PROGRAMACIÓN ORIENTADA A OBJETOS

TRABAJO PRÁCTICO FINAL

Integrantes:

- Cabrera, Claudio / claudiocabrera12@gmail.com

- Iarussi, Diego / diego.iarussi@gmail.com

INTRODUCCIÓN:

Se nos solicitó realizar un modelado de objetos de una ciudad digital, la cual está formada por calles y lugares.
Cada calle tiene un nombre que lo identifica y está formada por tramos, los cuales poseen características propias. Por ejemplo, una misma calle puede tener una velocidad máxima en un tramo y otra velocidad en otro. O puede ser doble mano en un tramo y una mano en otro. Por otra parte, los lugares también tienen un nombre que los identifica y características propias que lo definen.
Tanto los tramos como los lugares poseen una forma que los define. Estas, poseen un conjunto de vértices donde cada uno está formado por dos atributos: latitud y longitud. Las formas pueden ser: punto, línea, polígono, multipunto, multilínea o multipoligono.

Objetivo General

- ***Buscar una calle o lugar por el nombre:*** Se debe poder ingresar el nombre de una calle/lugar, y si existe dicha búsqueda, el programa deberá devolver los vértices que conforman la búsqueda solicitada.
- ***Buscar la ubicación de una calle con altura:*** Dado el nombre de una calle y la altura (número entero), si existe, el programa deberá devolver la longitud y latitud del vértice de dicha calle.
- ***Buscar la ubicación de la intersección entre dos calles:*** El programa solicita que se ingresen dos calles, si existen y se intersectan, deberá devolver el vértice de dicha intersección.
- ***Buscar el punto más cercano entre una calle y un lugar:*** El servicio solicita que se ingrese una calle y un lugar, si existen, deberá devolver el punto más cercano entre la calle y la forma del lugar, devolviendo entonces el vértice de dicho punto.

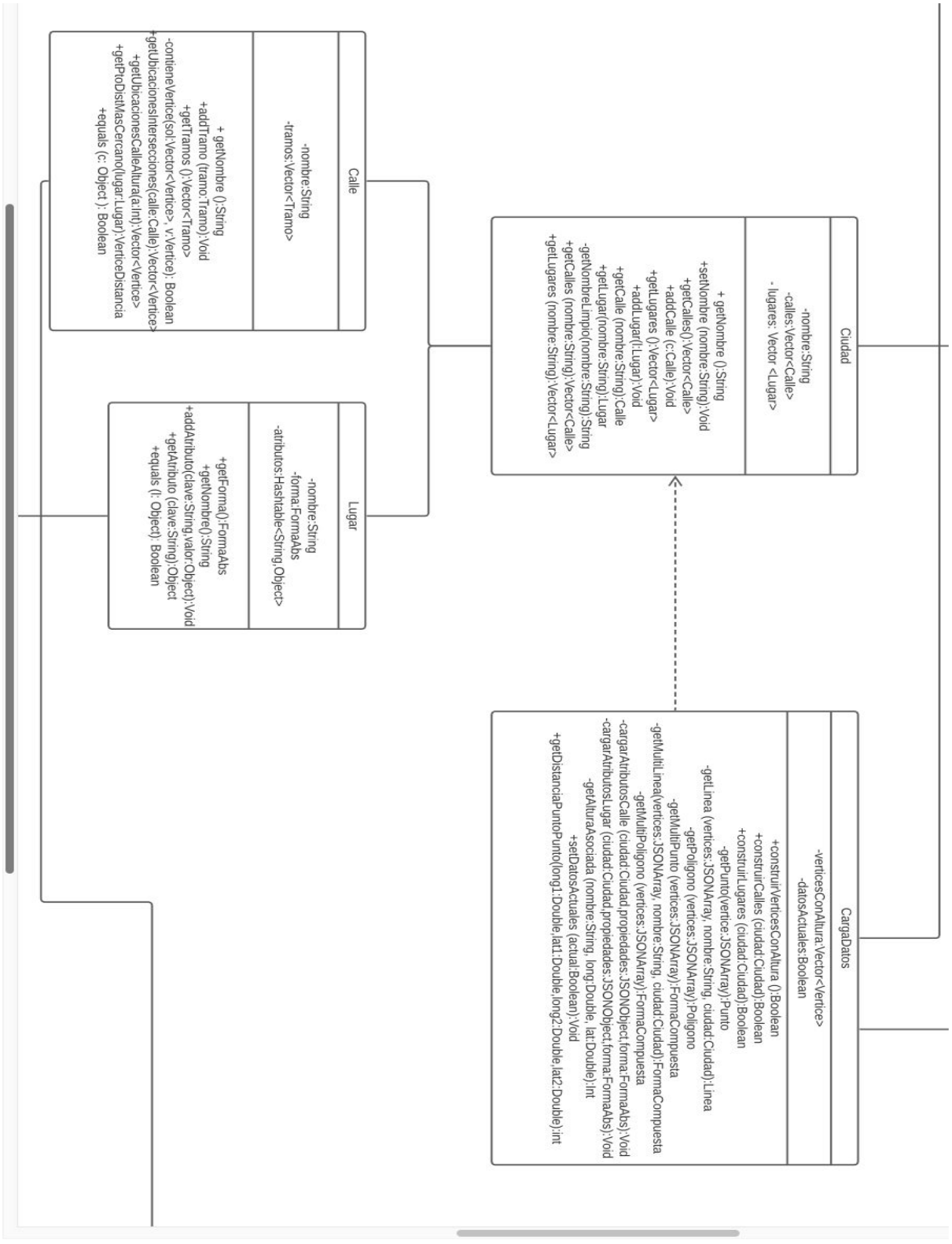
En la sección siguiente se explicará de manera detallada todas las decisiones de implementación tomadas que nos llevaron al desarrollo de la ciudad digital solicitada.

DECISIONES DE IMPLEMENTACIÓN:

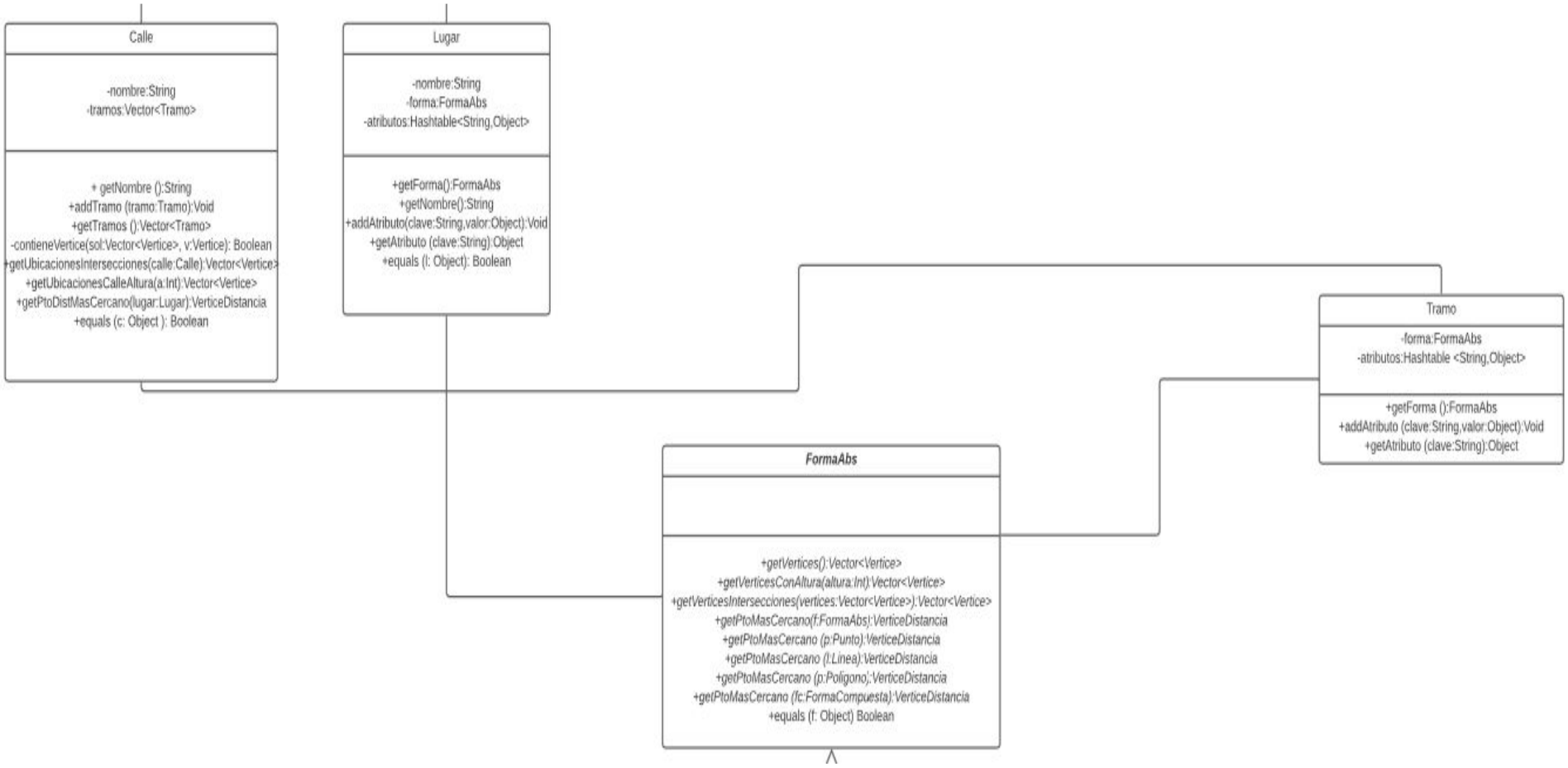
A la hora de realizar la implementación, se decidió utilizar el lenguaje de programación Java, por cuestiones de comodidad, ya que dicho lenguaje lo veníamos aplicando durante la cursada de la materia. El IDE que se utilizó fue Eclipse.
Al afrontar el diseño de la ciudad digital, nos encontramos con un problema inicial ya que para poder tener los datos de las calles, los lugares y todos los atributos que tenían estos objetos, había que recuperarlos en formato JSON desde la API overpass-turbo. El hecho de tener archivos con la extensión .geojson fue un problema, ya que el IDE de Eclipse no posee una librería que permita manipular estas extensiones. Por lo tanto, hicimos uso de la librería llamada “JSON.Simple”.

Diseño de la estructura

Para el diseño de la estructura, lo primero que se identificó fueron las clases “Calle” y “Lugar”. De estas clases, se llegó a que para construir la ciudad digital, era necesario una clase “Ciudad”, la cual contiene un conjunto de calles y lugares. Esta clase, es la encargada de generar la ciudad agregando todas las calles y lugares.
Una clase que se modeló y de la cual no se pensó en base a la programación orientada a objetos, fue la clase “CargarDatos”. Esta clase, se creó para tener un código limpio y ordenado, ya que se encarga de leer los archivos Geojson descargados de overpass-turbo y crear todos los objetos con sus características correspondientes. Entonces, para que no quedara todo en el main y sea difícil de entender, se decidió crear esta clase que, junto con la clase “Ciudad”, son las encargadas de levantar la estructura de la ciudad digital.
En esta clase, también se realiza la asociación de vértices con altura en el momento de la creación de las calles de la ciudad. Esto es, se recupera de overpass-turbo un archivo(además del archivo de calles.geojson y lugares.geojson) con los vértices que poseen el atributo addr:housenumber(vertices.geojson) y, cada vez que se crea una calle, por cada vértice agregado a la forma que posee cada tramo de dicha calle, se calcula si existe algún vértice con altura que esté a una distancia inferior a 30 mts. Para poder realizar esta asociación, se utiliza una función que calcula la distancia entre dos puntos en la tierra(lat/lng) diseñada por Haversine.

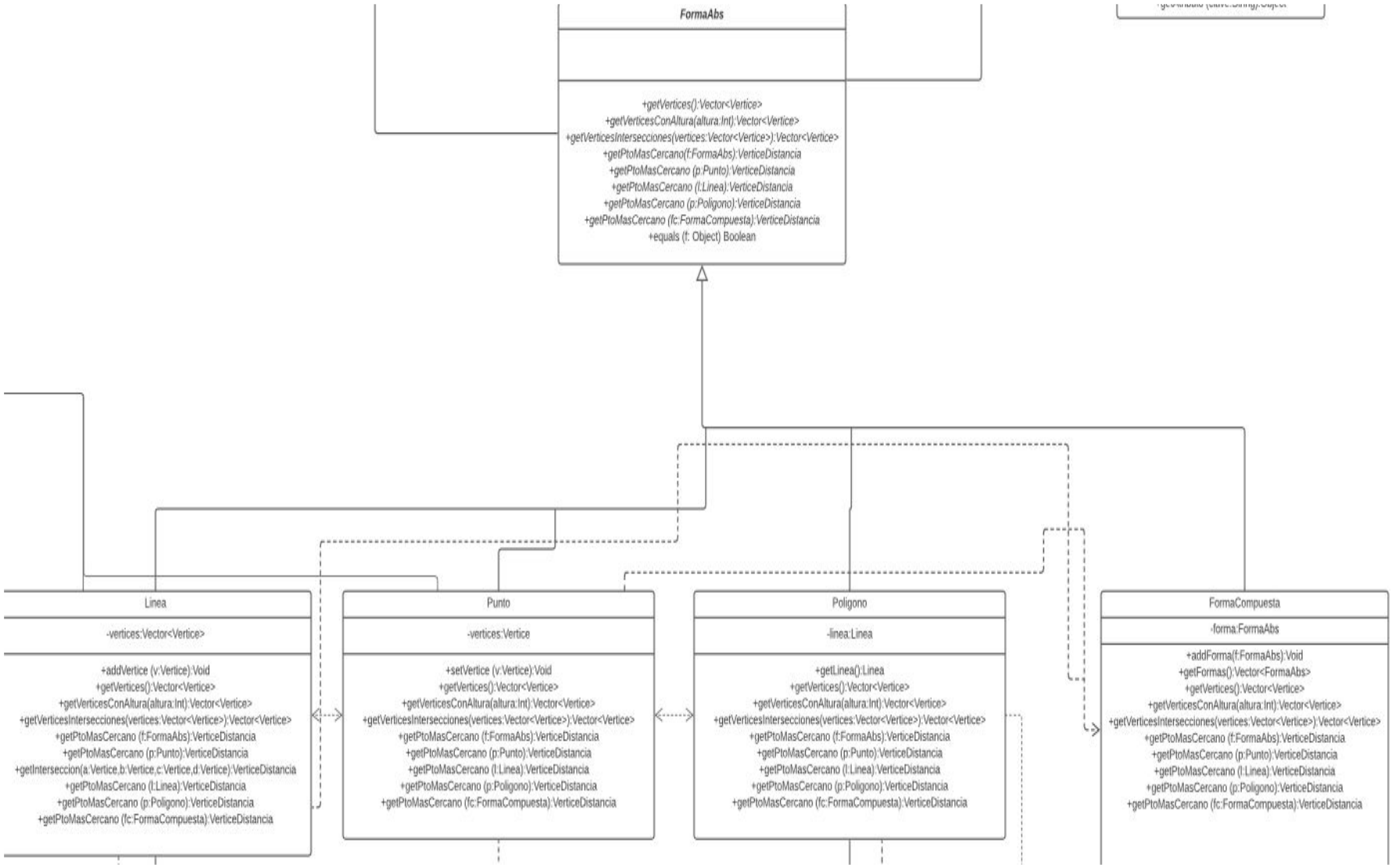


Como bien se mencionó en la introducción, una calle está formada por distintos tramos que contienen características propias. De esta manera, una calle contiene un conjunto de la clase “Tramo”, además del nombre de la misma. Tanto los tramos como los lugares poseen una forma que los describe, es decir, poseen una “FormaAbs”.


Calle
-nombre:String
-tramos:Vector<Tramo>
+ getNombre ():String
+addTramo (tramo:Tramo):Void
+getTramos ():Vector<Tramo>
-contieneVertice(sol:Vector<Vertice>, v:Vertice): Boolean
+getUbicacionesIntersecciones(calle:Calle):Vector<Vertice>
+getUbicacionesCalleAltura(a:Int):Vector<Vertice>
+getPtoDistMasCercano(lugar:Lugar):VerticeDistancia
+equals (c: Object): Boolean
Lugar
-nombre:String
-forma:FormaAbs
-atributos:Hashtable<String,Object>
+getForma():FormaAbs
+getNombre():String
+addAtributo(clave:String,valor:Object):Void
+getAtributo (clave:String):Object
+equals (l: Object): Boolean
Tramo
-forma:FormaAbs
-atributos:Hashtable <String,Object>
+getForma ():FormaAbs
+addAtributo (clave:String,valor:Object):Void
+getAtributo (clave:String):Object
FormaAbs
+getVertices():Vector<Vertice>
+getVerticesConAltura(altura:Int):Vector<Vertice>
+getVerticesIntersecciones(vertices:Vector<Vertice>):Vector<Vertice>
+getPtoMasCercano(f:FormaAbs):VerticeDistancia
+getPtoMasCercano (p:Punto):VerticeDistancia
+getPtoMasCercano (l:Linea):VerticeDistancia
+getPtoMasCercano (p:Poligono):VerticeDistancia
+getPtoMasCercano (fc:FormaCompuesta):VerticeDistancia
+equals (f: Object) Boolean

La clase abstracta “FormaAbs” obliga a sus hijos a definir ciertos métodos abstractos, los cuales se utilizan para la correcta implementación de los servicios solicitados. Dicha implementación, se detallará más específicamente en la sección: “Implementación de los servicios solicitados”.
Las formas que heredan de “FormaAbs” poseen un conjunto de la clase “Vertice”, dicha clase contiene dos atributos principales: latitud y longitud. Los dos atributos restantes que posee esta clase(altura y nombre) se utilizan para la asociación de altura a los vértices que forman los tramos que componen las calles, mencionada anteriormente.
Las formas pueden ser:

- “Punto”: Posee un “Vértice” y puede ser utilizada para definir un lugar.
- “Linea”: Posee un conjunto de “Vértice” y puede ser utilizada para definir una calle.
- “Poligono”: Posee una “Linea” que representa el área que forma el polígono. Con esta implementación se ahorra la duplicación de código de la clase “Linea”. También puede ser utilizada para definir un lugar.
- “FormaCompuesta”: Posee un conjunto de la clase “FormaAbs”. Con esta implementación, se pueden representar las formas compuestas siguientes: MultiPunto(puede utilizarse para definir un lugar), MultiLinea(puede utilizarse para definir una calle) y MultiPoligono(puede utilizarse para definir un lugar).

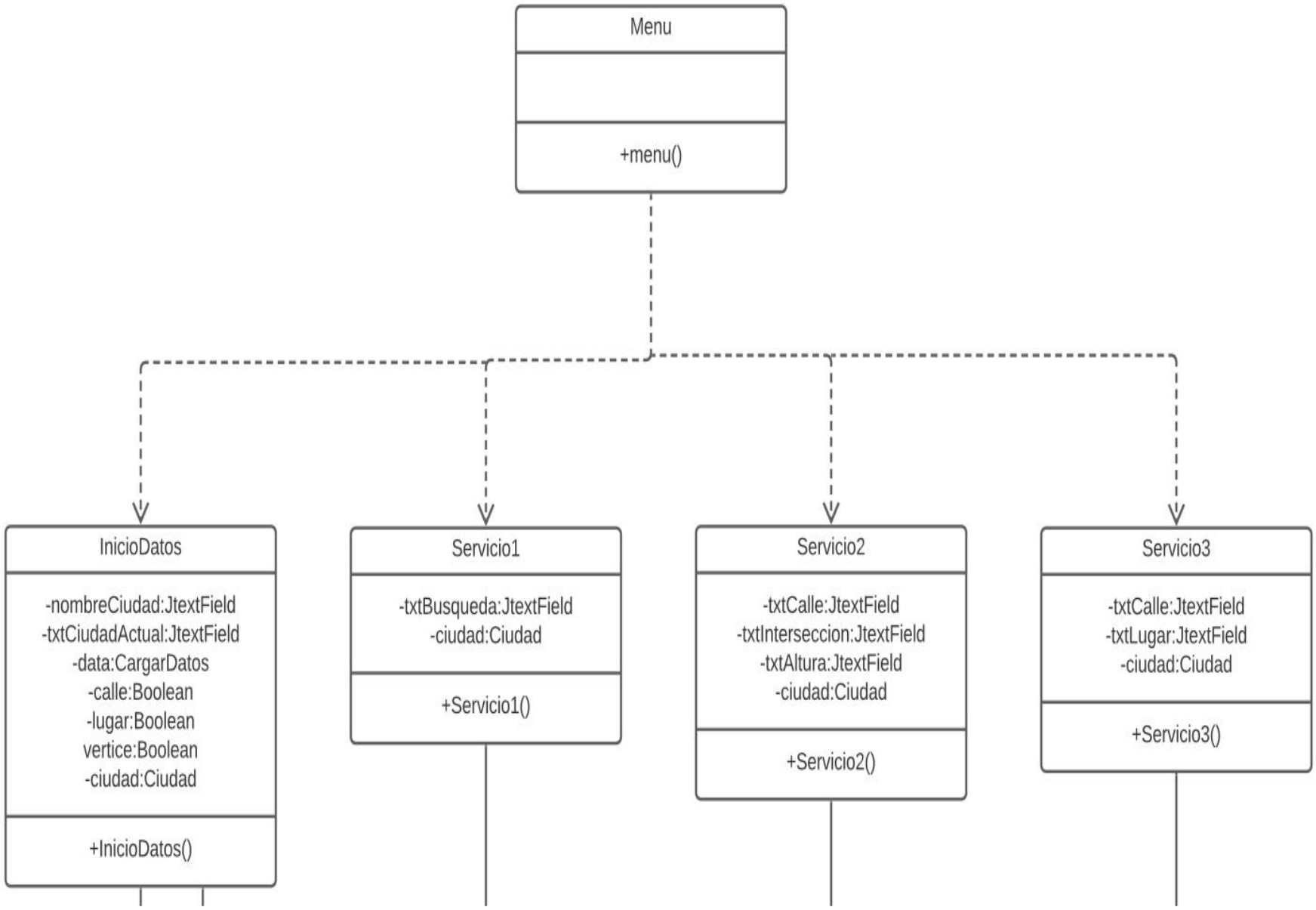

FormaAbs
+getVertices():Vector<Vertice>
+getVerticesConAltura(altura:Int):Vector<Vertice>
+getVerticesIntersecciones(vertices:Vector<Vertice>):Vector<Vertice>
+getPtoMasCercano(f:FormaAbs):VerticeDistancia
+getPtoMasCercano (p:Punto):VerticeDistancia
+getPtoMasCercano (l:Linea):VerticeDistancia
+getPtoMasCercano (p:Poligono):VerticeDistancia
+getPtoMasCercano (fc:FormaCompuesta):VerticeDistancia
+equals (f: Object) Boolean
Linea
-vertices:Vector<Vertice>
+addVertice (v:Vertice):Void
+getVertices():Vector<Vertice>
+getVerticesConAltura(altura:Int):Vector<Vertice>
+getVerticesIntersecciones(vertices:Vector<Vertice>):Vector<Vertice>
+getPtoMasCercano (f:FormaAbs):VerticeDistancia
+getPtoMasCercano (p:Punto):VerticeDistancia
+getInterseccion(a:Vertice,b:Vertice,c:Vertice,d:Vertice):VerticeDistancia
+getPtoMasCercano (l:Linea):VerticeDistancia
+getPtoMasCercano (p:Poligono):VerticeDistancia
+getPtoMasCercano (fc:FormaCompuesta):VerticeDistancia
Punto
-vertices:Vertice
+setVertice (v:Vertice):Void
+getVertices():Vector<Vertice>
+getVerticesConAltura(altura:Int):Vector<Vertice>
+getVerticesIntersecciones(vertices:Vector<Vertice>):Vector<Vertice>
+getPtoMasCercano (f:FormaAbs):VerticeDistancia
+getPtoMasCercano (p:Punto):VerticeDistancia
+getPtoMasCercano (l:Linea):VerticeDistancia
+getPtoMasCercano (p:Poligono):VerticeDistancia
+getPtoMasCercano (fc:FormaCompuesta):VerticeDistancia
Poligono
-linea:Linea
+getLinea():Linea
+getVertices():Vector<Vertice>
+getVerticesConAltura(altura:Int):Vector<Vertice>
+getVerticesIntersecciones(vertices:Vector<Vertice>):Vector<Vertice>
+getPtoMasCercano (f:FormaAbs):VerticeDistancia
+getPtoMasCercano (p:Punto):VerticeDistancia
+getPtoMasCercano (l:Linea):VerticeDistancia
+getPtoMasCercano (p:Poligono):VerticeDistancia
+getPtoMasCercano (fc:FormaCompuesta):VerticeDistancia
FormaCompuesta
-forma:FormaAbs
+addForma(f:FormaAbs):Void
+getFormas():Vector<FormaAbs>
+getVertices():Vector<Vertice>
+getVerticesConAltura(altura:Int):Vector<Vertice>
+getVerticesIntersecciones(vertices:Vector<Vertice>):Vector<Vertice>
+getPtoMasCercano (f:FormaAbs):VerticeDistancia
+getPtoMasCercano (p:Punto):VerticeDistancia
+getPtoMasCercano (l:Linea):VerticeDistancia
+getPtoMasCercano (p:Poligono):VerticeDistancia
+getPtoMasCercano (fc:FormaCompuesta):VerticeDistancia

Interfaz Gráfica

Para realizar la GUI (Graphic User Interface) se implementó una clase “Menu”, esta se encarga de hacer la llamada de los 3 servicios solicitados para este trabajo ya mencionados anteriormente. Para esto se implementaron 4 clases:

- ***InicioDatos:*** Tiene la funcionalidad para crear la ciudad entera, en esta clase podemos cargar los archivos descargados de OverPass Turbo.Esta clase es importante, ya que si previamente no se carga la ciudad, no se podrá acceder a la utilización de los servicios de la aplicación.
- ***Servicio1:*** Esta clase tiene la funcionalidad de realizar el servicio 1, nos permite elegir buscar una calle o un lugar (no ambos a la vez).
- ***Servicio2:*** Esta clase implementa la funcionalidad del servicio 2, nos permite elegir entre: buscar una intersección entre dos calles o buscar una calle con altura (como en el servicio 1, no se permite realizar ambas búsquedas al mismo tiempo)
- ***Servicio3:*** La clase implementa el servicio 3, nos permite buscar el punto más cercano entre una calle y un lugar solicitado. Entregándonos el vértice resultante (lat, long) y la distancia a la que se encuentra del lugar.

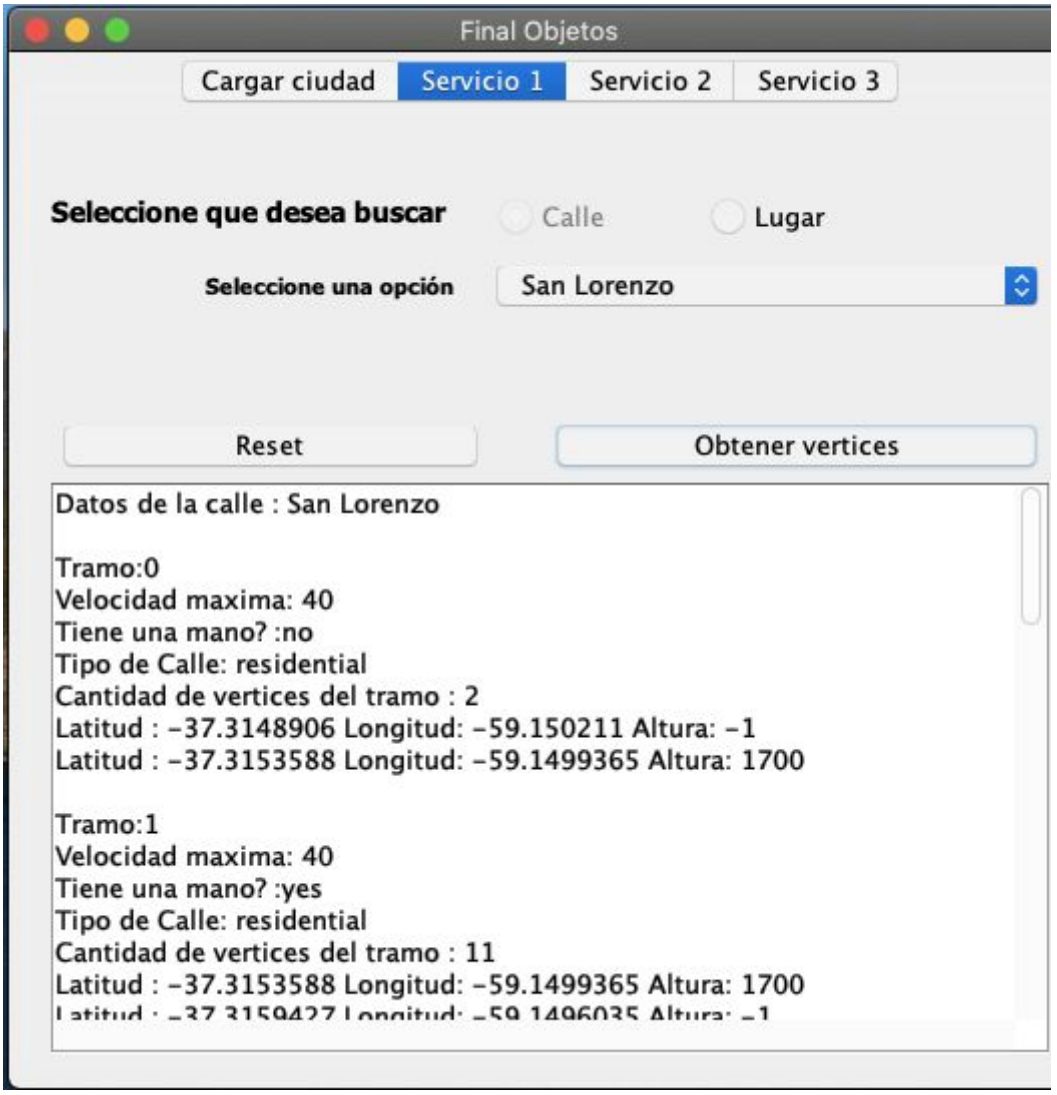
Para poder implementar todo esto, se investigó el uso de la biblioteca “SWING”, la cual sirve para realizar interfaces gráficas en Java. La biblioteca se puede usar de forma gráfica (arrastrando los elementos a los layout), como así también escribiendo código directamente en la clase. Una vez colocados los elementos, se debe implementar la funcionalidad de los eventos (como por ejemplo hacer click en un botón). En esta parte, SWING no tiene un autocompletado para la funcionalidad, por lo cual es importante ser prolijos, ponerle identificadores a los elementos para poder manipularlos correctamente y darles la funcionalidad deseada.


Menu
+menu()
InicioDatos
-nombreCiudad:JtextField
-txtCiudadActual:JtextField
-data:CargarDatos
-calle:Boolean
-lugar:Boolean
vertice:Boolean
-ciudad:Ciudad
+InicioDatos()
Servicio1
-txtBusqueda:JtextField
-ciudad:Ciudad
+Servicio1()
Servicio2
-txtCalle:JtextField
-txtInterseccion:JtextField
-txtAltura:JtextField
-ciudad:Ciudad
+Servicio2()
Servicio3
-txtCalle:JtextField
-txtLugar:JtextField
-ciudad:Ciudad
+Servicio3()

Implementación de los servicios solicitados

- ***Buscar una calle o lugar por el nombre:***

Para resolver este servicio se implementaron los métodos “getCalle” y “getLugar” en la clase “Ciudad”, a los cuales se les pasa un nombre por parámetro y se devuelve el objeto correspondiente. Esto es, se recorre el conjunto de calles o de lugares respectivamente, y se devuelve el objeto en que el nombre coincida. Si el objeto es una calle, se muestran los tramos que la componen, los vértices que componen la forma de cada tramo y las características propias de cada uno. En cambio, si el objeto es un lugar, se muestran sus características y los vértices que componen su forma. Por ejemplo, si se desea buscar la calle “San Lorenzo” en la ciudad de Tandil:



Cabe destacar que estos métodos, además de utilizarse para la implementación del servicio solicitado, también son utilizados en la clase “CargaDatos” para saber si una calle o un lugar ya fueron cargados en la ciudad correspondiente.

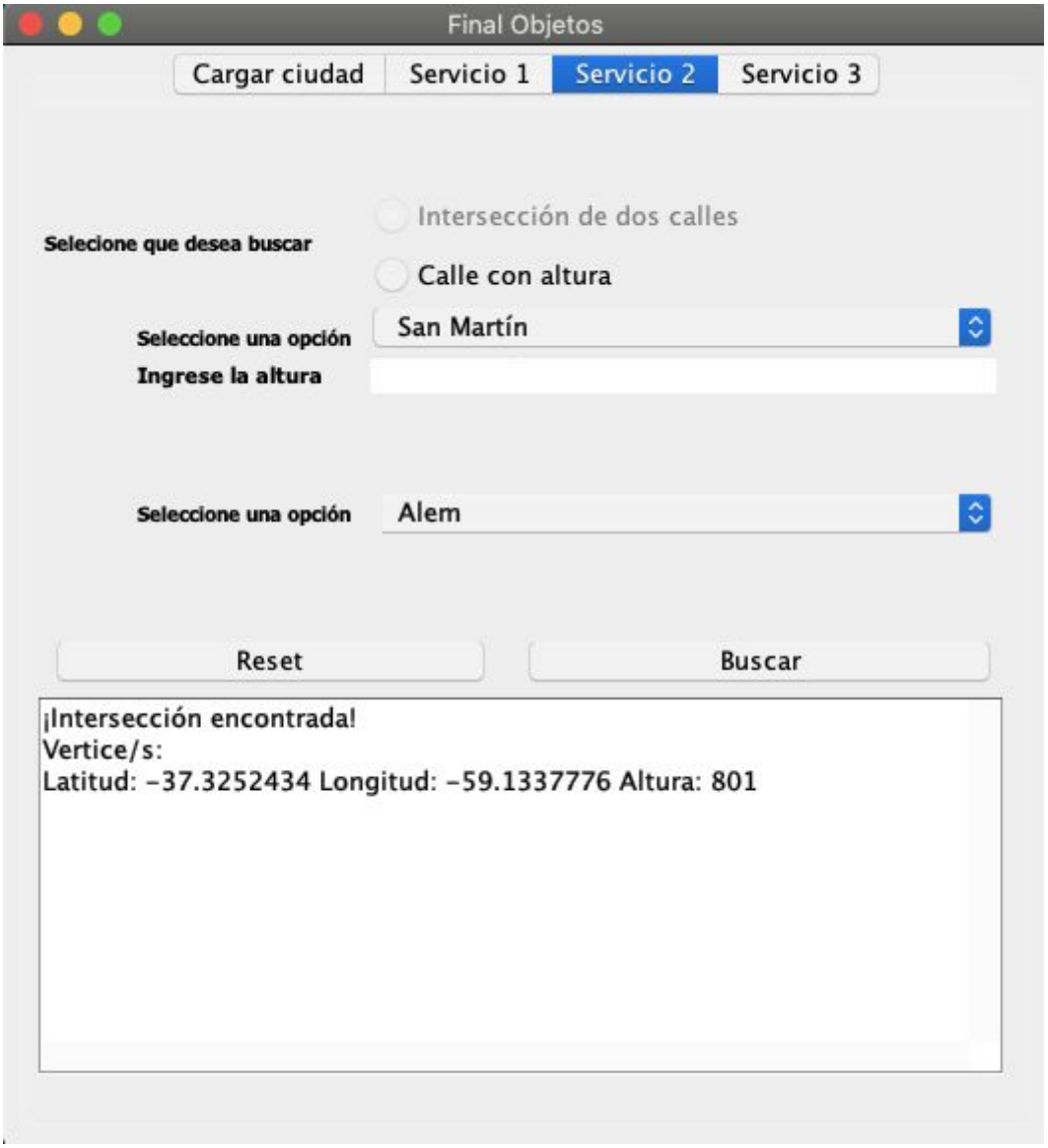
- ***Buscar la ubicación de una calle con altura:***

Para la resolución de este servicio, se ven involucrados los métodos “getUbicacionesCalleAltura” y “getVerticesConAltura” definidos en las clases “Calle” y “FormaAbs” respectivamente. De manera que, se recorren los tramos de la calle seleccionada pasándole una altura(entero) por parámetro y por cada uno de estos, se busca dentro de su forma si posee vértices que coincidan con esta altura. Si alguno de estos vértices ya fue agregado al conjunto solución, no se agrega nuevamente. De esta manera, se evita mostrar vértices repetidos. Por ejemplo, si se desea buscar la altura 702 de la calle Arana:

Debido a que la asociación de las alturas de las calles se realizó de manera manual, resulta difícil encontrar vértices con la altura deseada por el usuario aleatoriamente. Esto podría mejorarse si desde overpass-turbo, se pudieran recuperar más vértices que posean el atributo de altura. Y así, la asociación sea más exacta.

- ***Buscar la ubicación de la intersección entre dos calles:***

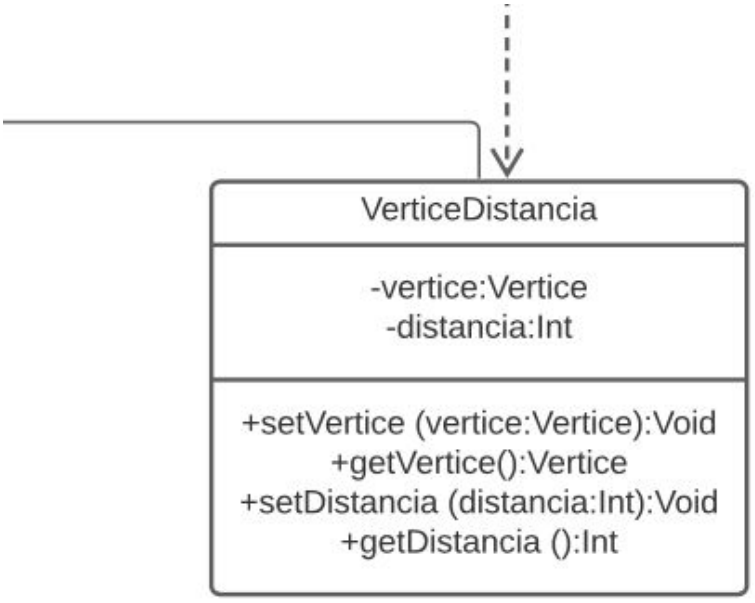
De manera similar al servicio anterior, los métodos involucrados en este servicio son “getUbicacionesIntersecciones” y “getVerticesIntersecciones” definidos en las clases “Clase” y “FormaAbs” respectivamente. Primero se obtienen los vértices que componen la forma de la calle que es pasada por parámetro mediante el método “getVertices”. Luego se recorren los tramos de la calle restante y por cada uno de estos, se busca dentro de su forma los vértices obtenidos de la primer calle. De esta manera, se agregan a la solución los vértices que coinciden, siempre y cuando no hayan sido agregados aun. Por ejemplo, si se desea buscar la intersección entre las calles San Martín y Alem:



Se puede dar el caso en que dos calles pueden intersecarse más de una vez como es el caso de las calles Machado y Moreno, por ejemplo. El programa también cubre estos casos.

- ***Buscar el punto más cercano entre una calle y un lugar:***

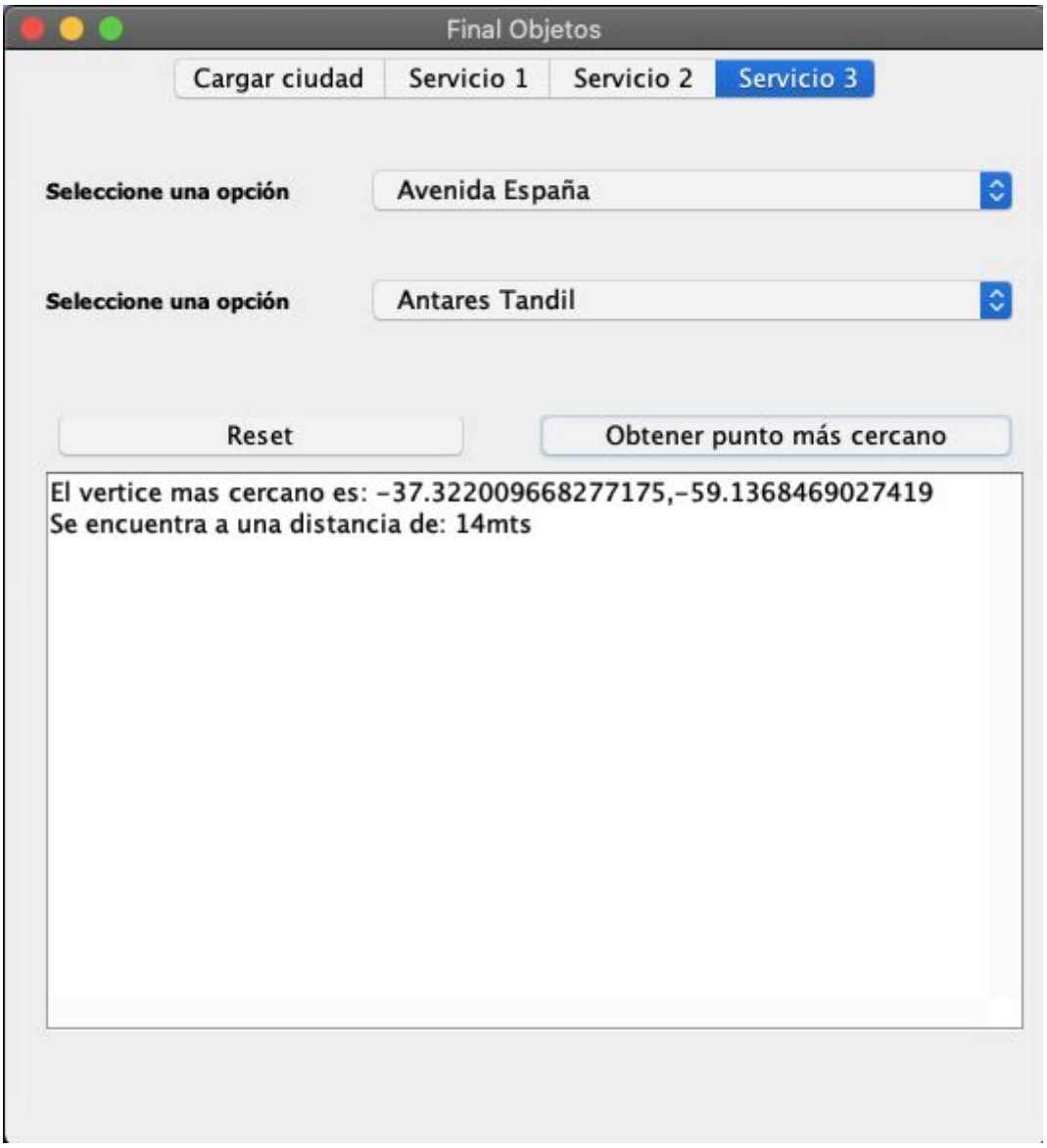
Para la implementación de este servicio se creó una nueva clase “VerticeDistancia” que permite devolver el vértice más cercano buscado junto con la distancia en un mismo objeto.



También se utilizaron los métodos “getPtoDistMasCercano” y “getPtoMasCercano” definidos en las clases “Calle” y “FormaAbs” respectivamente. Cada clase que implementa la clase abstracta “FormaAbs” tiene definido el metodo nombrado anteriormente, “getPtoMasCercano” con distintos parametros, estos pueden ser: “FormaAbs”, ”Punto”, “Linea”, “Poligono” o “FormaCompuesta”. Esto se debe, a que el cálculo de las distancias depende de la forma que llegue por parámetro, es decir, no se emplea la misma función para calcular la distancia punto/punto, punto/linea, linea/linea, etc.

- Punto/Punto: simplemente se utiliza la función “getDistanciaPuntoPunto” especificada anteriormente.
- Punto/Línea o Línea/Punto: se van tomando dos vértices de la línea y se calcula la distancia de esta nueva línea geodésica con el punto. Se termina retornando el vértice que tenga la menor distancia.
- Línea/Polígono o Polígono/Línea: en este caso, lo que en realidad se está calculando es la distancia entre dos líneas, debido a que un polígono tiene una línea que representa su perímetro. Este cálculo es más complejo, ya que inicialmente hay que verificar si se intersectan en algún punto, en este caso, la distancia es 0. En el caso que esto no suceda, se van tomando dos vértices de la línea de la calle y dos vértices de la línea del lugar. De esta manera, se van calculando las distancias entre la nueva línea geodésica de la calle contra los dos vértices del lugar. Una vez que se hayan recorrido todos los vértices del lugar como de la calle, se retorna el vértice que tenga la menor distancia.

Por ejemplo, si se desea buscar el punto más cercano entre la calle Av España(Linea) y el lugar Antares Tandil(Punto):



En nuestro caso de estudio, el cálculo del punto más cercano entre dos formas siempre va a ser: Linea o MultiLinea(FormaCompuesta) representando las calles contra Punto, Poligono, MultiPunto(FormaCompuesta) o MultiPoligono(FormaCompuesta) representando los lugares. Esto es debido a como están definidos los objetos geojson. Independientemente de esto, en el caso que se desee calcular el punto más cercano entre dos formas aleatorias, el programa está en condiciones de realizarlo eficientemente.

CONCLUSIONES:

En el trabajo final de la materia pudimos aplicar el paradigma de la programación orientada a objetos. Como así también nos adentramos a realizar una interfaz gráfica en Eclipse, la cual antes nunca habíamos podido realizar. Esto realmente fue interesante, ya que en muchas materias de programación, la parte del GUI (Graphic User Interface) no se llega a ver e implementar.

También se investigó mucho sobre herramientas interesantes como OSM. Se utilizó archivos de extensión como GeoJson, lo cual tuvimos que aprender a manipular su estructura e investigar sobre librerías para la cual facilitar su uso.

En conclusión es un trabajo lleno de investigación y que nos permitió aplicar y aprender muchas cosas que se ven en la materia. Reforzamos muchos conceptos de la programación orientada a objetos y nos llevamos el aprendizaje de la GUI, aprender dónde buscar información útil cuando se trata de funcionalidades específicas (Como la búsqueda en Stack OverFlow) y el uso de GitHub. Esto último lo aplicamos exclusivamente para el aprendizaje personal. El desarrollo de este trabajo nos sirvió para poder también aprender a usar Git (aunque este no era un requisito excluyente). Por todo lo mencionado, quedamos muy conformes con el desarrollo del trabajo y el aprendizaje personal de todo lo que investigamos y desarrollamos en equipo.