Tecnologías en el Servidor
Java - MVC

Curso 2020/21

Javier Albert Segui

Universidad de Alcalá

Indice

- Introducción al patrón MVC
- Implementación del patrón MVC
  - Tecnología Servlets
  - Tecnología JSP
  - Tecnología JDBC

Introducción

- ¡Hacer software no es fácil!
- Diseñar software orientado a objetos es difícil, y diseñar software orientado a objetos reutilizable es todavía más difícil

  Chapter 1: Introduction. Design Patterns, The Gang of Four
- Un software capaz de evolucionar tiene que ser reutilizable
- ¡Hay que diseñar para el cambio!
- Para anticiparse a los cambios en los requisitos hay que diseñar pensando en qué aspectos pueden cambiar
- Los patrones de diseño están orientados al cambio

Introducción

- Los **patrones de diseño** son unas **técnicas** para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.
- Un patrón de diseño resulta ser una solución a un problema de diseño.
- Para que una solución sea considerada un patrón debe poseer ciertas características.
  - Debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores.
  - Debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Introducción

- Los patrones de diseño pretenden:
  - Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
  - Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
  - Formalizar un vocabulario común entre diseñadores.
  - Estandarizar el modo en que se realiza el diseño.
  - Facilitar el aprendizaje a las nuevas generaciones de diseñadores condensando conocimiento ya existente.
- Asimismo, no pretenden:
  - Imponer ciertas alternativas de diseño frente a otras.
  - Eliminar la creatividad inherente al proceso de diseño.
- "Abusar o forzar el uso de los patrones puede ser un error"

Patrón MVC

- Es un patrón de arquitectura de las aplicaciones software
- Separa la lógica de negocio de la interfaz de usuario
  - Facilita la evolución por separado de ambos aspectos
  - Incrementa reutilización y flexibilidad
- Utilizado en múltiples frameworks:
  - Java Swing
  - Apache Struts
  - ASP.net MVC
  - Ruby on Rails
  - GTK+
  - .....

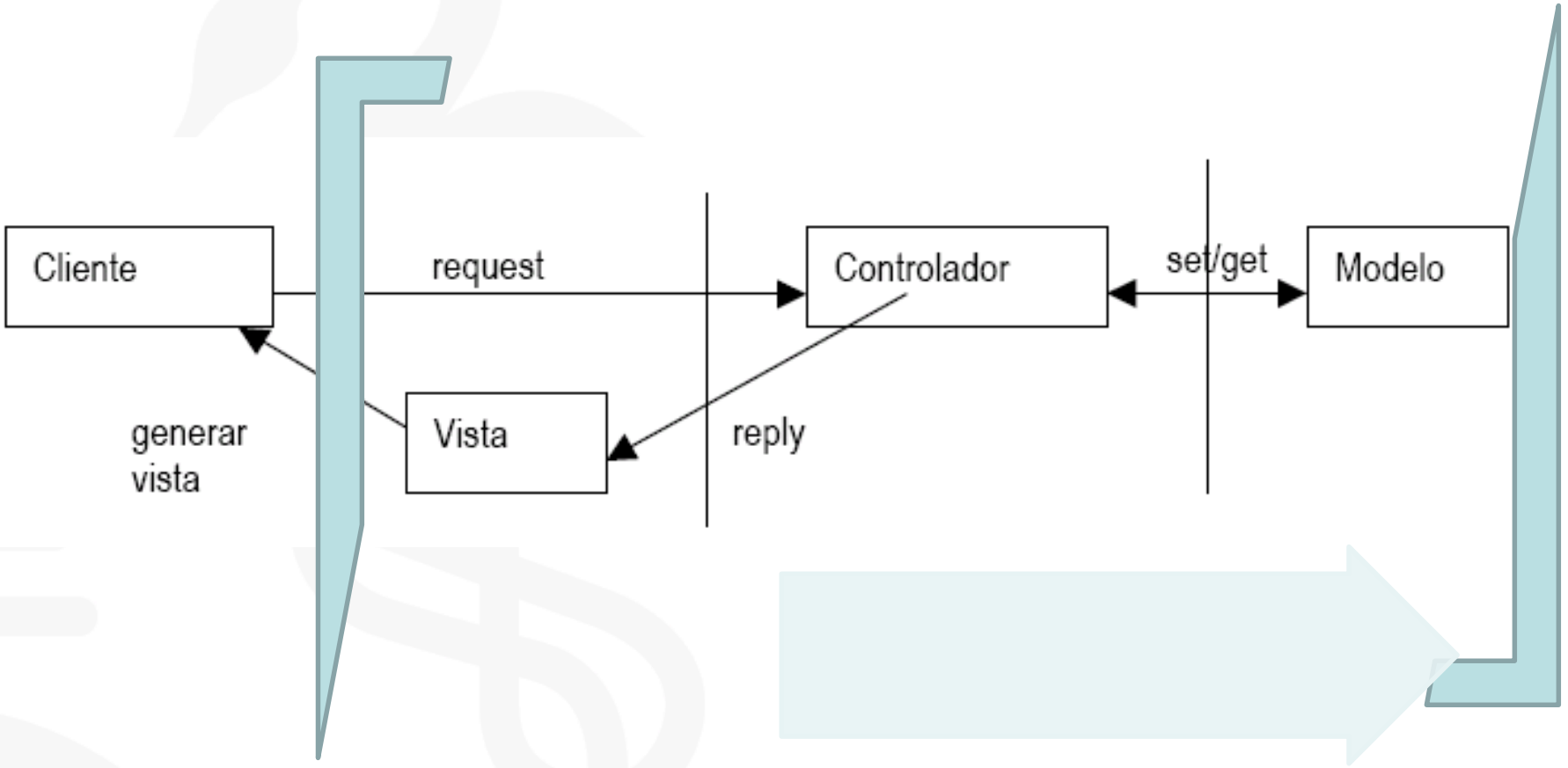
Patron MVC

- Modelo – Vista – Controlador
  - Un modelo
  - Varias vistas
  - Varios controladores
- Las vistas y los controladores suelen estar muy relacionados
- Los controladores tratan los eventos que se producen en la interfaz gráfica (vista)
- Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador

Patron MVC

- Flujo de control
  - El usuario realiza una acción en la interfaz
  - El controlador trata el evento de entrada
    - Previamente se ha registrado
  - El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta)
  - Se genera una nueva vista. La vista toma los datos del modelo
    - El modelo no tiene conocimiento directo de la vista
  - La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

Patron MVC


Cliente
request
Controlador
set/get
Modelo
generar
vista
Vista
reply

Patron MVC

- Vista:
  - página HTML
- Controlador:
  - código que obtiene datos dinámicamente y genera el contenido HTML
- Modelo:
  - la información almacenada en una base de datos o en XML junto con las reglas de negocio que transforman esa información (teniendo en cuenta las acciones de los usuarios)

Patron MVC

- Ventajas:
  - La implementación se realiza de forma modular.
  - Sus vistas muestran información actualizada siempre.
  - Cualquier modificación que afecte al dominio, como aumentar métodos o datos contenidos, implica una modificación sólo en el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
  - Las modificaciones a las vistas no afectan al modelo de dominio, simplemente se modifica la representación de la información.

Patrón MVC

- Desventajas:
  - Para desarrollar una aplicación bajo el patrón de diseño MVC es necesario una mayor dedicación en los tiempos iniciales del desarrollo.
  - MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación.

Patrón MVC - Modelo

- El **modelo** es un conjunto de clases que representan la información del mundo real que el sistema debe reflejar. Es la parte encargada de representar la lógica de negocio de una aplicación.
- El modelo, a nivel teórico, no debe tener conocimiento acerca de la existencia de las vistas y del controlador.
- **Modelo de Dominio**: Se entiende por modelo de dominio al conjunto de clases definidas a través del análisis de la situación real del problema que queremos solucionar.
- **Modelo de la aplicación**: El modelo de la aplicación es un conjunto de clases que sirven de puente en la relación de las vistas con el modelo de dominio. Tienen conocimiento de las vistas e implementan los mecanismos necesarios para notificar a éstas los cambios

Patrón MVC - Vistas

- Las **vistas** son las encargadas de la representación de los datos, contenidos en el modelo, al usuario.
- La relación entre las vistas y el modelo son de muchas a uno, es decir cada vista se asocia a un modelo, pero pueden existir muchas vistas asociadas al mismo modelo.

Patrón MVC - Controlador

- El **controlador** es el encargado de interpretar y dar sentido a las instrucciones que realiza el usuario, realizando actuaciones sobre el modelo.
- Si se realiza algún cambio, comienza a actuar, tanto si la modificación se produce en una vista o en el modelo. Interactúa con el Modelo a través de una referencia al propio Modelo.

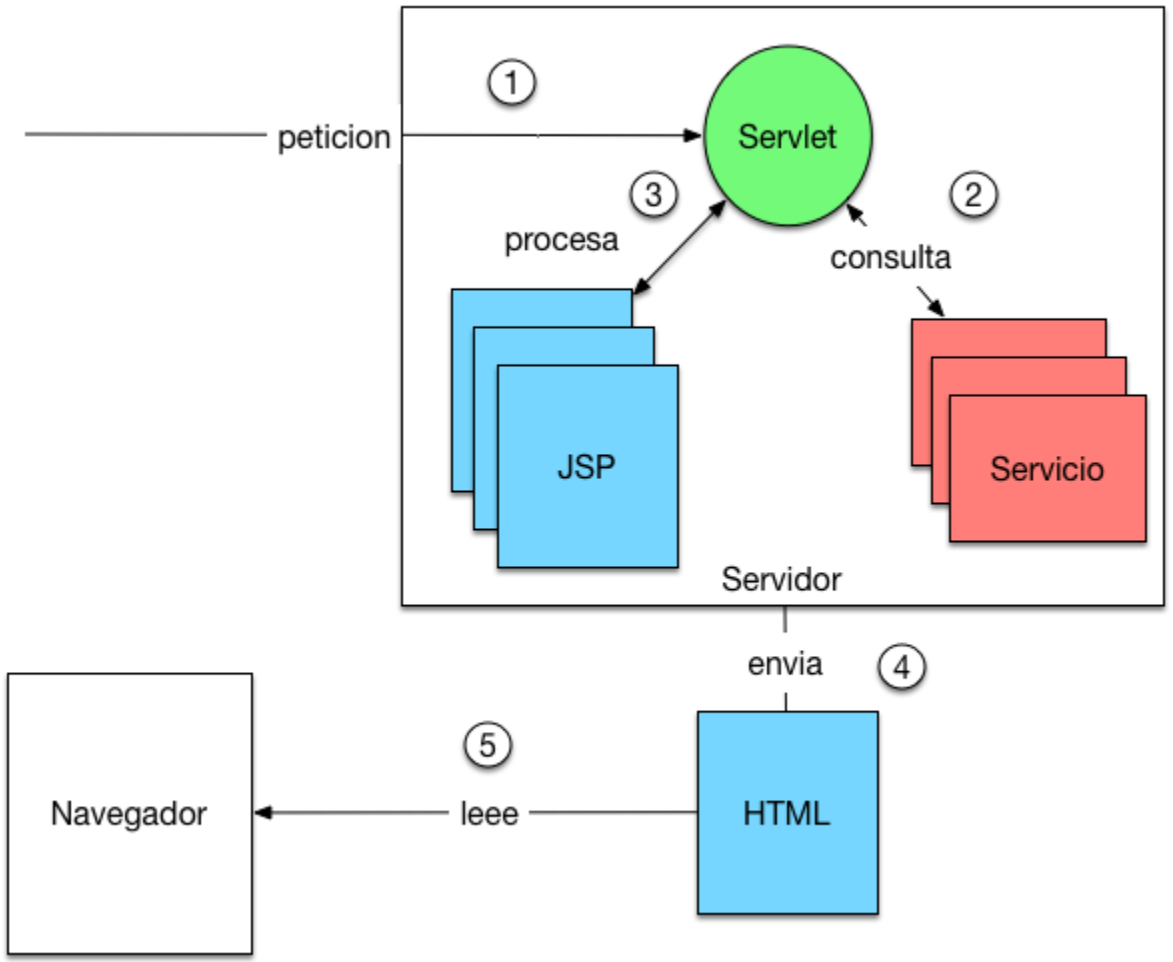
Patrón MVC - Funcionamiento

- Captura de la petición en el controlador
- Procesamiento de la petición
- Generación de respuestas

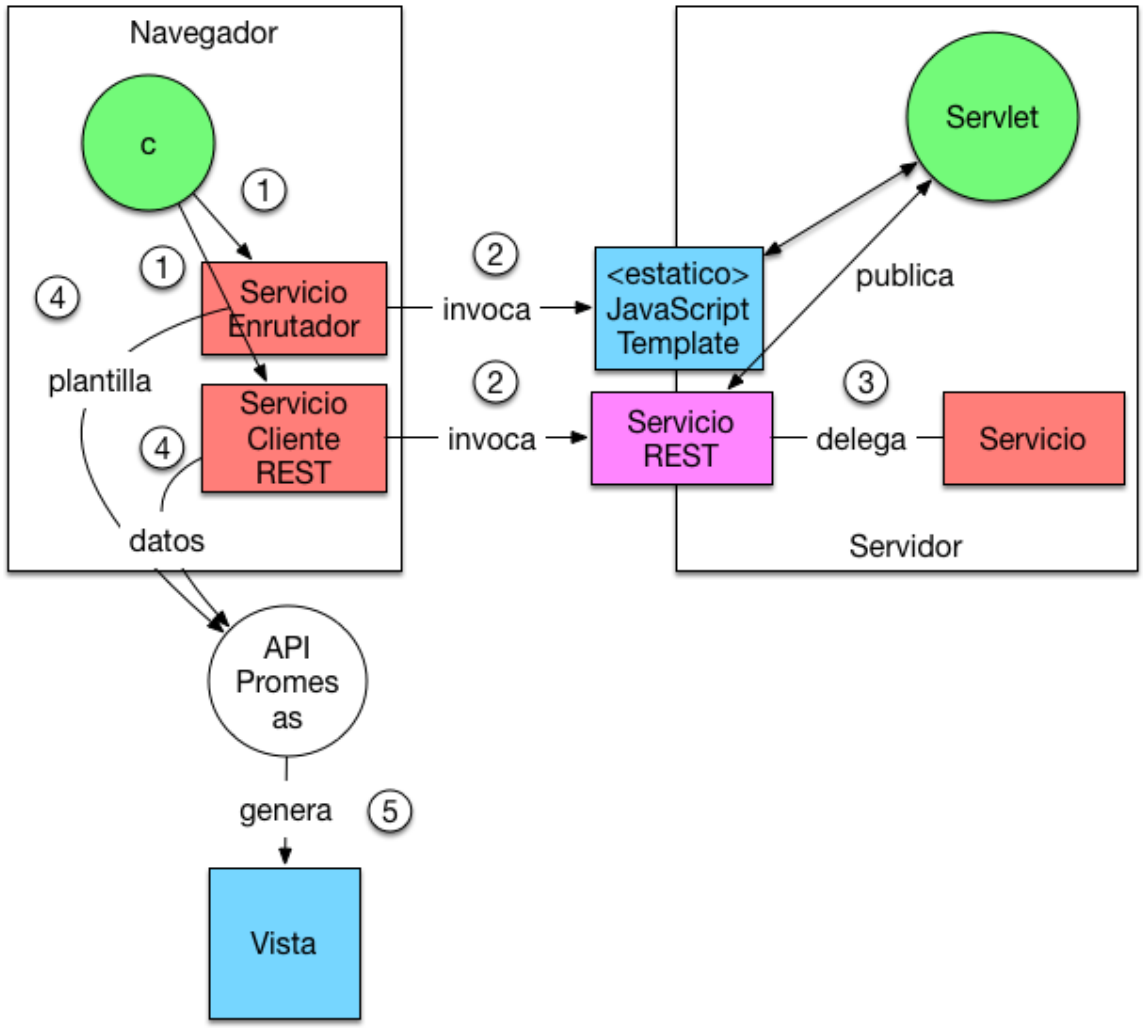
Patron MVC

- Para implementar el patrón lo primero que tengo que conocer son mis herramientas de trabajo dentro del lenguaje.
- Con un único objetivo: Desarrollar de manera eficiente una aplicación Web.
- No existe una única implementación del patrón de diseño MVC: Struts, JavaServerFaces (JSF), Spring, “la nuestra”, etc.

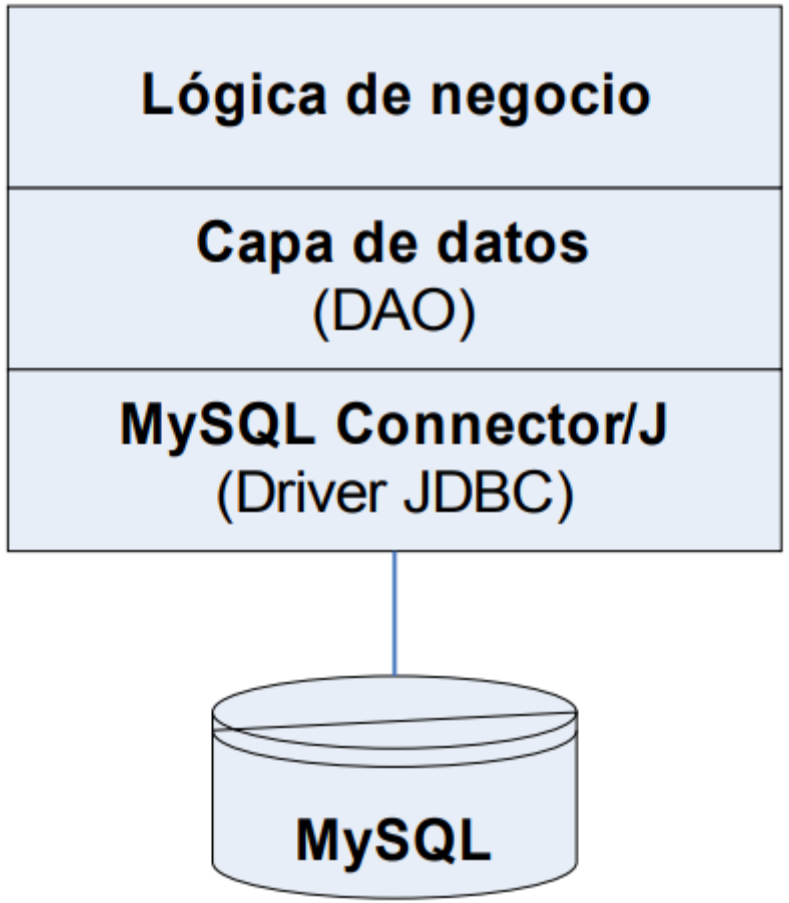
Patrón MVC - Servidor



Patron MVC - Cliente



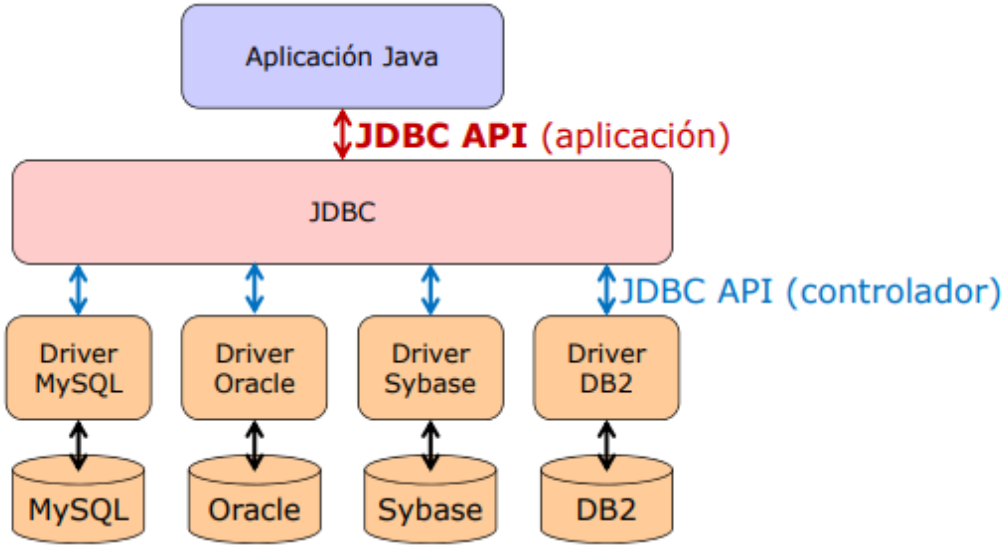
Patrones y Datos



JDBC

es un **API** (Application programming interface) que describe o define una librería estándar para acceso a fuentes de datos, principalmente orientado a Bases de Datos relacionales que usan **SQL (Structured Query Language)**.

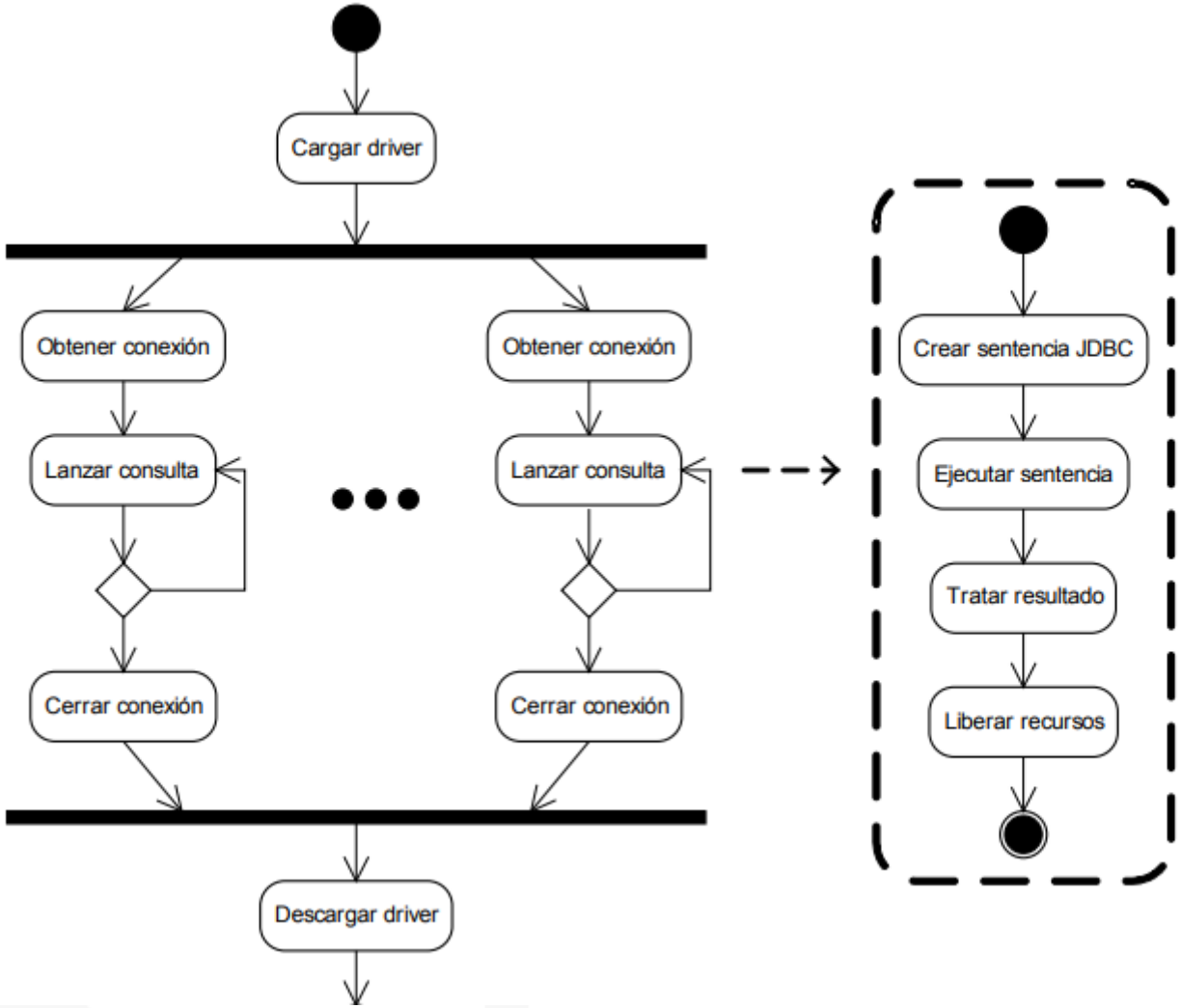
JDBC no sólo provee un interfaz para acceso a motores de bases de datos, sino que también define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones java el acceso a los datos.



JDBC

- Los drivers JDBC son adaptadores que convierten la petición proveniente del programa JAVA a un protocolo que el SGBD pueda entender.
  - Driver JDBC Tipo 1 (también llamado Puente JDBC-ODBC) convierte el método JDBC a una llamada a una función ODBC. Utiliza los drivers ODBC para conectar con la base de datos.
  - Driver JDBC Tipo 2 (también llamado driver API-Nativo) convierte el método JDBC a llamadas nativas de la API de la base de datos.
  - Driver JDBC Tipo 3. Hace uso de un Middleware entre el JDBC y el SGBD.
  - Driver JDBC Tipo 4 (también llamado Driver Java Puro directo a la base de datos). Es independiente a la plataforma.

JDBC


Cargar driver
Obtener conexión
Obtener conexión
Lanzar consulta
Lanzar consulta
Cerrar conexión
Cerrar conexión
Descargar driver
Crear sentencia JDBC
Ejecutar sentencia
Tratar resultado
Liberar recursos

JDBC

```
Connection conn = DriverManager.getConnection(
     "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
     "myLogin",
     "myPassword");
try {
     /* you use the connection here */
} finally {
    //It's important to close the connection when you are done with it
    try {
        conn.close();
    } catch (Throwable e) { /* Propagate the original exception
                               instead of this one that you want just logged */
        logger.warn("Could not close JDBC Connection",e);
    }
}
```

JDBC

```
try (Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM MyTable")
) {
    while (rs.next()) {
        int numColumns = rs.getMetaData().getColumnCount();
        for (int i = 1; i <= numColumns; i++) {
           // Column numbers start at 1.
           // Also there are many methods on the result set to return
           //  the column as a particular type. Refer to the Sun documentation
           //  for the list of valid conversions.
           System.out.println( "COLUMN " + i + " = " + rs.getObject(i));
        }
    }
}
```