

**Curso de Especialização em
Desenvolvimento de
Aplicações para Dispositivos
Móveis e Cloud Computing**

DM110

Desenvolvimento JavaEE

Prof. Márcio Emílio Cruz Vono de Azevedo
E-mail: marcioe@inatel.br

Agenda

- Aula 1:
 - Introdução ao Java EE
 - Preparação do ambiente de desenvolvimento
 - Utilização de componentes Web
- Aula 2:
 - Enterprise JavaBeans
 - Java Persistence API
- Aula 3:
 - Processamento concorrente com JMS e MDB

Introdução do Java EE

Introdução ao Java EE

- Java Platform, Enterprise Edition
- Tecnologia Java para construção de aplicações corporativas baseadas em Web
- Lançado pela Sun Microsystems em 1999 (empresa posteriormente adquirida pela Oracle Corporation)
- Extensão ao Java SE (Standard Edition)

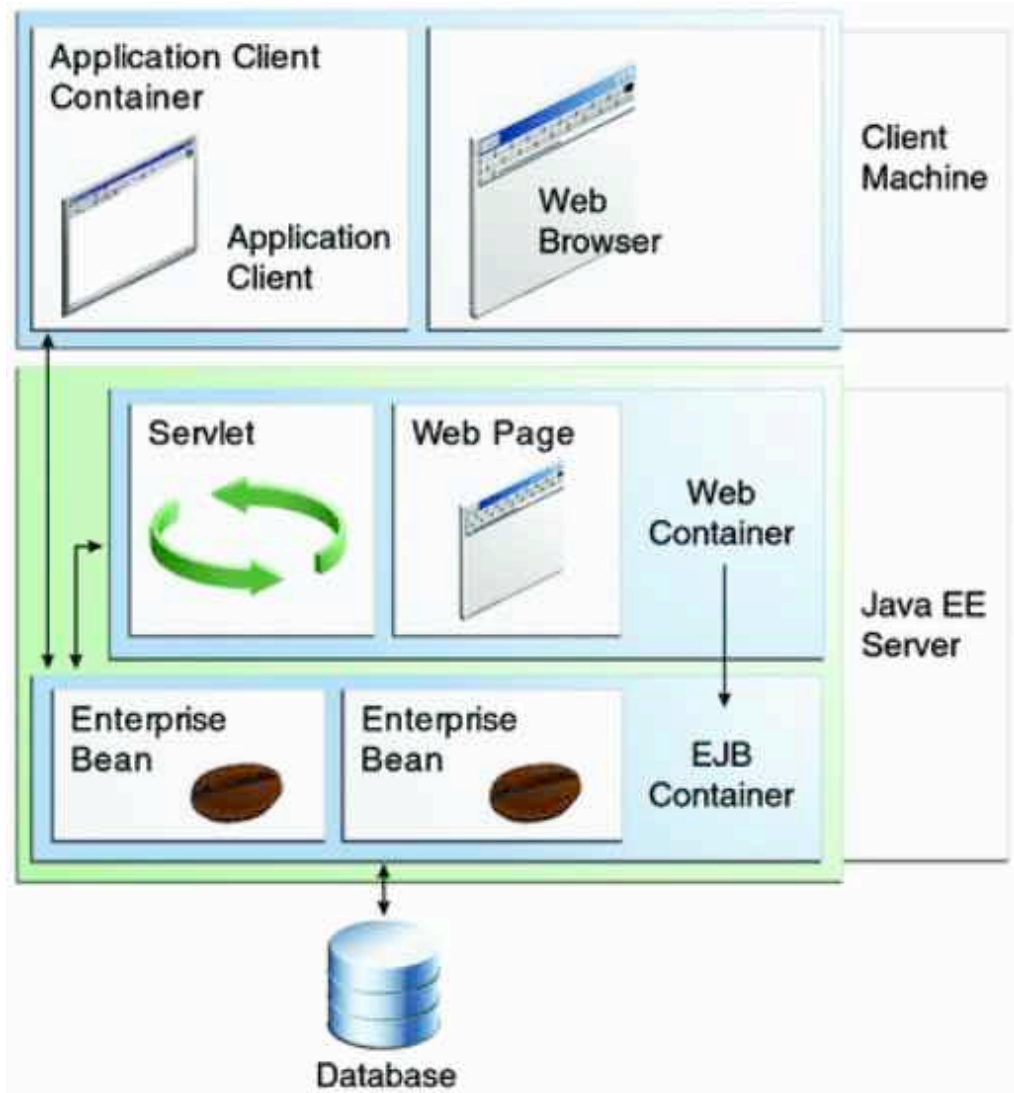
Introdução ao Java EE

- Engloba diversas tecnologias, como:
 - JNDI – Java Naming and Directory Interface
 - EJB – Enterprise JavaBeans
 - Servlets, JSP (JavaServer Pages) e JSF (JavaServer Faces)
 - JMS – Java Message Service
 - JPA – Java Persistence API
 - JMX – Java Management Extensions

Servidor de Aplicações Java EE

- Framework de software que provê recursos para execução de aplicações Java EE, como:
 - Containers para execução de componentes específicos, como:
 - Web Container
 - EJB Container
 - Serviços disponíveis para os componentes:
 - Barramento JNDI
 - Filas e tópicos JMS
 - JDBC Datasources

Servidor de Aplicações Java EE



Arquitetura em Camadas

- Camada Cliente (aplicação cliente)
- Camada Web
- Camada de Negócios
- Camada EIS (Enterprise Information Systems – Sistemas de Informações Corporativas)

Camada Cliente

- Composta por aplicações clientes
- Localizadas em uma máquina diferente do servidor Java EE
- Aplicações clientes podem ser:
 - Web browsers
 - Aplicações stand alone
 - Aplicações móveis
 - Outros servidores

Camada Web

- Gera conteúdo dinamicamente para os clientes
- Coleta entradas dos usuários e processa os resultados dos componentes da camada de negócios
- Controla o fluxo de janelas e páginas no cliente
- Realiza lógicas básicas
- Mantém dados temporariamente em componentes JavaBeans
 - Classes serializáveis
 - Construtor padrão
 - Getters e Setters para suas propriedades

Tecnologias da Camada Web

- Servlets – classes Java que processam requisições HTTP
- JavaServer Pages (JSP) – documentos textos, semelhantes a páginas HTML, que processam conteúdo dinâmico gerando resultados estáticos, como páginas HTML
- JavaServer Pages Standards Tag Library (JSTL) – biblioteca de tags que encapsulam funcionalidades em páginas JSP
- JavaServer Faces – framework baseado em componentes para construção de interfaces de usuário
- Componentes JavaBeans – componentes que armazenam temporariamente os dados de uma aplicação

Camada de Negócios

- Possui componentes que encapsulam a lógica de negócios da aplicação
- Processa e envia para armazenamento os dados de negócios
- Recupera as informações de negócios e disponibiliza para a aplicação corporativa

Tecnologias da Camada de Negócios

- Enterprise JavaBeans (EJB) – componentes que lidam com a parte lógica de negócio da aplicação corporativa
- Java Persistence API (JPA) – suporta o mapeamento objeto-relacional, permitindo o acesso aos bancos de dados relacionais
- Java Message Service (JMS) – permite aos componentes criarem, enviarem, receberem e processarem mensagens. Integrado com Message-Driven EJB para processamento das mensagens
- Java API for XML Web Services (JAX-WS) – permite a criação de Web Services

Camada EIS

- Sistemas de informações corporativos, externos ao servidor Java EE
- EIS podem ser:
 - Servidores de Banco de Dados
 - Mainframes
 - ERPs
 - Sistemas legados

Tecnologias da Camada EIS

- Java Database Connectivity API (JDBC) – provê conexão com bases de dados
- Java EE Connector Architecture – permite a conexão entre as aplicações corporativas Java EE e os demais Sistemas de Informações Corporativas

Servidor de Aplicações Java EE

- Hospeda as aplicações corporativas
- Provê serviços como segurança, gerenciamento de transações, serviços de pesquisa de nomes (JNDI – Java Naming and Directory Interface) e conectividade remota
- Gerencia os componentes das aplicações corporativas através de containers
 - Web Container – gerencia a execução das páginas Web dinâmicas e Servlets
 - EJB Container – gerencia a execução dos componentes corporativos

Ambiente de Desenvolvimento de Aplicações Java EE

Ferramentas de Desenvolvimento

- Eclipse IDE for Java EE Developers
- WildFly Application Server
- Ferramentas de construção de builds:
 - Apache Maven
 - Gradle

WildFly Application Server

- Servidor de aplicações Java EE livre e de código aberto (LGPL)
- Anteriormente conhecido como JBoss AS
- Desenvolvido pela Red Hat, Inc.

Apache Maven

- Ferramenta de construção de aplicações e gerenciamento de dependências
- Configuração por arquivos XML
- Provê ciclo de vida pré-definido (ao contrário do Ant onde o programador era responsável por criar o ciclo de vida)
- Possui plugins específicos para aplicações Java EE

Estrutura de Projeto Maven Java EE

- newproject/
 - | -- pom.xml
 - | -- newproject-war/
 - | `-- pom.xml
 - | -- newproject-ejb/
 - | `-- pom.xml
 - | -- newproject-ejb-client/
 - | `-- pom.xml
 - `-- newproject-ear/
 - `-- pom.xml

Projeto Pai/Agregador Maven

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.company.newgroup</groupId>
  <artifactId>newproject</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>Example Project</name>
  <modules>
    <module>newproject-war</module>
    <module>newproject-ejb</module>
    <module>newproject-ejb-client</module>
    <module>newproject-ear</module>
  </modules>
  <dependencyManagement>
    <!-- ... -->
  </dependencyManagement>
</project>
```

Projeto EAR Maven

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>newproject</artifactId>
    <groupId>com.company.newgroup</groupId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <artifactId>newproject-ear</artifactId>
  <name>Example Project EAR</name>
  <packaging>ear</packaging>
  <dependencies>
    <!-- ... -->
  </dependencies>
  <!-- ... -->
```


Projeto EAR Maven

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-ear-plugin</artifactId>
      <configuration>
        <modules>
          <ejbModule>
            <groupId>com.company.newgroup</groupId>
            <artifactId>newproject-ejb</artifactId>
          </ejbModule>
          <webModule>
            <groupId>com.company.newgroup</groupId>
            <artifactId>newproject-war</artifactId>
            <context-root>/newproject</context-root>
          </webModule>
          <jarModule>
            <groupId>com.company.newgroup</groupId>
            <artifactId>newproject-ejb-client</artifactId>
          </jarModule>
        </modules>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Projeto WAR Maven

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>newproject</artifactId>
    <groupId>com.company.newgroup</groupId>
    <version>1.0.0-SNAPSHOT</version>
  </parent>
  <groupId>com.company.newgroup</groupId>
  <artifactId>newproject-war</artifactId>
  <packaging>war</packaging>
  <name>Example Project Web ARchive</name>
  <dependencies>
    <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>newproject-ejb-client</artifactId>
      <version>${project.version}</version>
    </dependency>
  </dependencies>
</project>
```

Gradle

- Ferramenta de construção de aplicações e gerenciamento de dependências
- Configuração em linguagem Groovy
- Configuração muito menos “prolixa” do que o Maven
- Possui plugins específicos para aplicações Java EE

Estrutura de Projeto Gradle Java EE

- newproject/
 - | -- build.gradle
 - | -- settings.gradle
 - | -- newproject-war/
 - | `-- build.gradle
 - | -- newproject-ejb/
 - | `-- build.gradle
 - | -- newproject-ejb-client/
 - | -- newproject-ear/
 - | `-- build.gradle

Configuração do projeto raiz (build.gradle)

```
subprojects {  
    apply plugin: 'java'  
    apply plugin: 'eclipse'  
    repositories {  
        mavenCentral()  
    }  
    configurations {  
        provided  
    }  
    sourceSets {  
        main {  
            compileClasspath += configurations.provided  
            eclipse.classpath.plusConfigurations += [configurations.provided]  
        }  
        test {  
            compileClasspath += configurations.provided  
            runtimeClasspath += configurations.provided  
            eclipse.classpath.plusConfigurations += [configurations.provided]  
        }  
    }  
    dependencies {  
        testCompile 'junit:junit:4.12'  
        provided 'javax:javaee-api:7.0'  
    }  
    version = '1.0.0-SNAPSHOT'  
}
```

Configuração do projeto raiz (settings.gradle)

- Todos os subprojetos listados herdarão as configurações definidas na área “subprojects” do projeto raiz:

include "newproject-ear", "newproject-war", "newproject-ejb", "newproject-ejb-client"

Projeto EAR (build.gradle)

```
apply plugin: 'ear'

dependencies {
    deploy project(path: ':newproject-war', configuration: 'archives')
    deploy project(path: ':newproject-ejb')
    earlib project(':newproject-core')
    earlib project(':newproject-api')
    earlib project(':newproject-ejb-client')
}

ear {
    deploymentDescriptor {
        def warName = project(':newproject-war').name
        def warVersion = project(':newproject-war').version
        def warFileName = warName + '-' + warVersion + '.war'
        webModule(warFileName, 'newproject')

        def ejbName = project(':newproject-ejb').name
        def ejbVersion = project(':newproject-ejb').version
        def ejbFileName = ejbName + '-' + ejbVersion + '.jar'
        module(ejbFileName, 'ejb')
    }
}
```

Projeto WAR (build.gradle)

```
apply plugin: 'war'
```

```
jar.enabled = false
```


Servlets

Servlets

- Classes Java que respondem a requisições HTTP:
 - doGet – HTTP GET
 - doPost – HTTP POST
 - doPut – HTTP PUT
 - delete – HTTP DELETE
- Implementação da classe:
`javax.servlet.http.HttpServlet`

Hello World Servlet

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello World!</h1>");
    }
}
```

Exercício Proposto

- Criar uma calculadora de 4 operações através de um Servlet

Enterprise JavaBeans

Enterprise JavaBeans

- Componentes que encapsulam a lógica de negócios de uma aplicação
- Gerenciados pelo EJB container
- Existem dois tipos de EJB:
 - Session Beans
 - Message-Driven Beans
- Antigamente existiam os Entity Beans, mas foram substituídos pela especificação JPA

Enterprise JavaBeans

- Session Beans:
 - Componentes que encapsulam a lógica de negócios da aplicação e podem ser invocados programaticamente através de suas interfaces
- Message-Driven Beans:
 - Componentes capazes de processar mensagens de forma assíncrona

Session Beans

- Stateful session beans
 - Mantém estado entre as chamadas
 - Permanecem associados exclusivamente a um cliente
- Stateless session beans
 - Não mantém estado entre as chamadas
 - Podem ser compartilhados por mais de um cliente, sendo liberados após a invocação
- Singleton session beans
 - Possui apenas uma instância em todo o ciclo de vida da aplicação

Interfaces de em Session Bean

- Session Beans podem possuir interfaces locais ou remotas:
 - Local: para acessos locais, em uma mesma JVM
 - Remote: para acessos remotos, entre diferentes instâncias de JVMs

Definindo as Interfaces do Session Bean

```
public interface Hello {  
    public String sayHello(String name);  
}
```

```
public interface HelloLocal extends Hello {  
}
```

```
public interface HelloRemote extends Hello {  
}
```

Criando um Session Bean

@Stateless

@Remote(HelloRemote.class)

@Local(HelloLocal.class)

public class HelloBean implements Hello {

 @Override

 public String sayHello(String name) {

 return "Saying hello to " + name + " from EJB!!!";

 }

}

Usando o Session Bean

```
@WebServlet("/hello/servlet")
public class HelloServlet extends HttpServlet {

    @EJB(mappedName="java:app/prop-poc-ejb-1.0-SNAPSHOT/HelloBean!
br.inatel.poc.ejb.interfaces.HelloRemote")
    private Hello helloBean;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String name = req.getParameter("name");
        resp.setContentType("text/html");
        resp.getWriter().print("<h1>" + helloBean.sayHello(name) + "</h1>");
    }
}
```

Java Persistence API

Java Persistence API

- Provê mapeamento de dados em bancos de dados relacionais para classes e objetos Java

Principais Annotations JPA

- `@Entity`: declara uma classe como entidade ou tabela
- `@Table`: declara o nome de uma tabela
- `@Column`: declara o nome de uma coluna
- `@Id`: especifica a chave primária de uma tabela
- `@GeneratedValue`: especifica valores gerados automaticamente
- `@SequenceGenerator`: especifica uma sequência
- `@Transient`: Especifica um atributo não persistente
- `@OneToOne`, `@OneToMany`, `@ManyToOne` e `@ManyToMany`: especifica como duas tabelas são ligadas (join)

Exemplo de Mapeamento Objeto/Relacional

```
create table product (  
    id integer not null,  
    name varchar(50) not null,  
    quantity integer not null default 0,  
    constraint pk_product primary key (id)  
);  
  
create sequence seq_product;
```


Exemplo de Mapeamento Objeto/Relacional

```
@Entity
@SequenceGenerator(name = "seq_product",
                    sequenceName = "seq_product", allocationSize = 1)
public class Product {
    @Id
    @GeneratedValue(generator = "seq_product",
                    strategy = GenerationType.SEQUENCE)
    private Integer id;

    private String name;
    private Integer quantity;

    // Getters e setters...

}
```

Configurando Persistence Unit

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="dm110">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <non-jta-data-source>java:/jdbc/dm110-ds</non-jta-data-source>
    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.PostgreSQLDialect" />
      <property name="hibernate.show_sql" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

Criando o DAO

@Stateless

```
public class ProductDAO {
```

```
    @PersistenceContext(unitName = "dm110")
```

```
    private EntityManager em;
```

```
    public void insert(Product product) {
```

```
        em.persist(product);
```

```
    }
```

```
}
```

Configuração do DataSource no WildFly

Configurando o Driver JDBC do PostgreSQL

- Criar a pasta:
\${WILDFLY_HOME}/modules/system/layers/base/org/postgresql/main/
- Adicionar o jar do Driver JDBC do PostgreSQL
- Criar o arquivo module.xml com o seguinte conteúdo:

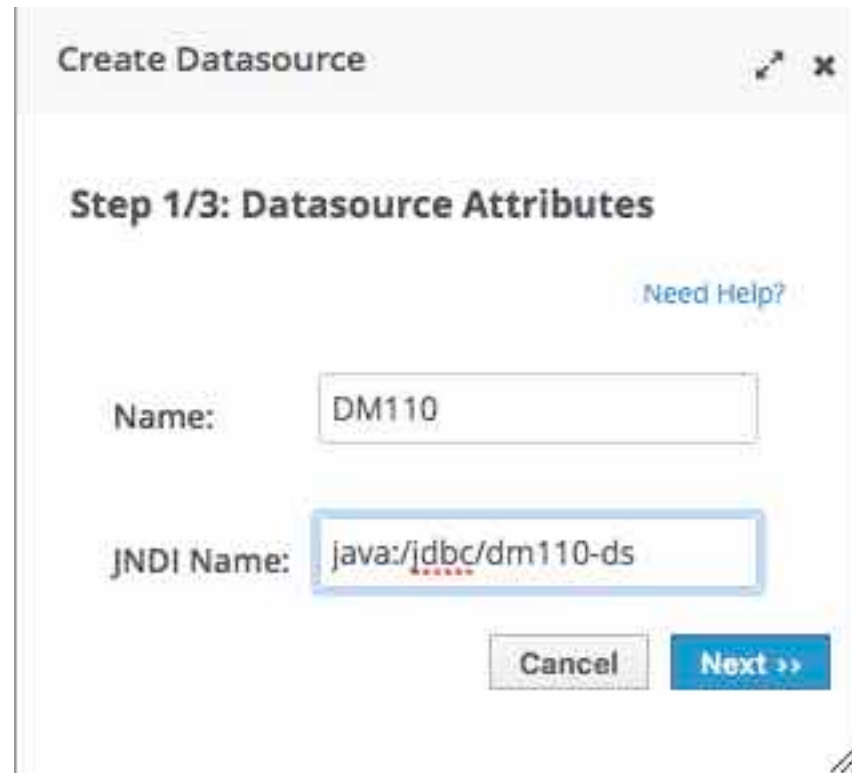
```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.postgresql">
  <resources>
    <resource-root path="postgresql-9.3-1101.jdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api" export="true"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Configurando o Driver JDBC do PostgreSQL

- Localize a tag `<drivers>...</drivers>` no arquivo de configuração do WildFly (ex.: `standalone.xml`)
- Inclua o Driver do PostgreSQL:

```
<driver name="PostgreSQL" module="org.postgresql">  
    <driver-class>org.postgresql.Driver</driver-class>  
</driver>
```

Criando o DataSource



The screenshot shows a 'Create Datasource' dialog box with the title bar 'Create Datasource' and standard window controls. The main content area is titled 'Step 1/3: Datasource Attributes' and includes a 'Need Help?' link. There are two text input fields: 'Name' with the value 'DM110' and 'JNDI Name' with the value 'java:/jdbc/dm110-ds'. The 'JNDI Name' field has a blue border and red dots below it, indicating it is the current focus. At the bottom right, there are 'Cancel' and 'Next >>' buttons.

Create Datasource

Step 1/3: Datasource Attributes

[Need Help?](#)

Name: DM110

JNDI Name: java:/jdbc/dm110-ds

Cancel Next >>

Criando o DataSource

Create Datasource

Step 2/3: JDBC Driver

Select one of the installed JDBC driver. Don't see your driver? Please make sure it's deployed as a module and properly registered.

Detected Driver

Specify Driver

Name
oracle
h2
PostgreSQL

<<

<

1-3 of 3

>

>>

Cancel

Next >>

Criando o DataSource

Create Datasource

Step 3/3: Connection Settings

[Need Help?](#)

Connection URL: jdbc:postgresql://localhost:5432/dm110

Username: postgres

Password: *****

Security Domain:

Test Connection

Cancel Done

Exercício

- Criar um cadastro de clientes

Message-Driven Bean

Message-Driven Bean

- Tipo de EJB usado para que aplicações Java EE possam processar mensagens assincronamente
- Age como um listener JMS (Java Messaging System)
- Podem receber mensagens de uma fila ou tópico

Tópicos JMS

- Possui semântica publish/subscribe
- Quando uma mensagem é publicada ela é recebida por todos os subscribers que tenham interesse
- Zero ou mais subscribers receberão uma cópia da mensagem
- Somente os subscribers ativos no momento em que a mensagem for publicada irão receber uma cópia da mensagem

Filas JMS

- Implementa semântica de balanceamento de carga
- Cada mensagem será recebida por exatamente um consumidor
- Se nenhum consumidor estiver disponível, a mensagem será mantida na fila até que um consumidor possa processá-la
- A fila pode ter muitos consumidores, mas as mensagens serão entregues de forma balanceada aos consumidores disponíveis

Criando um JMS Endpoint

- No console de administração do WildFly, acesse Configuration → Messaging → Destinations → Default → View → Queue/Topics → Queues → Add
- Adiciona a fila conforme a configuração abaixo:

Name:	<input type="text" value="DM110Queue"/>
JNDI Names:	<input type="text" value="java:/jms/queue/dm110queue"/>
Durable?:	<input checked="" type="checkbox"/>

Criando o Message-Driven Bean

```
@MessageDriven(activationConfig = {  
    @ActivationConfigProperty(propertyName = "destinationType",  
        propertyValue = "javax.jms.Queue"),  
    @ActivationConfigProperty(propertyName = "destination",  
        propertyValue = "java:/jms/queue/dm110queue")  
})  
  
public class HelloMDB implements MessageListener {  
  
    @Override  
    public void onMessage(Message message) {  
        // Message processing...  
    }  
  
}
```


Produzindo Mensagens

@Stateless

```
public class HelloMessageSender {  
    @Resource(mappedName = "java:/ConnectionFactory")  
    private ConnectionFactory connectionFactory;  
    @Resource(mappedName = "java:/jms/queue/dm110queue")  
    private Queue queue;  
    public void sendMessage() {  
        try (  
            Connection connection = connectionFactory.createConnection();  
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  
            MessageProducer producer = session.createProducer(queue);  
        ) {  
            MyObject obj = new MyObject();  
            ObjectMessage message = session.createObjectMessage(obj);  
            producer.send(message);  
        } catch (JMSException e) {  
            // exception handling  
        }  
    }  
}
```

Projeto

- Detecção de equipamentos ativos em uma sub-rede