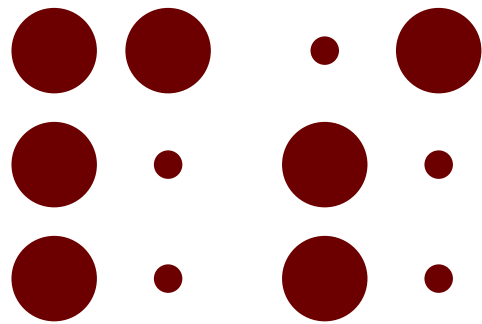


Jakarta EE Programming

Instructor: Juan Carlos Trejo

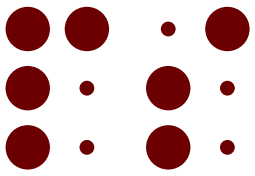




Módulo 1

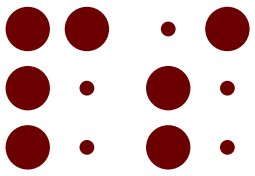
Introducción a Jakarta EE



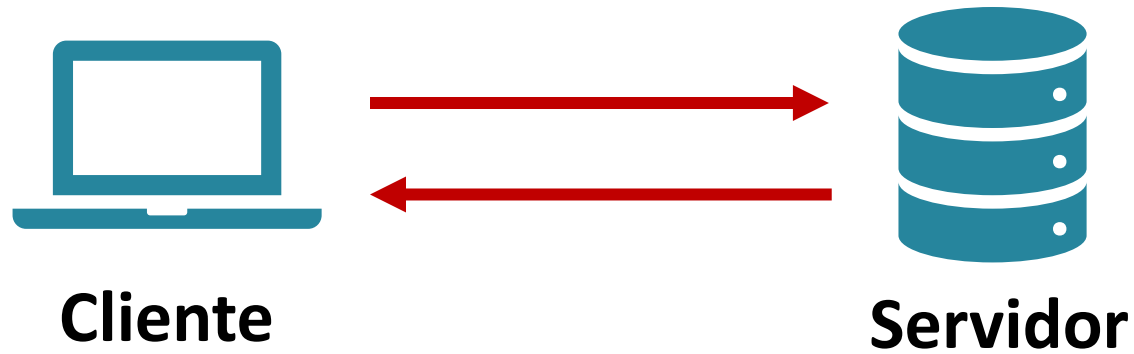


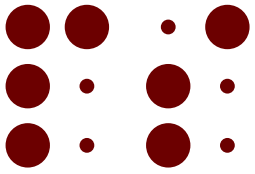
Sistemas distribuidos

- Los sistemas distribuidos dividen el trabajo entre módulos independientes.
- La división del trabajo tiene las siguientes características:
 - ✓ Disponibilidad
 - ✓ Escalabilidad
 - ✓ Mantenimiento



Sistemas distribuidos

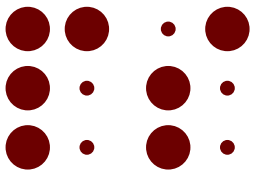




Introducción a Jakarta EE

La plataforma Jakarta EE es un modelo de aplicación distribuida de varios niveles que tiene tres niveles comunes ampliamente utilizados. Estos niveles son el nivel de presentación (o nivel web), el nivel comercial o de negocios y el nivel de integración (o nivel EIS).

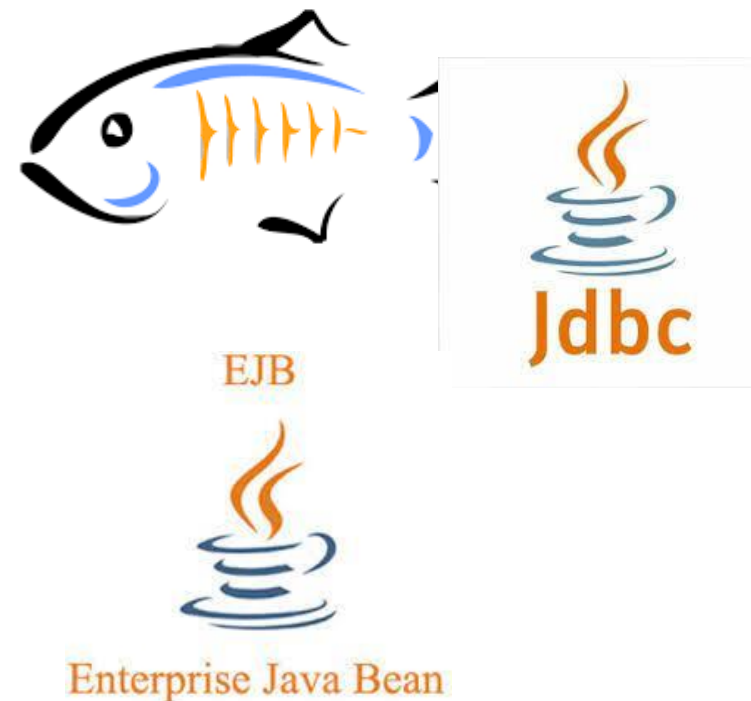


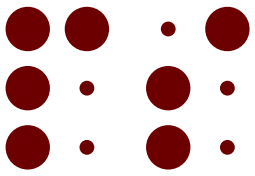


Introducción a Jakarta EE

La plataforma Jakarta EE también es una especificación:

- JDBC
- WS
- EJB
- JSF
- JSP
- Servlets



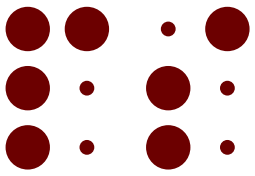


Java Naming Directory Interface

Es un API de Java para acceso a servidores de nombramiento y directorio.

JNDI es utilizado para trabajo con:

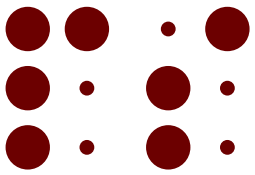
- RMI
- JMS
- DataSource para JDBC.



Jakarta Servlet

Introducción a Jakarta EE

- Un Servlet es un programa Java que se ejecuta del lado del servidor, acepta peticiones de clientes y genera respuestas dinámicamente
- El tipo más común de los Servlets es una clase Java que extiende de `HttpServlet`, acepta peticiones y genera respuestas HTTP

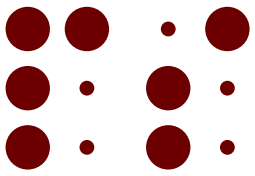


Jakarta Server Page

Introducción a Jakarta EE

Una JSP es un documento HTML, con código Java embebido entre código HTML. Una JSP realiza las siguientes tareas:

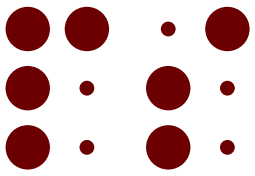
- Provee una respuesta dinámica basada en la petición que solicitó el cliente.
- Se compila y se ejecuta como un Servlet.



Enterprise Java Beans EJB

Un EJB es un componente de modelo que está escrito en Java y tiene la siguientes características:

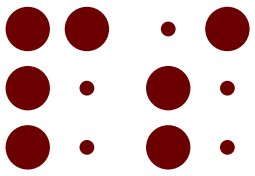
- Provee unidades de negocio.
- Se ejecutan dentro de un contenedor que provee manejo y control de servicios.



Jakarta Transaction API

El JTA es una API de Java para la gestión de transacciones dentro de un programa Java.

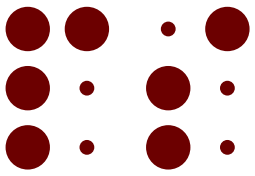
- Begin
- Commit
- Rollback



Jakarta Faces JSF

Es un framework de desarrollo web Java que proporciona componentes de interfaz de usuario y una arquitectura para crear aplicaciones web con arquitectura MVC.

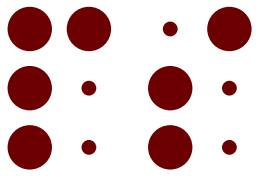
JSF administra la creación, actualización y destrucción de los componentes de la interfaz de usuario, y permite la validación de datos y la navegación entre páginas de manera sencilla.



Applets

Programas Java que son obtenidos desde el y lanzados en el cliente a travez de un navegador.

Por seguridad, estos no pueden acceder al sistema de archivos de la maquina local donde se ejecutan.



Introducción a Jakarta EE

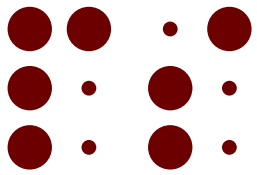
Servidor Web



VS

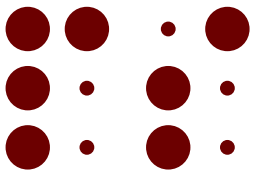


Servidor de
Aplicaciones



Un **Servidor web** es generalmente cualquier software de servidor que sirve contenido utilizando los protocolos http (o https). Estos servidores tienden a escuchar en puertos específicos como los puertos (80 o 443) por ejemplo.

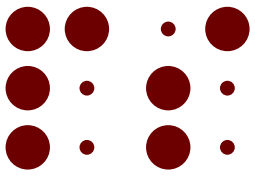




Introducción a Jakarta EE

Un **servidor de aplicaciones** generalmente gestiona la mayor parte de las funciones de lógica de negociación y de acceso a los datos de las aplicaciones. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. Contiene todos los servicios de Jakarta EE.





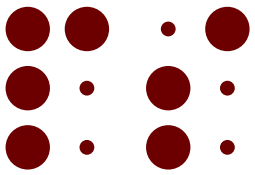
JavaBean

Una JavaBean es una clase de Java que sigue ciertas convenciones de programación para ser utilizada en entornos de desarrollo de software basados en Java. Estas convenciones permiten que las JavaBeans sean fácilmente integradas y reutilizadas en aplicaciones Java.

1. Atributos
2. Métodos setter y getter
3. Constructores
4. Serializable

Introducción a Jakarta EE

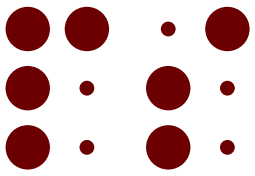
```
public class Alumno {  
  
    private int id;  
    private String nombre;  
  
    // Getter y Setter para la propiedad "id"  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    // Getter y Setter para la propiedad "nombre"  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```



Introducción

GlassFish Server es un servidor de aplicaciones de código abierto que implementa las tecnologías de la plataforma Java Enterprise Edition (Java EE). Fue desarrollado por Sun Microsystems y posteriormente continuó su desarrollo bajo Oracle Corporation.

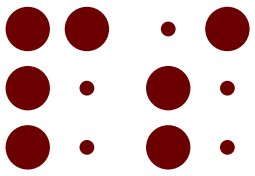




Introducción

- 1. Compatibilidad con Jakarta EE:** GlassFish es compatible con las especificaciones Jakarta EE, lo que permite crear y desplegar aplicaciones empresariales.
- 2. Administración y monitoreo:** Proporciona herramientas de administración y monitoreo para gestionar el ciclo de vida de las aplicaciones y garantizar su rendimiento.



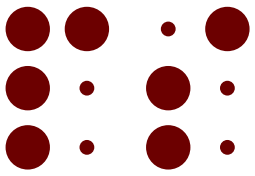


Introducción a Jakarta EE

Introducción

Desde el 2018 Java EE es gestionado por la Fundación Eclipse. La Fundación Eclipse fue forzada a eliminar la palabra Java del nombre debido a que Oracle posee el registro de la marca "Java". El 26 de febrero de 2018 se anunció que el nuevo nombre de Java EE sería Jakarta EE.





Software a utilizar

- GlassFish Server 5.1

[Eclipse GlassFish 7.x Downloads](#)

- Eclipse IDE

[Eclipse downloads - Select a mirror | The Eclipse Foundation](#)

- MySQL Server 8.0

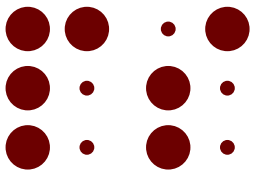
[MySQL :: Download MySQL Community Server](#)

- Base de datos de ejemplo

[MySQL :: Other MySQL Documentation](#)

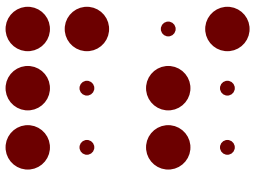
- SOAP UI

[Download REST & SOAP Automated API Testing Tool | Open Source | SoapUI](#)



Instalación de Glassfish

1. Descargar del servidor de aplicaciones Glassfish
2. Descomprimir el archivo zip en la ubicación de preferencia.
3. Abrir el archivo **asenv.bat** ubicado en *\glassfish\config y agregar al final de este, la siguiente instrucción
 - `set AS_JAVA=C:\Program Files\Java\jdk-17.0.4`
4. Agregar al *path* del SO la ruta a la carpeta de *bin* de la instalación de Glassfish.



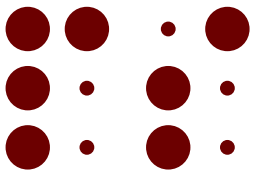
Instalación y ejecución de Glassfish

5. Para iniciar el servidor, ejecutar el comando:

- `C:>asadmin start-domain`

6. Para detener el servidor, ejecutar el comando:

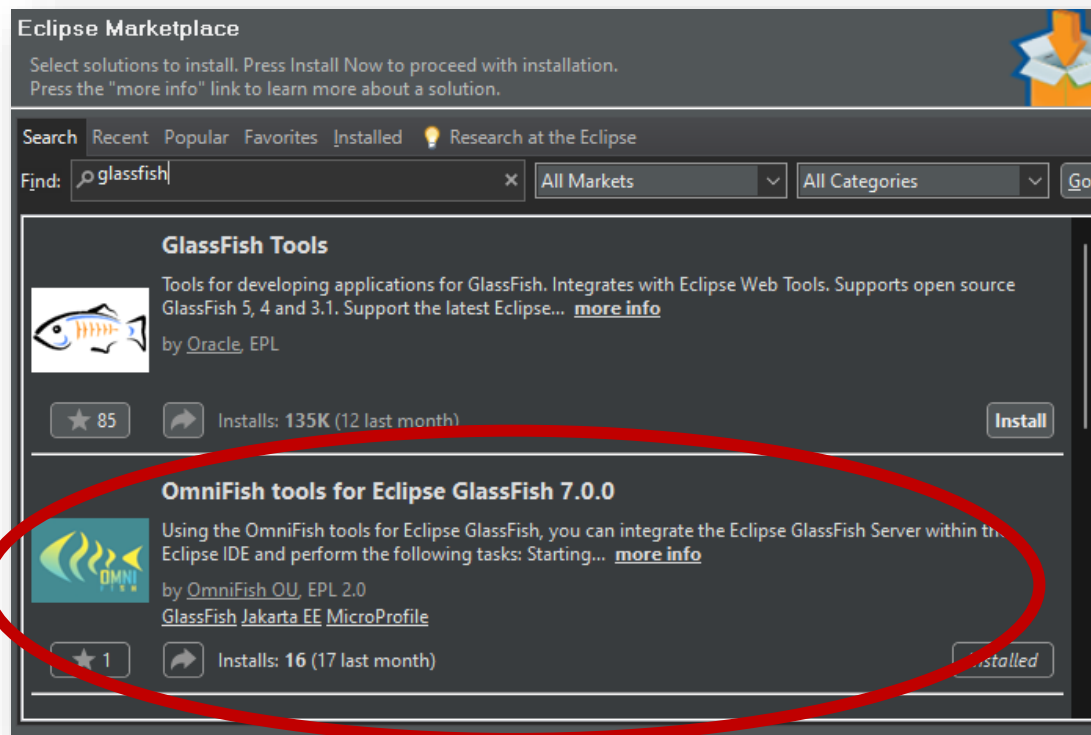
- `C:>asadmin stop-domain`

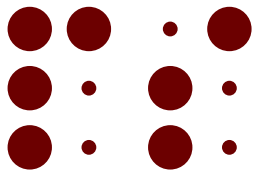


Introducción a Jakarta EE

Instalación y ejecución de Glassfish desde Eclipse

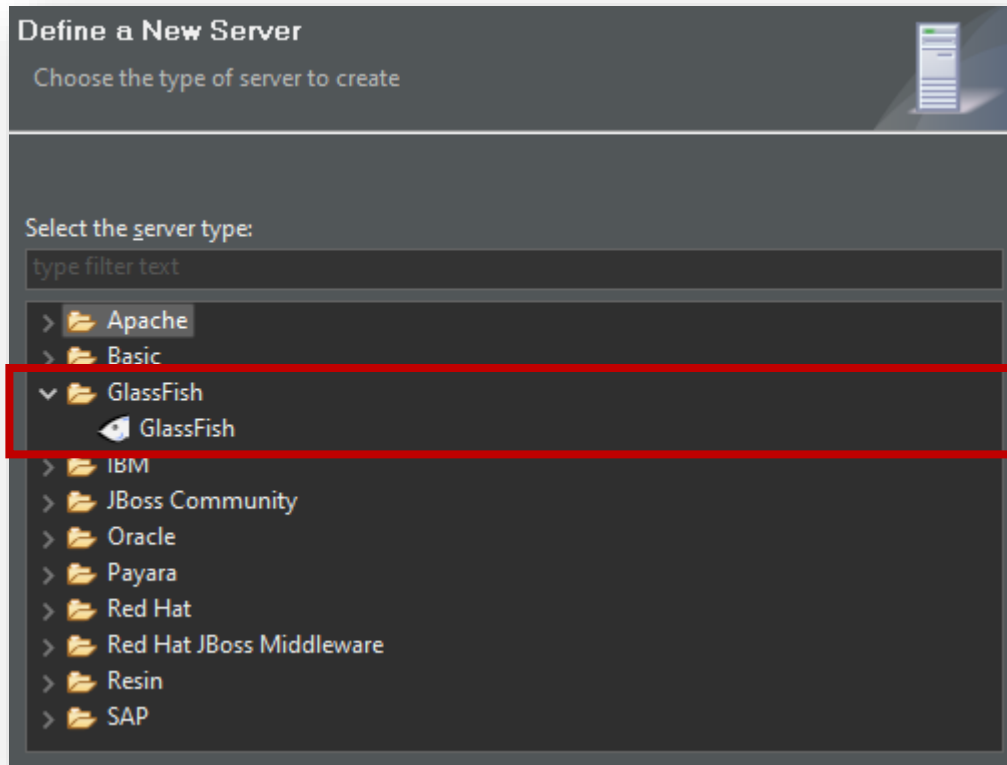
Instalar el plugin de Glassfish

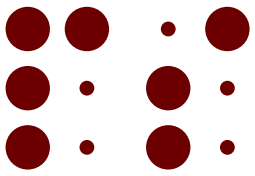




Instalación y ejecución de Glassfish desde Eclipse

Agregar el servidor de Glassfish





Instalación y ejecución de Glassfish desde Eclipse

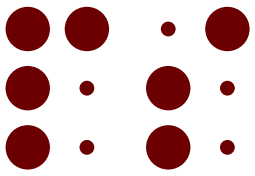
Agregar la ubicación del servidor de Glassfish

GlassFish
Define GlassFish runtime properties.

Name:
GlassFish 5

GlassFish Location:
C:\servers\glassfish5.0\glassfish5

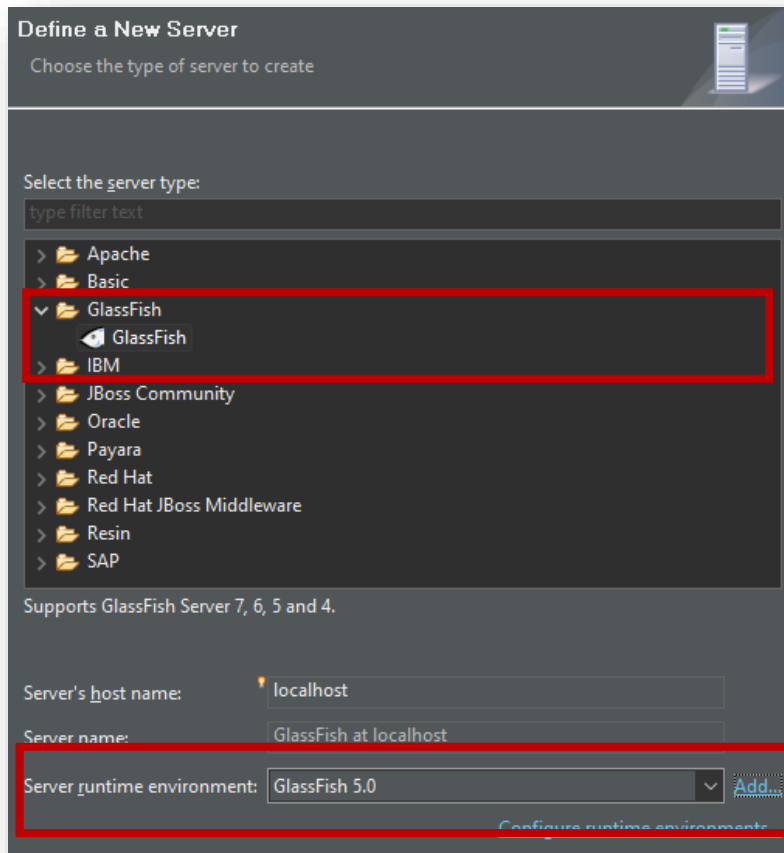
JRE:
jdk1.8.0_202

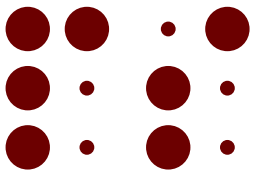


Introducción a Jakarta EE

Instalación y ejecución de Glassfish desde Eclipse

Seleccionar el runtime para el servidor





Introducción a Jakarta EE

Instalación y ejecución de Glassfish desde Eclipse

Dejar todos los valores por default

The screenshot shows the 'GlassFish' dialog box with the title 'Define GlassFish Application Server properties.' and the GlassFish logo. The fields are filled with default values: Name is 'GlassFish 5.0 [domain1]', Host name is 'localhost', Domain path is 'C:\servers\glassfish5.0\glassfish5\glassfish\domains\domain1', Admin name is 'admin', Admin password is empty, Debug port is '9009'. The 'Preserve sessions across redeployment' checkbox is checked, while 'User jar archives for deployment' is unchecked. The 'Restart pattern' is set to '\.(jar|class|xml)\$'. The 'Enable the Hot Deploy mode' and 'Attach debugger early' checkboxes are also unchecked.

GlassFish
Define GlassFish Application Server properties.

Name:
GlassFish 5.0 [domain1]

Host name:
localhost

Domain path:
C:\servers\glassfish5.0\glassfish5\glassfish\domains\domain1 Browse

Create

Admin name:
admin

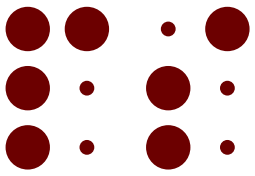
Admin password:

Debug port:
9009

☒ Preserve sessions across redeployment:
☐ User jar archives for deployment:

Restart pattern:
\.(jar|class|xml)\$

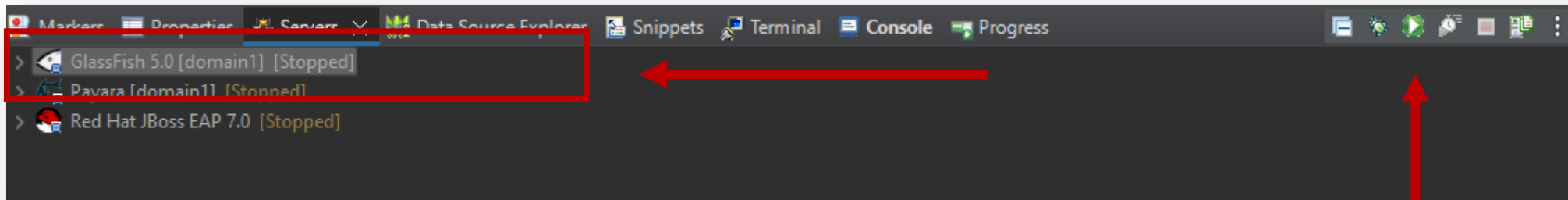
☐ Enable the Hot Deploy mode:
☐ Attach debugger early:



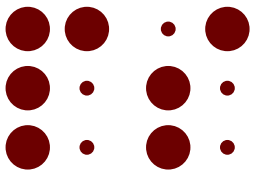
Introducción a Jakarta EE

Instalación y ejecución de Glassfish desde Eclipse

Iniciar el servidor



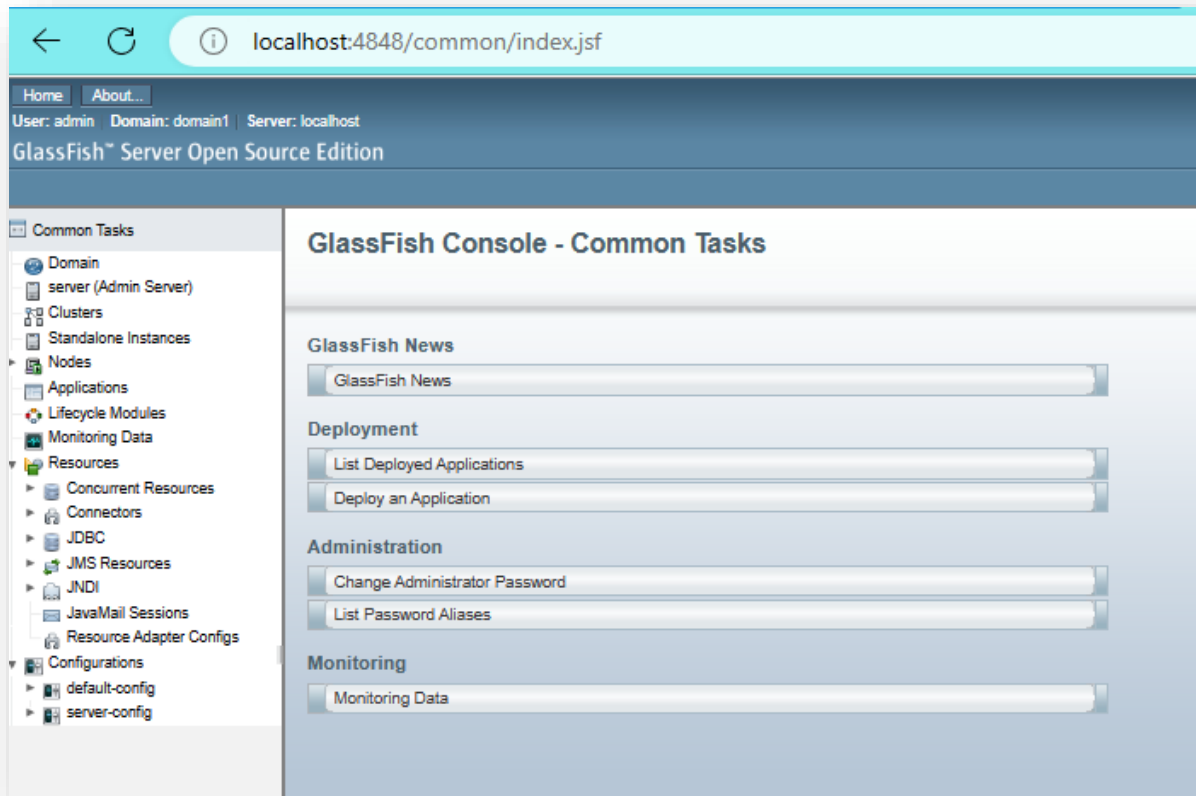
Iniciar el servidor

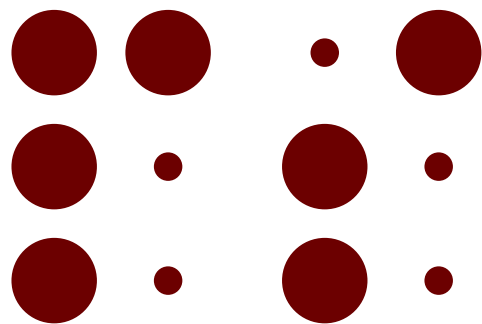


Introducción a Jakarta EE

Instalación y ejecución de Glassfish desde Eclipse

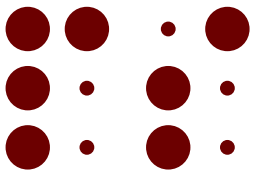
Visitar la url <http://localhost:4848>





Módulo 2

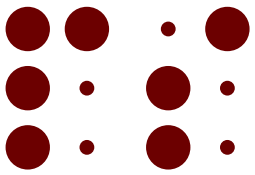
Introducción a Maven



Introducción a Maven

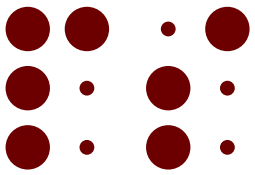
Es una herramienta de gestión y de construcción de proyectos Java a partir de un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.





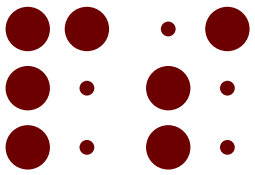
Características:

- Gestión de dependencias que incluye actualización automática.
- Capaz de trabajar fácilmente con múltiples proyectos al mismo tiempo.
- Un repositorio grande y creciente de bibliotecas.
- Compilaciones basadas en modelos: Maven puede compilar cualquier número de proyectos en tipos de salida predefinidos, como JAR, WAR



Archivo POM

POM son las siglas de "Project Object Model". Es una representación XML de un proyecto Maven que se encuentra en un archivo llamado pom.xml. De hecho, en el mundo de Maven, un proyecto no necesita contener ningún código, simplemente un pom.xml.



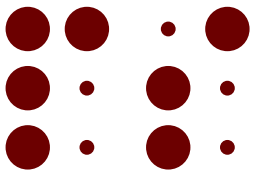
Introducción a Maven

groupId

Representa el grupo o la organización a la que pertenece el proyecto, y usualmente sigue una convención de nombres similar a la de los paquetes de Java, usando un nombre de dominio invertido para garantizar que sea único.

mx.org.banxico, org.apache.maven

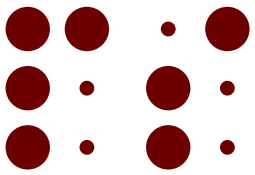




artifactId

Es el nombre del archivo de salida sin versión. Se puede elegir el nombre con letras minúsculas y sin caracteres especiales a excepción del guión medio (-)

- cei
- maven-project



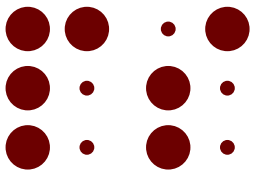
version

Se puede elegir cualquier versión típica con números y puntos (1.0, 1.1, 1.0.1, ...). No se debe utilizar fechas, ya que generalmente se asocian con compilaciones SNAPSHOT.

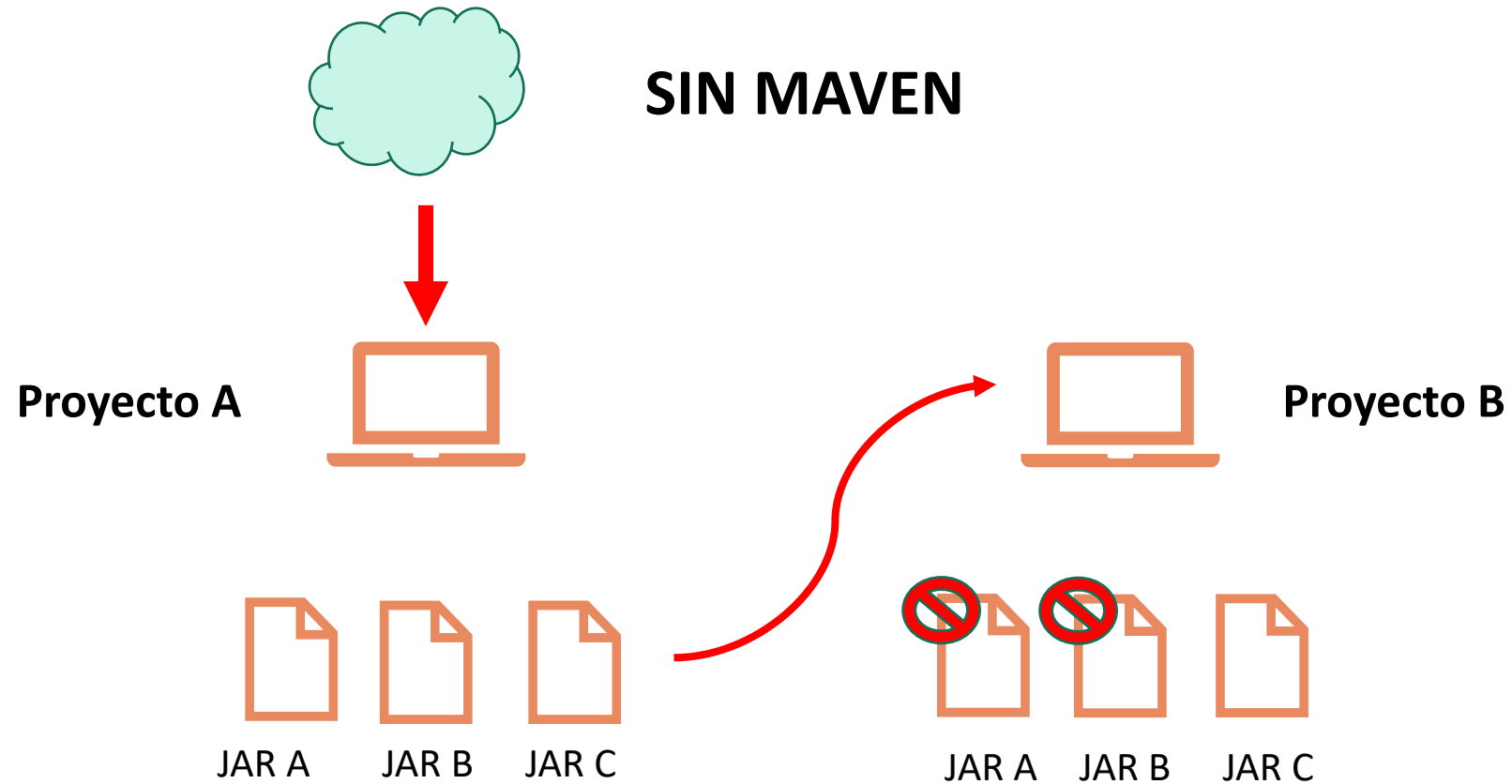
RC-1.0.0

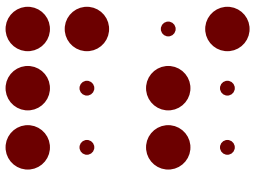
BETA-2.0.1

SNAPSHOT-1.0.1

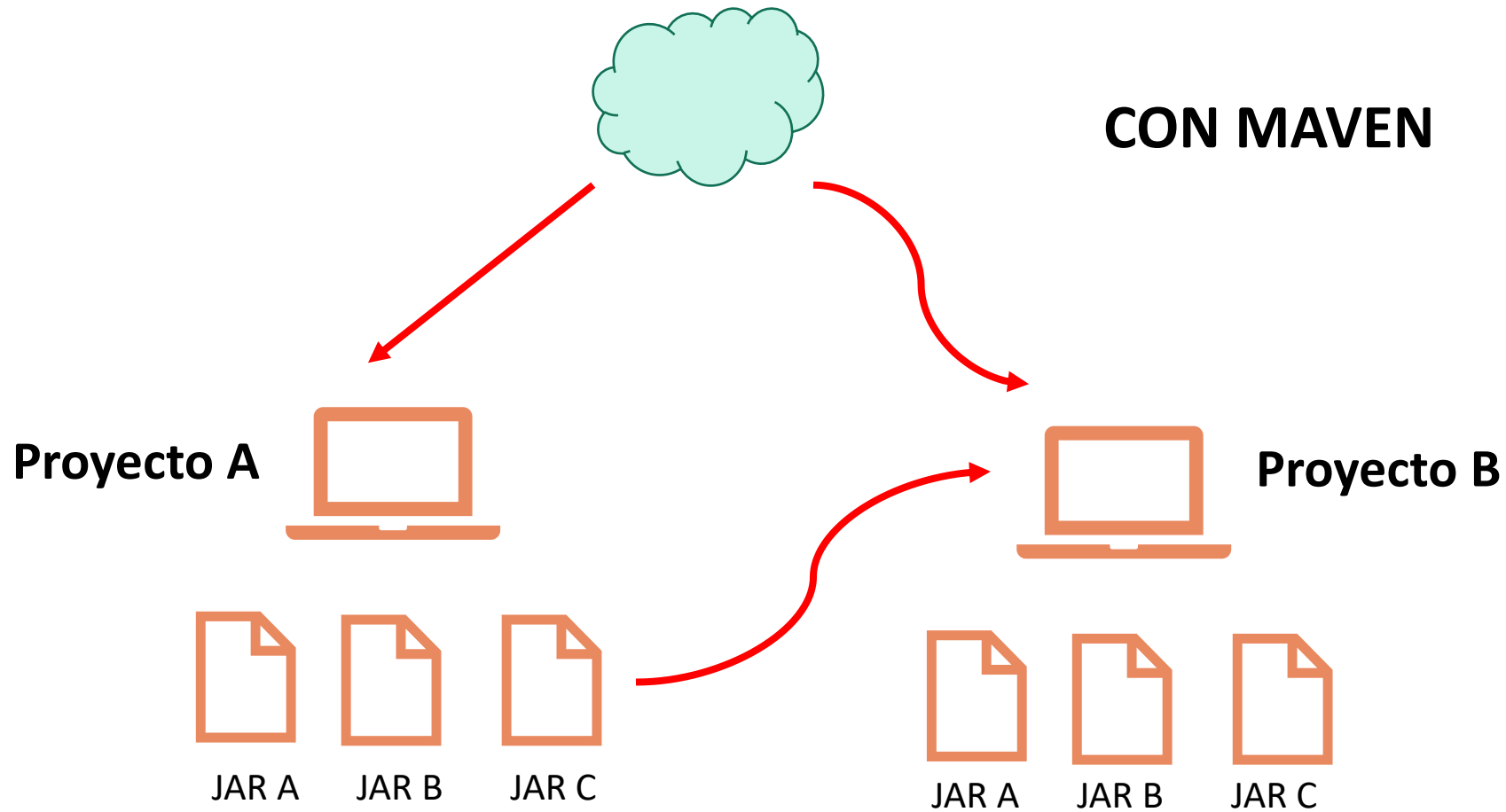


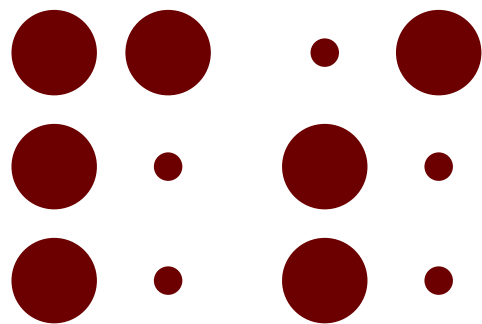
Introducción a Maven





Introducción a Maven

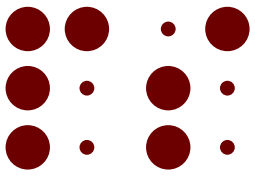




Módulo 3

Aplicaciones Web



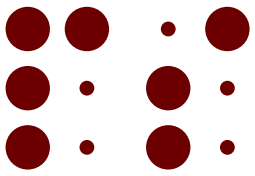


Introducción

Aplicaciones web

Aplicaciones WEB (Web app)

- Es un grupo de recursos cuyo acceso es a través de un cliente y el cual se hará un deployment.
- Contiene una estructura definida de proyecto.
- Es ejecutada sobre un Contenedor Web o Servidor de Aplicaciones.
- Para tener acceso a los recursos se utiliza el protocolo HTTP.

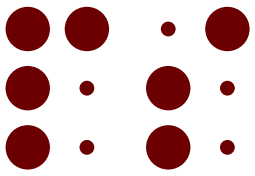


Archivo descriptor

El archivo descriptor de una aplicación web Jakarta EE es básicamente el archivo de configuración, que proporciona métodos y opciones para establecer parámetros de la misma aplicación.

- Definen el ambiente de ejecución de una Web App.
- Mapean URLs a Servlets y JSPs.
- Se definen páginas de bienvenida y errores.
- Se pueden definir estrategias de seguridad





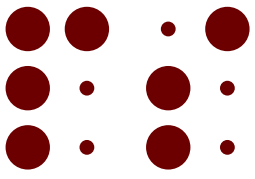
Archivo descriptor

Aplicaciones web

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>Aplicacion Web</display-name>
  <servlet>
    <servlet-name>test</servlet-name>
    <servlet-class>
      com.mycompany.test.Servlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>test</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```



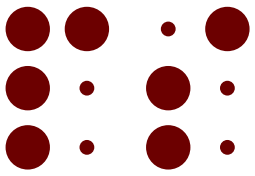


WAR

Aplicaciones web

Web Application archives (.war files).

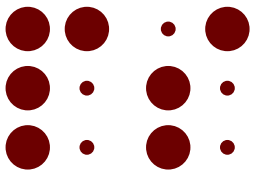
Un WAR es un archivo empaquetado que contiene la estructura de directorios de una Web Application. Son creados utilizando la utilidad jar y simplifican la distribución de Web Applications.



Deployment

Las aplicaciones creadas bajo el estándar dde JEE, son publicadas a un contenedor a través de los siguientes tipos de archivos de salida:

- Java Application Resource (JAR)
- Web Application Resource (WAR)
- Enterprise Application Resource (EAR)

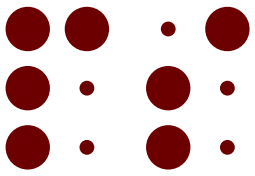


Servlets

Aplicaciones web

- Los Servlets son clases Java que proveen un protocolo para petición/respuesta.
- Los Servlets pueden ser accedidos a través de comandos HTTP.





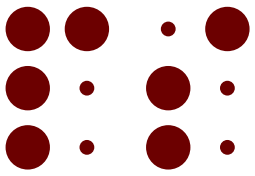
Servlets

- Extiende de HttpServlet.
- El método service() es invocado por el contenedor en cada petición al Servlet.

Aplicaciones web

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.io.*;

public class MyServlet extends HttpServlet {
    public void service(
        HttpServletRequest req,
        HttpServletResponse resp)
        throws ServletException, IOException {
        // Procesamiento
    }
}
```

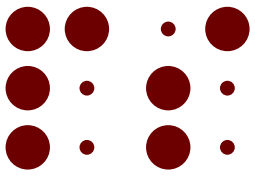


Servlets

El método `service()` es utilizado para procesar las invocaciones a los métodos `doGet()`, `doPost()`.

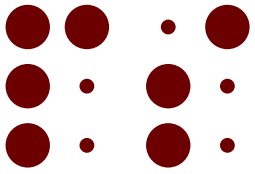
Aplicaciones web

```
public class MyServlet extends HttpServlet {  
    public void doGet(  
        HttpServletRequest req,  
        HttpServletResponse resp) {  
        // Procesamiento  
    }  
  
    public void doPost(  
        HttpServletRequest req,  
        HttpServletResponse resp) {  
        // Procesamiento  
    }  
}
```

Procesamiento del formulario

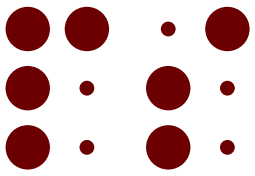
- Cuando un formulario es enviado (submitted), los campos y sus valores son enviados con la petición.
- Con el método GET, los envíos de los campos y sus valores van pegados a la URL.
- Con el método POST, los envíos de los campos y sus valores viajan en el cuerpo de la petición.



Configuración de un Servlet

Las clases de los Servlets son:

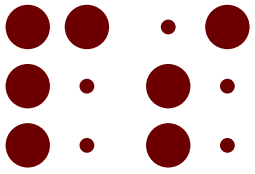
- Colocados en el directorio WEB-INF/classes de la Web App.
- Se debe registrar en el archivo descriptor de la Web App.



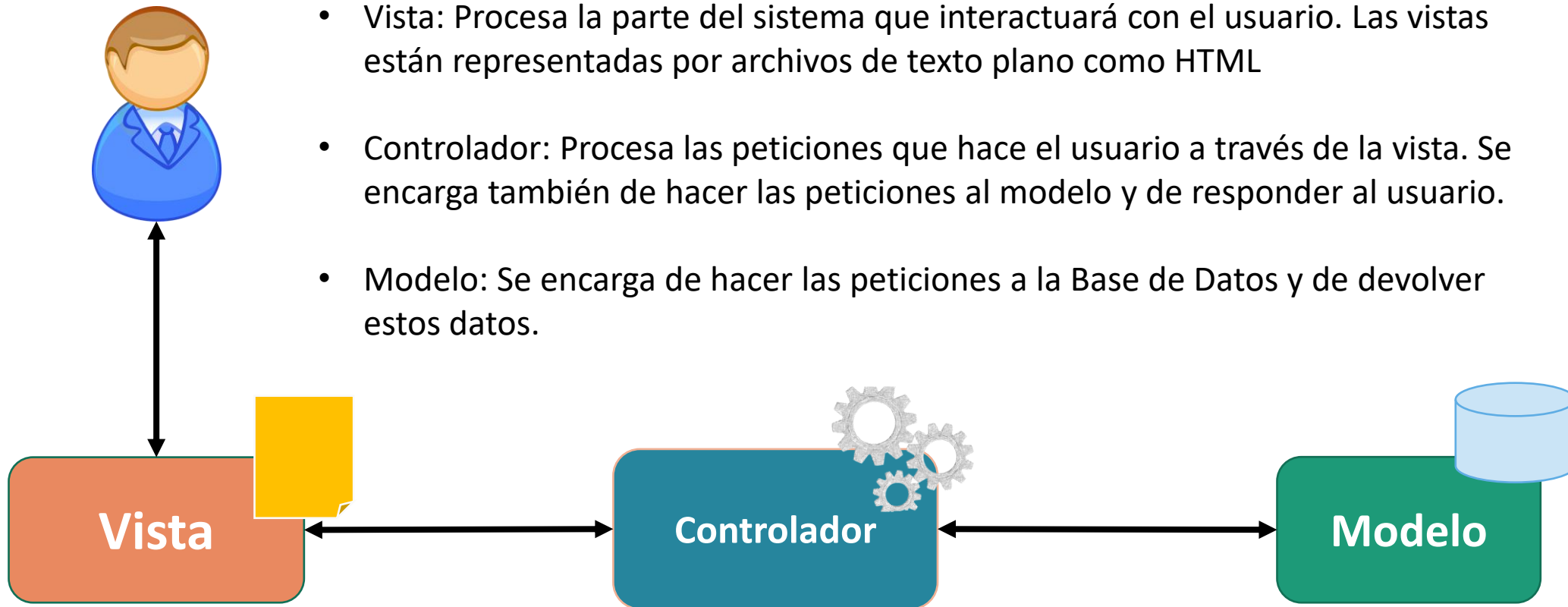
Configuración de un Servlet

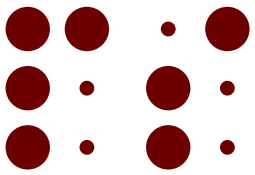
Configuración de un Servlet en web.xml

```
<web-app>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>servlettest</url-pattern>
  </servlet-mapping>
</web-app>
```



Arquitectura Modelo – Vista – Controlador





Aplicaciones web

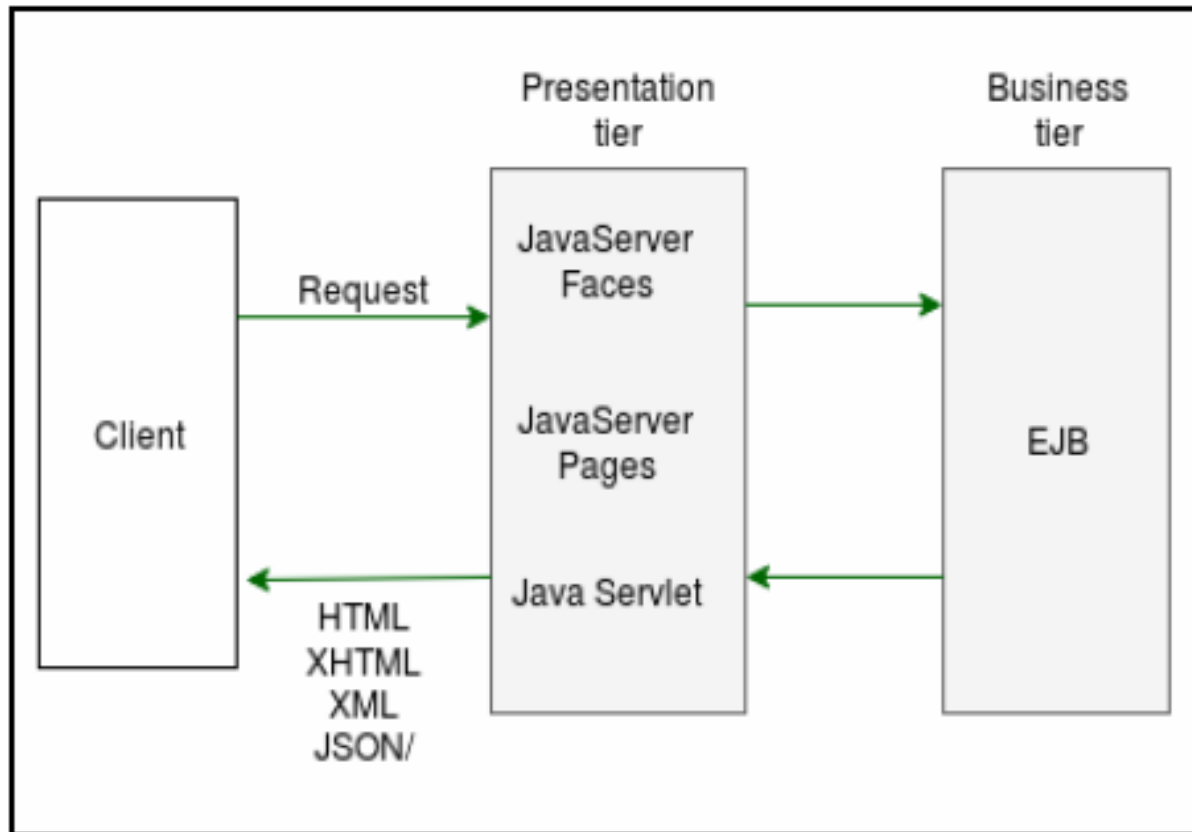
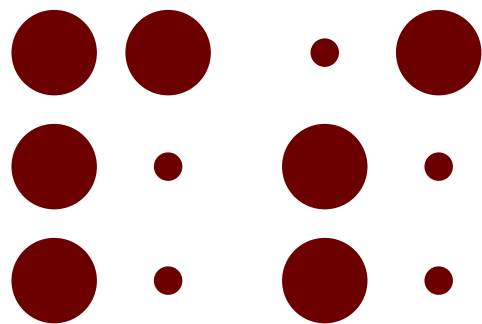


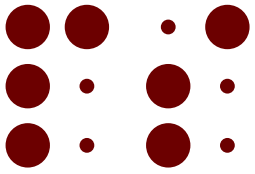
Diagrama general de una aplicación JEE distribuida



Módulo 4

Servicios Web

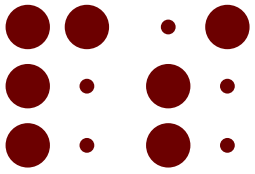




Servicios Web

La plataforma Java EE es un modelo de aplicación distribuida de varios niveles que tiene tres niveles comunes ampliamente utilizados. Estos niveles son el nivel de presentación (o nivel web), el nivel comercial o de negocios y el nivel de integración (o nivel EIS).





XML

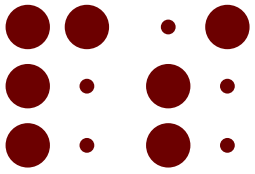
Es lenguaje de marcado extensible (XML) es un formato de texto simple y muy flexible para el intercambio de información estructurada entre diferentes plataformas.

<https://www.w3.org/XML/>

JAX-WS

Java API for XML Web Services (JAX-WS) es una API Java utilizada para desarrollar servicios web SOAP.

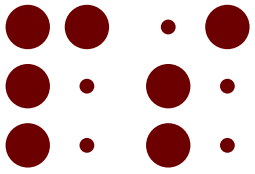
<https://javaee.github.io/metro-jax-ws/>



Apache eXtensible Interaction System (AXIS)

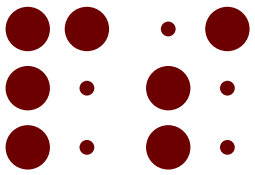
Es un marco de servicio web basado en XML y de código abierto. Consiste en una implementación Java y C++ del servidor SOAP, y varias utilidades y API para generar y desplegar aplicaciones de servicios web.

<https://axis.apache.org/axis/>



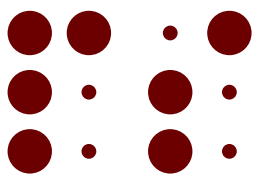
WebServices

¿Qué es un servicio web? No es más que una invocación de método remoto (RMI) realizada a través de HTTP utilizando un mecanismo de transporte basado en texto (un documento XML en este caso).

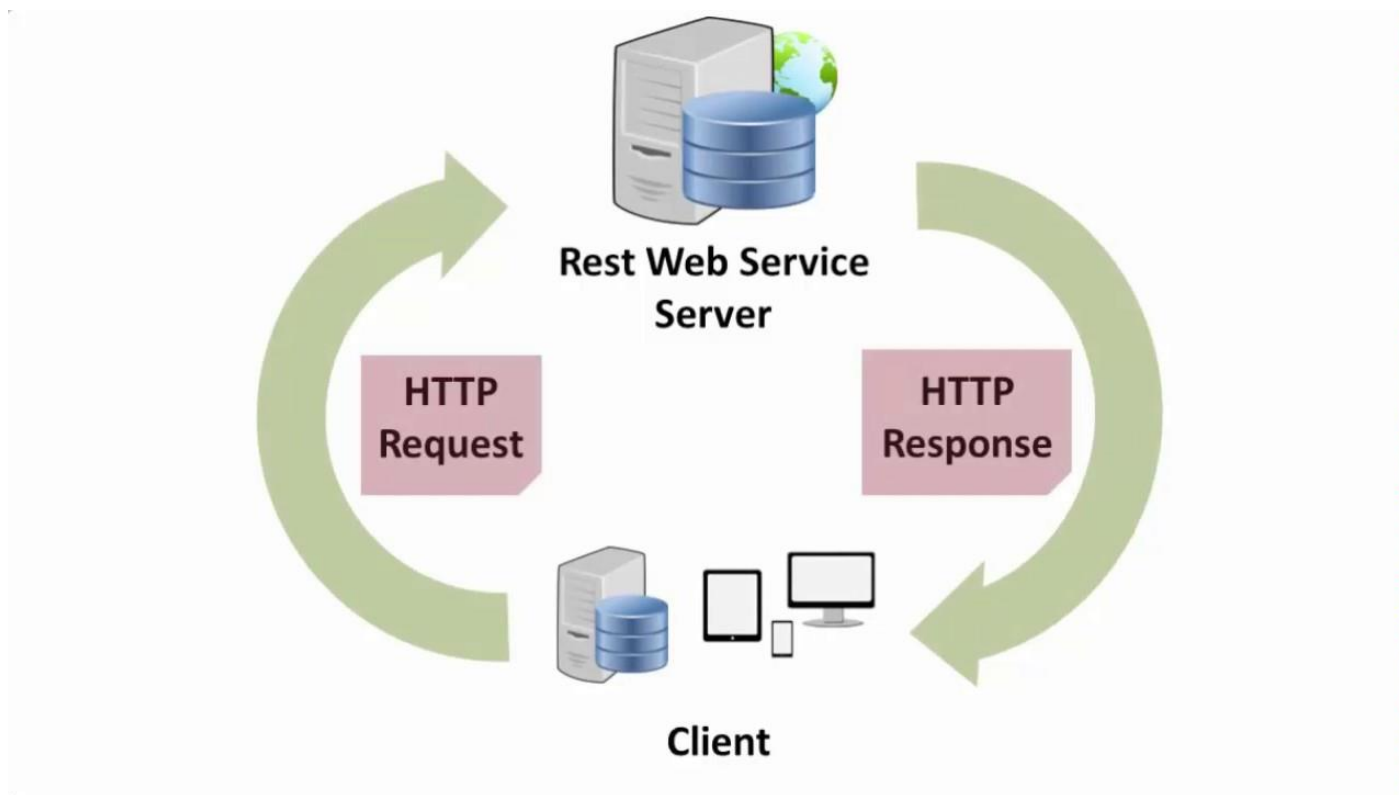


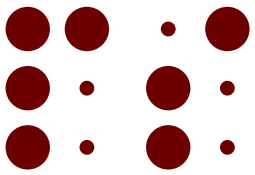
Java RMI

RMI (Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.

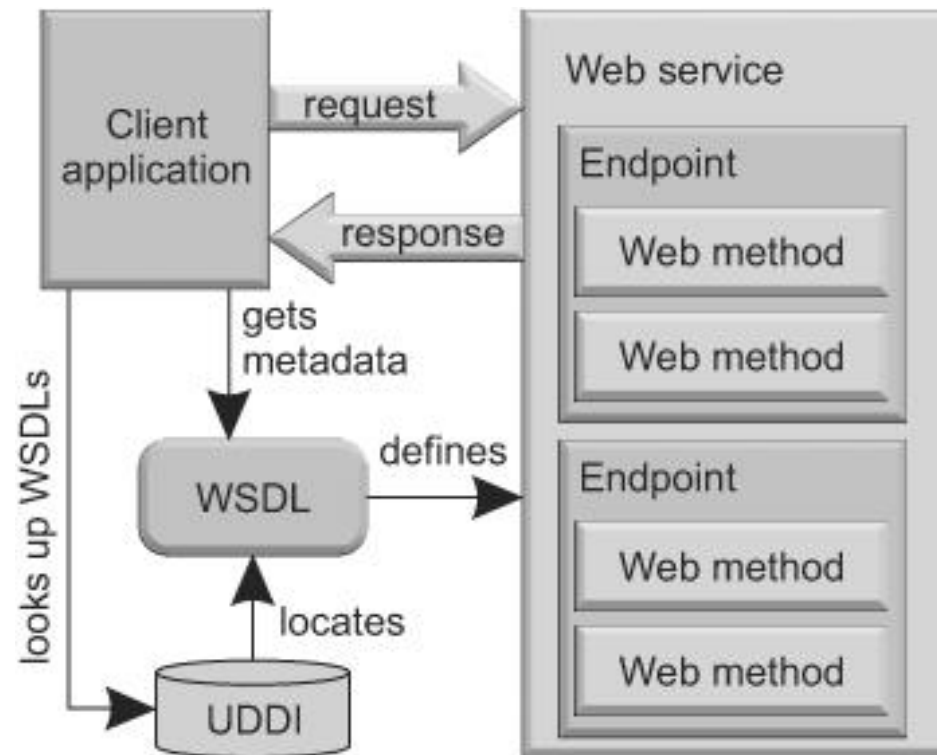


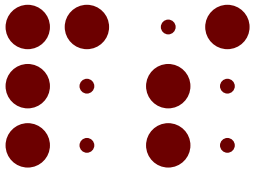
WebServices





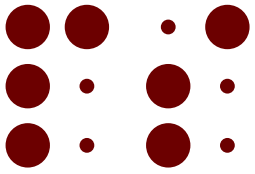
Terminología





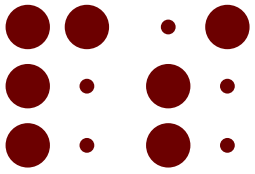
Endpoint

Un servicio web es una colección de endpoints. Cada endpoint se implementa en Java como una clase. Un endpoint puede contener uno o más métodos web.



Web Service Description Language (WSDL)

Es un documento XML que describe el servicio web. Aunque puede crear un WSDL desde cero para definir su servicio web y luego generar los stubs necesarios a partir de él, generalmente es más fácil definir el servicio web en términos del punto final escrito en Java y generar el WSDL a partir de eso.

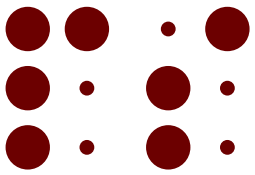


Universal Description, Discovery, and Integration (UDDI)

La especificación UDDI (Universal Description, Discovery, and Integration) define un modo de publicar y encontrar información sobre servicios Web.

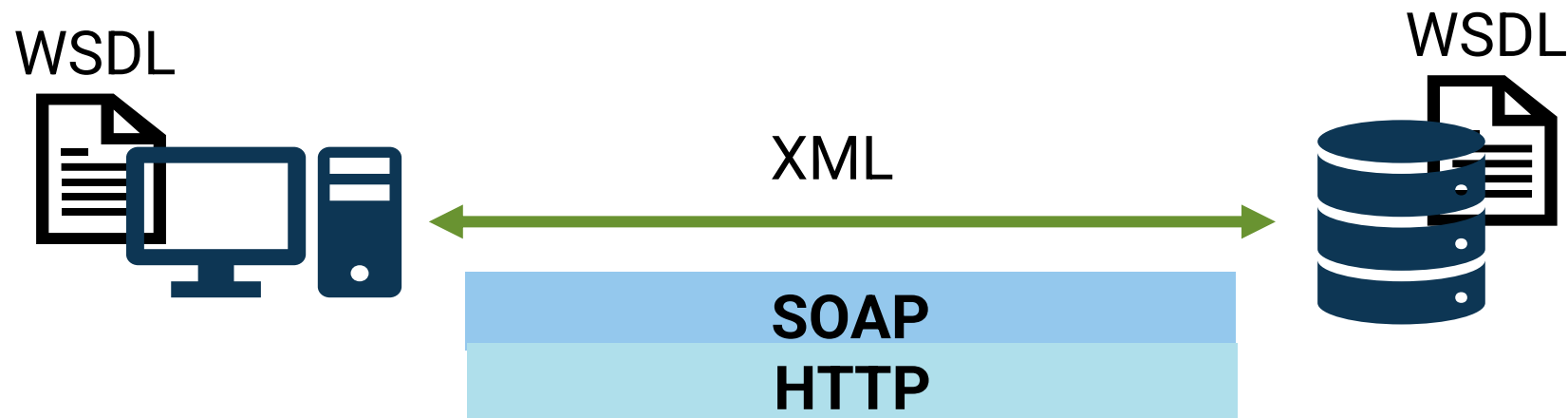
UDDI tiene dos funciones:

1. Es un protocolo basado en SOAP que define cómo se comunican los clientes con los registros UDDI.
2. Es un conjunto de registros duplicados globales en particular.



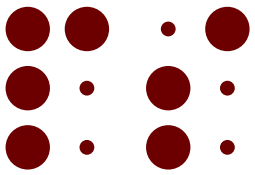
Simple Object Access Protocol (SOAP)

SOAP es un protocolo que permite el intercambio de datos entre sistemas heterogéneos.



SOAP proporciona dos estilos de enlaces:

1. Remote Procedure Call (RPC)
2. Documento



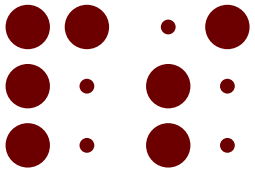
Remote Procedure Call (RPC)

Los clientes suelen pasar parámetros a un método web utilizando tipos de datos simples como String, Integer etc.

Parámetros

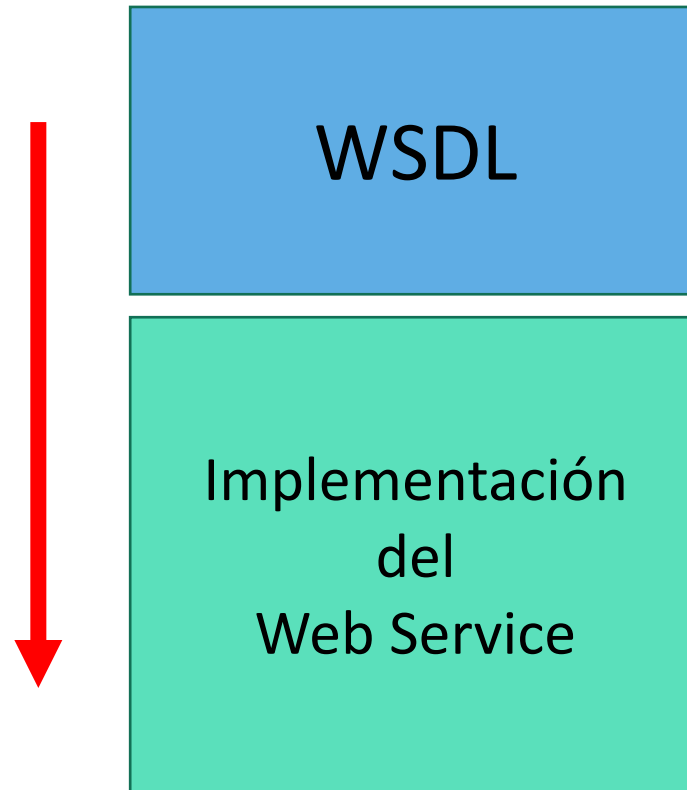


```
public String saludarse(String x, Integer y) {  
    return "Hola" + x;  
}
```

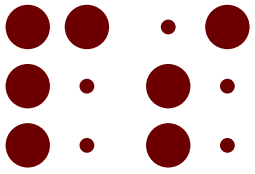


Tipos de desarrollo de WS

Desarrollo
Top – Down



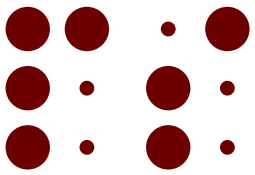
Desarrollo
Down - Top



¿Qué es una API REST?

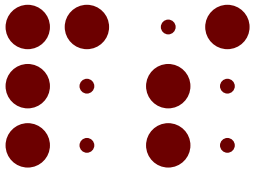
API
Application Programming Interface
+
REST
Representational State Transfer

Interfaz de aplicaciones para transferir
datos

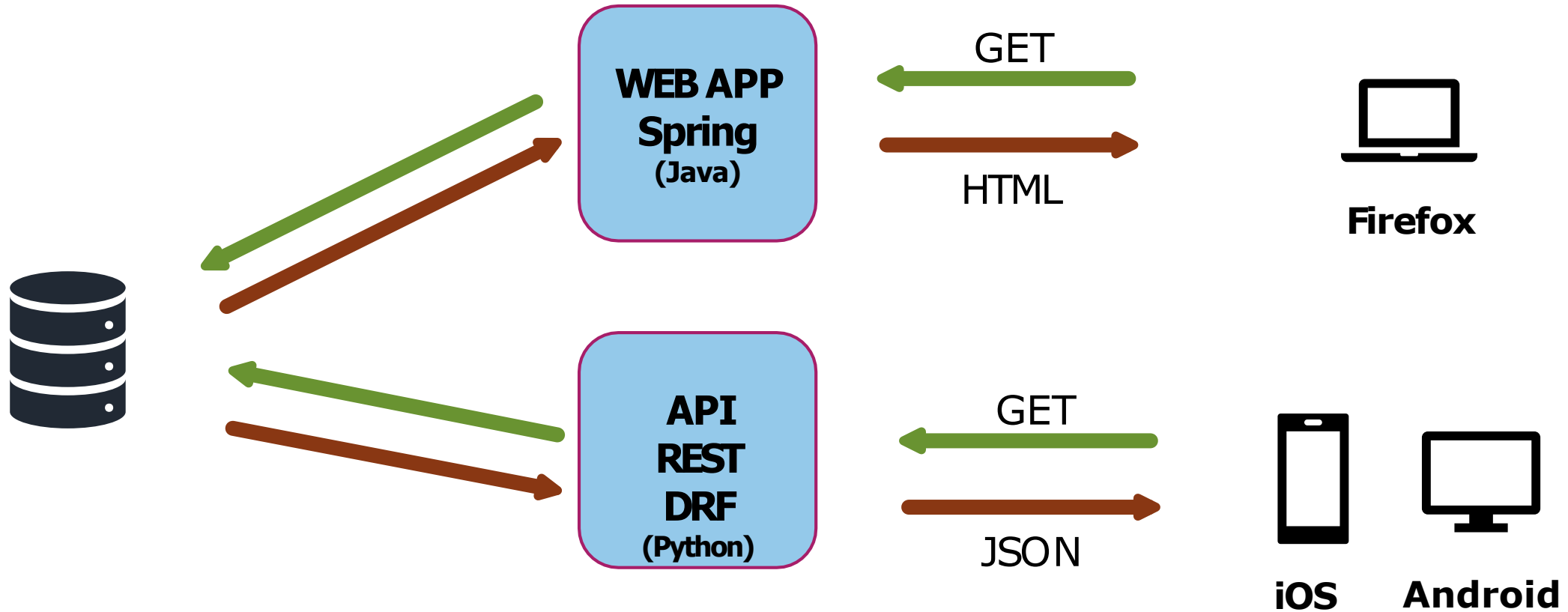


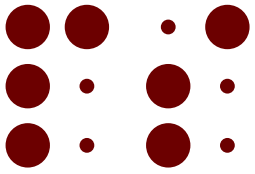
¿Qué es una API REST?





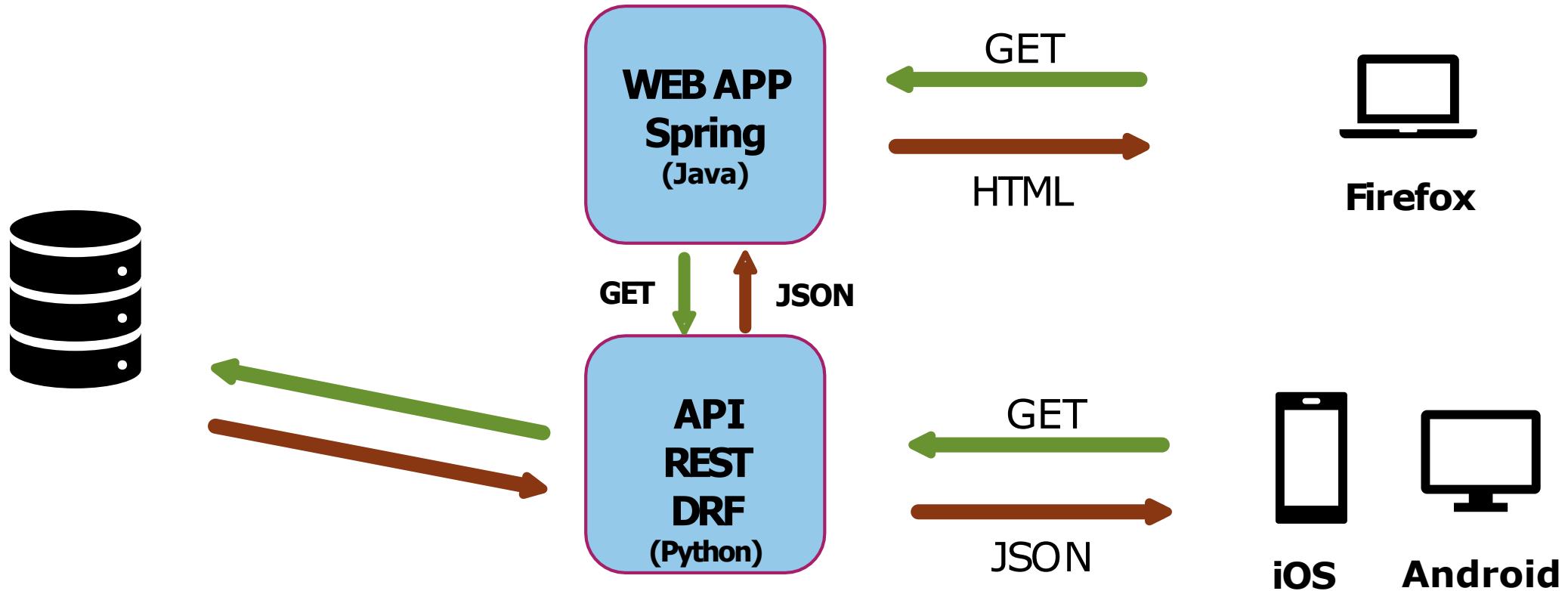
¿Qué es una API REST?

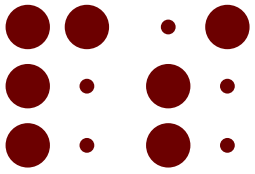




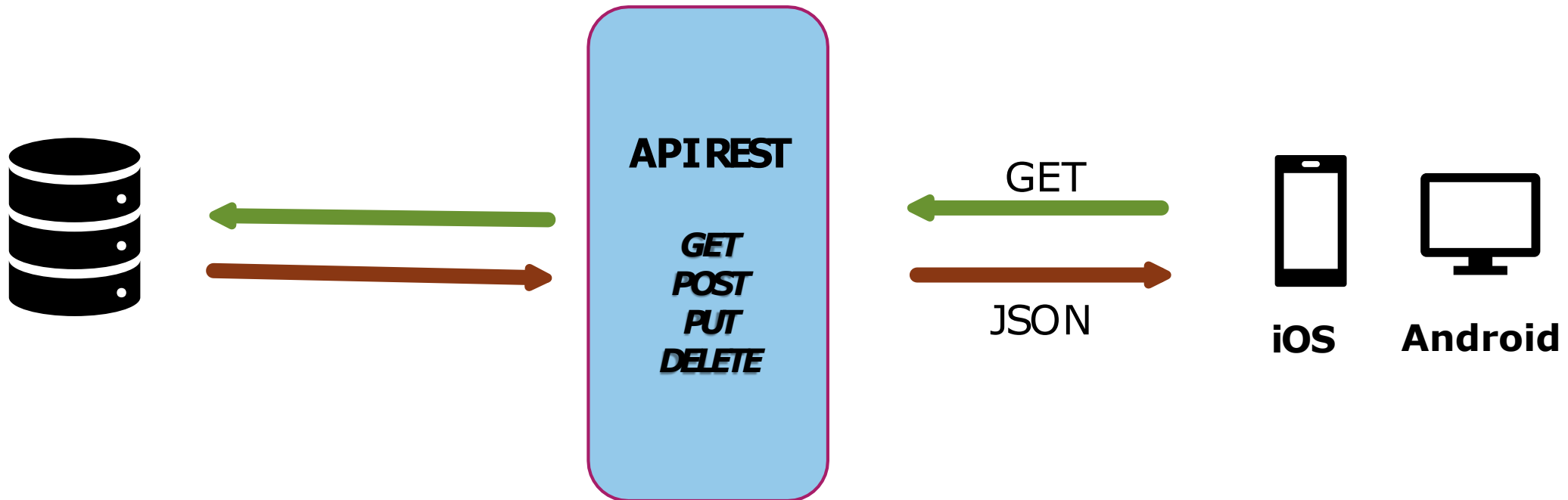
Servicios Web

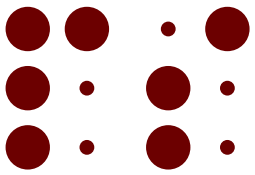
¿Qué es una API REST?





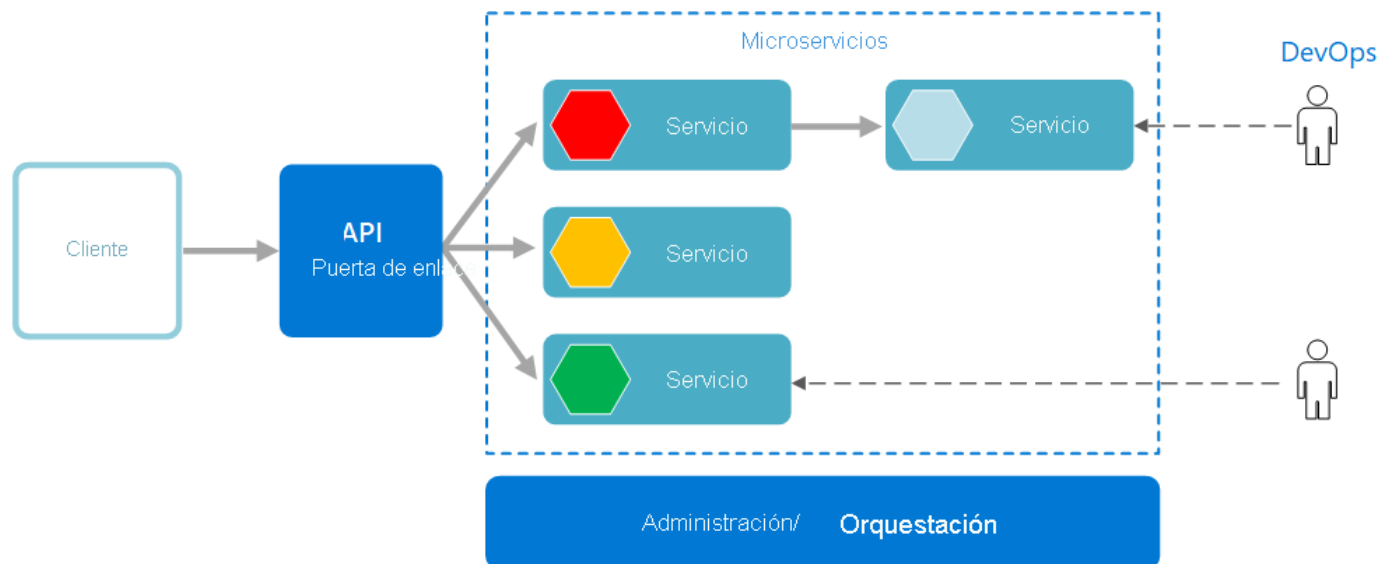
¿Qué es una API REST?

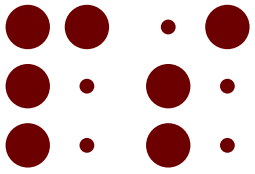




Microservicios

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de servicio es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado.

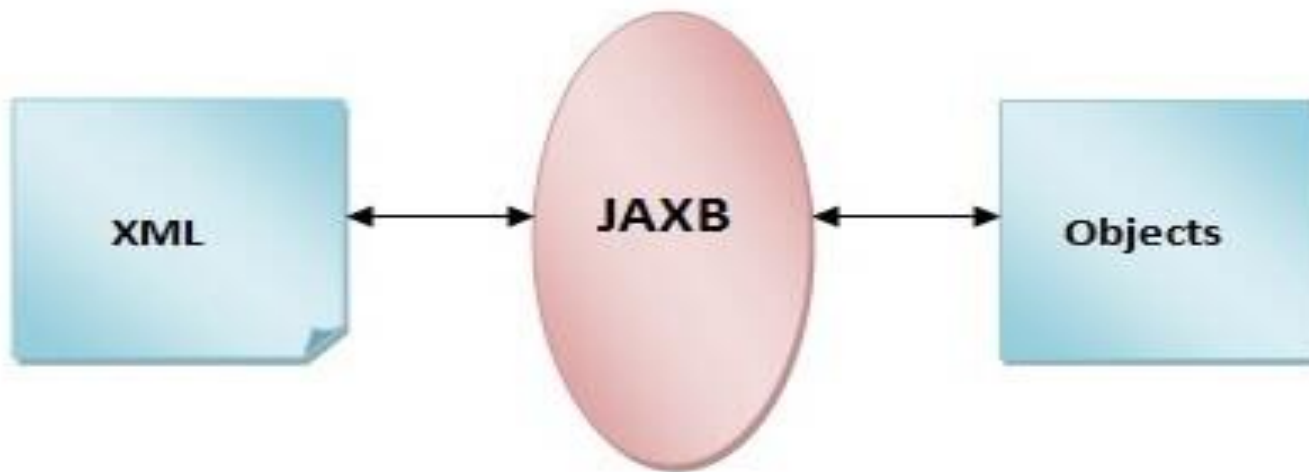


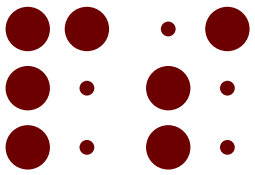


Java Architecture for XML Binding (JAXB)

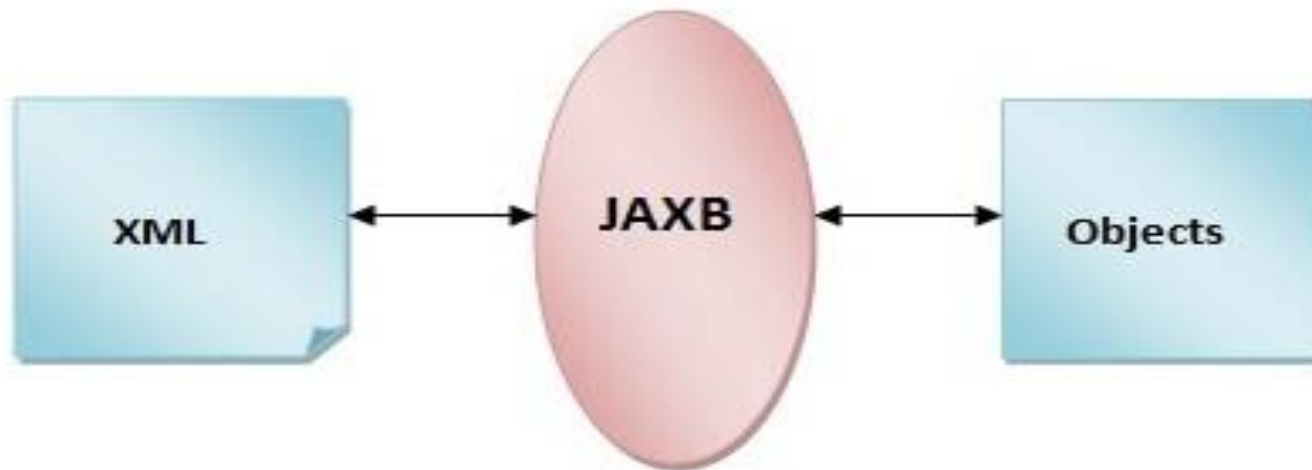
Es una API y herramientas que automatiza el mapeo entre documentos XML y objetos Java

<https://javaee.github.io/jaxb-v2/>



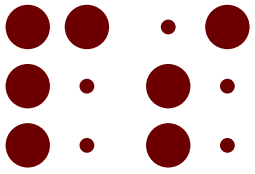


Java Architecture for XML Binding (JAXB)



Se pueden ejecutar dos operaciones generales con JAXB:

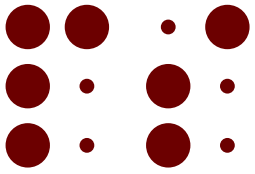
1. Marshalling: Convertir un objeto Java a XML
2. Unmarshalling: Convertir un XML a un objeto Java



Java Architecture for XML Binding (JAXB)

JAXB usa anotaciones de Java para configurar las clases generadas con información adicional.

- **@XmlElement**: Es el nombre del elemento raíz XML y se deriva del nombre de la clase.
- **@XmlType**: Define el orden en que se escriben los campos en el archivo XML.
- **@XmlElement**: Define el nombre del elemento XML real que se utilizará.
- **@XmlAttribute**: Define que el campo de identificación que se asigna como un atributo en lugar de un elemento.
- **@XmlTransient**: Anotar aquellos campos que no incluyen en el XML.

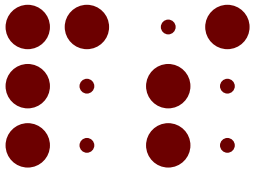


Ejercicio 1

1. Dado los siguientes atributos generar la clase Java y operaciones Marshall y Unmarshall

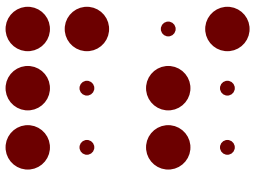
Atributos:

- Id
- Nombre
- Apellido
- Fecha de nacimiento



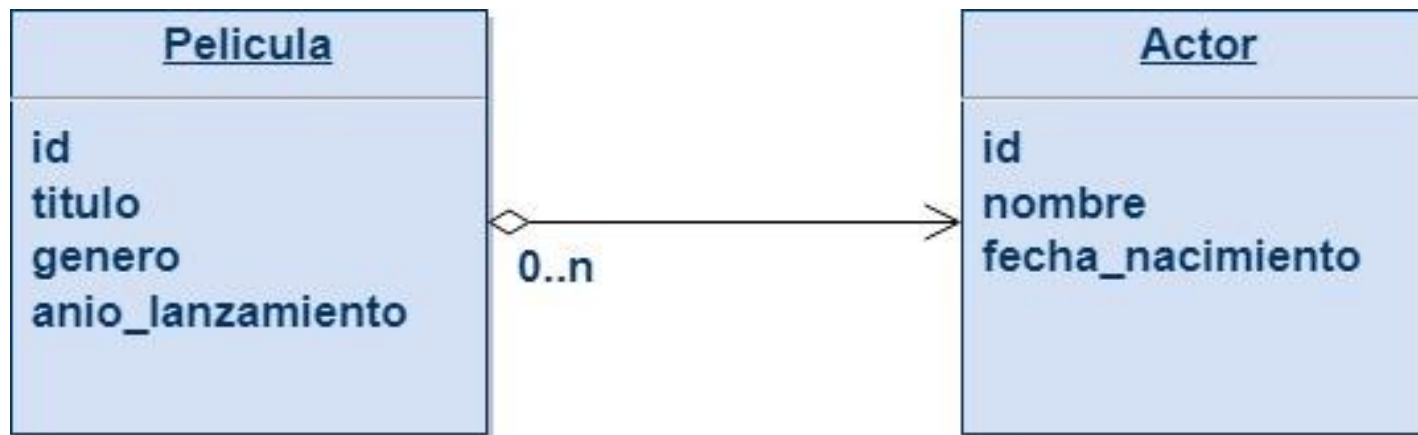
Ejercicio 2

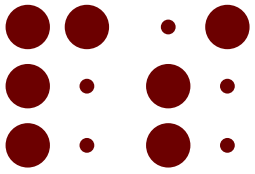
1. Modificar el ejercicio anterior para que el atributo fecha de nacimiento muestre solo la fecha en formato DD-MM-YYYY



Ejercicio 3

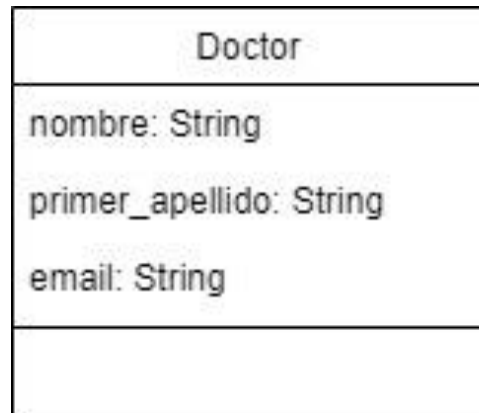
1. A partir del siguiente modelo, obtener el archivo XML.

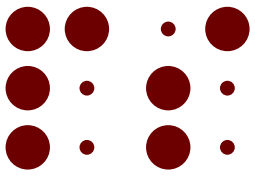




Ejercicio 4

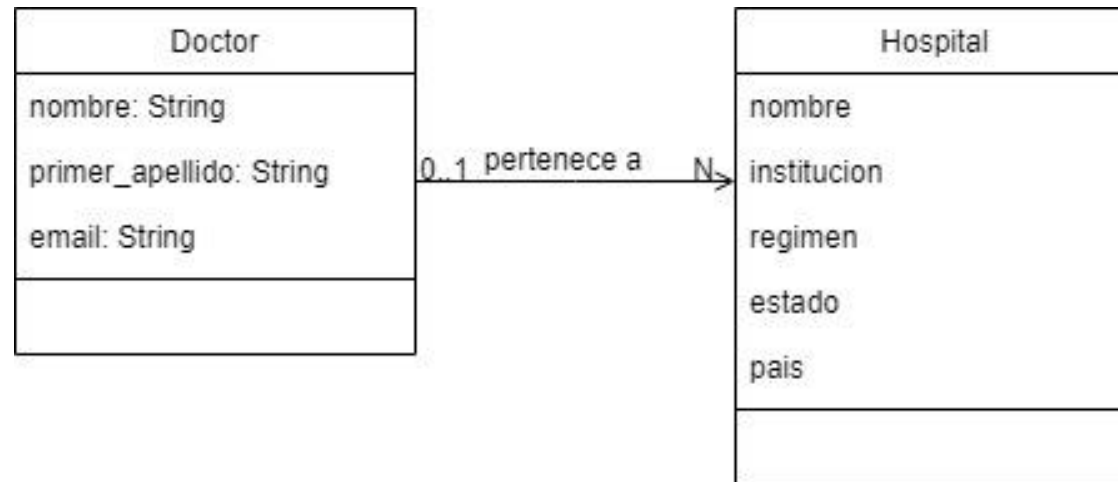
1. Publicar un WS que devuelva una cadena de texto.
2. Publicar un WS que implemente el siguiente diagrama de clases y devuelva un objeto.

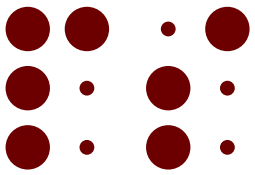




Ejercicio 5

1. Publicar un WS que implemente el siguiente diagrama de clases y devuelva una lista de objetos.

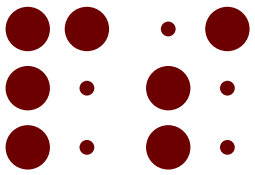




Ejercicio 6

1. Publicar un WS que:

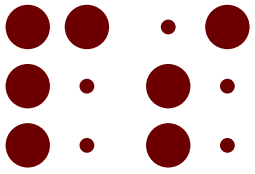
- a) Reciba una cadena en formato JSON con los atributos de la clase Doctor
- b) Que transforme la cadena JSON en un objeto Doctor
- c) Devuelva un objeto Doctor



Ejercicio 7

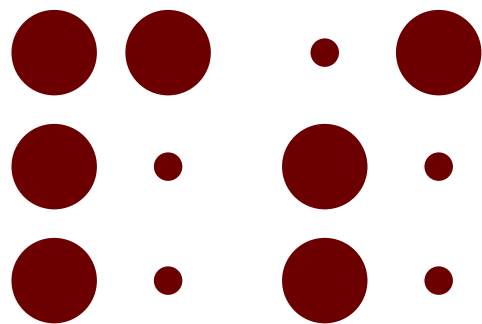
1. Publicar un WS que:

- a) Reciba una cadena en formato XML con los atributos de la clase Doctor
- b) Que transforme la cadena XML en un objeto Doctor
- c) Devuelva un objeto Doctor



Ejercicio 9

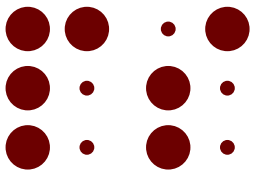
1. Publicar un WS que pueda recibir devolver diferentes valores en variables separadas.
2. Publicar un WS que reciba un archivo como parámetro.



Módulo 5

Java JDBC

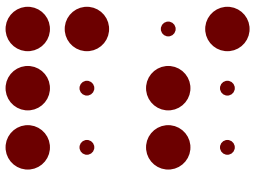




Java Data Base Connectivity

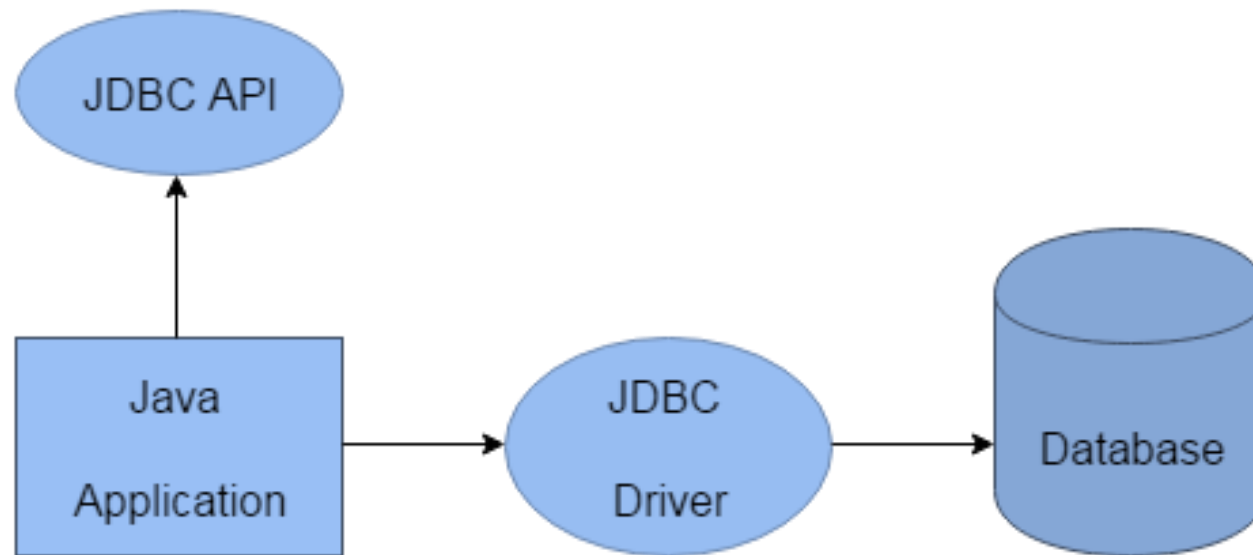
La API de Java Database Connectivity (JDBC) proporciona acceso universal a datos desde el lenguaje de programación Java. Con la API de JDBC, puede acceder a prácticamente cualquier fuente de datos, desde bases de datos relacionales hasta hojas de cálculo y archivos sin formato.

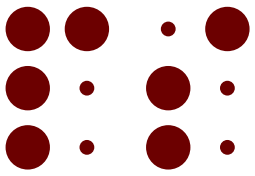




Java Data Base Connectivity

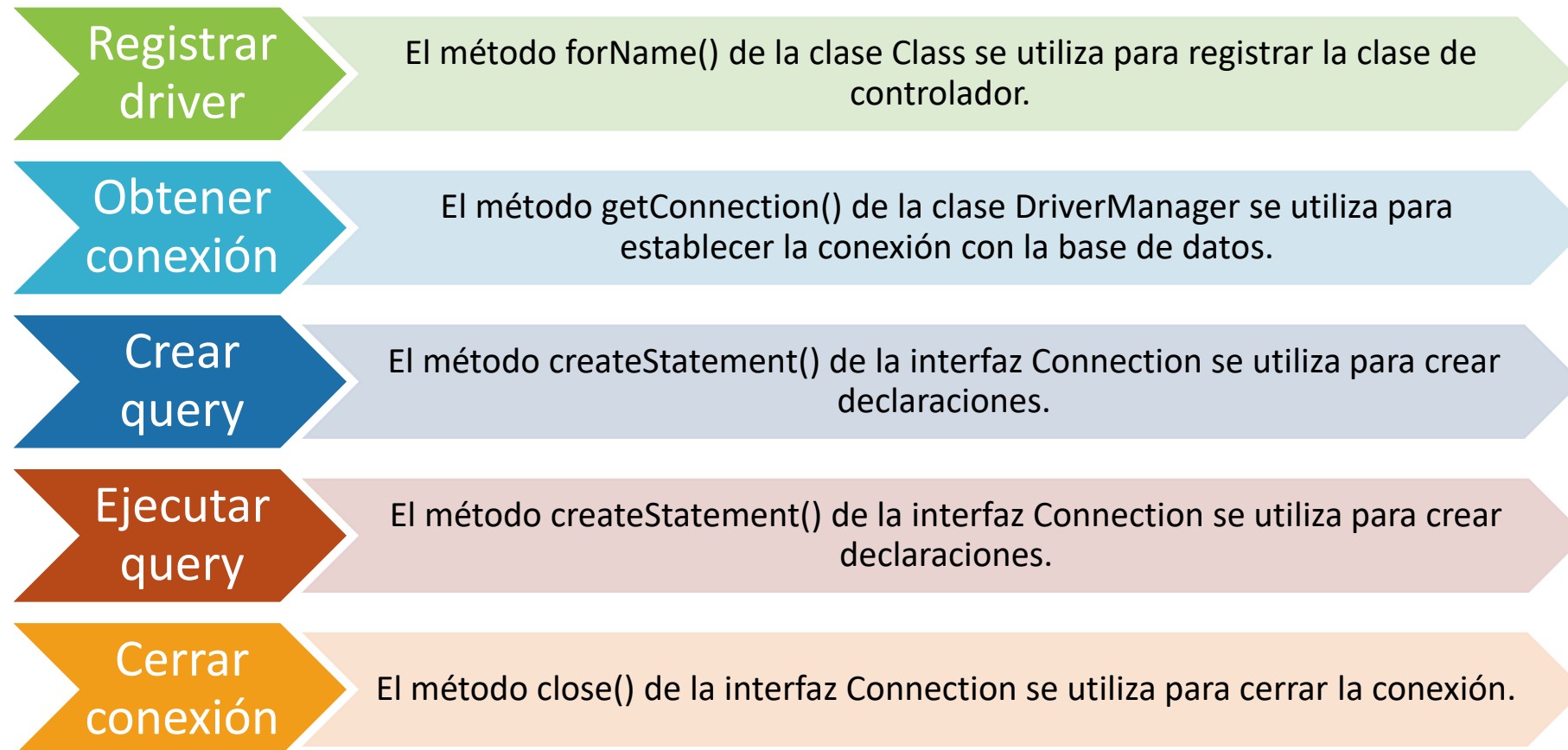
Flujo general de un aplicación JDBC

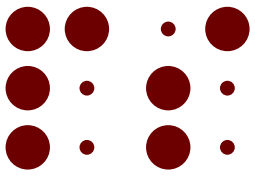




Java Data Base Connectivity

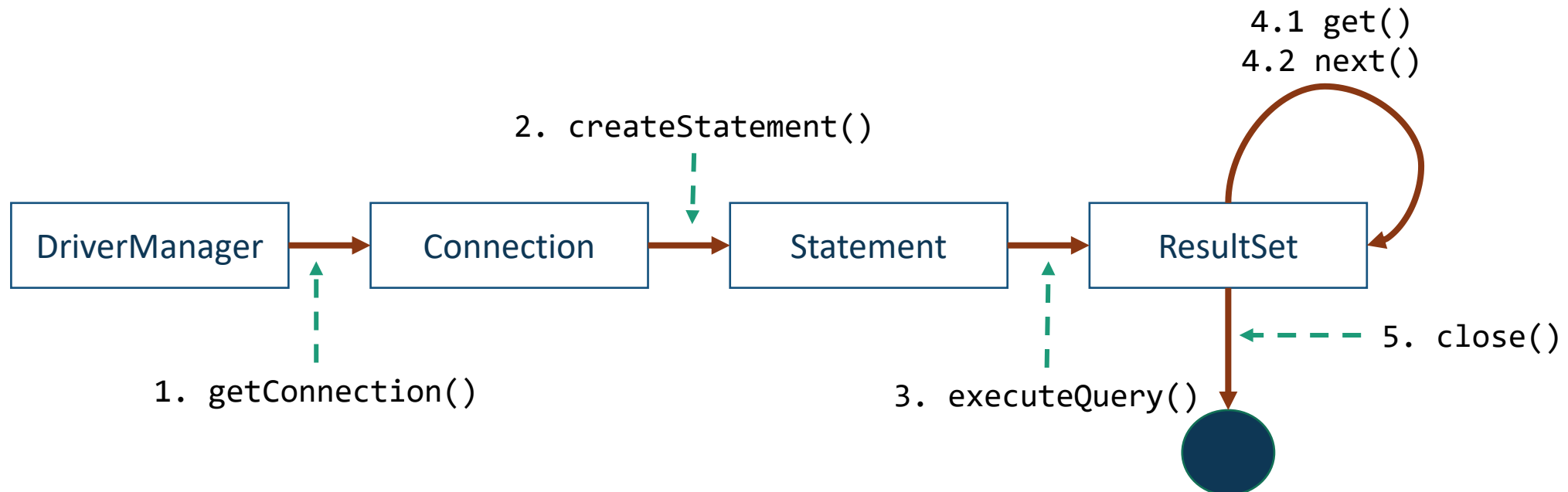
Flujo general de una consulta JDBC

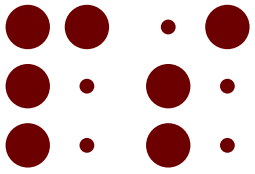




Java Data Base Connectivity

Flujo general de una consulta JDBC





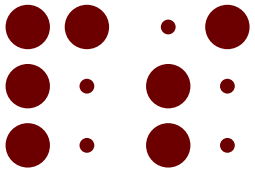
Java Data Base Connectivity

DriverManager

La clase DriverManager es el componente de la API de JDBC y también un miembro del paquete java.sql.

Connection

Una conexión es una sesión entre una aplicación Java y una base de datos. Ayuda a establecer una conexión con la base de datos.

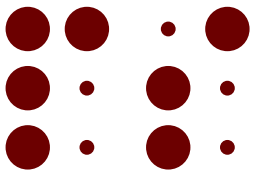


Java Data Base Connectivity

Statement

La interfaz de declaración proporciona métodos para ejecutar consultas con la base de datos.

- **ResultSet executeQuery(String sql):** Se utiliza para ejecutar la consulta SELECT. Devuelve el objeto de ResultSet.
- **int executeUpdate(String sql):** Se utiliza para ejecutar una consulta específica, puede ser crear, soltar, insertar, actualizar, eliminar, etc.
- **boolean execute(String sql):** Se utiliza para ejecutar consultas que pueden devolver múltiples resultados.

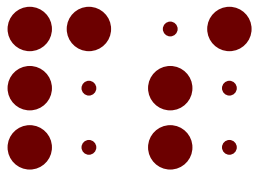


Java Data Base Connectivity

ResultSet

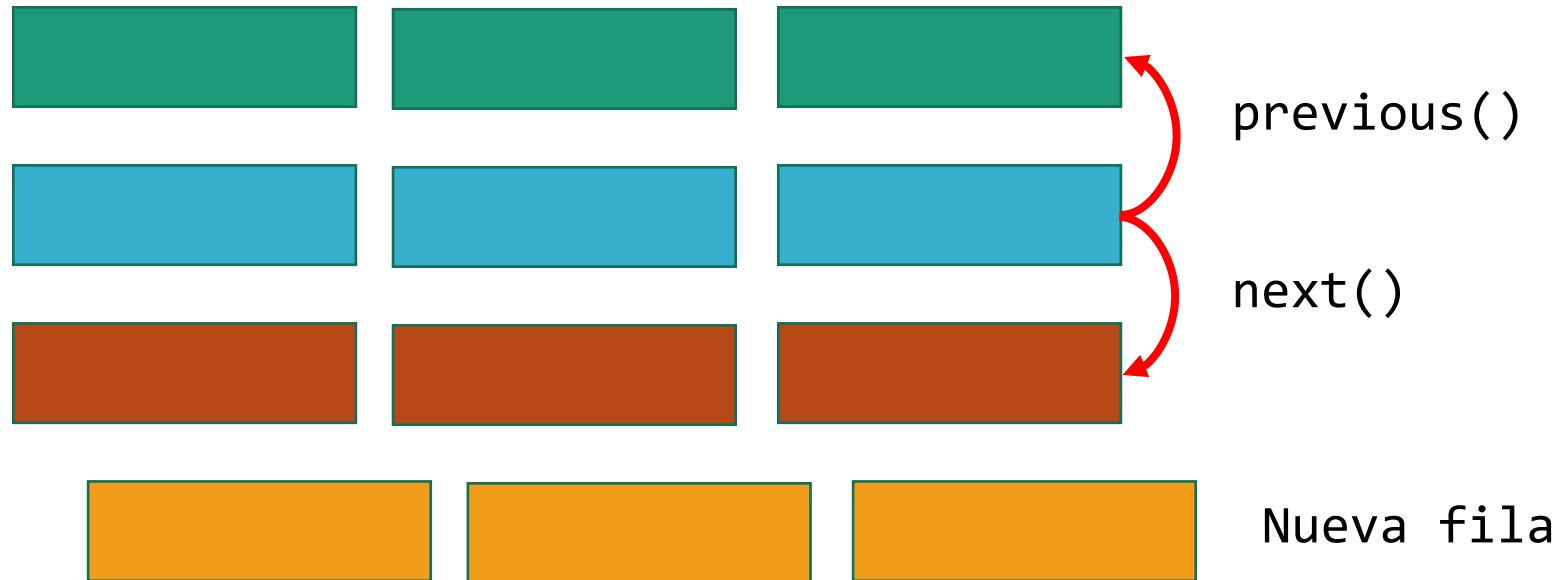
El objeto de ResultSet mantiene un cursor que apunta a una fila de una tabla. Inicialmente, el cursor apunta antes de la primera fila.

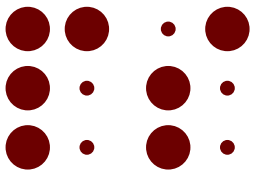
next()	Se utiliza para mover el cursor a la fila siguiente a la posición actual.
previous()	se utiliza para mover el cursor a la fila anterior a la posición actual.
first()	se utiliza para mover el cursor a la primera fila en el objeto de conjunto de resultados.
last()	se utiliza para mover el cursor a la última fila en el objeto de conjunto de resultados.
getInt(String/int)	se utiliza para devolver los datos de la fila actual como int.
getString(String/int)	se utiliza para devolver los datos de la fila actual como String.



Java Data Base Connectivity

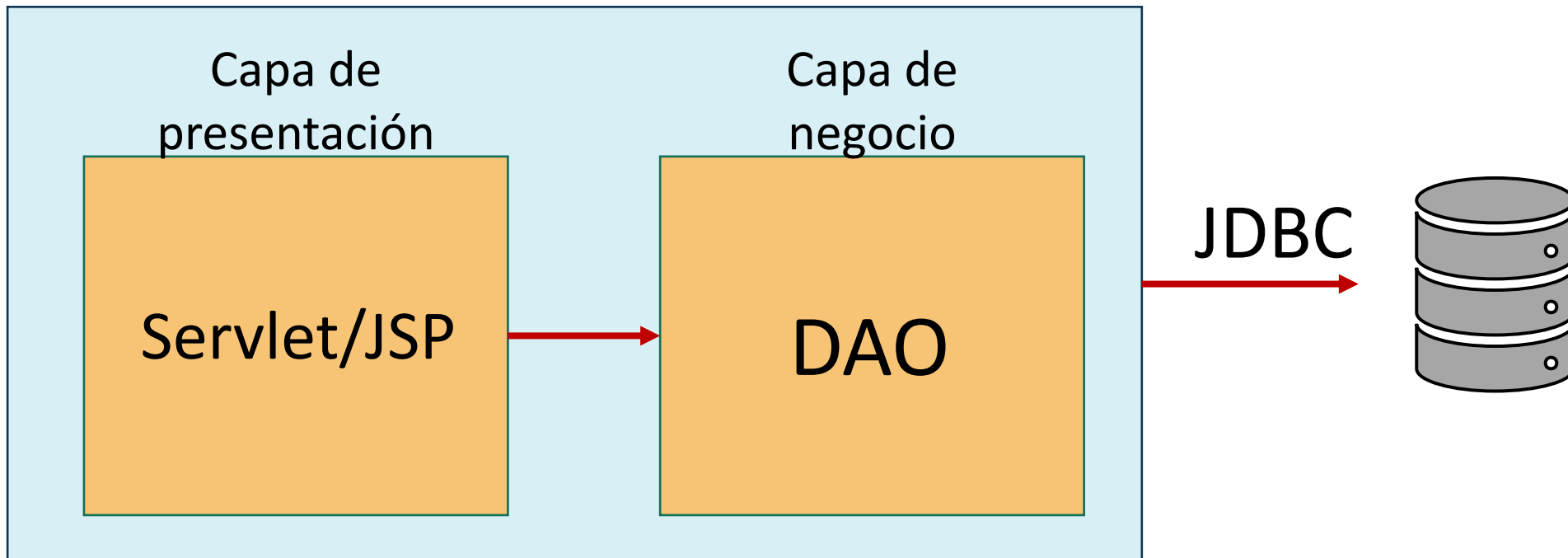
ResultSet

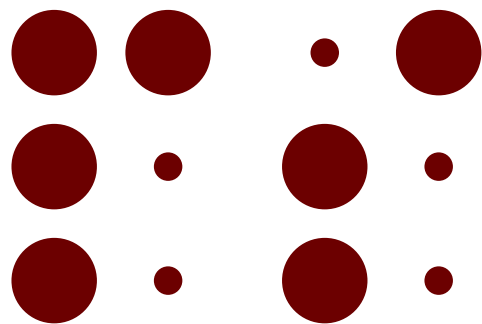




- Implementar las siguientes tecnologías en proyecto JEE.

GlassFish Server

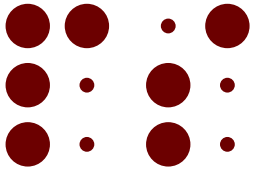




Módulo 6

Java EJB

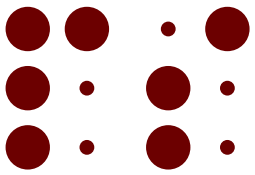




Enterprise JavaBeans

- Framework orientada al desarrollo de aplicaciones empresariales.
- Creación de componentes de negocio o servicios
- Reusabilidad de componentes

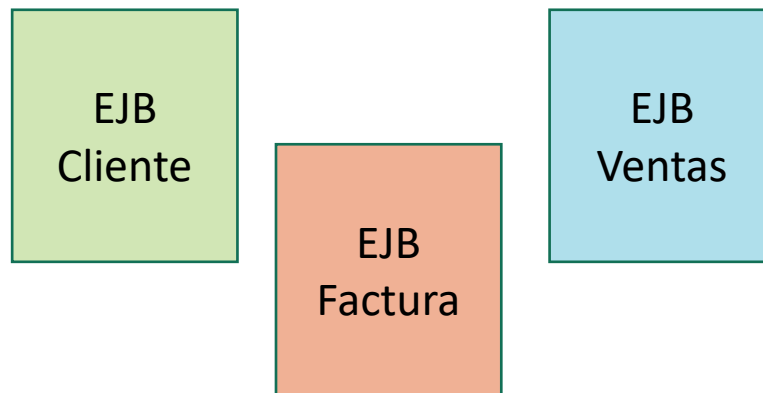


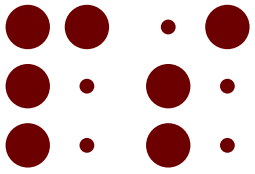


Enterprise JavaBeans

Components Model

- Session Beans -> Componente de negocio.
- Entities -> POJO's
- Message Beans

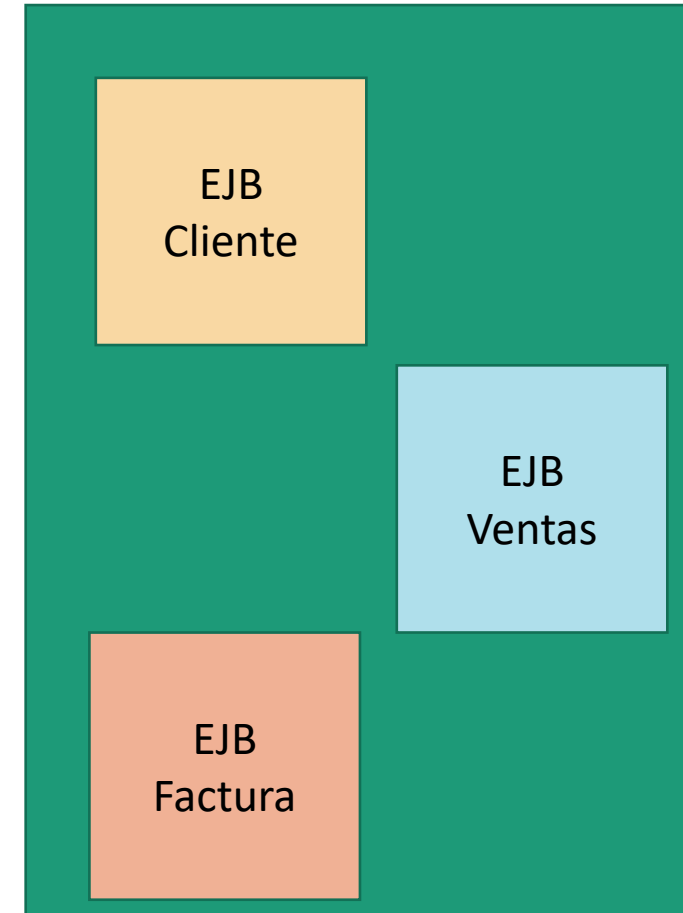


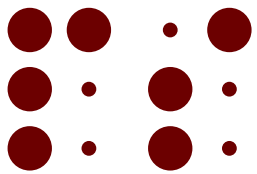


Enterprise JavaBeans

Container EJB

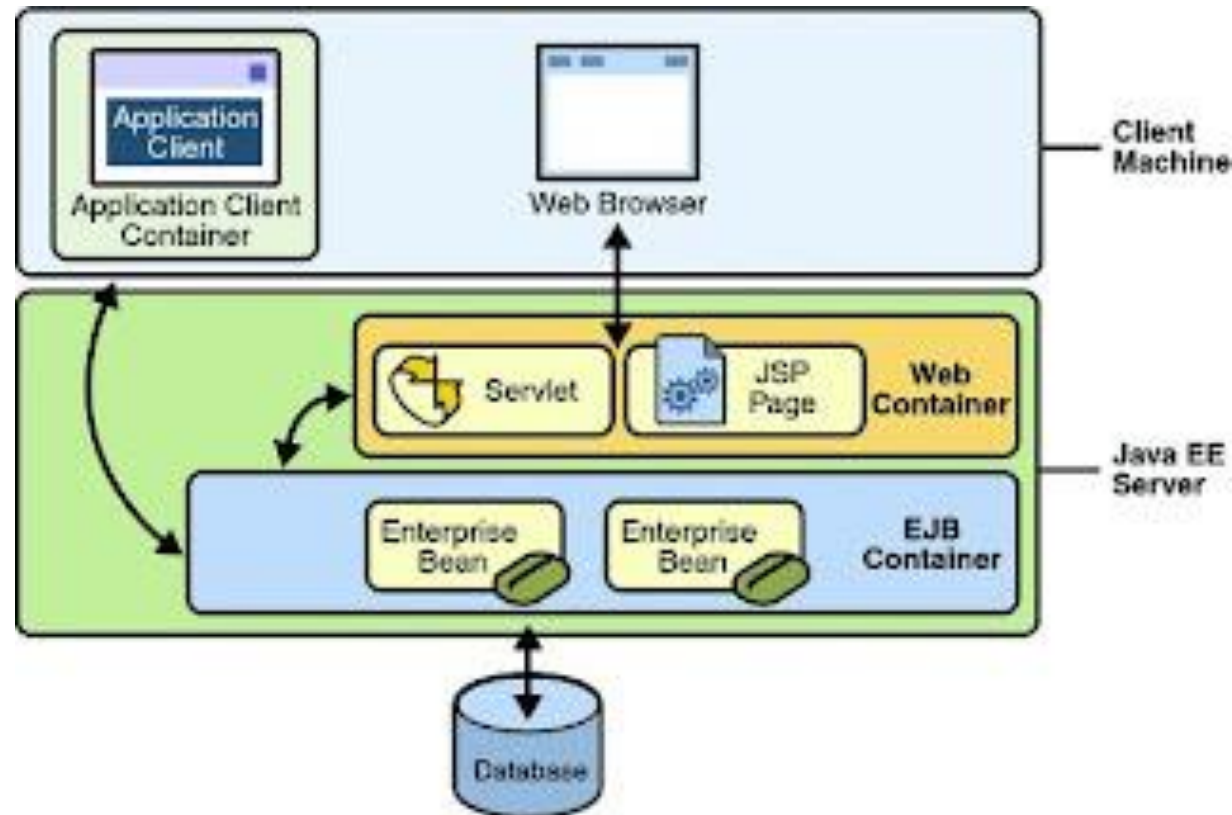
- Servicios de transacción y seguridad
- Pooling de recursos
- Control de ciclo de vida de EJB
- Concurrencia

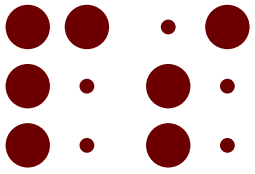




Enterprise JavaBeans

Esquema General

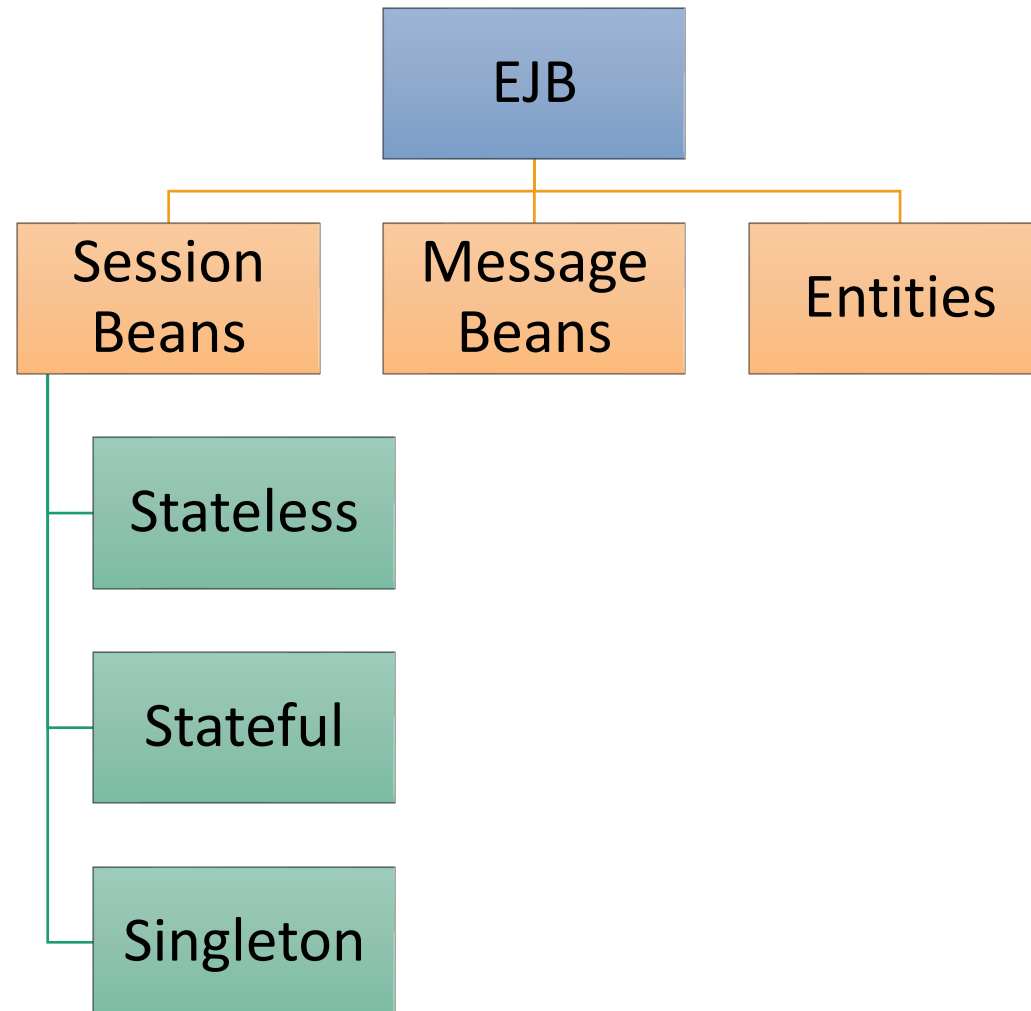


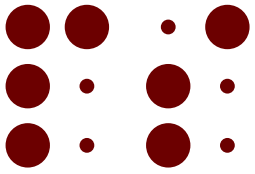


Enterprise JavaBeans

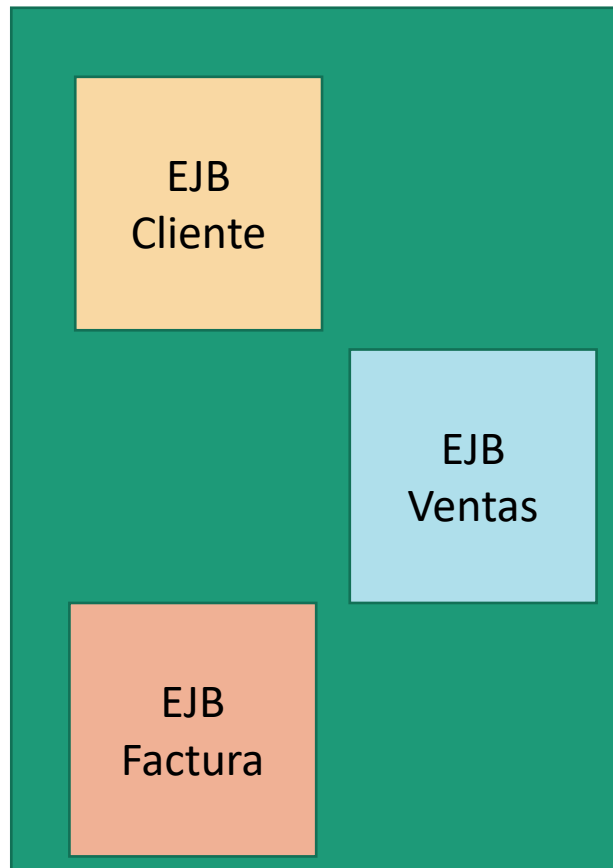
Esquema General

Model components

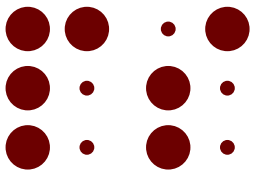




Session Beans



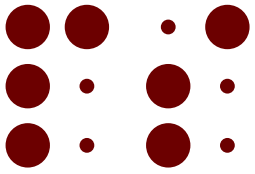
- Componentes Java
- Permiten establecer la lógica del negocio
- Viven dentro de un contenedor de EJB's



Stateless Bean

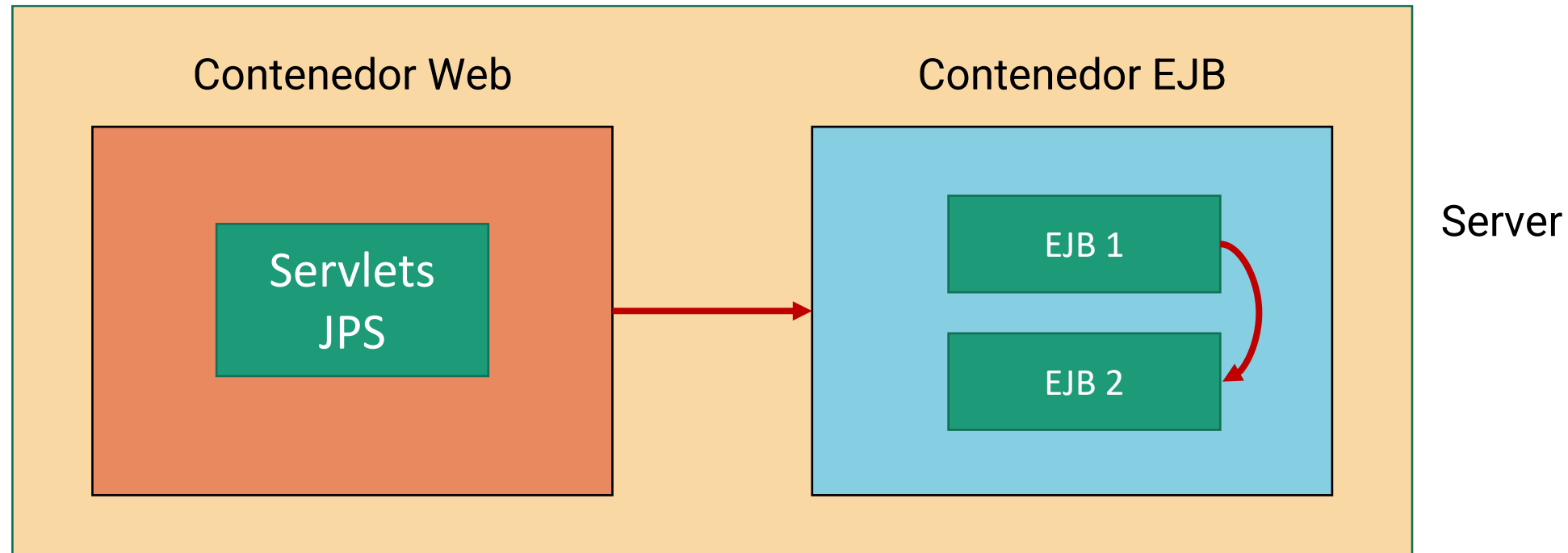
Es un tipo de Session Bean que conserva el estado con el cliente y se conforma por:

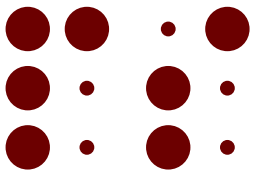
- Clase Java (con la implementación de los métodos de negocio).
- Una o más vistas de negocio (Interfaces POJI).
- Usará la anotación `@Stateless`



Stateless Bean Local

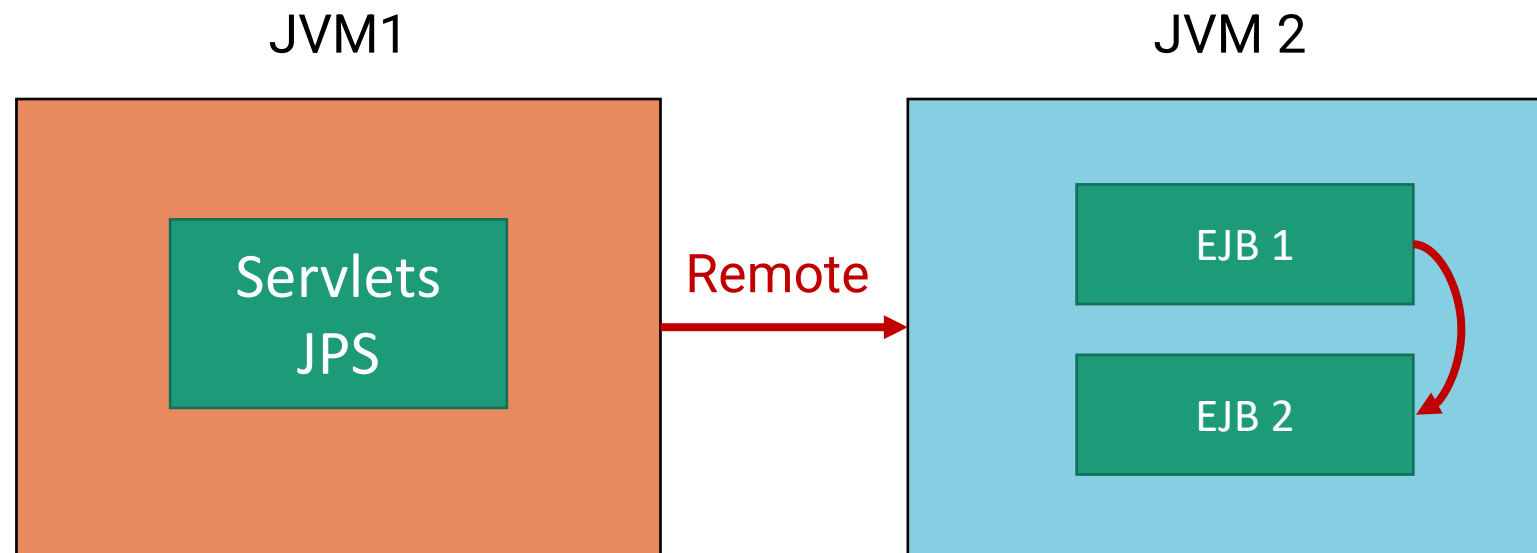
Los EJB's marcados como locales son accesibles desde la misma JVM ya sea por otro EJB o

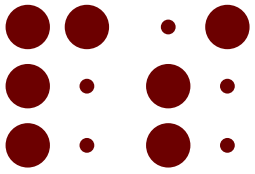




Stateless Bean Remote

Los EJB's marcados como remotos son accesibles desde otras JVM.

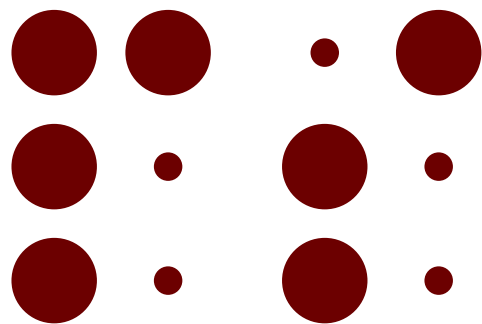




Stateful Bean

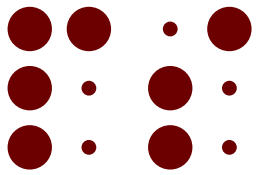
Es un tipo de Session Bean que permite mantener un estado conversacional con el cliente. Se conforma de:

- Clase Java (con la implementación de los métodos de negocio).
- Una o más vistas de negocio (Interfaces POJI).
- Usará la anotación `@Statefull`



Módulo 7

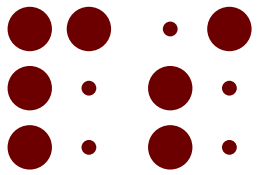
Java Naming Directory Interface



Java Naming Directory Interface

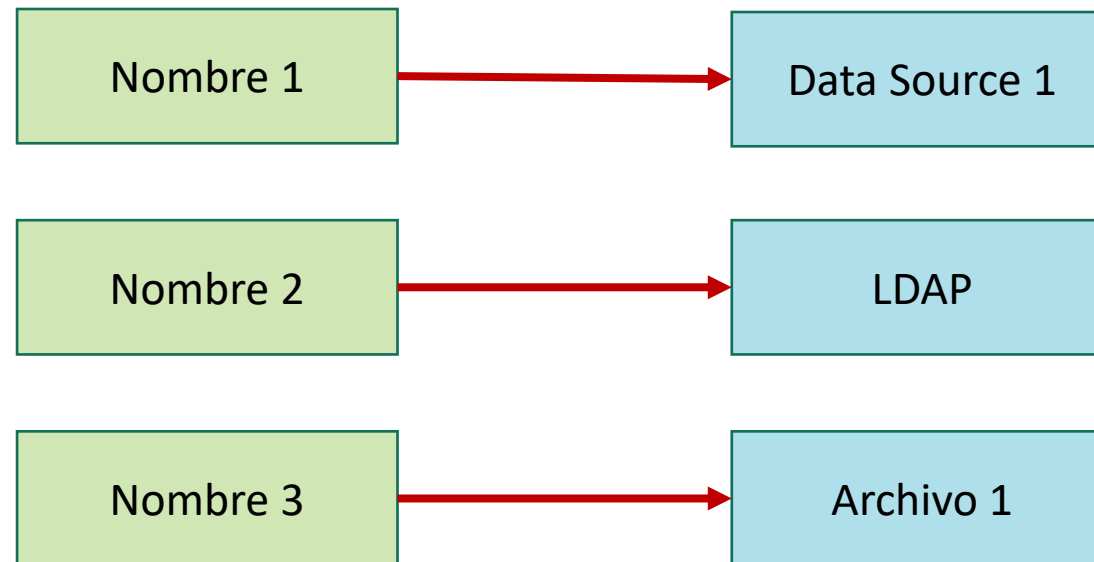
JNDI es una capa de abstracción Java para servicios de directorio, del mismo modo que JDBC (Java Database Connectivity) es una capa de abstracción para bases de datos.

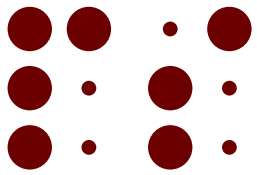
Estandariza el nombre y acceso a los servicios de nombres y directorios a través del cual podemos hacer búsquedas (look up).



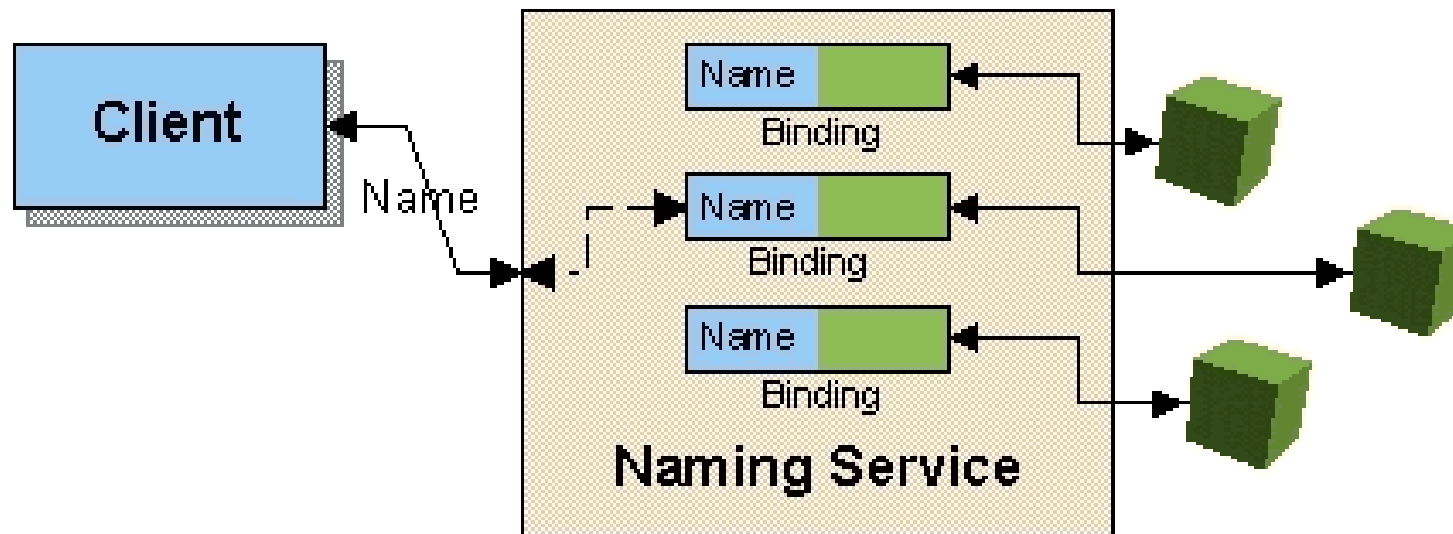
Java Naming Directory Interface

Naming system: Brinda un mecanismo para asociar nombres con objetos y provee una forma para buscar esos objetos a través del nombre.

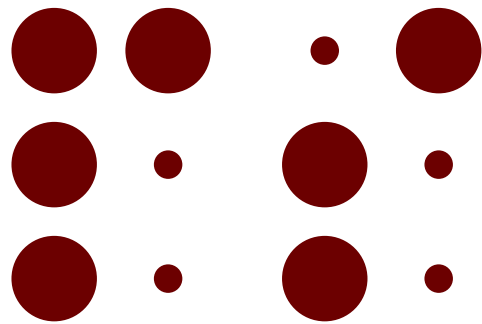




Java Naming Directory Interface

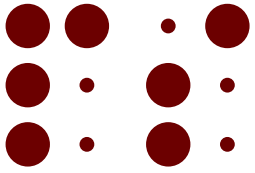


- El cliente obtiene un contexto.
- El cliente solicita un recurso a través del servicio de nombres.
- El cliente utiliza el recurso solicitado

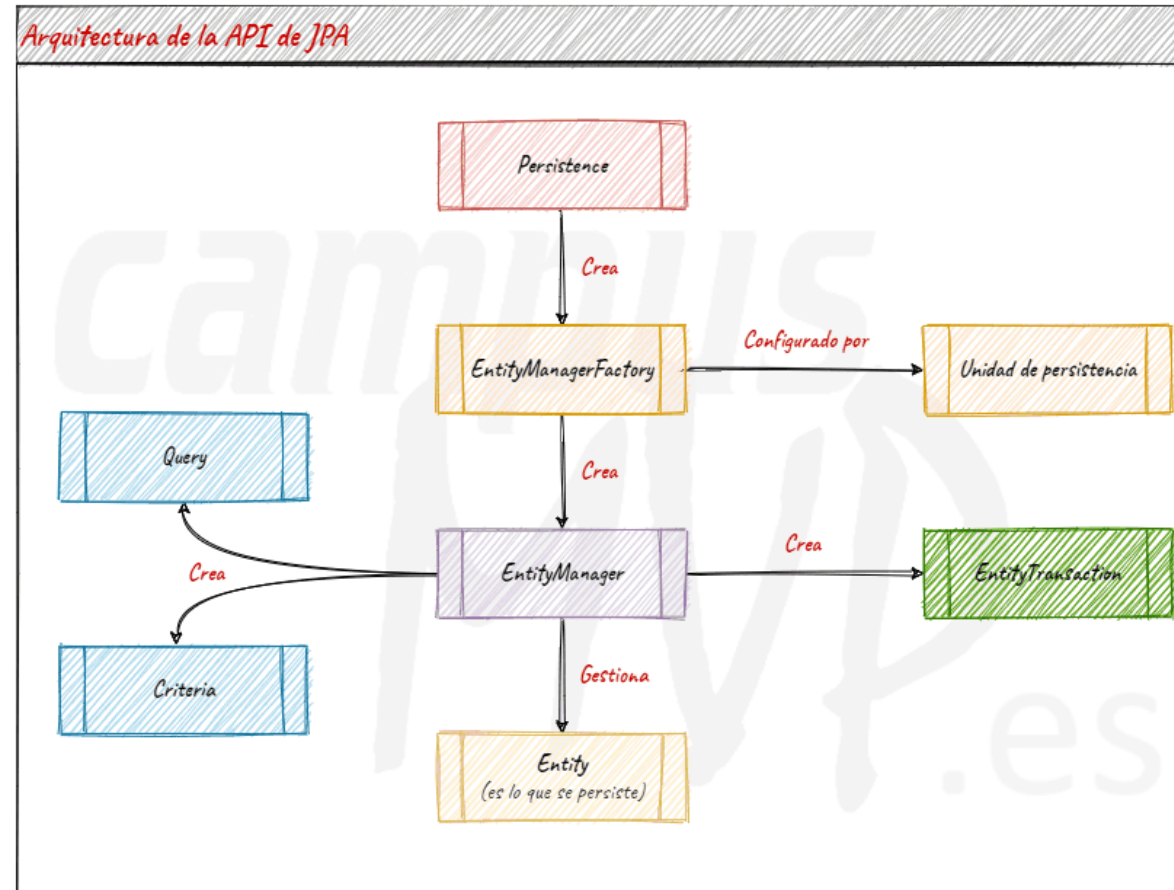
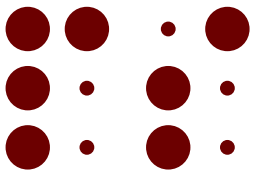


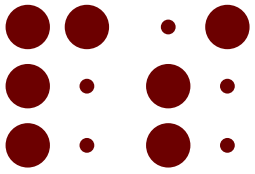
Módulo 8

Java Persistence API



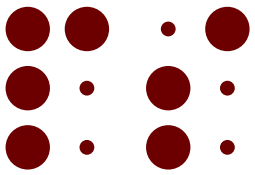
JPA es una especificación que indica cómo se debe realizar la persistencia (almacenamiento) de los objetos en programas Java.





Unidad de persistencia

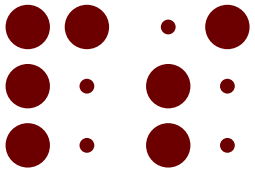
Este elemento es el encargado de configurar el acceso a la base de datos de tal forma que nuestras entidades pueden ser persistentes o seleccionadas de forma transparente. Este esta representado por el archivo persistence.xml



Entidad

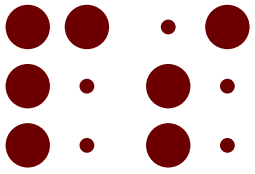
Las entidades en JPA no son más que POJO que representan datos que se pueden conservar en la base de datos. Una entidad representa una tabla almacenada en una base de datos. Cada instancia de una entidad representa una fila en la tabla.

```
import javax.persistence.Entity;  
  
@Entity  
public class Movie {  
  
}
```



EntityManager

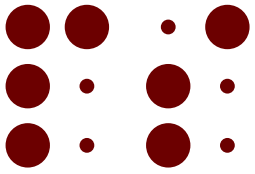
Es el componente que se encarga de controlar el ciclo de vida de todas las entidades definidas en la unidad de persistencia, y es mediante esta interface que se pueden realizar las operaciones básicas de una base de datos, como consultar, actualizar, borrar, crear (CRUD).



Java Persistence API

Relaciones: JPA permite definir relaciones entre entidades, como relaciones uno a uno, uno a muchos y muchos a muchos. Las anotaciones `@OneToOne`, `@OneToMany`, `@ManyToOne` y `@ManyToMany` se utilizan para configurar estas relaciones.

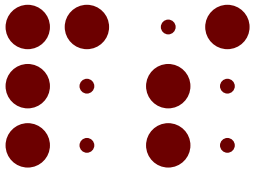
Transacciones: JPA proporciona soporte para gestionar transacciones en operaciones de persistencia. Puedes utilizar las anotaciones `@Transactional` o programáticamente mediante métodos como `begin`, `commit` y `rollback` en el `EntityManager`.



Anotaciones

Las anotaciones en JPA juegan un papel fundamental en el mapeo objeto-relacional, ya que definen cómo las entidades Java se relacionan con las tablas de la base de datos. Aquí hay algunas anotaciones comunes utilizadas en JPA:

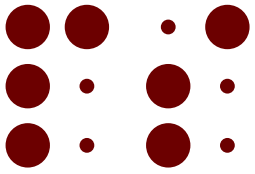
@Entity	Esta anotación se coloca encima de una clase Java para indicar que esa clase es una entidad, es decir, se mapea a una tabla en la base de datos.
@Table	Se utiliza para especificar el nombre de la tabla en la base de datos a la que se debe mapear la entidad.
@Column	Indica cómo un campo de la entidad se mapea a una columna de la tabla.
@Id	Marca un campo como la clave primaria de la entidad.
@GeneratedValue	Se utiliza junto con @Id para especificar cómo se generarán los valores de las claves primarias, como con estrategias de generación automática.



Anotaciones

Las anotaciones en JPA juegan un papel fundamental en el mapeo objeto-relacional, ya que definen cómo las entidades Java se relacionan con las tablas de la base de datos. Aquí hay algunas anotaciones comunes utilizadas en JPA:

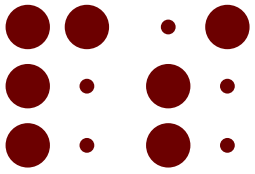
@JoinColumn	Se utiliza para especificar cómo se realiza el mapeo de una columna en una tabla de relación (tabla que maneja una relación entre dos entidades).
@NamedQuery @NamedQueries	Estas anotaciones se utilizan para definir consultas JPQL predefinidas que pueden ser reutilizadas en múltiples lugares de la aplicación.
@Transient	Marca un campo que no debe ser persistido en la base de datos. Es útil para campos calculados o temporales que no necesitan ser almacenados.



Relaciones

En JPA (Java Persistence API), se puede establecer diversas relaciones entre entidades para representar las asociaciones entre objetos en el modelo de datos. Aquí están las relaciones más comunes que se puede definir utilizando anotaciones en JPA:

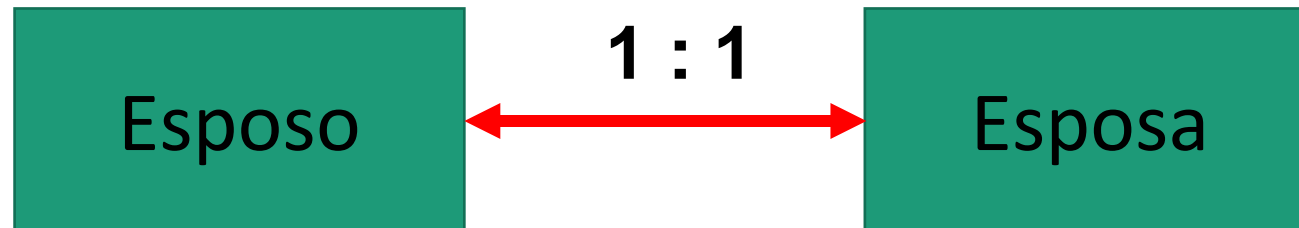
1. Uno a Uno
2. Uno a Muchos
3. Mucho a Uno
4. Muchos Muchos

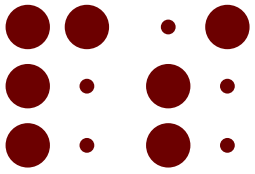


Relaciones

Uno a Uno

Relación Uno a Uno (@OneToOne): Esta relación indica que una entidad está relacionada con otra entidad de manera que cada registro de la primera entidad está asociado con exactamente un registro en la segunda entidad y viceversa.



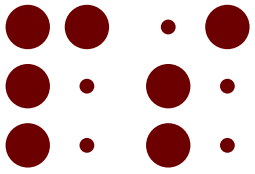


Relaciones

Uno a Muchos

Relación Uno a Muchos (@OneToMany): Esta relación se usa cuando una entidad está relacionada con una colección de otras entidades. Cada registro de la entidad principal está asociado con múltiples registros en la entidad relacionada.



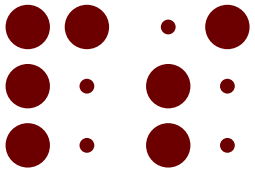


Relaciones

Muchos a Uno

Relación Muchos a Uno (@ManyToOne): Es la relación inversa de la relación uno a muchos. Indica que varias entidades de la entidad principal están relacionadas con una sola entidad de la entidad relacionada.



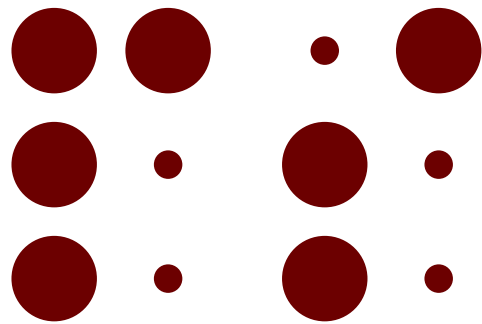


Relaciones

Muchos a Muchos

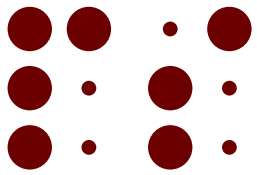
Relación Muchos a Muchos (@ManyToMany): Esta relación se usa cuando varias entidades de una clase están relacionadas con varias entidades de otra clase.





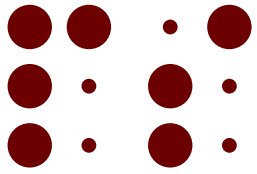
Módulo 9

Java Context Dependency Injection



Java Context Dependency Injection

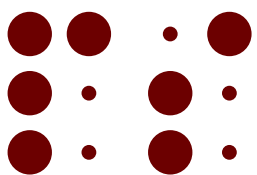
CDI (Contexts and Dependency Injection) es un marco de inyección de dependencia estándar incluido en Java EE 6 y superior. Nos permite gestionar el ciclo de vida de los componentes con estado a través de contextos de ciclo de vida específicos del dominio e inyectar componentes (servicios) en los objetos del cliente de forma segura.



Java Context Dependency Injection

Inversión de control

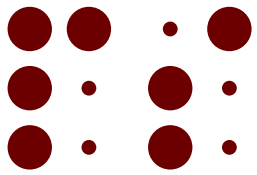
La inversión de control es un principio en la ingeniería de software mediante el cual el control de objetos o partes de un programa se transfiere a un contenedor o marco.



Java Context Dependency Injection

Inversión de control

El contenedor es responsable de crear instancias, configurar y ensamblar objetos conocidos como beans, así como de administrar su ciclo de vida y está representado por la interfaz `org.springframework.context.ApplicationContext`

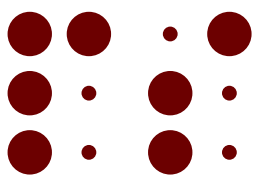


Java Context Dependency Injection

Inversión de control

Inyección de dependencias (DI)


Es el proceso a través del cual se conecta o se “inyecta” un objeto en otro que depende de este para su uso.



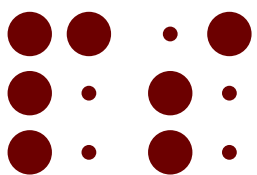
Java Context Dependency Injection

Inversión de control

```
public class Main {  
  
    public static void main(String[] args) {  
  
        ActorServiceImpl object = new ActorServiceImpl();  
        object.count();  
  
        if (object.equals("")) {  
            ActorServiceImpl objectTwo = new ActorServiceImpl();  
        }  
  
        object.toString();  
  
    }  
}
```

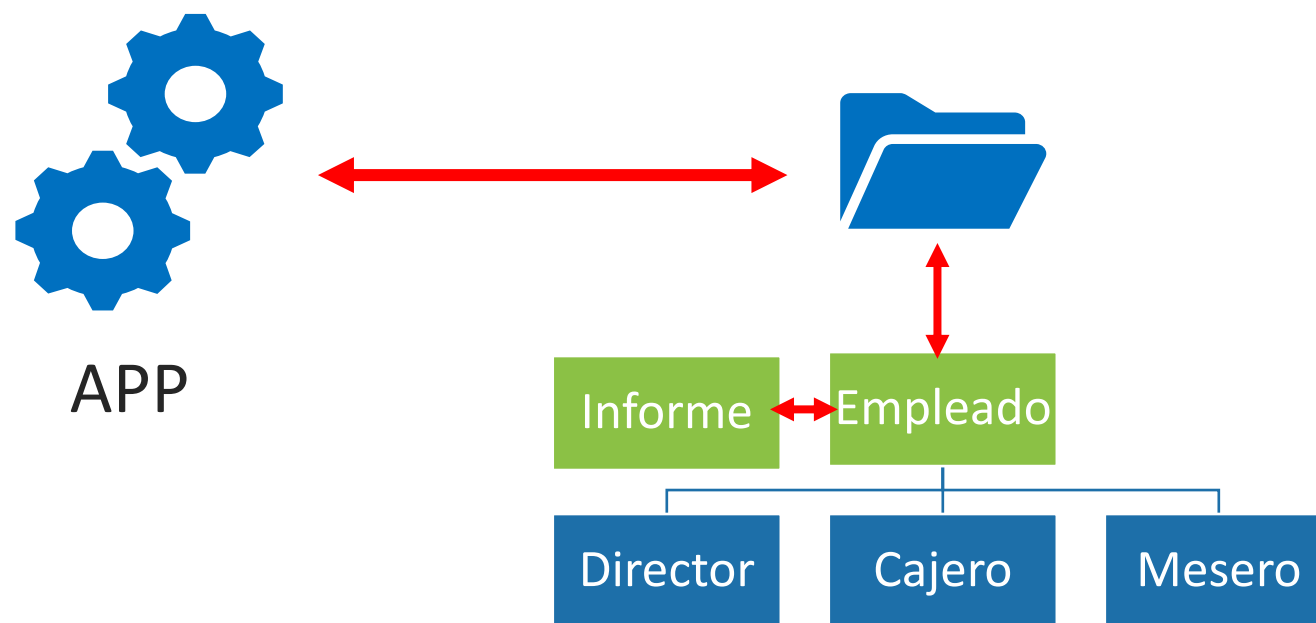


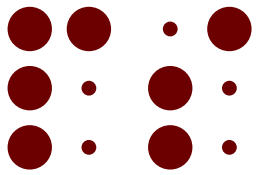
En el flujo normal de una aplicación, es esta quien crea los objetos y hace las llamadas a las funciones.



Inversión de control

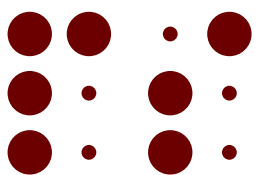
Java Context Dependency Injection





Java Context Dependency Injection

- `@RequestScoped`: El bean permanece durante la petición HTTP y es destruido al final.
- `@SessionScoped`: El bean es compartido por todas las peticiones HTTP que pertenecen a la misma sesión.
- `@ApplicationScoped`: El bean es creado en el arranque de la aplicación y destruido cuando la aplicación se detiene.
- `@Singleton`: Mantiene una instancia única del bean.
- `@Inject`: Permite hacer la inyección o instanciación de un bean.
- `@Named`: Permite declarar un servicio mediante un nombre.



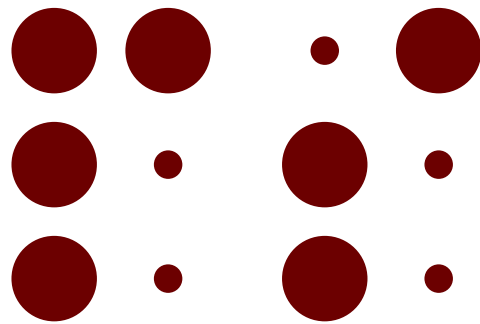
Java Context Dependency Injection

```
@Named
public class MiServicio {
    public void hacerAlgo() {
        System.out.println("Haciendo algo...");
    }
}

public class MiControlador {
    @Inject
    private MiServicio miServicio;

    public void realizarOperacion() {
        miServicio.hacerAlgo();
    }
}
```

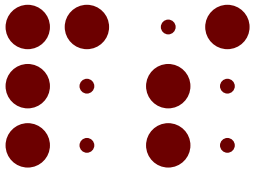
- Se tiene una clase de servicio llamada MiServicio la cual se anota @Named
- Se tiene un clase de tipo controlador llamada MiControlador, la cual tiene una propiedad privada de tipo MiServicio que se anota con la etiqueta @Inject



Módulo 10

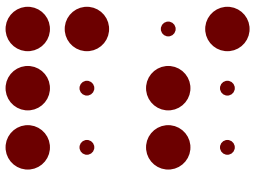
Java Transaction API





Java Transaction API

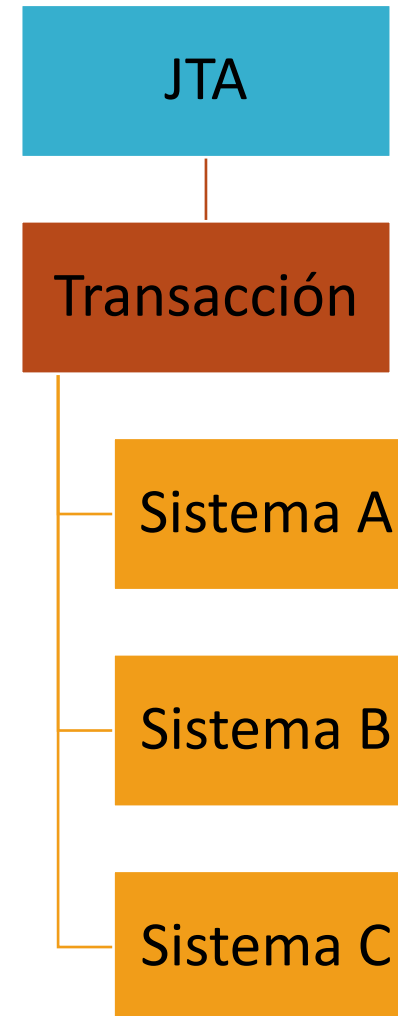
Java Transaction API, que es una especificación de programación para la gestión de transacciones en aplicaciones Java. Las transacciones son operaciones que involucran la actualización de recursos, como bases de datos, y se deben realizar de manera coherente y fiable, incluso en presencia de errores o fallas en el sistema.

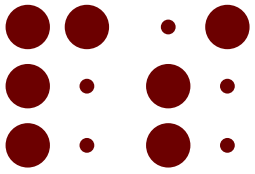


Java Transaction API

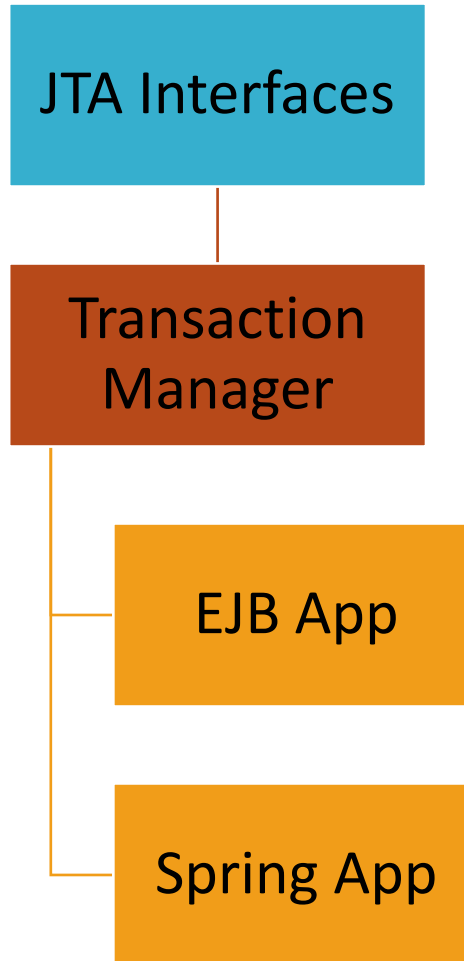
JTA proporciona una interfaz estándar para la gestión de transacciones en entornos Java, permitiendo a las aplicaciones realizar operaciones atómicas, consistentes, aisladas y duraderas (ACID).

La API permite a los desarrolladores demarcar bloques de código como transacciones, lo que garantiza que todas las operaciones dentro de ese bloque se completen correctamente o se reviertan si se produce un error.



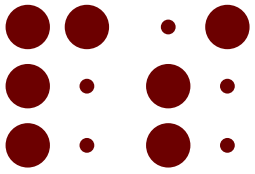


Java Transaction API



Transactional Manager que define como ha de implementarse un gestor de transacciones para ser compatible con JTA.

Las JTA interfaces delimitan las interfaces y anotaciones que los desarrolladores usan para gestionar las transacciones a nivel de código.

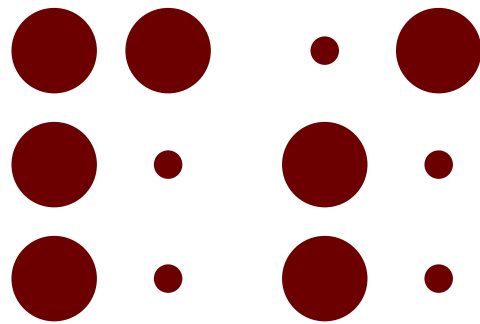


Java Transaction API

```
@Entity
@Setter
@Getter
public class Alumno {
    @Id
    private int id;
    @Column
    private String nombre;
}
```

```
@Stateless
@LocalBean
public class PersoDao {
    @PersistenceContext
    private EntityManager em;

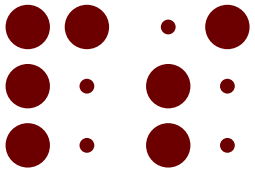
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void insertar(Persona personaA, Persona personaB) {
        em.persist(personaA);
        em.persist(personaB);
    }
}
```



Módulo 11

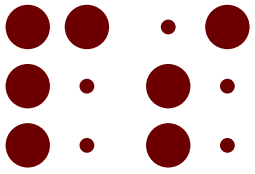
Java Message Service





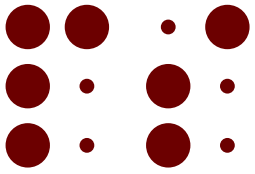
Java Message Service

Es una API de Java que proporciona un estándar para la mensajería asincrónica entre componentes de software en una red. JMS permite a los programas Java enviar, recibir, y leer mensajes de manera asíncrona, lo que facilita la comunicación entre aplicaciones distribuidas y la integración de sistemas.



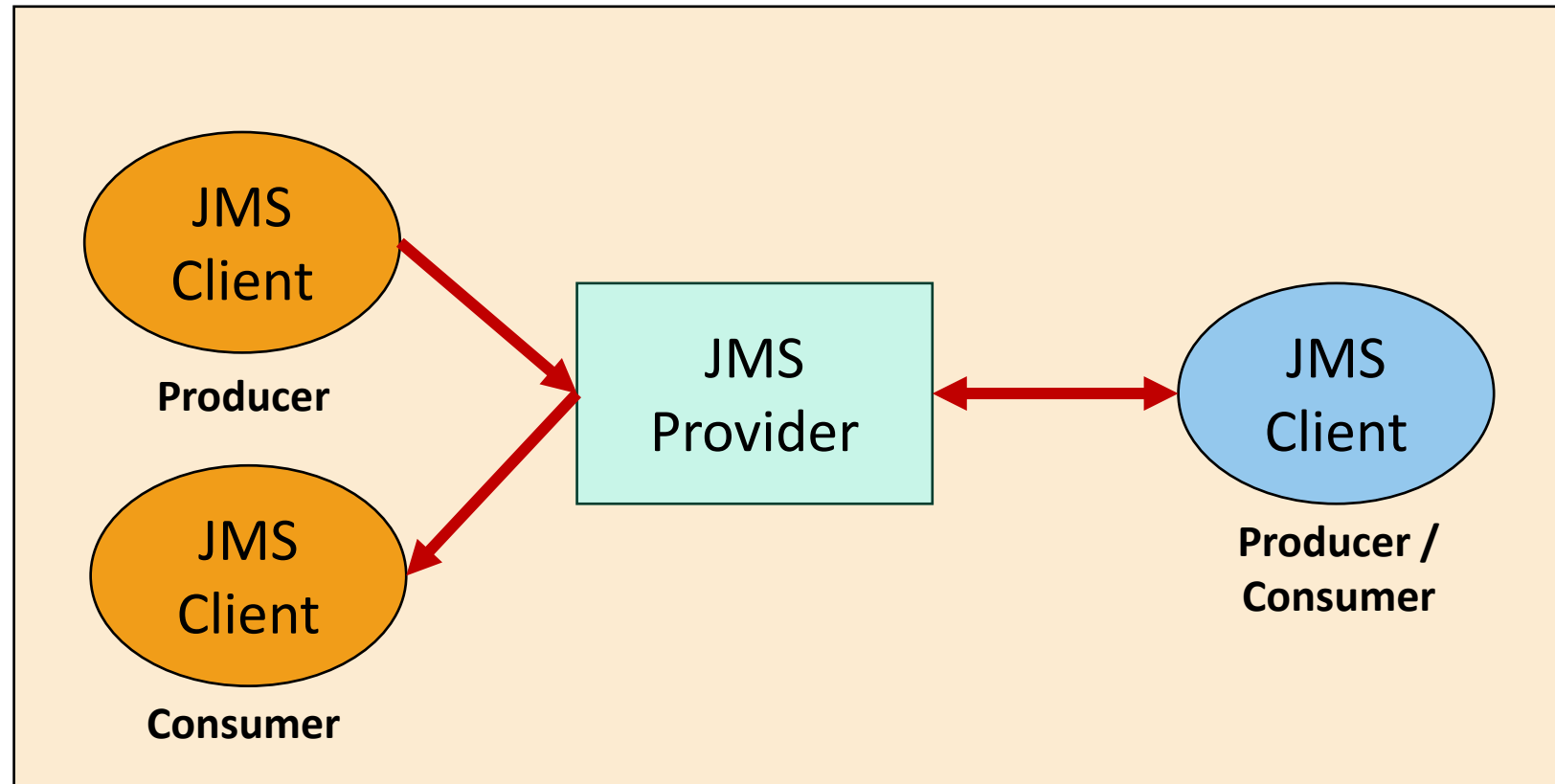
Productores y Consumidores de Mensajes

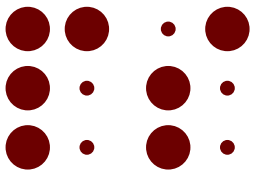
JMS define dos tipos principales de componentes: productores, que envían mensajes, y consumidores, que reciben mensajes. Los mensajes contienen información que puede ser compartida entre aplicaciones.



Java Message Service

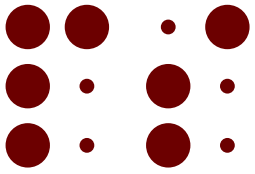
JMS Application





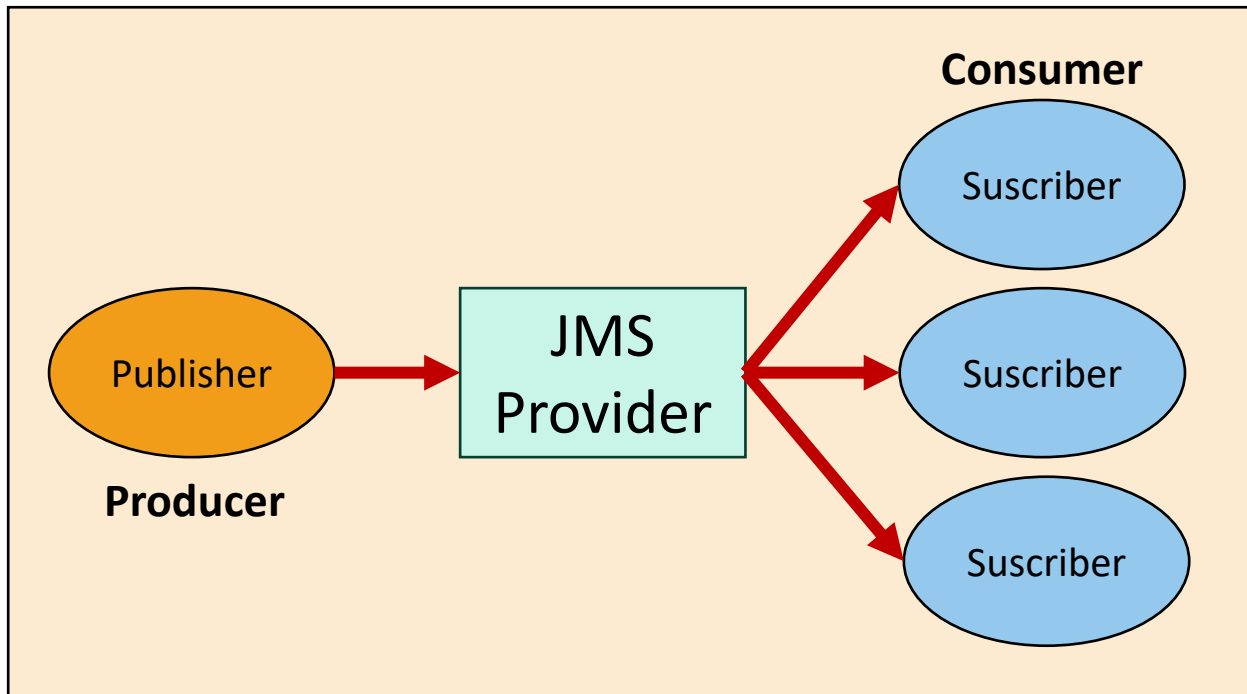
Java Message Service

- **JMS Client:** Es una aplicación Java que hace uso de la API de JMS.
- **JMS Provider:** Sistema que administra el envío/recepción de mensajes entre clientes.
- **Productor:** Es el cliente que envía el mensaje.
- **Consumidor:** Es el cliente que recibe el mensaje.
- **JMS Application:** Sistema de negocio compuesto de uno o mas clientes y un provider.

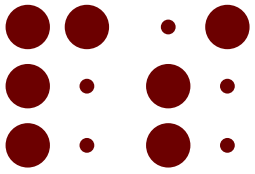


Java Message Service

Modelo Publish/Suscribe

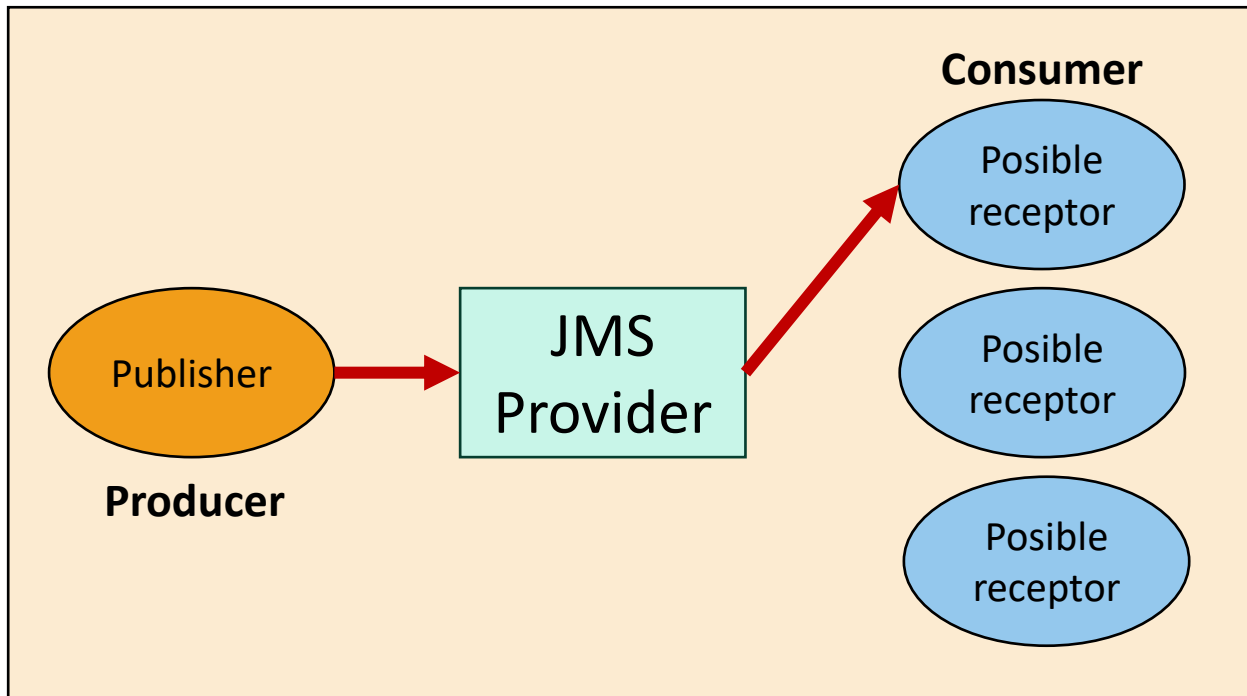


Este modelo se utiliza para la comunicación de publicación/suscripción, donde los mensajes se envían a múltiples suscriptores.



Java Message Service

Modelo P2P



Las colas se utilizan para la comunicación punto a punto, donde cada mensaje se entrega a un único consumidor.