Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

PECL 3 – ARTEFACTO 12

Aplicación de patrones

Ing. Software

Laboratorio Martes 12:00 – 14:00

Grado en Ingeniería Informática – Curso 2018/2019

Eduardo Graván Serrano – 03212337L

Marcos Barranquero Fernández – 51129104N

Sonia Rodríguez-Peral Bustos - 54302528B

Adrián Montesinos González – 51139629A

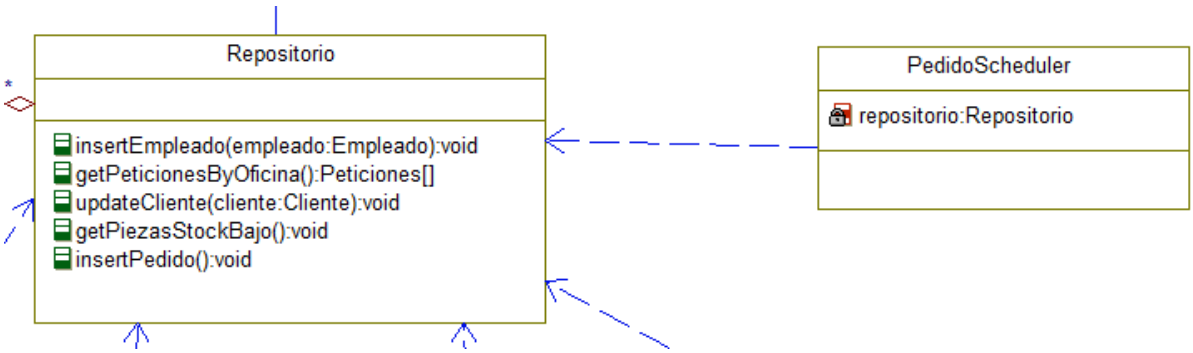
Alejandro Caballero Platas – 50891258D

INTRODUCCIÓN

En este informe se detallan los patrones de software sobre los que nos apoyamos para la elaboración de nuestro diseño del software.

Se da una explicación sobre los patrones Modelo-Vista-Controlador (MVC), Repositorio e Inyección de Dependencias (DI). Cada apartado justifica cómo se integra esto con el framework de aplicación Spring que hemos elegido para estructurar el proyecto y cómo nos ayuda esto a tener un software más fácil de mantener.

Cabe destacar que los tres patrones elegidos se complementan bastante bien: el repositorio nos permite centralizar todo el modelo de MVC, mientras que la inyección de dependencias de Spring nos desacopla las vistas, controladores y el repositorio hasta tal punto que las relaciones entre estas partes se convierten en relaciones de dependencia, no de agregación.



Ejemplo de las relaciones resultantes gracias a estos patrones.

PATRÓN MODELO VISTA – CONTROLADOR

El patrón Modelo-Vista-Controlador (MVC) es ampliamente usado en el desarrollo de aplicaciones web/de interfaz gráfica. MVC nos permite aumentar el desacoplamiento de sus partes pero manteniendo claras las responsabilidades de cada parte del código, siendo este un gran atractivo a la hora de desarrollar software en equipo y mantenerlo. Este patrón divide el código relevante en tres responsabilidades:

- Modelo: El modelo se corresponde con la parte del modelo de dominio relevante a la vista. Pueden emplearse directamente las clases del modelo de dominio de la aplicación o puede construirse alguna fachada encima suya para encapsular detalles del modelo que sean irrelevantes a la vista. En este segundo caso, el modelo podría ser incluso único a una sola vista.
- Vista: La vista se corresponde con el código que refleja la interfaz que verá el usuario. La vista es un concepto algo abstracto, ya que el código responsable de la interfaz no es necesariamente el código que ejecuta la propia interfaz: por ejemplo, la vista en un servidor web es la respuesta HTTP con la página web resultante. Esta vista es un elemento sin estado propio, simplemente recibe su modelo correspondiente y emite la interfaz como salida. Según la plataforma, puede ser responsabilidad de la vista asegurarse de que las acciones que el usuario realice sobre esta interfaz sean dirigidas al controlador.
- Controlador: El controlador desacopla la vista de su modelo. Este controlador construye las clases del modelo correspondientes a la vista, solicita a la vista que emita su interfaz a partir del modelo, y maneja todas las acciones de usuario que modifiquen dicho modelo.

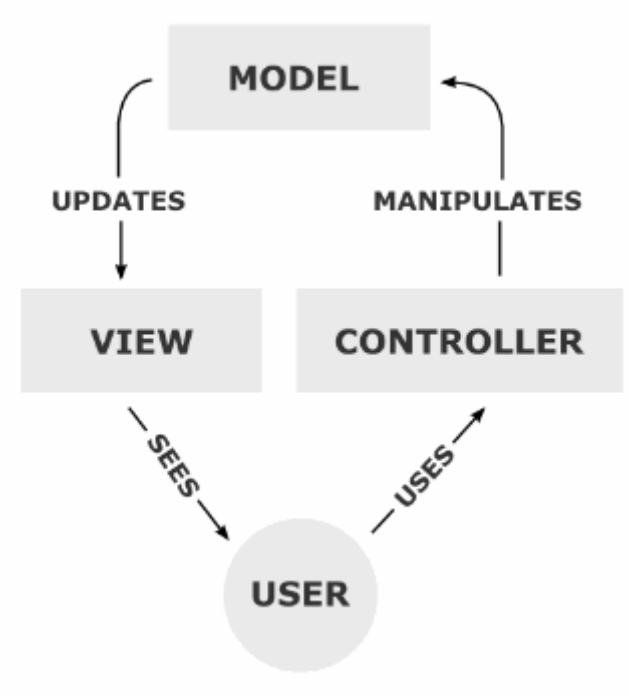


Diagrama del ciclo de ejecución de una aplicación MVC.

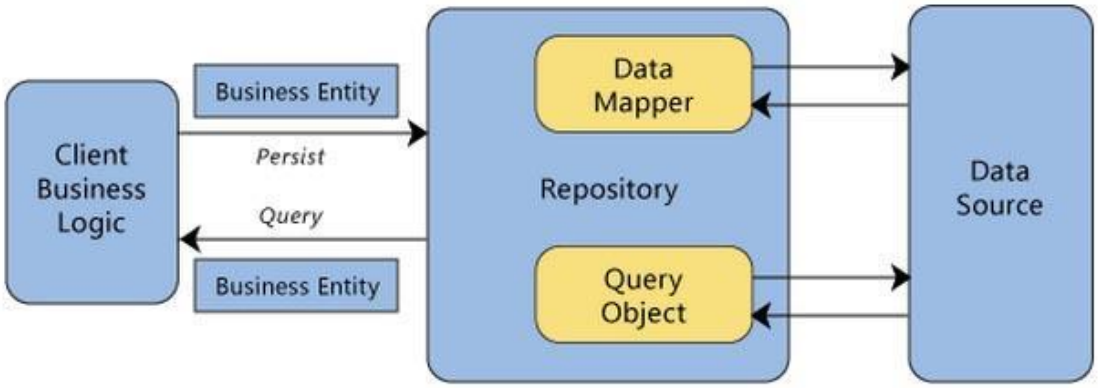
En nuestra aplicación, hemos optado por usar la proyección objeto-relacional de la base de datos como modelo de todas las vistas, sin ninguna abstracción extra, ya que las clases de esta capa de persistencia ya son suficientemente abstractas como para trabajar con ellas en nuestras interfaces. Además, la capa de persistencia y las consultas del repositorio de datos nos aseguran que hacemos los accesos de base de datos mínimos necesarios para implementar las vistas y los controladores.

Las vistas se corresponden una por cada menú de la aplicación, recibiendo estas una instancia de la clase del modelo que le interesa, y accediendo al resto del modelo mediante los métodos de dicha instancia. Todos los eventos se enlazan mediante callbacks a métodos en su controlador.

Cada controlador obtiene el modelo correspondiente a mostrar mediante el repositorio e activa la vista con este modelo. El controlador además tiene métodos para las interacciones que pueda realizar el usuario desde la interfaz, siendo este el único que modifica el modelo de dominio y actualiza los cambios de vuelta a la base de datos a través del repositorio.

PATRÓN REPOSITORIO

El patrón Repositorio es un patrón muy empleado para implementar el modelo de dominio de la aplicación sobre una capa de persistencia. El repositorio abstrae el acceso al origen de datos de la aplicación ofreciendo métodos para realizar consultas y actualizaciones, y devolviendo instancias de clases que reflejan el modelo de dominio. Estas clases suelen ser parte de la proyección objeto-relacional de la capa de persistencia, funcionando entonces el repositorio como la "raíz" de esta capa por la cual se accede al resto del dominio. Con esto, centralizamos el acceso al modelo de dominio desde los controladores a la vez que los desacoplamos de la API de la proyección objeto-relacional.



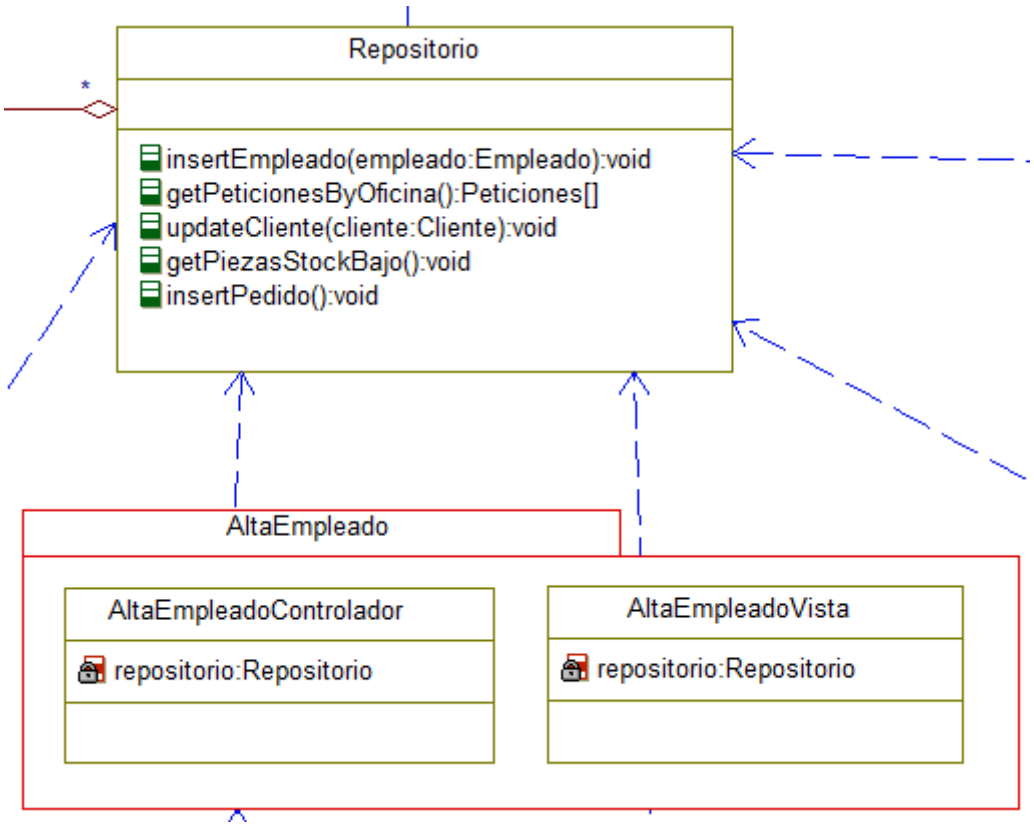
El patrón repositorio abstrae y centraliza el acceso a la capa de persistencia, desacoplándolo del resto de la aplicación.

Para facilitar el mantenimiento, se suele definir un repositorio por cada entidad que se quiera emplear como raíz, pero consideramos que esto habría complicado demasiado el diagrama de clases de diseño de un proyecto simple como este.

Para este proyecto hemos empleado creado un repositorio usando los patrones del framework Spring y la capa de persistencia de Hibernate. Este nos implementa automáticamente una instancia del repositorio solo con añadir anotaciones Java a la interfaz que lo define y emplea automáticamente la proyección objeto-relacional de Hibernate que tenemos definida.

PATRÓN INYECCIÓN DE DEPENDENCIAS (DI)

El patrón Inyección de Dependencias (*Dependency Injection,* DI) es un patrón utilizado para desacoplar la creación de objetos cliente y las implementaciones de los servicios de los que estos dependen. La inyección suele ser realizada por un framework de aplicaciones, en nuestro caso Spring. El framework de aplicación detecta qué interfaces servicio necesita el objeto cliente al ser creado y le inyecta al cliente las instancias de estas interfaces. Esto pone en el framework de aplicación la responsabilidad de seleccionar las implementaciones y manejar su ciclo de vida, no en los objetos cliente. Además, esto facilita la realización de pruebas integración con mock ups de los servicios, ya que el framework puede inyectar el mismo mock up a través de todos los objetos a probar simultáneamente.



Bajo inyección de dependencias, el ciclo de vida de nuestro repositorio lo maneja el framework de aplicación Spring.

El framework Spring nos ofrece una API para la inyección de dependencias en nuestras vistas y controladores de MVC. El servicio que necesitamos incondicionalmente a través de nuestra aplicación es el objeto Repositorio que nos permite interactuar con la capa de persistencia. Cuando se crea una instancia de una vista o de un controlador, Spring inyecta la instancia del Repositorio en un campo privado de nuestra clase que le indicamos mediante metadatos (anotaciones de Java). De este modo tenemos acceso al mismo repositorio desde cualquier vista y controlador sin necesidad de pasar los mismos parámetros a través de toda la aplicación.