



UNIVERSIDADE DA CORUÑA

Facultade de Informática

Trabajo fin de grado

Grado en Ingeniería Informática

Mención en Tecnologías de la Información

Aplicación para el análisis de carteras de fondos de inversión

Autor: López López, Ángel

Director: Castro Castro, Paula María

Director: González Coma, José Pablo

A Coruña, Junio de 2016

Índice

I	Introducción	3
1.	Introducción al mundo financiero	3
1.1.	Fondos de inversión	3
1.1.1.	Tipos de fondos	4
1.1.2.	Criterios para elegir un fondo de inversión.	4
1.1.3.	Operaciones y seguimiento de fondos	6
2.	Revisión de fundamentos tecnológicos	7
2.1.	Herramientas para la gestión de proyectos	7
2.2.	Herramientas para el modelado de software	7
2.3.	Herramientas para el desarrollo del proyecto	7
2.4.	Herramientas de bases de datos	8
2.5.	Herramientas para pruebas	8
2.6.	Herramientas de documentación	8
II	Metodología	9
3.	Proceso Unificado de Desarrollo Software	9
III	Planificación y evaluación de costes	10
IV	Desarrollo	12
4.	Primera iteración: Capa modelo	12
4.1.	Diseño de la base de datos	12
4.2.	Implementación de la base de datos mediante Hibernate	14
4.3.	Implementación del servicio del modelo	17
5.	Segunda iteración: Creación de la GUI	21
5.1.	Diseño de la interfaz	21
5.2.	Implementación de la interfaz	23
6.	Tercera iteración: Creación de las gráficas	31
6.1.	Preparación de las gráficas	31
6.2.	Gráficas de carteras	32
6.2.1.	Distribución de capital de la cartera	32
6.2.2.	Fondos normalizados de la cartera	33
6.2.3.	Fondos más y menos rentables de la cartera	34
6.2.4.	Comparativa de inversión	35
6.2.5.	Rentabilidad total de la cartera	36
6.3.	Gráficas de fondos	37

6.3.1.	Historial del Valor Liquidativo	37
6.3.2.	Historial de rentabilidades	38
6.3.3.	Rentabilidad esperada	39
6.3.4.	Medias Móviles	40
7.	Cuarta iteración: Ejecución de tareas en segundo plano	41
7.1.	SwingWorker	41
7.2.	Implementación de SwingWorker	43
7.2.1.	ChartWorker	43
7.2.2.	NodesWorker	44
7.2.3.	ImportFundWorker	44
V	Bibliografía	46

Capítulo I

Introducción

1. Introducción al mundo financiero

Para poder llevar a cabo este proyecto, ha sido necesario realizar un primer paso de búsqueda de información acerca del mundo de las finanzas, mas concretamente sobre los fondos de inversión, para poder conocer su funcionamiento, sus métricas y los tipos de datos que en ellos se utilizan.

1.1. Fondos de inversión

Para comenzar definiremos qué es un fondo, cómo funciona y los elementos que en el intervienen:

Un **fondo de inversión** es un capital compuesto por la suma de las aportaciones monetarias realizadas por varias personas. Este capital se invertirá en una serie de activos con el objetivo de obtener la máxima rentabilidad posible. Dependiendo de la evolución de estos activos, el fondo arroja resultados positivos o negativos, los cuales se repartirán entre cada inversor según la proporción que represente su inversión sobre el total del patrimonio del fondo. Cada fondo se encuentra identificado por un International Securities Identification Number (ISIN). Este código identifica unívocamente un valor mobiliario a nivel internacional.

Los fondos de inversión se dividen en partes proporcionales llamadas **participaciones** y sus propietarios se denominan **participes**. El número de participaciones no es fijo, sino que depende de las compras y ventas de las mismas. Su valor, denominado **Valor Liquidativo (VL)** de la participación, se calcula diariamente de la siguiente manera:

$$Valor\ liquidativo = \frac{Patrimonio\ del\ fondo}{N\ de\ participaciones\ en\ circulacion} \quad (1)$$

Este valor depende, por tanto, de la evolución diaria de los valores que componen el patrimonio del fondo y será uno de los indicadores fundamentales que utilizará la aplicación a la hora de realizar los históricos de los diferentes fondos. Otra medida importante es la **rentabilidad del fondo**. Esta se calcula mediante la diferencia porcentual entre el VL en la fecha de compra de la participación (suscripción) y la fecha de venta (reembolso), de la siguiente manera:

$$Rentabilidad = \frac{Valor\ liquidativo\ final - Valor\ liquidativo\ inicial}{Valor\ liquidativo\ inicial} \times 100. \quad (2)$$

El resultado no es percibido de manera efectiva hasta que no se produzca el reembolso de las participaciones y será en ese momento en el que el partícipe deberá tributar por el resultado de su inversión.

Otro aspecto importante es que las decisiones de la inversión las toma una **gestora**, que administra y representa el fondo, mientras que la función de custodiar y vigilar los activos la realiza el llamado **depositario**, generalmente una entidad financiera. Normalmente la gestora cobra una serie de comisiones de gestión que se restan al fondo, lo cual disminuye el VL de cada participación.

Los siguientes puntos se centrarán en ver los distintos tipos de fondos que podemos encontrar, los criterios que se deben de utilizar para su elección y las operaciones que podemos realizar sobre ellos.

1.1.1. Tipos de fondos

En el mercado existen una amplia gama de fondos de inversión. Es tarea del propio inversor elegir aquél que más se adapte a sus necesidades.

- **Fondos de renta fija:** Son fondos donde la mayoría de sus activos son de renta fija (obligaciones y bonos, letras, pagarés, etc). Normalmente, la rentabilidad de estos fondos va ligado al plazo de vencimiento de dichos activos, es decir, a menor plazo, menos riesgo y por lo tanto, menos rentabilidad prevista, y viceversa.
- **Fondos de renta variable:** Son fondos donde la mayoría de sus activos son de renta variable (acciones). Por lo general, los fondos de renta variable reportan ganancias o rendimiento a largo plazo, a cambio de un mayor riesgo.
- **Fondos Mixtos:** Son fondos en los que sus activos se encuentran divididos entre activos de renta fija y renta variable. Cuanto mayor sea el porcentaje de activos de renta variable mayor sera el riesgo y la rentabilidad potencial.
- **Fondos globales:** Son fondos que suelen incluir renta variable, fija y activos monetarios en diferentes localizaciones geográficas, en determinados porcentajes dependiendo de la política del fondo, de forma que sus inversiones estén muy diversificadas.
- **Fondos garantizados:** Son fondos que aseguran la recuperación del capital inicialmente invertido más una rentabilidad fija o variable, en una fecha futura determinada.
- **Fondos monetarios:** Son fondos basados en la adquisición de activos a corto plazo para minimizar el riesgo de la inversión obteniendo la máxima rentabilidad posible.

1.1.2. Criterios para elegir un fondo de inversión.

Como hemos visto en el apartado anterior, existen varios tipos de fondos de inversión adaptados a diferentes necesidades. A la hora de elegir un fondo en particular existen varios ratios e indicadores que pueden ayudar a determinar cual es el más adecuado a las preferencias del inversor.

Normalmente, a la hora de seleccionar un fondo, el inversor debe considerar cual es su capacidad para asumir pérdidas (pues cuanto mayor es el riesgo también lo es la rentabilidad) así como el horizonte temporal durante el cual desea mantener la inversión, pues, dependiendo de la política del fondo, puede ser aconsejable estar dispuesto a mantener la inversión un determinado período de tiempo.

Otro aspecto a tener en cuenta son las comisiones que se cargan a los fondos de inversión, puesto que pueden afectar a la rentabilidad. Es posible que un fondo aplique distintos tipos de comisiones a las diferentes tipos de participaciones que emita.

También hemos de considerar el comportamiento histórico que ha tenido un fondo a lo largo del tiempo. Es importante conocer las rentabilidades obtenidas en el pasado, aunque esto no signifique que se siga una línea similar en el futuro. En la aplicación a desarrollar se incluirán históricos de las rentabilidades referidas a un determinado período (trimestre, semestre ...) para que al comparar distintos fondos se puedan contrastar las rentabilidades en los mismos períodos. Cabe mencionar que es necesario que los fondos sigan una misma política de inversión para que la comparación sea significativa.

Es posible que durante la vida de un fondo éste cambie su política de inversión e incluso de grupo gestor, por lo que al consultar rendimientos pasados hay que tener en cuenta que puede que éstos hayan cambiado. Es importante conocer la fecha de dicho cambio y tener en cuenta sólo las rentabilidades a partir de ese momento.

Por último, algunas métricas o indicadores que se deben utilizar para elegir un fondo de inversión son los siguientes:

- **Volatilidad:** es una medida de variación (cambios) en el precio de un activo. Mide cuanto varía el precio de un activo respecto a su precio medio y cuantifica el riesgo del activo financiero.
- **Alfa:** mide la capacidad o habilidad que tiene el gestor de generar valor al fondo de inversión.
- **Beta:** mide la sensibilidad del VL de un fondo a los movimientos de su índice de referencia.
- **Ratio de Sharpe:** nos dice lo bueno que es un fondo de inversión en la relación rentabilidad-riesgo.
- **Ratio de Información:** es una medida que se emplea para determinar la influencia que ha tenido un gestor en la rentabilidad del fondo en comparación con el comportamiento del mercado.
- **Máximo Drawdown:** se define como la máxima caída experimentada por un fondo en el periodo comprendido desde que se registra un máximo hasta que vuelve a ser superado.

1.1.3. Operaciones y seguimiento de fondos

En este último punto hablaremos sobre las operaciones de suscripción, reembolso y traspaso de un fondo de inversión y de cómo realizar el seguimiento de su rentabilidad.

El método para realizar una inversión en un fondo consiste en la **suscripción** de participaciones. La entidad gestora emite una serie ellas y cada inversor obtiene tantas como el resultado de dividir el capital invertido entre el VL (1.1) aplicable a la operación. Normalmente el VL aplicable es el del mismo día de la solicitud o el del día siguiente a la solicitud. Algunos fondos pueden estar sujetos a comisiones de suscripción, de hasta un 5 % de la inversión.

Si un inversor quiere recuperar su dinero debe solicitar un **reembolso** de todas o parte de sus participaciones, recibiendo el resultado de multiplicar el el VL (1.1) de la participación por el número de participaciones que quiera reembolsar. El VL aplicable es el mismo que en el caso anterior, el del mismo día o el del día siguiente. El plazo en el que el inversor recibe su dinero es como máximo de 3 a 5 días, pudiendo tener dicho reembolso una comisión de hasta el 5 %, como en el caso anterior. El inversor conocerá el resultado de la inversión (positivo o negativo) cuando se le abone el reembolso.

En el caso de querer realizar un **traspaso** de un fondo a otro se produce un reembolso del primero y la inmediata suscripción al segundo. Este método tiene una ventaja, pues se conserva la antigüedad de la primera inversión a efectos fiscales, por lo que las plusvalías no se tributan hasta que se produzca el reembolso definitivo.

Existen cuatro partes que intervienen en un traspaso:

- **Fondo de origen:** fondo en el que se mantiene la inversión antes del traspaso.
- **Fondo de destino:** fondo en el que quiere invertir el capital que se reembolse del fondo de origen.
- **Entidad de origen:** la que comercializa o gestiona el fondo de origen.
- **Entidad de destino:** la que comercializa o gestiona el fondo de destino.

Sin embargo, al tratarse de de una operación de reembolso y suscripción, se deberán abonar las respectivas comisiones que tengan establecidas ambos fondos.

El proceso de **seguimiento** de un fondo de inversión puede realizarse principalmente a través de dos fuentes:

- La documentación que proporcione la entidad gestora o depositaria. Pues es obligatorio que se proporcione a los partícipes información periódica acerca de la evolución de sus inversiones.
- La divulgación de datos sobre fondos de inversión que proporcionan periódicos o diversos portales de internet. De esta última fuente obtendremos los datos necesarios para el funcionamiento inicial de la aplicación.

2. Revisión de fundamentos tecnológicos

En este apartado se describen las herramientas y tecnologías usadas para el desarrollo del proyecto.

2.1. Herramientas para la gestión de proyectos

- **Git:** Es un software de control de versiones distribuido que permite trabajar en grupo y mantener accesibles las diferentes versiones de la aplicación del proyecto. Se ha elegido git como sistema de gestión de versiones del proyecto por encima de su principal alternativa, subversion porque proporciona un repositorio local sobre el que se puede trabajar off-line.

2.2. Herramientas para el modelado de software

- **Visual Paradigm Community Edition:** Es una herramienta para el desarrollo de aplicaciones utilizando Lenguaje Unificado de Modelado (UML) recomendada para la aplicación y el seguimiento del Proceso Unificado de Desarrollo Software (PUD). Proporciona asistencia para realizar los análisis, diseño, casos de uso y modelos UML del proyecto. Se trata de un software con licencia pero se utiliza su versión gratuita (Community Edition).

2.3. Herramientas para el desarrollo del proyecto

- **Eclipse IDE Neon:** Es una plataforma de desarrollo de software de código abierto y multiplataforma basada en Java. Proporciona Entornos de Desarrollo Integrados (IDEs) prácticamente para cualquier lenguaje, siendo el mas utilizado el de Java. Provee también de una serie de plugins para el control de versiones y frameworks para el desarrollo de aplicaciones gráficas.

Se ha elegido Eclipse como plataforma de desarrollo de este proyecto en lugar de Netbeans debido a que ha sido la plataforma utilizada en la mayoría de las asignaturas del grado, por lo que su funcionamiento es mas conocido. Para la sincronización con el repositorio git se utiliza el plugin EGit.

- **Apache Maven:** Es un software de gestión de proyectos software desarrollado por Apache. Esta basado en el concepto de Project Object Model (POM), maven permite gestionar dependencias, módulos, componentes y el orden de construcción.

2.4. Herramientas de bases de datos

- **MySQL:** Es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation y está considerada como la base datos open source más popular del mundo. Se utiliza en este proyecto para guardar los datos de los diferentes fondos de inversión de tal forma que puedan estar disponibles off-line.

Su principal alternativa es PostgreSQL, pero se ha decidido utilizar MySQL porque nuestra aplicación utilizará principalmente consultas sencillas, normalmente de lectura y MySQL esta orientado a este tipo de tareas proporcionando un mayor rendimiento que su competidor.

- **Hibernate:** Es un framework para el mapeo objeto-relacional de código abierto para la plataforma Java. Su uso facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos denominados eXtensible Markup Language (XML) o anotaciones en los beans de las entidades para poder establecer estas relaciones.

2.5. Herramientas para pruebas

- **JUnit:** Es un framework utilizado para realizar pruebas unitarias a aplicaciones Java, permitiendo realizar estas pruebas de forma controlada y evaluar el funcionamiento de cada uno de los métodos de las clases.

2.6. Herramientas de documentación

- **LaTeX:** Es un software gratuito de composición de textos para la elaboración de documentos de índole científica. Latex proporciona una serie de características que proporcionan una gran calidad tipográfica en sus documentos.

Capítulo II

Metodología

3. Proceso Unificado de Desarrollo Software

Para la realización de este proyecto se utilizará la metodología de **PUD**. El PUD * es un marco de desarrollo extensible, dirigido por casos de uso, iterativo e incremental, en el cual los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones.

El PUD presenta las siguientes características:

- **Está dirigido por casos de uso:** Cada caso de uso representa un requisito funcional y su conjunto forma el modelo de casos de uso.
- **Esta centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo y describe los elementos del modelo que son más importantes para poder desarrollarlo.
- **Iterativo e incremental:** El trabajo es dividido en tareas más pequeñas o iteraciones. El resultado de cada iteración es un sistema ejecutable, una nueva versión del producto final. Cada una de estas iteraciones resulta en un incremento en el proyecto y se divide a su vez en: análisis de requisitos, diseño, implementación y prueba.

Como lenguaje de representación visual el PUD utiliza el UML y se ha seleccionado para este proyecto porque está concebido para la programación orientada a objetos, acelera el ritmo del desarrollo y reduce el coste del riesgo a un solo incremento.

Capítulo III

Planificación y evaluación de costes

En este apartado se detalla cómo se ha aplicado el PUD para gestionar el desarrollo de la aplicación. Debido a que se utiliza un marco de desarrollo incremental, en cada iteración se presentará un nuevo caso de uso que conformará una nueva versión del producto final.

El objetivo de este Trabajo Fin de Grado (TFG) es implementar una aplicación en la que los usuarios puedan obtener gráficos, datos numéricos y resultados de una o varias carteras de fondos de inversión, por lo cual las fases en las que se divide el proyecto son las siguientes:

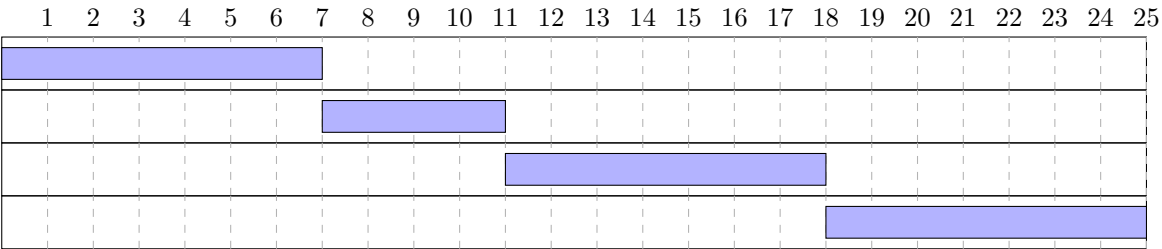
Tabla 1: Planificación del proyecto

Fase	Iteración	Tareas
Inicial	Preliminar	Inmersión en el mundo financiero
		Elección de indicadores, medidas y criterios más significativos
		Localización y selección de las fuentes de datos
		Definición las diferentes iteraciones en las que se realizará el proyecto
Diseño	1	Búsqueda de requisitos funcionales
	2	Realización de los modelos de los casos de uso
Desarrollo	1	Desarrollo de la IT 1
	2	Desarrollo de la IT 2
	3	Desarrollo de la IT 3
	4	Desarrollo de la IT 4
Documentación	Final	Redacción del manual de usuario

Para una mayor monitorización de las tareas a realizar se incluye a continuación un diagrama de Gantt con la planificación temporal estimada para cada tarea.

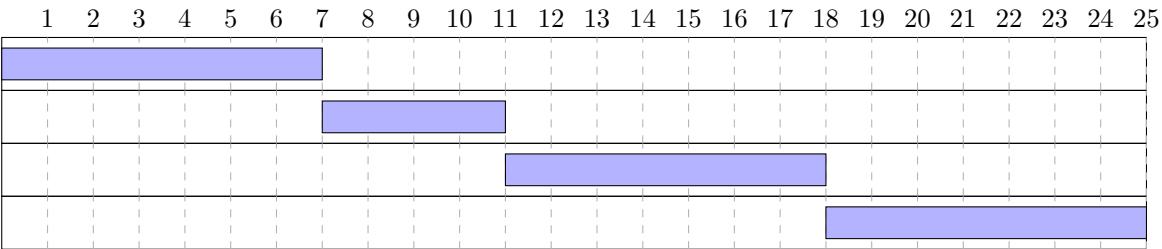
Tareas

- 1 Introducción al mundo financiero
- 2 Metodología, Planificación y costes
- 3 Formato de la base de datos
- 4 Tecnologías empleadas



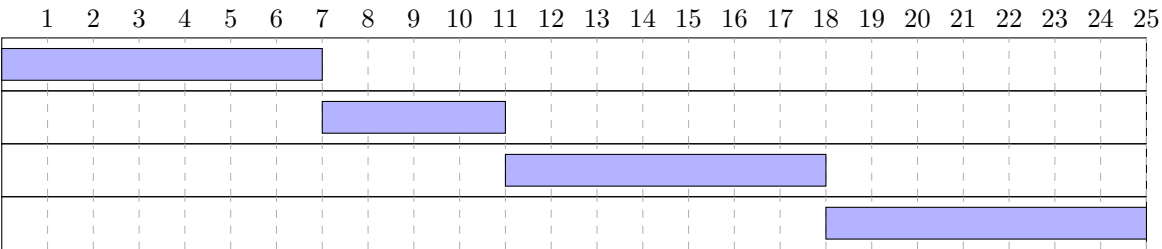
Tareas

- 1 Introducción al mundo financiero
- 2 Metodología, Planificación y costes
- 3 Formato de la base de datos
- 4 Tecnologías empleadas



Tareas

- 1 Introducción al mundo financiero
- 2 Metodología, Planificación y costes
- 3 Formato de la base de datos
- 4 Tecnologías empleadas



Capítulo IV

Desarrollo

4. Primera iteración: Capa modelo

Esta primera iteración de desarrollo tiene como objetivo sentar las bases de la aplicación, proporcionando una capa modelo que exponga las funcionalidades necesarias para el acceso a los datos de los diferentes fondos de inversión.

La base de datos utilizada en la aplicación es MySQL junto con el Framework Hibernate.

4.1. Diseño de la base de datos

Para comenzar el diseño de la base de datos debemos analizar cuáles son las funcionalidades y las estructuras de datos que necesitamos para la aplicación.

Por un lado, hemos de ser capaces de guardar los principales campos que contiene un fondo de inversión: ISIN del fondo, nombre de la entidad gestora, el tipo de fondo que es, la categoría, la divisa con la que operan sus inversiones y sus comisiones de apertura y cancelación.

Por otro, cada fondo puede tener una cantidad variable de VLs, normalmente uno por día durante el período en el que el fondo se encuentra activo, exceptuando los festivos.

Una primera aproximación del Modelo Entidad-Relación (ER) es el siguiente:

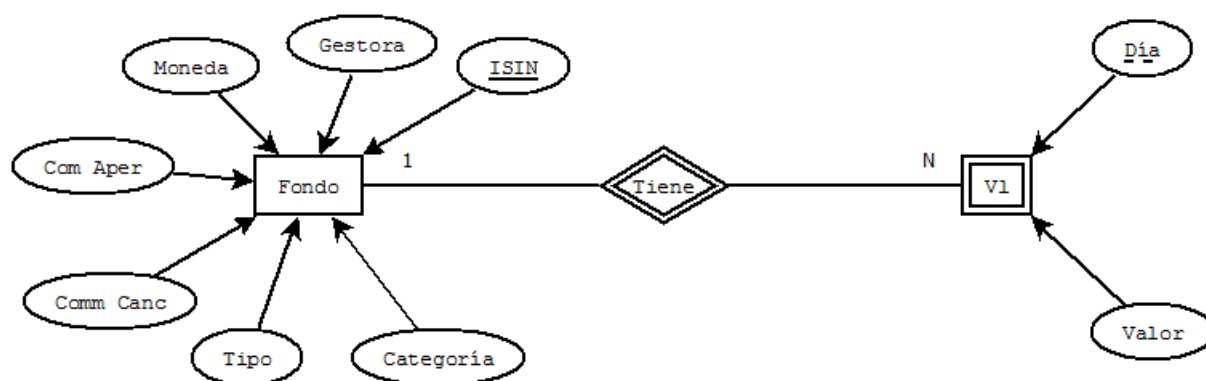


Figura 1: Diagrama ER inicial.

Este modelo está compuesto por una entidad fuerte encargada de guardar la información referente a los fondos y una débil, que presenta una dependencia de identificación con la primera, encargada de guardar los VLs de cada fondo. Por tanto, cada fondo puede tener N VLs y cada VL sólo puede pertenecer a un fondo.

Este modelo permite almacenar los datos básicos tanto de un fondo como el histórico de sus VLs, pero no permite realizar operaciones de compra/venta de participaciones. Para poder realizar operaciones sobre los diferentes fondos es necesario añadir un poco de complejidad al modelo:

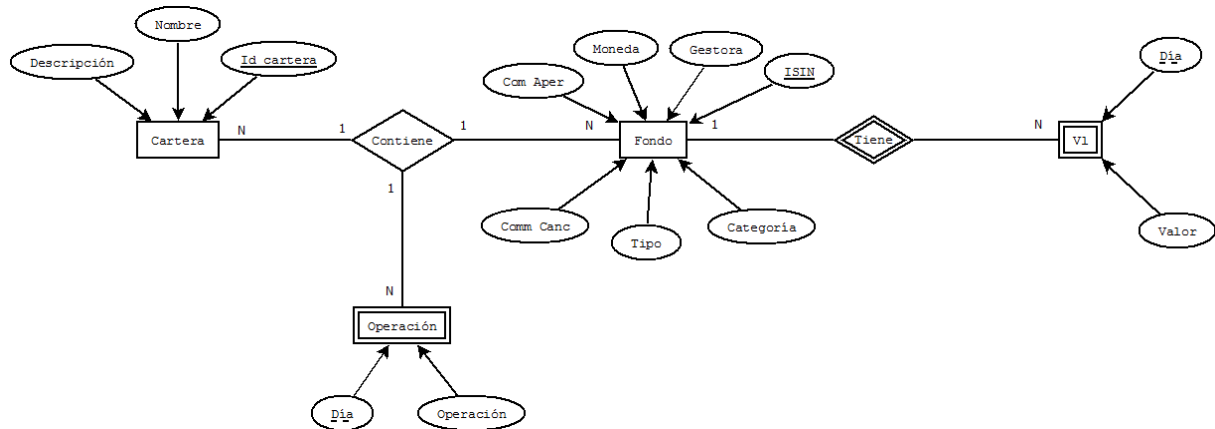


Figura 2: Diagrama ER final

De esta forma incorporamos la posibilidad de tener carteras de fondos y realizar operaciones de compra/venta de participaciones en diferentes fondos pertenecientes a diferentes carteras.

Cada fondo de inversión puede pertenecer a varias carteras distintas y cada cartera estará compuesta por varios fondos. Las operaciones sobre ellos sólo estarán permitidas si pertenecen a una cartera de inversión. En caso contrario, sólo se podrá acceder a su histórico de VLs.

Una vez terminado el diagrama ER, el siguiente paso es realizar su conversión al modelo relacional. El resultado es el siguiente:

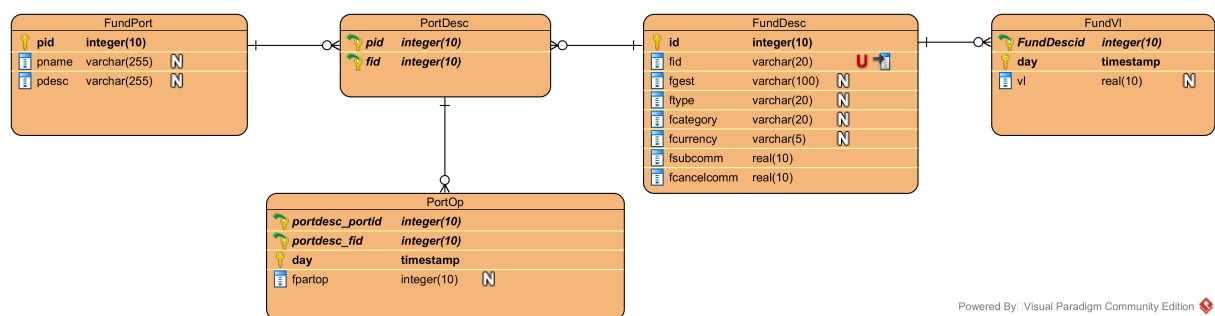


Figura 3: Diagrama del modelo relacional (eliminar los nulos)

4.2. Implementación de la base de datos mediante Hibernate

Para implementar la base de datos usando Hibernate, primero debemos modelar las clases Java correspondientes a las tablas del modelo relacional:

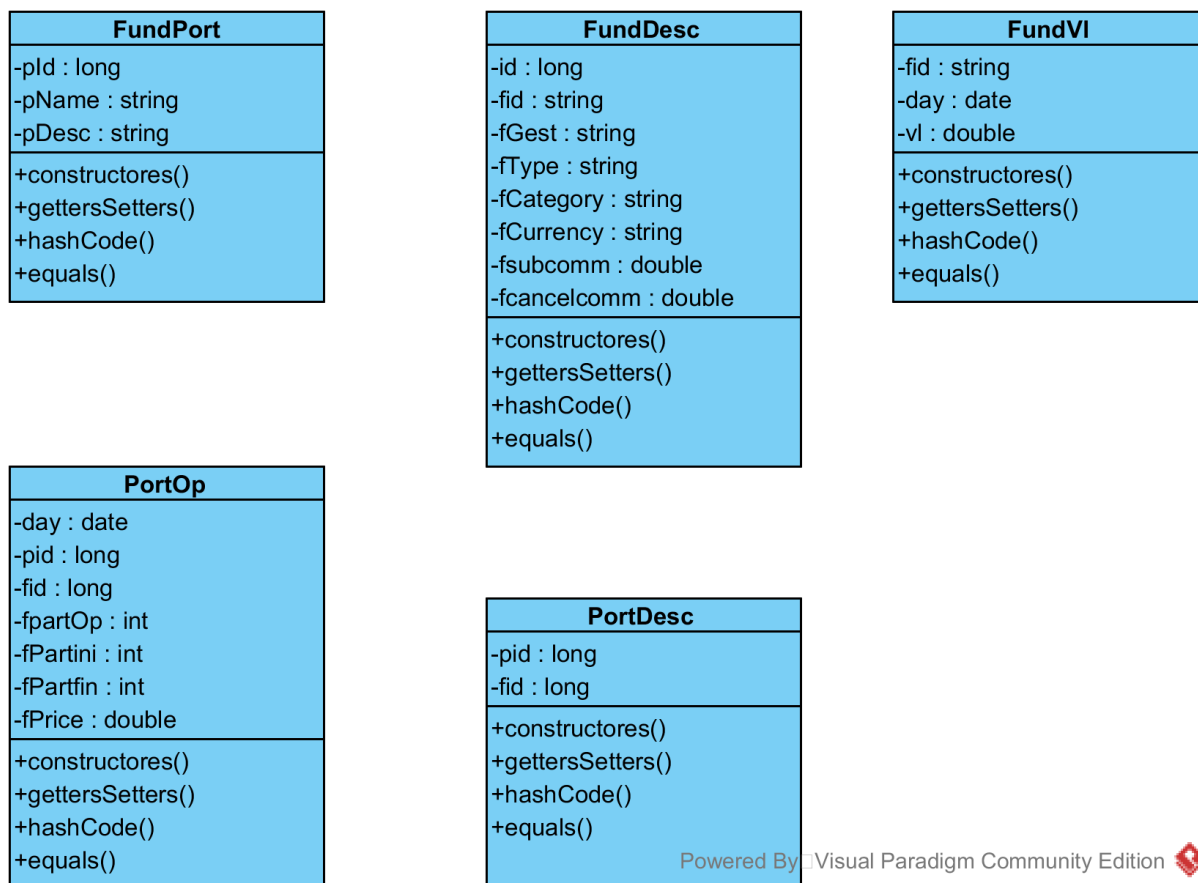


Figura 4: Diagrama de clases del modelo.

Los atributos de FundDesc, **id** y de FundPort **pId**, están formados por valores autogenerados por la base de datos mediante la estrategia de generación “*GenerationType.IDENTITY*” y actúan como claves primarias de las tablas. En el caso de FundDesc el atributo **fid** (que se corresponde con el ISIN del fondo) es una clave candidata que se guardará en la base de datos como un Varchar de longitud 12, indicando a Hibernate que es un valor único y no puede ser nulo. Este atributo podría haberse utilizado como clave primaria de la tabla FundDesc, pero al tratarse de un tipo de dato Varchar que se repetirá, como norma general, más de un millar de veces por cada fondo en la tabla FundVI, se ha decidido utilizar un id autogenerado por motivos de rendimiento.

Para que Hibernate mapee las relaciones es necesario incluir en cada una de las clases con relación 1:N una lista con los objetos relacionados. Por ejemplo, cada objeto FundDesc tiene, a mayores de los atributos mostrados en el UML, uno llamado fundVls y otro llamado portDescs, que contienen una lista de los FundVls y PortDescs relacionados con ese fondo, respectivamente.

En el caso de las relaciones N:1 basta con incluir un atributo del tipo del objeto con el que se relaciona. Por ejemplo, cada objeto `FundVl` tiene un atributo llamado `fundDesc` que contiene el fondo al que pertenece ese `FundVl`. Por este motivo, ha sido necesario modificar la implementación por defecto de los métodos `equals` y `hashCode`, para que su uso no provocase una llamada recursiva infinita que causase una excepción por `stack overflow`.

Para poder implementar las clases correspondientes a tablas con claves compuestas en Hibernate, es necesario implementar clases serializables que contengan únicamente los atributos que componen la clave primaria de la tabla. En este caso se han implementado las tres necesarias:

- **FundVIPk:** Con los atributos `fundDesc` y `day`.
- **PortDescPk:** Con los atributos `fundDescId` y `fundPortId`.
- **PortOpPk:** Con los atributos `portDesc` y `day`.

En el caso de `PortOpPk`, el atributo `portDesc` es mapeado en base de datos como el conjunto de las ids del fondo y de la cartera.

A continuación, se definen las anotaciones de Hibernate utilizadas para crear la base de datos.

- **@Id:** Indica que el siguiente atributo es la clave primaria de la tabla.
- **@IdClass:** Indica la clase que contiene los atributos que componen la clave primaria de la clase/tabla actual.
- **@Column:** Indica que el siguiente atributo se modela como una columna en la base de datos.
- **@Transient:** Indica que el siguiente atributo no se modela como una columna en la base de datos.
- **@GeneratedValue:** Indica que el siguiente atributo es generado automáticamente por la base de datos, utilizando la estrategia asignada.
- **@OneToMany:** Indica que el objeto tiene una relación 1:N con otra tabla.
- **@ManyToOne:** Indica que el objeto tiene una relación N:1 con otra tabla.
- **@JoinColumn:** Indica que el objeto obtiene una columna de otra tabla, normalmente se utiliza junto con la anotación anterior.

La anotación **@Transient** se utiliza en la clase `PortOp` para indicar que los atributos `fPartIni`, `fPartFin` y `fPrice` no se guardan en la base de datos y serán calculados bajo demanda cuando se realice una operación de búsqueda.

Las anotaciones `@OneToMany` y `@ManyToOne` tienen algunas opciones interesantes:

- La opción **fetch** indica si las listas de objetos relacionados se cargaran bajo demanda (EAGER) o inmediatamente (LAZY).
- La opción **cascade** hace referencia a las acciones que se deben de llevar a cabo tras la eliminación de un objeto, `CascadeType.REMOVE` indica que se deben eliminar todos los objetos relacionados con él (Por ejemplo, si se elimina un `FundDesc` se eliminan también todos sus `FundVls`).
- La opción **mappedBy** se utiliza para definir el atributo que mapea la relación entre las tablas.

El último paso para crear la base de datos consiste en indicar a Hibernate cuáles son las clases que debe mapear en su fichero de configuración *“src/main/resources/hibernate.cfg.xml”*.

De esta forma, cuando realicemos la llamada de inicio de sesión *“sessionFactory = new Configuration().configure().buildSessionFactory()”* Hibernate generará las tablas en la base de datos, si no existían previamente.

La propiedad *“hbm2ddl.auto”* nos permite controlar el comportamiento de hibernate cuando se inicia la sesión. Podemos, desde eliminar las tablas y volver a crearlas en cada inicio, hasta simplemente actualizarlas si hay cambios.

4.3. Implementación del servicio del modelo

El servicio del modelo será el encargado de exponer las funcionalidades necesarias para el funcionamiento lógico de la aplicación.

Como funcionalidades mínimas necesitamos tener la capacidad de añadir, buscar, actualizar y eliminar elementos de cada una de las tablas de la base de datos, pero a la hora de realizar gráficas es muy interesante, por ejemplo, la posibilidad de obtener los VLS y las operaciones sobre un fondo en un rango de fechas determinado, por lo que el servicio consta de los siguientes métodos:

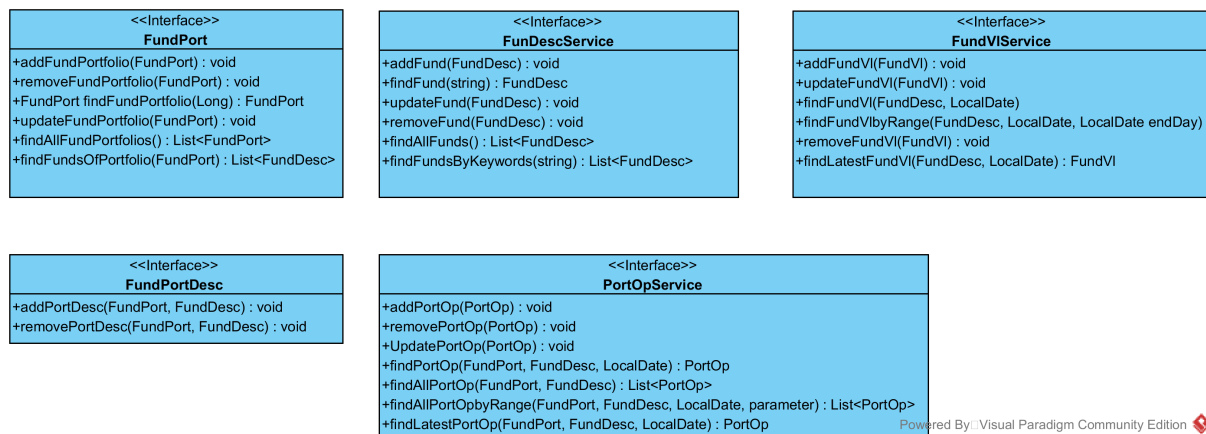


Figura 5: Diagrama del servicio

Aunque en este diagrama existe una interfaz del servicio para cada clase, en la implementación se han agrupado todos en una única interfaz denominada **FundService**.

Cada uno de los métodos del servicio lanza una **RuntimeException** si se produce algún error a la hora de conectarse a la base de datos. A mayores, se han implementado dos excepciones más:

- **InputValidationException:** Se produce cuando los datos introducidos por el usuario no son correctos.
- **InstanceNotFoundException:** Se produce cuando no se puede encontrar el objeto solicitado en la base de datos.

Para llevar a cabo la validación de los datos de entrada se ha incorporado un validador:

uml del validador

Existen una serie de métodos en el servicio que utilizan este validador para asegurar que los datos introducidos son correctos:

- **validateFundVl:** Se encarga de validar si el campo VL del FundVl es mayor o igual que cero.
- **validateFund:** Se encarga de validar que el ISIN del fondo es correcto y llama a su vez a validateFundVl para validar cada uno de sus FundVl.
- **validateFundPort:** Se encarga de validar que el nombre de la cartera de fondos no sea un string vacío.
- **validatePortOp:** Se encarga de validar que la operación añadida, eliminada o actualizada no deje en ningún momento a la base de datos con un total de participaciones negativo.

Ahora analizaremos cada método por separado:

- FundDesc
 - **addFund:** añade un FundDesc y toda su lista de FundVLs a la base de datos. La forma más común de representación de un fondo es mediante un fichero excel (.xls) que contenga un conjunto de pares día-VL, en un futuro la aplicación permitirá importar los VLs a partir de estos ficheros, por lo que se podrán guardar en la base de datos simplemente llamando a esta operación.
 - **findFund:** devuelve el FundDesc a partir de su ISIN. A pesar de que la clave primaria de la tabla sea la clave subrogada “*id*”, es más natural buscar un fondo en concreto por su ISIN.
 - **updateFund:** actualiza los campos de un FundDesc.
 - **removeFund:** elimina un FundDesc y toda su lista de FundVLs de la base de datos.
 - **findAllFunds:** obtiene todos los FundDesc de la base de datos.
 - **findFundsByKeywords:** obtiene los fundDesc que coinciden con una serie de caracteres en alguno de sus campos. Esta función es útil a la hora de buscar varios fondos de un mismo tipo, moneda o gestora, por ejemplo.
- FundPort
 - **addFundPortfolio:** añade una cartera de fondos.
 - **removeFundPortfolio:** elimina una cartera de fondos.
 - **findFundPortfolio:** obtiene una cartera de fondos a partir de su Id.
 - **updateFundPortfolio:** actualiza los campos de una cartera de fondos.
 - **findAllFundPortfolios:** obtiene todas las carteras de la base de datos.
 - **findFundsOfPortfolio:** obtiene todos los fondos de una cartera.

■ FundVl

- **addFundVl:** añade un unico FundVl a un fondo en un día concreto.
- **removeFundVl:** elimina una fila de la tabla VL de un fondo en un día concreto.
- **findFundVl:** obtiene el FundVl de un fondo en un día concreto.
- **updateFundVl:** actualiza un único FundVl de un fondo en un día concreto.
- **findFundVlByRange:** obtiene los Vl de un fondo dado en el intervalo de tiempo deseado.
- **findLatestFundVl:** obtiene el FundVl del día mas próximo a uno dado (se comporta exactamente igual a findFundVl si existe un valor vl en ese día). Esta función se utiliza para calcular el valor monetario de una operación. Cuando no se disponga del VL de un fondo el día de la operación, se utilizará en su lugar el valor del día anterior más cercano.

■ PortDesc

- **addPortDesc:** añade un fondo a una cartera.
- **removePortDesc:** elimina un fondo de una cartera.

■ PortOp

- **addPortOp:** añade una operación (con participaciones como unidad) sobre un fondo en una cartera en un día determinado.
- **removePortOp:** elimina una operación realizada sobre un fondo en una cartera en una fecha.
- **findPortOp:** obtiene una operación sobre un fondo en una cartera en un día determinado.
- **updatePortOp:** actualiza un PortOp (con participaciones como unidad) sobre un fondo en una cartera en un día determinado.
- **findAllPortOp:** devuelve todas las operaciones realizadas sobre un fondo en una cartera.
- **findPortOpByRange:** devuelve todas las operaciones realizadas sobre un fondo en una cartera entre dos fechas.
- **findLatestPortOp:** obtiene la operación sobre un fondo en una cartera del día mas próximo a un día dado (Se comporta exactamente igual a findPortOp si existe un PortOp en ese día). Esta función se utiliza para conocer cual ha sido la operación más reciente sobre un fondo, lo que permite ver las participaciones que tenemos invertidas en él.

Como se ha comentado anteriormente, los campos de los objetos PortOp: fPartIni, fPartFin y fPrice, se calculan bajo demanda en los métodos de búsqueda findPortOp, findAllPortOp, findPortOpByRange y findLatestPortOp. La operación encargada de calcularlos es calculatePortOp.

El método **calculatePortOp** revisa todas las operaciones anteriores y obtiene el número de participaciones inicial y final, además del precio de la operación, utilizando para ello el VL más próximo y teniendo en cuenta las comisiones de apertura y cancelación del fondo.

Se utiliza esta operación en lugar de almacenar directamente los valores de los campos en la base de datos debido a que si se guardasen, cada inserción, actualización o borrado de una operación, implicaría actualizar todos los valores de las operaciones siguientes, además de estar guardando información redundante que se puede calcular directamente con los datos que ya se encuentran en la base de datos.

5. Segunda iteración: Creación de la GUI

Esta segunda iteración del desarrollo tiene como objetivo construir la Interfaz Gráfica de Usuario (GUI), que será la encargada tanto de mostrar los datos al usuario como de proveer de datos al modelo.

La biblioteca gráfica utilizada es Swing, un framework MVC para desarrollar interfaces gráficas para Java con independencia de la plataforma.

5.1. Diseño de la interfaz

Como en el apartado anterior, para comenzar a diseñar la interfaz debemos tener en cuenta la estructura de datos que tenemos, las funcionalidades del modelo que queremos exponer, y la forma en que el usuario interactuará con ellas.

Nuestro modelo de carteras con fondos se asemeja mucho a una estructura de árbol, las carteras serían los nodos padre mientras que cada fondo sería una hoja conectado con una o varias carteras. imagen del árbol genérico.

De esta forma con un simple vistazo podemos ver la distribución de los diferentes fondos de la aplicación.

Para que la interacción con la aplicación por parte del usuario sea más sencilla, de cada nodo del árbol emergerá un menú con las diferentes opciones de cada elemento, dependiendo de si es un fondo o una cartera:

- **Fondo:**
 - **Añadir VL:** muestra un dialogo para añadir un VL al fondo.
 - **Actualizar:** muestra un dialogo para actualizar los campos del fondo.
 - **Borrar:** elimina el fondo de la base de datos.
 - **Ver Vls:** muestra una tabla con los VLS del fondo.
 - **Importar Vls:** permite importar un conjunto de VLS de un fichero .xls si este tiene el formato correcto.
 - **Exportar Fondo:** exporta el fondo a un fichero .xls en el directorio seleccionado.

■ Cartera:

- **Añadir fondo:** muestra un dialogo para añadir un fondo a la cartera.
- **Eliminar fondo:** muestra un dialogo para eliminar un fondo de una cartera.
- **Actualizar cartera:** muestra un dialogo para actualizar los campos de una cartera.
- **Borrar cartera:** elimina la cartera de la base de datos.
- **Añadir operación:** muestra un dialogo para añadir una operación a la cartera sobre un fondo determinado.
- **Ver operaciones:** muestra una tabla con las operaciones de la cartera sobre un fondo determinado.

Las funcionalidades que no se encuentran en los anteriores menús estarán disponibles en el menú principal de la aplicación.

■ Archivo:

- **Añadir fondo:** muestra un dialogo para añadir un fondo a la base de datos.
- **Añadir cartera:** muestra un dialogo para añadir una cartera a la base de datos.
- **Importar fondo:** permite importar un fondo (previamente exportado por la aplicación) de un fichero .xls.

Para poder representar correctamente las gráficas necesitaremos algunos elementos adicionales:

- Un **panel** en el que representar las gráficas.
- Un **desplegable** donde seleccionar la gráfica deseada.
- Un **campo de texto** donde incluir una pequeña descripción de lo que representa cada gráfica.
- Para las gráficas en las que necesitemos seleccionar un intervalo de tiempo determinado, debemos incluir **dos calendarios** que nos permitan acotar las fechas de inicio y fin del intervalo.
- Para la gráfica de la rentabilidad estimada, necesitamos un pequeño **campo de entrada** donde poder introducir una rentabilidad por si el usuario no quiere usar la del fondo.
- Para la gráfica de medias móviles, necesitaremos **botones** para seleccionar los intervalos de tiempo.

Una primera aproximación de la interfaz principal constará del árbol de las carteras, una barra de menú principal y los elementos necesarios para representar las gráficas.

5.2. Implementación de la interfaz

Con las necesidades de diseño que hemos establecido en el punto anterior, el siguiente paso es seleccionar los componentes para crear la interfaz en Swing.

Comenzamos por implementar la ventana principal de la aplicación, para ello, Swing nos proporciona la clase **JFrame**. Un **JFrame** es una ventana de alto nivel sobre la que podemos añadir el resto de componentes de la interfaz de manera sencilla.

Los primeros elementos que añadiremos serán la barra del menú (**JMenuBar** en Swing), que se ajusta automáticamente a la parte superior de la ventana, y el panel para las gráficas (**JPanel** en Swing).

Los componentes en Swing se alinean unos con otros mediante “*gaps*”. Estos espacios pueden ser de tamaño fijo (se respeta su tamaño aunque se redimensione la interfaz) o de tamaño variable (se expanden o se contraen si se redimensiona la interfaz).

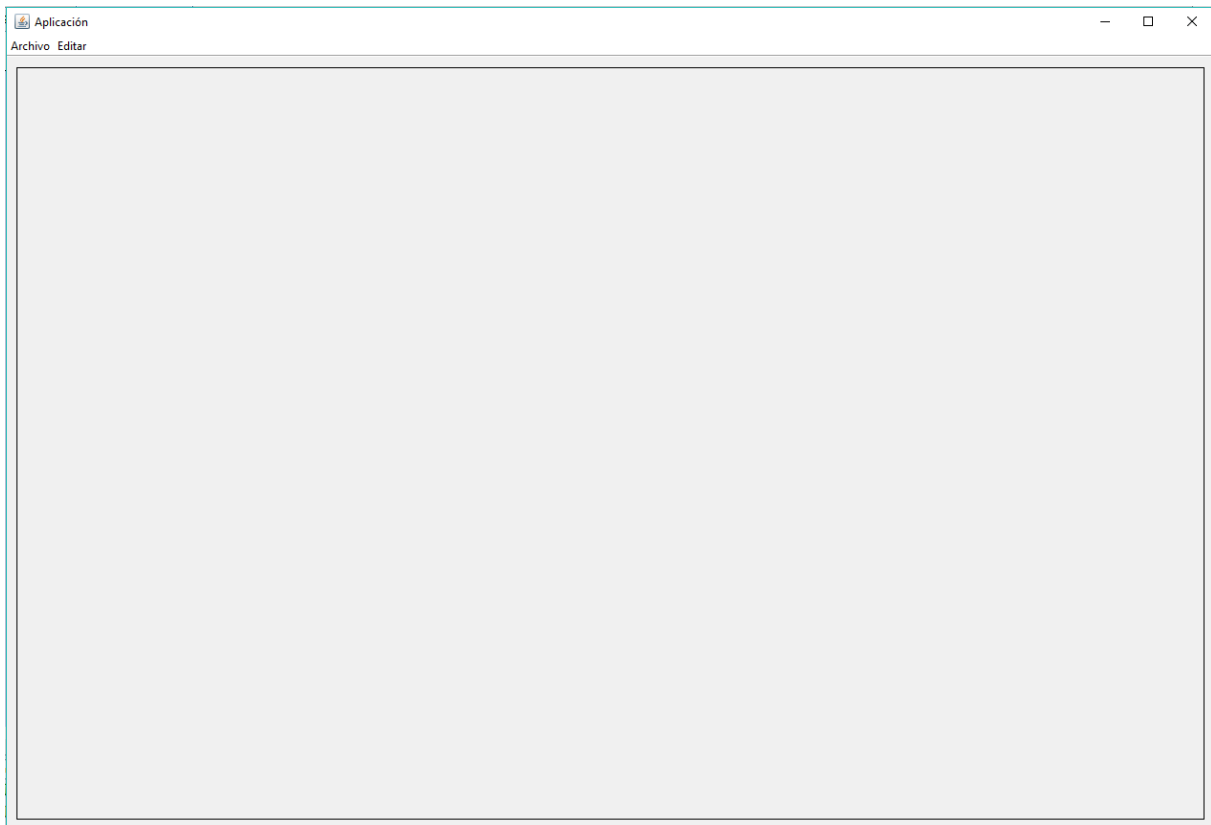


Figura 6: JFrame

Una vez creada la ventana principal, el siguiente paso es añadir el árbol para representar las carteras y los fondos. Swing nos proporciona esta estructura en la clase **JTree**, mediante la cual podemos crear un árbol con la estructura de datos deseada.

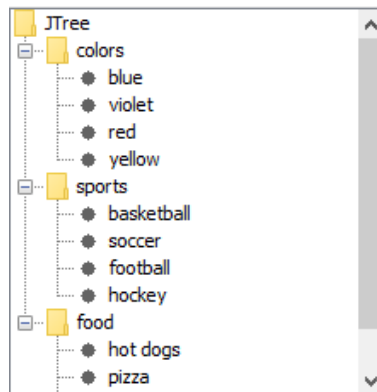


Figura 7: JTree

Como se puede observar en la figura anterior, cada uno de los nodos padre se puede expandir o contraer a voluntad del usuario. Por ello es obligatorio incluir el árbol en un contenedor que permita el desplazamiento vertical. En Swing contamos con la clase **JScrollPane** que nos permite realizar esta función.

El desplegable para seleccionar las gráficas se implementa mediante un **JComboBox**, de tal forma que el nombre de cada gráfica será un elemento de dicho desplegable. Cada componente de Swing nos permite añadirle “*action listeners*” que se encargan de capturar los eventos que van sucediendo, por ejemplo, la selección de un elemento del desplegable o del árbol, permitiéndonos reaccionar ante ellos de la forma adecuada.

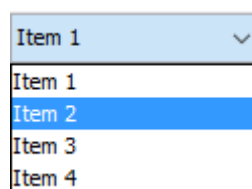


Figura 8: JComboBox

Para el campo de texto de descripción de las gráficas se utiliza un **JEditorPane**, que nos proporciona un recuadro para mostrar texto formateado. Como en el caso anterior, es necesario incluir este componente en un JScrollPane.



Figura 9: JEditorPane

El siguiente paso es añadir a la interfaz los elementos adicionales que necesitan las gráficas.

Para los calendarios de las fechas de inicio y fin de intervalo utilizamos el componente **JDatePanel**, que nos proporciona un calendario en el que el usuario puede seleccionar una fecha. En nuestra aplicación se utiliza la clase *“LocalDate”* para los datos de tipo fecha, por lo tanto para que los calendarios nos muestren las fechas en el formato correcto es necesario crear una clase que redefina los métodos *“stringToValue”* y *“valueToString”* de la clase *“AbstractFormatter”* con el formato utilizado por *“LocalDate”*: *“yyyy-MM-dd”*.



Figura 10: JDatePanel

Los botones de selección de período, utilizados para calcular las medias móviles, están formados por tres **JRadioButton** incluidos en un **ButtonGroup**. De esta manera solo se podrá seleccionar uno de los tres botones a la vez.

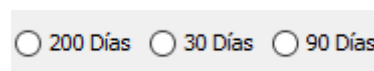


Figura 11: ButtonGroup

Para poder introducir una rentabilidad estimada, se utilizan en conjunto una **JLabel** un **JFormattedTextField** y un **JButton**. La etiqueta contiene la descripción del parámetro que se debe introducir en el campo de texto y el botón simplemente llevará a cabo los cálculos.

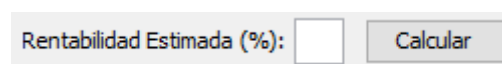


Figura 12: Rentabilidad estimada

La entrada de texto acepta entradas de números reales con dos decimales, si se introduce texto, por ejemplo, el campo lo descarta y toma el último valor válido.

Con todos estos elementos añadidos la interfaz queda de la siguiente manera:

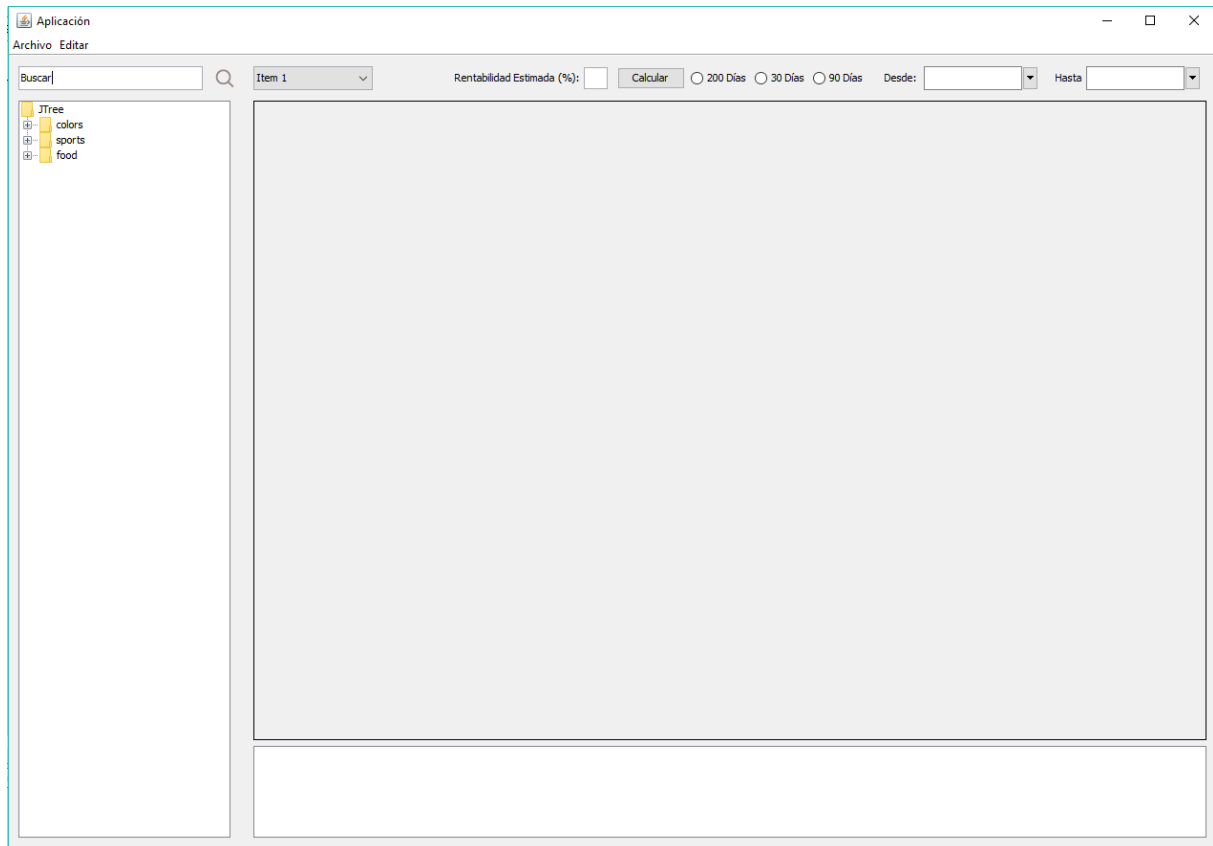


Figura 13: GUI

Swing nos permite ocultar y mostrar componentes de la interfaz bajo demanda. Dependiendo de la gráfica seleccionada las fechas, los botones o el recuadro de introducir la rentabilidad esperada se ocultarán. Los restantes elementos visibles se alinearan siempre a la derecha.

El árbol, una vez actualizado, es el contenedor de los datos de carteras y de fondos. Como se había establecido, cuando hagamos click derecho en una de las carteras o fondos, se nos mostrará un menú emergente (**JPopupMenu** en Swing) con los elementos seleccionados en el diseño (**JMenuItem** en Swing):

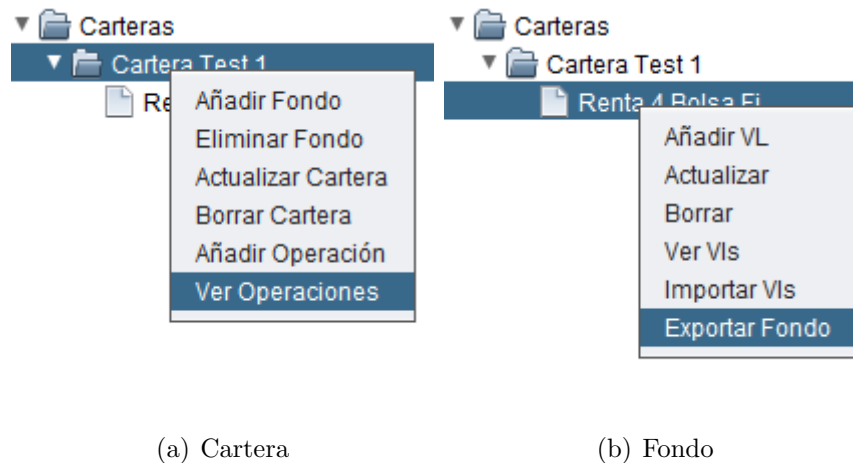


Figura 14: Popup menú

Cada elemento del popup menú crea una ventana emergente que muestra lo siguiente:

■ **Fondo:**

- **Añadir VI :** muestra un dialogo compuesto por varias etiquetas indicando los nombres de los campos de entrada, un calendario para seleccionar la fecha del VL, un campo de texto formateado para indicar el valor y dos botones para aceptar o cancelar la operación. Si al aceptar, la operación no se puede realizar, se muestra una ventana de error.
- **Actualizar:** muestra un dialogo compuesto por varias etiquetas, una para cada atributo de la clase FundDesc, y entradas de texto con los valores actuales del fondo, de esta manera el usuario puede sobrescribir cualquier campo viendo el valor que posee actualmente el fondo.
- **Borrar:** muestra un dialogo de confirmación con las opciones si o no.
- **Ver Vls:** muestra un dialogo con una tabla que contiene todos los VLs del fondo seleccionado.
- **Importar Vls:** muestra un dialogo de selección del fichero de origen. Si el fichero no tiene un formato válido se muestra una ventana de error.
- **Exportar Fondo:** muestra un dialogo de selección del fichero de destino.

■ **Cartera:**

- **Añadir fondo:** muestra un dialogo compuesto por una etiqueta indicando la cartera seleccionada, un desplegable con los fondos que todavía no han sido añadidos a la cartera y dos botones para aceptar o cancelar la operación.
- **Eliminar fondo:** muestra un dialogo compuesto por una etiqueta indicando la cartera seleccionada, un desplegable con los fondos que han sido añadidos a la cartera y dos botones para aceptar o cancelar la operación.
- **Actualizar cartera:** muestra un dialogo compuesto por dos etiquetas indicando los nombres de los campos de entrada y entradas de texto con los valores actuales del nombre y la descripción de la cartera. Si al aceptar, la operación no se puede realizar, se muestra una ventana de error.
- **Borrar cartera:** muestra un dialogo de confirmación con las opciones si o no.
- **Añadir operación:** muestra un dialogo compuesto por varias etiquetas indicando los nombres de los campos de entrada, un desplegable para seleccionar el fondo, un calendario para seleccionar la fecha de la operación, un campo de texto formateado para indicar el total de participaciones de la operación, dos botones para seleccionar si la operación es de compra o de venta y otros dos botones para aceptar o cancelar la operación. Si al aceptar, la operación no se puede realizar, se muestra una ventana de error.
- **Ver operaciones:** muestra un dialogo con una tabla que contiene todas las operaciones del fondo seleccionado.

Todos los métodos que abren un dialogo con el usuario utilizan la clase **JDialog** como ventana emergente, excepto las ventanas de error y las de selección de fichero.

Las ventanas de error y de confirmación de borrado se crean utilizando la clase **JOptionPane**, que nos proporciona una manera fácil y rápida de proveer al usuario de información, así como de solicitar confirmación.

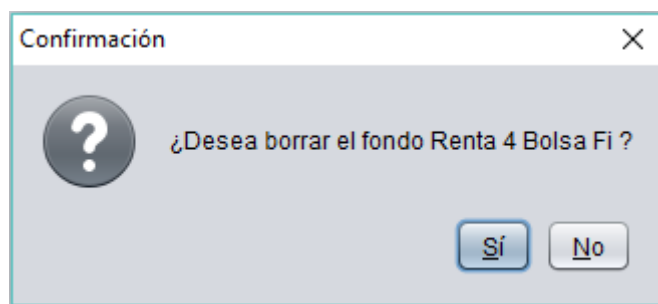


Figura 15: Ventana de confirmación

La selección de ficheros se implementa usando la clase **JFileChooser** que nos provee de un pequeño explorador de archivos para seleccionar la ruta del fichero a importar o exportar. Es posible personalizar el tipo de archivos que admite o seleccionar un nombre y extensión predeterminados a la hora de exportar los fondos.

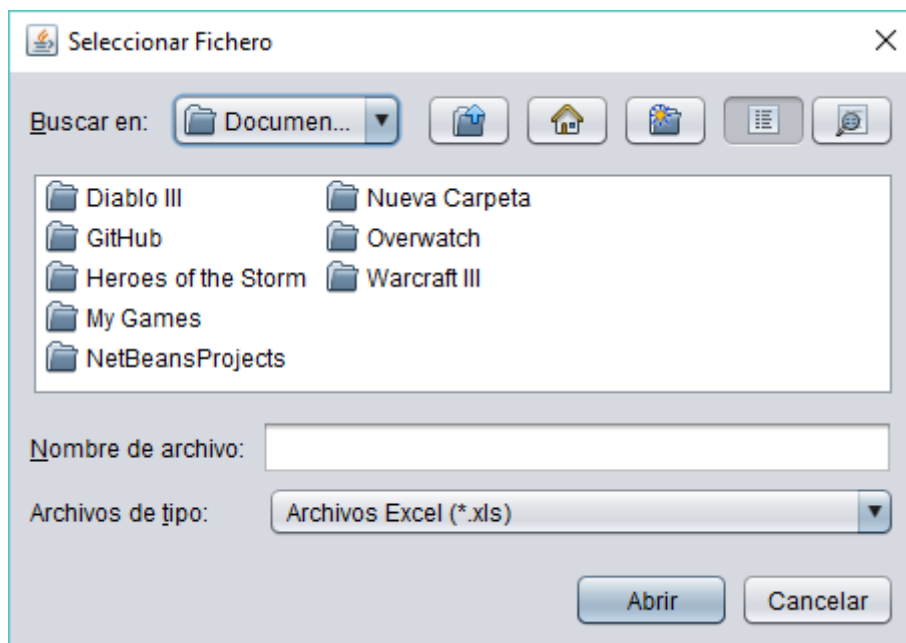


Figura 16: Ventana de selección de fichero

La tabla de VLs esta implementada mediante la clase **JTable** en un contenedor de tipo **JScrollPane**.

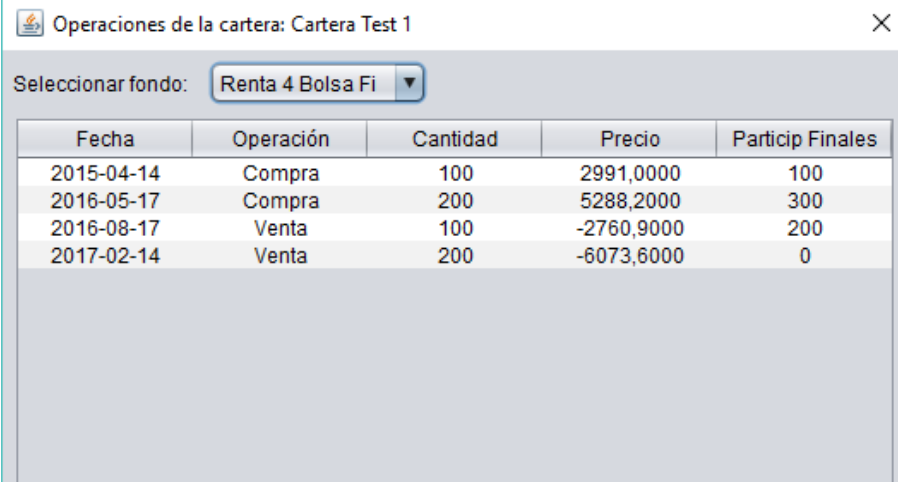
ISIN	Fecha	Valor Liquidativo	Rentabilidad (%)
ES0173394034	1993-11-12	6,0400	0.0
ES0173394034	1993-11-13	6,0700	0.4967
ES0173394034	1993-11-14	6,0700	0.0
ES0173394034	1993-11-15	6,0900	0.3295
ES0173394034	1993-11-16	6,1700	1.3136
ES0173394034	1993-11-17	6,2000	0.4862
ES0173394034	1993-11-18	6,2000	0.0
ES0173394034	1993-11-19	6,1500	-0.8065
ES0173394034	1993-11-20	6,1400	-0.1626
ES0173394034	1993-11-21	6,1400	0.0
ES0173394034	1993-11-22	6,0500	-1.4658

Figura 17: Tabla de valores liquidativos

Como se puede observar en la figura anterior, la columna de rentabilidad muestra un color distinto en función de su valor, para conseguir este efecto ha sido necesario sobrescribir el método *“prepareRenderer”* de la tabla, de tal forma que si el valor de la columna es negativo se muestre rojo y si es positivo verde.

Si seleccionamos un día concreto de la tabla y hacemos click derecho sobre él, se muestra un nuevo menú emergente que nos proporciona la posibilidad de actualizar o de borrar el VL. El dialogo de actualización es muy similar al de añadir un nuevo VL con la diferencia de que el día no puede ser modificado. Si se selecciona la operación de borrado se muestra el menú de confirmación.

La tabla de operaciones es similar a la de los VLs:



Fecha	Operación	Cantidad	Precio	Particip Finales
2015-04-14	Compra	100	2991,0000	100
2016-05-17	Compra	200	5288,2000	300
2016-08-17	Venta	100	-2760,9000	200
2017-02-14	Venta	200	-6073,6000	0

Figura 18: Tabla de operaciones

En ella se muestran todas las operaciones de la cartera para el fondo seleccionado en el desplegable. Como en el caso anterior, si seleccionamos una fila de la tabla, y hacemos click derecho sobre ella, se muestra un nuevo menú emergente que nos proporciona la posibilidad de actualizar o de borrar la operación.

Por último, el menú de archivo contiene las operaciones de añadir un fondo o una cartera e importar un fondo. Los dos primeros muestran un dialogo muy similar a sus versiones de actualizar y el último muestra el selector de ficheros. Todos ellos muestran la ventana de error si no se puede completar la operación.

6. Tercera iteración: Creación de las gráficas

Esta tercera iteración tiene como objetivo crear el conjunto de gráficas que proveerán de información al usuario acerca de los fondos y las carteras.

La librería de gráficas utilizada es JFreeChart, un marco de software open source para el lenguaje de programación Java, el cual permite la creación de gráficos complejos de forma simple.

6.1. Preparación de las gráficas

El primer paso antes de comenzar a programar las gráficas es decidir donde se van a implementar. En nuestro caso se implementan en una clase aparte llamada *“ChartMaker”*. Esta clase será la encargada de recibir los datos necesarios de la interfaz, crear la gráfica seleccionada y su descripción a partir de ellos y devolverlas a la interfaz para que esta pueda representarlas en el panel.

Para pasar la gráfica y la descripción al controlador, se ha creado una clase sencilla denominada *“Chart”*. Esta clase está compuesta por dos atributos, un *ChartPanel* que será el contenedor y un *String* con la descripción de la gráfica.

JFreeChart funciona de la siguiente manera: primero debemos crear un *“dataset”* acorde con el tipo de gráfica que queramos realizar, por ejemplo, para gráficas en forma de tarta se utiliza *“DefaultPieDataset”* y para gráficas de barras *“DefaultCategoryDataset”*.

A continuación, se insertan los datos en el dataset, normalmente mediante el método *“dataset.addValue()”* o *“dataset.setValue()”*. Estos métodos utilizan parámetros distintos dependiendo del tipo de dataset, por ejemplo para las gráficas en forma de tarta se le pasan los siguientes campos.

```
pie_chart_dataset.setValue(nombre, valor);
```

Una vez añadidos todos los valores se invoca a *“ChartFactory”*, en este caso:

```
JFreeChart chart = ChartFactory.createPieChart(nombre de la gráfica,  
pie_chart_dataset, leyenda, herramientas, urls);
```

Donde los campos leyenda, herramientas y urls son de tipo booleano para indicar si se deben incluir dichos elementos en la gráfica.

La llamada a *“ChartFactory”* nos devuelve la gráfica creada, ahora solamente necesitamos crear un nuevo *ChartPanel* con la gráfica y añadirlo al panel de la interfaz.

```
ChartPanel cP = new ChartPanel(chart);  
panelGraficas.add(cp);
```


6.2. Gráficas de carteras

6.2.1. Distribución de capital de la cartera

Esta primera gráfica tiene como objetivo, que el usuario pueda ver la distribución del capital invertido en los diferentes fondos de una cartera. Para ello se utiliza una gráfica en forma de tarta en la cual cada una de sus porciones representa el tanto por ciento del capital de la cartera que está invertido en un fondo.

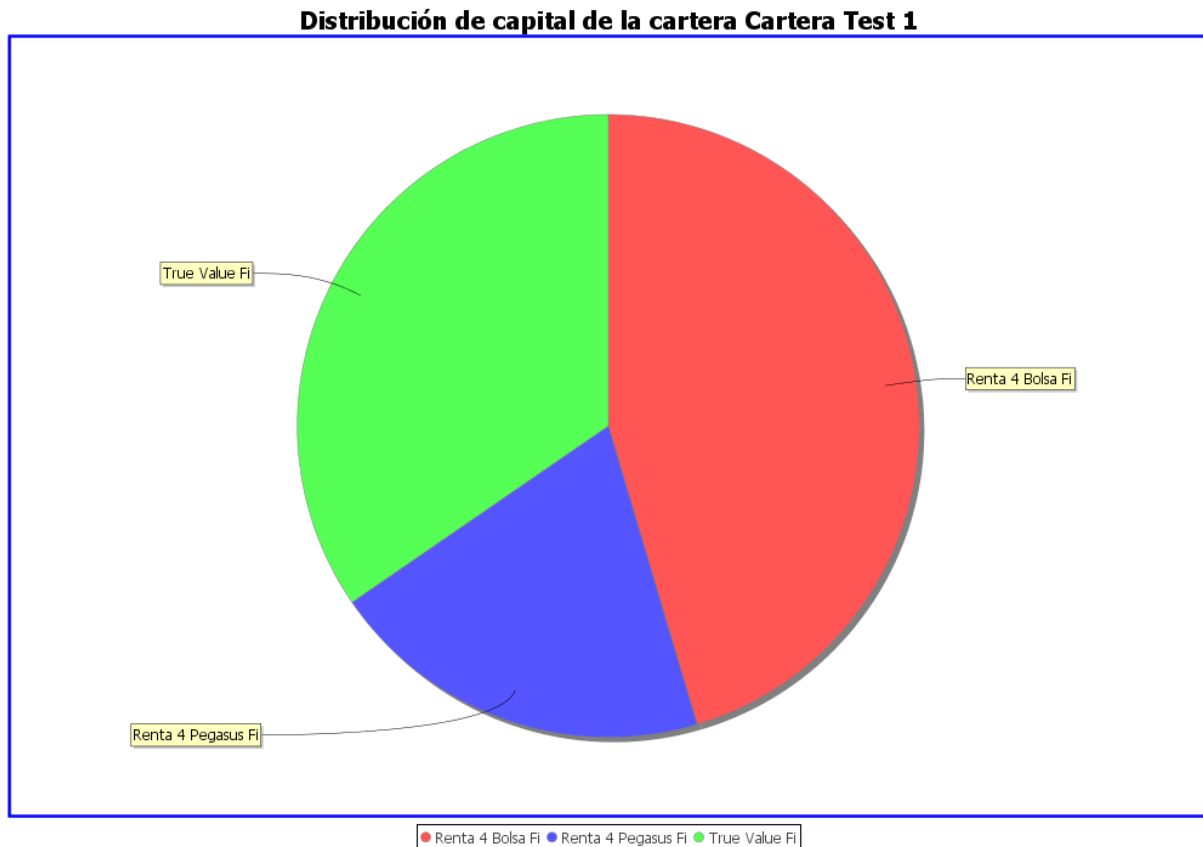


Figura 19: Gráfica de distribución de capital de la cartera

Para el cálculo de esta gráfica se toma el valor del número de participaciones de cada fondo y se multiplica por el valor de cada una a día de hoy, si no existe un VL en el día de hoy se utiliza el más próximo.

6.2.2. Fondos normalizados de la cartera

Esta gráfica tiene como objetivo mostrar un histórico de los VLs de todos los fondos de la cartera partiendo todos de la misma base, en este caso 100. De esta manera es posible comparar los fondos entre ellos y ver cuales han alcanzado una mayor rentabilidad. Para ello se utiliza una gráfica de serie temporal. Este tipo de gráficas utilizan valores de tipo temporal en el eje de abscisas y valores de tipo double en el de ordenadas.

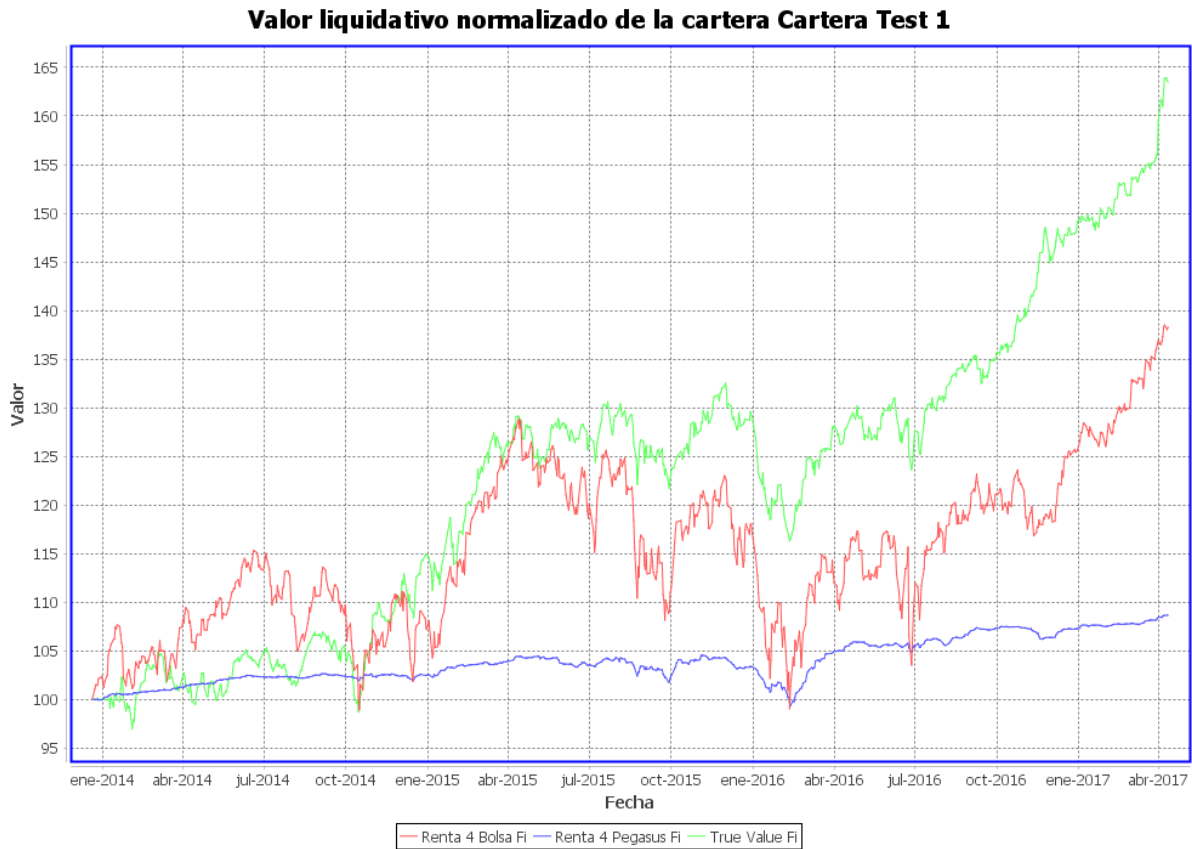


Figura 20: Gráfica de fondos normalizados de la cartera

Para el cálculo de está gráfica supongamos que el fondo tiene los siguientes valores liquidativos VL_1, VL_2, \dots, VL_d para los días $1, 2, \dots, d$, y que queremos calcular sus VLs normalizados $VLN_1, VLN_2, \dots, VLN_d$ para que comiencen en 100 . Los valores se calculan de la siguiente manera.

$$VLN_i = VL_i \left(\frac{100}{VL_1} \right) \quad (3)$$

6.2.3. Fondos más y menos rentables de la cartera

Esta gráfica tiene como objetivo mostrar los cinco fondos más y menos rentables de una cartera. De esta manera es posible ver los mejores y los peores fondos de la cartera y tomar decisiones acerca del capital invertido en ellos, como por ejemplo, sacar fondos del peor y comprar participaciones del mejor con ellos.

Para visualizar los datos se utiliza una gráfica de barras donde cada barra corresponde a la rentabilidad del fondo en la cartera.

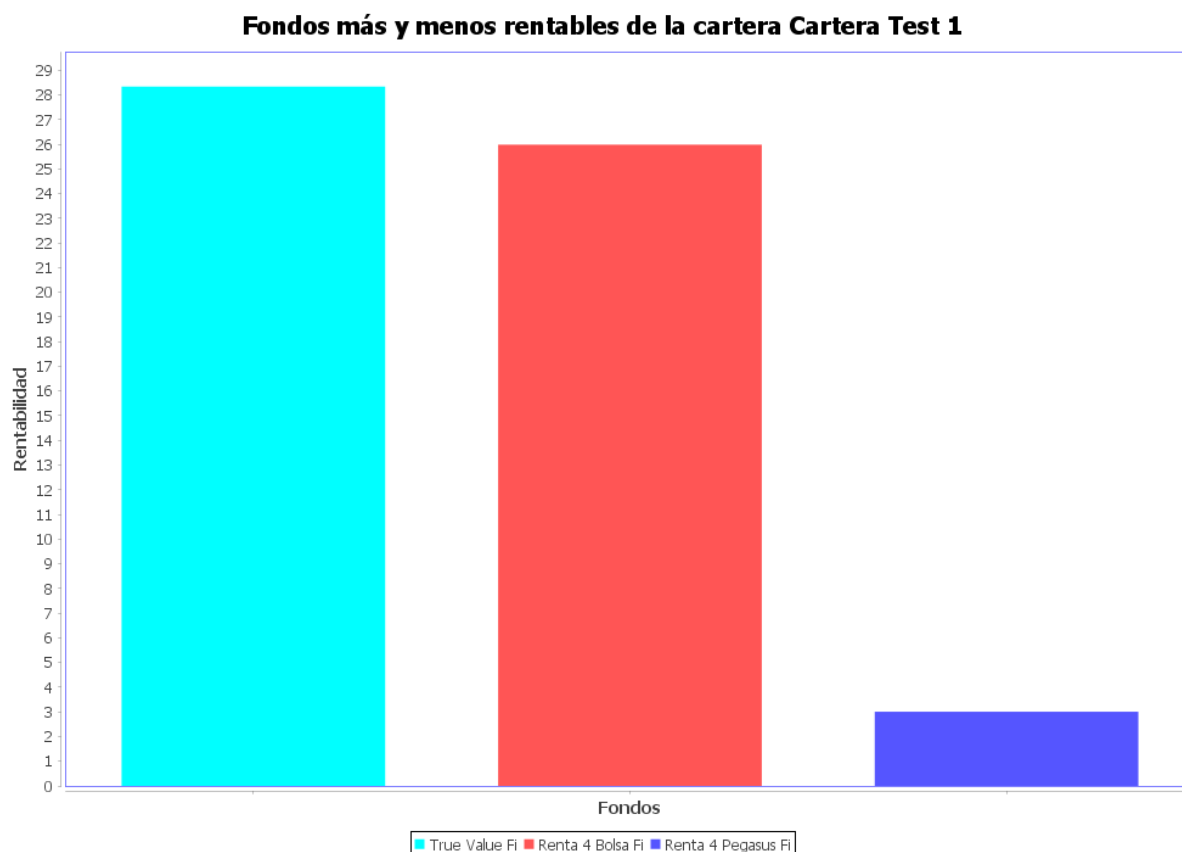


Figura 21: Gráfica de los fondos más rentables de la cartera

Para calcular esta gráfica se tienen en cuenta la rentabilidad de todas las operaciones realizadas en la cartera, es decir, se calcula la rentabilidad entre el coste total de compras de participaciones y la remuneración de las ventas. Si las participaciones todavía no se han vendido entonces se calcula su valor a día de hoy y se suma al valor de venta.

En este caso nuestra cartera de prueba solo tiene tres fondos por lo que son, a la vez, los más y menos rentables de la cartera.

Es posible que alguno de los fondos menos rentables tengan una rentabilidad negativa. Si esto ocurre, simplemente la barra estará en sentido inverso.

6.2.4. Comparativa de inversión

Esta gráfica tiene como objetivo mostrar una comparativa entre el valor de nuestras participaciones en la última operación realizada y el valor que poseen actualmente. Con esta información podemos ver rápidamente si el capital invertido en un determinado fondo ha crecido o menguado desde el último movimiento de participaciones.

Para visualizar los datos se utiliza una gráfica de barras, cada categoría es un fondo, la barra roja indica el valor de las participaciones en la última operación y la barra roja el valor actual.

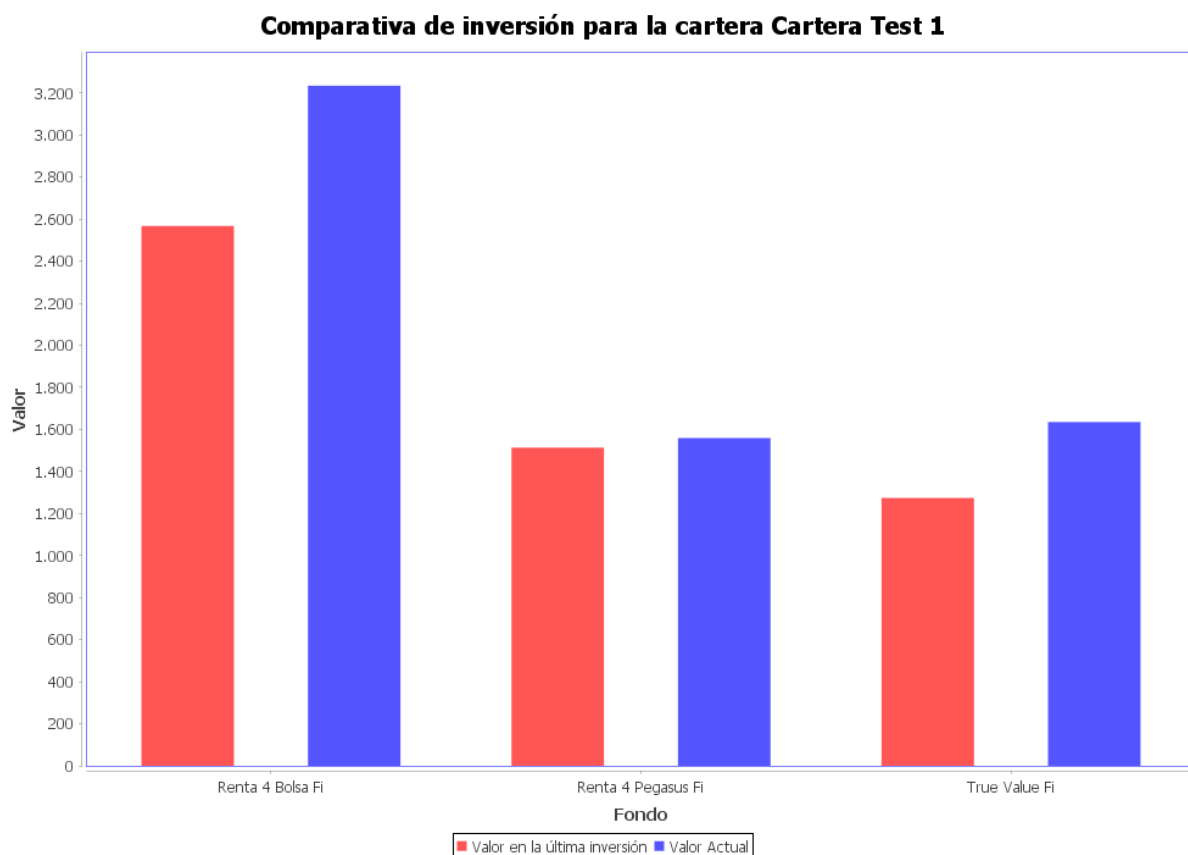


Figura 22: Gráfica de la comparativa de inversión de la cartera

Para calcular esta gráfica simplemente deberemos obtener las participaciones finales de la última operación, su valor entonces y su valor actual.

Si un fondo de la cartera no tiene ninguna operación registrada o se han liquidado todas sus participaciones no aparecerá en esta gráfica.

6.2.5. Rentabilidad total de la cartera

Esta gráfica tiene como objetivo mostrar la rentabilidad total de la cartera en un período de tiempo determinado. Con esta información podemos comprobar como evoluciona en el tiempo el conjunto de nuestras inversiones.

Para visualizar los datos se utiliza una gráfica de serie temporal.

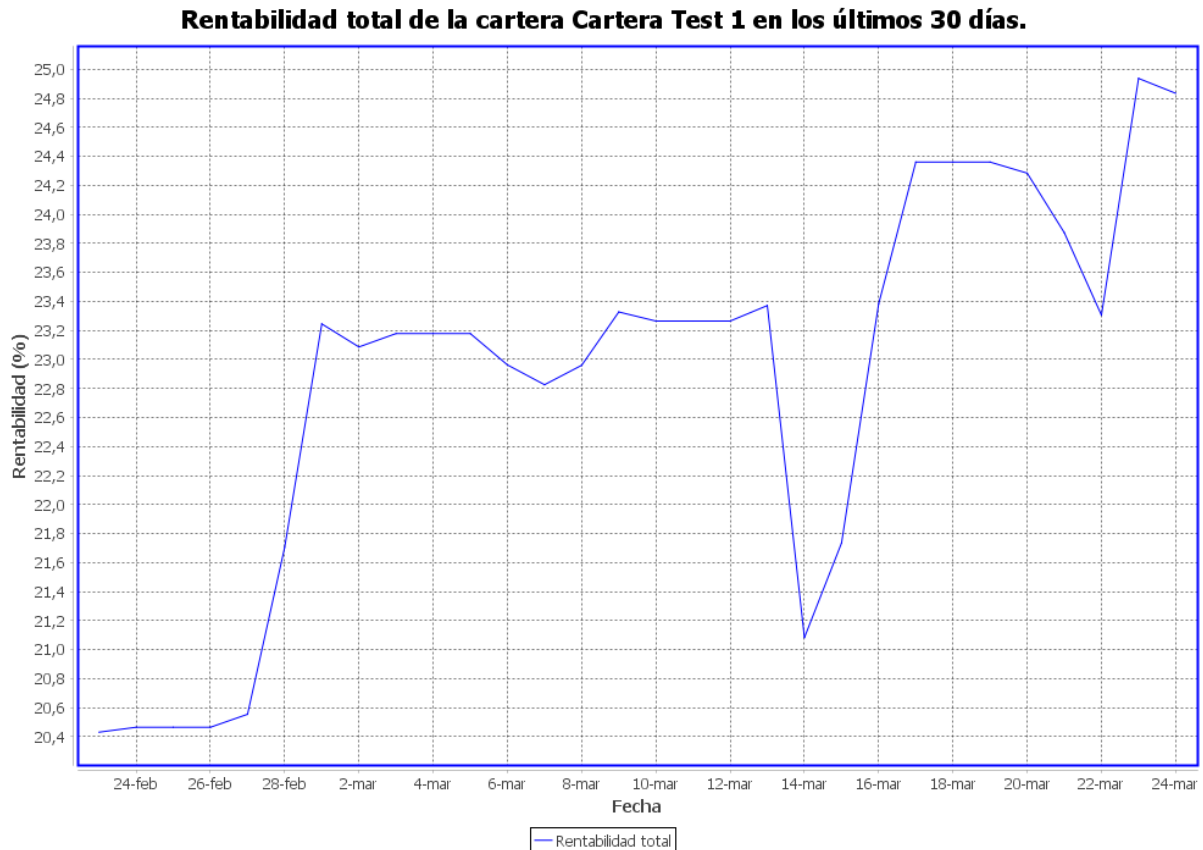


Figura 23: Gráfica de la rentabilidad en conjunto de la cartera

Para calcular esta gráfica se calcula la rentabilidad de todos los fondos en función de las operaciones realizadas para cada día de la gráfica y luego se suman, obteniendo el total. Si para un día concreto tenemos participaciones de un fondo en la cartera, su precio se calcula multiplicándolas por el VL más cercano.

6.3. Gráficas de fondos

6.3.1. Historial del Valor Liquidativo

Esta gráfica tiene como objetivo mostrar un histórico de los VLs del fondo seleccionado. Para ello se utiliza una gráfica de serie temporal. Este tipo de gráficas utilizan valores de tipo temporal en el eje de abscisas y valores de tipo double en el de ordenadas.

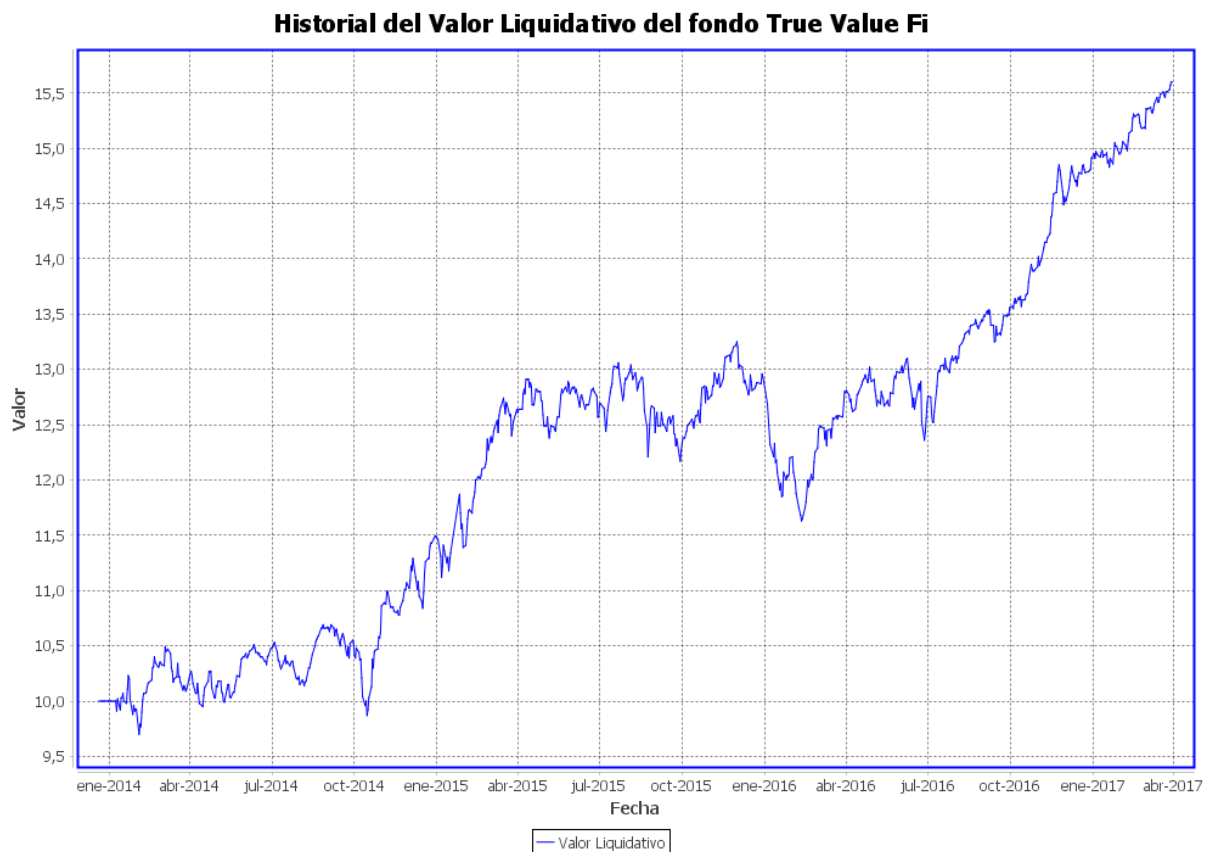


Figura 24: Gráfica historial del Valor Liquidativo

El cálculo de esta gráfica es muy sencillo, basta con incluir en ella los VLs del fondo en el período seleccionado.

6.3.2. Historial de rentabilidades

Esta gráfica tiene como objetivo mostrar las rentabilidades históricas del fondo para los siguientes períodos de tiempo: último año fiscal, último semestre, último trimestre y último mes. Con esta información podemos ver como ha evolucionado el fondo en los últimos períodos de tiempo, y comprobar si ha tenido variaciones o si ha seguido una tendencia.

Para visualizar los datos se utiliza una gráfica de barras, cada categoría es un período de tiempo y la barra indica la rentabilidad en dicho período.

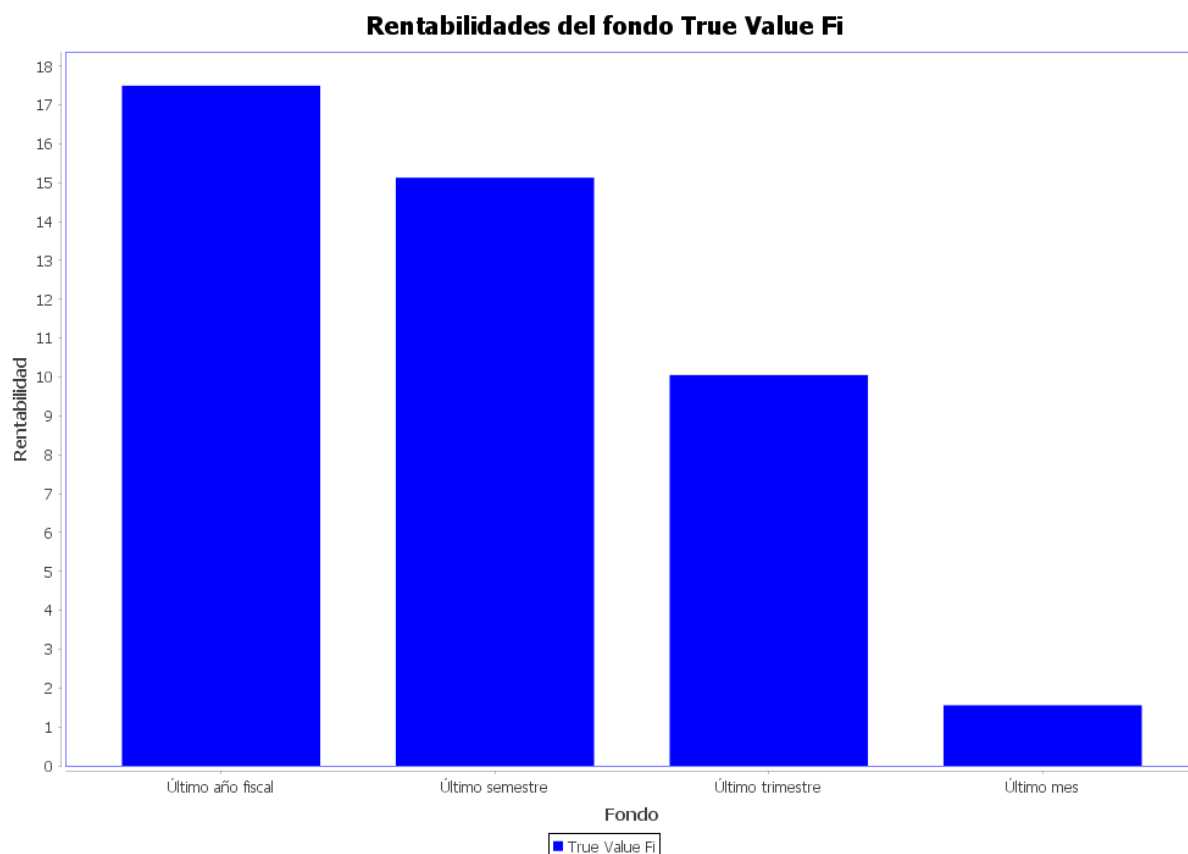


Figura 25: Historial de rentabilidades

Para calcular esta gráfica obtenemos los VL más próximos a las fechas de inicio y fin del período, si existe la suficiente distancia entre ellas calculamos sus rentabilidades, si por el contrario no hay suficientes días entre ellos obviamos ese período y lo indicamos en la descripción.

6.3.3. Rentabilidad esperada

Esta gráfica tiene como objetivo mostrar un histórico de los VLs del fondo seleccionado y los VL esperados en función de la rentabilidad media diaria. Con esta información podemos ver una estimación de los VLs del fondo y comparar los reales con los estimados.

Para visualizar los datos se utiliza una gráfica de serie temporal.

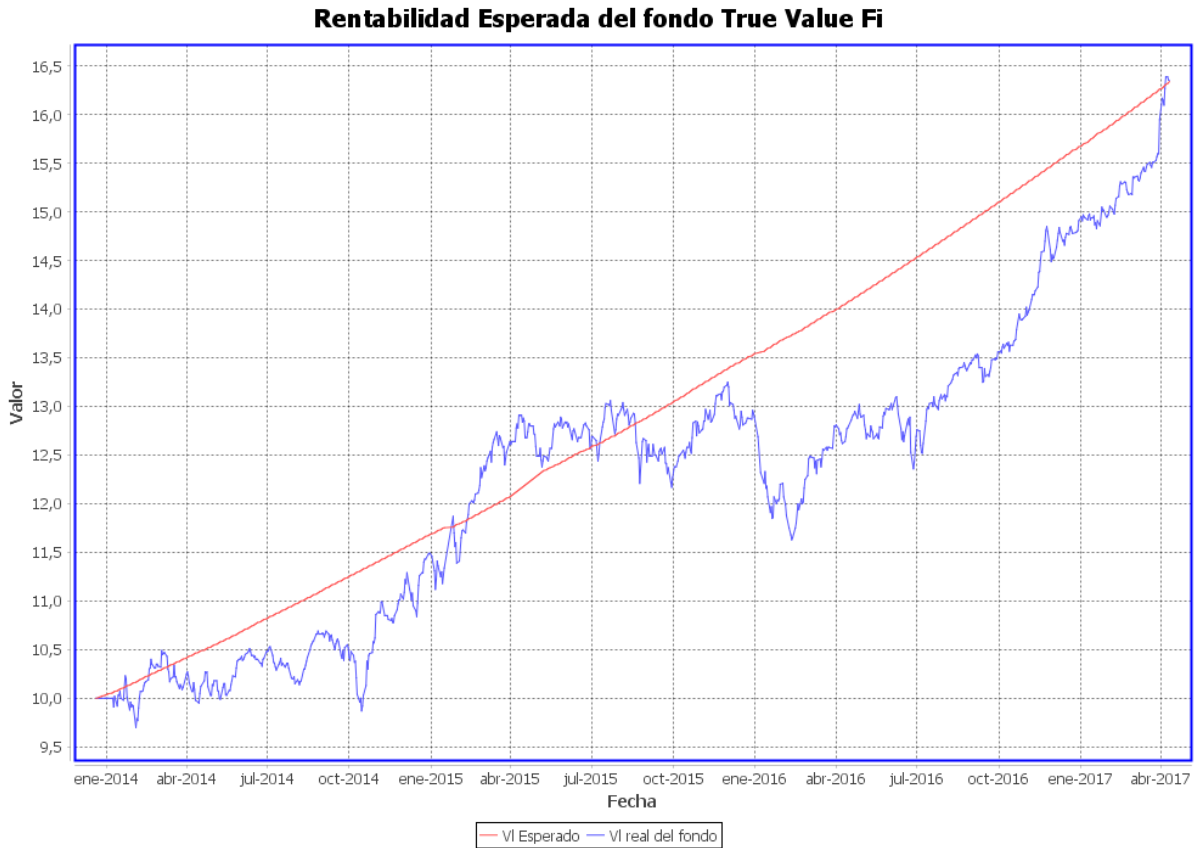


Figura 26: Rentabilidad esperada

Para calcular esta gráfica se utiliza la rentabilidad media diaria r . Este valor nos indica la variación diaria teniendo en cuenta todos los datos disponibles del fondo. Para calcularla se utilizan los valores liquidativos más antiguo y más reciente, esto es, VL_I y VL_F , respectivamente.

$$r = \left(\frac{VL_F}{VL_I} \right)^{1/d} - 1 \quad (4)$$

Donde d es el número de días transcurridos entre los valores liquidativos inicial y final.

Una vez calculado r se añade el primer valor a la gráfica, el resto se calculan de la siguiente manera.

$$e_i = e_{i-1} (1 + r), \quad \text{para } i > 1 \quad (5)$$

6.3.4. Medias Móviles

Esta gráfica tiene como objetivo mostrar un histórico de los VLs del fondo seleccionado y su media móvil a un determinado intervalo de tiempo. Una de las grandes ventajas de usar medias móviles es que se mueven en la dirección de la tendencia, lo que permite que avancen los beneficios y se frenen las pérdidas. Pero hay que tener en cuenta que su uso es para seguir la pista de la tendencia, la media no anticipa nada.

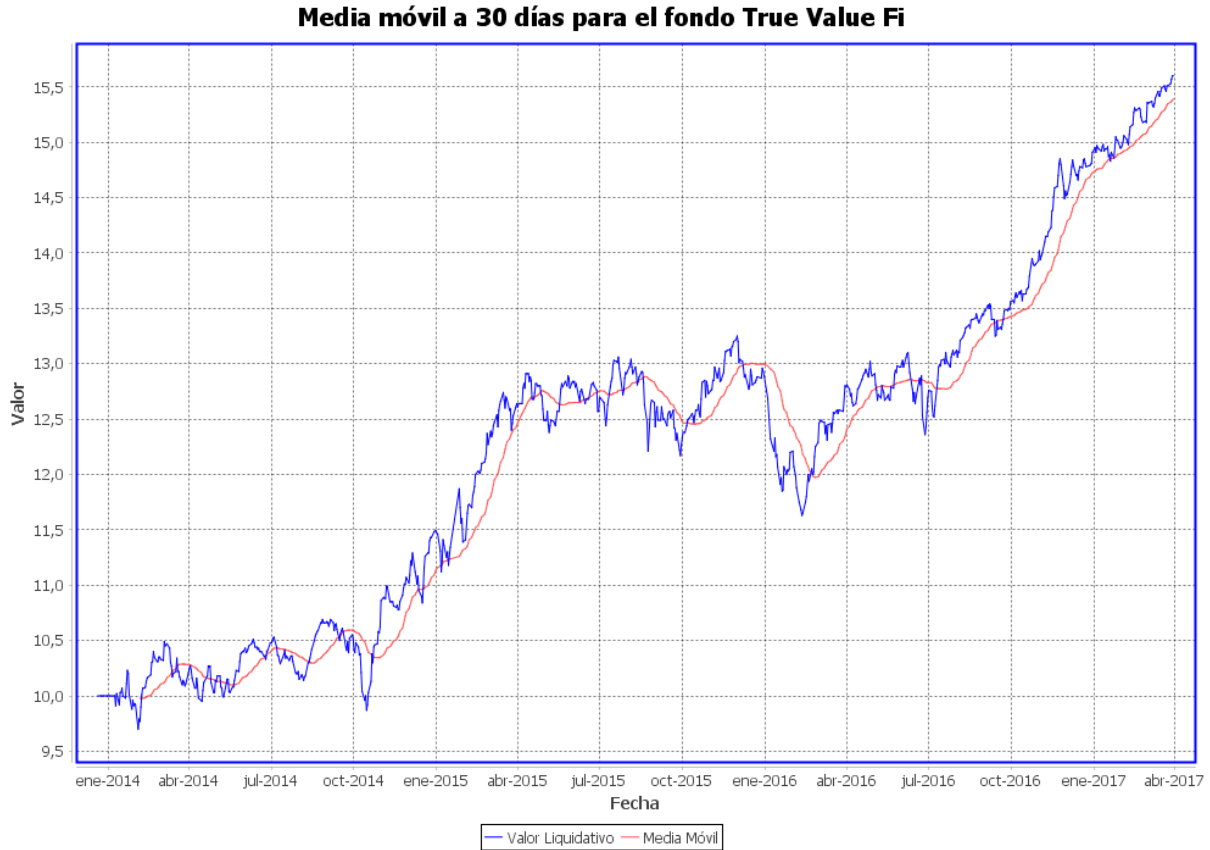


Figura 27: Media Móvil

Para calcular esta gráfica supongamos el fondo tiene los siguientes valores liquidativos VL_1, VL_2, \dots, VL_d para los días $1, 2, \dots, d$, y que queremos calcular la media móvil para m días. En ese caso, tendremos una secuencia de $d - m + 1$ valores y_m, y_{m+1}, \dots, y_d . Los primeros $m - 1$ valores de la secuencia y se pueden poner a 0. El resto de valores se calculan de la siguiente manera:

$$y_j = \frac{1}{m} \left(\sum_{i=j-m+1}^j VL_i \right) \quad (6)$$

De forma más eficiente, suponiendo que se ha calculado el primer valor de la serie, y_m , tenemos:

$$y_j = \frac{1}{m} (m \cdot y_{j-1} + VL_j - VL_{j-m}), \quad \text{para } j > m \quad (7)$$

7. Cuarta iteración: Ejecución de tareas en segundo plano

Esta cuarta iteración tiene como objetivo implementar la ejecución en segundo plano de las tareas que requieren un cierto tiempo de procesamiento.

Para ello, Swing nos proporciona **SwingWorker**, una clase abstracta para realizar interacciones con la GUI en un thread en segundo plano.

7.1. SwingWorker

Cuando construimos una aplicación multi-thread utilizando Swing, debemos tener en cuenta dos restricciones:

- Las tareas que consumen tiempo no deberían correr en el Event Dispatch Thread (EDT), pues la aplicación se quedaría congelada.
- Los componentes de swing únicamente deberían ser accesibles en el EDT.

Estas restricciones significan que una GUI con tareas que requieren un tiempo de cómputo grande necesitan por lo menos dos threads: uno para realizar las tareas largas y otro, el EDT, para las actividades relacionadas con los componentes de la interfaz.

Podríamos realizar esta tarea a mano, usando la clase Thread para crear un hilo cada vez que realizemos una tarea que requiera tiempo, pero desde java 6 tenemos SwingWorker.

SwingWorker está diseñado para situaciones en las que se necesita tener una tarea larga corriendo en segundo plano y proveer actualizaciones a la GUI acerca de si ha terminado, o sigue en proceso.

La clase SwingWorker es una clase abstracta en la que debemos definir dos tipos de datos: `Class SwingWorker<T,V>`. T representa el tipo de objeto devuelto cuando el worker termina la tarea, mientras que V es el tipo de información que usara el worker para informar de su progreso.

Todas las subclases de SwingWorker deben implementar el método `doInBackground()`, donde se implementará la tarea en segundo plano, y el método `done()`, en el que se muestra el resultado de la operación en pantalla. SwingWorker se encargará de que ambos métodos se ejecuten en los threads adecuados.

Cuando creamos un SwingWorker debemos pasar como parámetros aquellos elementos que queramos modificar cuando se complete la operación, y modificarlos unicamente en el método `done()`. En este último método también se capturan las excepciones ocurridas durante el transcurso de `doInBackground()`.

Para ejecutar un worker, basta con llamar a su método `execute()`. Esta llamada crea el nuevo thread y devuelve el control inmediatamente.

Si queremos informar del progreso de la operación, en el método `doInBackground()` podemos llamar a `setProgress()` (un método que admite un entero entre 0 y 100) o `publish()` (que utilizara el tipo de dato que hayamos definido en `V`). Ambos métodos notifican a los `PropertyChangeListener` con un evento denominado `progress`, que podemos capturar y utilizar para mostrar al usuario o actualizar una barra de progreso.

Existen tres threads involucrados en el ciclo de vida de un `SwingWorker`:

- El thread actual: Es donde se llama el método `execute()`. Esto programa a `SwingWorker` para la ejecución en segundo plano de la tarea en lo que se denomina `worker thread`, y vuelve al hilo actual. Podemos esperar a que se complete dicha operación con el método `get()`, lo que bloquearía la interfaz.
- El worker thread: el método `doInBackground()` se llama en este thread. En él se llevan a cabo las tareas necesarias y se notifica a los `PropertyChangeListeners` acerca del estado de la tarea.
- El EDT: todas las actividades relacionadas con Swing ocurren en este thread. `SwingWorker` invoca el método `done()` y notifica a cualquier `PropertyChangeListener` en este hilo.

Normalmente el thread actual se corresponde con el EDT.

Antes de que se ejecute el método `doInBackground`, `SwingWorker` notifica a todos los `PropertyChangeListeners` acerca del cambio de estado a `StateValue.STARTED`. Una vez termina `doInBackground` y se ejecuta el método `done`, se notifica que el estado pasa a `StateValue.DONE`.

En última instancia hemos de tener en cuenta que:

- Si por alguna razón queremos cancelar un worker en ejecución, es necesario llamar al método `cancel(true)`, que interrumpe la operación.
- `SwingWorker` está diseñado para ser ejecutado una única vez. Ejecutar el mismo worker una segunda vez no tendrá ningún efecto.

7.2. Implementación de SwingWorker

Existen tres tareas en nuestra aplicación que pueden tardar un tiempo suficiente en realizarse como para que el usuario perciba que la interfaz se ha quedado bloqueada: Importar un fondo o los VLs de un fondo, crear las gráficas y actualizar el árbol de carteras.

7.2.1. ChartWorker

Hasta ahora la clase encargada de crear las gráficas era “*ChartMaker*”, una clase a la que le pasábamos los parámetros necesarios y nos devolvía un objeto de tipo **Chart**, el cual contenía la gráfica y una descripción.

Para poder ejecutar las gráficas en segundo plano y tener información acerca de su progreso, ha sido necesario modificar esta clase de tal manera que herede de **SwingWorker**, de esta manera podemos redefinir los métodos `doInBackground()` y `done()`. Para indicar que esta clase se ejecutará en un worker thread se ha cambiado su nombre a “*ChartWorker*”.

Como hemos definido en el punto anterior, la tarea que se realizará en segundo plano es la contenida en el método `doInBackground()`, con el objetivo de evitar el tener que crear una clase worker para cada gráfica, se ha creado un constructor genérico donde se proporcionan los elementos de la interfaz que se actualizan cuando la gráfica está terminada (cuando se llama al método `done()`).

Para seleccionar la gráfica que queremos crear debemos llamar a un método de la clase y pasarle los valores necesarios para crear esa gráfica en concreto. Por ejemplo, para crear la gráfica de distribución del capital primero deberemos llamar a `setPortfolioDistributionChart()` donde indicamos la cartera de la cual queremos obtener la gráfica y a continuación al método `execute()`.

Cuando la tarea termina, el resultado obtenido es exactamente el mismo que antes, un objeto **Chart** con la gráfica y la descripción. Sin embargo, el panel y el cuadro de la descripción se actualizan directamente en el método `done()`, en lugar de en la clase principal, para asegurarnos que los componentes siempre se actualizan en el EDT una vez ha terminado la operación. Si por cualquier motivo falla la ejecución de la tarea, en el método `done()` capturamos la excepción *ExecutionException* y reseteamos el panel de las gráficas.

Normalmente en un equipo medio y para fondos normales (el fondo más extenso que se ha utilizado cuenta con alrededor de 7000 valores desde 1993) todas las gráficas excepto la de la rentabilidad total de la cartera no tardan más de medio segundo en crearse. Esta última en cambio tarda alrededor de unos cinco segundos porque tiene que calcular un gran número de operaciones y será la única que reporte su progreso mediante el método `setProgress()`.

Para poder mostrar el progreso en la barra de la ventana principal es necesario añadir un `PropertyChangeListener` que reaccione ante el evento `progress` cuando creamos el worker. De esta manera la barra se actualizará conforme avance la tarea indicando al usuario el progreso.

Durante la realización de la gráfica es posible que el usuario cambie a otra o realice otra operación sin esperar a que esta termine. Si esto sucede, el usuario podría estar viendo ya otra gráfica y de repente se le actualizaría el panel con la rentabilidad total de la cartera. Para evitar esta situación, cada vez que se seleccione otra gráfica mientras se está calculando la rentabilidad, cancelaremos la operación pendiente mediante el método `cancel(true)`.

7.2.2. NodesWorker

La segunda tarea que queremos ejecutar en segundo plano es la de actualizar el árbol de carteras.

Esta operación estaba realizada por la función `createNodes()`, que solicitaba los datos de carteras y fondos a la base de datos y construía el árbol. Para ejecutar esta tarea en un worker thread, se ha creado una nueva clase denominada `NodesWorker`.

El método `doInBackground()` realiza la misma función que `createNodes()`. Al actualizar el modelo del árbol con los datos nuevos este se contrae por defecto y el último elemento seleccionado se des selecciona. Para evitar esto, en el método `done()` se ha establecido un comportamiento para el último elemento:

- Si el nodo aún existe, se selecciona de nuevo y se expande el recorrido hasta él.
- Si el nodo ya no existe, se selecciona su nodo padre.

La operación informa periódicamente del progreso mediante el método `setProgress()`, por lo que, al igual que en el caso anterior, es necesario añadir un `PropertyChangeListener` que reaccione ante el evento `progress` cuando creamos el worker, pero en este caso será la barra inferior de progreso la que se actualice a medida que se actualiza el árbol.

7.2.3. ImportFundWorker

Por último, nos queda la tarea de importar fondos o VLs.

El funcionamiento de este worker es muy similar al de las gráficas. De nuevo se ha creado un constructor genérico donde se proporcionan los elementos de la interfaz que se actualizan cuando la operación está terminada. A mayores se le proporciona una instancia de un `NodesWorker`, de esta manera es posible actualizar el modelo una vez se termine de importar.

Para seleccionar la operación que queremos utilizar debemos llamar a un método de la clase y pasarle los valores necesarios para realizar esa operación en concreto. Por ejemplo, para importar los VLs de un fondo, debemos proporcionar el fondo en el que se importarán y el formato de la fecha del fichero.

Cuando importamos los VLs de un fondo comprobamos si existe un valor para el día de cada uno en la base de datos. Si existe, se actualiza el que tenemos, sino, se añade el nuevo. En este caso, como en los anteriores, se utiliza `setProgress()` para indicar el progreso de la operación.

Cuando importamos un fondo, el servicio del modelo nos devuelve el fondo contenido en el fichero junto con sus VLs asociados. En este caso no es posible conocer el progreso de la operación en un momento determinado, por lo que utilizamos el método `setIndeterminate()` de la barra de progreso para indicar que se está realizando una operación.

Mientras se está importando un fondo o sus VLs, se bloquean los botones de importar fondo e importar VLs.

Capítulo V

Bibliografía

Referencias

[CNMV] *Comisión Nacional del Mercado de Valores.*
<https://www.cnmv.es/>

[Rankia] *Todo lo que hay que saber de Fondos de inversión en un único artículo.*
<http://www.rankia.com/blog/fondos-inversion/3208096-todo-que-hay-saber-fondos-inversion-unico-articulo>

[Rankia] *¿Qué es un fondo de inversión y cómo funciona?.*
<http://www.rankia.com/blog/fondos-inversion/952310-que-fondo-inversion-como-funciona>

[Wikipedia] *Proceso Unificado de Desarrollo Software.*
https://es.wikipedia.org/wiki/Proceso_unificado

Acrónimos

ER Modelo Entidad-Relación. 12, 13

GUI Interfaz Gráfica de Usuario. 21, 41

IDE Entornos de Desarrollo Integrado. 7

ISIN International Securities Identification Number. 3, 12, 14, 18

POM Project Object Model. 7

PUD Proceso Unificado de Desarrollo Software. 7, 9, 10

TFG Trabajo Fin de Grado. 10

UML Lenguaje Unificado de Modelado. 7, 9, 14

VL Valor Liquidativo. 3–6, 12, 13, 17–21, 27, 29, 30, 32, 33, 36–40

XML eXtensible Markup Language. 8