1. Supuestos

Para la realización del código se supuso que:

- Las coordenadas de entrada vienen en formato sexagesimales, para posteriormente ser transformadas a radianes
- Las coordenadas no pueden tener cualquier valor, sus valores deben encontrarse dentro de un punto físico real.
- Cuando se evalúa el costo del paquete esta también agrega los costos adicionales de asistencia al viajero y/o comida especial, para finalmente aplicar los descuentos a el total.
- No hay restricciones para los Strings ingresados ya sean nombres, código de ciudad, etc.
- Un viaje que contiene un solo vuelo cuenta como un día de viaje.
- En el caso de que haya asistencia al viajero y comida especial a bordo, cada uno de los costos agregados es independiente del otro.
- La fecha de entrada debe ser anterior a la de salida.
- No se debe modificar la clase AlgoTripTest (manejo de excepciones).
- Los paquetes no deben contener otros paquetes.

2. Modelo de dominio

Se desarrollaron las siguientes clases para la solución del programa:

- **AlgoTrip.java** : Es la clase principal y la que se encarga de manejar todo. Tiene como composición colecciones de las clases Vuelo, Ciudades, Hoteles y Paquetes (para los paquetes sueltos). Es la responsable de llamar a las clases pertinentes para usar las funciones más importantes del programa como: obtenerCosto(), agregarVueloEnViaje(), obtenerDuracionEnDiasDelViaje(), etc.
- **Viaje.java** : Es la clase más extensa seguida de AlgoTrip, estando compuesta por colecciones de vuelos, estadias, adicionales y paquetes. Sus responsabilidades principales son: cacular el costo del viaje, calcular la duración del viaje y agregar vuelos y estadias. Todo esto lo lleva acabo delegando algunas responsabilidades, con ayuda de otras clases.
- **Ciudad.java** : Esta clase contiene toda la información pertinente a una ciudad como el nombre de la misma, su código, el nombre de su país y además la ubicación con los valores de latitud y longitud. Su responsabilidad principal es calcular la distancia entre dos ciudades dadas con la ayuda de la fórmula de Haversine y calcular el tipo de tarifa de un vuelo(nacional o internacional).
- **Hotel.java**: Esta clase simplemente es un contenedor de un nombre de hotel asociado a un precio por noche. Su única responsabilidad es guardar y proporcionar esos datos.
- **Vuelo.java**: Solo contiene la referencia de la ciudad de partida y la referencia de la ciudad de destino. Su única responsabilidad es relegarle a la clase Ciudad el cálculo del costo del vuelo.
- **Estadia.java**: Contiene una referencia a un hotel y la fecha de entrada y salida al mismo. Sus responsabilidad responsabilidades son calcular el costo de la estadía y calcular la duración de la misma en días.
- **Adicionales.java**: Es una clase abstracta con dos declaraciones del método modificarCosto() (sobrecarga) una para la asistencia al viajero y la otra para la comida especial.
- **AsistenciaAlViajero.java**: Hereda de la clase Adicional debiendo implementar los métodos de modificarPrecio() y su responsabilidad es modificar el costo de un vuelo y/o una estadía.
- **ComidaEspecialABordo.java**: Al igual que asistencia al viajero hereda de Adicional, implementa ambas funciones sobrecargadas y su responsabilidad es modificar el costo de un vuelo y/o estadía.
- **Paquete.java**: Esta clase hereda de la clase Viaje, ya que comparté muchos métodos similares y comportamiento (menos la colección de paquetes como atributo). Su responsabilidad es obtener el costo de los vuelos y estadias que

se adhieran al paquete, sobreescribiendo el método obtenerCostoDelViaje() para hacerlo de manera distinta a la clase Viaje.

Sumado a esto, se creo al principio una clase de test para cada clase individual para ir modelando y corrigiendo el correcto funcionamiento de cada clase.

También en el modelo se incluyo un paquete con varias clases de excepciones para manejar todos los imprevistos necesarios y que se consideraron relevantes.

3. Diagrama de clases

En el siguiente diagrama de clases se muestra la interacción entre todas las clases del proyecto. En el mismo se obviaron tanto las variables como los métodos irrelevantes para una mayor comprensión del mismo.

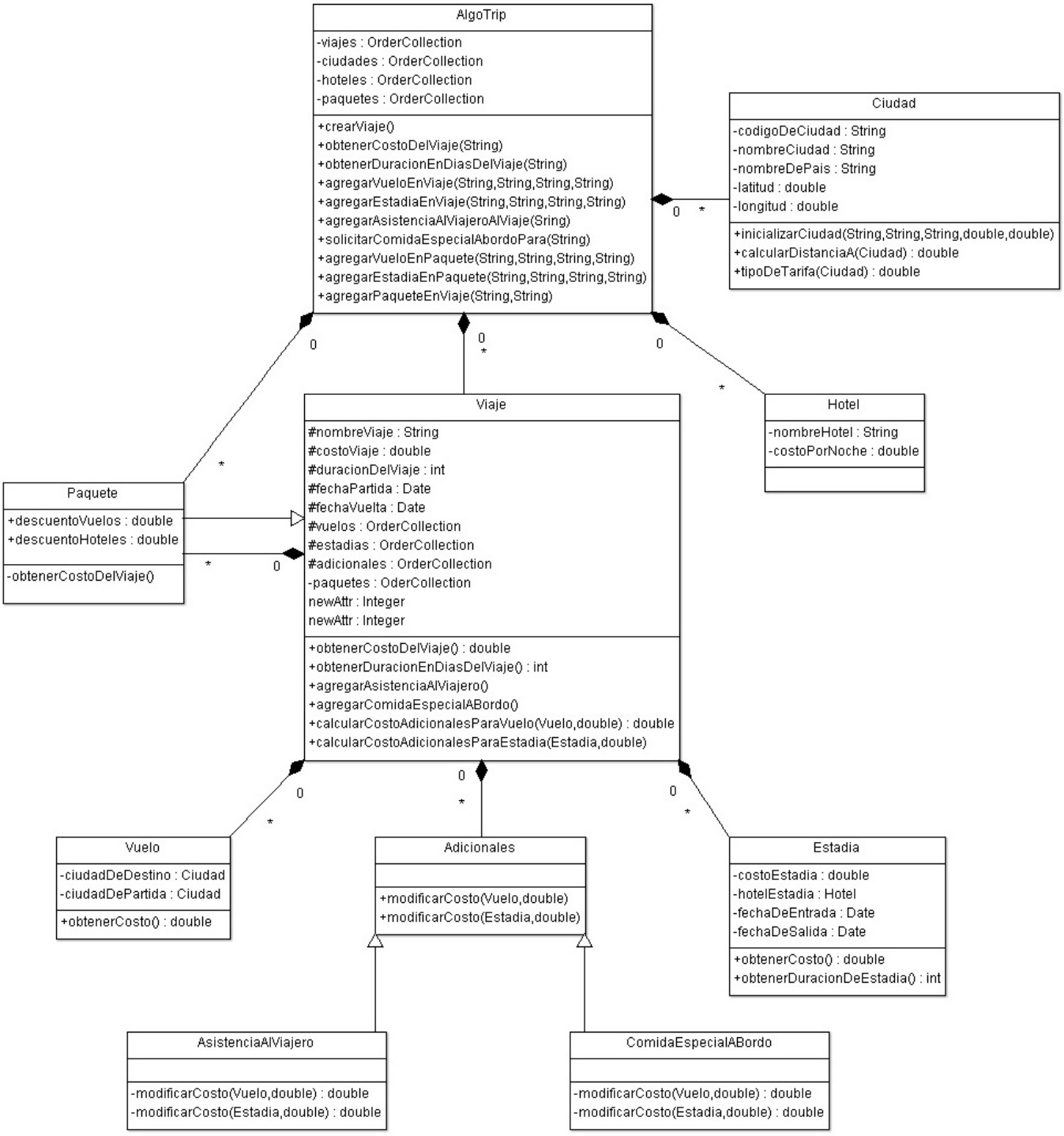

Figura 1: Diagrama de clases

4. Detalles de implementacion

Para la realización de este proyecto se tuvieron en cuenta distintas opciones pero siempre en dirección a una solución orientada a objetos en mayor medida posible.

La clase **AlgoTrip** (la principal) se implemento con cuatro colecciones de las clases Viaje, Hotel, Ciudad y Paquete. Para ir cargando las colecciones a medida que se agrega un objeto nuevo. Esta clase es la que se encarga de verificar los datos de entradas para, en caso de necesitarlo, lanzar una excepción. Verificando así el valor de las coordenadas, la existencia de los objetos requeridos, la 'lógica' de las fechas y el valor de los precios. También se responsabiliza de la conversión de los Strings a Dates cuando se maneja fechas. Por último, para obtener costos, calcular la duración de viajes y agregar adicionales (asistencia al viajero y comida especial) esta clase delega esa responsabilidad a la clase Viaje (o Paquete).

Dentro de la clase **Viaje** se agregaron colecciones para vuelos, estadías, adicionales y paquetes. En un primer momento también se intento englobar a los vuelos y estadías en una sola colección de 'Items de Viaje' que iba a ser una clase abstracta de la que heredaban Vuelo y Estadía. Pero debido a problemas de sintaxis y de tiempo se resolvió en dejarlo en dos colecciones por separado. Esta clase se encarga de calcular la duración del viaje en cuestión con la ayuda de la librería *java.util.date*. Además también agrega cualquier tipo de 'adicional' al viaje, para posteriormente modificar el precio del mismo. Por último, para obtener su precio esta clase itera sobre las colecciones de vuelos, estadías y paquetes, y al mismo tiempo, por cada iteración de estas tres colecciones itera toda la colección de adicionales para ir modificando el precio acorde a lo agregado. En última instancia obtiene los costos parciales de los vuelos, las estadías y los paquetes para sumarlos.

En la clase **Vuelo** se optó por poner solo una referencia de la ciudad de partida y la ciudad de destino del vuelo. De modo que, cuando la clase necesita saber el costo del vuelo, le relega a la clase Ciudad el cálculo de la distancia y averiguar si el vuelo es internacional o doméstico.

La clase **Ciudad** se implementó guardando los nombres de la ciudad, el país y el código de la ciudad. Además contienen los datos de latitud y longitud de la ciudad en cuestión. De este modo, puede calcular la distancia entre una ciudad y otra (con la fórmula de Haversine) y también puede conocer si dos ciudades están en distintos paises

La clase **Hotel** simplemente es un contenedor de datos, con el nombre del hotel como identificador y el precio por noche del mismo.

En la clase **Estadia** podemos encontrar una referencia al hotel de donde pertenece la estadía y una fecha de entrada y otra de salida para el cálculo de la duración en días de la estadía que esta misma clase se encarga. También es la que se encarga de calcular el costo de la estadía pidiendo el precio por noche a el hotel que tiene como referencia.

Adicional es una clase abstracta que se implemento de forma que la clase Viaje pueda iterar y pedir modificación de precios de manera polimórfica en su colección de adicionales. Tiene la función abstracta modificarCosto sobrecargada dos veces para que sea implementada por sus clases hijas.

AsistenciaAlViajero y **ComidaEspecialABordo** son dos clases hijas de la

clase adicional y lo que hacen las dos es devolver el valor adicional al vuelo o estadía. Ambas tienen la función modificarCosto implementada dos veces.

Por último la clase **Paquete** es una clase hija de la clase Viaje, compartiendo todos sus atributos (menos la colección de paquetes como se aclaró en el supuesto) y casi todos sus métodos. Lo único que difiere, es que esta clase tiene sobre escrito el método obtenerCostoDeViaje, comportándose casi de la misma manera pero agregando los descuentos de la clase Paquete.

5. Excepciones

En el trabajo se crearon la siguientes clases de expeciones:

- **CoordenadasInvalidadError.java**: Esta excepción fue creada para que las coordenadas que entran como datos tengan su representación en el mundo físico. Todas las funciones que tienen coordenadas como argumentos se llaman internamente a verficarCoordenadas para chequear si es necesario arrojar la excepción.
- **FechaInvalidadError.java**: Esta excepción fue creada con el motivo de no tener una fecha de entrada o inicio posterior a una fecha de salida. Se usa dentro de las funciones que agregan nuevas estadías, verificando previamente las fechas.
- **InexistenteError.java**: Fue creada para informar sobre la falta de un objeto requerido de cualquier tipo: Viaje, Hotel, Ciudad, etc. Cuando una función pasa como parámetro un nombre de un objeto que no esta incluido en alguna colección de AlgoTrip, entonces esta excepción es arrojada.
- **PrecioNegativoError.java**: Para un caso particular se creo esta excepción: verificar que el precio por noche del hotel no sea negativo. No deja de ser importante ya que no tendría sentido un precio menor a cero. Se utiliza cuando se va agregar un hotel.

6. Diagramas de secuencia

En el siguiente diagrama se muestra como es la secuencia al llamar al método obtenerCostoDelViaje de la clase AlgoTrip.

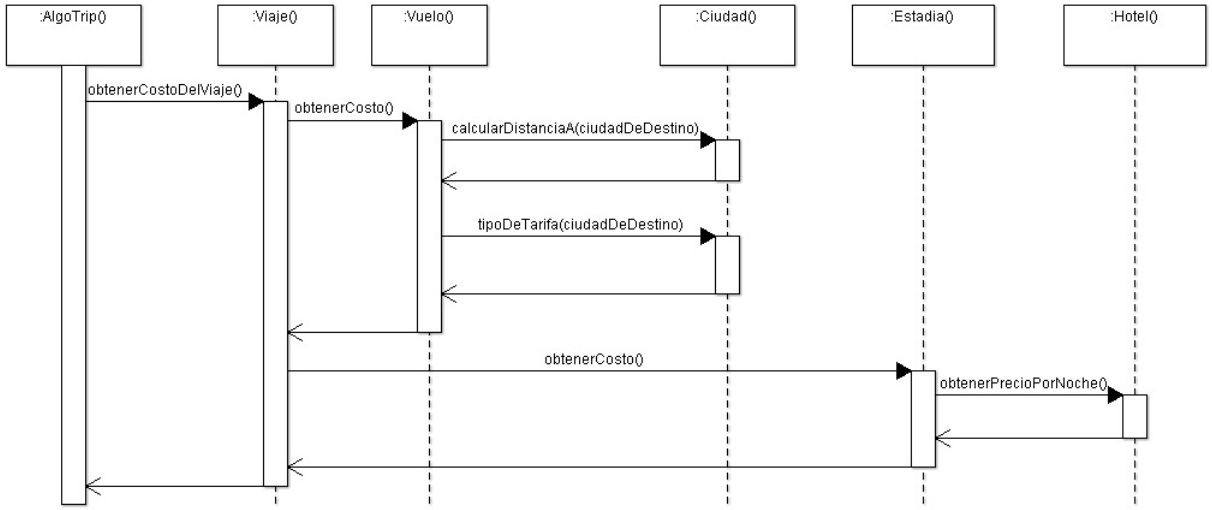

Figura 2: Diagrama de secuencia 01

En el siguiente diagrama (bastante más simple) se puede ver como se agrega un vuelo en la colección que posee la clase Viaje.

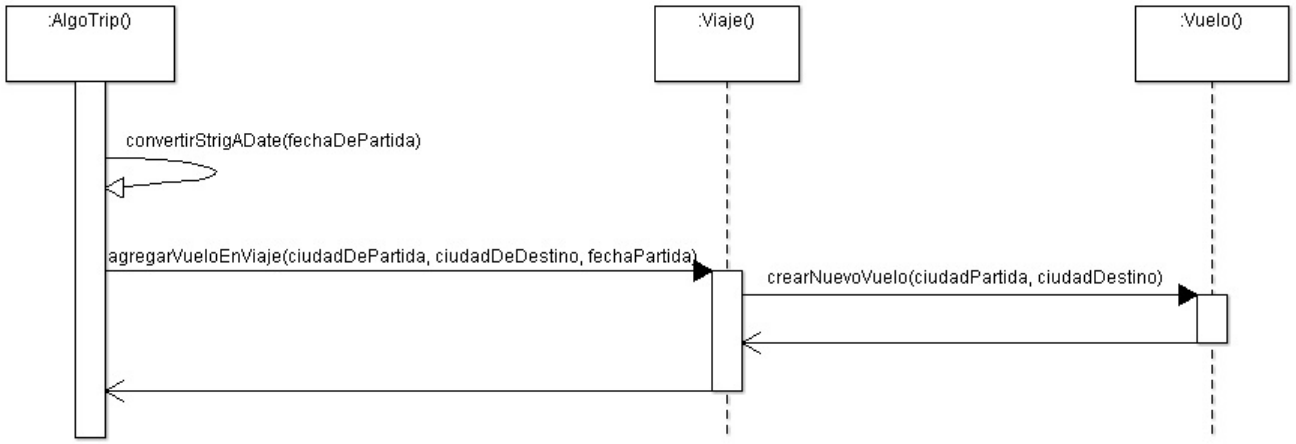

Figura 3: Diagrama de secuencia 02