Profesjonalne aplikacje dla biznesu klasy Enterprise

Wprowadzenie do Jakarta EE (Java EE 8)

Piotr Apollo
Krzysztof Grel

FIS-SST

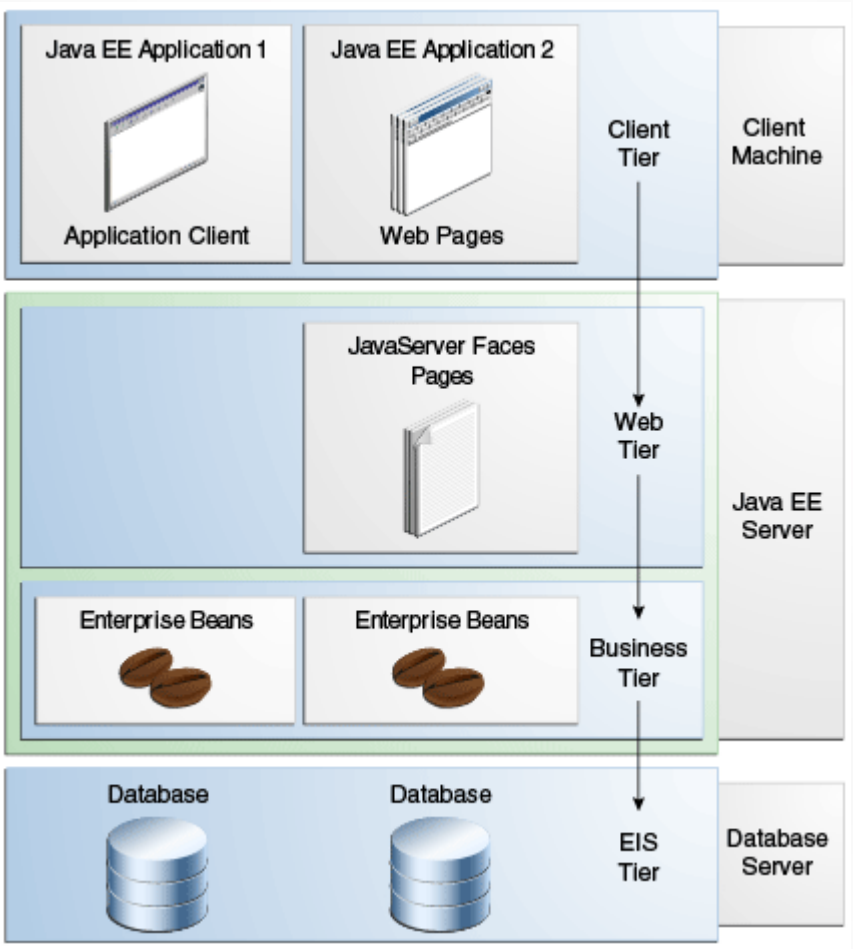
What is the Java EE 8 platform?

The Java EE platform provides an **API** and **runtime** environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network (enterprise) applications.

Java EE 8 platform

- Specification – defines a set of APIs and their interaction
- Runtime
  - Application server
    - Full Profile
    - Web Profile
  - GlassFish 5 – a reference implementation
- Extends Java SE APIs

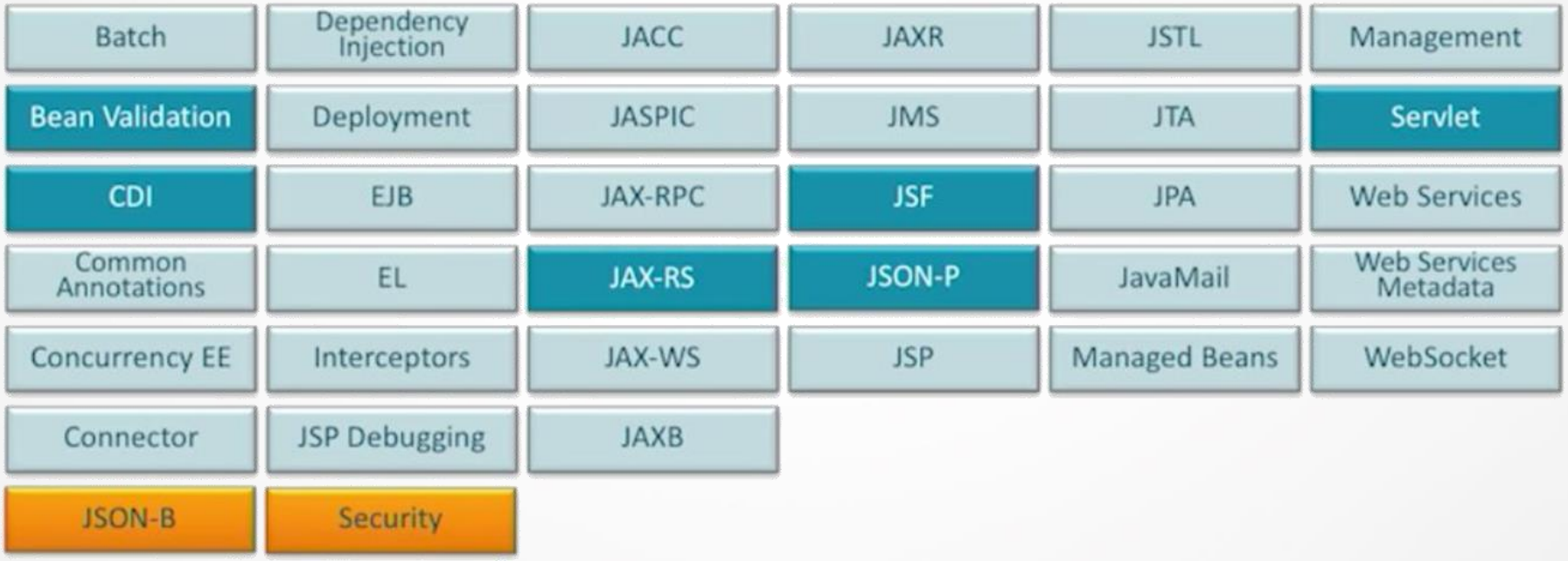
Distributed Multi-tiered Applications



The key goals of the Java EE 8 platform

- To modernize the infrastructure for enterprise Java for the cloud and microservices environments,
- emphasize HTML5 and HTTP/2 support,
- enhance ease of development through new Contexts and Dependency Injection features,
- and further enhance security and reliability.

Java EE 8 – APIs Overview



Java EE 8

- New
  - Java API for JSON Binding (JSON-B)
  - Java EE Security API
- Updated
  - Java Servlet (3.1 -> 4.0)
  - Contexts and Dependency Injection (CDI, 1.1. -> 2.0)
  - JavaBean Validation (1.1 -> 2.0)
  - JSON Processing (JSON-P, 1.0 -> 1.1)
  - RESTful web services (JAX-RS, 2.0 -> 2.1)

Java Servlet 4.0 (Servlet)

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a **request-response** programming model.

Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

Servlet - Lifecycle

The lifecycle of a servlet is controlled by the container in which the servlet has been deployed.

When a request is mapped to a servlet, the container performs the following steps.

1. If an instance of the servlet ***does not exist***, the web container:
2. Loads the servlet class
3. Creates an instance of the servlet class
4. Initializes the servlet instance by calling the ***init*** method
5. The container invokes the ***service*** method, passing request and response objects.

If it needs to remove the servlet, the container finalizes the servlet by calling the servlet's ***destroy*** method.

Servlet - Lifecycle

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@WebServlet(„first-servlet")
public class FirstServlet extends HttpServlet
{
  @Override
  protected void doGet(
    HttpServletRequest req, HttpServletResponse resp)
  throws ServletException, IOException
  {
    // implement logic
  }
}
```

Servlet – Lifecycle Events

Monitoring and reacting to events in a servlet's lifecycle by defining listener objects whose methods get invoked when lifecycle events occur. Use the ***@WebListener*** annotation to define a listener to get events for various operations on the particular web application context.

Classes annotated with ***@WebListener*** must implement one of the following interfaces:

- `javax.servlet.Servlet`**`Context`**`tListener`
- `javax.servlet.Servlet`**`Context`**`AttributeListener`
- `javax.servlet.http.Http`**`Session`**`Listener`
- `javax.servlet.http.Http`**`Session`**`ActivationListener`
- `javax.servlet.http.Http`**`Session`**`AttributeListener`
- `javax.servlet.Servlet`**`Request`**`Listener`
- `javax.servlet.Servlet`**`Request`**`AttributeListener`

Servlet – Lifecycle Events

```
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

@WebListener
public class SessionListener implements HttpSessionListener
{
  public void sessionCreated(HttpSessionEvent se)
  {
    // implement logic
  }

  public void sessionDestroyed(HttpSessionEvent se)
  {
    // implement logic
  }

}
```

Servlet – Scope Objects

- Collaborating web components share information by means of objects that are maintained as attributes of four scope objects:
  - **Web context** (`javax.servlet.ServletContext`)
  - **Session** (`javax.servlet.http.HttpSession`)
  - **Request** (Subtype of `javax.servlet.ServletRequest`)
  - Page (JSP) (`javax.servlet.jsp.JspContext`)
- In a multithreaded server, shared resources can be accessed concurrently.

Servlet - Filtering Requests and Responses

- Can transform the header and content (or both) of a request or response
- Provides functionality that can be "attached" to any kind of web resource
- The order of the filters in the chain is the same as the order in which filter mappings appear in the web application deployment descriptor (web.xml).
- *doFilter()* do the job

Servlet - Filtering Requests and Responses

```
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;

@WebFilter(filterName = "FirstFilterExample",
urlPatterns = {"first-servlet"},
initParams = { @WebInitParam(name = "mode", value = "test")})
public class FirstFilter implements Filter
{
  public void doFilter(ServletRequest request, ServletResponse
response,      FilterChain chain)
  throws IOException, ServletException
  {
    // implement you logic
    chain.doFilter(request, response);
  }
}
```

Context and Dependency Injection (CDI)

A major theme of CDI is **loose coupling**. CDI does the following:

- Decouples the server and the client by means of well-defined **types** and **qualifiers**, so that the server implementation may vary
- Decouples the **lifecycles** of collaborating components by
  - Making components contextual, with automatic lifecycle management
  - Allowing stateful components to interact like services, purely by message passing
- Completely decouples message producers from consumers, by means of events
- Decouples orthogonal concerns by means of Java EE **interceptors**

CDI – Provided Services

- Contexts
- Dependency injection
- The ability to decorate injected components
- Typesafe interceptor bindings
- An event-notification model
- Additional web conversation scope
- Integration with the Expression Language (EL)
- A complete Service Provider Interface (SPI) that allows third-party frameworks to integrate cleanly in the Java EE environment

CDI – Managed Bean

- A (nonempty) set of bean types
- A (nonempty) set of qualifiers
- A scope
- Optionally, a bean EL name
- A set of interceptor bindings
- A bean implementation

CDI - Injection

The following kinds of objects can be injected:

- Almost any Java class
- Session beans
- Java EE resources
  - data sources, Java Message Service topics, queues, connection factories, and the like
- Persistence contexts (Java Persistence API EntityManager objects)
- Producer fields
- Objects returned by producer methods
- Web service references
- Remote enterprise bean references

CDI – Predefined Beans

Interface	Example
javax.transaction.UserTransaction	**@Resource** UserTransaction transaction;
java.security.Principal	**@Resource** Principal principal;
javax.validation.Validator	**@Resource** Validator validator;
javax.validation.ValidatorFactory	**@Resource** ValidatorFactory factory;
javax.servlet.http.HttpServletRequest	**@Inject** HttpServletRequest req;
javax.servlet.http.HttpSession	**@Inject** HttpSession session;
javax.servlet.ServletContext	**@Inject** ServletContext context;

CDI - Qualifiers

- Allows providing various implementation of bean type
- Java annotation

```
@Qualifier
@Retention(RUNTIME)
@Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Electric {}
```

```
public class DieselCar extends Car {
public int emmission()
{ return 98; }
}
```

```
@Electric
public class ElectricCar extends Car {
public int emmission()
{ return 0; }
}
```

```
public class ElectricCarFootprint
extends CartFootprint {

@Electric @Inject private Car car;

// some business logic
}
```

CDI – Producer Method

Producer methods provide a way to inject objects that are not beans, objects whose values may vary at runtime, and objects that require custom initialization.

Qualifier

```
@MaxSpeed
```

Injection Point

```
@Inject @MaxSpeed
private int maxSpeed;
```

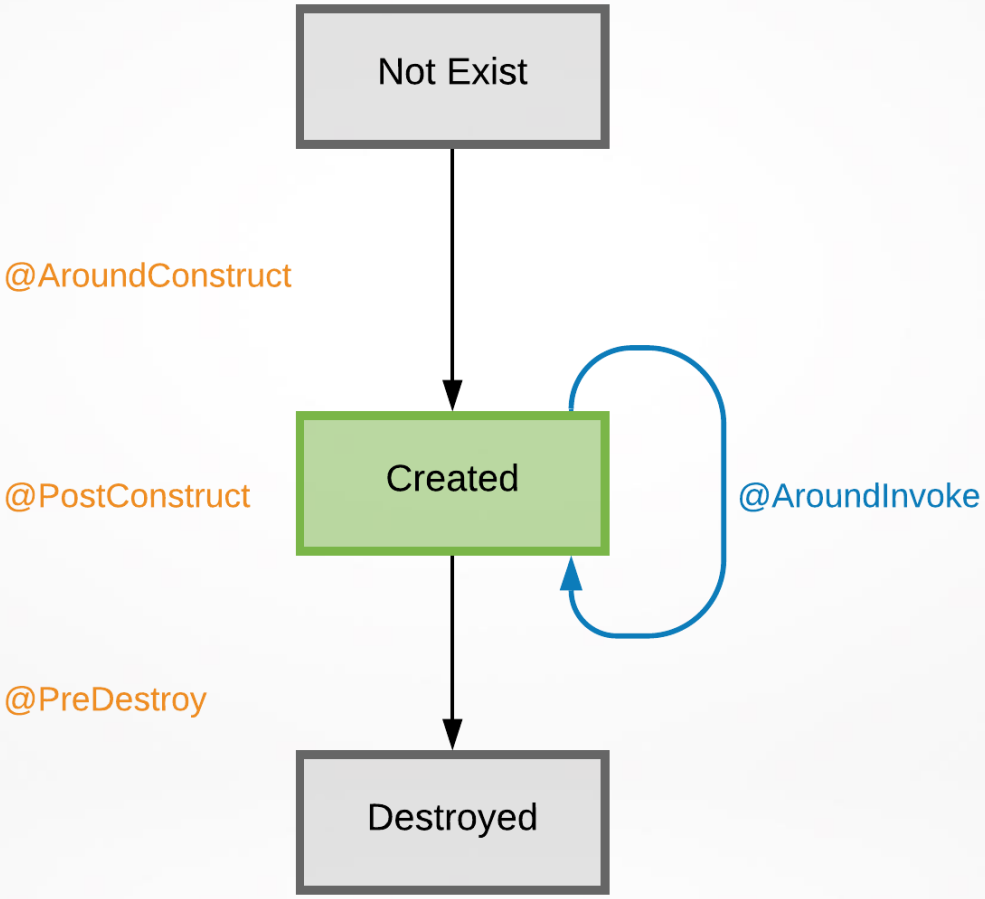
Producer Method

```
private int maxSpeed = 250;
...
@Produces @MaxSpeed
int getMaxSpeed() {
    return maxSpeed;
}
```

CDI - Interceptor

An interceptor is a class used to interpose in method invocations or lifecycle events - **cross-cutting** tasks - that occur in an associated target class.

CDI – Interceptor lifecycle callback methods



CDI - Scope

`@RequestScoped`

A user's interaction with a web application in a single HTTP request.

`@SessionScoped`

A user's interaction with a web application across multiple HTTP requests.

`@ApplicationScoped`

Shared state across all users' interactions with a web application.

`@Dependent`

The default scope if none is specified; it means that an object exists to serve exactly one client (bean) and has the same lifecycle as that client (bean).

`@ConversationScoped`

A user's interaction with a servlet, including JavaServer Faces applications. The conversation scope exists within developer-controlled boundaries that extend it across multiple requests for long-running conversations. All long-running conversations are scoped to a particular HTTP servlet session and may not cross session boundaries.

Enterprise Bean 3.2 (EJB)

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application.

The EJB **container** provides system-level services to enterprise beans.

EJB

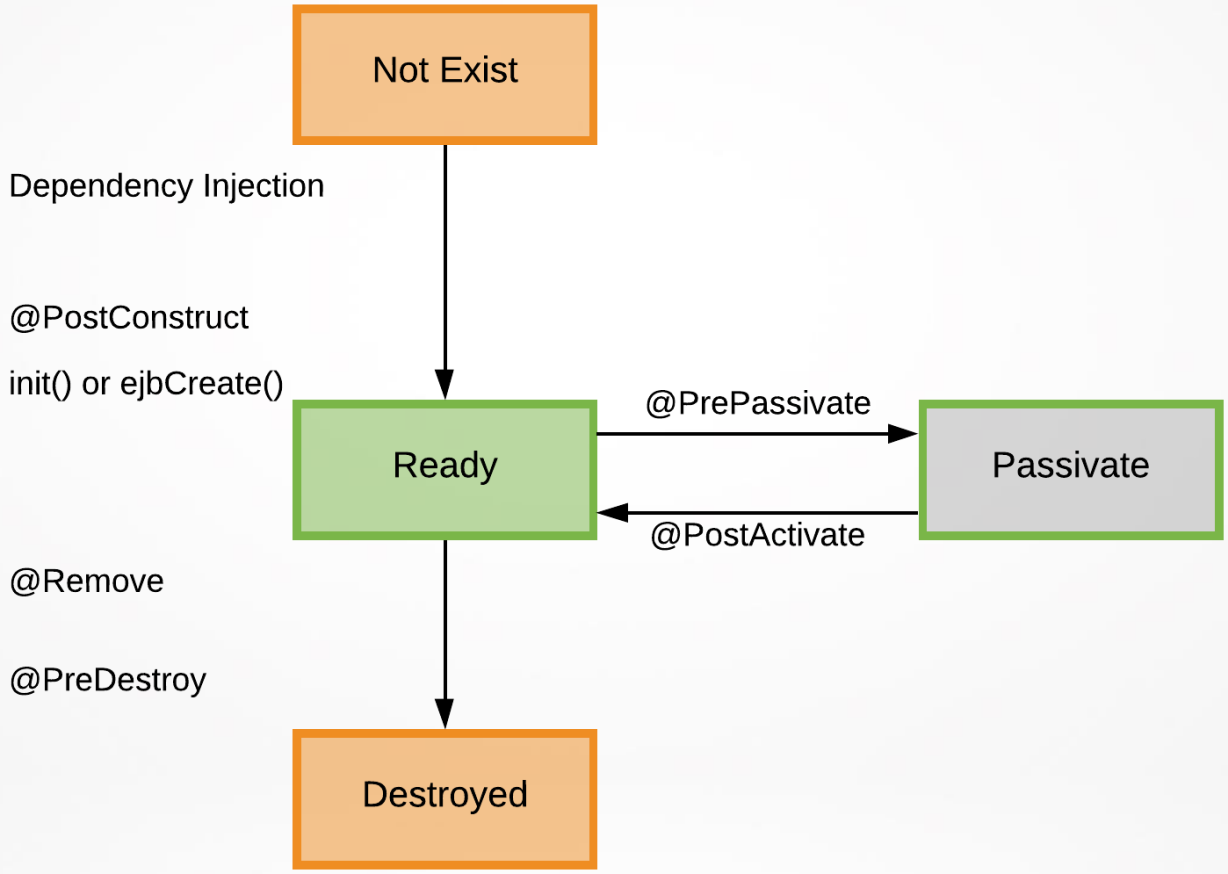
- Run in the EJB **container**
  - provides system-level services, such as transaction management and security authorization.
- Supports Client-Server architecture
  - Separates business logic from presentation logic
- Component scaling
  - Transparent component distribution

EJB - Types

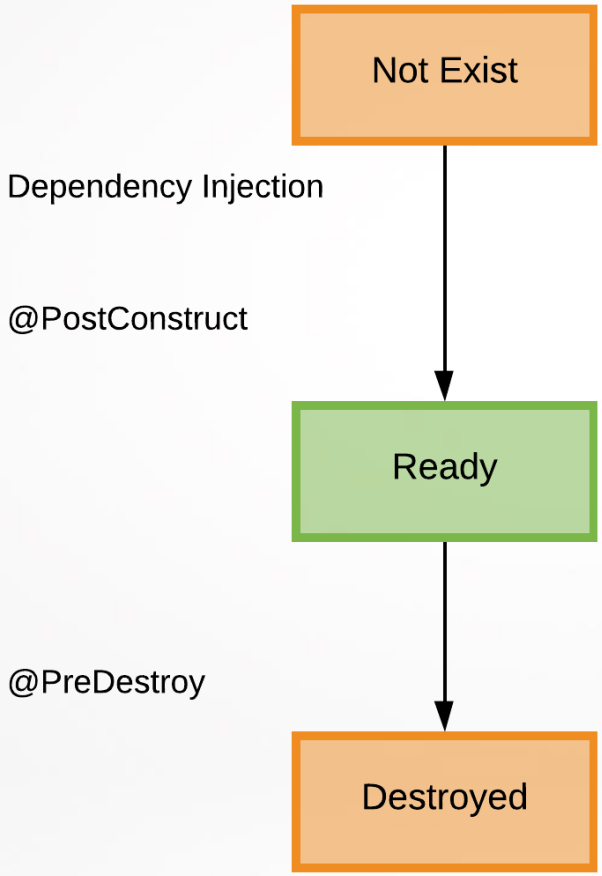
- Session Bean
  - `@Stateless`
    - `@Local`
    - `@Remote`
  - `@Stateful`
    - `@Local`
    - `@Remote`
  - `@Singleton`
    - Lazy loading strategy
    - Eagle loading strategy by using `@Startup`

- Message-Driven Bean
  - messages are processed asynchronously
  - normally acts as a JMS message listener
  - can process messages from different clients

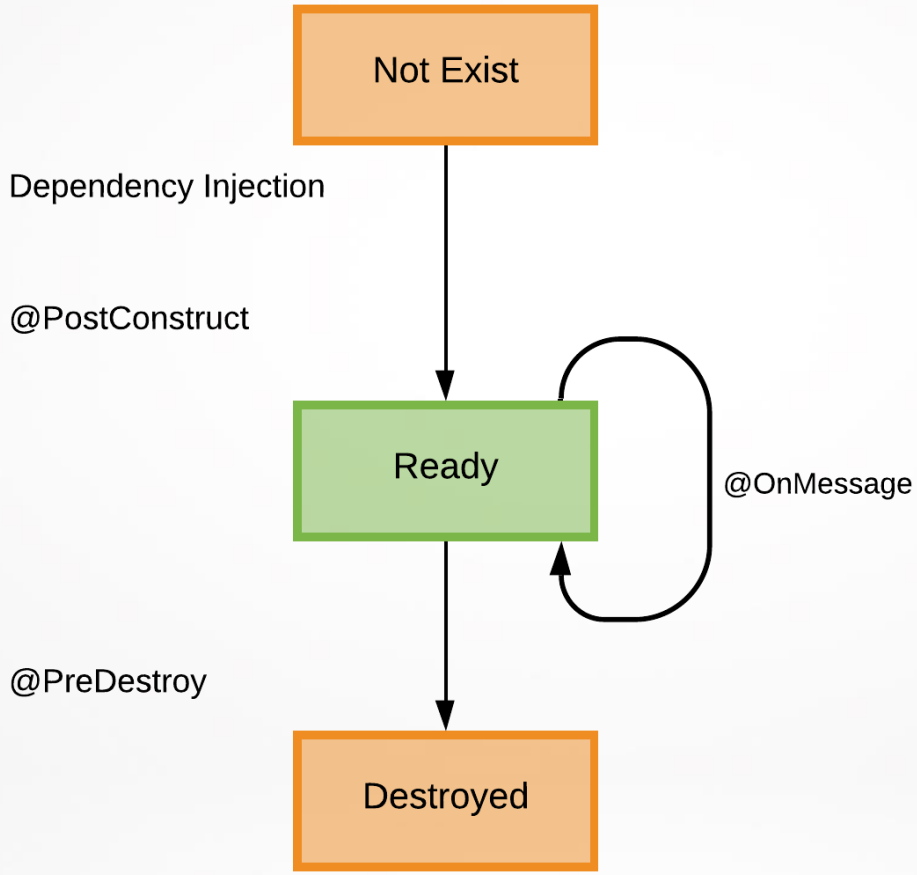
EJB – Stateful Session Bean



EJB – Stateless Session Bean



EJB – Message-Driven Bean



Summary

- Servlet
  - Well defined lifecycle
  - Scopes
  - Filter
- CDI
  - **loose coupling**
  - Provides services – eg. Contexts, dependency injection
  - Well defined lifecycle – Interceptor
  - Enhnace servlet scopes
- EJB
  - Session Bean
    - Statfull
    - Stateless
    - Sngleton
  - Message-Driven Bean

Thank You. Any questions?

Piotr Apollo
Krzysztof Grel

FIS-SST

Additional information

- https://www.oracle.com/technetwork/java/javaee/tech/index.html
- https://javaee.github.io/tutorial/
- https://javaee.github.io/tutorial/toc.html
- https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition