

Pregunta 1:

Ejecutando la carpeta Anterior:

```
Lista de pasajeros de vuelos de negocios:
Cesar
Lista de pasajeros de vuelos economicos:
Jessica
```

Ejecutando prueba de cobertura desde IntelliJ IDEA.

Element ▲	Class, %	Method, %	Line, %
all	100% (6/6)	80% (16/20)	82% (66/80)
Airport	100% (1/1)	100% (1/1)	100% (16/16)
Flight	100% (1/1)	66% (4/6)	63% (12/19)
Passenger	100% (1/1)	100% (3/3)	100% (5/5)

Presentan diferente porcentaje de cobertura porque por ejemplo en el caso de Flight solo 4 de 6 métodos se han ejecutado en un conjunto de pruebas, mientras que en Passenger sus 3 métodos si han sido ejecutados en un conjunto de pruebas al igual que en Airport.

Pregunta 2:

John tiene la necesidad de refactorizar la aplicación de pasar del estilo procedimental al polimorfismo, porque emplea 3 tipos diferentes de vuelo (BusinessFlight, EconomyFlight y PremiumFlight) que extiende de Flight, debido a que la forma general los 3 tipos de vuelos agregan, remueven pasajeros, etc ..., mientras que en cada tipo de vuelo específico se categoriza a los pasajeros si son VIP o no.

Pregunta 3 - Fase\_3 :

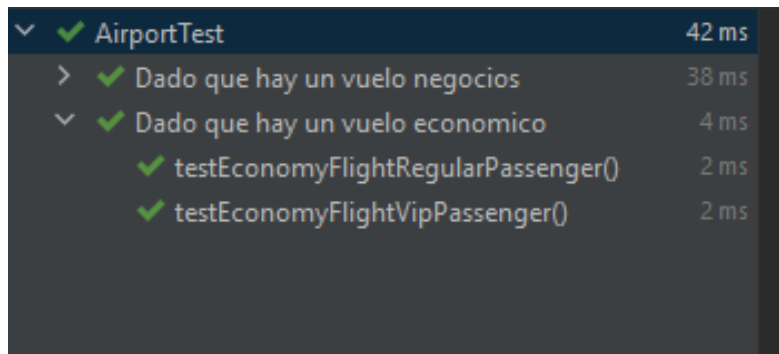
en la clase BusinessFlightTest:

<default package>	39 ms	"C:\Program Files\Java
AirportTest	39 ms	
Dado que hay un vuelo negocios	37 ms	
testBusinessFlightVipPassenger()	33 ms	org.opentest4j.Asserti
testBusinessFlightRegularPassenger()	4 ms	Expected :false
Dado que hay un vuelo economico	2 ms	Actual :true

en la clase EconomyFlightTest:

<default package>	45 ms	"C:\Program File
AirportTest	45 ms	
Dado que hay un vuelo negocios	37 ms	org.opentest4j.A
Dado que hay un vuelo economico	8 ms	Expected :false
testEconomyFlightRegularPassenger()	6 ms	Actual :true
testEconomyFlightVipPassenger()	2 ms	<Click to see di

Luego de implementar las correcciones:



✓	AirportTest	42 ms
>	✓ Dado que hay un vuelo negocios	38 ms
✓	Dado que hay un vuelo economico	4 ms
✓	testEconomyFlightRegularPassenger()	2 ms
✓	testEconomyFlightVipPassenger()	2 ms

Pregunta 4:

La refactorización consiste en el cambio interno sin cambiar su funcionalidad externo, esto relacionado al problema de la evaluación se tuvo que refactorizar en la clase abstracta Flight de la fase 2 y la fase 3 el nombre del getPassenger a getPassengerList, todo lo demás no necesita refactorización porque su funcionalidades es suficiente hasta el momento.

Pregunta 5:

se procedió a implementar los métodos abstractos que la clase PremiunFlight hereda de Flight

```
public class PremiumFlight extends Flight {  
  
    1 usage  👤 ARMANDO\Armando  
    public PremiumFlight(String id) { super(id); }  
  
    7 usages  👤 ARMANDO\Armando +1  
    @Override  
    public boolean addPassenger(Passenger passenger) {  
        return false;  
    }  
  
    6 usages  👤 ARMANDO\Armando +1  
    @Override  
    public boolean removePassenger(Passenger passenger) {  
        return false;  
    }  
}
```

### Pregunta 6:

Se procedió a escribir las pruebas a la clase PremiumFlight, tomando en cuenta el resultado de los casos específicos para esta clase y se procedió a verificar que la implementación inicial no satisface las pruebas.

```
ARMANDO\Armando +1
@Nested
@DisplayName("Cuando tenemos un pasajero regular")
class PremRegularPassenger {

    ARMANDO\Armando
    @Test
    @DisplayName("Entonces no puede agregarlo o eliminarlo de un vuelo premium")
    public void testBusinessFlightRegularPassenger() {
        assertAll( heading: "Verifica todas las condiciones para un pasajero regular y un vuelo premium",
            () -> assertEquals( expected: false, premiumFlight.addPassenger(jessica)),
            () -> assertEquals( expected: 0, premiumFlight.getPassengersList().size()),
            () -> assertEquals( expected: false, premiumFlight.removePassenger(jessica)),
            () -> assertEquals( expected: 0, premiumFlight.getPassengersList().size())
        );
    }
}
```

```
ARMANDO\Armando *
@Nested
@DisplayName("Cuando tenemos un pasajero VIP")
class VipPassenger {

    ARMANDO\Armando
    @Test
    @DisplayName("Luego puedes agregarlo pero no puedes eliminarlo de un vuelo premium")
    public void testBusinessFlightVipPassenger() {
        assertAll( heading: "Verifica todas las condiciones para un pasajero VIP y un vuelo premium",
            () -> assertEquals( expected: true, premiumFlight.addPassenger(cesar)),
            () -> assertEquals( expected: 1, premiumFlight.getPassengersList().size()),
            () -> assertEquals( expected: true, premiumFlight.removePassenger(cesar)),
            () -> assertEquals( expected: 0, premiumFlight.getPassengersList().size())
        );
    }
}
```

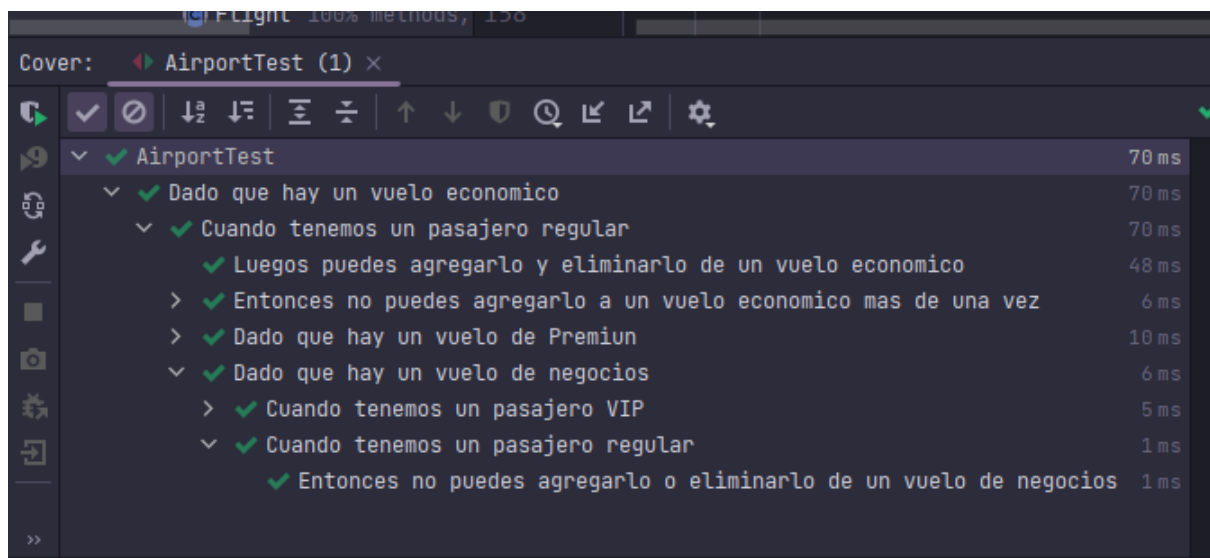
✖ AirportTest	43 ms	
✖ Dado que hay un vuelo economico	43 ms	expected: <true> but was: <false> Comparison Failure:
✖ Cuando tenemos un pasajero regular	43 ms	Expected :true
✓ Luegos puedes agregarlo y eliminarlo de un vuelo economico	23 ms	Actual :false
> ✓ Entonces no puedes agregarlo a un vuelo economico mas de una vez	8 ms	<a href="#">&lt;Click to see difference&gt;</a>
> ✖ Dado que hay un vuelo de Premiun	6 ms	
✓ Dado que hay un vuelo de negocios	6 ms	
> ✓ Cuando tenemos un pasajero VIP	4 ms	
✓ Cuando tenemos un pasajero regular	2 ms	
✓ Entonces no puede agregarlo o eliminarlo de un vuelo de negocios	2 ms	expected: <1> but was: <0> Comparison Failure:
		Expected :1
		Actual :0
		<a href="#">&lt;Click to see difference&gt;</a>

Problema 7:

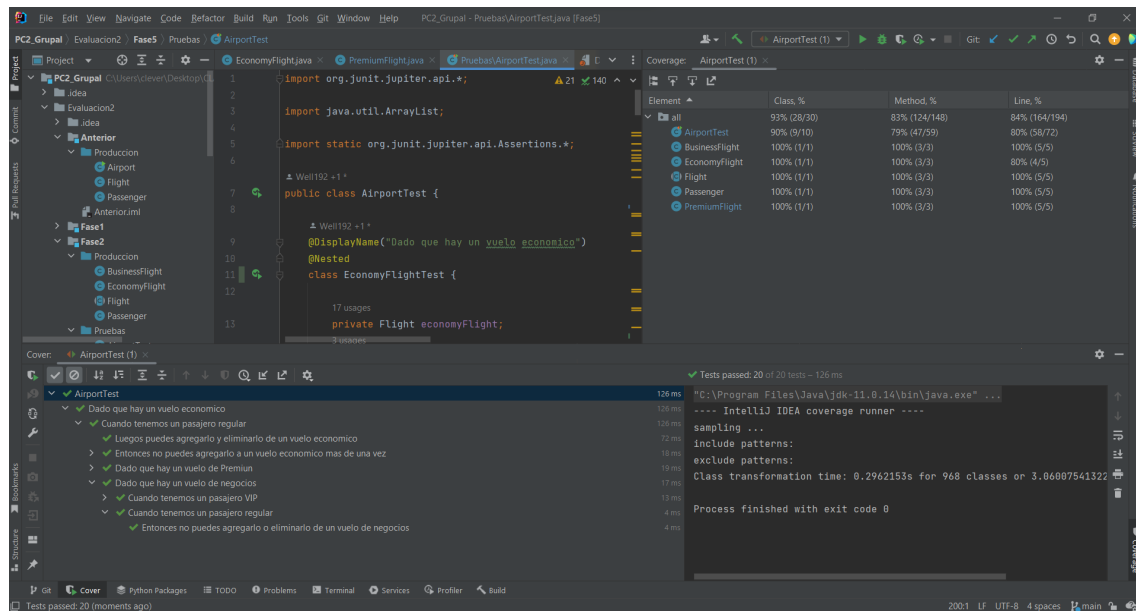
Se agregó la lógica comercial a los métodos addPassenger y removePassenger de la clase PremiumFlight.

```
public boolean addPassenger(Passenger passenger) {  
    if (passenger.isVip()) {  
        return passengers.add(passenger);  
    }  
    return false;  
}  
}  
  
4 usages  fiorellamr *  
@Override  
public boolean removePassenger(Passenger passenger) {  
    return passengers.remove(passenger);  
}
```

Se corrieron las pruebas con cobertura, todas las pruebas pasaron pero se obtuvo una cobertura menor al 100%.



Cover: AirportTest (1) x	
✓ AirportTest	70 ms
✓ Dado que hay un vuelo economico	70 ms
✓ Cuando tenemos un pasajero regular	70 ms
✓ Luego puedes agregarlo y eliminarlo de un vuelo economico	48 ms
> ✓ Entonces no puedes agregarlo a un vuelo economico mas de una vez	6 ms
> ✓ Dado que hay un vuelo de Premiun	10 ms
✓ Dado que hay un vuelo de negocios	6 ms
> ✓ Cuando tenemos un pasajero VIP	5 ms
✓ Cuando tenemos un pasajero regular	1 ms
✓ Entonces no puedes agregarlo o eliminarlo de un vuelo de negocios	1 ms



Pregunta 8:

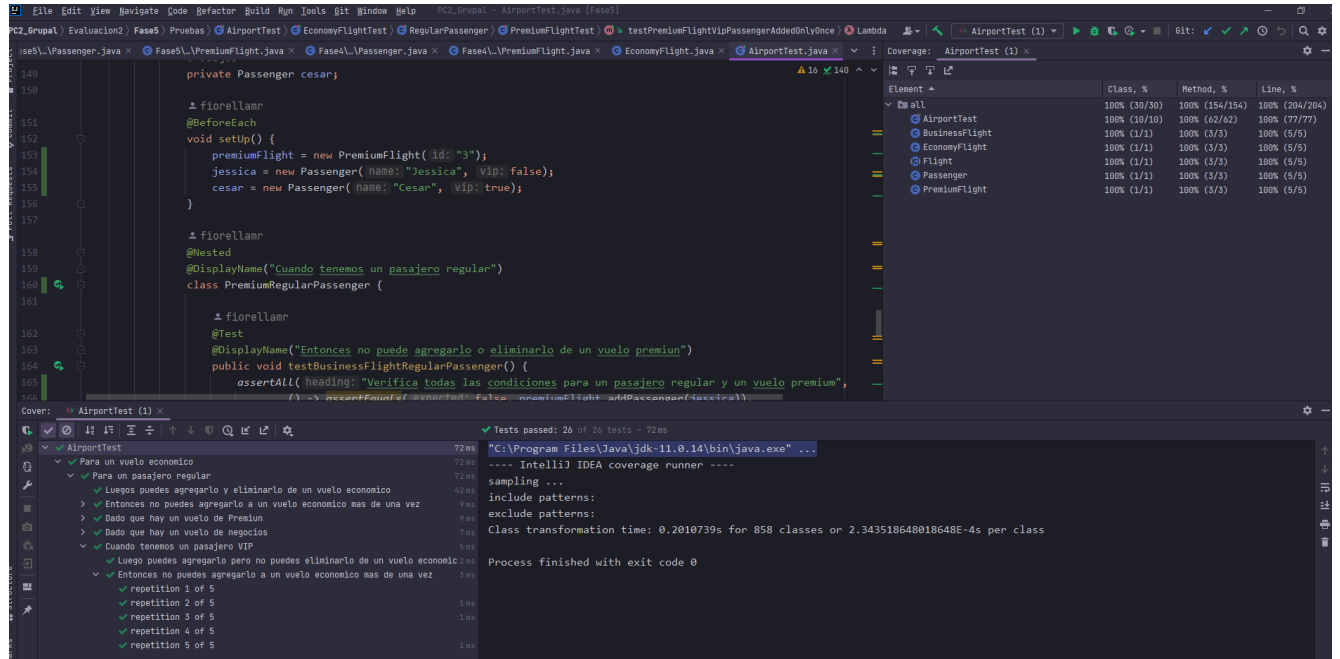
Se completó el test que verifica que un pasajero solo se puede agregar una vez a un vuelo.

```

@DisplayName("Entonces no puedes agregarlo a un vuelo economico mas de una vez")
@RepeatedTest(5)
public void testEconomyFlightRegularPassengerAddedOnlyOnce(RepetitionInfo repetitionInfo) {
    for (int i = 0; i < repetitionInfo.getCurrentRepetition(); i++) {
        economyFlight.addPassenger(jessica);
    }
    assertAll( heading: "Verifica que un pasajero regular se pueda agregar a un vuelo de negocios solo una vez",
        () -> assertEquals( expected: 1, economyFlight.getPassengersSet().size()),
        () -> assertTrue(economyFlight.getPassengersSet().contains(jessica)),
        () -> assertTrue(new ArrayList<>(economyFlight.getPassengersSet()).get(0).getName().equals("Jessica"))
    );
}

```

Se procedió a verificar la cobertura de código



La ejecución de las pruebas es exitosa y tiene una cobertura del 100%