

Taller Patrones de Diseño

# Vistazo general

Vamos a estar haciendo un taller de Patrones de diseño. La idea es agarrar un problema no-trivial, y pensar en patrones que mejor se ajustan para solucionarlo. Los patrones que se tienen en mente son los de cualquier libro de patrones de OOP.

[The Catalog of Design Patterns](https://refactoring.guru/design-patterns/catalog) contiene un listado muy completo.

Este taller va a estar en Java 21, y apunta además a fomentar la búsqueda de soluciones originales para salir del mindset de piloto automático.

Intentar apoyarse en IA cuando sea necesario (Chat GPT, Copilot, EYQ)

# Descripción del problema

En el contexto de un asistente inteligente de carteras de inversión, se pretende usar un archivo json para describir políticas de cartera.

Estas políticas se aplican en un proceso Bach que levanta de un .txt movimientos de bolsa de alguna cartera.

Para cada movimiento, se aplican todas las políticas, una por una, en orden. Una vez aplicada todas las políticas, se resuelve un resultado. Por ejemplo: EXTREME\_RISK, BEAR o BULL.

Segundo paso, se actualiza el estado de la cartera en base al resultado del paso anterior. Para saber a qué estado cambiar nos han mandado una lista de reglas de negocio (ver [anexo 3](#_Anexo_3:_Cambios) ) .

# Parámetros del Taller

Al inicio del taller se cuenta con:

* Un **json** con las políticas a ejecutar
* Un **txt** que contiene movimientos de carteras mock.
* El **proyecto base,** hecho en java 21. Este a su vez contiene:
  + un mini proceso que levanta registros del txt, pero no hace nada más.
  + un DAO que da un método para cambiar el estado de una cartera y uno para consultarlo.
  + Un Enum con todos los eventos posibles. Para usar con las políticas. ([anexo 2](#_Anexo_2:_Políticas))
  + Un Enum con todos los estados posibles de una Cartera
  + Una lista de pruebas para verificar que la solución implementada es correcta.

Hay que implementar la lógica necesaria completando el método **PolicyProcessor::process()** para que se ejecuten las políticas y en base al resultado arrojado, se actualice la cartera al estado correcto.Además, debe ser posible agregar nuevas políticas al json.

**El trabajo está ok si pasa las pruebas proveídas.**

Se deben aplicar patrones de diseño apropiados para implementar la lógica. **Aplicar Strategy, State, y Factory. Elegir por lo menos uno más y aplicarlo**. Para “fomentar” esta práctica, algunas restricciones:

* No se puede agregar Spring Framework al proyecto.
* Limitar niveles de IF anidados. Preferentemente no más de dos.
* Limitar bloques de código repetidos. Idealmente ninguno.

Repo: htps://github.com/SantiCaboEY/taller-patrones/

# Anexo 1: Archivo de movimientos

Un CSV, con formato:

carteraId | amount | company | type | isForeign

Ej:

12559 | 200.00 | APPLE | STOCK | TRUE

6998| 10000.00 | LELIQ | BOND| FALSE

100065| 600000000000.00 | NASDAQB30 | BOND| TRUE

# Anexo 2: Políticas

Se entra a las políticas con una lista resultante que contiene todos los eventos (O sea, todos los valores del enum Events).

Las políticas alteran la lista resultante. Si se ejecutan 0 políticas, la lista de salida es la misma que la de entrada.

El resultado es el primer evento de la lista resultante, una vez corrida las políticas.

Además, la variable de configuración POLICY\_TYPE indica que tipo de políticas ejecutar.

## Política Simple

POLICY\_TYPE = SIMPLE

Cada política del json siempre evalúa el campo monto y quita elementos de la lista resultante.

|  |
| --- |
| {  "comparator":**"greater\_than "**,  "compareTo": **"1000.00"**,  "events": **["EXTREME\_RISK** **","BEAR** **"]**  }, |

**Si amount > 1000.00, quitar (si están) de la lista EXTREME\_RISK y BEAR**

## Política Completa

|  |
| --- |
| [  {  "field": **"string"**,  "comparator":**"greater\_than | greater\_or\_equal| less\_than| less or equal | equal"**,  "compareToValue": **"string"**,  "operator": **"NOT | ONLY | RETURN"**,  "events": **["",""]**  },  {...},  ...  ] |

**field, comparator** y **compareToValue**: Forman una expresión que puede ser verdadera o falsa. **field** hace referencia a algún campo del .txt. ([anexo 1](#_Anexo_1:_Archivo))

**operator**: en caso de que la expresión sea true, en base a la lista **events**  se hace lo siguiente:

NOT -**>** se eliminan los eventos de **events** de la lista resultante.

ONLY **->** **events** reemplaza y pasa a ser la lista resultante.

RETURN **->** No se ejecutan más políticas y el resultado es el primer elemento de **events.**

**Ejemplo:**

|  |
| --- |
| [  {  "field": **"isForeign"**,  "comparator":**"equal "**,  "compareToValue": **"True"**,  "operator": **"ONLY "**,  "events": **["EXTREME\_RISK** **","BEAR** **"]**  },  …  ] |

**Si isForeign == True, reemplazar la lista resultante por ["EXTREME\_RISK** **", "BEAR** **"]**

# Anexo 3: Cambios de estado.

Negocio nos dio un listado de los criterios que fueron implementando. Esto es lo mejor que tienen, nos pidieron disculpas:

|  |
| --- |
| *Ante un evento* ***EXTREME\_RISK****, la cartera pasa al estado* ***CLOSED***  *Estos casos se resuelven manual, por lo que las carteras en* ***CLOSED*** *nunca deben salir de este estado.* |
| *Ante un evento* ***BULL****, Si la cartera estaba en* ***EMPTY*** *o* ***DEFENSIVE****, se pasa a* ***ACTIVE****. En otro caso se pasa a* ***VIP*** |
| *Ante un evento* ***BEAR,*** *la cartera pasa a* ***EMPTY*** |
| *Ante un evento* ***DEBT\_DEFAULT****, si la cartera no estaba en estado* ***EMPTY,*** *pasa a estado* ***DEFENSIVE****. Si ya estaba en* ***DEFENSIVE*** *pasa a* ***EMPTY*** *(¿?)* |
| *Ante un evento* ***OUT\_OF\_INVESTORS*** *, si cartera estaba* ***ACTIVE****, pasa a* ***DEFENSIVE****. Si ya estaba en* ***DEFENSIVE****, pasa a* ***CLOSED*** |
| *Ante un evento* ***MARKET****\_****COLLAPSE****, la cartera pasa e* ***EMPTY****. A menos que sea* ***VIP****, en cuyo caso pasa a* ***CLOSED*** *(¿?)* |
| *Ante cualquier otro evento, no modificar el estado de la cartera* |