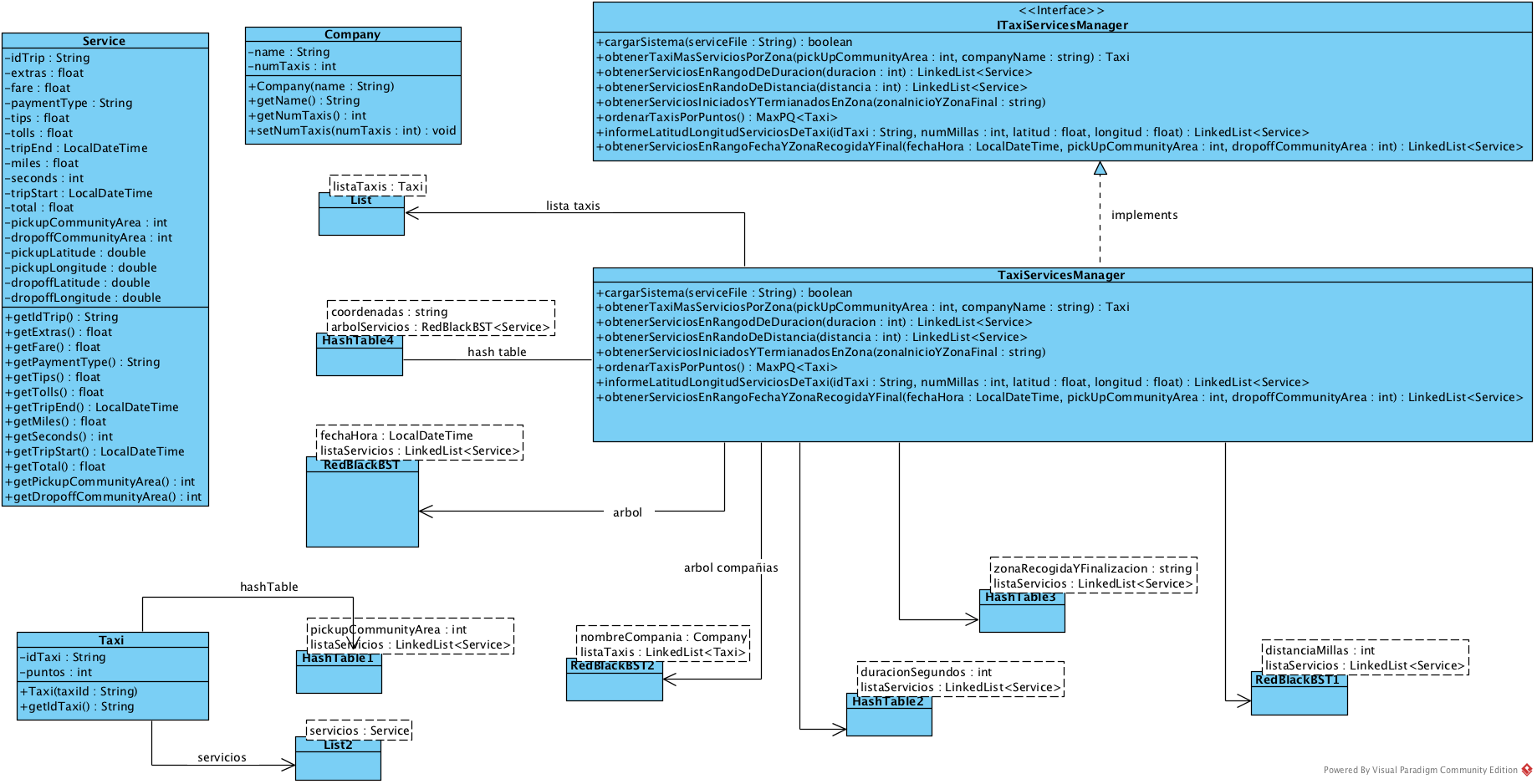
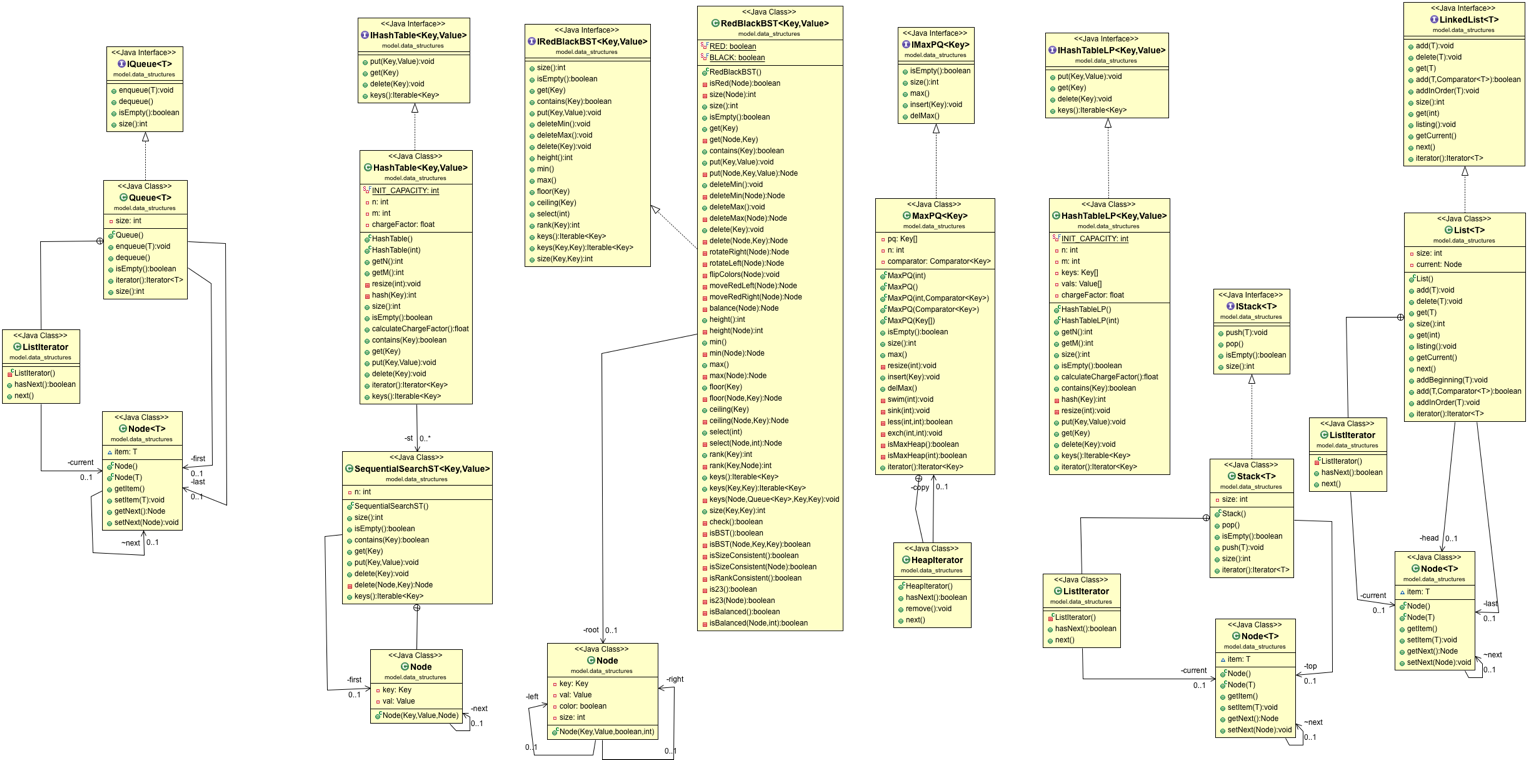
**Documento de Diseño Proyecto 2**

1. **Diagrama UML del mundo:**

****

1. **Diagrama UML de estructuras de datos:**

****

**3.** Estimación de complejidades de requerimientos:

1. Para el primer requerimiento (1A), usaremos como método de ordenamiento, un árbol binario rojo y negro. La llave será un objeto “Company” y el valor será una lista de objetos “Taxi”. Cada objeto “Taxi” tendrá una HashTable que tendrá como llave un entero que identificará el “pickup community área” y su valor será una lista de objetos “Service”.

Para encontrar la complejidad del requerimiento es necesario sumar la complejidad de acceder a un valor de el árbol y acceder a un valor en la tabla de hash lo cual es:

* C: Número de compañías, es decir número de llaves en el árbol.
* T: Numero de taxis de la compañía a buscar en el árbol balanceado.
* A: Número de “Community áreas” de cada taxi.
* M: Número de entradas del arreglo contenido en la tabla de hash que tiene como llave una lista de servicios.

Simplificando términos obtenemos que la complejidad es lineal:

1. Para el segundo requerimiento (2A), se usará una tabla de hash que tendrá como llave un entero que será la duración en segundos redondeado para asi generar un rango en el cual se guardaran los valores. El valor de la tabla de símbolos en la tabla de hash será una lista de servicios. Para acceder a esa lista de servicios el algoritmo tendrá una complejidad de:

G: Numero de grupos de rangos de segundos, es decir el numero de llaves en la HashTable.

M: Número de entradas del arreglo de la HashTable.

1. Para el tercer requerimiento (1B), se usará como método de ordenamiento, un árbol binario rojo y negro. La llave será un entero y el valor será una lista de servicios. Para acceder a la lista de servicios, que es lo que nos pide el requerimiento, simplemente se necesita buscar en el árbol. Por lo cual el requerimiento tendrá una complejidad de:
2. Para el cuarto requerimiento (2B), usaremos una tabla de hash para agrupar los servicios, teniendo como llave un “string” que identificara la zona de recogida y llegada del árbol balanceado de servicios asociado, siendo así este arbol el valor asociado a la llave de esa tabla de símbolos. Para encontrar la complejidad del algoritmo sumamos las contribuciones de la complejidad de cada estructura de datos:

* A: Número de llaves en la tabla de hash.
* M: Número de entradas en el arreglo contenido en la tabla de hash.
* E: Número de llaves en el árbol balanceado el cual es el valor de cada llave en la tabla de hash.

El término será un número muy pequeño, sabiendo que en una tabla de hash que utiliza “Separate Chaining” el número máximo de elementos en cada lista contenida en cada entrada del arreglo contenido en la tabla de hash nunca es mayor que 6 por el factor de carga que es manejado. Simplificando la expresión obtenemos que la complejidad del algoritmo es:

1. Para el quinto requerimiento (1C) necesitamos primero recorrer completamente la lista encadenada que contiene todos los taxis registrados en el sistema, para de esta manera poder calcular el puntaje de cada taxi. Posteriormente se organizara esta lista con el algoritmo HeapSort y se utilizara como criterio de ordenamiento el puntaje de cada taxi. Por ende la complejidad del requerimiento es sumar la contribución que hace el recorrido total de la lista de taxis y la complejidad del ordenamiento HeapSort. La expresión tilda es:

* T: Número de taxis en la lista.

Simplificando la expresión obtenemos que la complejidad del algoritmo es:

1. Para el sexto requerimiento (2C), se tiene una tabla de hash para guardar los servicios con ciertos parámetros, la cual tendrá complejidad de:

* S: Servicios, se asigna de esta manera, refiriéndose a la cantidad de servicios.

De igual manera, estos servicios se representaran en un árbol balanceado rojo y negro de complejidad: .

1. Para el séptimo y último requerimiento (3C), utilizaremos un árbol balanceado rojo y negro que tendrá como llave un rango en minutos y como valor una lista de servicios. Por ende, la complejidad del requerimiento es la misma complejidad que tiene buscar un elemento en un árbol balanceado rojo y negro la cual es: