Pareja: Victoria Pelayo e Ignacio Rabuñal. Grupo 2201.

En esta práctica hemos creado las clases que se nos pedían en los distintos apartados.

Las interfaces que se nos proporcionaban son las que hemos usado. Hemos realizado algunos cambios y añadido algunas funciones en estas interfaces.

Interfaz Algoritmo e interfaz Dominio: en estas interfaces no hemos realizado ningún cambio.

Interfaz Individuo: en esta interfaz hemos añadido las funciones etiqueta\_nodos() y copy(). Las hemos añadido porque en el caso de que se crearan más clases que heredaran de esta interfaz, además de la existente, sería útil tanto que se pudiera crear una copia como que se etiqueten sus nodos.

Interfaz Nodo: En esta interfaz hemos añadido los métodos getDescendientes() y setDescendientes() ya que cualquier clase que heredara de esta interfaz tendría estas funciones, incluirlo simplifica la programación ya que nos tocaría estar haciendo con todas las clases castings previamente. Por motivos de diseño hemos decidido que nuestros nodos tengan un atributo que sea su nodo padre, por ello hemos añadido las funciones getPadre() y setPadre(). También todos nuestros nodos tendrán una etiqueta, para ser distinguibles,y que se simplifique la tarea de búsqueda, del resto, por ello hemos añadido las funciones getPadre() y setPadre().

La decisión sobre implementar las clases la hemos tomado en función de los main de prueba que nos proporcionaban y de lo que nos pedía la práctica.

La clase Nodo, la hemos definido como abstracta porque nunca vamos a declarar un nodo. Siempre que queramos declarar un objeto que sea un nodo declaremos o una función o un terminal, clases hijas de Nodo.

Tenemos 3 clases hijas de función: suma, resta y multiplicación.

Tenemos una clase hija de terminal: terminal aritmético que tendrá una valor real asociado.

En el apartado opcional de la práctica nos piden hacer algo parecido a lo que hemos hecho pero con funciones lógicas y como terminales o true o false. Para esto yo ampliaría mi diseño de la siguiente manera: crearía las clases hijas de función correspondientes(las funciones lógicas) y crearía otra clase hija de terminal, TerminalLogico, esta clase en vez de terner asociado un real tendría asociado un booleano.

La clase individuo tiene un fitness asociado, una lista de los nodos etiquetados, su etiqueta corresponde al índice en esta lista y un nodo raíz. La lista de nodos etiquetados la hemos hecho para poder acceder a un nodo directamente sin necesidad de recorrer el árbol siempre.

La clase dominio Aritmético tiene un método que lee de un fichero de prueba los valores correspondientes y los almacena. Tenemos una lista con los valores y un mapa en el que asociamos cada valor con el resultado correspondiente que se debería obtener.

En la clase PruebaCruce solo tenemos dos métodos: cruce, que implementa el cruce entre dos individuos y un método interno buscar que devuelve un nodo que pertenece a un individuo, pasándole como argumentos el nodo raíz y la etiqueta del nodo correspondiente.

En la clase algoritmo tenemos una lista de INodos(población) y los datos correspondientes al algoritmo, profundidad, número de nodos, K, probabilidad…

En esta clase tenemos también dos listas, una de funciones y otra de terminales, estas listas las utilizaremos para crear individuos aleatorios de la población.

CrearPoblacion() la hemos implementado como una función que crearía una población a raíz de los datos iniciales, es decir, crearía N individuos con M profundidad(argumentos introducidos por el usuario).

CrearNuevaPoblacion() creará una nueva población a raíz de la existente, en esta función será dónde realmente se implemente el algoritmo.

Ejecutar() será una función que se llamará desde el tester correspondiente y podrá en prueba el algoritmo, en esta función crearemos la población inicial, y la imprimiremos por pantalla. Posteriormente, mientras la generación actual sea menor que el número de generaciones máximo se irá llamando a la función crearNuevaPoblacion() y las iremos imprimiendo por pantalla para observar su evolución.

Para esta última clase hemos creado el Testeralgoritmo en el que se comprueba su correcto funcionamiento.

Por último, hemos creado dos tipos de excepciones:

ArgDistintosFunciones y CruceNuloException. La primera salta cuándo se introducen ml los argumentos, respecto al número indicado. La segunda salta cuándo al hacer un cruce, como aleatorio obtenemos dos 1, ya que esto supondrá hacer el intercambio entre dos nodos raíz que es equivalente a hacer un intercambio de nodos, no un cruce entre ellos.