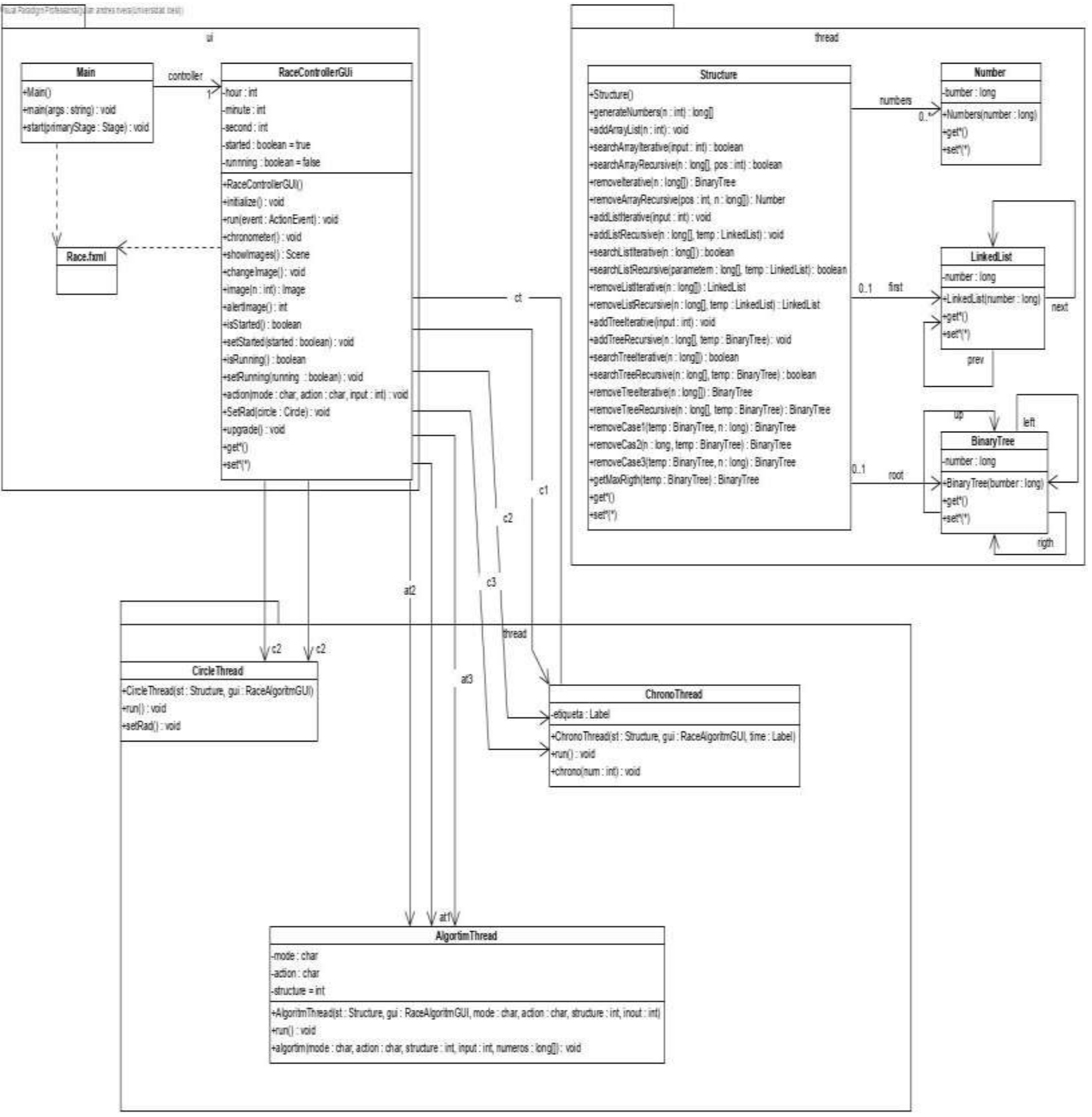
ui
Main
+Main()
+main(args : string) : void
+start(primaryStage : Stage) : void
controller
1
RaceControllerGUi
-hour : int
-minute : int
-second : int
-started : boolean = true
-running : boolean = false
+RaceControllerGUI()
+initialize() : void
+run(event : ActionEvent) : void
+chronometer() : void
+showImages() : Scene
+changeImage() : void
+image(n : int) : Image
+alertImage() : int
+isStarted() : boolean
+setStarted(started : boolean) : void
+isRunning() : boolean
+setRunning(running : boolean) : void
+action(mode : char, action : char, input : int) : void
+SetRad(circle : Circle) : void
+upgrade() : void
+get*()
+set*(*)
Race.fxml
thread
Structure
+Structure()
+generateNumbers(n : int) : long[]
+addArrayList(n : int) : void
+searchArrayIterative(input : int) : boolean
+searchArrayRecursive(n : long[], pos : int) : boolean
+removeIterative(n : long[]) : BinaryTree
+removeArrayRecursive(pos : int, n : long[]) : Number
+addListIterative(input : int) : void
+addListRecursive(n : long[], temp : LinkedList) : void
+searchListIterative(n : long[]) : boolean
+searchListRecursive(parametern : long[], temp : LinkedList) : boolean
+removeListIterative(n : long[]) : LinkedList
+removeListRecursive(n : long[], temp : LinkedList) : LinkedList
+addTreeIterative(input : int) : void
+addTreeRecursive(n : long[], temp : BinaryTree) : void
+searchTreeIterative(n : long[]) : boolean
+searchTreeRecursive(n : long[], temp : BinaryTree) : boolean
+removeTreeIterative(n : long[]) : BinaryTree
+removeTreeRecursive(n : long[], temp : BinaryTree) : BinaryTree
+removeCase1(temp : BinaryTree, n : long) : BinaryTree
+removeCas2(n : long, temp : BinaryTree) : BinaryTree
+removeCase3(temp : BinaryTree, n : long) : BinaryTree
+getMaxRigth(temp : BinaryTree) : BinaryTree
+get*()
+set*(*)
numbers
0..*
Number
-bumber : long
+Numbers(number : long)
+get*()
+set*(*)
0..1
first
LinkedList
-number : long
+LinkedList(number : long)
+get*()
+set*(*)
next
prev
0..1
root
up
left
BinaryTree
-number : long
+BinaryTree(bumber : long)
+get*()
+set*(*)
rigth
ct
c1
c2
c3
at2
at3
at1
c2
c2
thread
CircleThread
+CircleThread(st : Structure, gui : RaceAlgoritmGUI)
+run() : void
+setRad() : void
ChronoThread
-etiqueta : Label
+ChronoThread(st : Structure, gui : RaceAlgoritmGUI, time : Label)
+run() : void
+chrono(num : int) : void
AlgoritmThread
-mode : char
-action : char
-structure = int
+AlgoritmThread(st : Structure, gui : RaceAlgoritmGUI, mode : char, action : char, structure : int, inout : int)
+run() : void
+algoritm(mode : char, action : char, structure : int, input : int, numeros : long[]) : void

Requerimientos Funcionales

- **RF1.** EL programa tiene la capacidad de crear un arrayList con N datos generados automáticamente por el mismo y agregarlos . Las entradas son: N que indica la cantidad de números que se generan para agregar al arraylist
- **RF2.** El programa está en la capacidad de generar N números aleatoriamente, la cantidad será ingresada por el usuario apenas se inicie el programa. Las entradas son: un entero N que indica la cantidad de números a generar.
- **RF3.** El programa es capaz de agregar números a una lista doblemente enlazada, de forma iterativa; dichos números son generados aleatoriamente anteriormente. Las entradas son: el N que indica la cantidad de números generados que serán agregados
- **RF4.** el programa cuenta con la opción de agregar números de forma iterativa a un árbol binario de búsqueda, a partir de N números generados aleatoriamente. El N es ingresado por el usuario al inicio del programa. Las entradas son: el N que indica la cantidad de números a generar para, posteriormente agregarlos
- **RF5.** El programa cuenta con la opción, a partir de N números generados aleatoriamente, agregarlos a una lista doblemente enlazada de forma recursiva. Las entradas son: Un arreglo de números generados y el primer elemento de la lista, sea vacío o no.
- **RF6.** el programa tiene la capacidad de agregar cierta cantidad de números aleatorios de manera recursiva dentro de un árbol binario de búsqueda. Las entradas son: el arreglo de los números generados con anterioridad y la raíz del árbol, o primer elemento (sea vacío o no)
- **RF7.** El programa cuenta con la funcionalidad de buscar N elementos aleatorios dentro de un arrayList de manera iterativa y recursiva, devolviendo un valor de verdadero o false dependiendo si lo encuentra o no. Las entradas para opción iterativa son: el N que indica la cantidad números que se van a buscar dentro de la estructura. Las entradas para la recursividad son: un arreglo con los números que se van a buscar en el arraylist y la última posición del arraylist.
- **RF8.** El programa es capaz de buscar elementos dentro de una lista doblemente enlazada de manera iterativa o recursiva, dependiendo de como lo desee el usuario. Las entradas para la opción iterativa son: el arreglo de los N números a buscar. Las entradas en la opción recursiva son: el arreglo de los N números autogenerados a buscar en la estructura y el primer elemento de la lista
- **RF9.** El programa cuenta con la opción de buscar de manera iterativa o recursiva N números en un árbol binario de búsqueda, dependiendo de la selección del usuario. Las entradas para la manera iterativa son: el N que indica la cantidad de números que se vana a buscar. Las entradas de la manera recursiva son: el arreglo de los números que se desea buscar y el primer elemento del árbol(raíz)
- **Rf10.** El programa esta en la capacidad de eliminar N datos de un arraylist de manera iterativa como también recursiva, como lo desee el usuario. Las entradas para hacerlo iterativamente son: el arreglo de los números que se van a eliminar de la estructura. Las entradas para hacerlo recursivamente: el arreglo de números generados y la última posición del arraylist
- **Rf11.** el programa cuenta con la facilidad de manejar la manera iterativa y recursiva para eliminar datos de la lista doblemente enlazada, según lo decida el

usuario. Las entradas para forma iterativa son: el arreglo de números generados. Las entradas para forma recursiva son: el arreglo de números que se quiere eliminar y el primer elemento de la lista.

- **Rf12.** Continuando con la multifuncionalidad, el programa cuenta con formas iterativas y recursivas de eliminar N cantidad de datos de un árbol binario de búsqueda. Las entradas para forma iterativa son: el arreglo de números generados con anterioridad. Las entradas para forma recursiva son: el arreglo de datos que se eliminaran y el primer elemento del árbol (raíz).
- **Rf13.** El programa cuenta con una funcionalidad adicional de tipo carrera de algoritmos. El programa esta en la capacidad de mostrar el tiempo que se demora cada estructura pasar por el proceso que el usuario solicite con un cronometro independiente para cada estructura; iniciando al mismo tiempo y deteniéndose a medida que cada estructura termine su hilo de ejecución

Requerimientos no funcionales

- **Rnf1.** El programa cuenta con un cronometro general que inicia junto con los cronómetros de cada estructura y, deteniéndose cuando la ultima estructura termine de realizar su hilo de ejecución, es decir, la acción que el usuario haya elegido}
- **Rnf2.** el sistema le permite al usuario seleccionar una imagen como Avatar para cada estructura antes de ejecutar la ventana principal, con el fin de que el sistema tome una forma semejante a una competición, y sea mas llamativa para el usuario.