Jakarta EE Cheat Sheet

A quick reference guide to Jakarta EE, covering essential APIs and components for building enterprise Java applications.

Core Concepts

Containers

Definition: A container provides a runtime environment for Jakarta EE components, managing their lifecycle, dependencies, and security.

Types:

- **Web Container:** Manages web components like Servlets, JSPs, and Faces.
- **EJB Container:** Manages Enterprise JavaBeans (EJBs).

Responsibilities:

- Lifecycle management (creation, destruction).
- Dependency injection.
- Security.
- Transaction management.

Components

Servlets:	Java classes that handle HTTP requests and responses. Used for dynamic web content generation.
JSPs (JavaServer Pages):	Text-based documents that contain static content and dynamic Java code. Compiled into Servlets.
EJBs (Enterprise JavaBeans):	Server-side components that encapsulate business logic. Support transactions, security, and concurrency.
CDI Beans (Contexts and Dependency Injection):	Beans managed by the CDI container. Provide dependency injection and contextual lifecycle management.
JAX-RS Resources:	Java classes that expose RESTful web services. Annotated with `@Path`, `@GET`, `@POST`, etc.

Annotations

`@WebServlet`: Defines a Servlet.

`@EJB`: Injects an EJB.

`@Inject`: Injects a CDI bean.

`@Path`: Defines a resource path in JAX-RS.

`@GET`, `@POST`, `@PUT`, `@DELETE`: HTTP method annotations in JAX-RS.

`@Produces`, `@Consumes`: Defines MIME types in JAX-RS.

`@ApplicationScoped`, `@RequestScoped`, `@SessionScoped`: CDI scopes.

Web Tier

Servlets

Servlets handle client requests and generate responses. They are configured using annotations or deployment descriptors.

Example:

```
@WebServlet("/hello")
public class HelloServlet extends
HttpServlet {
    protected void
doGet(HttpServletRequest request,
HttpServletResponse response) throws
ServletException, IOException {

response.getWriter().println("Hello,
world!");
    }
}
```

Key methods: `init()`, `service()`, `doGet()`, `doPost()`, `destroy()`

JavaServer Pages (JSPs)

JSPs allow embedding Java code within HTML. They are compiled into Servlets.

Example:

```
<%@ page language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>Hello</title>
</head>
<body>
    <h1>Hello, <%=
request.getParameter("name") %>!</h1>
</body>
</html>
```

Implicit Objects: `request`, `response`, `session`, `application`, `out`, `pageContext`, `config`, `page`, `exception`

Jakarta Faces

A component-based UI framework for building web applications. Uses Facelets (XML-based view technology).

Key Components:

- **UI Components:** Reusable UI elements like buttons, input fields, and data tables.
- **Managed Beans:** Java classes that back the UI components.
- **Navigation:** Defining page transitions.

Example Facelets:

```
<h:inputText value="#{myBean.name}" />
<h:commandButton value="Submit"
action="#{myBean.submit}" />
```

Business Tier

Enterprise JavaBeans (EJBs)

EJBs are server-side components for encapsulating business logic. They are managed by the EJB container.

Types:

- **Session Beans:** Represent business tasks.
- **Message-Driven Beans (MDBs):** Handle asynchronous messages.

Example Session Bean:

```
@Stateless
public class MyBean {
    public String doSomething() {
        return "Something";
    }
}
```

Annotations: `@Stateless`, `@Stateful`, `@Singleton`, `@MessageDriven`, `@EJB`

Contexts and Dependency Injection (CDI)

CDI provides dependency injection and contextual lifecycle management for Java EE applications.

Key Concepts:

- **Beans:** Managed objects.
- **Scopes:** Define the lifecycle of beans (e.g., `@ApplicationScoped`, `@RequestScoped`).
- **Qualifiers:** Used to differentiate between beans of the same type.
- **Events:** Enable loosely coupled communication between components.

Example:

```
@ApplicationScoped
public class MyService {
    @Inject
    private MyRepository repository;
}
```

Transactions

Jakarta EE provides transaction management capabilities through JTA (Java Transaction API).

Annotations:

- `@Transactional`: Marks a method or class as transactional.
- `@TransactionAttribute`: Specifies the transaction behavior.

Example:

```
@Transactional
public void transfer(Account from,
Account to, double amount) {
    //...
}
```

Transaction Attributes: `REQUIRED`, `REQUIRES_NEW`, `MANDATORY`, `SUPPORTS`, `NOT_SUPPORTED`, `NEVER`

Data Tier & APIs

Jakarta Persistence (JPA)

JPA provides an API for managing relational data in Java applications. It supports object-relational mapping (ORM).

Key Concepts:

- **Entities:** Java classes that represent database tables.
- **EntityManager:** Used to persist, retrieve, update, and delete entities.
- **JPQL (Java Persistence Query Language):** Used to query entities.

Example Entity:

```
@Entity
public class Customer {
    @Id
    private Long id;
    private String name;
}
```

Annotations: `@Entity`, `@Id`, `@GeneratedValue`, `@Column`, `@Table`, `@ManyToOne`, `@OneToMany`

Jakarta RESTful Web Services (JAX-RS)

JAX-RS is an API for building RESTful web services.

Key Annotations:

- `@Path`: Defines the resource path.
- `@GET`, `@POST`, `@PUT`, `@DELETE`: HTTP method annotations.
- `@Produces`, `@Consumes`: Define MIME types.
- `@PathParam`, `@QueryParam`: Inject request parameters.

Example:

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Produces("application/json")
    public List<Customer> getAll() {
        //...
    }
}
```

Jakarta Messaging (JMS)

JMS provides an API for asynchronous messaging between applications.

Key Concepts:

- **Message:** The unit of communication.
- **Destination:** The target of a message (Queue or Topic).
- **ConnectionFactory:** Used to create connections to the JMS provider.
- **Session:** Used to create messages, producers, and consumers.

Example MDB:

```
@MessageDriven(activationConfig = {

@ActivationConfigProperty(propertyName =
"destinationType", propertyValue =
"javax.jms.Queue")
})
public class MyMDB implements
MessageListener {
    public void onMessage(Message
message) {
        //...
    }
}
```