

PREGUNTA1:

ANTES:

En esta fase, vemos y analizamos cómo está constituido la lógica del negocio.

Tenemos tres clases: Airport, Flight y Passenger.

Airport funciona como un cliente de las clases Flight y Passenger. Se crea un pasajeroVip lo agregamos y luego la eliminamos pero por política no se pudo eliminar, luego creamos un pasajeroRegular y no se puede añadir al vueloNegocio pero si al vueloEconomico.

FASE 1:

Sigue la lógica comercial para un vuelo comercial y traduce eso escribiendo una prueba llamada AirportTest.

- **¿Cuáles son los resultados de las pruebas con cobertura obtenidad?**

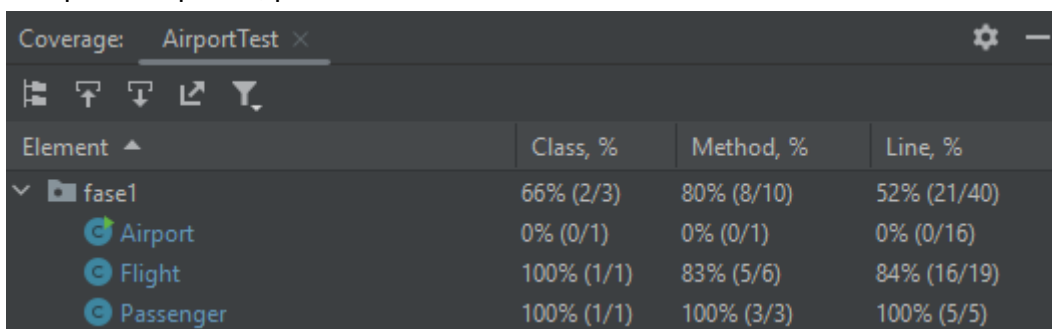
En el Test AirportTest proporcionado: Ya estaban creados dos clases para cada tipo de pasajero.(aun no existen esas clases, en la fase2 las veremos). Con esto no pasan mis pruebas

```
//economyFlight = new EconomyFlight("1");  
//businessFlight = new BusinessFlight("2");
```

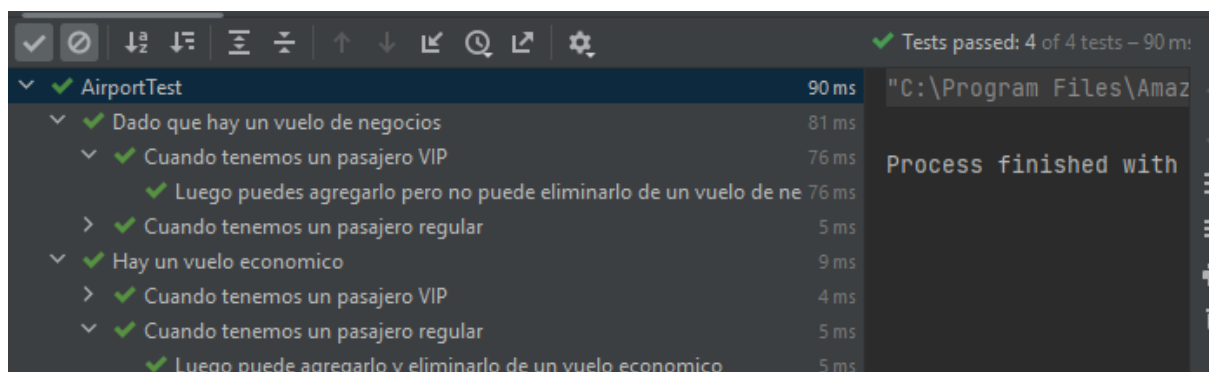
Lo cambiamos de esta forma para que pasen las pruebas.

```
economyFlight = new Flight("1","Economico");  
businessFlight = new Flight("2","Negocios");
```

Todas las pruebas pasan, pero mi cobertura NO es del 100%. es del 66%.



Element	Class, %	Method, %	Line, %
▼ fase1	66% (2/3)	80% (8/10)	52% (21/40)
Airport	0% (0/1)	0% (0/1)	0% (0/16)
Flight	100% (1/1)	83% (5/6)	84% (16/19)
Passenger	100% (1/1)	100% (3/3)	100% (5/5)



Test Case	Duration
▼ AirportTest	90 ms
▼ Dado que hay un vuelo de negocios	81 ms
▼ Cuando tenemos un pasajero VIP	76 ms
Luego puedes agregarlo pero no puede eliminarlo de un vuelo de ne	76 ms
> Cuando tenemos un pasajero regular	5 ms
▼ Hay un vuelo economico	9 ms
> Cuando tenemos un pasajero VIP	4 ms
▼ Cuando tenemos un pasajero regular	5 ms
Luego puede agregarlo y eliminarlo de un vuelo economico	5 ms

Tests passed: 4 of 4 tests - 90 ms

Process finished with

- **¿Puedes indicar algunas conclusiones de lo anterior, necesitamos refactorizar?**

El problema está en la clase Flight, porque falta el caso por defecto en los switch en el que no es "Economico" ni "Negocios". Podemos evitar usar ese switch. Para tenemos que refactorizar usando el principio Abierto-Cerrado, usando polimorfismo.

FASE 2:

En esta fase modificamos la clase Flight a una clase abstracta que contendrá los métodos abstractos (addPassenger, removePassenger), que serán implementadas por las clases derivadas (BusinessFlight, EconomyFlight), en estas nuevas clases modificaremos esos métodos dependiendo el tipo de vuelo.

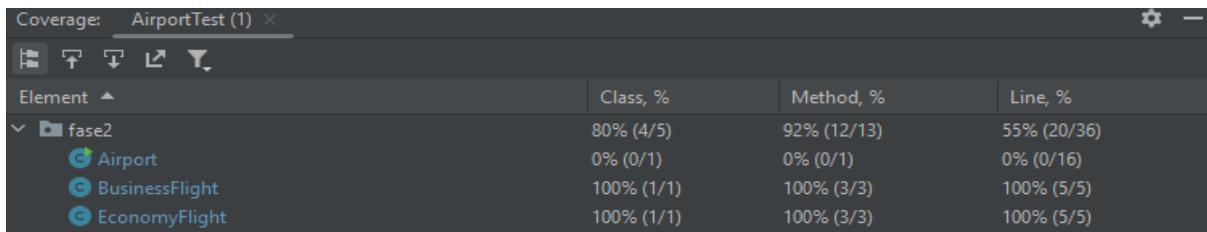
En el **AirportTest**, declaramos la variable **economyFlight** de tipo **Flight** y luego lo instanciamos como **EconomyFlight** esto para aprovechar la herencia y la capacidad de polimorfismo.

- **La refactorización y los cambios de la API se propagan a la implementación de las pruebas. ¿Cómo?**

si, ya refactorizado aprovechamos el uso de polimorfismo.

- **¿Cuál es el código de cobertura ahora?. ¿Ayudó la refactorización a la mejor calidad de código?**

Mejoro la calidad del código testAirport ya que ahora se logra el 100% (sin contar la main). con la main 80%.



Coverage: AirportTest (1) ×			
Element	Class, %	Method, %	Line, %
▼ fase2	80% (4/5)	92% (12/13)	55% (20/36)
Airport	0% (0/1)	0% (0/1)	0% (0/16)
BusinessFlight	100% (1/1)	100% (3/3)	100% (5/5)
EconomyFlight	100% (1/1)	100% (3/3)	100% (5/5)

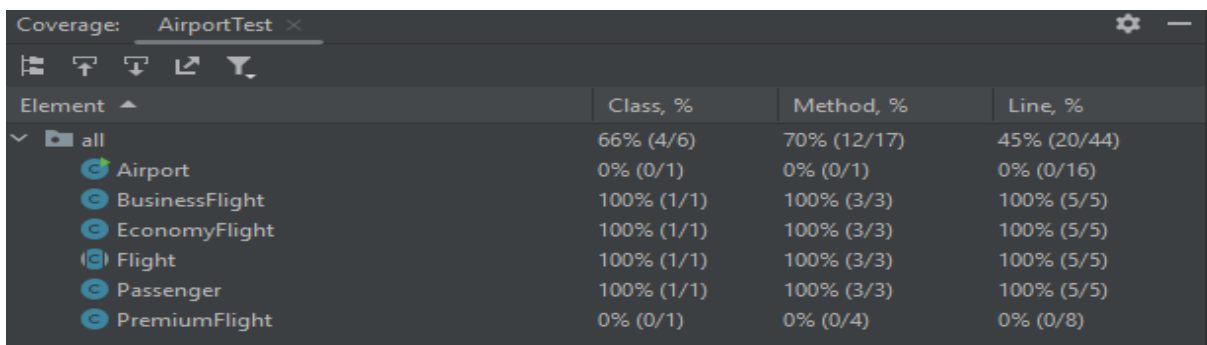
FASE 3:

Creamos una clase PremiumFlight sin extender de Flight, con métodos agregar y eliminar. considerando las políticas establecidas.

- **Utiliza la clase AirportTest refactorizada antes de pasar al trabajo para el vuelo premium en el código desarrollado como mejora a tus resultados. Ver el código entregado en la evaluación.**

Con la clase creada, el coverage es de 60%(con la main).

Ahora se agregan sus pruebas unitarias en el archivo AirportTest la cual el coverage es del 83%(con la main) ,sin el main llega al 100%.



Coverage: AirportTest ×			
Element	Class, %	Method, %	Line, %
▼ all	66% (4/6)	70% (12/17)	45% (20/44)
Airport	0% (0/1)	0% (0/1)	0% (0/16)
BusinessFlight	100% (1/1)	100% (3/3)	100% (5/5)
EconomyFlight	100% (1/1)	100% (3/3)	100% (5/5)
Flight	100% (1/1)	100% (3/3)	100% (5/5)
Passenger	100% (1/1)	100% (3/3)	100% (5/5)
PremiumFlight	0% (0/1)	0% (0/4)	0% (0/8)

Coverage: AirportTest			
Element	Class, %	Method, %	Line, %
all	83% (5/6)	94% (16/17)	63% (28/44)
Airport	0% (0/1)	0% (0/1)	0% (0/16)
BusinessFlight	100% (1/1)	100% (3/3)	100% (5/5)
EconomyFlight	100% (1/1)	100% (3/3)	100% (5/5)
Flight	100% (1/1)	100% (3/3)	100% (5/5)
Passenger	100% (1/1)	100% (3/3)	100% (5/5)
PremiumFlight	100% (1/1)	100% (4/4)	100% (8/8)

FASE 4:

Cambiaremos la clase PremiumFlight a que extienda de Flight y sobreescribimos sus metodos , el estilo TDD significa ser impulsado por las pruebas, por lo que primero creamos la prueba para fallar (retornamos false) y luego escribimos el código que hará que la prueba pase.

Al ejecutar el codigo de cobertura es de 83% (con la main) y 100% (sin la main). por la refactorizacion.

FASE 5:

- ¿Cómo cambia la clase Flight en este contexto?.

En la clase cambiamos de tipo List a Conjunto con hashset, para garantizar la unicidad de los pasajeros.

- Luego crea una nueva prueba para verificar que un pasajero solo se puede agregar una vez a un vuelo ¿Consigues una mejor cobertura de código?

Modificamos el AirportTest, pero trabajamos con TDD, primero creamos los test en rojo, despues ya lo implementamos.

- Para @DisplayName("VueloEconomico")

PasajeroRegular:

Agregamos una prueba que se repita 2 veces, hacemos que se agregue 2 veces el pasajero "checha", mediante un for usamos repetitionInfo.getCurrentRepetition(), para obtener el numero de repeticion actual, y cuando hacemos el assertAll verificamos que solo se agrego una vez el pasajero checha, por mas que hicimos 2 veces

"economyFlight.addPassenger(checha)"

Analogamente lo hacemos para PasajeroVip, y @DisplayName("VueloNegocios"), @DisplayName("VueloPremium"), respetando las políticas.

Se consigue el 100% de cobertura.

Coverage: AirportTest ×

Element ▲	Class, %	Method, %	Line, %
▼ all	83% (5/6)	93% (15/16)	60% (25/41)
🟢 Airport	0% (0/1)	0% (0/1)	0% (0/16)
🟢 BusinessFlight	100% (1/1)	100% (3/3)	100% (5/5)
🟢 EconomyFlight	100% (1/1)	100% (3/3)	100% (5/5)
🟢 Flight	100% (1/1)	100% (3/3)	100% (5/5)
🟢 Passenger	100% (1/1)	100% (3/3)	100% (5/5)
🟢 PremiumFlight	100% (1/1)	100% (3/3)	100% (5/5)