**Propuesta de un framework para aplicaciones Java / Java EE que proporcione la máxima independencia entre las capas**

Francisco Javier Gil Gala. [UO230948@uniovi.es](mailto:UO230948@uniovi.es). Universidad de Oviedo.

V2: Se replantea por completo toda el framework así como su funcionalidad. Se acota la aplicabilidad del famework para aplicaciones Java / Java EE, especialmente aplicaciones ligeras. Se incorporan además generadores automáticos de código y de paquetes así como una nueva forma de gestión de la persistencia.

Sobre esta nueva versión se presentan dos proyectos que se presentan como ejemplo para las dos implementaciones del framework: la basada en JDBC y la basada en JPA.

1. Introducción

La siguiente propuesta de arquitectura de aplicaciones esta basada en el uso de un modelo definido en 3 capas: lógica, persistencia y presentación. Este diseño intenta primar la organización de la aplicación en varios paquetes permitiendo, ente otras cosas, una gran escalabilidad en las aplicaciones construidas. Esta arquitectura no tiene solo un ámbito educativo o de investigación si no que pretende ser un completo framework para aplicaciones Java.

La arquitectura define una serie de clases e interfaces con objeto de estructurar toda la aplicación. Estas clases e interfaces no deben ser modificadas por el programador, ya que el comportamiento de dichas clases se encuetra parametrizado a través de un archivo de propiedades o se puede modificar extendiendo la funcionalidad de las clases.

No estamos ante nada nuevo, simplemente es una arquitectura basada en el modelo de capas n-capas pero aplicando los principios SOLID y algunos de los patrones de diseño propuestos en [1] y [2]

Además de incorporar todo el marco de trabajo también incopora clases de utilidad para generar bases de datos embebidas en la aplicación o para generar la estructura de paquetes y clases de ejemplo.

1. Requisitos y librerias necesarias
2. Diseño propuesto

Respecto a la versión 1, en esta revisión del framework no se impone una estructura de paquetes si no que será el usuario el encargado de generarla a partir de las clases del paquete generadores. Tambien existe la opción de generar versiones demo del framework.

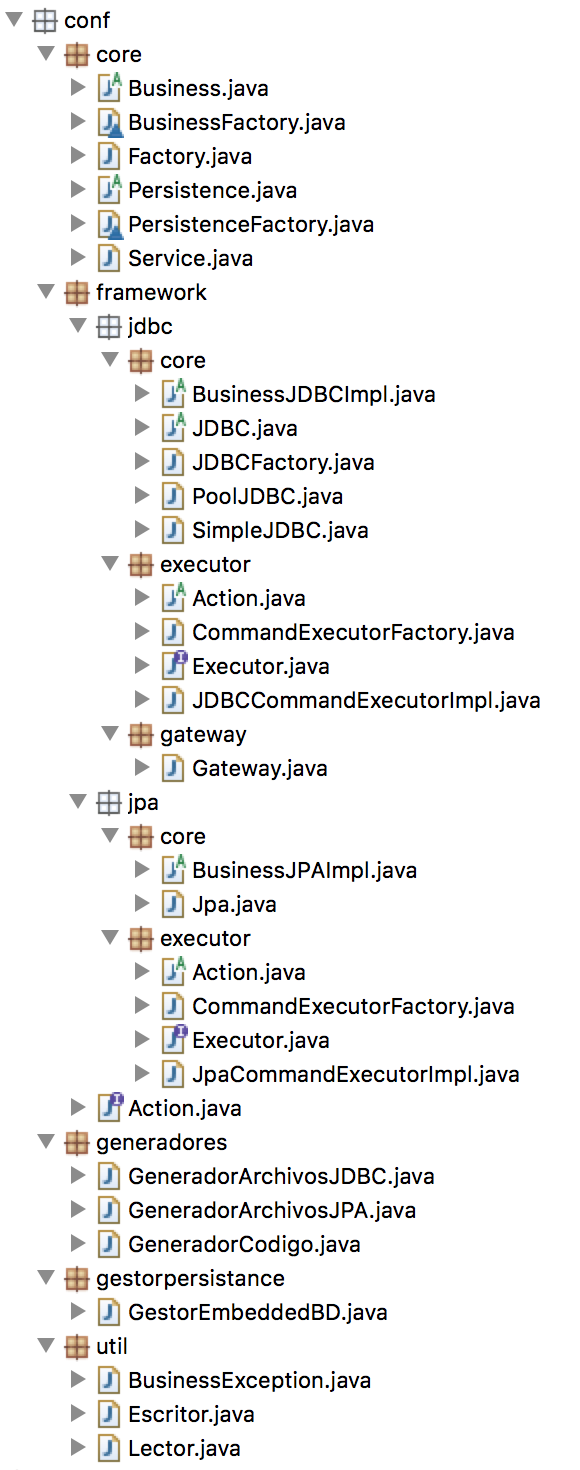


Ilustración 1

El corazón de esta arquitectura se articula en el paquete conf. Como podemos observar en la ilustración 1 principalmente tendremos dos bloques estructurados en dos paquetes: el paquete core y el paquete framework, además de los paquetes util, gestorpersistance y generadores que sirven de apoyo a los otros dos paquetes.

* 1. Paquete core

Este paquete apenas a sufrido cambios respecto a la versión 1. En la ilustración 2 se muestra la relación entre las distintas clases:

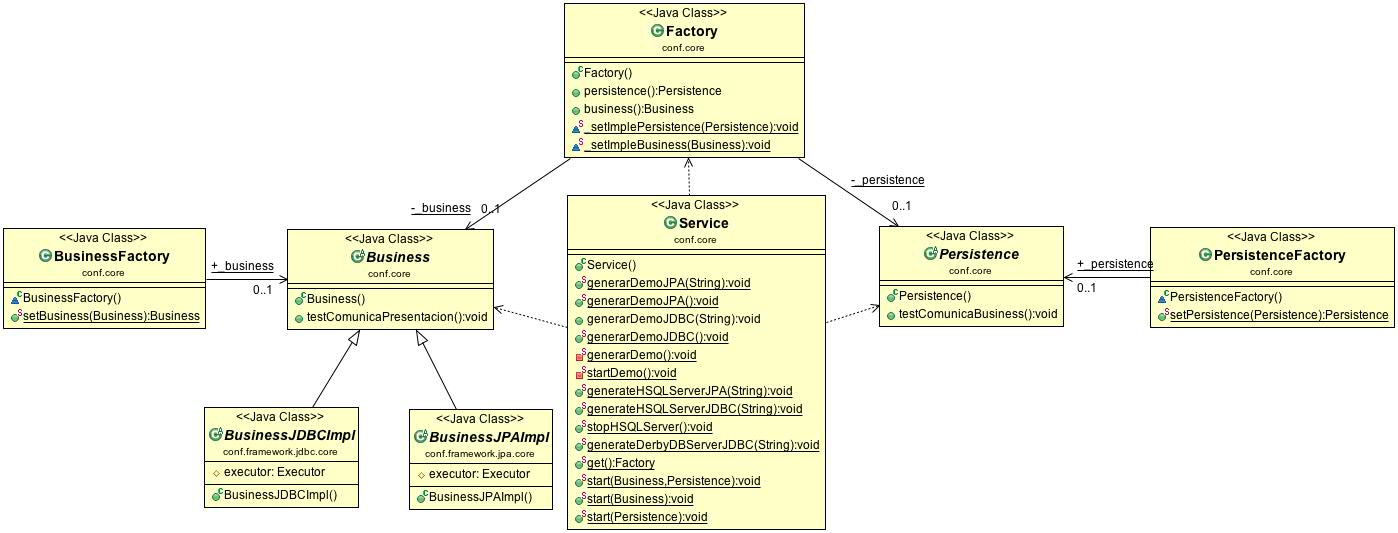


Ilustración 2

Se observa como la clase de entrada será la clase Service que será la encargada de proporcionar todos los servicios básicos del framework:

* Iniciar el framework.
* Proporcionar y gestionar el acceso entre las distintas capas.
* Iniciar y parar una base de datos embebida.
* Generar clases de ejemplo para JPA o para JDBC.

Las demás clases que sirven de apoyo a la clase Service son:

* Factory será una factoria de factorias encargada de crear las factorias de persistencia y de negocio.
* BusinessFactory y PersistenceFactory serán las encargadas de crear las implementaciones concretas del usuario de Business y Persistence.
* Business y Persistence serán las clases que el usuario deberá extender para utilizar el framework, dependiendo de como se extienda (por BusinessJDBCImpl o BusinessJPAImpl) el framework utilizará las clases de JPA o las de JDBC.
  1. Paquete framework
     1. Paquete jpa

El paquete executor es la implementación concreta de JPA sin especificar aun que implementación concreta se esta utilizando. La idea de este conjunto de código es mantener la máxima independencia entre la implementación concreta de JPA y la capa business.

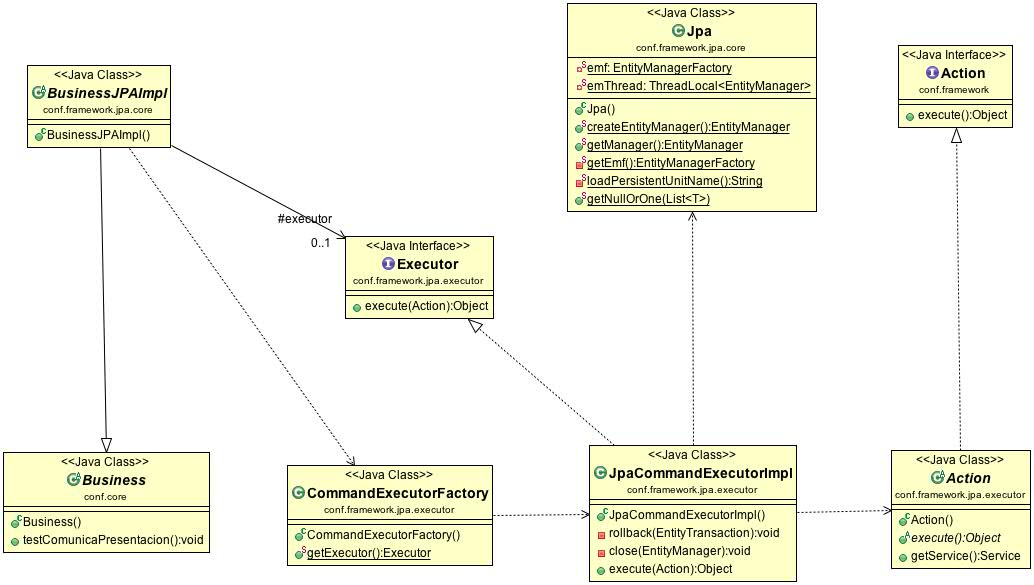


Ilustración 3

Como se puede observar en la ilustración 3 JPACommandExecutorImpl será una implementación de la interfaz Executor que recibirá una implementación concreta de la interfaz Action. Como se verá en los siguientes apartados cada una de las operaciones de negocio estará representada por una implementación de Action que será pasada por parámetro a la implementación concreta de Executor.

Ilustración 4

**private** Executor executor = CommandExecutorFactory.*getExecutor*();

Como se puede observar en la ilustración 4 en algun lugar de la capa business deberá existir una llamada a la factoria de Executor, en este caso CommandExecutorFactory, que devuelva la implementación concreta de Executor, en este caso JPACommandExecutorImpl. El framework ya lo proprociona a través de la clase BusinessJPAImpl que deberá ser extendida por la clase

Por otro lado comentar que, aunque aparezca en el diseño final, la clase JPA unicamente se encarga en gran medida de buscar la EntityManager y de crearla a partir de una la unidad de persistencia, por lo que su diseño no tiene gran interés en el diseño de esta arquitectura.

* + 1. Paquete jdbc

POR HACER

* 1. Otros aspectos relevantes
     1. Generadores
     2. Gestor de persistencia
     3. Paquete util
  2. Ejemplos

Se basa en un simple ejemplo para crear y listar libros mediante una aplicación de consola.

* + 1. Ejemplo JPA

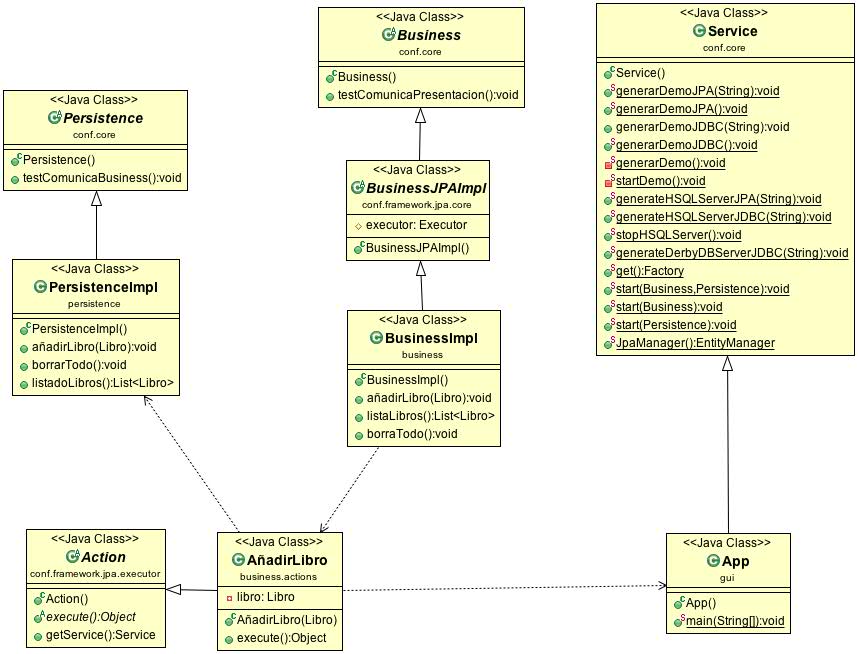


Ilustración 5

En la ilustración 5 se observa el funcionamiento del framework. Unicamente se debe extender las clases especificas del framework en las clases concretas del usuario, es decir, la clase BusinessImpl, creada por el usuario, extenderá la clase BusinessJPAImpl proporcionada por el framework. De forma similar pasará con las clases concretas del usuario: PersistenceImpl y App, que extenderán de Persistence y de Service respectivamente.

El usuario podrá acceder de esta forma a los servicios de la clase Service a través de su propia implementación, la clase App, y si tiene que modificarlos podrá modificarlos libremente.

En la ilustración 6 se muestra la distribución de paquetes del ejemplo.

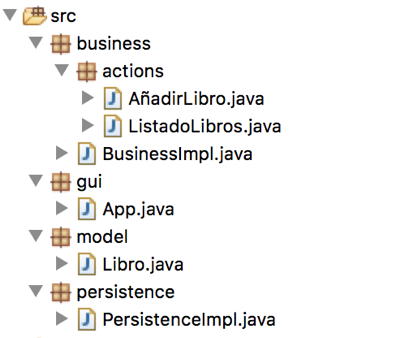


Ilustración 6

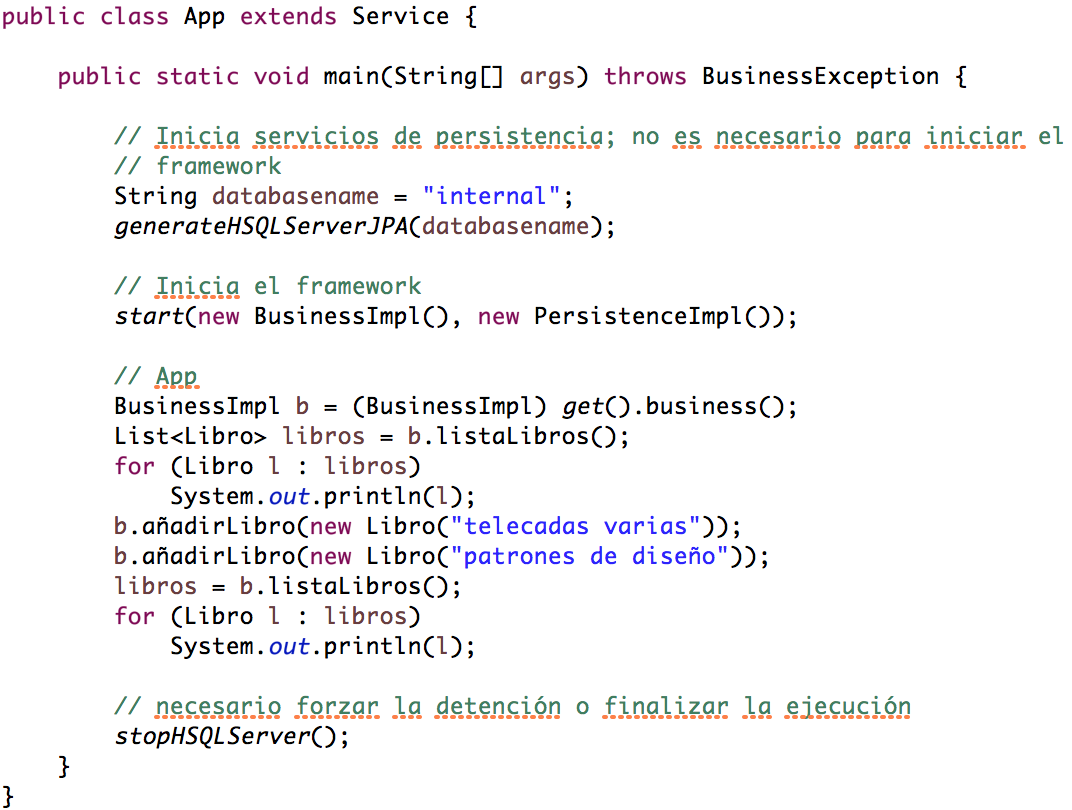


Ilustración 7

Como se observa en la ilustración X, la clase principal de la aplicación debe extender de la clase Service. Para la inicialización del framework es necesario llamar al metodo start(…) y pasarle las implementaciones concretas de las clases Business y Persistance del usuario. Será dependiendo de la implementación que estas clases usen (de que clase extiendan) la que determine como funcionará el framework, si estará basado en JPA o en JDBC.

Cada una de las operaciones de negocio asociadas com servicio deberán ser implementada en una clase que extienda de Action, como se puede observar en la ilustración X.

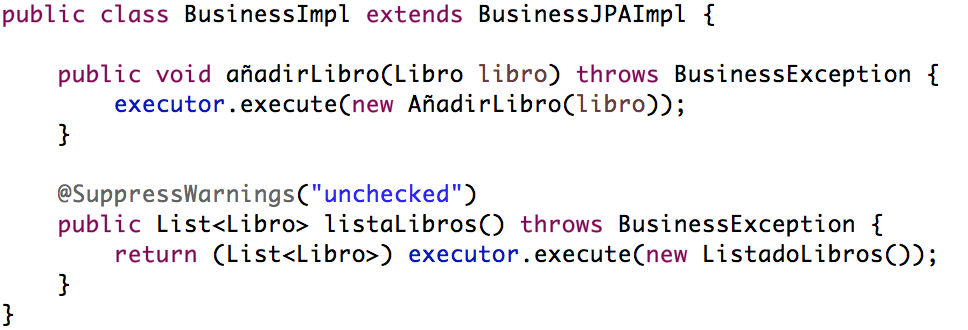


Ilustración 8

En este ejemplo la operacion de añadir libro se implementará mediante la creación de una clase AñadirLibro que extienda de la clase Action del paquete conf.framework.jpa.executor

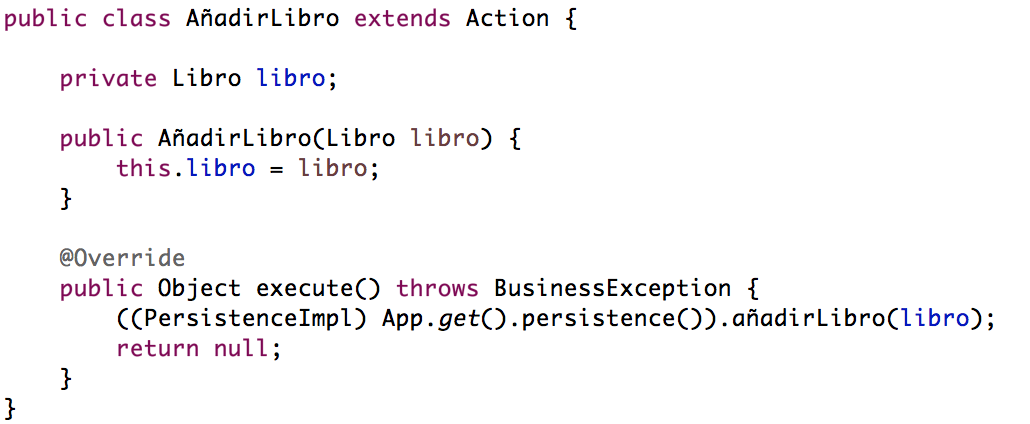


Ilustración 9

La forma de realizar las operaciones de persistencia no se especifica, unicamente se especifica que debe existir una clase que extienda de Persistence, en este caso PersistenceImpl, que implemente las operaciones que se hagan necesarias desde la capa de negocio.

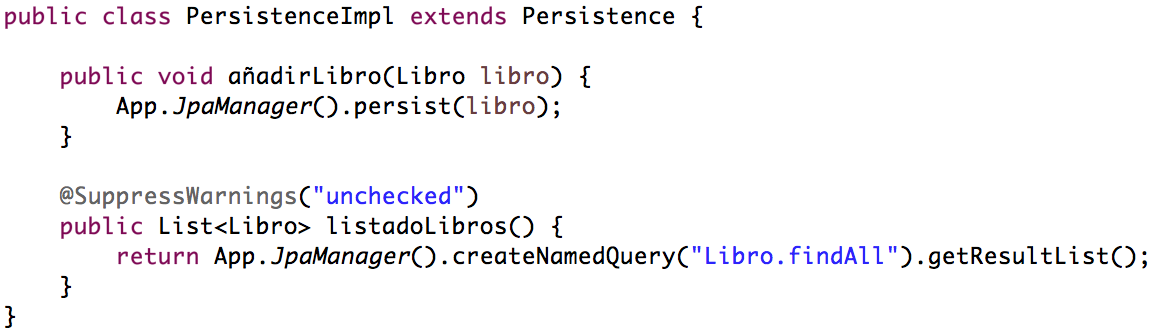


Ilustración 10

Será habitual ver código como el que se muestra en la ilustración 6 donde se accede al controlador de JPA a través de la clase cocnreta del usuario App que está extendiendo la funcionalidad de la clase Server del framework.

* + 1. Ejemplo JDBC

POR HACER

Referencias bibliográficas

[1] Design Patterns, Gang of Four

[2] Publicaciones de Martin Fowler