**Título del Trabajo Fin de Grado o de la memoria de un trabajo experimental**

**Resumen:**

**El resumen contendrá las ideas principales del trabajo realizado y no superará las seis líneas [Garamond 12, negrita, interlineado 1.5]**

**Puede contener un subtítulo.**

**TRABAJO FIN DE GRADO**

**Nombre: César**

**Apellidos: Valdés Martínez**

**D.N.I.: 48204413-R**

**Correo electrónico: cesaramvm@gmail.com**

**Director: Nombre y apellidos del director (imprescindible presentar autorización de defensa firmado)**

**Doble Grado en Ingeniería Informática e Ingeniería del Software**

**Curso: 2016/2017 – convocatoria: noviembre/marzo/junio**

**ÍNDICE**

[1. INTRODUCCIÓN (<10 págs) - 1 -](#_Toc484453322)

[2. OBJETIVOS (<1 pág) - 7 -](#_Toc484453323)

[3. DESCRIPCIÓN ALGORÍTMICA (<15 págs. 7-8 para cada cosa) - 8 -](#_Toc484453324)

[3.1. Redes neuronales - 8 -](#_Toc484453325)

[3.2. Metaheurística - 11 -](#_Toc484453326)

[4. DESCRIPCIÓN INFORMÁTICA (<15 págs) - 12 -](#_Toc484453327)

[4.1. Lenguajes y programas. - 12 -](#_Toc484453328)

[4.2. Librerías y herramientas - 15 -](#_Toc484453329)

[4.3. Metodología de desarrollo software - 17 -](#_Toc484453330)

[4.4. Especificación de requisitos - 20 -](#_Toc484453331)

[4.5. Estructura del software - 21 -](#_Toc484453332)

[5. EXPERIMENTOS (<10 págs) - 31 -](#_Toc484453333)

[6. CONCLUSIONES Y TRABAJOS FUTUROS (2 págs) - 32 -](#_Toc484453334)

[6.1. 1.1 Epígrafe [Garamond 12, negrita, 1.5 líneas, minúsculas, espacio antes 24 puntos] - 33 -](#_Toc484453335)

# DESCRIPCIÓN ALGORÍTMICA (<15 págs. 7-8 para cada cosa)

Este experimento, estos datos, lo que he probado… Explicar todo lo que no sean números, de implementación nada. Funciones de activación, etc. etc.

COMO MUCHO LLEGAR A PSEUDOCÓDIGO, quizás mejor diagrama de flujo…

Hablar un poco de redes neuronales, se inspiran en cerebro humano, poner una neurona, dendrita… Hay varios tipos perceptrón, bla bla, nos centramos en la multilayer.

Después descripción particular de la red, con tantas neuronas de entrada, 1 de salida, tantas capas internas, como he llegado.

## Metaheurística

Una metaheurística es

Hablar un poco de metah: técnicas algorítmicas para resolver problemas complejos, hay unas cuantas, bla bla.

Y la concreta que yo he implementado GRASP, desde el puntod e vista algoritmico.

Explicar el constructivo y la búsqueda local

# DESCRIPCIÓN INFORMÁTICA (<15 págs)

Todos los experimentos y codificación se han realizado en dos ordenadores:

* Sobremesa:
  + Procesador: AMD FX-8350 4.0GHz
  + RAM: 16GB DDR3

Estos datos son meramente informativos ya que el objetivo del proyecto es el desarrollo de un algoritmo que resuelva el problema planteado al inicio. Cualquier cambio en las características del dispositivo que ejecute el código simplemente repercutiría en el tiempo de ejecución y en ningún caso en la efectividad del algoritmo.

A continuación, se presentan los lenguajes, programas, librerías, herramientas, metodología y requisitos **a satisfacer**

## Lenguajes y programas.

En primer lugar, comenzaremos explicando los lenguajes que se han utilizado: Python y Java, aunque el primero fue usado solo en la parte más inicial del proyecto.

La elección de estos dos lenguajes de alto nivel es debido a la naturaleza del proyecto ya que estos lenguajes presentaban librerías que nos resultaron atractivas y con potencial para ayudarnos a resolver el problema de las cuales hablaremos más adelante.

### Python



Es un lenguaje nacido a finales de los años 80 de la mano de Guido van Rossum, un holandés que desarrolló este lenguaje de programación para el Centro para las Matemáticas y la Informática de los Países Bajos con la intención de sustituir al lenguaje ABC, que a su vez surgió como alternativa a BASIC.

Las características de Python son las siguientes:

1. Interpretado: No requiere compilación, ya que se interpreta en tiempo real.
2. De alto nivel: El lenguaje está más cerca del lenguaje humano que del de la máquina. Se premia la facilidad sobre la eficiencia.
3. Propósito general: Puede ser utilizado con diversos propositos.
4. Multiplataforma: Funciona en un gran número de plataformas incluyendo los sistemas operativos más utilizados en este momento.
5. Open Source: El código puede ser visto por cualquiera que lo desee, además gracias a esto es gratuito.
6. Diversidad de paradigmas: Soporta orientación a objetos, programación imperativa e incluso funcional
7. Tipado dinámico: Las variables pueden tomar valores de distinto tipo sin que ésto produzca un fallo.
8. Extensa colección de librerías tanto built-in como en la web.
9. Sintaxis indentada: Python utiliza la indentación para diferenciar los bloques de código. De esta manera los programadores estan obligados a usar el mismo estilo, lo que facilita la similitud y entendimiento de código.
10. Multitarea: Compatible con el uso de threads y concurrencia.

Este lenguaje se utilizó en los primeros meses, junto a la librería PyBrain (punto 4.2.1), pero debido a la poca documentación y al aparente estado de abandono en el que se encontraba la librería se decidió hacer un cambio de lenguaje y librerías a Java y Neuroph.



### Java

Se creo a principios de los 90 y fue desarrollado por James Gosling (Sun Microsystems). El objetivo era crear un lenguaje independiente de la plataforma y un entorno (JVM) ligero y gratuito para que las aplicaciones se pudieran ejecutar en la mayor parte de plataformas.

Sus carácteristicas son:

1. Compilado e interpretado: El código fuente se traduce a bytecode (.class) los cuales serán interpretados por la maquina virtual de Java.
2. De alto nivel
3. Propósito general
4. Multiplataforma: Dado que el bytecode es independiente del sistema, simplemente se tiene que implementar una máquina virtual para que cada sistema sea capaz de ejecutarlo. (Write once run anywhere)
5. Open Source
6. Diversidad de paradigmas.
7. Tipado estático: Los tipos de las variables deben estar definidos, y nunca pueden tomar un valor que no sea de dicho tipo.
8. Extensa colección de librerías tanto built-in como en la web.
9. Sintaxis no indentada: La sintaxis de java no funciona mediante indentación sino mediante el uso de llaves **{}**.
10. Multitarea: Compatible con el uso de threads y concurrencia.
11. Dinámico: Las clases compiladas pueden ser cargadas bajo demanda en tiempo de ejecución

Al igual que con los lenguajes los IDE’s también han sido diversos. Al estar usando Python al comienzo del proyecto, el IDE que elegimos para este lenguaje fue PyCharm.

Con el cambio de lenguaje se produjo un cambio en los IDE’s. En el caso de Java primero se utilizó NetBeans y para finalmente acabar usando la plataforma Eclipse

### https://www.fourdigits.nl/images/pycharm-logo.gifPyCharm

Es un entorno de desarrollo integrado (IDE) específico para el lenguaje Python desarrollado por la empresa JetBrains en 2010 (también creadora de otros famosos IDE’s como WebStorm, RubyMine…). Es multiplataforma con versiones para Linux, Windows y Mac.

Entre sus características podemos encontrar: Refactorizaciónes automáticas, soporte para diferentes frameworks, debugger, unidad de testeo intergada, integración de control de versiones, navegador de proyecto con vistas y estructuras especializadas y análisis y asistencia de código con autocompletado, subrayado de errores y arreglos de diversos problemas.

### Resultado de imagen de eclipse software logoNetBeans y Eclipse

Dos de los IDE más conocidos hoy en día. Netbeans, desarrollado por Oracle Corporation en el año 2000. Ambos disponen de versiones para Windows, Linux y Mac, y son compatibles con desarrollos en Java, Javascript, C, C++, desarrollo web y más. Disponen de Add-Ons para añadir funcionalidades a la aplicación, por ejemplo, Eclipse tiene una version para programación en Android, en cambio Netbeans necesita la previa instalación de un plugin.

Sus características son las mismas que cualquier otro IDE como PyCharm. Soporte para refactorizaciones automaticas, debugger, integracion de control de versiones, autocompletado, etc...

## Librerías y herramientas

A continuación, se exponen las librerías de redes neuronales que se han usado en el proyecto, así como herramientas secundarias para la gestión del mismo:

### Resultado de imagen de pybrainPyBrain y Neuroph



PyBrain es una librería para Python. Su objetivo es ofrecer algoritmos Machine Learning potentes, flexibles y fáciles de usar. Su nombre proviene de las siglas **Py**thon-**B**ased **R**einforcement Learning, **A**rtificial **I**ntelligence and **N**eural Network Library. Su código es abierto y de uso gratuito para cualquiera ([Licencia BSD](https://es.wikipedia.org/wiki/Licencia_BSD#Licencia_BSD_modificada_.28de_3_cl.C3.A1usulas.29)), pero por desgracia los últimos cambios en su [repositorio](https://github.com/pybrain/pybrain) datan de hace más de 1 año, por lo que podemos considerar que ya se encuentra muy desactualizada y podemos encontrar otras opciones mejores en liberías como TensorFlow, Blocks, Deepy, Neupy, etc.

En el caso de Neuroph, hablamos de una librería Java que también cuenta con un IDE completo basado en NetBeans que añade una interfaz gráfica intuitiva a la hora de programar, la cual no se ha utilizado en este proyecto.  
Está orientado al desarrollo de redes neuronales comunes, con un reducido número de clases asociadas a los conceptos básicos de redes.  
Al igual que PyBrain, es Open Source y gratuito, bajo licencia [Apache 2.0](http://www.apache.org/licenses/LICENSE-2.0.html). Su [repositorio](https://github.com/neuroph/neuroph) se encuentra activo desde mediados de 2015 y el último commit es de Marzo de 2017.

Ambas están orientadas al trabajo con redes neuronales, desde la creación de conjuntos de entrenamiento, configuración de redes, entrenamiento etc. Útiles para resolver cualquier tipo de problema que pueda ser planteado desde un enfoque de redes neuronales.

### https://maven.apache.org/images/maven-logo-black-on-white.pngMaven

Es una herramienta Open Source, creada por Jason van Zyl, de Sonatype en el año 2002. Es similar a las herramientas Apache Ant, PEAR (php) y CPAN (Perl)

Su objetivo es simplificar la gestión de un proyecto software de tal manera que un desarrollador pueda extraerse de ciertos procesos con la ganancia de tiempo que esto conlleva. Con Maven la build de un proyecto se basa en tener un fichero pom.xml donde tengamos definida la configuración de nuestro proyecto con sus módulos, dependencias, librerías, etc… y ejecutar el comando mvn install.

Aunque en realidad, Maven es capaz de gestionar completamente el ciclo de un software ya que puede gestionar Validación, Compilación, Tests unitarios, Empaquetado, Pruebas de Integración, Verificado, Instalación y Despliegue de nuestro proyecto.

Como añadido, cuenta con un repositorio en internet llamado Maven Central. En él se encuentra una colección de librerías asociadas a sus posibles dependencias, de tal manera que con definir en el pom.xml las librerías que necesita nuestro proyecto maven accederá al almacén central y nos descargará automáticamente todo lo que necesitemos (incluyendo todas las librerías que se necesiten a niveles más bajos).

### https://upload.wikimedia.org/wikipedia/commons/thumb/e/e0/Git-logo.svg/1200px-Git-logo.svg.pngGit (GitHub)

En el año 2002, el proyecto del núcleo de Linux empezó a usar un software de control de versiones llamado “BitKeeper” de manera gratuita. Tres años más tarde (2005) la compañía propietaria decidió dejar de ofrecer los servicios de manera gratuita a la comunidad Linux. Esto propició que el propio Linus Torvalds comenzara a desarrollar esta herramienta gratuita de control de versiones: Git.

El desarrollo vino marcado por las lecciones que aprendieron de BitKeeper, con los siguientes objetivos:

Este tipo de programas se centran en la gestión de los cambios que se producen en un proyecto a lo largo del tiempo, de tal manera que todos esos cambios se guarden y se organicen en versiones incrementales. Otros VCS (versión control system) son SVN. Mercurial, CVS…

Gracias a esto tenemos distintas snapshots de nuestro código, que siempre pueden ser revisadas, y pueden servir para revertir todos los cambios hasta ese punto si en algún momento se ha producido un fallo y se quiere volver al estado de un commit concreto.

### Resultado de imagen de objectaidObjectAid

Es un plugin de Eclipse encargado de generar los diagramas UML a partir de nuestro código fuente Java, que se actualizan automáticamente con los cambios.

### SonarQube

Como añadido, se ha utilizado la herramienta SonarQube para analizar la calidad del código, detectar posibles defectos en el mismo, y poder así subsanarlos.

**Alargar un poco más**

## Metodología de desarrollo software

El término metodología se puede definir como el conjunto de métodos que se emplean para lograr un objetivo.

A su vez, un método se compone de herramientas, técnicas y procedimientos.

Así, podemos decir que una metodología de desarrollo de software define el conjunto de herramientas, procedimientos y técnicas que se emplean con el objetivo de construir el software deseado.

Dicho de otra manera, es el conjunto de las estrategias de desarrollo que nos ayudan a organizar los procesos y actividades dentro del ciclo de vida del software definiendo así el marco de trabajo que se va a utilizar durante el desarrollo de un proyecto. Entre todos los modelos de desarrollo podemos destacar el modelo en Cascada, el modelo Incremental, el modelo en Espiral y los modelos Ágiles

### Modelo en Cascada

La característica principal del modelo en cascada es que carece de ningún tipo de iteración.

El proceso se divide en etapas secuenciales, de tal manera que una etapa solo empieza cuando ha acabado la anterior.

La calidad del proyecto se evalúa solo al final del proceso, en un momento en el que puede que ya sea demasiado tarde como para subsanar posibles errores, y como añadido, cualquier cambio de requisitos es completamente incompatible con este modelo, ya que una vez finalizada esa etapa no se deberían de modificar, pues estaríamos volviendo a una etapa anterior.

Esto hace que este modelo solo sea factible en proyectos muy cerrados y con unos requisitos que no tienen ningún margen de cambio.

Requisitos

Diseño

Implementación

Verificación

Mantenimiento

### Modelo incremental

El modelo incremental se basa en repetir el ciclo cascada un número de veces. Al tener varias iteraciones, permite añadir nuevas funcionalidades, requisitos o especificaciones, creando al final de cada ciclo una evolución del software.

Requisitos

Diseño

Implementación

Verificación

### Modelo en espiral

En este modelo se divide el proyecto en distintos ciclos. En cada ciclo se desarrollan una serie de funcionalidades. Una vez hechas, se prepara otra iteración.

La principal diferencia entre el modelo en espiral y otros modelos es la consideración explicita del riesgo.

Análisis de requisitos

Discusión de alternativas

Desarrollo

Validación

Planificación



### Modelos ágiles

El desarrollo ágil tiene como objetivo desarrollar software rápidamente, de calidad y con una capacidad muy elevada de respuesta a cambios o imprevistos en el proyecto. Se enfocan principalmente en la gente y en sus resultados. En el año 2001 se creó el llamado “Manifiesto Ágil”, que consta de los siguientes valores:

1. Valorar más a los individuos que a las herramientas
2. Valorar más el software que funciona sobre la documentación exhaustiva
3. Valorar más la colaboración con el cliente que los contratos contractuales
4. Valorar más la respuesta a cambios que tener planes concretos e inmutables.

En los modelos ágiles los ciclos (llamados Sprints) se reducen a un mínimo de 2 semanas. Antes de cada sprint se eligen las funcionalidades a desarrollar. Cada tarea debe ser testeada antes de ser considerada como hecha.

La elección ha sido utilizar el sistema en espiral. El motivo principal …………………………………….

Metodología de desarrollo, scrum o lo que sea. Iterativo principalmente y tal. Diagramas y tal y cual… De 2 a 6 páginas.

## Especificación de requisitos

Nuestro software tiene que cumplir una serie de condiciones expresadas a modo de requisitos funcionales y no funcionales:

### Funcionales

Son una descripción de lo que debe o no debe hacer el sistema a bajo nivel, que servicios debe proporcional, cuestiones técnicas… “Qué” debe hacer

* Fasdf
* Asdf
* Asdfa
* Ssdf

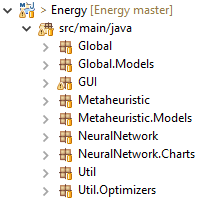
### No funcionales

Especifica criterios para juzgar la operación de un sistema. “Cómo” debe hacerlo

* Fghj
* Fghj
* Fghj
* Fghj
* Fgh
* df

## Estructura del software

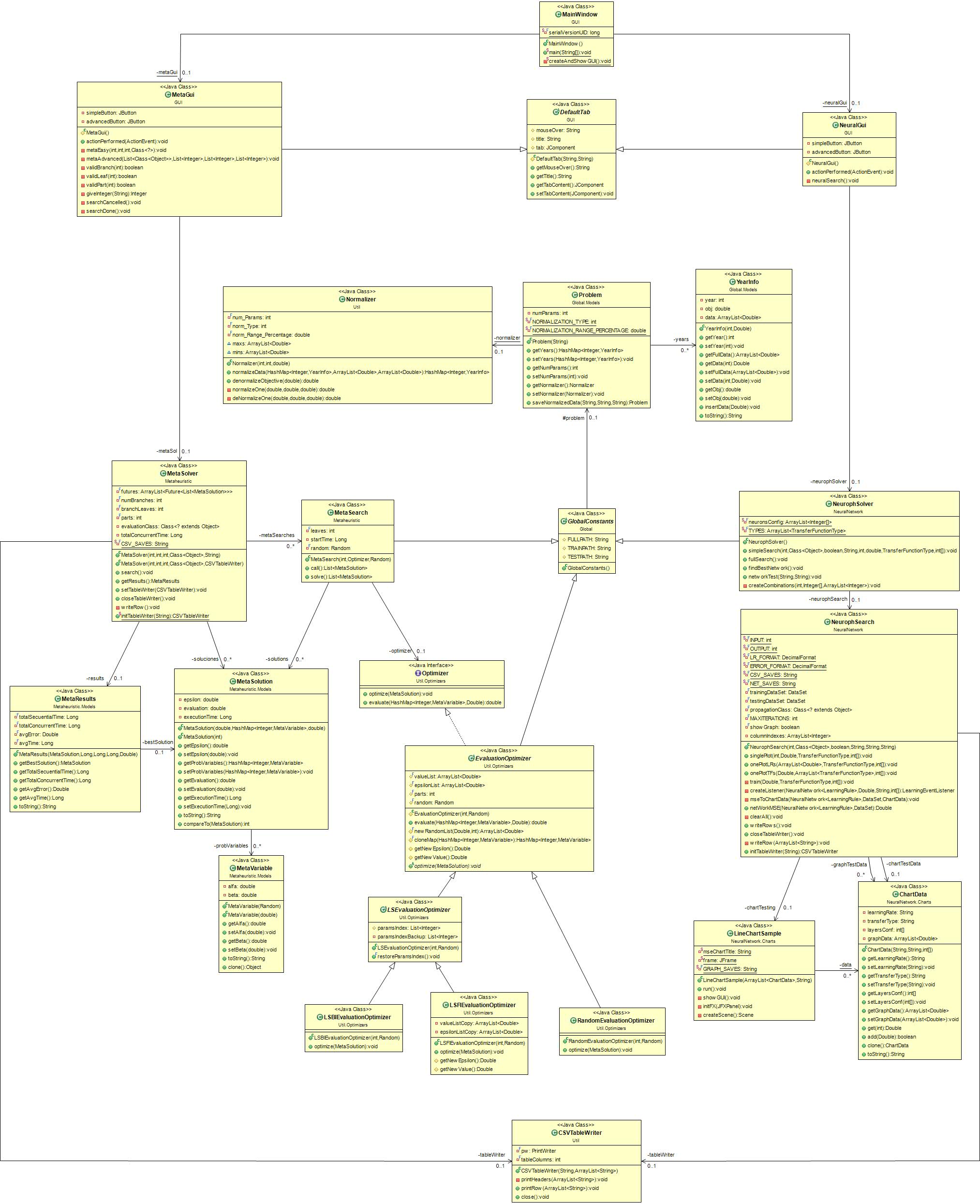
En cuanto a la organización interna del proyecto, cada clase se ha diseñado dentro de un paquete que engloba su objetivo. Así, los paquetes existentes son los siguientes:



* Global: Contiene lo relacionado con variables globales **GlobalConstants**
  + Models: Clases de los modelos asociados al paquete **Problem** y **YearInfo**
* GUI: Clases asociadas a la interfaz gráfica de la aplicación **MainWindow**, **DefaultTab**, **MetaGui** y **NueralGui**
* Metaheuristic: Clases asociadas a la resolución de la parte de metaheurísticas **MetaSolvers** y **MetaSearch**.
  + Models: Clases de los modelos asociados al paquete **Metaresults**, **MetaSolution** y **MetaVariable**.
* NeuralNetwork: Clases asociadas a la resolución de la parte de redes neuronales **NeurophSolver** y **NeruophSearch**.
  + Charts: Clases de los modelos asociados al paquete **ChartData** y **LineChartSample**.
* Util: Clases generales con diversos usos **CSVTableWriter** y **Normalizer**
  + Optimizers: Al tener una estructura más compleja y tener varias clases, un paquete interno a útil para las clases **Optimizer**, **EvaluationOptimizer**, **RandomEvaluationOptimizer**, **LSEvaluationOptimizer**, **LSFIEvaluationOptimizer** y **LSBIEvaluationOptimizer**.

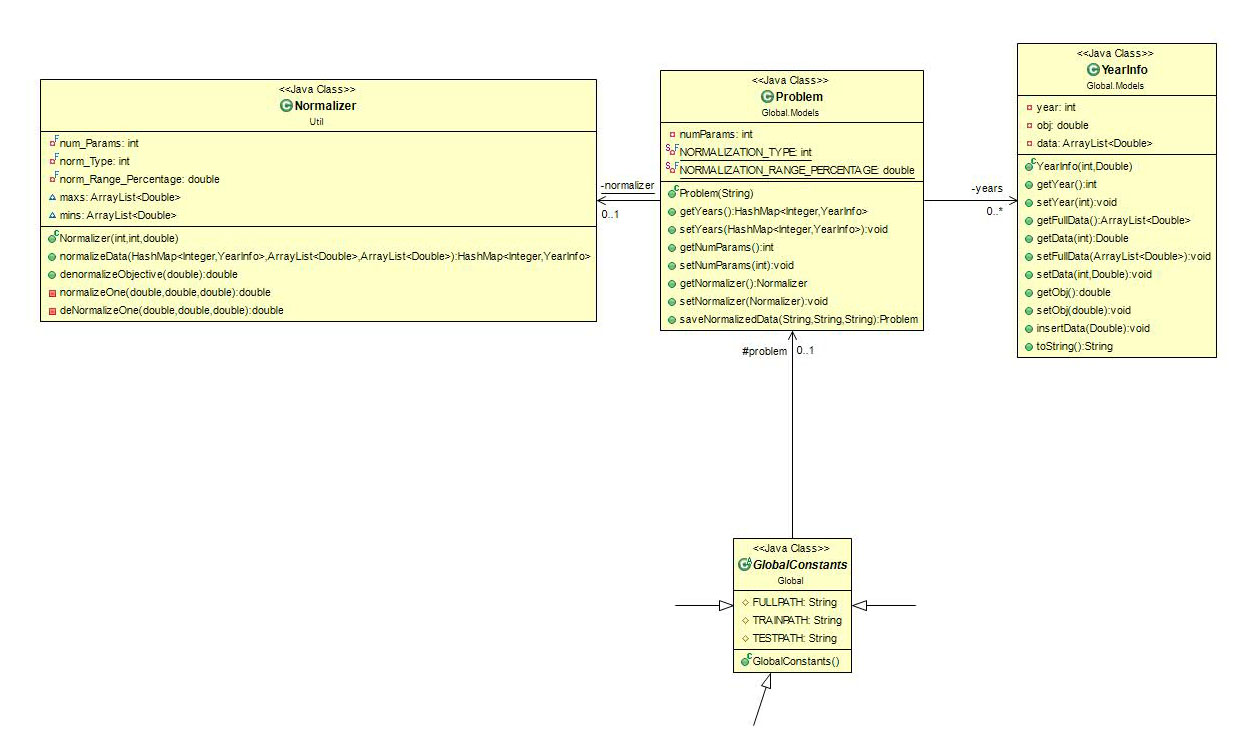
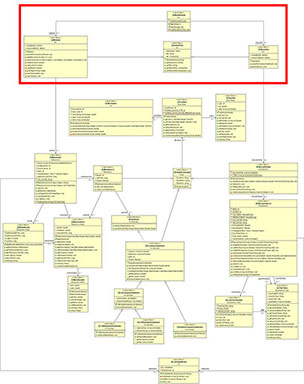
Las siglas UML vienen del inglés Unified Modeling Language (lenguaje unificado de modelado). Es un lenguaje gráfico que sirve para visualizar sistemas, procesos, bases de datos…

En este caso se utiliza para representar las clases que tiene nuestro proyecto y todas las interrelaciones que hay entre ellas:



En el diagrama hemos podido ver todas las clases de nuestro proyecto, pero para facilitar el entendimiento se explicará cada una de sus partes por separado:

### Constantes Globales



Estas clases están encargadas de facilitar el acceso a los datos globales del problema.

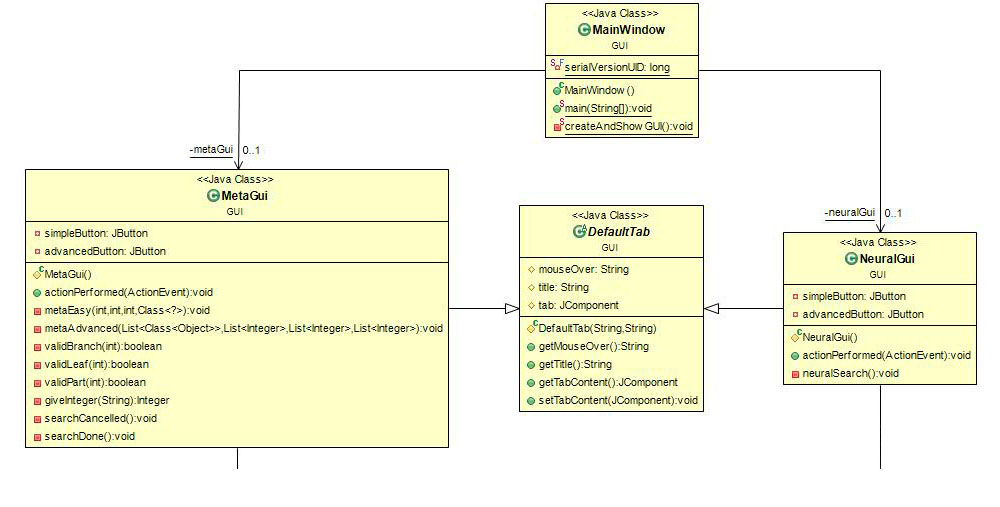
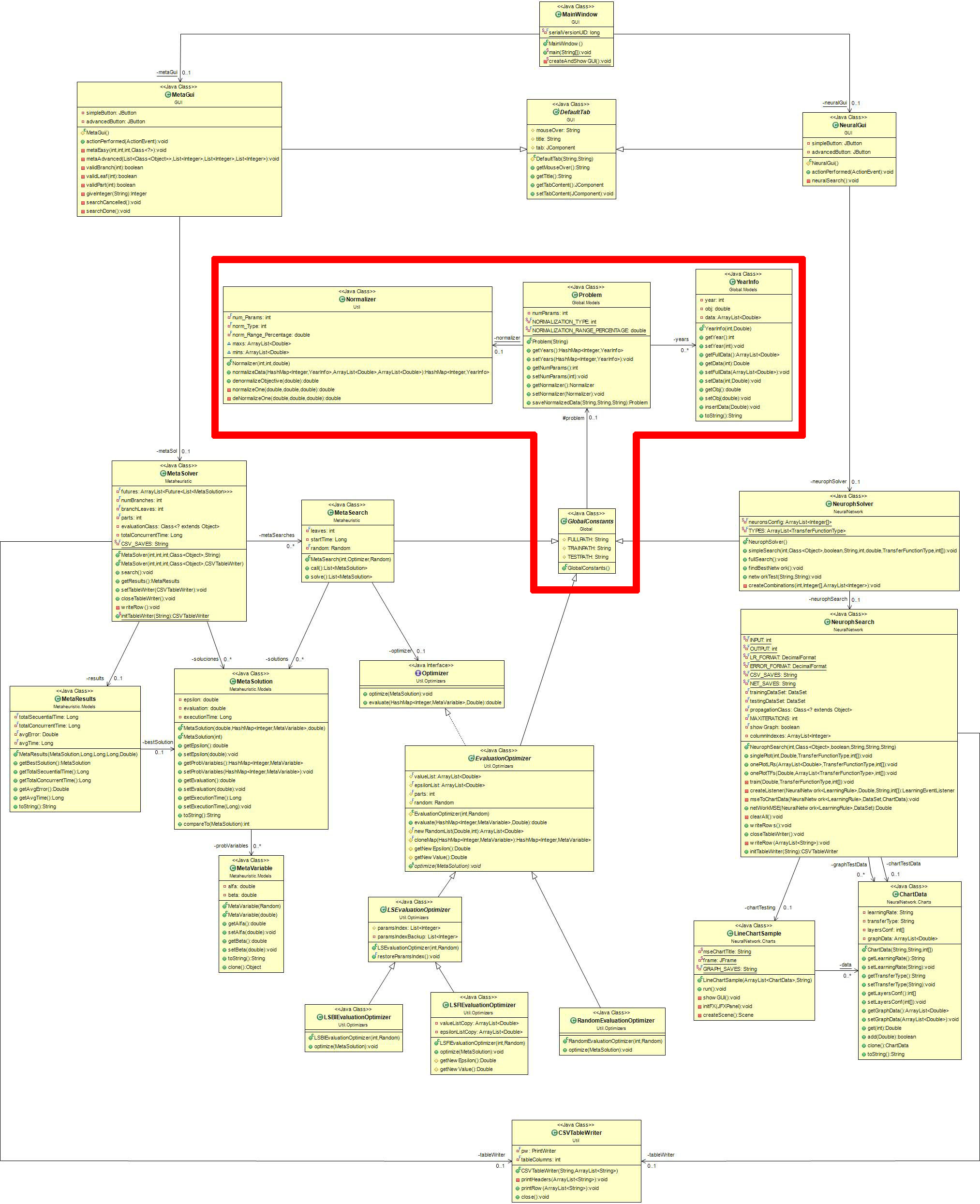
**YearInfo** es la encargada de guardar cada una de las 14 variables macroeconómicas para cada año.

**Normalizer** estará encargada de realizar todas las labores de normalización y desnormalización de los datos.

**Problem** mediante un *HashMap* contendrá tantos **YearInfo** como años de datos dispongamos, aparte de una instancia de **Normalizer.**

**GlobalConstants** funcionará como punto de acceso a nuestro problema, y además dispondrá de las rutas de entrenamiento que serán usadas por la parte de redes neuronales. En caso de no necesitar acceder a esas rutas, no tendría sentido la existencia de esta clase.

### Interfaz Gráfica



En cuanto a la interfaz gráfica, disponemos de una ventana principal, correspondiente a la clase **MainWindow**, que funciona como punto de entrada de nuestro programa, con la función main. Es un JPanel con pestañas. Cada una de esas pestañas dispondrá de una clase propia donde se encontrará su diseño. Al tener un esquema compartido se crea la clase **DefaultTab**, de la que heredarán ambas.



Las clases **MetaGui** y **NeuralGui** contienen el diseño interno que se mostrará al pulsar cada pestaña.

***En el caso de MetaGui, el diseño final es éste:***

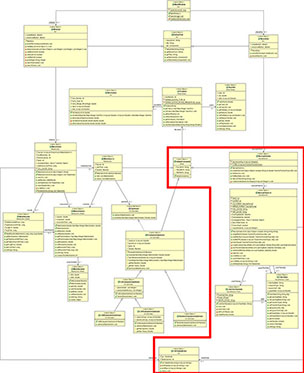
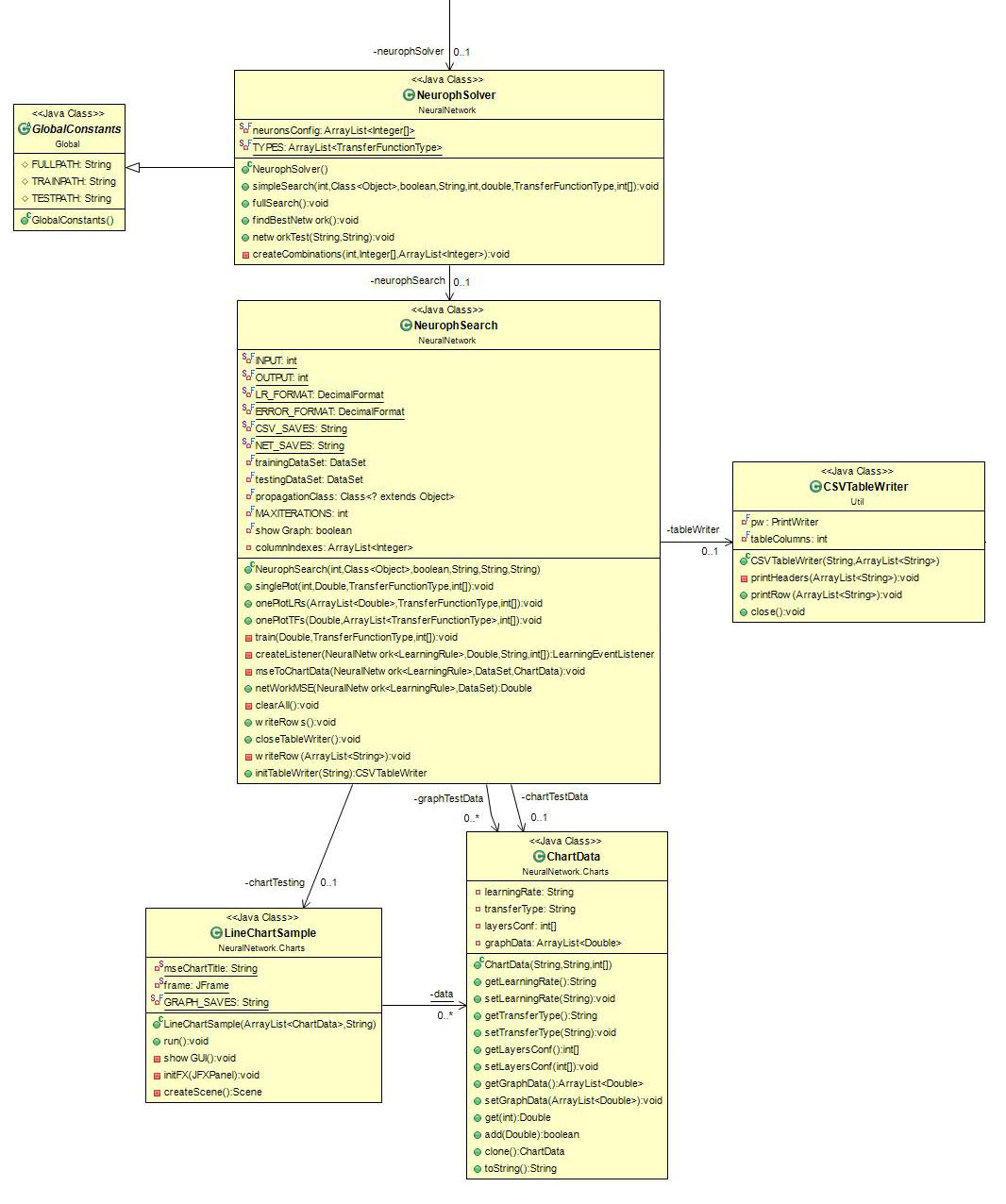


***El diseño final de NeuralGui:***



Además, también están encargadas de lanzar el procedimiento que surge tras pulsar un botón, incluyendo todo el proceso de recogida de datos y validación de los mismos, seguido del posterior lanzamiento de la búsqueda por parte de la clase MetaSolver o NeurophSolver, que serán explicados detalladamente más adelante. Es importante que estos procesos sean lanzados en segundo plano para permitir que la interfaz gráfica continúe actualizándose.

### Redes Neuronales



Podemos dividir las clases relacionadas   
con redes neuronales en 3:

* Clases de interfaz

**ChartData** contiene la información correspondiente a los errores que se han ido obteniendo a lo largo de un entrenamiento y la configuración del mismo.

**LineChartSample** genera un *Thread* que se encarga de crear una ventana y pintar tantas líneas como ChartData tenga de entrada.

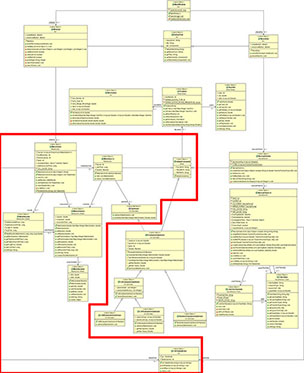
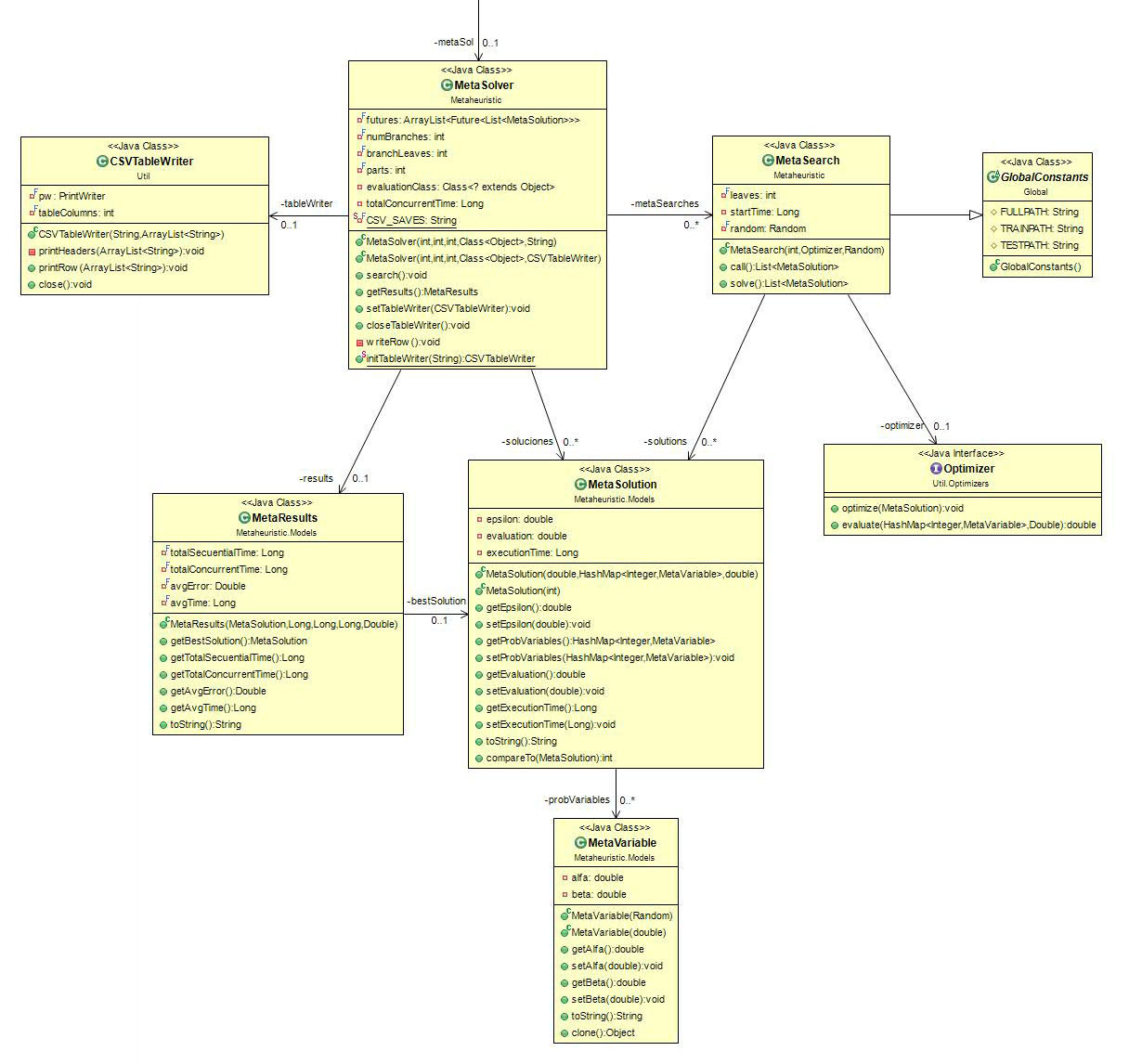
* Clases externas

**GlobalConstants** es utilizada para acceder a variables globales como las rutas que contienen los datos de entrada y la clase Problem.

**CSVTableWriter** es una clase que se utiliza de manera global en el proyecto cada vez que se quiere generar un archivo Excel con formato csv. En este caso, se utiliza para guardar númericamente los datos que previamente se han mostrado gráficamente a través de LineChartSample.

* Clases ***“importantes”***:
  + **NeurophSolver:** Cuenta con diversos procedimientos en función de la tarea a desarrollar, en concreto:
  + **Reducir a tiene varios metdoso que hacen tal cosa…**
    - **simpleSearch** Dados todos los parámetros de manera única, **PONER LOS DATOS DE ENTRADA????**manda esos datos a NeurophSearch, que se encargará de crear una red neuronal apropiada, hacer los entrenamientos oportunos, calcular los errores tras finalizar cada epoch, etc; finalmente, se llaman a las funciones de NeurophSearch para imprimir los resultados.
    - **advancedLRSearch** La única diferencia con respecto a la búsqueda simple es que la avanzada está pensada para admitir combinaciones y entrenar una detrás de otra. Así, tiene los mismos valores de entrada que la función simpleSearch, pero cambiando el parámetro learningRate por una lista de LearningRates a estudiar, así como el array que define las capaz ocultas y el número de neuronas que hay en cada capa por dos parámetros que indicarán el máximo número de capas ocultas y el número de neuronas que puede haber en cada capa. Con estos datos, irá generando todas las combinaciones de capas ocultas y llamará a la función de entrenamiento basado en varios learningRates de NeurophSearch.
    - **findBestNetwork:** Recorre todas las redes neuronales guardadas en la ruta por defecto (NeurophSolutions/Networks/\*.nnet) y calculará el error medio cuadrático tras pasar todos los datos de prueba por la red. Una vez haya recorrido todas, imprimirá por pantalla el archivo con la red más prometedora y el error medio cuadrático obtenido.
    - **networkTest:** Dada la ruta de la red neuronal a testear, genera un archivo csv con el nombre deseado en el que se encuentra la comparativa entre los datos obtenidos y los reales en cada año, para posteriormente generar con un programa como Excel gráficas que representen fielmente los resultados.
  + **NeurophSearch:** Es la clase que conecta directamente con la librería de Neuroph, encargada de realizar las labores de entrenamiento, generación de gráficos, guardar esos gráficos, guardar las redes una vez han finalizado el entrenamiento, etc… De todas las funciones las más relevantes son las relacionadas con el entrenamiento, y el guardado de los datos que se van obteniendo:

### Metaheurísticas



En este caso, dividimos lo relacionado con metaheurísticas en 4 apartados:

* Clases de modelos
  + MetaVariable
  + MetaSolution
  + MetaResults
* Clases externas

**GlobalConstants** es utilizada para acceder a variables globales como las rutas que contienen los datos de entrada y la clase Problem.

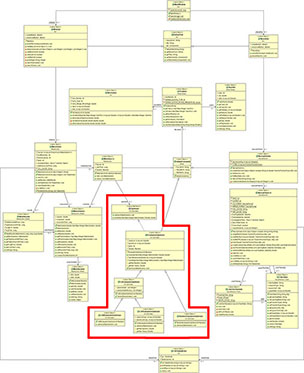
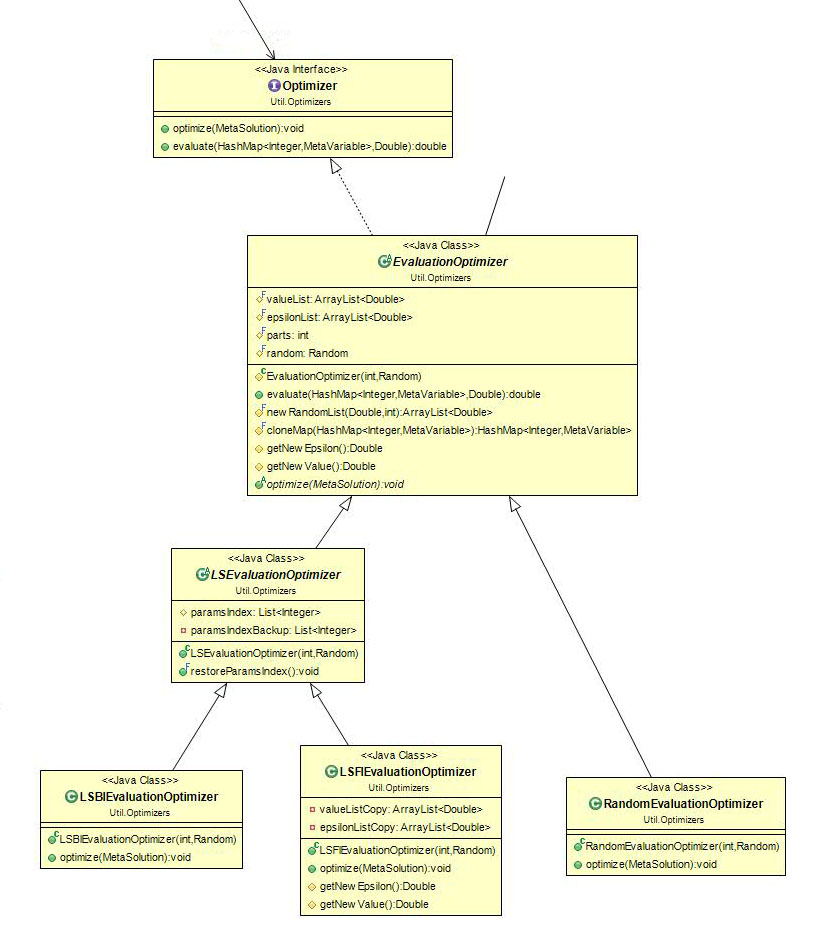
**CSVTableWriter** es una clase que se utiliza de manera global en el proyecto cada vez que se quiere generar un archivo Excel con formato csv. En este caso, se utiliza para guardar numéricamente los datos que previamente se han mostrado gráficamente a través de LineChartSample.

* Clase Optimizador
* Clases **“importantes”**

**MetaSolver** mapsodjaops

**MetaSearch** sapodjoasp

### Optimizadores de metaheurísticas



Si bien todo lo relacionado con optimizadores podía haber sido desarrollado en una sola clase, para ser fieles a la programación orientada a objetos, su diseño ha sido el que vemos en el diagrama UML, teniendo así que dedicarle un apartado completo.

**Clase Optimizer**: Interfaz con las funciones optimize y evaluate, que serán desarrolladas en clases que implementen la interfaz.

**Clase EvaluationOptimizer**: Esta clase implementa la anterior, pero al ser abstracta no necesita definir todas las funciones de su interfaz, así, solo implementa la función evaluate, que será global para todas las subclases que vengan detrás, deja abstracta la función optimize que será específica, define los primeros atributos de clase y funciones auxiliares.

**Clase RandomEvaluationOptimizer**: Esta es la primera clase no abstracta, y es el optimizador más sencillo con el que contamos. Como su nombre indica, genera cambios al azar sobre las variables presentes en la instancia MetaSolution que le entre como parámetros. Simplemente extiende la clase EvaluationOptimizer y termina de implementar la función Optimize.

**Clase LSEvaluationOptimizer**: **localsearch?linesearch? Xd** Esta clase se bifurca de la anterior, redefiniendo el comportamiento del constructor y otra función que será usada por sus subclases. De nuevo, es otra clase abstracta, **por lo que no puede ser instanciada**.

Tecnologías: Eclipse, NetBeans, UML el GoUML o lo que sea para hacer diagramas de clases, la librería del Neuroph, java v8. Lo que tengo que instalar para que funcione todo. Hablar de todas las tecnologías. HABLAR TAMBIÉN DE PYTHON, aunque lo dejara a posteriori.

Diagramas de clases. Describir a nivel de clase relevante los métodos más importantes, (mirar en otros proyectos). Figura, explicar un poco y tal, que ilustre la cantidad de código que he elaborado, como se integran las librerías. A nivel de bloque el nivel de código que se ha desarrollado (El código se sube al aula, aquí no hay que explicar nada de código, solo ver que se ha hecho), quizás algo de pseudocódigo de la búsqueda o algo concreto, que sea específico y relevante.34:00

# EXPERIMENTOS (<10 págs.)

## Redes Neuronales

Para entrenar una red neuronal satisfactoriamente hay varios parámetros que hay que ajustar para que los resultados sean los buscados. Esos parámetros son los siguientes:

* En cuanto a Iteraciones de entrenamiento hemos definido tal para evitar que se vaya de madre
* Tipo de propagación en la red:
* LearningRates:
* Función de transferencia de las neuronas:
* Capas ocultas: en la siguiente tabla se muestra el comportamiento con x,y y z. Como se puede ver en tal tabla lo mejor es tantas capaz ocultas
* Número de neuronas presentes en cada capa:

Una vez explicado esto, el proceso que se ha seguido para determinar la mejor combinación ha sido el siguiente:

**Explicar método de experimentación**

## Metaheurísticas

Las metaheurísticas también cuentan con parámetros que hay que ajustar, específicamente:

* Clase de optimizador: **Explicar**
* Número de hilos de búsqueda (ramas) : **Explicar**
* Número de soluciones que va a optimizar cada hilo (hojas) : **Explicar**
* Número de partes en las que vamos a dividir el rango de valores que pueden tomar las variables de nuestra MetaSolution: **Explicar**

Así, para realizar el estudio de todos estos parámetros **Explicar método de experimentación**

Ajuste y estudio de parámetros, como afectan sus valores…Tanto de la red (LR, Epochs), como de la meta (optimización elegida, iteraciones…) que cosas permiten elegir los mejores parámetros teniendo en cuenta los objetivos (si tarda más o menos o que)

TIRAR A PONER GRÁFICOS O FIGURAS, mucho mejor que una tabla.

Comparar con métodos previos (En la introducción, nombras gente que ha trabajado, es posible que sean comparables con esto de aquí)

# CONCLUSIONES Y TRABAJOS FUTUROS (2 págs)

Lo de objetivos que iba en presente, ahora en pasado, que he logrado de todo.

En futuros, cosas mejorables o que probar. Interfaz gráfica, paralelización, redes neuronales Deep learning, otra metaheurística... O que habría hecho si hubiera tenido más tiempo.