¿Qué es Spring Framework?

Es un robusto Framework para el Desarrollo de Aplicaciones Empresariales en el lenguaje Java

- Aplicaciones Web MVC.
- Aplicaciones empresariales.
- Aplicaciones de escritorio.
- Aplicaciones Batch.
- Integración con REST/SOA.
- Spring Data.
- Spring Security.

Características

Entre las características de Spring, tenemos las siguientes que ofrecen una cantidad considerable de servicios:

- **Tecnologías**: como la inyección de dependencias, eventos, recursos, validación, enlace de datos, etc.
- **Acceso a datos**: soporte **DAO, JDBC, ORM, Marshalling XML**.
- **Gestión de transacciones**.
- **Pruebas (Testing)**: simulacro de objetos, el framework TestContext, Spring MVC prueba, WebTestClient.
- **Programación orientada a aspectos (AOP)**: permite la implementación de rutinas transversales.
- MVC (**Modelo Vista Controlador**).
- **Seguridad**.
- **Frameworks web**: Spring WebFlux y Spring MVC.

Arquitectura Spring

La arquitectura se compone en distintas capas, cada una tiene su función específica:

Capa web: Spring simplifica el desarrollo de interfaces de usuario en aplicaciones web MVC mediante el soporte de varias tecnologías para generación de contenido, entre ellas JSP, Thymeleaf, FreeMaker, Velocity, etc.

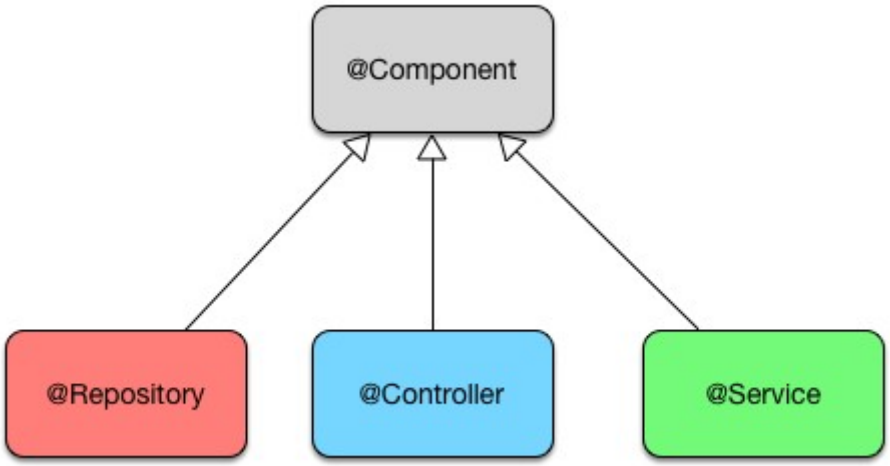
Capa lógica de negocio: en esta capa podemos encontrar tecnología como los Java Beans (POJOS), Dao Support, Services, EJBs y clases Entities.

Capa de Datos: aquí vamos a encontrar tecnologías JDBC, ORM (JPA, Hibernate, etc), Datasource y conexiones a bases de datos.

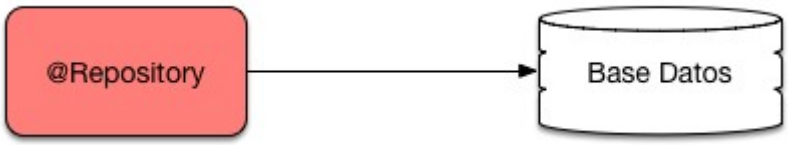
Spring Stereotypes y anotaciones

Spring define un conjunto de anotaciones core que categorizan cada uno de los componentes asociándoles una responsabilidad concreta.

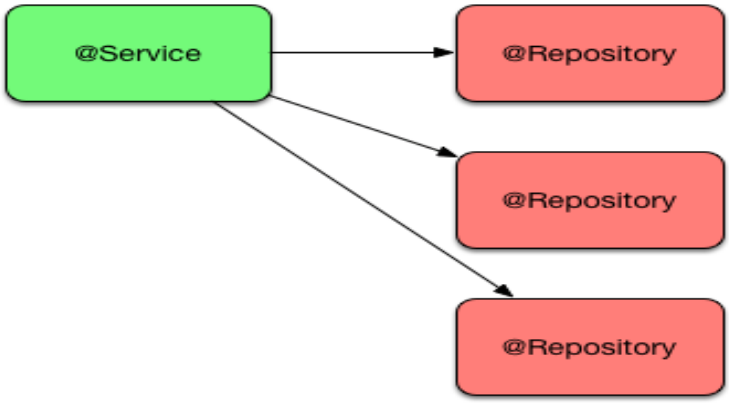
@Component: Es el estereotipo general y que permite anotar un bean para que spring lo considere uno de sus objetos.



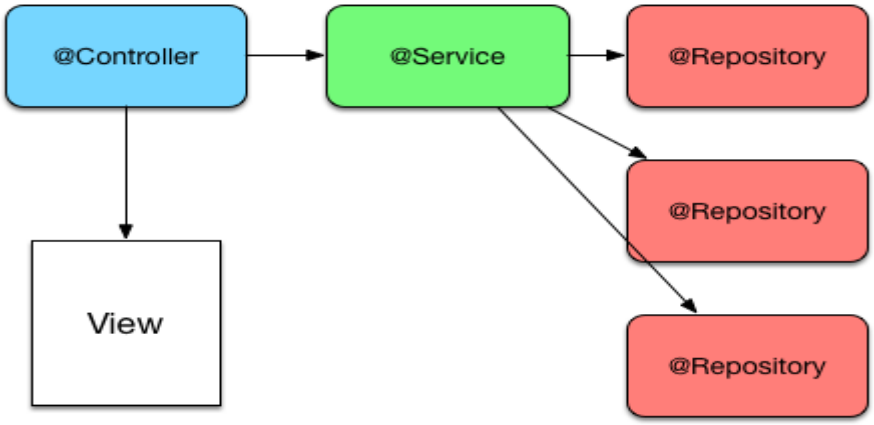
@Repository: Es el estereotipo que se encarga de dar de alta un bean para que implemente el patrón repositorio que es el encargado de almacenar datos en una base de datos o repositorio de información que se necesite. al marcar el bean con esta anotación Spring aporta servicios transversales como conversión de tipos de excepciones.



@Service: Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. su tarea fundamental es la de agregador.



@Controller: El último de los estereotipos que es el que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo.



Soporte ORM en Spring

Spring soporta administración de recursos, implementación data acces object (DAO) y control de transacción.

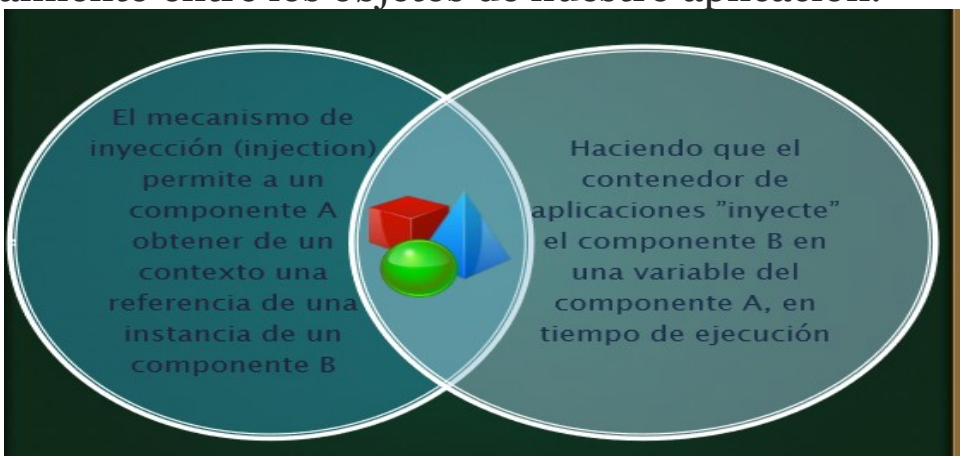
Beneficios de usar DAOs de Spring

- Fácil de testear.
  Excepciones de acceso a datos más entendibles.
- Mejor administración de recursos .
- Facil configuración de JPA e Hibernate.

Inyección de dependencias

Se establecen a través de los atributos de un componente Spring usando la anotación @Autowired. Existen tres variantes para implementar inyección de dependencias:

- Inyección mediante atributos es justamente suministrar a un objeto una referencia de otros que necesite según la relación, tiene que plasmarse mediante configuración **XML** o la anotación **@Autowired .** El objetivo es lograr un bajo acoplamiento entre los objetos de nuestro aplicación.



En general, las dependencias se establecen a través de los atributos de un componente Spring usando la anotación @Autowired. Existen tres variantes para implementar inyección de dependencias:

En general, las dependencias se establecen a través de los atributos de un componente Spring usando la anotación @Autowired. Existen tres variantes para implementar inyección de dependencias:

•Inyección mediante atributo (La mas recomendada).

```
public class InyeccionSetter {

    @Autowired
    private Dependencia miDependencia;

}
```

•Inyección de dependencia vía constructor.

```
public class InyeccionConstructor {

    private Dependencia miDependencia;

    @Autowired
    public InyeccionConstructor(Dependencia dep) {
        this.miDependencia = dep;
    }
}
```

•Inyección mediante método Setter.

```
public class InyeccionSetter {

    private Dependencia miDependencia;

    @Autowired
    public void setMiDependencia(Dependencia dep) {
        this.miDependencia = dep;
    }
}
```