Trabajo Práctico 1 — Smalltalk

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

Alumno:	Natale Nicolas
Número de padrón:	108590
Email:	nnatale@fi.uba.ar

Índice

1. Introducción

La consigna de este TP1 habla del desarrollo de un modelo de clases utilizando la metodología TDD. Haciendo uso de especificaciones dadas en un archivo SUnit (.st) en forma de *pruebas de casos de usos*. Respecto a esta información se espero que el alumno genere una solución que permitiera ejecutar exitosamente las pruebas especificadas manteniendo una alta cobertura y evitando la supervivencia de mutantes.

Toda siguiente información del desarrollo de este programa ara referencia a lo que decidí llamar *"Sistema de Viajes"*

2. Supuestos

Durante el desarrollo del programa se hicieron uso de supuestos (no explícitos en las *pruebas de casos de usos*) necesarios que se dictan debajo.

- Los kilómetros mínimos requeridos se determinaron por la ecuación $(x * 1{,}28) - 1500 > 0$ cuyo resultado es $x > 1{,}171,875$ siendo x los kilómetros de un viaje se redondeo a $x > 1{,}172$. Dada esta información se considero que todo destino menor a $1{,}172km$ debería levantar un tipo de excepción del cual hablaremos mas adelante.
- El criterio con el cual se inicializa el programa siempre debe existir
- Nunca puede existir una persona sin un pasaje guardado.
- La nacionalidad de la persona, junto con su país de nacionalidad y el país de destino siempre deben existir, caso contrario se levanta una excepción.

3. Modelo de dominio

Para la resolución del problema propuesto se desarrollo respecto a la clase *AlgoViajes* el cual deriva funcionalidades a *CriterioMax* y *CriterioMin* dependiendo del tipo de criterio con el cual se haya inicializado, ya sea Máximo o Mínimo. *CriterioMax* y *CriterioMin* guardan un registro de personas con una colección de *Pasajes* quien contiene toda información requerida para determinar nacionalidades y calcular tarifas según el destino (Nacional, Internacional, Mercosur).

4. Diagramas de clase

Debajo se presentan una serie de diagramas mostrando las relaciones estáticas entre las clases. Es necesario aclarar la distinción de colores, gris para *interfaces*, celeste para *clases* y rosa claro para *excepciones*.

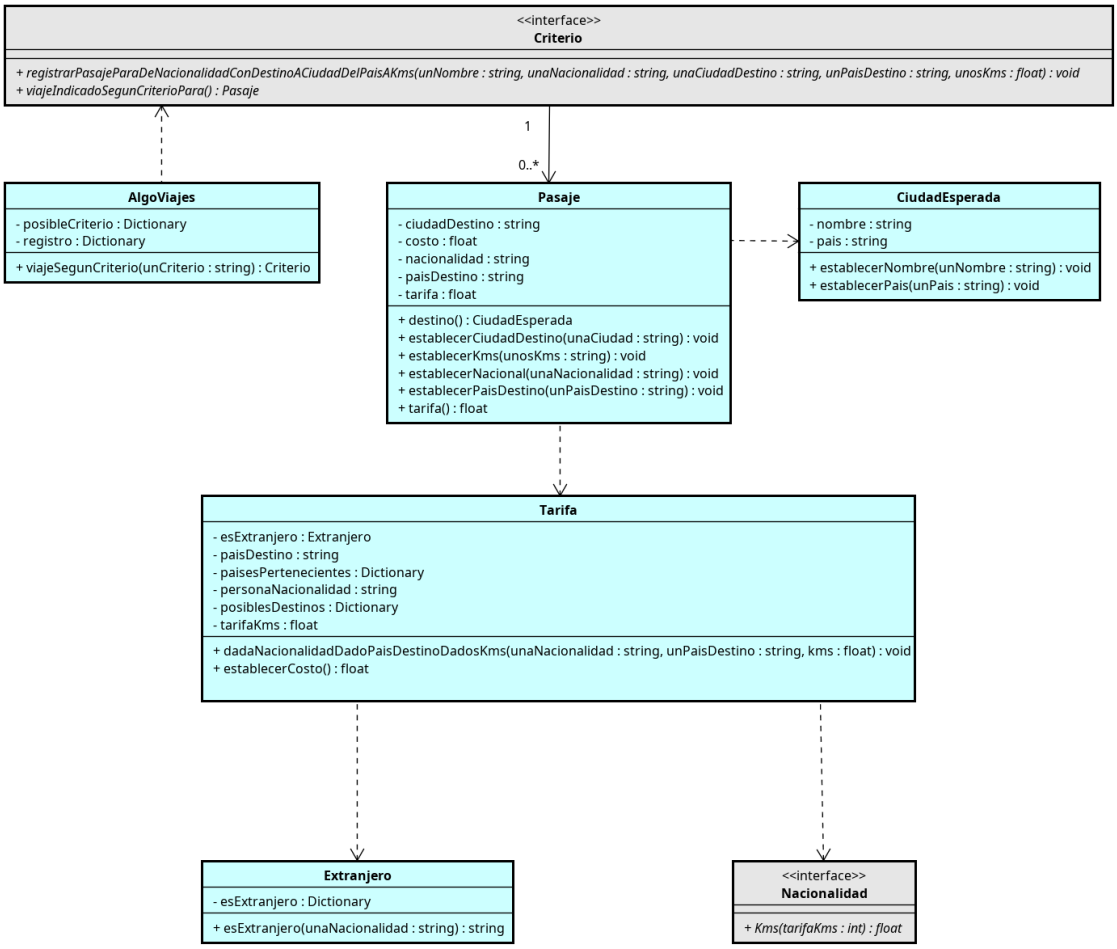

Figura 1: Diagrama de clases principal

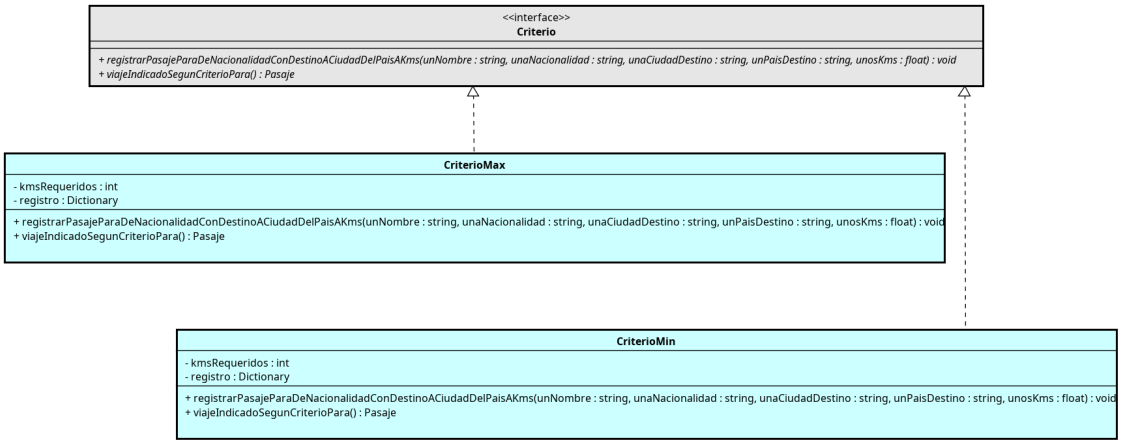

Figura 2: Diagrama de interfaz Criterio

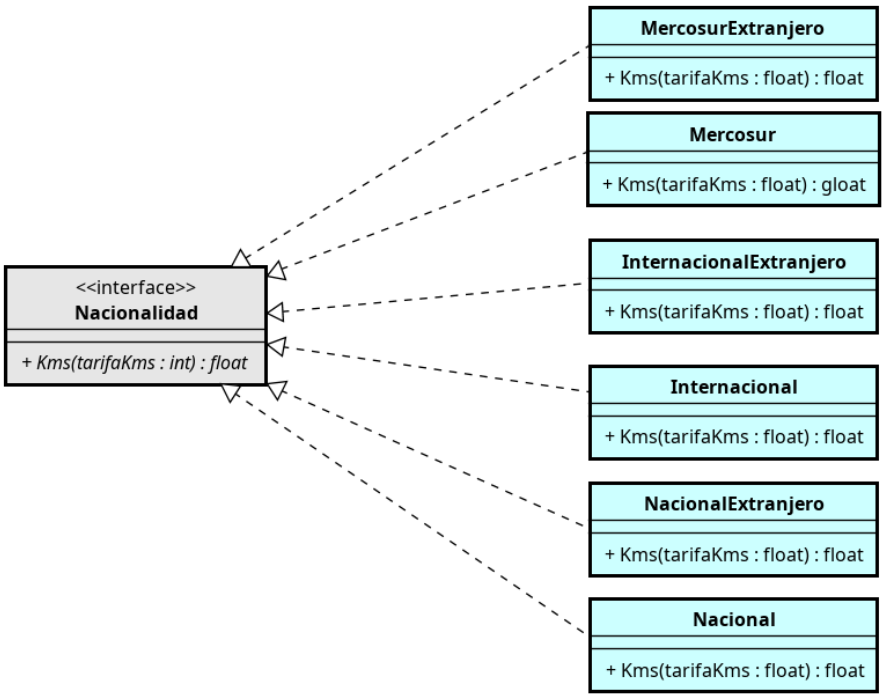

Figura 3: Diagrama de interfaz Nacionalidad

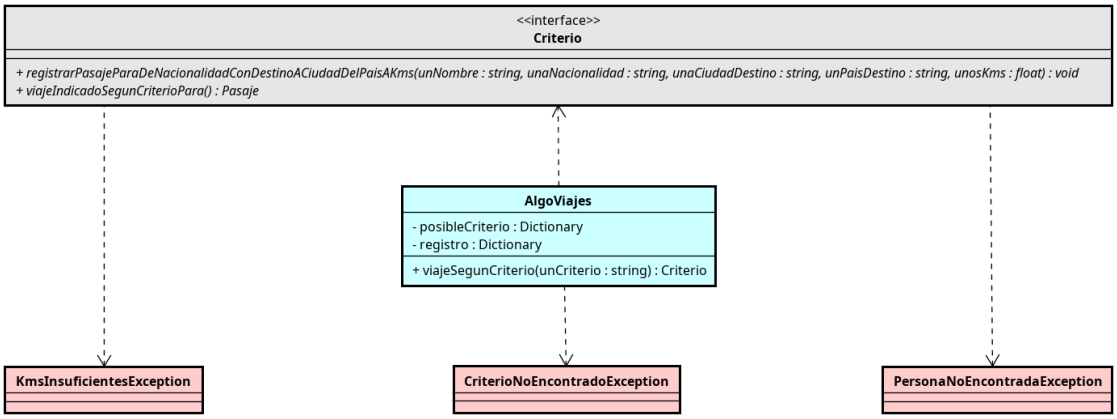

Figura 4: Diagrama de excepciones clases: Criterio y AlgoViajes

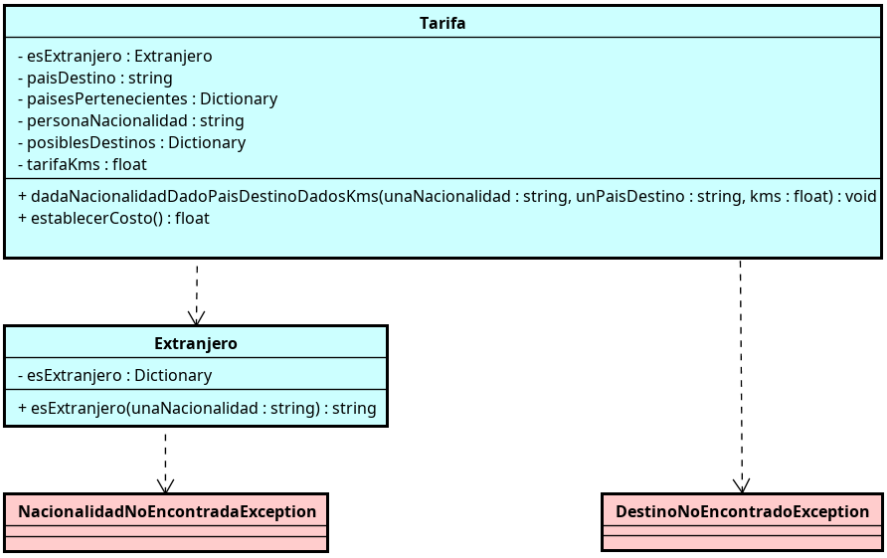

Figura 5: Diagrama de excepciones clases: Tarifa y Extranjero

5. Detalles de implementación

5.1. Clase de control del programa

Inicialmente los test dan a entender que la clase *AlgoViajes* tienen un comportamiento de inicialización del programa e implementando metodologías de la POO hice uso de programación por delegación para no tener una relación muy fuerte entre las clases permitiendo mayor flexibilidad frente a cualquier cambio necesario. Lo que lleva a que la clase *AlgoViajes* solo determine a quien le debe delegar la tarea según el criterio de inicialización.

```
viajeSegunCriterio: unCriterio

[ posibleCriterio at: unCriterio ]
on: KeyNotFound
do: [ CriterioNoEncontradoException signal: 'Criterio no encontrado' ].
^ posibleCriterio at: unCriterio
```

5.2. Tarea delegada

Una de las tareas delegadas mas importantes es luego de determinar un criterio, ya sea 'Máximo' o 'Mínimo', al llegar a la clase CriterioMax (para este ejemplo) quien guarda un 'registro' de personas junto con sus pasajes. Parte del código se muestra debajo sin los chequeos de excepciones.

```
| persona pasaje |

pasaje := Pasaje
          Nacionalidad: unaNacionalidad
          Destino: unaCiudadDestino
          Pais: unPaisDestino
          Kms: unosKms.

(persona := registro at: unNombre) add: pasaje
```

6. Excepciones

CriterioNoEncontradoException Se creo con el propósito de chequear que no se pase un criterio invalido al inicializar el programa.

KmsInsuficientesException Se creo con el propósito de chequear que los kilómetros sean los suficientes para no tener errores en futuros cálculos, dado detalles de implementación donde no se puede elevar un numero $<= 0$ a x.

NacionalidadNoEncontrada Se creo para determinar que la nacionalidad de la persona exista, levantar esta excepción equivaldria a hablar de una persona indocumentada.

PersonaNoEncontradaException Se creo con el propósito de chequear que la persona existe en el registro de personas que adquirieron al menos un pasaje en el programa.

7. Diagramas de secuencia

Para los siguientes diagramas de secuencias se determinaron momentos específicos de ejecución. Para la figura 6, se muestra como el programa se inicializa es un estado base donde no contiene nada que no sea necesario.

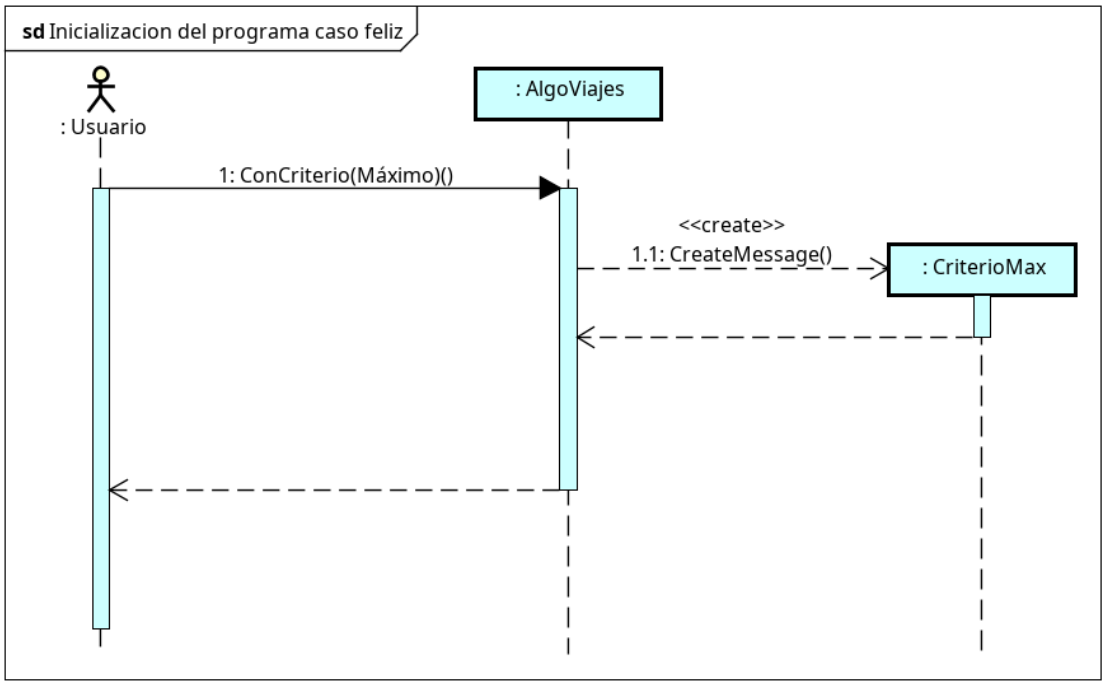

Figura 6: Inicialización del programa en un estado base.

La figura 7 muestra la inicialización del programa, el registro de un usuario y el obtener la tarifa de dicho viaje generado al registrarse la persona. Cabe aclarar que no se agregaron las excepciones por ser muy simples y para no empeorar la legibilidad del diagrama en cuestión.

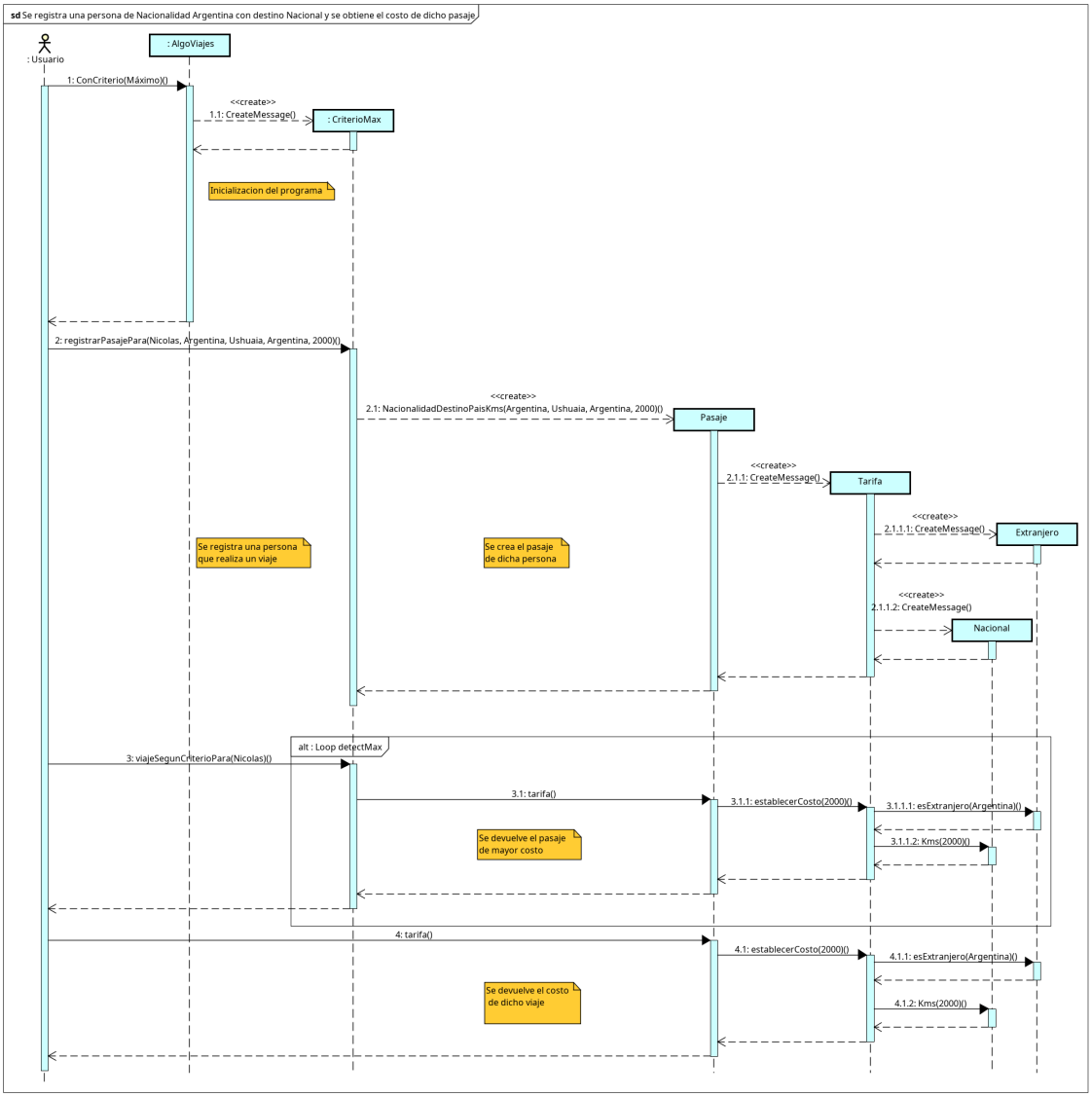

Figura 7: Registro de un usuario y costo de un viaje nacional.

La figura 8 muestra como al momento de inicializar el programa con un criterio invalido si se quiere decir, se lanza una excepción del tipo *CriterioNoEncontrado*.

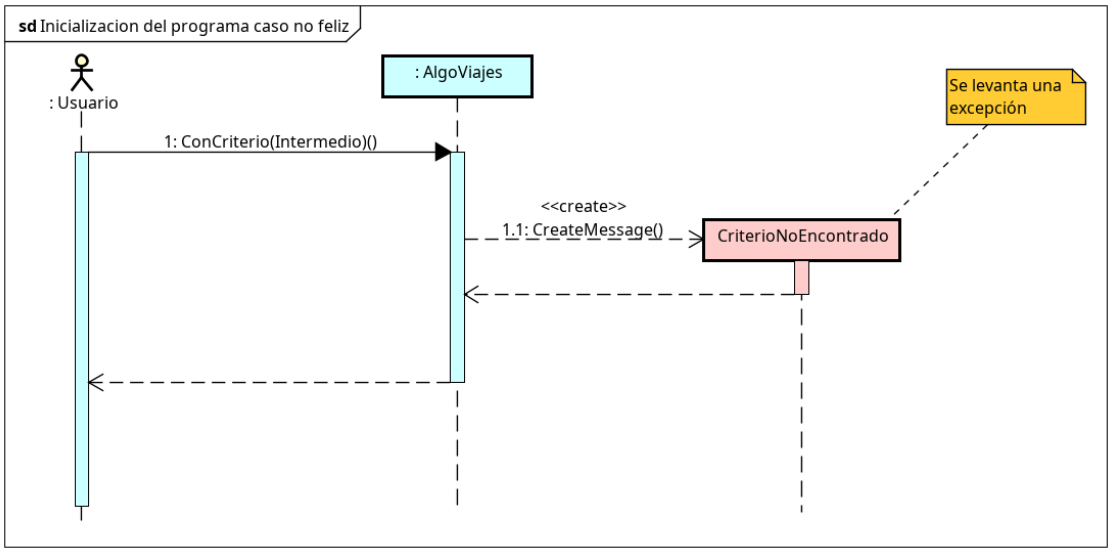

Figura 8: Lanzamiento de una excepción del tipo '*CriterioNoEncontrado*'.