EE Server
Database
Database Server

EJB components exists in a container.

The container and the components together can be viewed as a framework.

EJB as a Component 2-2

- When an EJB is deployed in the EJB container, the EJB container creates multiple instances of the EJB to serve multiple clients.
- Collection of bean instances or beans in the EJB container is referred as bean pool.
- Following figure shows a container with bean pools:



Evolution of EJB 1-2

EJB 1.0	• Session beans • Entity beans
EJB 1.1	• Introduced XML based deployment descriptors
EJB 2.0	• Introduced local interfaces • Message-driven beans
EJB 2.1	• Introduced timer service and Web services

Evolution of EJB 2-2

EJB 3.0

- Annotation based programming model
- Java Persistence API instead of entity beans

EJB 3.1

- No-interface view
- Singleton pattern of development
- Java EE profile

EJB 3.2

- Enhancements in timer service
- Enhancements in managing bean lifecycle methods

EJB 3.0

- Some of the important features introduced in EJB 3.0 are as follows:
  - Use of annotations
  - Callback Methods
  - Elimination of Home interface
  - Elimination of component interface
  - Dependency Injection
  - Interceptors
  - Java Persistence API (JPA)
  - Timer service

Use of Annotations 1-2

- EJB 3.0 uses metadata annotations to specify the services used by the EJB components.
- Metadata annotations simplify the development and testing of the application.
- Annotations:
  - Enable developer to provide the specification based on which code is added.
  - Processed at compile time.

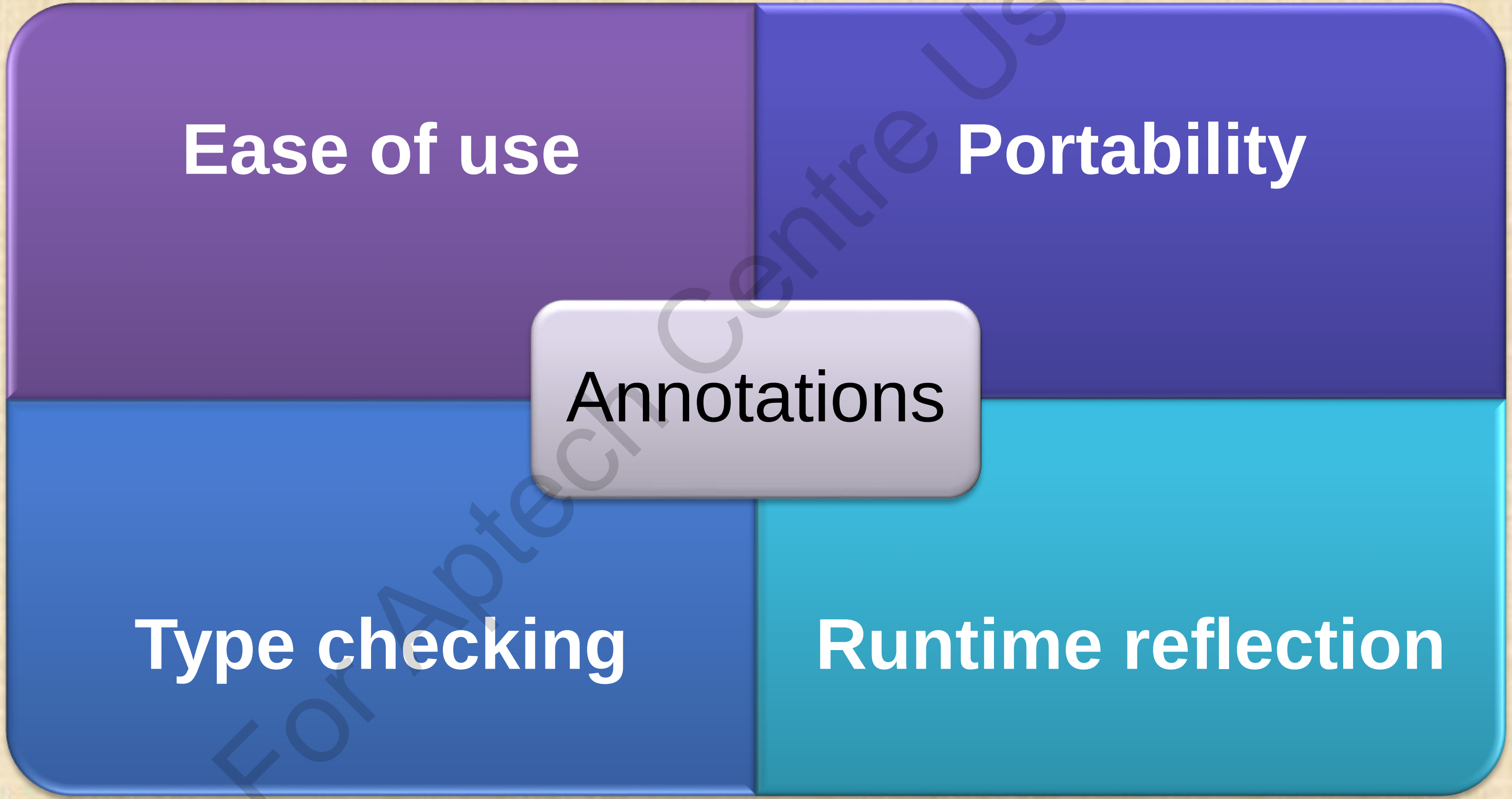
- Following code snippet shows the use of annotations:

```
...
@Stateful
public class HelloBean
implements
HelloInterface {

@Remove
public void removeBean()
   {
   //close all resources
   }
}
...
```

Use of Annotations 2-2

- Following are the advantages of using annotations:



Dependency Injection

- Means through which the container creates the required operational environment for the application.
- Required resources are injected based on annotations and deployment descriptors.
- Annotations used are:
  - `@Inject` and `@Resource` annotations.
- Dependency injection is also supported through JNDI lookup.

Callback Methods

- Callback methods are those methods which are invoked when various lifecycle events with respect to the enterprise bean occur in the application.
- Following are different callback methods defined in EJB 3.0:

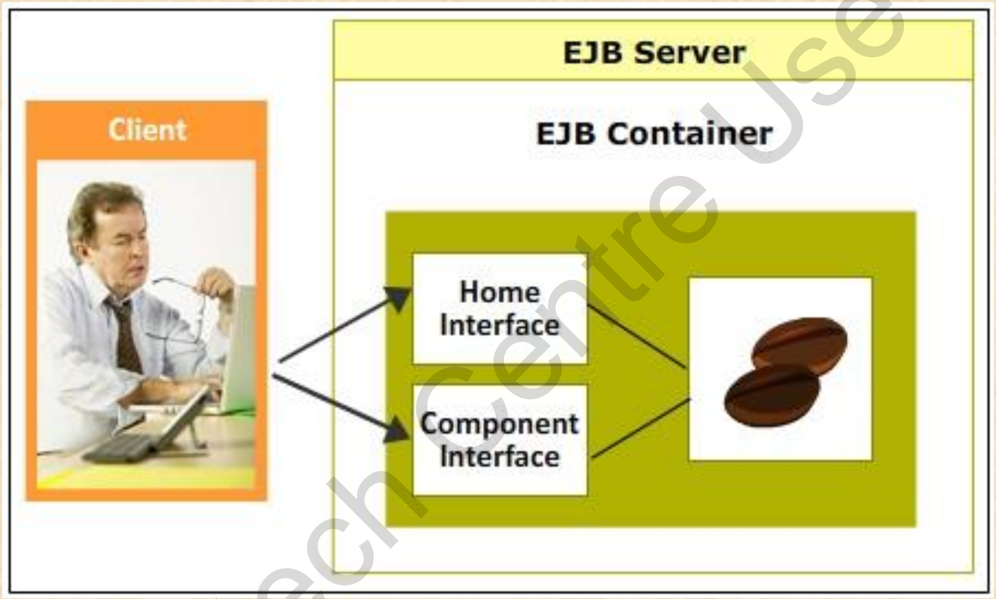
ejbActivate ejbPassivate

ejbLoad ejbStore

- Another change introduced in EJB 3.0 is that any method can be designated as callback method to listen to lifecycle events.

Elimination of Home and Component Interface 1-2

- Following figure shows the architecture of EJB 2.0 application:



- EJB 3.0 has simplified the development of bean by removing the home and object interfaces.

Elimination of Home and Component Interface 2-2

- Home interfaces in EJB 2.0 are replaced by Plain Old Java Interfaces and classes.
  - EJB 3.0 introduces a business interface which is designated by the bean developer as remote business interface or local business interface.
- In earlier version, the component interfaces were used as they provided a way for the container to notify the bean instance of the various lifecycle events affecting it.
  - **EJB 3.0 can receive notification in two ways:**
    - First, the developer can write a separate class containing the implementation of callback notification method. The container is then informed to treat the class as the bean's callback listener class.
    - Second, the developer can implement the callback notification methods within the bean class and designate each of these methods to handle appropriate events.

Interceptors

- Are the methods used to intercept business method calls or lifecycle callback method calls.
- Can be used by Stateless, Stateful, and Message-driven beans.
- Perform operations such as application auditing, logging, and so on.
- Are defined as methods in the bean class or as a separate interceptor class in the application.

Java Persistence API

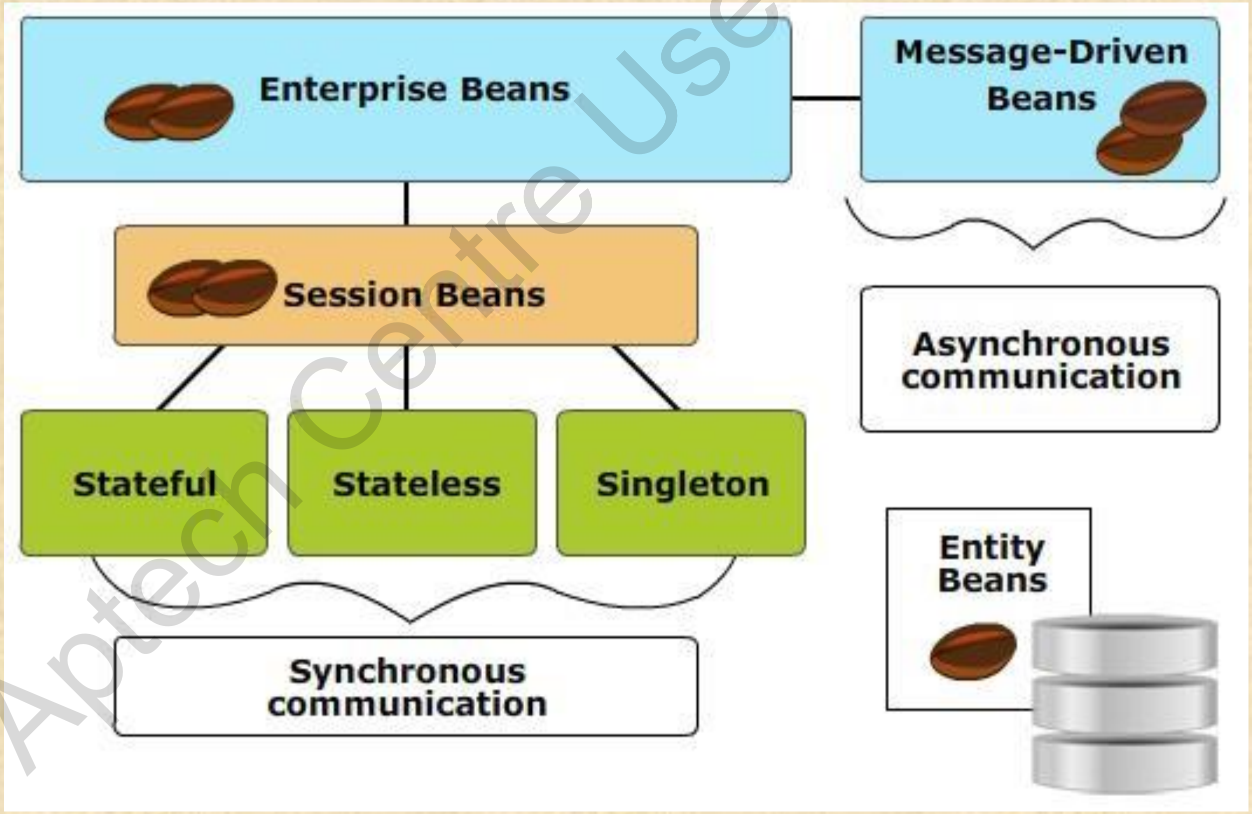
- Is the persistence technology used to persist enterprise beans in the databases.
- Bridges the gap between object-oriented interpretation in Java programs and relational database.
- Enables writing code independent of underlying database provider.
- Provides an enhanced EJB query language which supports execution of EJB queries.

Timer Service

- Timer service is used to schedule application at specific time intervals.
- Timer service enables the developer to schedule various events of the application without manual intervention.
- Developers can set up timers and define timer callback methods in the application which are invoked when the timer expires.

Types of Enterprise Java Beans

- Following figure shows categories of enterprise beans and their utility:



Session Beans

- Session beans:
  - Are used to implement business logic of the application.
  - Are deployed in the EJB container.
  - Can be invoked by the client through a local, remote, or Web service through a business interface.
- Following are different types of session beans:

Stateless	Stateful	Singleton

Message-Driven Beans

- Enterprise beans asynchronously invoked through Java Message Service(JMS) messages.
- Messages can be sent from the application client or another application component.
- Message-driven beans cannot be invoked through interfaces.

Entity Beans

- There data gets persisted in the database storage.
- They also implements business logic pertaining to the data in the application database.

Container Services for EJB 1-3

- According to the security model implemented by a Java EE container, the container authenticates and authorizes users wanting to access the application and system resources.

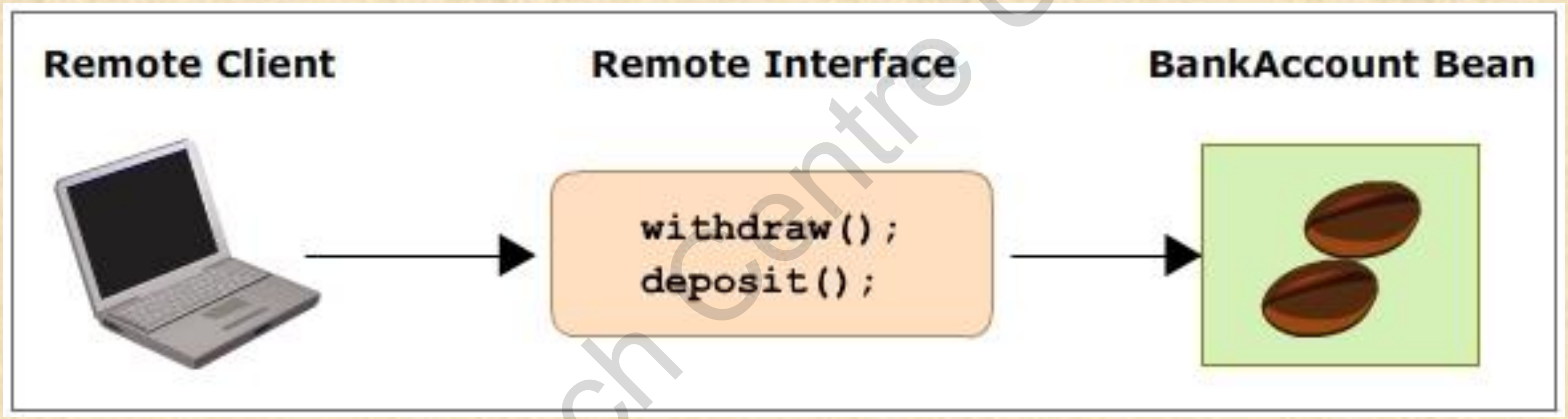
- The transaction model of the container ensures that the methods in a single transaction are appropriately executed without leading to an inconsistent application state.

Container Services for EJB 2-3

- The container provides naming services to access different components of the application without any conflict.
- The container also provides connectivity services to the application. When an application component deployed in the container attempts to connect to an external component, then the container manages all the lower-level communication tasks.

Container Services for EJB 3-3

- Following figure demonstrates accessing EJBs through remote interface:



Accessing Enterprise Java Beans 1-2

- Enterprise beans are accessed through the following interfaces:
  - Business interface refers to the set of methods provided by the bean class through which the enterprise bean can be invoked.
  - No-interface view refers to all the public methods of the bean class which can be used to access the bean.

Accessing Enterprise Java Beans 2-2

- Following code snippet demonstrates the business interface of `Calculator` bean:

```
package Test;
import javax.ejb.Remote;
@Remote
public interface Calculator {

  public String sayHello(String name);
  public int addition(int a,int b);
  int multiplication(int a,int b);
  int subtraction(int a,int b);
}
```

- The interface has four methods `sayHello()`, `addition()`, `multiplication()`, and `subtraction()`.

Local Clients

- A local client runs in the same application where enterprise bean is accessing.
- A local client can either be a bean component or a Web component.
- A local client can access the enterprise bean through:
  - No-interface view
  - Business interface annotated with `@Local`

Remote Clients

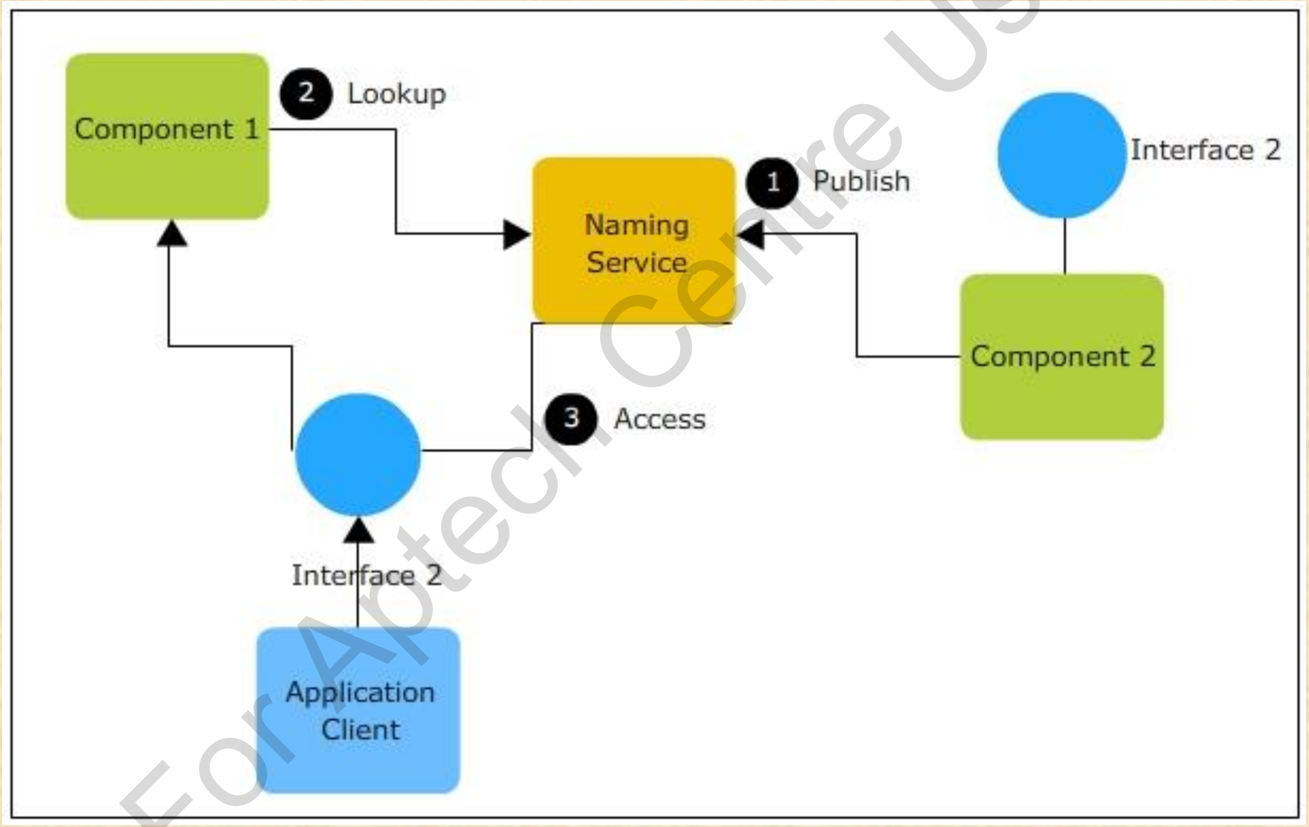
- Remote client can run on a different machine, different JVM or application.
- Remote client can be a bean component, Web component, or application client.
- Remote clients can access the enterprise bean through a remote interface.
- Remote interface should be annotated with `@Remote`.
- Remote clients cannot access enterprise bean through no-interface view.

Java Naming and Directory Interface 1-9

- A large scale enterprise domain has large number of objects interacting with each other.
- These applications require a well-defined naming infrastructure to access objects as per the requirement.
- JNDI is an API which provides these naming services.
- JNDI binds a name with an object in the application.

Java Naming and Directory Interface 2-9

- Following figure demonstrates how clients can access objects through a naming service:

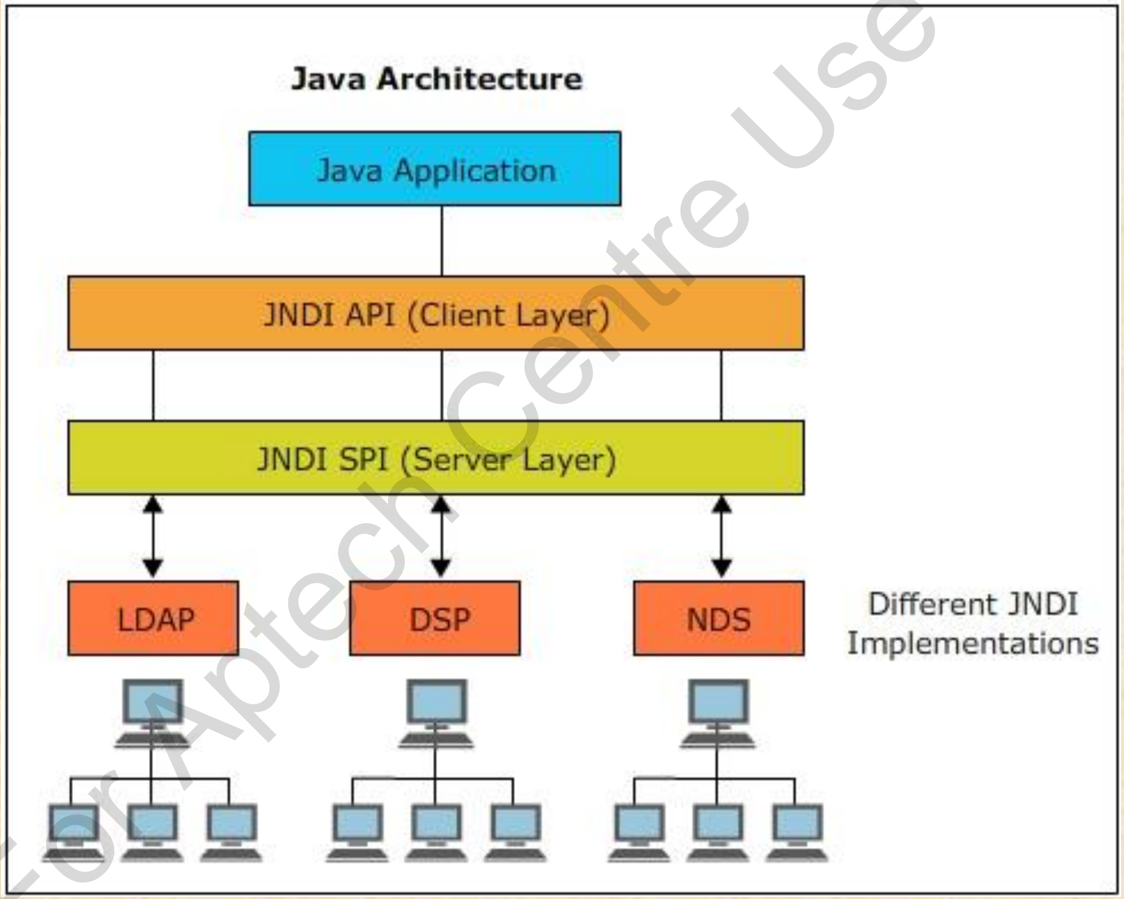


Java Naming and Directory Interface 3-9

- All application components publish their identity to the naming service.
- When an application component has to access a bean it performs a JNDI lookup.
- The naming service then provides reference to the application component.
- All remote clients of the application access the object through the JNDI name.
- JNDI organizes the objects in the namespace in a hierarchy.

Java Naming and Directory Interface 4-9

- Following figure shows the JNDI architecture:



Java Naming and Directory Interface 5-9

- The JNDI architecture contains two parts:
  - **JNDI API** – Are used by the remote applications to access the mapped components or objects from JNDI registry.
  - **JNDI Service Provider Interface (SPI)** – Is a mechanism to plugin naming and directory services of different vendors on the Java EE platform.

Java Naming and Directory Interface 6-9

- `java.naming` is the main package of JNDI API.
- `java.naming` has the following important classes:
  - `InitialContext` – used to establish connection and retrieve `Context` object.
  - `Context` – provides methods for binding and unbinding JNDI names with objects.
- Following code snippet shows how to retrieve the Context object:

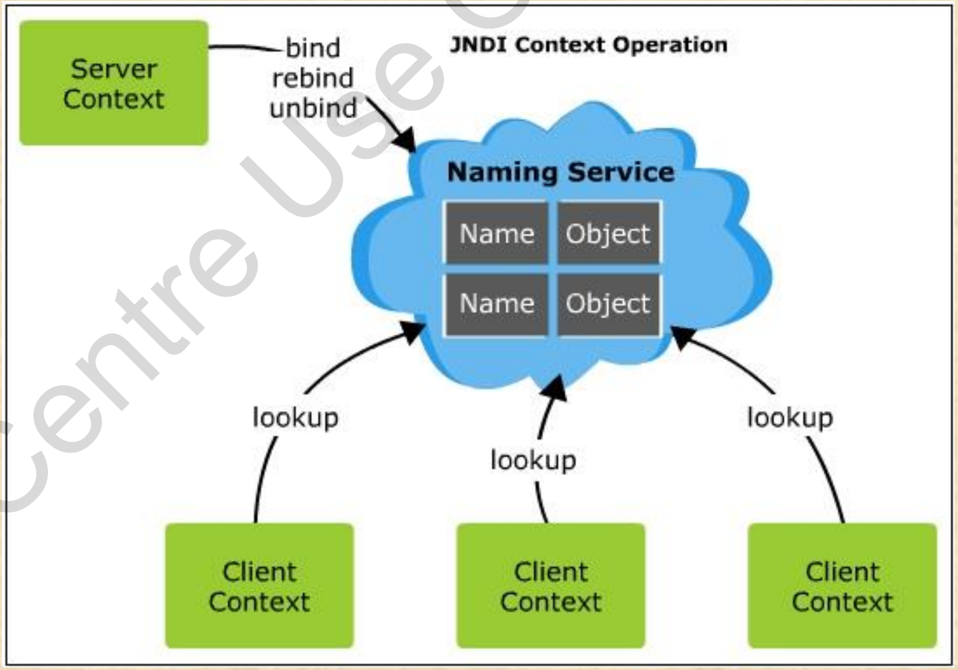
```
Context ctx = new InitialContext();
```

Java Naming and Directory Interface 7-9

- Following are the methods which are part of `Context` class are:
  - `bind()`
  - `unbind()`
  - `lookup()`
  - `rebind()`

Java Naming and Directory Interface 8-9

- The given figure demonstrates how to obtain a JNDI context using `InitialContext()`:
  - The client performs the lookup of the objects registered in the JNDI registry.
  - The application server provides the services for binding and unbinding the objects to the JNDI registry.



Java Naming and Directory Interface 9-9

❑ JNDI provides three namespaces:

java:global identifies an object through a hierarchy of application name

- Format: `java:global[/`*`application name`*`]/`*`module name`*`/`*`enterprise bean name`*`[/`*`interface name`*` ]`

java:module identifies the component from module level of naming hierarchy

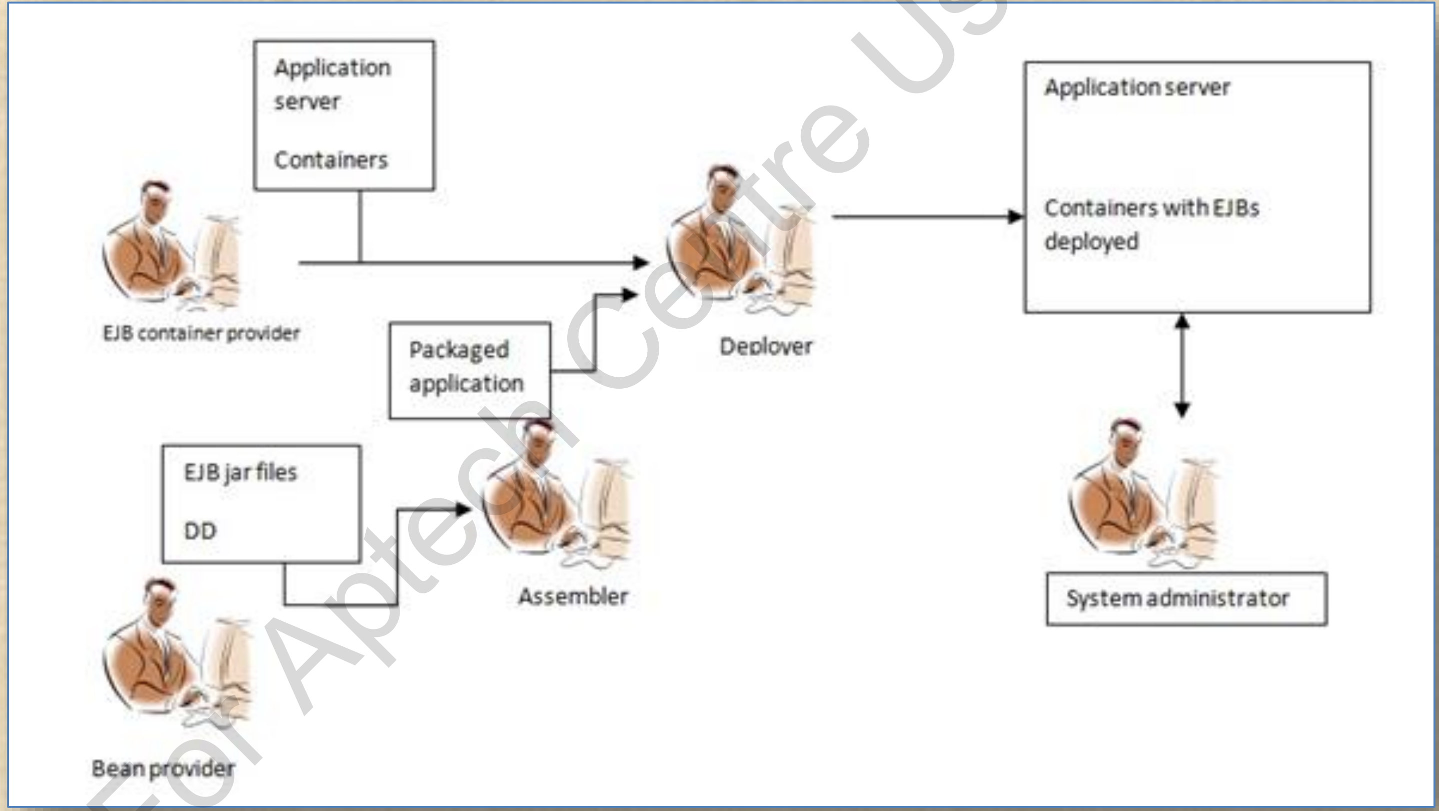
- Format: `java:module/`*`enterprise bean name`*`/[`*`interface name`*`]`

java:app identifies the enterprise beans packaged within the application

- Format: `java:app[/`*`module name`*`]/`*`enterprise bean name`*` [/`*`interface name`*`]`

EJB Roles 1-2

- Following figure demonstrates various roles in an enterprise application development and deployment:



EJB Roles 2-2

EJB container provider

- Is responsible for implementing all the container services upon which the application can be deployed.

Bean provider

- Implements business logic through bean classes.

Application assembler

- Assembles all the application components.

EJB deployer

- Puts the application onto the production environment.

System administrator

- Provides access rights to the application end users and monitoring the application performance.

Developing a Java EE Application

❑ Following are different steps in enterprise application development:

Designing the application

↓

Developing the components of the application

↓

Assembling and packaging of the components as a single unit

↓

Deploying the file on the Java-enabled application server

Application Design

❑ Following are the mechanisms for enterprise application design:

Design patterns

- Reusable solutions for common design problems
- Solution is described based on a consistent format

UML diagrams

- Object diagrams which can be translated into classes and objects

Coding

- ❑ Involves writing code for various application components.
- ❑ Application components include user interface, database, enterprise beans, and so on.
- ❑ Application is developed using NetBeans IDE.
- ❑ Developers develop application components, debug, and perform unit testing during this phase of application development.

Deployment Descriptor 1-2

- Describes interaction between various components of the application in declarative fashion.
- Is an XML file used by Java-enabled application server to obtain the references of the components.
- Includes the following information:
  - Transaction management policy of the application.
  - Security specifications and authorization constraints.
  - Configuration variables used during application development.
  - Resource access dependencies of the application.

Deployment Descriptor 2-2

- Following code snippet shows the code of the deployment descriptor `glassfish-ejb-jar.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-ejb-jar PUBLIC . . .>
<glassfish-ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CalculatorImplBean</ejb-name>
      <jndi-name>caljndi</jndi-name>
      <business-local>CalculatorLocal</business-local>
      <business-remote>CalculatorRemote</business-
                                              remote>
      <ejb-class>beans.CalculatorImpl</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</glassfish-ejb-jar>
```

Packaging 1-2

- The steps for packaging and deploying a Java EE application are as follows:
  - The Application Component Providers create Java EE modules, such as EJB, Web, Resource Adapter, and Application clients.
  - These modules can be deployed independently without being packaged into a Java EE enterprise application.
  - The Application Assembler packages these modules to create a complete Java EE enterprise application.
  - Deployer deploys the deployable unit.

Packaging 2-2

- The Java EE specification provides different types of archive files to package the modules. These are as follows:

EJB modules	• Comprise all class files • Class files are packaged as .jar files
Web modules	• Comprise all HTML files, JSPs, JSFs, and Servlets • These files are packaged as .war files
Resource adapter modules	• Components for resource adaptation • Packages a .rar files
Application client modules	• These are class files packaged as .jar files

Assembling and Deployment 1-4

- Independent components of the application are to be assembled together.
- Assembled application has to be deployed on the application server.
- Application is deployed either manually or through a tool.
- Deployment process also involves vendor specific configuration with respect to load balancing and performance tuning of application components.

Assembling and Deployment 2-4

- The Application Assembler packages the components into an Enterprise Archive (EAR) file.
- The assembler should ensure the root contains a folder called `META-INF`.
  - This folder contains a deployment descriptor called application.xml, one or more container-specific deployment descriptors, and `manifest.mf` file.
  - The `application.xml` file contains the names of all the Java EE archives that are packaged in the EAR file.
  - The `manifest.mf` file contains additional meta-information about the EAR file.

Assembling and Deployment 3-4

- Besides the META-INF folder, the root of the file structure contains the archives of the Java EE modules that constitute an EAR file.

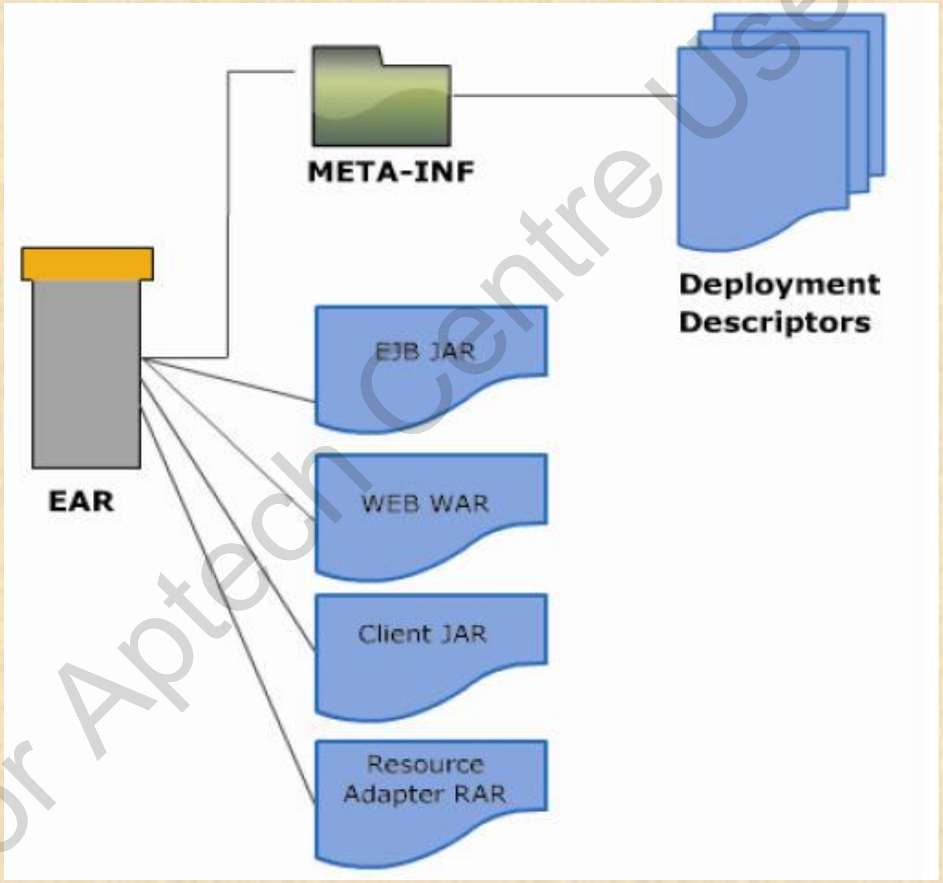
- These archives can be:
  - .jar files for EJB module
  - .war files for Web modules
  - .jar files for Application Clients
  - .rar files for Resource Adapters

- These archives can be either placed in the root or in one or more sub-folders.

- To manually package the components in an EAR file, the command can be run as:
  - `jar cvf <name>.ear *`

Assembling and Deployment 4-4

- Following figure shows the file structure of EAR file:

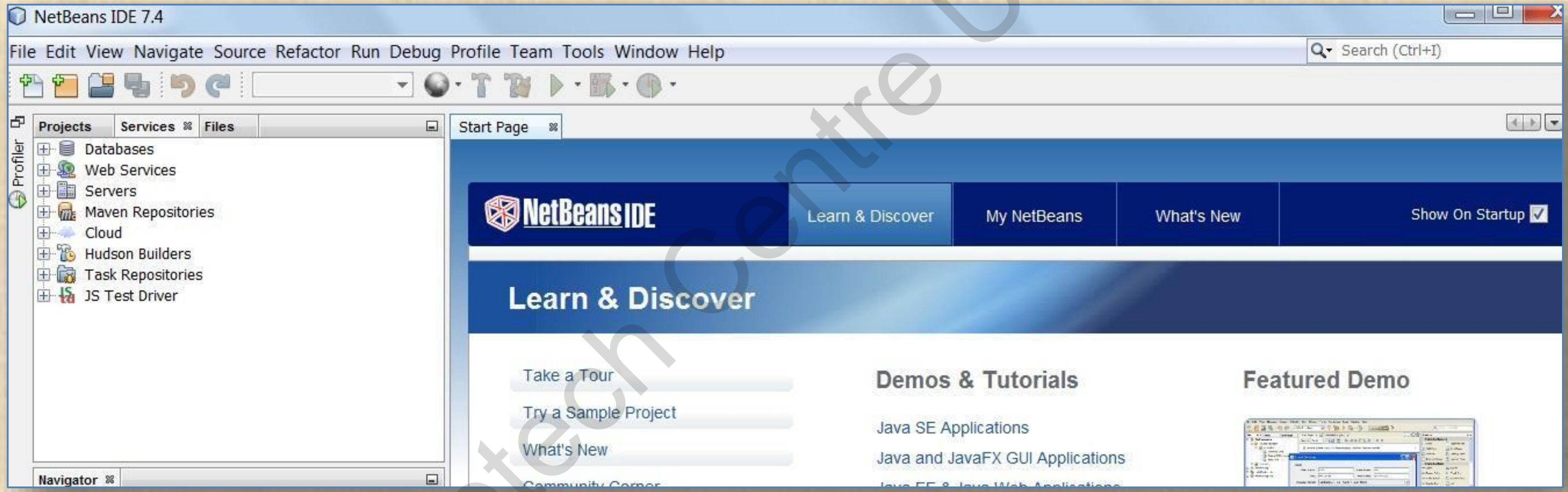


EJB Development Tools 1-2

- The EJB development environment is provided by Integrated Development Environments (IDEs) such as NetBeans, Eclipse, and so on.
- Following are different categories of tools provided by the development environment:
  - J2EE Perspective editor
  - Tools for creating and accessing enterprise beans
  - Tools for accessing the database
  - Tools for generating deployment code from the annotations
  - Tools for creating applications based on design patterns

EJB Development Tools 2-2

- Following figure shows the NetBeans IDE used for enterprise application development:



Summary

- Enterprise JavaBeans are application components which implement business logic of the application.
- There are three types of enterprise beans – Session beans, Entity Beans, and Message-driven beans.
- Session beans can be stateless, stateful, and singleton, they are synchronously invoked by the application client.
- Message-driven beans are enterprise beans which are asynchronously invoked through JMS messages.
- Entity beans are used to implement persistence in enterprise applications.
- Entities are enterprise beans which implement database operations, they are deprecated and entity persistence to database is implemented by Java Persistence API in EJB 3.0.
- Enterprise beans are deployed in a container which provides services such as transaction management, component security, naming services, and so on.
- Enterprise beans can be accessed through business interface view or no interface view of the application.
- Annotations are metadata used in the application for deployment and dependency injection.
- Enterprise beans are packaged into EAR file.