Requerimientos Funcionales

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 1** |
| **Resumen** | Buscar sitios de interés a partir de una posición y un radio de búsqueda |
| **Entradas** | Latitud, longitud y radio de búsqueda |
| **Salidas** | Lista con los sitios de interés |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 2** |
| **Resumen** | Refinar resultado por categorías |
| **Entradas** | - |
| **Salidas** | Lista refinada |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 3** |
| **Resumen** | Guardar el resultado de una búsqueda |
| **Entradas** | - |
| **Salidas** | Resultado de la búsqueda |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 4** |
| **Resumen** | Crear categorías |
| **Entradas** | Datos de la categoría |
| **Salidas** | Nueva categoría |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 5** |
| **Resumen** | Crear un evento en un lugar determinado |
| **Entradas** | Lugar del evento  Datos del evento |
| **Salidas** | Nuevo evento |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 6** |
| **Resumen** | Consultar los eventos que han sido cancelados |
| **Entradas** | - |
| **Salidas** | Lista de los eventos cancelados |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 7** |
| **Resumen** | Enviar invitaciones a un evento por correo electrónico |
| **Entradas** | Evento  Destinatarios |
| **Salidas** | Correo electrónico con los datos del evento |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 8** |
| **Resumen** | Eliminar un evento y notificar a los invitados |
| **Entradas** | Identificador del evento |
| **Salidas** | Correo electrónico con la notificación |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 9** |
| **Resumen** | Buscar los eventos a los que ha sido invitado un usuario |
| **Entradas** | Correo electrónico del usuario |
| **Salidas** | Lista de eventos a los que ha sido invitado |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 10** |
| **Resumen** | Buscar los destinatarios que han sido invitados a un evento |
| **Entradas** | Identificador del evento |
| **Salidas** | Lista de destinatarios |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 11** |
| **Resumen** | Mostrar un resumen de todos los eventos que existen |
| **Entradas** | - |
| **Salidas** | Datos de todos los eventos disponibles |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 12** |
| **Resumen** | Encontrar eventos que se han realizado cerca a partir de otro evento o un radio de búsqueda |
| **Entradas** | Evento  Radio de búsqueda |
| **Salidas** | Eventos más cercanos |

|  |  |
| --- | --- |
| **Req.** | **Requerimiento 13** |
| **Resumen** | Consultar el evento con más invitados |
| **Entradas** | - |
| **Salidas** | Evento con más invitados |

Estructuras de Datos

En el ejercicio es necesario el uso de árboles AVL, pues debido a su recursividad y auto balance, la manipulación de información en el árbol, tanto agregar, eliminar y buscar, se realizan en una cómoda complejidad de . Para mantener esta característica, la estructura debe pasar por una serie de rotaciones especiales a la hora de agregar y eliminar, las cuales tienen complejidad constante, pues solo se realizan un par de asignaciones nuevas a lo sumo, y de esta forma se optimiza el uso de la estructura. A continuación se muestra la interfaz con las operaciones críticas de la estructura.

**package** estructuras;

**import** java.util.Iterator;

**import** java.io.Serializable;

**public interface** IArbolAVL<K,V> **extends** Iterable<V>, Serializable

{

/\*\*

\* Agrega un objeto a la estructura

\* **@param** objeto Objeto que se quiere agregar

\* **@param** llave Llave de acceso al objeto

\*/

**publicvoid** agregar(K llave, V objeto);

/\*\*

\* Elimina un objeto de la lista a partir de su llave

\* **@param** llave Llave de acceso al objeto

\* **@return** Objeto eliminado

\*/

**public** V eliminar(K llave);

/\*\*

\* Busca un objeto en la tabla a partir de su llave

\* **@param** llave Llave de acceso al objeto

\* **@return** Objeto que se busca

\*/

**public** V buscar(K llave);

/\*\*

\* Verificasiunobjetoseencuentraenlatabla

\* **@param** objeto Objeto que se quiere verificar

\* **@return** true si se encuentra en la estructura. false de lo contrario

\*/

**public boolean** contiene(V objeto);

/\*\*

\*

\* Verifica si un objeto se encuentra en la estructura

\* **@param** llave Llave del objeto que se quiere verificar

\* **@return** true si se encuentra en la estructura. false de lo contrario

\*/

**public boolean** contieneLlave(K llave);

/\*\*

\* Elimina todos los objetos de la estructura

\*/

**public void** limpiar( );

/\*\*

\* Retorna el número de elementos agregados a la estructura

\* **@return** El tamaño de la estructura

\*/

**public int** tamano( );

/\*\*

\* Retorna un iterador por defecto del arbol (Pre-orden)

\*/

@Override

**public** Iterator<V> iterator( );

/\*\*

\* Retorna un iterador en un recorrido in-orden

\*/

@Override

**public** Iterator<V> inOrden( );

/\*\*

\* Retorna un iterador en un recorrido post-orden

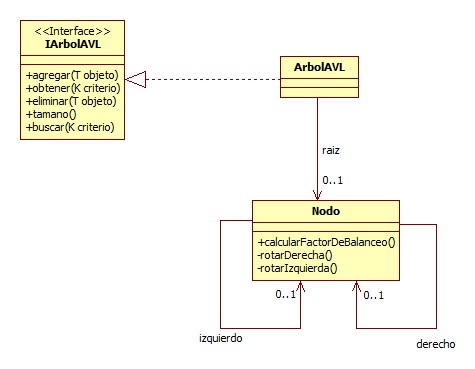
\*/

@Override

**public** Iterator<V> postOrden( );

}

A continuación un diagrama UML inicial de la estructura implementada.



El uso de listas definidas en el primer ejercicio también serán de utilidad, pues su uso es prácticamente universal y se adecuan a cualquier tipo de problema.

Mundo del Problema

Son 13 requerimientos funcionales descritos al inicio del documento. A continuación la interfaz principal junto con un diagrama UML analizando el problema.

**package** mundo;

**import** interfaces.IListaEstatica;

**import** interfaces.IArbolAVL;

**public interface** ICupiEventos

{

/\*\*

\* Busca los sitios de interés a partir de una posición y un radio de \* búsqueda

\*/

**public** IListaEstaticabuscarSitiosInteres (**int** latitud, **int** longitud, **int** radio);

/\*\*

\* Refinar búsqueda

\*/

**public void** refinarBusqueda ();

/\*\*

\* Guarda el resultado de una búsqueda

\*/

**public void** guardarResultado ();

/\*\*

\* Crea una categoría y permite agregar lugares

\*/

**public void** crearCategoria(Categoria datos);

/\*\*

\* Consulta los eventos que han sido cancelados

\*/

**public** IListaEstaticaeventosCancelados ();

/\*\*

\* Envia una invitación de un evento a una lista de destinatarios

\*/

**public void** enviarInvitacion (String[] destinatarios);

/\*\*

\* Elimina un evento

\*/

**public void** eliminarEvento (Evento datos);

/\*\*

\* Una lista con los eventos a los que el destinatario ha sido invitado

\*/

**public** IListaEstatica eventosPorDireccion (String direccion);

/\*\*

\* Lista de invitados a un evento especifico

\*/

**public** String[] invitados(Evento datos);

/\*\*

\* Muestra un resumen de todos los eventos

\*/

**public void** mostrarResumen ();

/\*\*

\* Busca los eventos cercanos a un evento, dentro de un radio de búsqueda

\*/

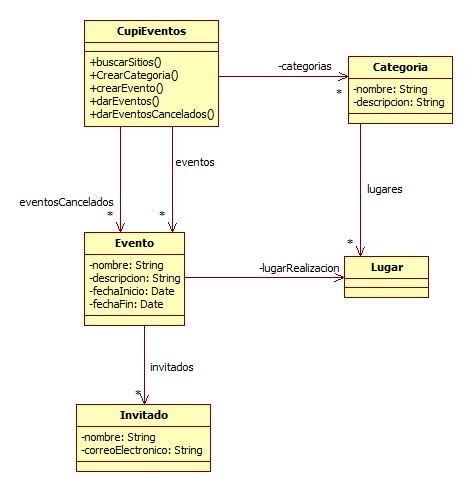
**public** IListaEstaticaeventosCercanos (Evento datos, **int** radio);

/\*\*

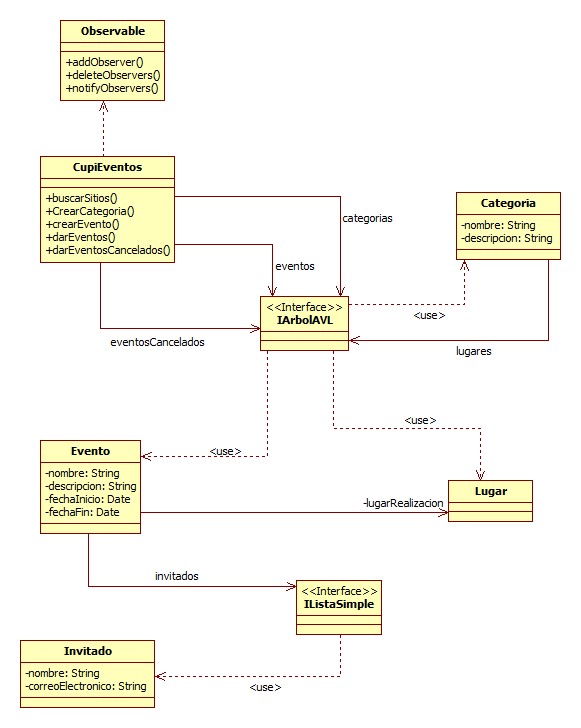
\* Evento con más invitados

\*/

**public** EventomasInvitados ();



A continuación el diagrama de diseño inicial, más específico que muestra el uso correcto de las estructuras implementadas en el problema.



Complejidades

|  |  |
| --- | --- |
| **Requerimiento** | **Req1** |
| **Complejidad** | – Se espera tener un árbol disponible para consultar eventos dada su posición |

|  |  |
| --- | --- |
| **Requerimiento** | **Req2** |
| **Complejidad** | – La refinación se realiza directamente de la interfaz, y a pesar de que la eliminación de un elemento tiene complejidad constante, la lista debe ser modificada. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req3** |
| **Complejidad** | – Una vez se tienen los datos de la búsqueda, guardarla consta de un par de asignaciones. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req4** |
| **Complejidad** | – Se espera que las categorías se encuentren disponibles en una lista y no en un árbol. Adicionar lugares a la categoría tendrá la misma complejidad. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req5** |
| **Complejidad** | – Al crear un evento, estos deben ser guardados en el árbol AVL. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req6** |
| **Complejidad** | – Resulta fácil la obtención de los eventos cancelados, pues solo consta de asignarlos a una lista en un atributo del mundo. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req7** |
| **Complejidad** | – Solo consta de enviar los datos por correo electrónico. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req8** |
| **Complejidad** | – Eliminar un evento del árbol de eventos tiene esta complejidad. Así mismo, enviar una notificación a los invitados tiene una complejidad constante. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req9** |
| **Complejidad** | – La lista de eventos y sus destinatarios serán guardados en una matriz ordenada, por lo tanto la búsqueda por destinatario será en esta complejidad. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req10** |
| **Complejidad** | – Si el evento ya es dado, consultar la lista de invitados es solo un llamado a un atributo. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req11** |
| **Complejidad** | – Se deben iterar todos los elementos del árbol, para poder mostrar un resumen de todos estos eventos. |

|  |  |
| --- | --- |
| **Requerimiento** | **Req12** |
| **Complejidad** | – Se deben iterar todos los elementos del árbol, y comparar su cercanía con la posición especificada |

|  |  |
| --- | --- |
| **Requerimiento** | **Req13** |
| **Complejidad** | – El evento con más invitados se puede guardar en un atributo que se verifica constantemente. |

Diseño Modificado