

|  |
| --- |
| Documento de Análisis y Diseño |
| Nivel 13: [F1]  [Edgar Andrés Margffoy] |
| [Camila García] |
| Estructuras de Datos  Universidad de Los Andes  [2015-1] |
|  |

Documento de Análisis y Diseño

[Nombre del ejercicio]

|  |  |  |
| --- | --- | --- |
| **Versión del documento** | **Fecha de modificación** | **Responsables** |
| 1.0 | 29/01/15 | Edgar Andrés Margffoy  Camila García |
|  |  |  |
|  |  |  |

Contenido

[1 Requerimientos Funcionales 3](#_Toc409507633)

[2 Análisis del modelo del mundo 4](#_Toc409507634)

[3 Diseño de Interfaces 5](#_Toc409507635)

[3.1 Interfaces del mundo de la aplicación 5](#_Toc409507636)

[3.2 Interfaces de las estructuras de datos 5](#_Toc409507637)

[4 Diseño de estructuras de datos 6](#_Toc409507638)

[5 Justificación de decisiones de diseño de las estructuras de datos 7](#_Toc409507639)

[6 Diseño final de la aplicación 8](#_Toc409507640)

[7 Justificación de decisiones de diseño del mundo 9](#_Toc409507641)

[8 Análisis de complejidad de operaciones del mundo 10](#_Toc409507642)

[9 Mapa de Navegación (GUI) 11](#_Toc409507643)

# Requerimientos Funcionales

A continuación se describen los requerimientos funcionales de la aplicación. La definición de estos es independiente de la tecnología que será usada para implementarlos.

|  |  |
| --- | --- |
| **ID** | R1 |
| **Nombre** | Almacenar la información en cache para un periodo dado |
| **Resumen** | El usuario solicita una búsqueda de un periodo histórico y el programa realiza una consulta, cuyos resultados guarda en caché. |
| **Entradas** | Periodo: tupla de enteros positivos. Da el periodo en el cual se quiere buscar la información |
| **Resultados** | Confirmación periodo de obtención información. Boolean. |

|  |  |
| --- | --- |
| **ID** | R2 |
| **Nombre** | Consultar la información de pilotos, carreras o constructores para una temporada (año) |
| **Resumen** | El usuario escoge un tipo de información y una temporada. El programa las visualiza. |
| **Entradas** | Información deseada. (pilotos, carreras o constructores) Año deseado. Entero positivo. |
| **Resultados** | Se muestra la información completa disponible. |

|  |  |
| --- | --- |
| **ID** | R3 |
| **Nombre** | Obtener el resultado de pilotos de una carrera/circuito para una temporada (año) |
| **Resumen** | El usuario pide el resultaso de pilotos de una carrera/circuito para un año específico. El programa lo provee |
| **Entradas** | Año. Entero positivo. Año del que se desean los resultados. Nombre de carrera/circuito. String. Nombre de la carrera o circuito del cual se quieren conocer los pilotos. |
| **Resultados** | Se muestra la información de los pilotos en el orden de llegada. |

|  |  |
| --- | --- |
| **ID** | R4 |
| **Nombre** | Buscar las carreras realizadas en un periodo. |
| **Resumen** | EL usuario define el periodo de busqueda y el programa provee toda la información de las carreras realizadas en ese periodo en orden cronológico. |
| **Entradas** | Fecha inicio. Fecha. Fecha final. Fecha. Las dos fechas representan el periodo de la búsqueda. |
| **Resultados** | Se muestran las fechas, el nombre de la carrera, nombre del circuito, ciudad y el país de la información encontrada en orden cronológico. |

|  |  |
| --- | --- |
| **ID** | R5 |
| **Nombre** | Mostrar un carrusel donde se muestren las carreras de una temporada (año) |
| **Resumen** | Muestra en un carrusel todas las carreras de un año determinado y permite recorrerlas de manera circular. |
| **Entradas** | Año de consulta. Año. |
| **Resultados** | Se muestra el carrusel con las carreras del año que llega por parametro. |

|  |  |
| --- | --- |
| **ID** | R6 |
| **Nombre** | Obtener la información de los resultados de un piloto en una temporada (año) |
| **Resumen** | Permite conocer todas las carreras de un piloto específico |
| **Entradas** | Año de consulta. Año. Nombre de corredor. String |
| **Resultados** | Se muestran los puntos obtenidos por el piloto en la temporada y la información completa de las carreras de ese año. |

|  |  |
| --- | --- |
| **ID** | R7 |
| **Nombre** | Obtener la información de los resultados de un constructor en una temporada (año) |
| **Resumen** | Permite conocer la información de un constructor específico en una temporada dada |
| **Entradas** | Año de consulta. Año. Nombre de constructor. String |
| **Resultados** | Se muestra el total de puntos obtenidos en la temporada y el detalle de puntos de cada carrera |

|  |  |
| --- | --- |
| **ID** | R8 |
| **Nombre** | Eliminar un piloto en una temporada (año) |
| **Resumen** | Permite eliminar la información de un piloto del sistema |
| **Entradas** | Año deseado. Año. Nombre corredor. String |
| **Resultados** | Se eliminan los datos del conductor del sistema |

|  |  |
| --- | --- |
| **ID** | R9 |
| **Nombre** | Eliminar una carrera en una temporada (año) |
| **Resumen** | Permite eliminar la información de una carrera del sistema |
| **Entradas** | Año deseado. Año. Nombre carrera. String |
| **Resultados** | Se eliminan los datos de la carrera del sistema |

|  |  |
| --- | --- |
| **ID** | R10 |
| **Nombre** | Mostrar la carrera con mayor duración en el periodo de consulta |
| **Resumen** | Muestra la carrera con más duración en el periodo de consulta |
| **Entradas** | - |
| **Resultados** | Se muestra el año en el que se corrió la carrera, el nombre de la carrera, su duración y el piloto que la ganó |

# Análisis del modelo del mundo

Diagrama de clases (UML) de la abstracción del problema que se pretende solucionar. Solo se tienen en cuenta entidades, características y relaciones relevantes para el mismo. Aún no se toman decisiones de diseño.

Ver Anexo 1.

# Diseño de Interfaces

## Interfaces del mundo de la aplicación

A partir de los requerimientos funcionales, defina las operaciones (a través de métodos) que debe proveer el mundo del problema. Incluya parámetros, tipos de retorno y documentación completa de cada operación (incluyendo precondiciones).

A partir de los requerimientos funcionales, se definen las operaciones (a través de métodos) que debe proveer el mundo del problema. Incluya parámetros, tipos de retorno y documentación completa de cada operación (incluyendo precondiciones).

Basándose en el análisis exhaustivo y profundo de los requerimientos del problema, y comprendiendo la relación entre las principales entidades involucradas en el proceso de diseño de la solución. Fue necesario realizar una disección del problema en distintos subproblemas, estos subproblemas se basan en cuatro aspectos, el primero corresponde al almacenamiento de la información obtenida (Haciendo uso de Estructuras de Datos), el segundo se encuentra asociado a la obtención de la información desde un servidor localizado en Internet, ahora bien, esta información requiere ser abstraída en un modelo de relaciones y clases con el fin de ser representada eficientemente. Finalmente, todo este proceso debe ser concluido en la implementación de un ambiente de interacción Humano-Computador, i.e., Interfaz Gráfica. Este ambiente, al igual que los anteriores aspectos, además de ser eficiente y prolijo en sus operaciones, debe ser amigable e intuitivo con el usuario final.

Teniendo en cuenta los aspectos mencionados anteriormente, se observa una dependencia abstracta entre cada módulo de la implementación, por ejemplo, la implementación de una lista lineal es usada en el modelo de clases, no obstante, la operación de este, no es dependiente en la implementación de las mismas, y por lo tanto, deben ser implementadas a través de una interfaz que describa como debería comportarse una estructura lineal. Asimismo, se observa que el ambiente de interacción con el usuario usa información proveída por el mundo. No obstante, el funcionamiento de la primera, no debe depender de la implementación y diseño de la segunda, y por lo tanto, requiere ser mediada a través de una interfaz pertinente.

## Interfaces de las estructuras de datos

Seleccione cuáles son las estructuras de datos genéricas que debe utilizar y las operaciones que estas deben proveer para minimizar la complejidad temporal de las operaciones definidas en el numeral 3.1.

Se definen las operaciones básicas que toda estructura lineal debe cumplir, tal como operaciones de agregar, eliminar, buscar por una posición, determinar si un elemento existe al interior de esta estructura.

A lo largo de la implementación de las estructuras se determinaron dos estructuras principales a usar, la primera (Lista [Doblemente] Enlazada) basa su funcionamiento y eficiencia al agregar elementos, no obstante, requiere al menos O(N) operaciones para buscar un elemento.

La segunda estructura implementada (Arreglo Dinamico), si bien tiene un tiempo de acceso de O(1) operaciones, en el peor de los casos, requiere O(N) operaciones para agregar un elemento.

# Diseño de estructuras de datos

Describa el detalle de las Estructuras de Datos que va a utilizar en su solución, a través de un diagrama UML de las mismas. No olvide incluir invariantes si las requiere.

Ver Anexo 2.

# Justificación de decisiones de diseño de las estructuras de datos

Describa en detalle los aspectos que tuvo en cuenta para seleccionar y diseñar las estructuras de datos descritas en el punto anterior.

Para empezar, sabíamos que debíamos utilizar una lista enlazada, así que partimos de ahí. Luego, pensamos en la necesidad de devolverse, en caso de que por ejemplo, se necesitara la carrera o piloto anterior. Para evitar que esta operación tuviera que darle la vuelta entera a la lista, nos decidimos por una lista doblemente enlazada.

Sin embargo, ya que una de las principales funciones de la aplicación es la búsqueda, incluímos también un Arreglo dinámico que nos permitiera, más tarde, poder implementar una búsqueda binaria, disminuyendo en gran medida la complejidad de buscar dentro de las listas.

Ya que teníamos dos listas diferentes con casi las mismas funciones, decidimos utilizar una interface para permitir el desacoplamiento y la fácil edición de la implementación de las listas en el futuro.

Dentro de la lista enlazada, no sólo pusimos un apuntador al primer elemento, sino también al último. De esta manera, agregar elementos al final de la lista se convierte en una operación de complejidad O(1), y no O(n), que sería si no tuviéramos la referencia al último elemento. Por otro lado, se lleva siempre el atributo de length, actualizado cada vez que se agreguen o eliminen elementos. De esta manera, conocer la longitud de la lista es una operación O(1), no O(n) en caso de que no tuviéramos este atributo.

Dentro del Array dinámico incluimos un iterador que permite avanzar hacia adelante y hacia atrás. De esta manera, cambiar entre uno y otro se vuelve fácil, y facilita la implementación de métodos donde se van a tener que recorrer las listas, como recorrer las carreras o escuderías de un año.

Además del iterador normal, decidimos crear también un iterador reverso, para poder recorrer la lista al revés y aprovechar así al máximo el potencial que nos ofrece una lista doblemente enlazada. Por eso, utilizamos una interfaz para los iteradores.

# Diseño final de la aplicación

Describa en detalle el diseño final de su aplicación a través de un diagrama UML, incluya atributos y métodos. Cuando se refiera a una estructura de datos no es necesario que incluya toda su definición dentro del diagrama. Basta con referenciar su interfaz.

# Justificación de decisiones de diseño del mundo

Describa en detalle los aspectos que guiaron el diseño final de su aplicación.

Para crear el mundo, pensamos en las diferentes necesidades del programa basados en los requerimientos funcionales. Primero, sabíamos que el programa debía estar dividido en temporadas (años), pues la mayoría de requerimientos funcionales se centran en búsquedas de años específicos. Luego, analizamos los requerimientos y nos dimos cuenta que casi todas las búsquedas se realizan dentro de un solo año, por lo que era necesario que cada año contuviera una lista de sus pilotos, circuitos y escuderías.

Para empezar, pensamos en la necesidad de una interface que permitiera la fácil conexión con la interfaz ignorando la implementación del mundo. En la interface incluímos todos los métodos necesarios para el cumplimiento de los requerimientos funcionales. Más allá, en las clases de carrera, piloto y escudería, incluimos atributos para permitir que se guardase la información necesaria proveniente del servidor de eargast, que luego se utilizaría para mostrarse en la interfaz. En la práctica, debimos hacer algunos cambios a estos atributos por cómo se consultaba la información, pero la mayoría permanecieron útiles.

Para las consultas, creamos un paquete completamente nuevo, Query, que se encarga de hacer las consultas y de persistir la aplicación. Pensamos que ésta era una manera mucho más ordenada de repartir responsabilidades que además permitía una alta cohesión.

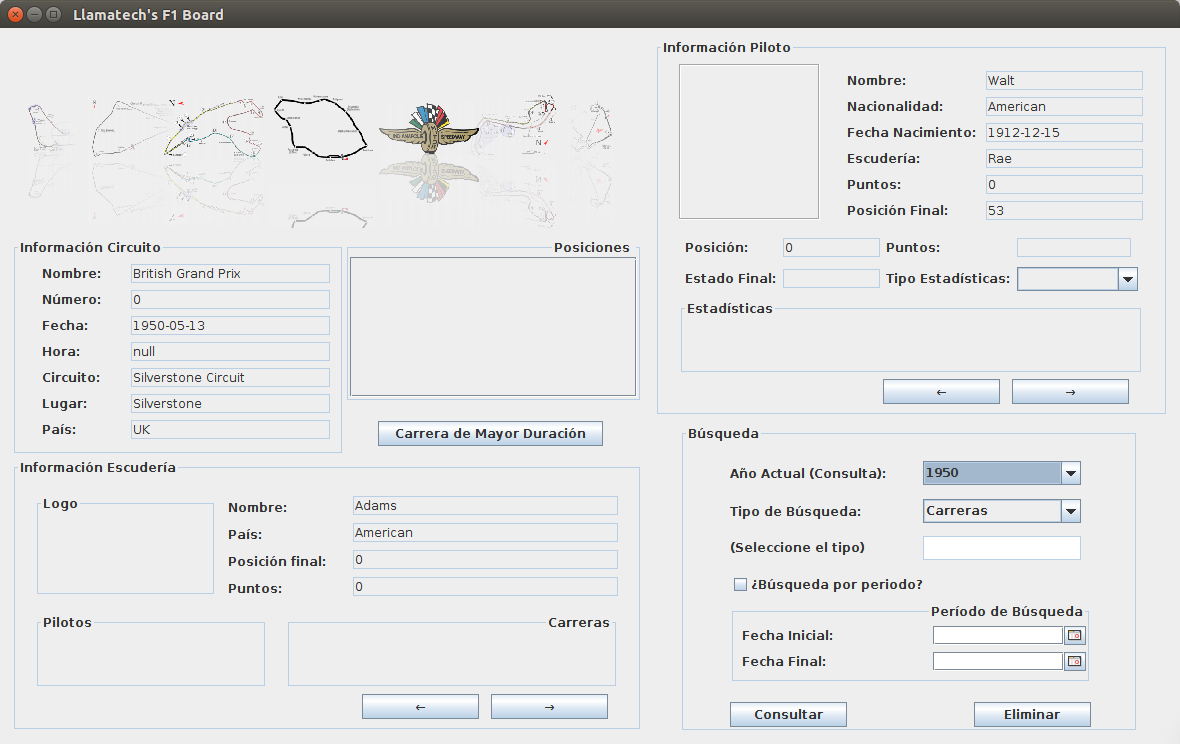
Para guardar las listas, tomamos un arreglo de tamaño fijo para las temporadas, pues es un valor que se conoce desde que se abre la aplicación y permite hacer búsquedas de años específicos muy eficientes. Por otro lado, para las listas de carreras, pilotos y escuderías, utilizamos nuestras estructuras de datos. Para pilotos y escuderías utilizamos Arrays dinámicos, pues estos permiten una búsqueda eficiente y se dejan recorrer hacia adelante y hacia atrás. Para las carreras, utilizamos una lista doblemente enlazada, que funcionaba muy bien con nuestro propósito de rotar las listas de un año con un carrusel de imágenes.

# Análisis de complejidad de operaciones del mundo

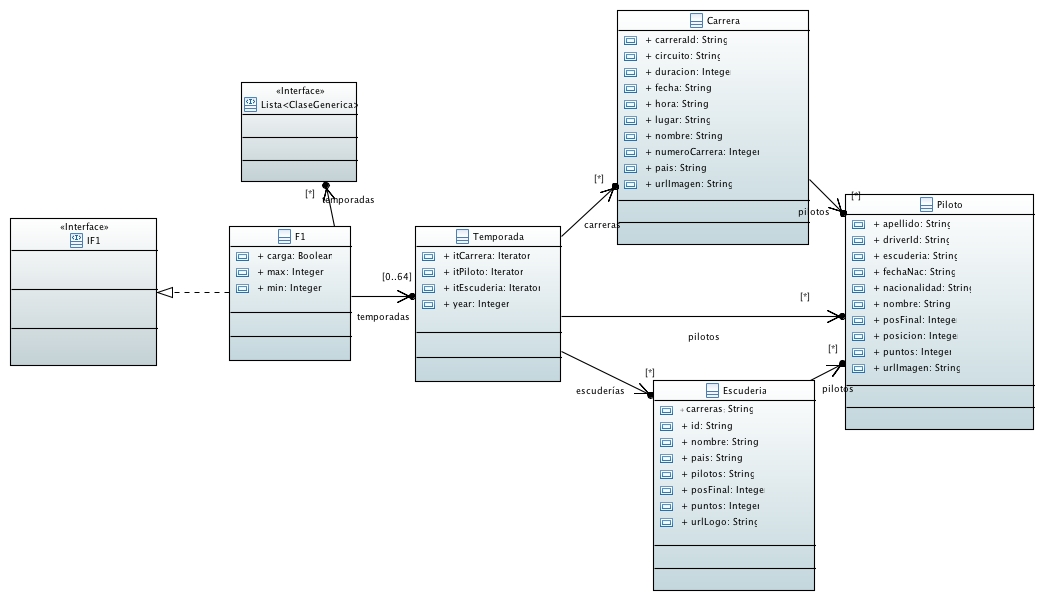
Basado en los diseños de las estructuras de datos y en el mundo de la aplicación, defina la complejidad de cada una de las operaciones. Utilice la notación O(f(n)). Justifique por qué considera que esta es la mínima posible para el problema que está resolviendo.

# Mapa de Navegación (GUI)

Basado en los requerimientos funcionales, defina un bosquejo general con los elementos que definen la interfaz gráfica (GUI) de la aplicación. Puede usar la herramienta *Balsamiq Mockups* (http://balsamiq.com/products/mockups/) u otra que permita una funcionalidad similar (NO requiere realizar la implementación de la Interfaz Gráfica)



Anexo 1.



Anexo 2. (También disponible en el pryecto en . docs/UML/Carpeta imágenes/Diagrama\_Estructuras.JPEG)

