# Web Applications Developments

# ITWT413

LECTURE 3: PLATFORM BASICS

DR. HALA SHAARI

## Course Structure

- Lecture ( Sunday& Tuesday)
- Time: 12:30-14:00
- Microsoft Teams Code (goi2d67)

(Lectures, labs, announcements, References):

- Grading :
  - 25%  Midterm exam.
  - 25% Assignments
  - 25% Group Project (Groups of two).
  - 25% Final Exam.

## Course References

Our Course Main References:

- Jakarta EE Tutorial
- Java EE to Jakarta EE 10 Recipes: A Problem-Solution Approach for Enterprise Java (2022)

## key course objectives

1) Master Core Jakarta EE Concepts: Understand the fundamental architecture and components of Jakarta EE, including Servlets, JSP, JSF, and Enterprise JavaBeans (EJB).
2) Develop RESTful Web Services: Build and deploy scalable RESTful APIs using Jakarta RESTful Web Services (JAX-RS), with advanced features like exception handling, filters, and security.
3) Implement Dependency Injection and Persistence: Leverage Contexts and Dependency Injection (CDI) and Jakarta Persistence (JPA) to manage beans and database interactions in enterprise applications.
4) Ensure Application Security and Transaction Management: Apply Jakarta EE’s security framework for authentication, authorization, and manage transactions using declarative and programmatic approaches.
5) Deploy Cloud-Native Applications: Utilize Jakarta EE to build, containerize, and deploy microservice-based applications in cloud environments, incorporating tools like Docker and Kubernetes.

## Lecture Agenda

- Required Software
- Platform Basics
  - Resource Creation
  - Injection
  - Packaging

## Required Software

The following software is required to run the examples

- Java Platform, Standard Edition
- Eclipse Glassfish Server
- Jakarta EE Tutorial Examples
- Apache NetBeans IDE
- Apache Maven
- Instructions from page 11 to page17 from Jakarta EE Tutorial tells you everything you need to know to install, build, and run the tutorial examples

## Understanding Jakarta EE Services

- Jakarta EE provides a wide range of services that can be combined to meet the specific needs of different applications.
- These services are grouped into three categories: Core, Web, and Platform.
- The Core profile contains the foundational services for enterprise application development, such as dependency injection, RESTful web services, and JSON processing.
- The Web profile extends the Core profile with services for building web applications, such as servlets and Jakarta Faces.
- The Platform profile provides the most comprehensive set of services, including the Core and Web profiles, as well as additional services for mail, batch processing, and messaging.

## Default Paths and File Names

- Jakarta EE applications use a set of default paths and file names that are commonly used in various environments.
- Understanding these default locations is crucial for configuring and deploying Jakarta EE applications.
- The as-install placeholder represents the base installation directory for GlassFish Server, which is typically located in the user's home directory.
- The as-install-parent placeholder represents the parent of the as-install directory.
- The jakartaee-examples placeholder represents the base installation directory for the Jakarta EE Tutorial examples.
- The domain-dir placeholder represents the directory where a domain's configuration is stored.

## Resource Creation

- Jakarta EE applications use program objects that provide connections to external systems, such as databases and messaging systems.
- These program objects are called resources.
- Resources are typically managed by a JNDI naming service.
- Jakarta EE provides mechanisms to create and access these resources programmatically using annotations or administratively using a deployment descriptor or tools.
- Commonly used resources include data sources, mail sessions, and Jakarta Messaging objects.

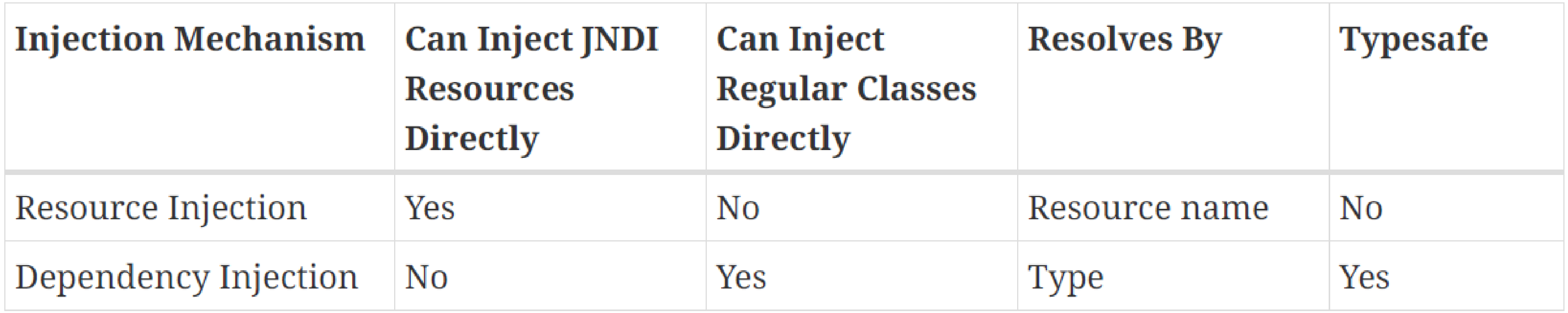
## Resource Injection

- Resource injection is a mechanism that enables components to obtain references to resources without explicitly creating them.
- You can inject resources into any Jakarta EE component using the @Resource annotation.
- The @Resource annotation can be used to inject resources for a specific name or for a default name.
- Resource injection simplifies application development by reducing the need to create and access resources manually.
- You can also use deployment descriptors to override the resource mapping specified by annotations.

# Jakarta Contexts and Dependency Injection (CDI)

- Jakarta CDI is a powerful framework that allows you to manage and inject components into Jakarta EE applications in a typesafe and loosely coupled way.
- It provides services for dependency injection, context management, interceptors, and decorators.
- You use @Inject to inject managed beans, which are classes annotated with a scope type and the @Alternative annotation.
- CDI simplifies application development by reducing the need to explicitly manage the lifecycle of components and by reducing the complexity of dependency management.
- You can use @Produces to create producer methods or producer fields, which generate beans and allow you to control the bean implementation at runtime.
- You can use @Disposes to define disposer methods, which release resources when they are no longer needed.

## Resource Injection Vs Dependency Injection



| Injection Mechanism | Can Inject JNDI Resources Directly | Can Inject Regular Classes Directly | Resolves By | Typesafe |
|---|---|---|---|---|
| Resource Injection | Yes | No | Resource name | No |
| Dependency Injection | No | Yes | Type | Yes |

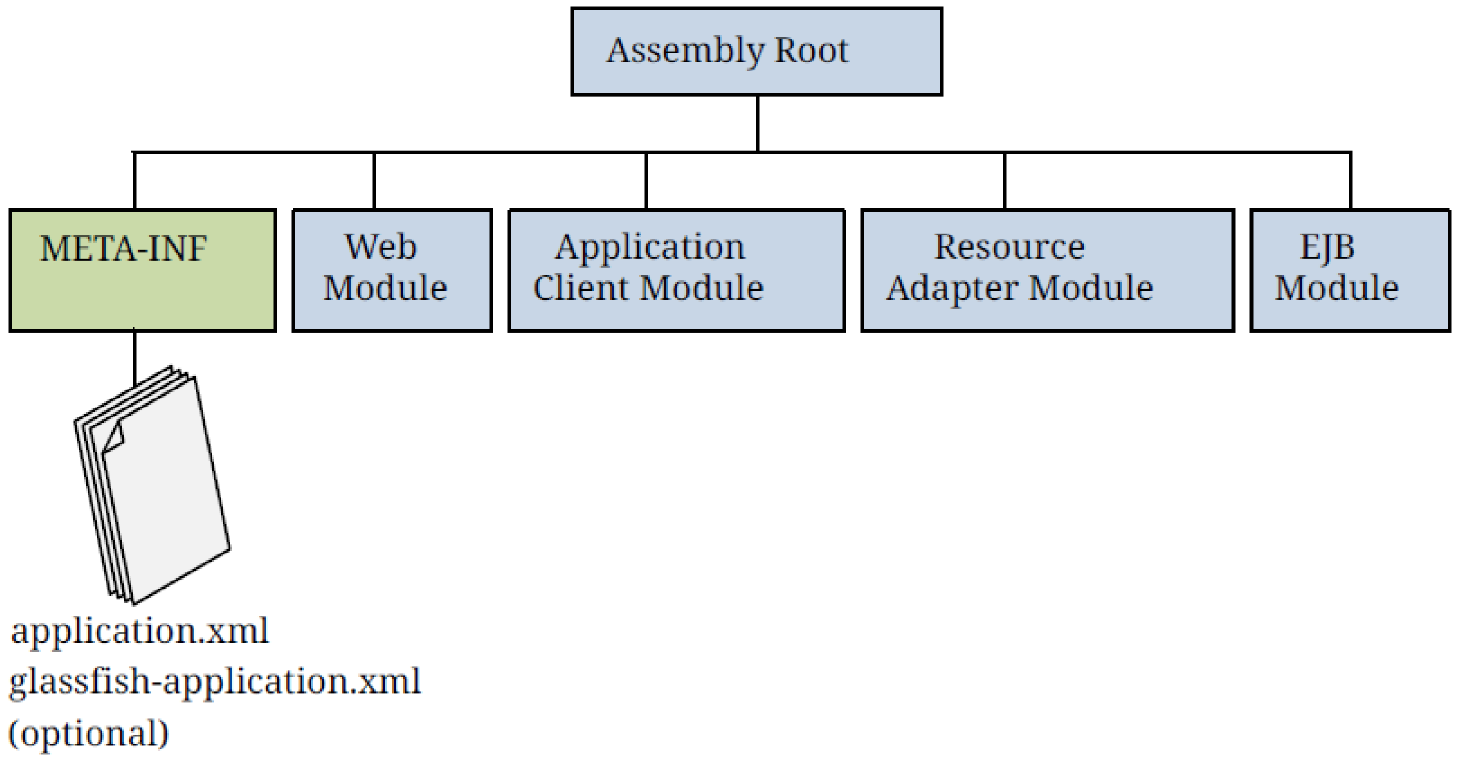
## Packaging Applications

- **Key Types**: Jakarta EE applications are built using components, which can be classified as `Web Components`, `Business Components`, or `Application Clients`.
- **Web Components**: Web components, such as servlets and Jakarta Faces components, handle HTTP requests and provide web content to users.
- **Business Components**: Business components, such as enterprise beans, encapsulate business logic for the application.
- **Application Clients**: Application clients are Java programs that consume the services provided by Jakarta EE applications.
- **Component Life Cycle**: Jakarta EE components have lifecycle methods, such as `init`, `service`, and `destroy` for servlets, and `PostConstruct`, `PreDestroy`, `PostActivate`, and `PrePassivate` for enterprise beans.

## Component Packaging

- **Component Packaging**: Components are packaged into standard units:
  - **JAR (Java Archive)** files for enterprise beans
  - **WAR (Web Archive)** files for web applications
  - **EAR (Enterprise Archive)** files for a collection of modules.

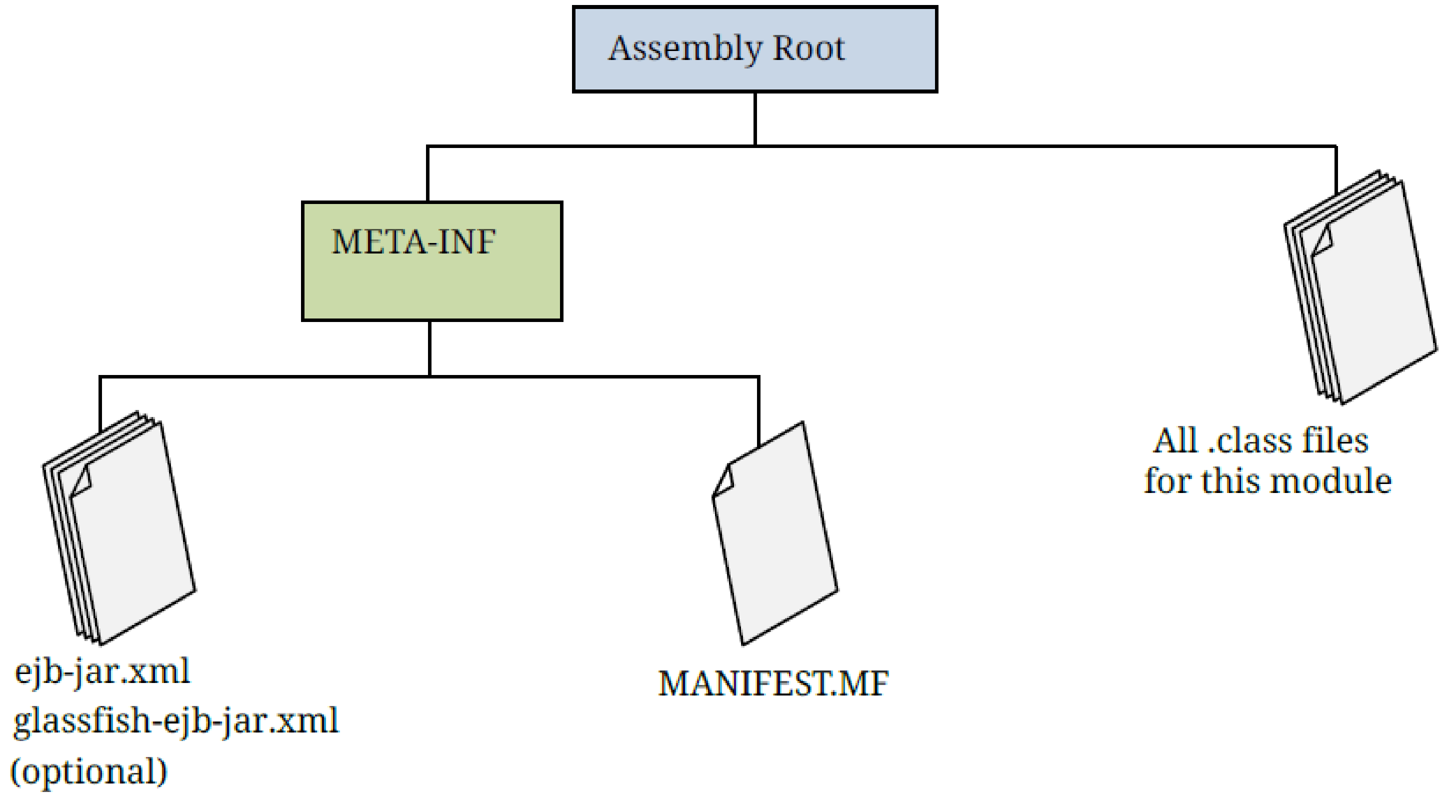
## EAR File Structure



## Packaging Enterprise Beans

- **Enterprise Bean JAR Modules**: Enterprise beans are typically packaged into JAR files, with a `beans.xml` deployment descriptor.
- **The `beans.xml` Deployment Descriptor**: This file defines the configuration for the CDI container within the JAR file.
- **EJB JAR Structure**: The structure of an enterprise bean JAR file consists of the META-INF directory,
  `ejb-jar.xml` deployment descriptor, `glassfish-ejb-jar.xml` optional deployment descriptor,
  `MANIFEST.MF` file, and the compiled class files.
- **Packaging Enterprise Beans in WAR Files**: Enterprise beans can be packaged within a WAR file for web applications.
- **Packaging Enterprise Beans in EAR Files**: Enterprise beans can be packaged within an EAR file along with other modules to create a composite application.

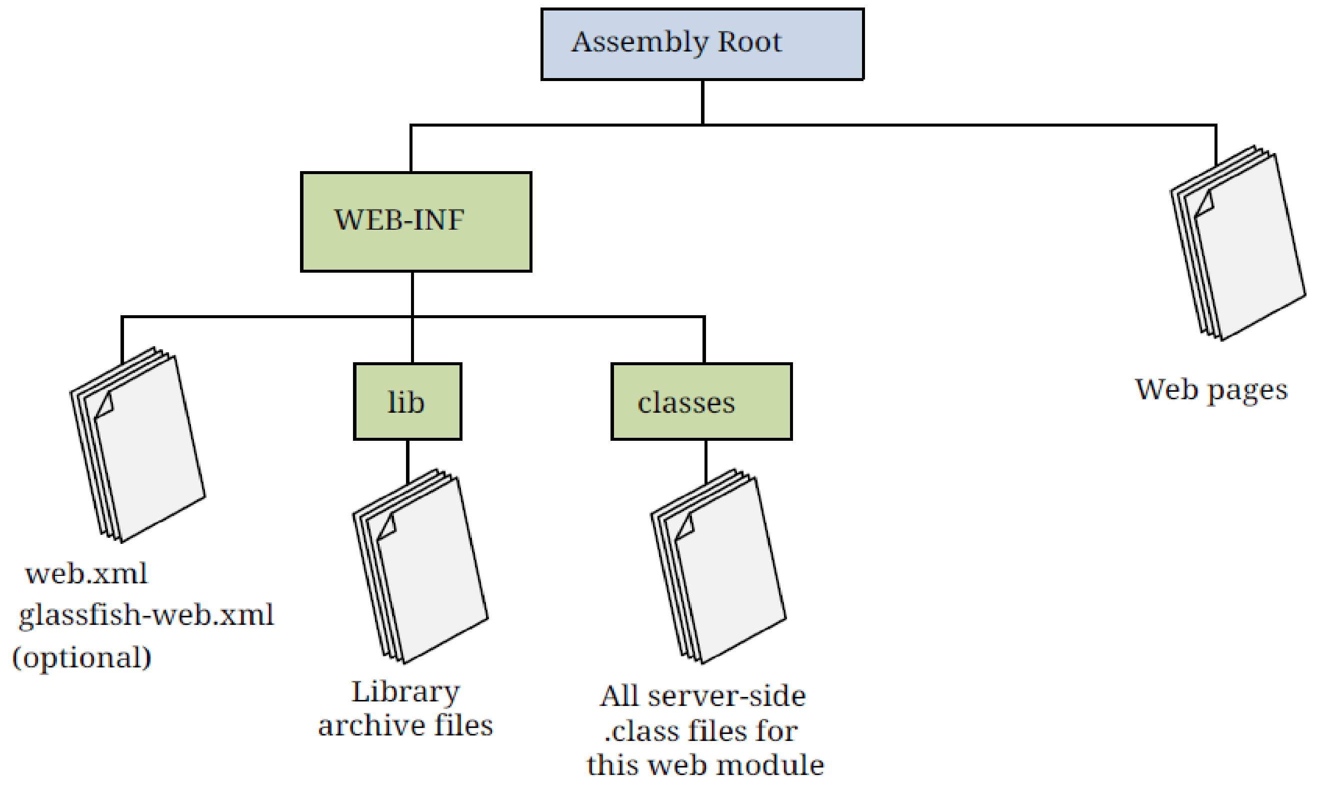
## Structure of an Enterprise Bean JAR



## Packaging Web Archives (WAR Files)

- **Web Module Definition**: A web module represents a single web application, typically using servlets and Jakarta Faces components.
- **WAR File Structure**: WAR files include the following:
  - **WEB-INF directory**: Contains the `web.xml` deployment descriptor, `beans.xml` deployment descriptor, `faces-config.xml` optional deployment descriptor, compiled class files, and JAR files.
  - **Content**: The root directory contains the web pages (HTML or XHTML) and any other static resources.
- **WAR Deployment**: WAR files are deployed to a servlet container, which provides the runtime environment for web applications.
- **WAR Deployment Example**: Consider deploying a WAR file named `myapp.war` to a GlassFish Server. The server will typically deploy the application to the `glassfish/domains/domain1/autodeploy` folder.
- **Jakarta Faces and WAR Files**: Jakarta Faces applications are commonly packaged as WAR files, including the necessary Facelets template and XHTML pages.

## Web Module Structure



Questions?