FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 2
Segundo cuatrimestre 2023

Alumno:	Lopez, Francisco
Número de padrón:	107614
Email:	frlopez@fi.uba.ar

Índice

1. Supuestos

1.1. Descuentos con tarjetas

Se da por supuesto que al momento de calcular los costos(independientemente de la ubicación que sea), es posible realizarlo con cualquier tarjeta y que además se realizara un descuento sin importar la marca de la tarjeta. Por lo tanto al usar la tarjeta "Patagonia","Visa", etc. Se producirá un descuento del 20

1.2. Cantidad de eventos posibles

Se da por supuesto que al realizar el registro de un evento pertenece a un unico artista y que por lo tanto no pueden existir dos eventos con el mismo nombre.

Además de que no podrán existir dos eventos juntos. Por ejemplo, crear un evento nacional y uno internacional al mismo tiempo.

2. Detalles de implementación

Abstracción

Al momento de modelar cada clase se pensó en el comportamiento que debería tener cada una en particular, siendo todas capaces de resolver cada una de sus responsabilidades únicamente (las responsabilidades ajenas a la clase las resolverá otra clase que sí le corresponda). De esta forma, al momento de relacionarse entre clases sólo será relevante qué es lo que hace cada una (y no el cómo).

Encapsulamiento

Acorde al principio de abstracción, en donde cada clase resuelve sus responsabilidades, cada clase dejará ver lo mínimo indispensable. Entonces, cada objeto resuelve los problemas que el mismo sabe y en caso contrario se comunicara con otros objetos sin revelar su implementación. Vemos que, por ejemplo, en Algotek que en lugar de pedirle los datos a los objetos directamente les decimos que ellos resuelvan el problema.

Un ejemplo donde se aplica el encapsulamiento es por ejemplo en Evento en el método costoDeEntrada, Evento no sabe ni tiene que saber cómo calculan los costos de ubicación y artista, tanto artista como ubicación son capaces de calcular su costo sin revelar su implementación.

Polimorfismo

Hay mensajes que son entendidos por más de un objeto que provocarán comportamientos diferentes según cuál de ellos lo reciba, esto significa que en lugar de haber código repetido en una serie de condicionales se utiliza un mensaje entendido por varias clases. Durante la ejecución será la clase correspondiente quién reciba este mensaje y llevará a cabo lo que le corresponda en ese determinado momento.

Un caso en donde se aplica polimorfismo es al momento de utilizar alguno de los métodos de pago (efectivo o tarjeta) donde las clases Efectio y Tarjeta, serian considerada implementaciones de la clase abstracta "MetodosDePago" (en java sería una interfaz). Entonces Efectivo y Tarjeta saben cómo interpretar costoDeEntrada y que deben devolver.

Como smallTalk no permite la implementacion de interfaces, mostramos como quedaria en el diagrama uml. Mostramos dichas clases:

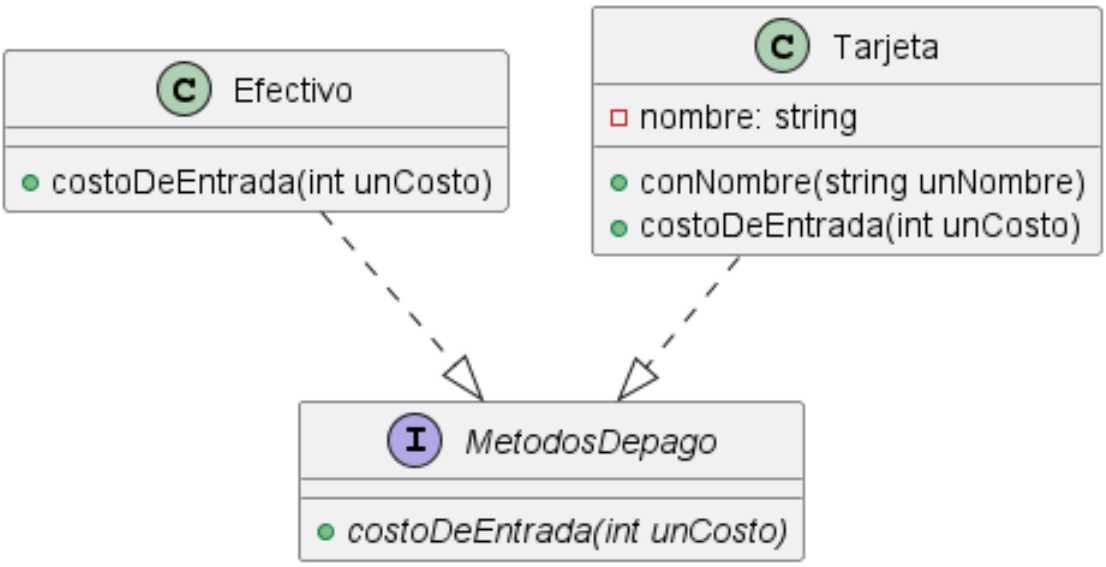

Figura 1: Relacion de los distintos metodos de pago

Observación: Por cuestiones de implementación, observamos que la clase Efectivo podría tender a ser una clase anémica (si no hubiéramos implementado la "interfaz"). En un futuro ante un cambio de requerimiento tanto efectivo como métodos de pago pueden cambiar.

Delegacion

Siguiendo el hilo de los pilares anteriores, cada objeto resuelve sus propias responsabilidades y al momento de encontrarse ante la profundización de un problema que escapa de esas responsabilidades, en lugar de romper con el encapsulamiento de otra clase pidiéndole información para resolverlo, lo que hace es pedir a otra clase (que sepa resolver el problema puntual) que resuelva el problema.

Delegación se aplica en todo el código, ya que en la resolución del tp se fueron delegando tareas a distintos objetos y esos objetos (en algunos casos) delegan en otros.

Sin ir más lejos, Algotek delega a Evento, mientras que Evento sigue delegando tareas a las demás clases construidas (Se espera que cada clase pueda resolver un problema puntual y en caso de no saber resolverlo, poder pedir a otra que si sepa cómo resolver el problema).

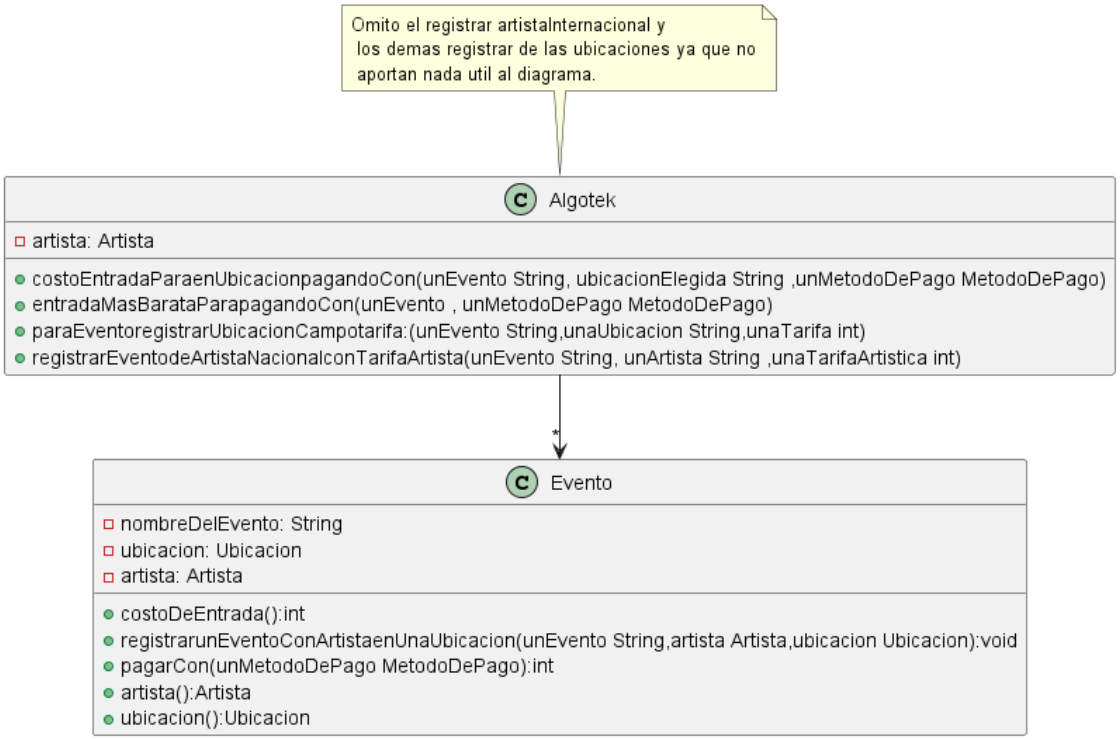

Figura 2: Relacion de Evento y Algotek

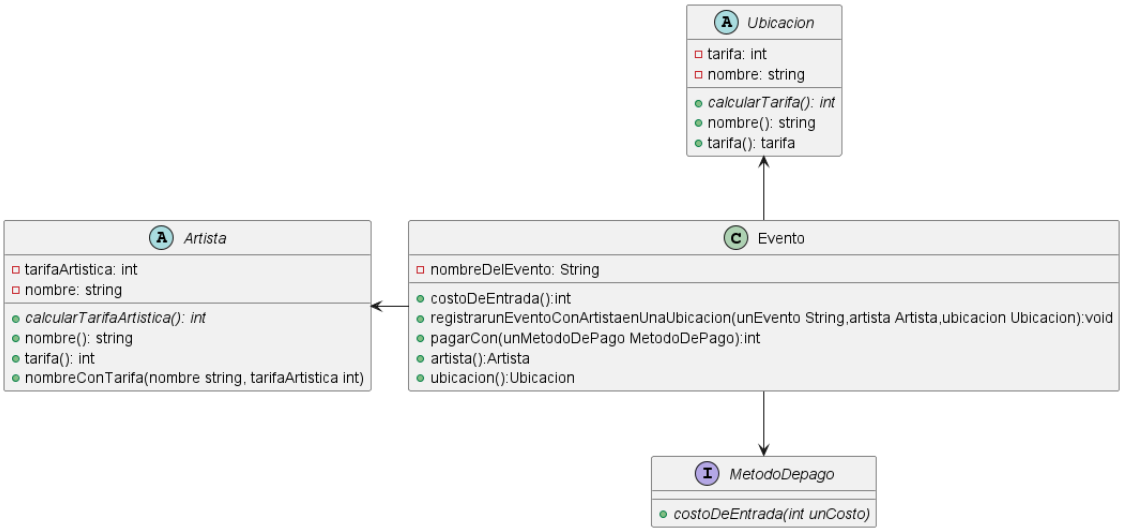

Figura 3: Relacion de Evento con Ubicacion,Artista y MetodoDePago

Herencia

Debido a que es una relación muy fuerte y que Smalltalk es bastante flexible, hay dos casos donde aplicamos herencia:

La clase madre Ubicacion (abstracta) la cual tiene los atributos: tarifa la y nombre que no se inicializan (las inicializan las clases que heredan de ubicación en el class side).

En cuanto a los métodos posee un método abstracto calcularTarifa y dos métodos de clase (no abstractos) los cuales devuelven el nombre y la tarifa.

En cuanto a las subclases tenemos tres: Platea, Vip y Campo las cuales ademas de implementar los métodos de la clase madre (tanto el abstracto como los dos que no lo son) utilizan los atributos heredados. Además tienen implementado un método para setear nombre y tarifa (en caso de la ubicación vip además setea las ubicaciones).

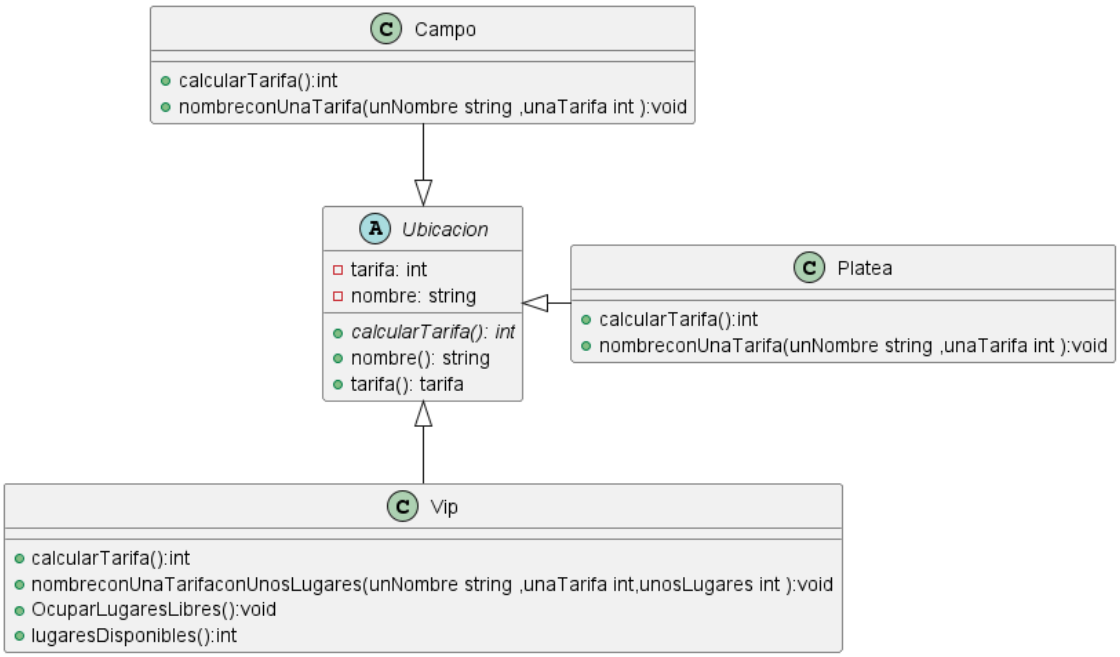

Figura 4: Herencia aplicada en las ubicaciones

El segundo caso se trata de los artistas.

La clase madre es Artista (clase abstracta) la cual tiene un metodo abstracto calcularTarifaArtistica, tres métodos de clase (no abstractos) nombre conTarifa cual setea el nombre con la tarifa, además tenemos nombre (el cual devuelve el nombre) y tarifa (el cual devuelve la tarifa). Como atributos tenemos tarifaArtistica y nombre. Las subclases son ArtistaNacional y ArtistaInternacional, las cuales heredan los atributos de la clase madre y los implementan

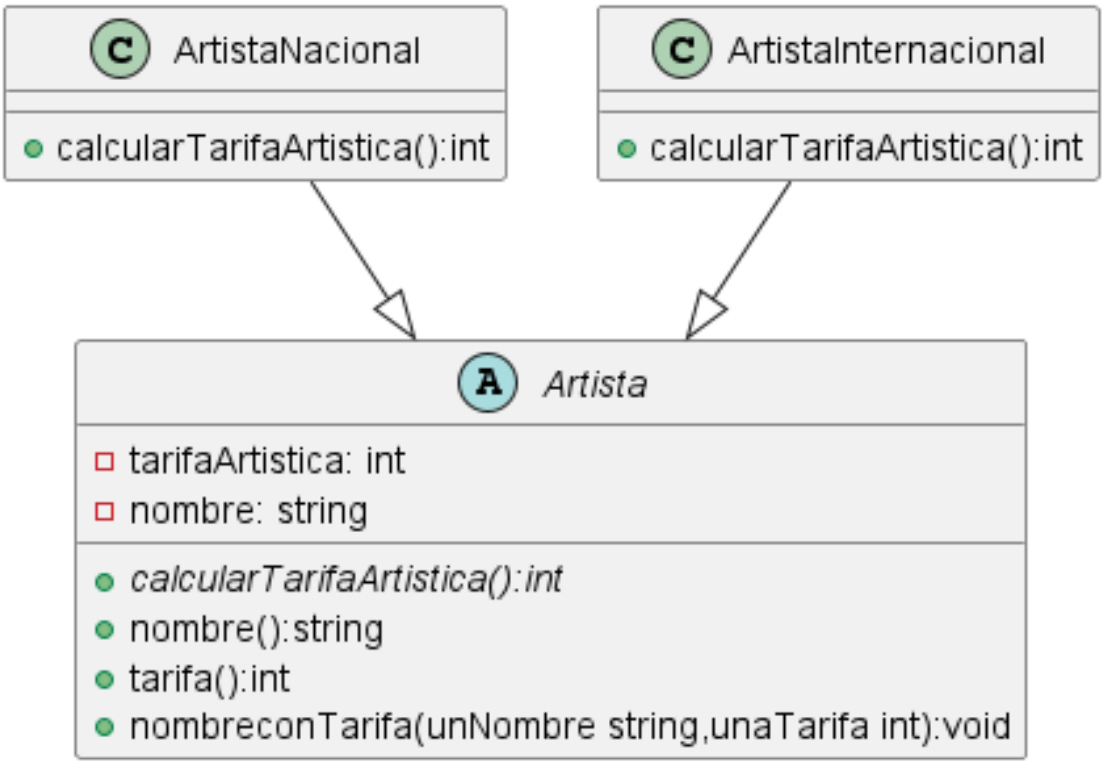

Figura 5: Herencia aplicada en los artistas

3. Excepciones

- **NoHayLugaresDisponiblesError**

  La excepción se lanza cuando se intenta comprar una entrada (se quiere calcular la tarifa de una ubicación vip) y no hay más lugares disponibles.

  Fue creada con el fin de que no se puedan sacar mas entradas cuando no hay mas entradas o se ingresa una cantidad no valida y se quiere calcular su coste al final.

  Mostramos la excepción:

```
OcuparLugaresLibres
    lugaresDisponibles := lugaresDisponibles - 1.

    (lugaresDisponibles < 0 ) ifTrue: [ NohayLugaresDisponiblesError  signal ].
```

Figura 6: Excepcion 1

- **NombreDeTarjetaInvalidoError**

  La excepción se lanza cuando se intenta ingresa un nombre de tarjeta vacío.

  Fue creada con el fin de gestionar el error en caso de que no se ingrese ninguna tarjeta (un caso sería que el usuario se saltee ese paso por accidente). .

  Mostramos la excepción:

```
conNombre:unNombre
   (unNombre = '') ifTrue: [ NombreDeTarjetaInvalidoError  signal ].
   nombre := unNombre .
```

Figura 7: Excepcion 2

- **TarifaInvalidadError**

  La excepción se lanza cuando se ingresa una tarifa invalida (menor que 0), se lanza tanto para la tarifa de un artista, como para la de una ubicación.

  La excepción se creó con el fin de evitar que se ingresen valores incorrectos o que no tengan sentido en nuestro modelo. Con esto también evitamos que se puedan realizar cálculos de costos que no sean validos o no tengan sentido

  Mostramos la excepción:

```
nombre:unNombre conTarifa:unaTarifa
   nombre := unNombre .
   (unaTarifa <0 ) ifTrue: [ TarifaInvalidaError signal ].
   tarifaArtistica := unaTarifa .
```

Figura 8: Excepcion 3

4. Diagramas de secuencia

Para los diagramas de secuencia se decidio tomar los test 1 y 5 de la catedra.

Test1SacarUnaEntradaEnUnaUbicacionParticularTieneElSiguienteCoste

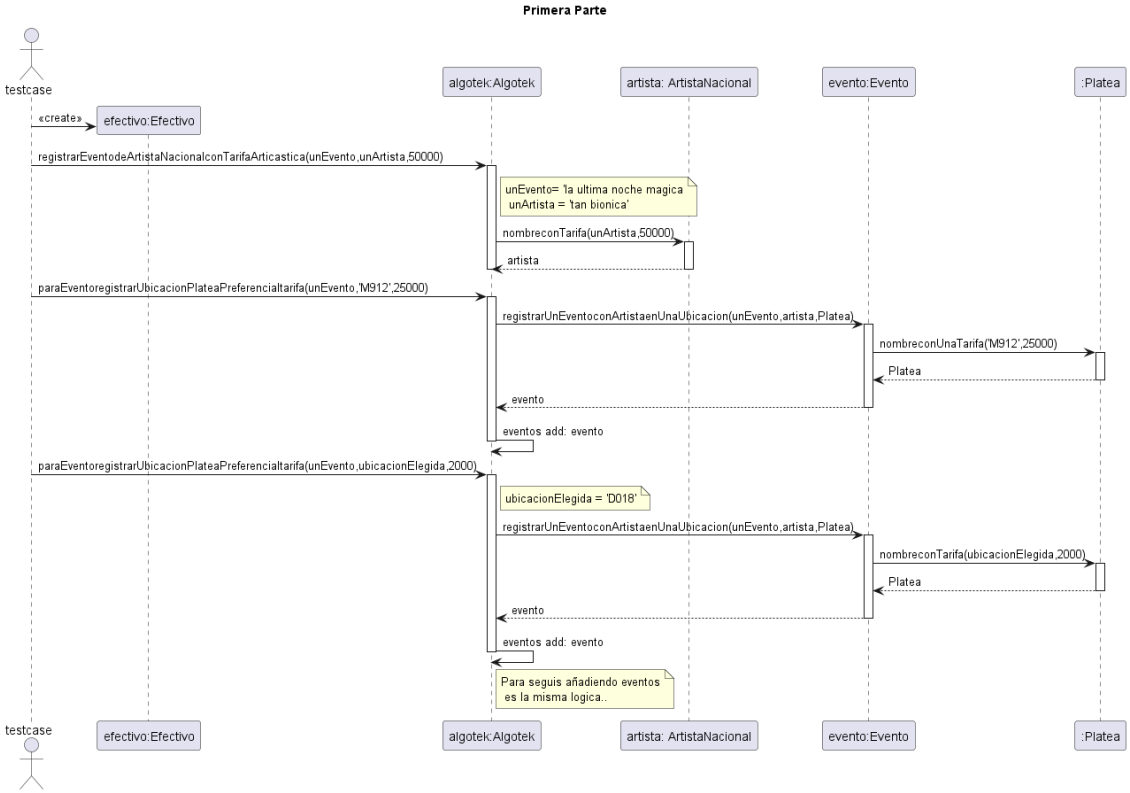

Figura 9: Parte 1

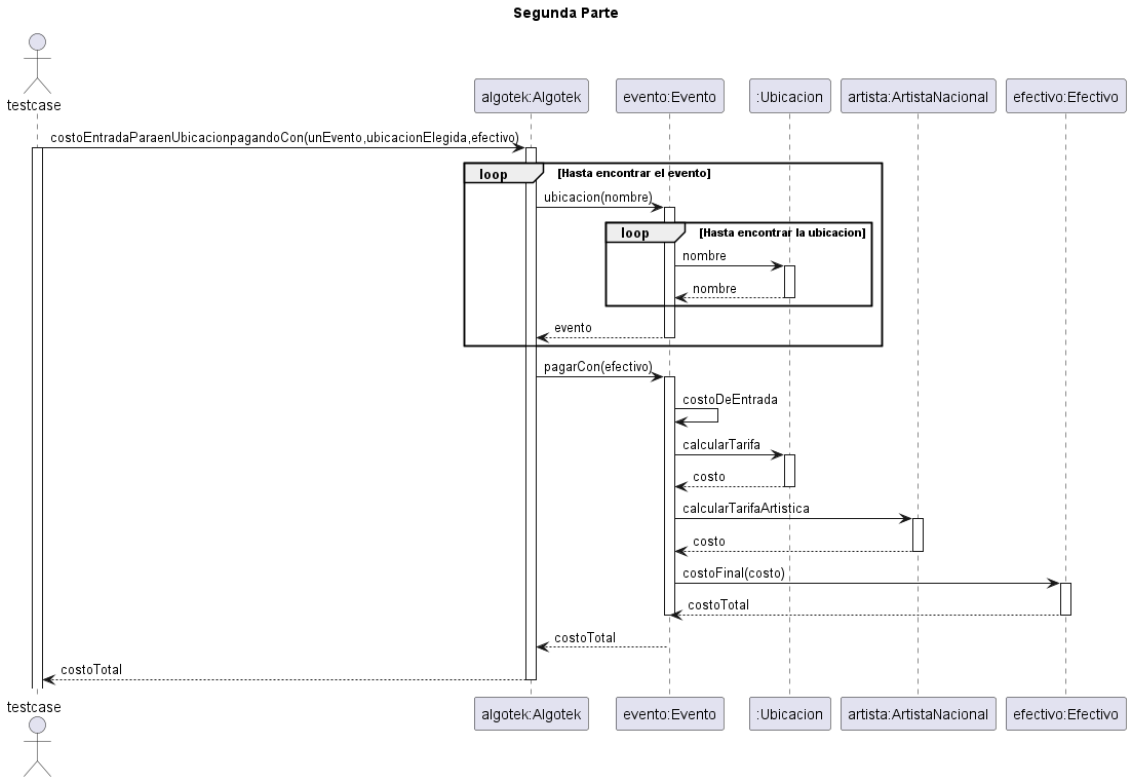

Figura 10: Parte 2