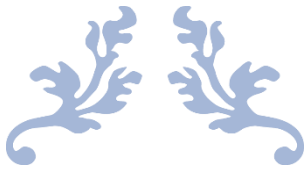


TRABAJO DE PRÁCTICAS

Desarrollo del Software

16 DE AGOSTO DE 2020

Ernesto Martínez del Pino
UNIVERSIDAD DE GRANADA

Contenido

Ejercicio 1: Patrón de “Factoria Abstracta” y patrón “Método Factoría” en Java.

Método Factoría

El patrón creacional “Método Factoría” o “Constructor Virtual” sirve para crear objetos dentro de un entorno de polimorfismo en base a un método plantilla que implementa un algoritmo. Dando la posibilidad de implementar distintas estrategias de fabricación de objetos. Mediante una superclase se muestra el comportamiento general de creación de objetos, cada una de las subclases implementará el algoritmo conveniente que devuelva el objeto deseado en función de unos parámetros.

Imaginemos que tenemos una interfaz de lectura de imágenes con un método leer imagen que nos devuelve un objeto con la imagen descodificada. Cada uno de los posibles formatos de imagen sería una subclase que implementa la interfaz de lectura de imágenes y por lo tanto, el algoritmo pertinente sobrecargando el método leer imagen devolviendo una subclase de imagen.

Factoría Abstracta

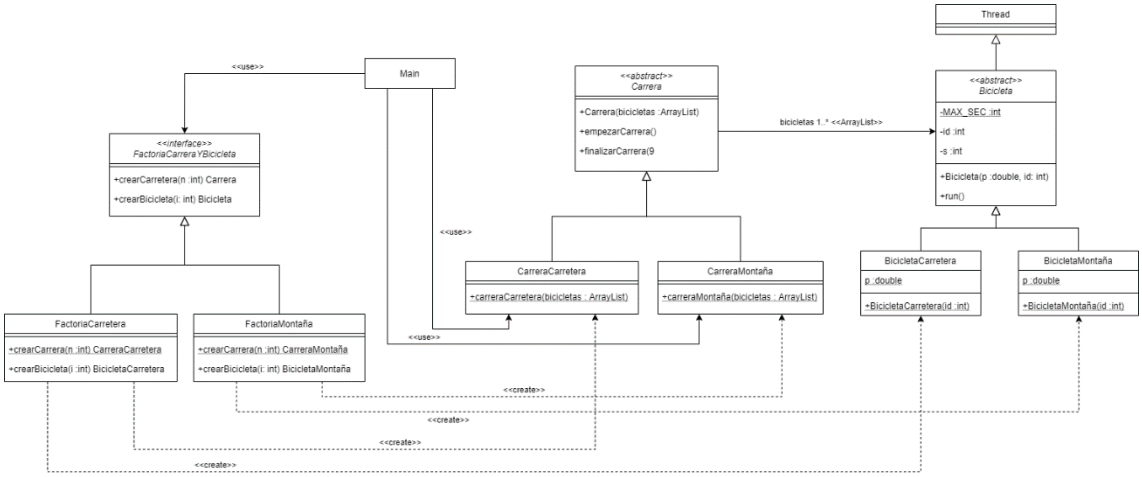
El patrón creacional “Factoría Abstracta” permite crear diferentes tipos de objetos relacionados entre sí, o dicho de otro modo familias de objetos. Una “Factoría Abstracta” no solo lo tiene un método de abstracción.

Un ejemplo claro de aplicación es el desarrollo de un framework multiplataforma, donde dependiendo del dispositivo los componentes a mostrar tendrán unas propiedades u otras. Imaginemos un cuadro de texto y una imagen con sus correspondientes polimorfismos para cada uno de los dispositivos. La interfaz de “Factoría Abstracta” tendría tantas implementaciones como dispositivos haya y tantos métodos como componentes tenga el framework. De esta forma el desarrollador podrá seleccionar la factoría asociada al dispositivo

y mantener la coherencia de creación de objetos (compatibilidad con el dispositivo) en todo su desarrollo.

Ejercicio

En este caso se ha implementado una “Factoría Abstracta” con el objetivo de crear objetos de tipo carrera y bicicleta dentro de un contexto de localización y uso de los mismos para carretera y montaña.



Ejercicio 2: Patrón “Visitante” en C++.

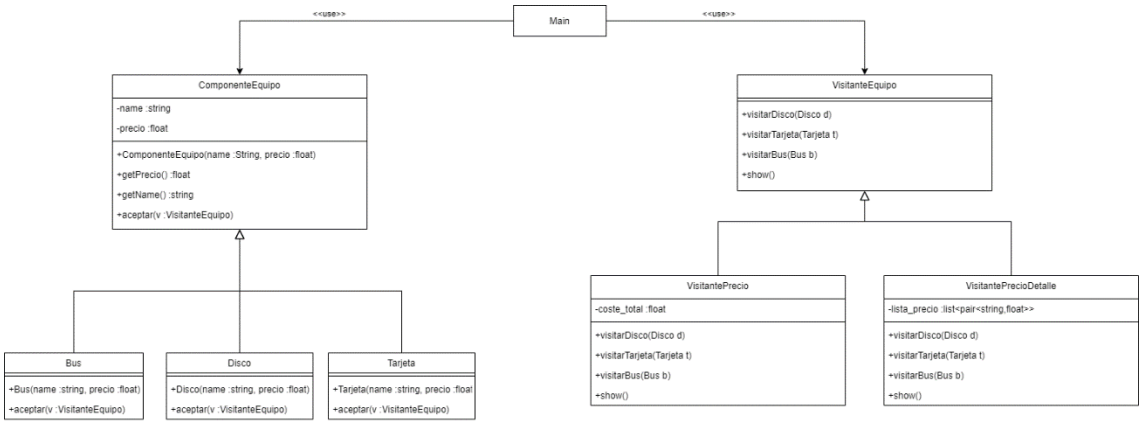
Visitante

El patrón de comportamiento “Visitante” es un patrón que permite añadir funcionalidades a una clase sin tener que modificarla, siendo usado para manejar algoritmos, relaciones o responsabilidades entre objetos. El patrón “Visitante” parte de un esquema de herencia de clases sobre la que se agrega una interface para el visitante.

Un ejemplo claro sería un esquema rígido de herencia sobre figuras (triángulo, cuadrado, pentágono) sobre las que se sabe que hay un conjunto de algoritmos que van a variar en el tiempo, pero no se van a añadir nuevas figuras. Se implementa una herencia llamada visitante con tantos herederos como algoritmos a implementar y tantos métodos como figuras. De esta forma la estructura de dados se separa de las operaciones. Se minimiza el coste de agregar un algoritmo a la estructura.

Ejercicio

En este ejercicio se ha implementado el patrón “Visitante” con el objetivo de separar los algoritmos del cálculo del precio y del precio detalle de cada uno de los componentes equipo.



Ejercicio 3: Patrón “Observador” para monitorización de datos meteorológicos en Java.

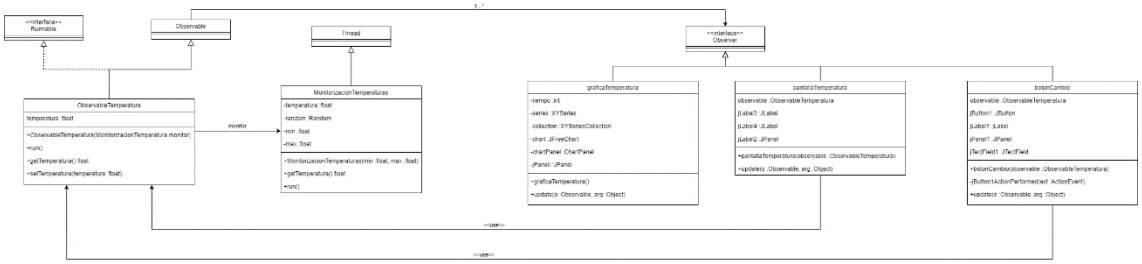
Observador

El patrón de comportamiento “Observador” permite gestionar la dependencia de varios objetos sobre uno solo. De esta forma el cambio de estado de del objeto podrá ser notificado al resto de objetos de manera óptima sin requerir consultar su actualización en bucle. El patrón “Observador” se implementa heredando el objeto a observar de la clase que gestiona el conjunto de observadores. Todos los observadores deben heredar de la misma interface.

Un ejemplo del patrón “Observador” puede ser cualquier sistema que tenga más de una forma de visualizar un dato. El dato es el observable y cada una de las vistas serían los observadores.

Ejercicio

En el ejercicio se ha hecho uso del patrón “Observador” implementado por Java para modelar la salida de datos de temperaturas mediante dos vistas, una gráfica y una pantalla. También permite de una modificación de la temperatura si se pulsa un botón.

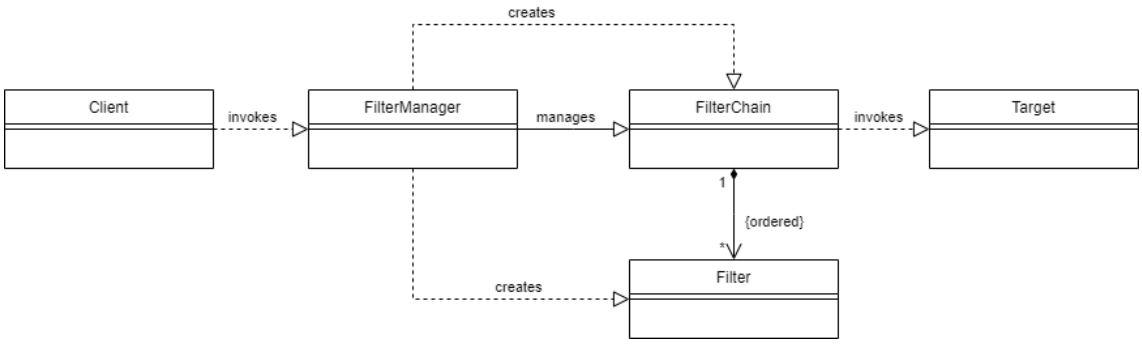


Ejercicio 4: Patrón “Filtros de intercepción” para realizar un simulador del movimiento de un vehículo con cambio automático en Java.

Filtros de intercepción

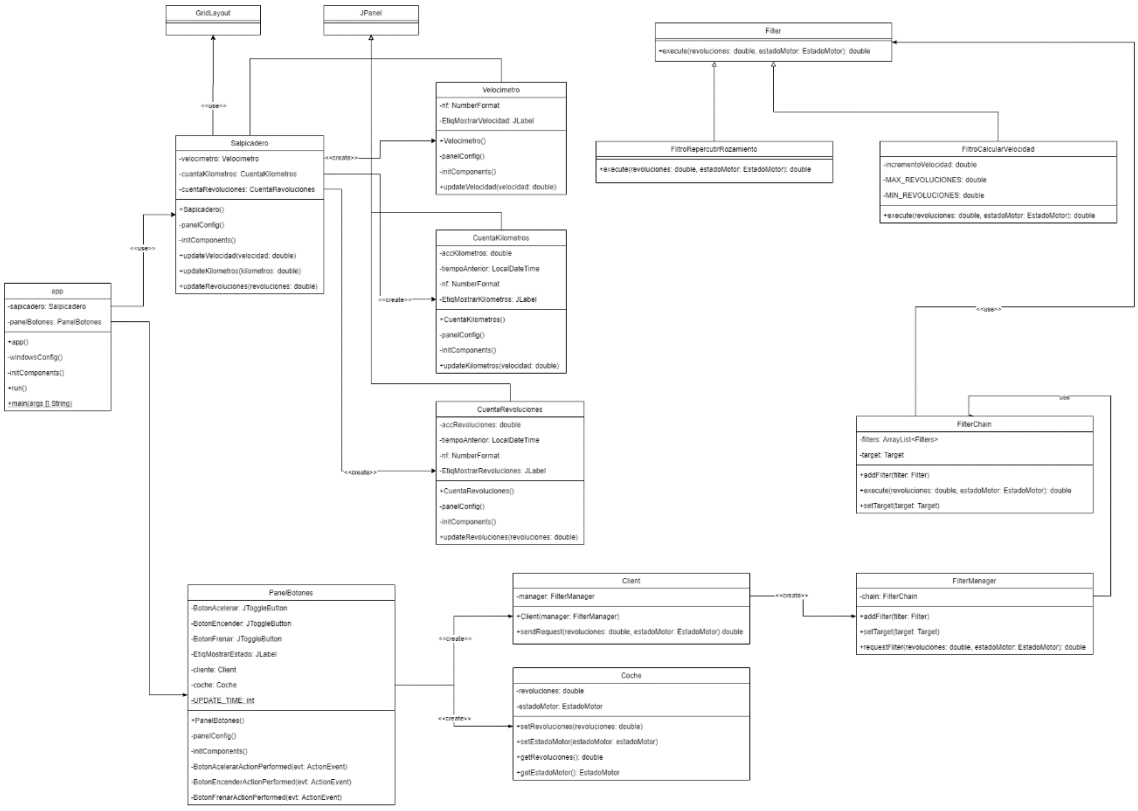
El patrón “Filtros de intercepción” es un patrón arquitectónico que permite la inclusión de nuevos servicios a un marco de trabajo de forma transparente para las aplicaciones y el sistema. Estos servicios se activan automáticamente cuando ocurren determinados eventos. Se utiliza cuando es necesario preprocesar una solicitud o una respuesta o también cuando posteriormente se procesa la solicitud o la respuesta.

Un ejemplo de aplicación de este patrón puede darse en una aplicación web mediante la implementación de una cadena de filtros que incluya operaciones relacionadas con seguridad o el proceso de registro.



Ejercicio

En el ejercicio se ha implementado dicho patrón sobre un sistema que simula parámetros del control de un coche sobre el salpicadero del vehículo.



Ejercicio 5: Simulador de control automático para la conducción de un vehículo (SCACV) desarrollado con Java/Swing.

Ejercicio 6: Desarrollo de una aplicación con tecnología autoaprendida.

Bibliografía

Balasch, M. C. (2017). *BettaTech*. Obtenido de Youtube: https://www.youtube.com/c/BettaTech

Gamma, E., Help, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns Elements of Reusable Objct-Oriented Software.* Indianapolis: Addison Wesley.

Source Making. (2006). Obtenido de https://sourcemaking.com/design_patterns

Trabajos propuestos para realización individual (2013-2014). (2014). Obtenido de https://lsi2.ugr.es/ist/trabajos.htm