**Departamento de TIC**

**Algoritmos y Programación II**

**Laboratorio de la Unidades 4, 5 y 6**

# **Objetivos**

Unidad 4: Estructuras y Algoritmos Recursivos

OE4.1 Emplear el concepto de recursividad como una alternativa a la estructura de control iterativa.

OE4.2 Aplicar la computación recursiva en la solución de problemas de naturaleza inherentemente autocontenida.

OE4.3 Utilizar árboles binarios de búsqueda para representar grupos de objetos que mantienen entre ellos una relación de orden.

OE4.4 Escribir algoritmos recursivos para manipular estructuras de información recursivas y explicar las ventajas que, en este caso, estos algoritmos tienen sobre los algoritmos iterativos.

Unidad 5: Construcción de la interfaz gráfica

OE5.1. Utilizar una arquitectura de tres capas para el desarrollo de un programa de computador, repartiendo de manera adecuada las responsabilidades entre la interfaz de usuario, el control de la interfaz y el modelo. El estudiante deberá poder explicar la importancia de mantener separadas las clases de estos tres dominios.

OE5.2. Construir las clases que implementan una interfaz de usuario.

OE5.3. Aplicar la técnica de descomposición de requerimientos para cumplir con la funcionalidad de un programa de computador.

Unidad 6: Concurrencia y Dibujo Básico en 2D

OE6.1 Desarrollar un programa que maneje concurrencia, de manera que sea posible que ejecute más de una parte del programa de manera simultánea, utilizando hilos de ejecución (threads).

OE6.2 Construir interfaces de usuario que incluyan gráficas en 2 dimensiones como una alternativa en la presentación de información al usuario.

**Preparación**

* Lea cuidadosamente el enunciado, la documentación suministrada y cada uno de los puntos que debe desarrollar antes de empezar su desarrollo. Pregunte a su profesor cualquier duda respecto al enunciado o a los requerimientos funcionales que debe desarrollar.
* El trabajo debe ser realizado de forma **individual**.
* El trabajo será entregado en la fecha y hora establecida en Moodle.

**Su entrega debe incluir los siguientes elementos**

* Requerimientos Funcionales (donde se explica la funcionalidad que ve el usuario).
* Requerimientos No Funcionales (donde se detallan las restricciones internas de funcionamiento que el usuario del programa no puede ver, por ejemplo: que un ordenamiento es burbuja, que la persistencia es utilizando serialización, etc).
* Diagrama de Clase.
* Implementación en un proyecto de eclipse.
* Los requerimientos funcionales, no funcionales y el diagrama de clases deben presentarse de un documento o informe del laboratorio en formato pdf.
* Debe entregar, a través de Moodle, el enlace de su repositorio del laboratorio en GitHub a más tardar el **Jueves 14 de Mayo** 11:00 AM. El nombre del repositorio debe cumplir con las condiciones establecidas para los nombres de repositorios en los cursos de algoritmos: totalmente en minúsculas, sin caracteres extraños, si tiene varias palabras éstas van separadas por un guión y muy importante: el nombre NO debe debe decir ap2, ni apII, laboratorio, lab4, ni similares, es decir, el nombre debe hacer referencia al dominio del problema que se está solucionando.

**Carrera de Algoritmos**

**Enunciado**

Debido a la cuarentena a la que la mayor parte de la humanidad ha tenido que acogerse debido a una enfermedad altamente contagiosa, los eventos deportivos se han suspendido y las personas amantes de este tipo de espectáculo están un poco decepcionadas. Las personas no pueden salir, pero los algoritmos no se contagian, entonces se le ha propuesto que usted organice un torneo en el cual puedan participar los algoritmos de estructuras de datos que almacenan elementos.

El gran torneo de algoritmos será como una especie de mini juegos olímpicos de la computación en la cual, se considera a cada estructura de datos como un país, y a cada tipo de algoritmo como un deporte o disciplina. A continuación se presenta el esquema de competición:

|  |  |  |  |
| --- | --- | --- | --- |
| Estructura de Datos **\** Algoritmo | Agregar - **A** | Consultar - **C** | Eliminar - **E** |
| ArrayList - **AL** | Esta competencia consiste en agregar N elementos generados aleatoriamente a la estructura de datos. | Esta competencia consiste en consultar N elementos generados aleatoriamente sobre una estructura de datos previamente creada con N elementos también aleatorios. Solo se toma el tiempo de las N consultas.  Los resultados de cada consulta retornarán que el elemento existe o que no existe (boolean), pero eso no será tenido en cuenta en la carrera. | Esta competencia consiste en el eliminar N elementos generados aleatoriamente sobre una estructura de datos previamente creada con N elementos aleatorios. Solo se toma el tiempo de las N eliminaciones.  Los resultados de la eliminación podrá ser que el elemento se elimina o que no se elimina porque no existe, pero ese resultado no será tenido en cuenta en la carrera. |
| Lista Enlazada - **LE** |
| Árbol Binario de Búsqueda - **ABB** |

La estructura de datos ganadora es aquella que logre completar la operación primero que las demás.

Las tres carreras anteriores se llevarán a cabo en dos modalidades: iterativo y recursivo, por lo que serán finalmente las siguientes seis (6) carreras:

**I** significa Iterativo y **R** significa Recursivo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **# Carrera** | **Algoritmo** |  | **Algoritmo** |  | **Algoritmo** |
| 1 | AL - A - I | vs | LE - A - I | vs | ABB - A - I |
| 2 | AL - C - I | vs | LE - C - I | vs | ABB - C - I |
| 3 | AL - E - I | vs | LE - E - I | vs | ABB - E - I |
| 4 | AL - A - R | vs | LE - A - R | vs | ABB - A - R |
| 5 | AL - C - R | vs | LE - C - R | vs | ABB - C - R |
| 6 | AL - E - R | vs | LE - E - R | vs | ABB - E - R |

Cada uno de los algoritmos correrá en su propio hilo. Esto es equivalente, en las carreras de atletismo, a que cada atleta corra en su propio carril.

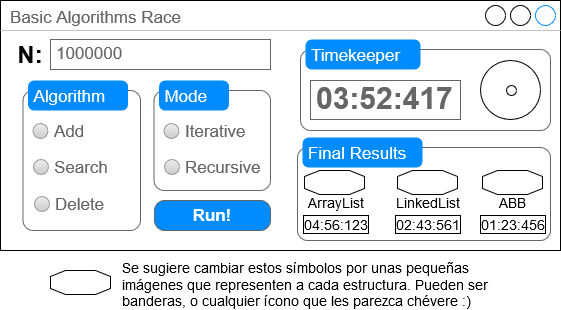
En el momento en que inicia la carrera, también iniciará un cronómetro que contará el tiempo en minutos:segundos:milisegundos transcurridos desde que inició la carrera. También iniciará una pequeña animación de un círculo que se reduce poco a poco hasta un punto y luego crece hasta el tamaño original del círculo y luego vuelve a reducirse, así hasta que termine la carrera y el último competidor termine. Ese círculo no estará solo, habrá otro círculo en la misma posición que hará lo mismo pero al contrario, se reduce cuando el otro aumenta y al contrario.

Los elementos a ser almacenados en las estructuras de datos anteriores son long. Recuerde que el mínimo valor de long es Long.MIN\_VALUE y el máximo valor es Long.MAX\_VALUE.

Usted deberá definir una GUI exclusivamente con JavaFX que permita correr las carreras anteriores con un N indicado por el usuario.

La GUI deberá tener al menos los elementos descritos en el siguiente wireframe. Dice *al menos*, porque usted puede agregar elementos adicionales que considere permitan mejorar la experiencia del usuario.

Los colores, posición y tamaño son de referencia, estas características de los elementos de la GUI pueden ser cambiados respecto de como están en el wireframe de ejemplo.



Una de las funcionalidades que se pueden hacer extra (adicionales, opcionales) es que se pueda ver el estado de cada algoritmo durante la carrera. Ya que la carrera de cada algoritmo consiste en hacer N operaciones, ver el estado de cada uno puede consistir en mostrar en la GUI cuantas operaciones lleva cada algoritmo hasta el momento, de esa manera se puede ver quién le está ganando a quien. Y así es mucho más parecido a los deportes!! Las personas estarán muy felices de ver como esos números se van actualizando. Si ya logra mostrar el número de operaciones ejecutadas por cada algoritmo y aún quiere hacer algo más, podría mostrar la carrera de una forma más visual aún. Puede hacerlo hacerlo con tres barras paralelas que se van llenando en la medida en que los algoritmos llevan a cabo más operaciones. Por ejemplo así:

AL

LE

ABB

Este laboratorio se calificará con la [Rúbrica para el Laboratorio 4 sobre Carrera de Algoritmos](https://docs.google.com/spreadsheets/d/1xFfLenGiA5exOA1uPx1JLNlRwXhae1DahySSmf5NZzQ/edit?usp=sharing).

Aclaraciones:

* En la estructura de datos de ArrayList no aplica agregar iterativo ni agregar recursivo, solo hagan el .add().
* En la estructura de datos de ArrayList al eliminar, no está permitido usar el remove(Object), desde luego si se puede usar el remove(int).
* La estructura de listas enlazadas puede ser doble pero no circular y sin puntero hacia el último elemento.
* Tenga en cuenta que la generación de los números aleatorios no se cuenta dentro del tiempo de cada operación de cada estructura de datos.
* Este laboratorio cubre los temas de 3 unidades, y aunque no vale por 3 laboratorios, su nota final si valdrá por 2 laboratorios de los anteriores. Es decir, los porcentajes de los laboratorios (para calcular la nota final de laboratorios) es Lab1 (20%), Lab2 (20%), Lab3 (20%) y Lab4 (40%).

**Requerimentos funcionales**

|  |  |
| --- | --- |
| **Name:** | R. #1. Agregar N elementos a la estructura ArrayList |
| **Summary:** | Agregar N elementos generados de manera aleatoria de tipo long a la estructura de ArrayList |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos agregados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #2. Agregar N elementos a la estructura lista recursivamente |
| **Summary:** | Agregar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera recursiva |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos agregados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #3. Agregar N elementos a la estructura lista de manera iterativa |
| **Summary:** | Agregar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera iterativa |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos agregados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #4. Agregar N elementos a la estructura ABB recursivamente |
| **Summary:** | Agregar N elementos generados de manera aleatoria de tipo long a la estructura ABB recursivamente |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos agregados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #5. Agregar N elementos a la estructura ABB de manera iterativa |
| **Summary:** | Agregar N elementos generados de manera aleatoria de tipo long a la estructura ABB de manera iterativa |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos agregados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #6. Eliminar N elementos de la estructura ArrayList recursivamente |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long de la estructura ArrayList recursivamente |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #7. Eliminar N elementos de la estructura ArrayList iterativamente |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long a la estructura de ArrayList |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #8. Eliminar N elementos de la estructura lista recursivamente |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera recursiva |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #9. Eliminar N elementos de la estructura lista de manera iterativa |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera iterativa |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #10. Eliminar N elementos de la estructura ABB recursivamente |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long a la estructura ABB de manera recursiva |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #11. Eliminar N elementos de la estructura ABB iterativamente |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera iterativa |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #12. Eliminar N elementos de la estructura ArrayList recursivamente |
| **Summary:** | Eliminar N elementos generados de manera aleatoria de tipo long de la estructura ArrayList recursivamente |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos eliminados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #13. Buscar N elementos de la estructura ArrayList iterativamente |
| **Summary:** | Buscar N elementos generados de manera aleatoria de tipo long a la estructura de ArrayList |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos buscados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #14. Buscar N elementos de la estructura lista recursivamente |
| **Summary:** | Buscar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera recursiva |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos buscados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #15. Buscar N elementos de la estructura lista de manera iterativa |
| **Summary:** | Buscar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera iterativa |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos buscados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #16. Buscar N elementos de la estructura ABB recursivamente |
| **Summary:** | Buscar N elementos generados de manera aleatoria de tipo long a la estructura ABB de manera recursiva |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos buscados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #17. Buscar N elementos de la estructura ABB iterativamente |
| **Summary:** | Buscar N elementos generados de manera aleatoria de tipo long a la estructura lista de manera iterativa |
| Enter: | int cantidadAlgoritmos |
|  |  |
| Results: | Algoritmos buscados correctamente |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #18. Iniciar un cronometro que contará el tiempo que demoré la carrera |
| **Summary:** | Se iniciará un cronometro que cuente el tiempo que demoré la carrera |
| Enter: |  |
|  |  |
| Results: | Cronometro |
|  |  |

|  |  |
| --- | --- |
| **Name:** | R. #19. Iniciar la animación de dos círculos que cambien su tamaño |
| **Summary:** | La animación de los dos círculos estaría constantemente cambiando su tamaño hasta que la carrera termine |
| Enter: |  |
|  |  |
| Results: |  |
|  |  |

**Requerimientos no funcionales**

* La estructura de datos ArrayList no aplica agregar iterativo ni agregar recursivo, solo se hace el add()
* La estructura de datos ArrayList no aplica remover un objeto utilizando el método propio de ArrayList .remove(Object)
* La estructura de listas puede ser doblemente enlazada, pero no puede ser circular
* No se cuenta dentro del tiempo de la carrera la generación aleatoria de los elementos
* Los elementos almacenados son tipo long
* Las figuras de los círculos deben de parar su hilo al momento que termine la carrera