Framework Spring

Programación Orientada a Objetos

Jonatan Goméz Perdomo, Ph.D.
jgomezpe@unal.edu.co

Carlos Andrés Sierra, M.Sc.
casierrav@unal.edu.co

Grupo de investigación en vida artificial – Research Group on Artificial Life – (Alife)
Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia

Agenda

Agenda

Desarrollo basado en capas: Definición

En el desarrollo de software, es buena práctica usar un **patrón** con el que se crean los diferentes componentes del programa. Cada patrón tiene ventajas, desventajas y herramientas específicas.

En Java, el patrón más popular es el **MVC**, o por sus siglas Modelo-Vista-Controlador. Dicho patrón permite separar la aplicación en tres categorías principales:

- El acceso a los datos.
- La lógica de la aplicación.
- La interfaz de usuario.

Desarrollo basado en capas: Modelo

El **Modelo** es la representación de los datos dentro de una aplicación y las distintas transacciones entre los datos.

El modelo se puede definir como **la información** almacenada en la base de datos, y las **reglas de negocio** que transforman esa información en objetos manipulables dentro de la aplicación.

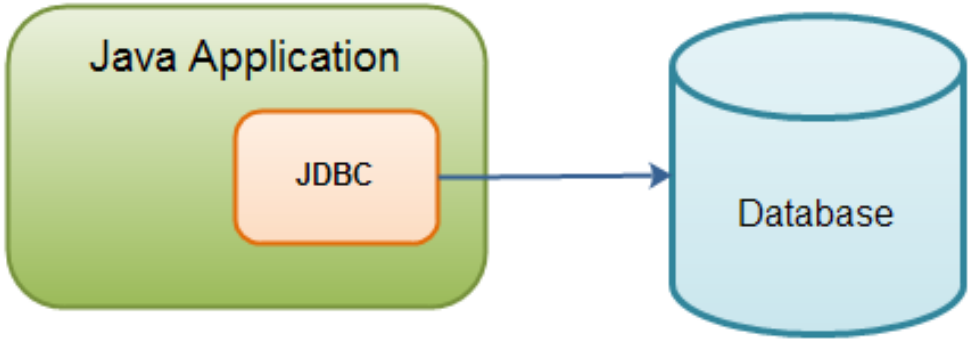


Figure: Imagen tomada desde este link

Desarrollo basado en capas: Controlador

El **Controlador** se refiere a la lógica de la aplicación, es decir, las reglas del negocio.
Esta capa es la encargada de manipular los datos conectando con la capa del Modelo, y gestionando las operaciones entre los eventos de la interfaz y los datos almacenados.

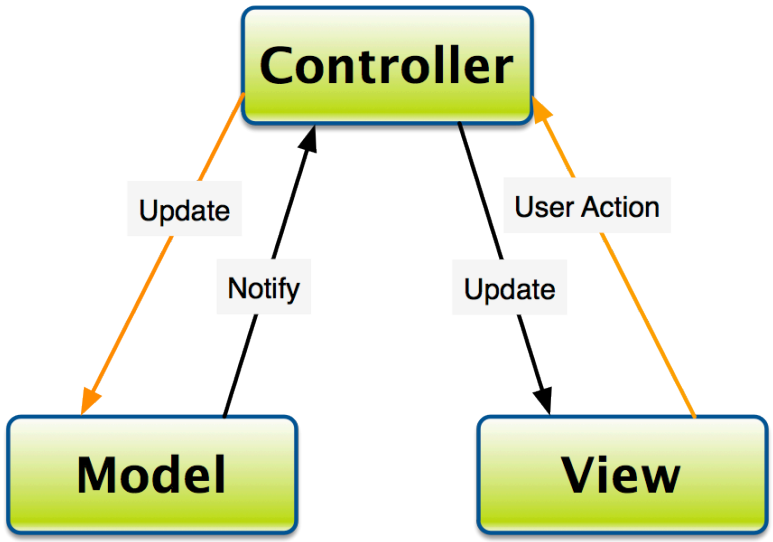


Figure: Imagen tomada desde este link

Desarrollo basado en capas: Vista

La **Vista** tiene que ver la interfaz con la que el usuario interactúa, la cual suele ser GUI. En esta capa no hay manipulación directa de datos, sino comunicación con la capa de Controlador para que esta sea la que obtenga y use los datos según la lógica del negocio.
Como tal, la **Vista** se encarga de mostrar los datos al usuario.

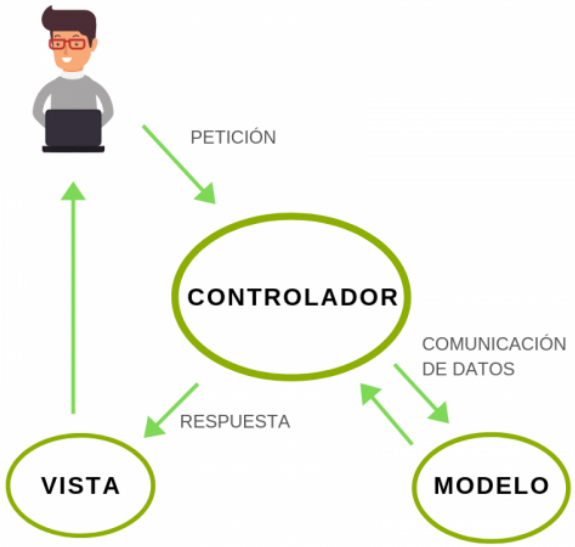


Figure: Imagen tomada desde este link

Desarrollo basado en capas: Ventajas

- Separación de la lógica de negocio de la interfaz de usuario.
- Escalabilidad y mantenimiento de la aplicación.
- Facilidad para pruebas unitarias.
- Reutilización de los componentes.
- Facilita el trabajo en equipo, ya que al separar los distintos componentes, se pueden trabajar por separado y luego integrarlos en una sola aplicación.

Agenda

Introducción a Spring Data I

Spring Data es un **Framework** de la familia de proyectos de *Spring*. Tiene como objetivo **simplificar** la capa de datos de la aplicación, accediendo tanto a bases de datos relacionales como no relacionales.

El objetivo de Spring Data es reducir la cantidad de código repetitivo, ofreciendo **interfaces** para el acceso a los datos, que pueden ser usadas y personalizadas fácilmente usando un conjunto específico de convenciones.

Introducción a Spring Data II

Todos los módulos de Spring Data se inspiran en los conceptos de **"agregado"**, **"agregado raiz"** y **"repositorio"** ("aggregate", "aggregate root", "repository"). Estos conceptos son particularmente relevantes cuando se trabaja con Spring Data JDBC y Spring Data JPA.

Introducción a Spring Data III

Un **Agregado** se define como un conjunto de entidades que se relacionan entre sí y dependen unas de otras. Cada agregado que se define tiene una entidad padre o raíz (**agregado raíz**) de la que dependerán el resto de entidades del agregado. Aunque los agregados tienen un significado propio, este no puede ser entendido por completo sin su objeto raíz.

Por ejemplo, para un *Usuario* se puede tener *Contacto*, pero dicho Contacto no tiene sentido fuera de la entidad Usuario. Es decir, cuando se actualice un Contacto, no se actualiza el *Contacto* en sí, sino que se actualiza el *Usuario*. Por tanto, en este caso, el *Usuario* es la entidad padre (agregado raíz) mientras que *Contacto* es su agregado.

Introducción a Spring Data IV

Los **repositorios** son colecciones en memoria de entidades o agregados. Sin embargo, en la práctica y con frecuencia se usan los repositorios como medio de acceso a la base de datos. Normalmente, se crea un repositorio por Entidad Raíz de una Agregación.

Los repositorios se usan para:

- Persistir entidades o agregados.
- Recuperar entidades o agregados conocida su identidad.
- Recuperar colecciones de entidades que cumplan alguna condición.

Agenda

Agenda

Spring Data JDBC

Spring Data JDBC es un módulo que hace parte de la familia Spring Data, facilita la implementación de los repositorios basados en JDBC.

Spring Data JDBC pretende ser conceptualmente sencillo. Con el fin de lograr esto, Spring Data JDBC **NO** ofrece muchas de las características encontradas en otros ORMs. Esto hace de Spring Data JDBC un opción simple pero limitada.

Agenda

Spring Data JDBC - Primeros pasos I

Paso 1: Agregar Spring Data JDBC al proyecto.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
```

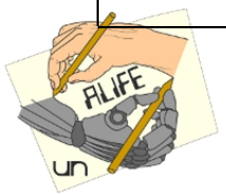
Esta dependencia no incluye el driver de la base de datos, este debe ser proporcionado aparte.

Spring Data JDBC - Primeros pasos II

Paso 2: Añadir Entidades.

En este paso se mapea o modela la base de datos en entidades.

```
public class Museo {
    @Id
    private long id;
    private String Fecha_presentacion;
    private int IdMuseo;
    private String NomMuseo;
    ...
    private int costo;
    // constructores, getters, setters
}
```



Spring Data JDBC - Primeros pasos III

Paso 3: Declarar el repositorio.

En este paso se declara un nuevo repositorio que hará uso del CRUD Repository Interface que provée Spring y que servirá para realizar operaciones CRUD básicas.

CRUD Repository Interface

```
public interface CrudRepository<T, ID>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID id);

  void delete(T entity);

  // more methods
}
```

Spring Data JDBC - Primeros pasos IV

El CRUD Repository Interface hace a su vez uso del Root Repository Interface el cual se muestra a continuación:

Root Repository Interface

entity's
id type

```
public interface Repository<T, ID> { }
```

type of entity
to persist

Spring Data JDBC - Primeros pasos V

Ejemplo

Declaracion de un repositorio para la entidad Museo.

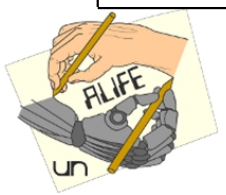
```
@Repository
public interface MuseoRepository extends
CrudRepository<Museo, Long> {

}
```

Spring Data JDBC - Create

La insercion de datos o creacion de registros puede hacerse de la siguiente manera:

```
@Component
public class DatabaseSeeder {
    @Autowired
    private JdbcTemplate jdbcTemplate;
    public void insertData() {
        jdbcTemplate.execute("INSERT INTO Museo
        (Fecha_presentacion, Id_museo,..,costo)
        VALUES('10/10/20', 101,..., 450)");
    }
}
```



Spring Data JDBC - Read

La consultas a los datos pueden hacerse de la siguiente manera:

```
@Repository
public interface MuseoRepository extends
CrudRepository<Museo, Long> {

  @Query("select * from museo where Id_museo=:IdMuseo")
  List<Museo> findByIdMuseo(@Param("IdMuseo") int IdMuseo);

}
```

Spring Data JDBC - Update

Las modificaciones o actualizaciones se realizan de forma muy similar a las consultas

```
@Repository
public interface MuseoRepository extends
CrudRepository<Museo, Long> {

  @Modifying
  @Query("UPDATE museo SET Nom_museo = :name
  WHERE Id_museo = :IdMuseo")
  boolean updateByNomMuseo(@Param("IdMuseo") int IdMuseo,
  @Param("name") String name);

}
```

Spring Data JDBC - Delete

Para realizar operaciones de borrado se procede de la siguiente manera:

```
@Component
public class DatabaseDelete {
    @Autowired
    private JdbcTemplate jdbcTemplate;
    public void delete(int id) {
        String sql="DELETE FROM museos WHERE Id_museo = ?";
        jdbcTemplate.update(sql, id);
    }
}
```

Agenda

1. Framework Spring Data
   - Desarrollo basado en capas
   - Introducción a Spring Data
2. Spring Data JDBC
   - Introducción a JDBC
   - CRUD usando JDBC
3. **Spring Data JPA**
   - Introducción a JPA
   - CRUD usando JPA
4. Ejercicios

Agenda

Introducción a JPA

Spring Data JPA es otro módulo que hace parte de la familia Spring Data. Facilita el uso de los repositorios basados en JPA (Java Persistence API).

Este módulo ofrece mejoras para capas de datos basadas en JPA, lo que hace que construir aplicaciones con Spring sea más poderoso para tecnologías orientadas al acceso a datos.

Spring Data JPA apunta a mejorar las implementaciones de las capas de acceso a datos reduciendo el código repetitivo a la hora de ejecutar consultas sencillas, paginación o auditorías.

Características de Spring Data JPA

- Soporte especializado para construir repositorios basados en Spring y JPA.
- Soporte para la paginación, ejecución de consultas dinámicas, y la posibilidad de integrar código personalizado para el acceso a datos.
- Validación de consultas marcadas por la convención `@Query` antes de su ejecución.
- Soporte para mapeo de entidades basadas en XML.

Agenda

Spring Data JPA - Primeros pasos I

Primer paso: Agregar Spring Data JPA al proyecto.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

* Esta dependencia no incluye el driver de la base de datos, este debe ser proporcionado por el desarrollador.

Spring Data JPA - Primeros pasos II

Paso 2: Añadir entidades.

En este paso se mapea o modela la base de datos en entidades.

```
public class Persona {
    private String numeroCedula;
    private String nombre;
    private String apellidos;
    ...
    // constructores, getters, setters
}
```

Spring Data JPA - Primeros pasos III

Paso 3: Crear la interfaz de tipo Repositorio que nos permita gestionar las operaciones CRUD.

En este paso se crea el repositorio, que extiende de JPARepository.

```
import org.springframework.data.jpa
        .repository.JpaRepository;

public interface PersonaRepository
    extends JpaRepository<Persona,String>{
}
```

Spring Data JPA - Primeros pasos IV

Paso 4: Usar el repositorio.

Es hora de usar el repositorio a través de Spring Framework. Para ello cargamos los diferentes beans y usamos el método `save()` del repositorio.

```
public class SpringDataApplication {

public static void main(String[] args) {
    AnnotationConfigApplicationContext contexto =
        new AnnotationConfigApplicationContext
            (SpringConfiguracion.class);
    Persona p = new Persona("101010","Juan","Quijote");
    PersonaRepository pRepositorio =
        contexto.getBean(PersonaRepository.class);
    pRepositorio.save(p);
```

Spring Data JPA - Primeros pasos V

Spring Data JPA nos ofreció la mayor parte del código a la hora de crear el repositorio. El trabajo que tendremos que hacer será crear el modelo de datos, puesto que el Framework aporta la implementación del código repetitivo en el repositorio.

Esto ofrece mucha **velocidad** y **versatilidad**, además de proveer distintas ventajas.

Agenda

1. Framework Spring Data
   - Desarrollo basado en capas
   - Introducción a Spring Data
2. Spring Data JDBC
   - Introducción a JDBC
   - CRUD usando JDBC
3. Spring Data JPA
   - Introducción a JPA
   - CRUD usando JPA
4. Ejercicios

Ejercicios

Implemente el modelo de datos `Persona`, que tendrá como atributos la cédula, el nombre y la edad. Haga la conexión con la base de datos usando:

- Spring Data JDBC
- Spring Data JPA

¿Cuáles son las diferencias de usar JDBC a JPA? ¿Qué ventajas tiene uno sobre el otro? ¿En qué casos podría resultar útil JDBC? ¿En qué casos resultaría útil JPA?