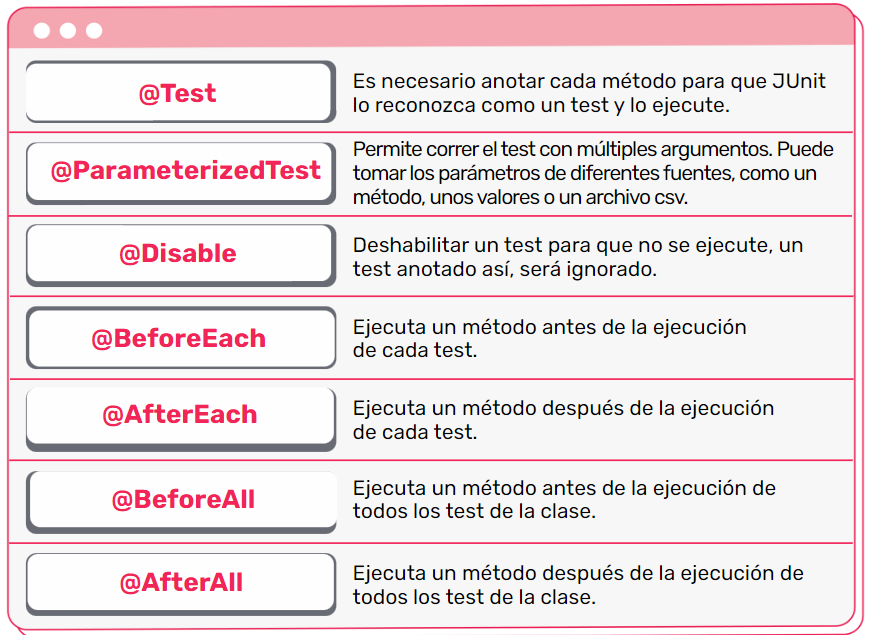
SEMANA 1

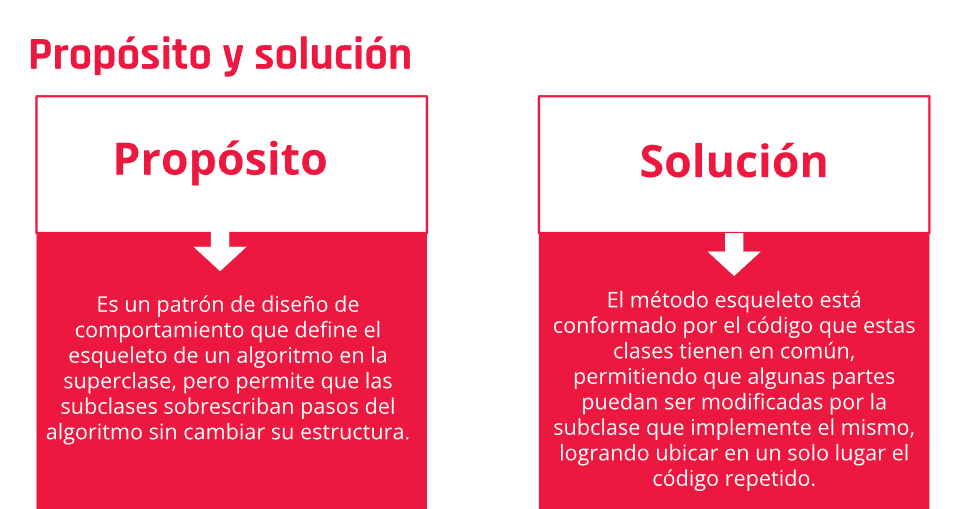
JUnit (http://junit.org) es el framework open source de testing para Java más utilizado. Nos permite escribir y ejecutar tests automatizados. Es soportado por todas las IDEs (Eclipse, IntelliJ IDEA), build tools (Maven, Gradle) y por frameworks como Spring.

Test en Junit:



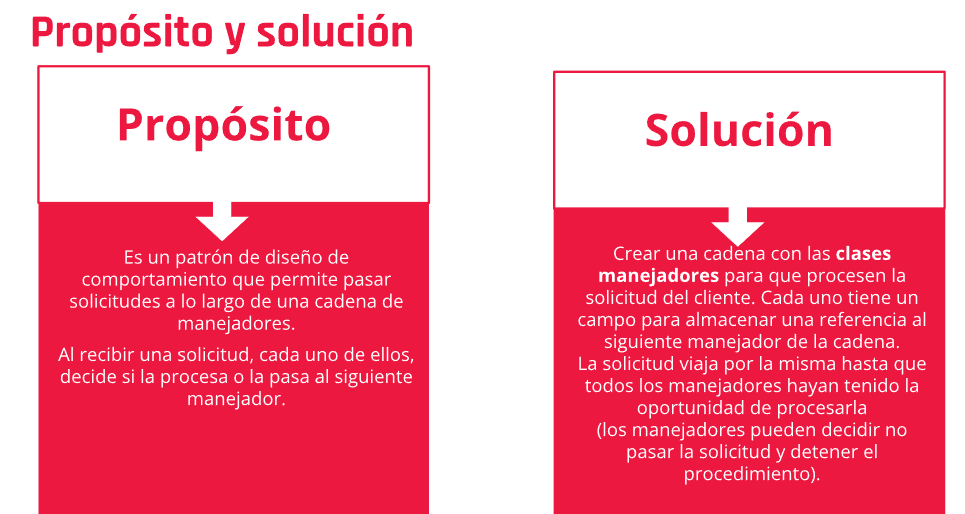
**PATRONES DE DISEÑO:**

PATRON TEMPLATE METHOD:



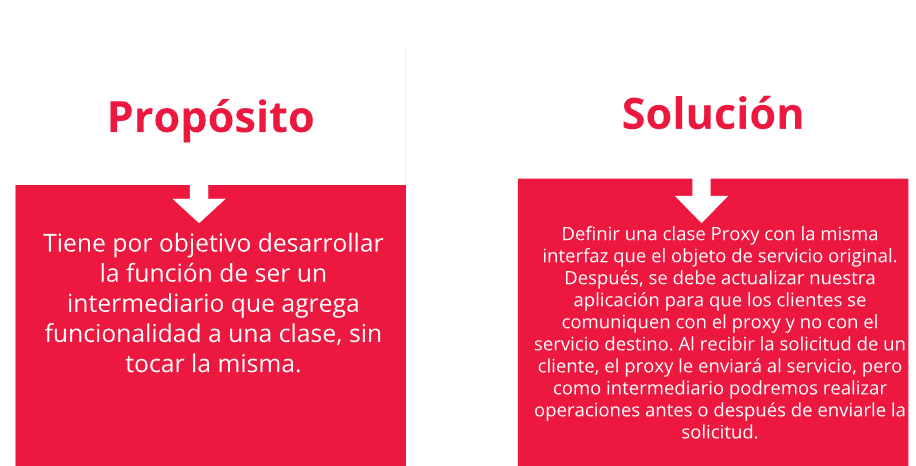
¿para qué nos sirve el patrón template method? Porque al eliminar el código repetido, nuestro código será más eficiente, legible y mantenible. Esto hará que sea más fácil de extender y mejorar.

PATRON CADENA DE RESPONSABILIDAD:



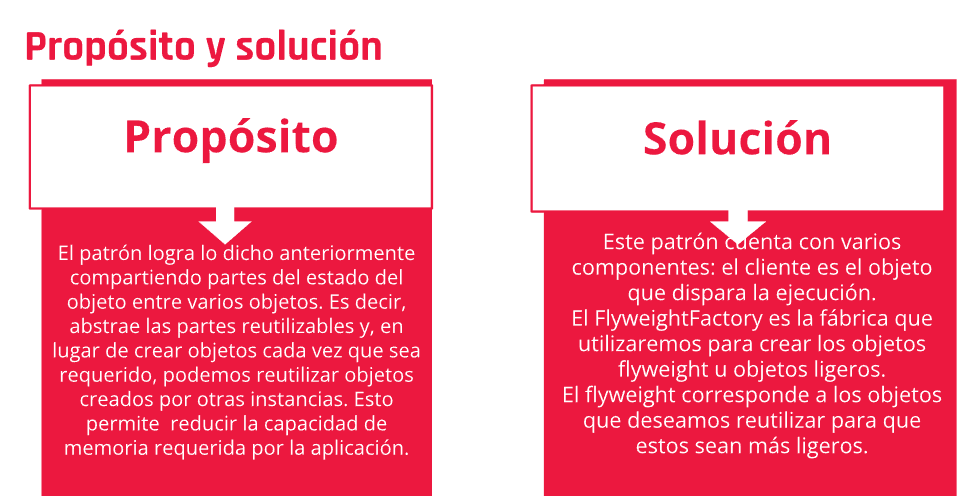
Entonces, podemos concluir que es beneficioso utilizar el patrón cadena de responsabilidad cuando esperamos que nuestra aplicación procese diferentes tipos de solicitudes y responda con diferentes resultados, pero no conocemos de antemano cuáles son estas solicitudes.

PATRON PROXY:



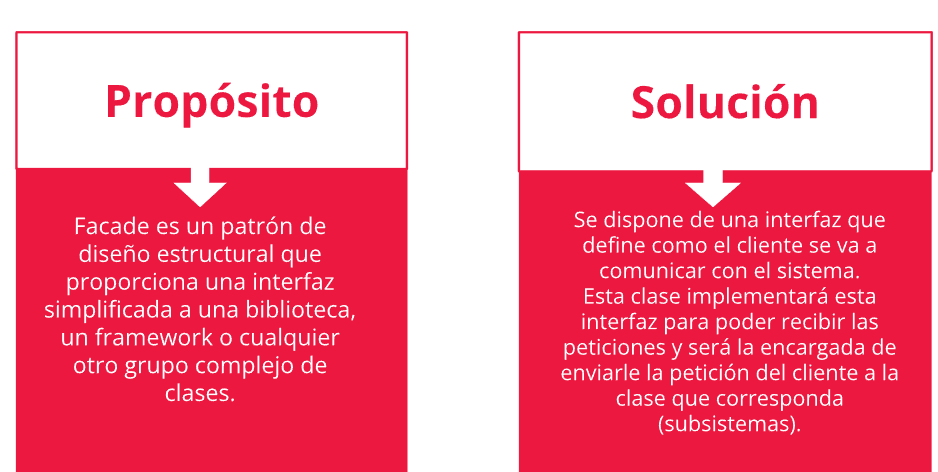
Entonces, podemos mencionar que es beneficioso utilizar el patrón proxy cuando queremos agregar funcionalidad adicional sin modificar los servicios actuales.

PATRON FLYWEIGHT:



Este patrón es utilizado cuando la optimización de los recursos es algo primordial, ya que elimina la redundancia de objetos con propiedades idénticas.

PATRON FACADE:



El patrón facade nos ayuda a reducir la complejidad de interactuar con un conjunto de subsistemas, actuando de intermediario entre el cliente y los subsistemas, permitiéndonos tener una única entrada, facilitando la comunicación entre estos.

SEMANA 2:

LOG4J: Es una librería desarrollada en Java por la **Apache Software Foundation** que permite a los desarrolladores elegir la salida y el nivel de granularidad de los mensajes o logs en tiempo de ejecución. En otras palabras, es utilizada para generar mensajes de **logging** de una forma limpia, sencilla, permitiendo filtrarlos por importancia y pudiendo configurar su **salida** tanto por **consola**, **fichero** u otras diferentes.

VENTAJA:

Permite tener un registro de lo que está pasando en nuestros sistemas, lo que nos posibilita entender mejor los errores.

DESVENTAJA:

La única desventaja es que a veces los archivos se hacen muy grandes y ocupan mucho espacio. Es por ello que debemos elegir bien qué tipo de información queremos almacenar

Por defecto Log4j tiene niveles de prioridad para los mensajes, entre ellos se encuentran:

* OFF: Este es el nivel de mínimo detalle, deshabilita todos los logs.
* FATAL: Se utiliza para mensajes críticos del sistema, generalmente después de guardar el mensaje, el programa se cierra.
* ERROR: Indica eventos de error que aún podrían permitir que la aplicación continúe ejecutándose.
* WARN: Se utiliza para mensajes de alerta sobre eventos.
* INFO: Se refiere a mensajes informativos que resaltan el progreso de la aplicación en un nivel aproximado.
* DEBUG: Designa los eventos informativos detallados más útiles para depurar una aplicación.
* TRACE: Se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.
* ALL: Es el nivel de máximo detalle, habilita todos los logs.

ACCESO A BASE DE DATOS

JDBC: Por las siglas de Java DataBase Conectivity, es un framework que consiste en múltiples interfaces y solo algunas clases de soporte. Esto se debe a que la idea detrás de JDBC es que cualquiera pueda crear su propia implementación del framework y adaptarla a sus necesidades. Dado que se trata de un conjunto de interfaces, cualquier código que interactúe con el framework no se verá afectado si se altera la implementación.  
  
Es por esto que JDBC define interfaces que **solo** declaran el comportamiento que debe llevarse a cabo para conectarse e interactuar con una base de datos. Así, nos encontramos con interfaces tales como: Connection (abstracción del comportamiento de una conexión), Statement (define el comportamiento para realizar sentencias contra una base de datos, sean queries y otras instrucciones), ResultSet (que abstrae el comportamiento para extraer resultados de las consultas), entre otras. Todas estas clases e interfaces están dentro del paquete java.sql.\*. Entonces, para interactuar con los diferentes motores de base de datos, debemos tener una implementación de estas y otras interfaces, es decir, una clase concreta que implemente cada interfaz.

H2: Es una base de datos open source escrita en Java que permite integrar aplicaciones en Java o ejecutarse en modo cliente-servidor. Principalmente, se puede configurar para que se ejecute como una base de datos en memoria. Entonces, los datos no persistirán en el disco, debido a que la base de datos no se utiliza para el desarrollo de producción, sino principalmente para el desarrollo y las pruebas.

SEMANA 4:

SPRING

¿Qué puede hacer Spring?

- Es un framework que permite hacer aplicaciones web rápidas, seguras y responsivas, además de conectarse a cualquier base de datos a través de tecnologías de ORM.

- Deploya fácilmente nuevas funcionalidades de manera independiente.

- La arquitectura asincrónica de Spring permite utilizar mejor los recursos informáticos.

- Al codificar para cualquier nube, posibilita escalar las aplicaciones.

- Permite armar aplicaciones serverless que reaccionen ante ciertos eventos de negocio.

- Posibilita armar aplicaciones que reaccionen ante ciertos eventos de negocio.

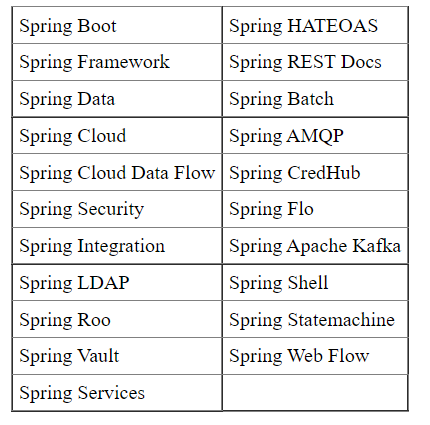
-Ofrece la posibilidad de automatizar flujos de procesos en línea y desatendidos.

¿Qué es Spring Platform?

Spring Platform es un conjunto de proyectos open source desarrollados en Java con el objetivo de agilizar el desarrollo de aplicaciones.

Cuenta con gran variedad de herramientas que nos facilitan el trabajo desde el acceso a datos, infraestructura, creación de aplicaciones web, microservicios, etc.

Ellos son: (https://spring.io/projects)



¿Qué es Spring Framework?

Como vimos anteriormente se trata de un framework para el desarrollo de aplicaciones y contenedor de inversión de control.

A su vez, puede ser usado en cualquier aplicación desarrollada en Java.

¿Qué es Spring Boot?

Es una extensión de Spring framework que permite la creación fácil y rápida de aplicaciones web listas para producción con el concepto de just run (solo ejecutar).

Requiere una mínima configuración y se complementa con muchos proyectos de Spring Platform y librerías de terceros.

Como conclusión podemos mencionar que reduce en gran medida el tiempo de desarrollo y aumenta la productividad.

CARACTERISTICAS DE SPRING BOOT

- Trae el servidor [Tomcat](http://tomcat.apache.org/" \t "_blank) ya incorporado sin necesidad de instalación previa, también es posible usar Jetty y Undertow.

- De manera predeterminada, el servidor incorporado escucha las solicitudes HTTP en el puerto 8080. Esto quiere decir que, cuando arrancamos el servidor, accedemos a la URL<http://localhost:8080/>**para ver nuestra aplicación. Podemos configurar otro puerto y otras propiedades, diferentes a las que se setean por defecto. Para esto, debemos colocarlas directamente en el archivo de configuración: application.properties**.

Los iniciadores (starters) se usan para limitar la cantidad de configuración manual de las dependencias que debemos hacer. Básicamente son dependencias de Maven que se registran en el archivo POM.xml

Al instalar los starters, Spring Boot se encargará de hacer encajar las dependencias de tal forma que estas puedan utilizarse de forma natural en nuestra solución con sus versiones correspondientes. Comienzan con**spring-boot-starter-**\*, donde \* es el tipo de aplicación que se quiere desarrollar.

**Los más populares**

**spring-boot-starter-web**se utiliza para desarrollar servicios web de RESTful con Spring MVC y Tomcat como el contenedor de aplicaciones incorporado.

**spring-boot-starter-jdbc** se utiliza para el agrupamiento de conexiones JDBC. Se basa en la implementación del grupo de conexiones JDBC de Tomcat.

PATRON MVC:

El patrón MVC es un patrón de diseño que nos permite organizar el código y los archivos en función de su responsabilidad, en otras palabras, en función de la tarea que debe realizar. 

VISTA:

La **vista** le muestra al usuario los datos que recibió.

Conforma la interfaz gráfica de la aplicación y contiene todos los elementos que son visibles al usuario. A través de ella, el usuario interactúa enviando y solicitando información al servidor. Sus responsabilidades son definir la apariencia de los datos y mostrarlos en pantalla. Las vistas no se comunican de forma directa con los modelos.

CONTROLADOR:

El **controlador** recibe la petición, valida que todos los datos sean correctos y le solicita al **modelo**el detalle del producto 20.

Conforma la capa intermedia entre las vistas y los modelos. Sus responsabilidades son procesar los datos que recibe de los modelos y elegir la vista correspondiente en función de aquellos datos. Tiene relación directa con las vistas y con los modelos y es un componente fundamental dentro del flujo del patrón.

MODELO:

El **modelo**busca la información solicitada y se la envía al **controlador**.

Conforma y contiene la lógica de la aplicación. Sus responsabilidades son conectarse con la base de datos, realizar consultas y administrar lo que se conoce como la lógica de negocio. Los modelos no se comunican de forma directa con las vistas.

