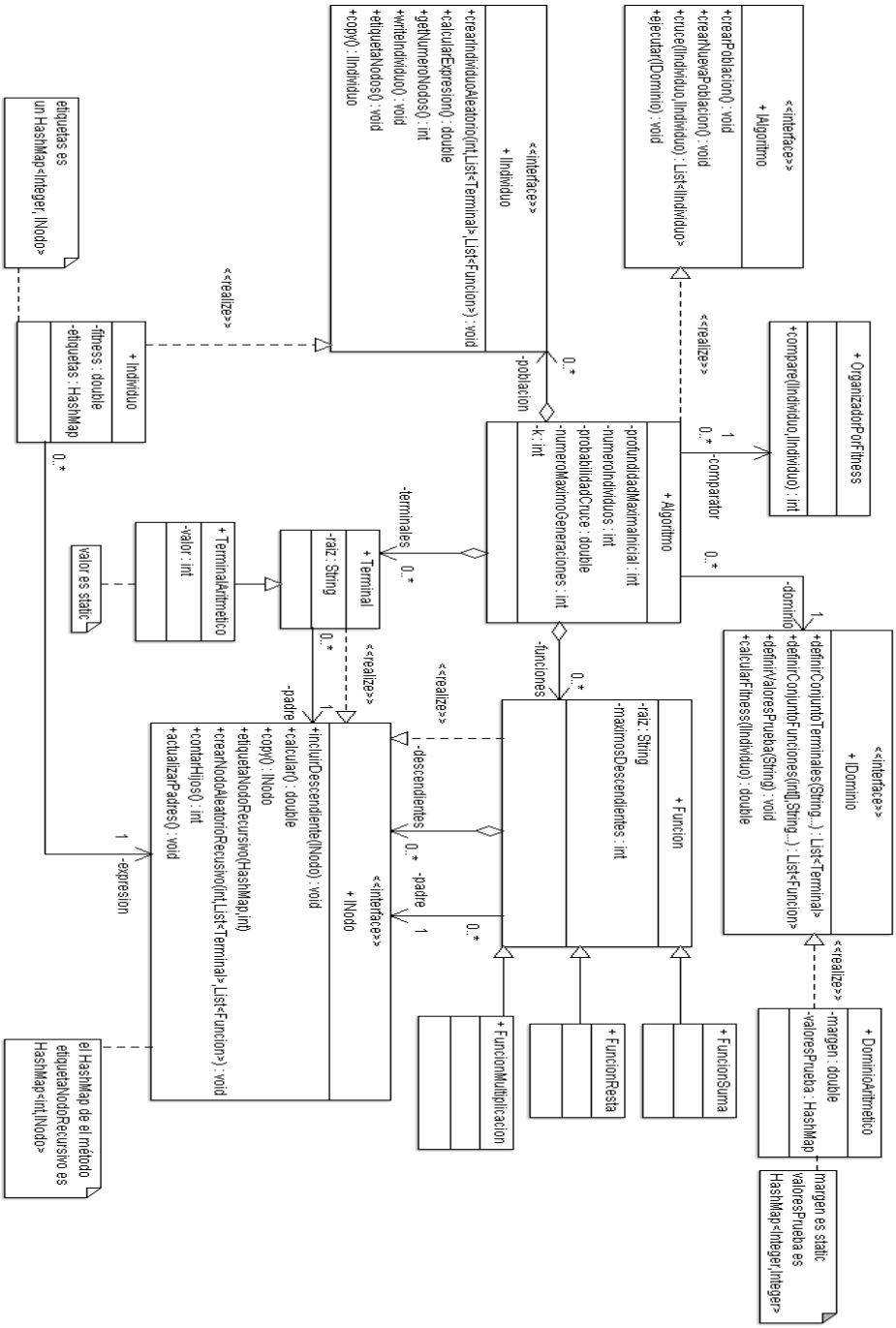
Miguel Arconada (miguel.arconada@estudiante.uam.es)
Alberto González (alberto.gonzalezk@estudiante.uam.es)

MEMORIA PRÁCTICA 4

a)



Miguel Arconada (miguel.arconada@estudiante.uam.es)
Alberto González (alberto.gonzalezk@estudiante.uam.es)

Decisiones de diseño:

Para comenzar con respecto a las interfaces hemos realizado unas ligeras modificaciones. Cabe destacar que añadimos el atributo padre de tipo INodo tanto en los terminales como en las funciones (con el fin de realizar el cruce). En la interfaz INodo añadimos getter y setter del atributo mencionado anteriormente, además de añadir etiquetaNodoRecursivo (para poder etiquetar los nodos y así poder realizar el cruce), crearNodoAleatorioRecursivo (para generar los individuos aleatorios de la población inicial) contarHijos (para contar los descendientes de un nodo) y actualizarPadres (para comprobar que tras el cruce el puntero del padre es el correcto). Todas estas funciones añadidas (a excepción de actualizarPadres) se tratan de funciones recursivas que son necesarias para algunos métodos de Individuo.

En las clases que implementan la interfaz IIndividuo hemos añadido un atributo de tipo HashMap<Integer,INodo> para guardar las etiquetas de los nodos que se utilizarán a la hora de cruzar los individuos. Por ello, hemos añadido un método getter de este atributo en IIndividuo. Además, añadimos el método etiquetaNodos, la cual llama a el método etiquetaNodoRecursivo mencionado con anterioridad, y copy, para copiar individuos. El método crearIndividuoAleatorio llama a la función recursiva crearNodoAleatorioRecursivo y el método contarNodos llama a contarHijos. El resto de las interfaces no han sufrido ninguna modificación.

A la hora de implementar el cruce nos encontramos con varias dificultades que hicieron del proceso de programación un arduo trabajo, resultando en un código menos eficiente de lo que hubiéramos deseado. Entre estos problemas destacamos que queríamos hacer uso del método indexOf propio de la interfaz List de Java, pero no funcionaba porque por alguna extraña y misteriosa razón debemos de estar devolviendo alguna copia de un objeto lo que causa el no funcionamiento de indexOf. Al no ser capaces ni nosotros ni el profesor de detectar la localización del error, optamos por iterar por la lista hasta encontrar el objeto, ralentizando la ejecución de nuestro programa.

A la hora de implementar los torneos, la documentación dada era bastante ambigua. En un primer momento, decidimos que se crucen los dos mejores individuos de cada grupo, los introducimos en la lista y posteriormente eliminamos los dos con peor fitness. Sin embargo, esto en la práctica provocaba que el programa fuese lento, además de necesitar de muchas generaciones más. Finalmente, realizamos unas modificaciones, cruzando los individuos, eliminando los dos peores y posteriormente introduciendo los individuos cruzados.

b) Sí es extensible, ya que todo depende de interfaces bastante genéricas, por lo que lo único que debería cambiar es crear una nueva clase que implemente una de las interfaces y cambiaría el código interno de los métodos.