Inyección de dependencias

Creación de objetos

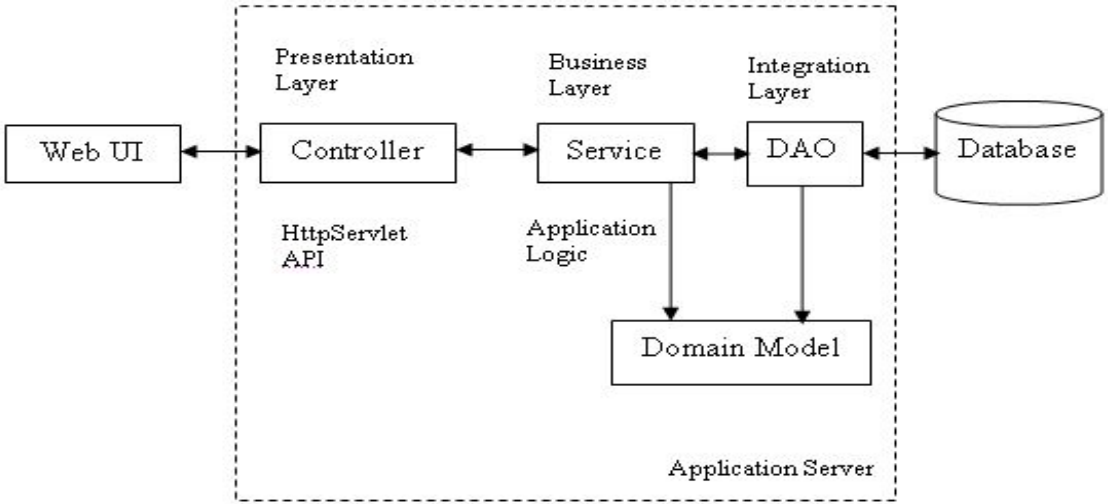
Operador “new”.

Patrones de diseño : Factory, Prototype, Builder, Singleton, etc.

Inconvenientes:

- Sobrecarga de constructores.
- Lógica de compleja de creación en constructores.
- Proliferación de Objetos auxiliares para creación de objetos.
- Multiplicación de código.
- Lógica compleja.

Modelo de capas



Los 5 principios SOLID de diseño de aplicaciones de software son:

- S: Single Responsibility Principle.
- O: Open/Closed Principle.
- L: Liskov Substitution Principle, declara que una subclase debe ser sustituible por su superclase
- I: se refiere al Principio de Segregación de Interfaces (Interface Segregation Principle), utilizar interfaces con propósito específicos, o sea que tengan responsabilidades únicas.
- D: Dependency Inversion Principle, para conseguirlo se hace uso de la inyección de dependencias.

Definiciones

Inyección de Dependencia (DI): es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar (de ahí el concepto de *dependencia*). Nuestras clases no crean los objetos que necesitan, sino que se los suministra otra clase 'contenedora' que inyectará la implementación deseada a nuestro contrato.

Inversión de Control (IoC): Delegar la creación de objetos a un framework, invirtiendo el la lógica de creación de objetos. Para esto se basa en la introspección (en Java llamado Reflection) el cual es un procedimiento por el cual leemos metadatos de la App que nos permita entender cómo funciona, los metadatos pueden estar expresados principalmente de dos formas, la primera es mediante archivos de configuración como XML o con metadados directamente definidos sobre las clases de nuestro programa.

Inyección de dependencias (DI)

Responsabilidades del patrón de DI:

- Crear Objetos
- Identificar las clases que usan estos objetos
- Proveer los objetos creados las clases

Resuelve

- Acoplamientos entre capas
- Lógica compleja al instanciar objetos.
- Reutilización del código.

CDI (Contexts and Dependency Injection)

CDI define un poderoso conjunto de servicios complementarios que ayudan a mejorar la estructura del código de la aplicación.

- Un ciclo de vida bien definido para objetos con estado vinculados a contextos de ciclo de vida.
- Un mecanismo de inyección de dependencias sofisticado y seguro.
- La capacidad de decorar objetos inyectados.
- La capacidad de asociar interceptores a objetos a través de enlaces de interceptores de tipos seguros.
- Un modelo de notificación de eventos.
- Un contexto de conversación web además de los tres contextos web estándar definidos por la especificación Java Servlets.

Definiendo Beans

En CDI, un Bean es una clase administrada por el contenedor CDI, es decir, una clase cuyas instancias son creadas, inyectadas y gestionadas automáticamente por CDI según las reglas de inyección de dependencias y contexto.

El contexto en CDI define cuánto vive un objeto (bean) y quién es responsable de manejar su ciclo de vida (creación, almacenamiento, destrucción).

Definiendo Beans

@ApplicationScoped

Vive mientras la aplicación esté en ejecución.
Instancia única compartida entre todos los usuarios y requests.
Útil para recursos compartidos como caches, config global o servicios singleton.

@SessionScoped

Vive mientras dure la sesión del usuario (HTTP Session).
Útil para guardar información del usuario logueado o estado entre páginas.

@RequestScoped

Vive durante una única solicitud HTTP.
Cada request genera una nueva instancia.
Ideal para manejar datos temporales del request.

@Dependent

El bean vive tanto como el objeto que lo inyecta.
No tiene ciclo de vida propio.
Es el scope por defecto.
Ideal para beans sin estado o temporales.

Ciclo de vida de los Beans

En cada bean se puede ejecutar código en momentos clave del ciclo de vida del bean: después de que se crea, y antes de que se destruye.

@PostConstruct
public void postConstruct() {

//Acciones después de construir la instancia del objeto
}

@PreDestroy
public void preDestroy() {

//Acciones antes de destruir la instancia del objeto.

}

Método generador de Beans.

@Produces y @Disposes son herramientas que usamos cuando CDI no puede instanciar directamente lo que necesitamos, o cuando queremos tener control fino sobre la creación y destrucción de dependencias. El scope del bean producido lo define la anotación de scope acompañe el método @Produces(@Dependent por defecto).

@Produce
Este método o campo produce una instancia que puede inyectarse en otros lados.
@Disposes
Libera recursos producidos por @Produces

Injectando Beans

Para indicarle a CDI que debe setear un Bean utilizamos la anotación @Inject.

```
@Inject    private UserRepository userRepository;

@Inject    public UsuarioResource(UsuarioService usuarioService,UppercaseService uppercaseService )
@Inject    public void setDao(UsuarioDao dao)
```

Ante un conflicto de nombres se debe usar la anotación@Named, para explicitar el bean a inyectar.

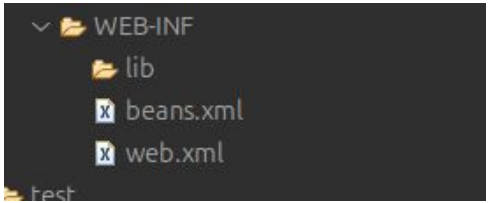
@Inject @Named("adminUser") private User user;

Configuración

1) Agregar dependencias al pom.xml

```
    <dependency>
        <groupId>org.glassfish.jersey.ext.cdi</groupId>
        <artifactId>jersey-cdi1x</artifactId>
        <version>${jersey.version}</version>
    </dependency>

    <!-- CDI Implementation (Weld) -->
    <dependency>
        <groupId>org.jboss.weld.servlet</groupId>
        <artifactId>weld-servlet-core</artifactId>
        <version>${weld.version}</version>S
    </dependency>
```



2)Agregar beans.xml en WEB-INF

3) Agregar el interceptor en web.xml

```
<!-- CDI Configuration -->
  <listener>
      <listener-class>org.jboss.weld.environment.servlet.Listener</listener-class>
  </listener>
```

Referencias

1. https://martinfowler.com/articles/injection.html
2. https://www.martinfowler.com/articles/dipInTheWild.html
3. https://jakarta.ee/specifications/cdi/4.1/jakarta-cdi-spec-4.1.pdf
4. https://www.cdi-spec.org/
5. https://www.ibm.com/docs/es/was/8.5.5?topic=cdi-contexts-dependency-injection