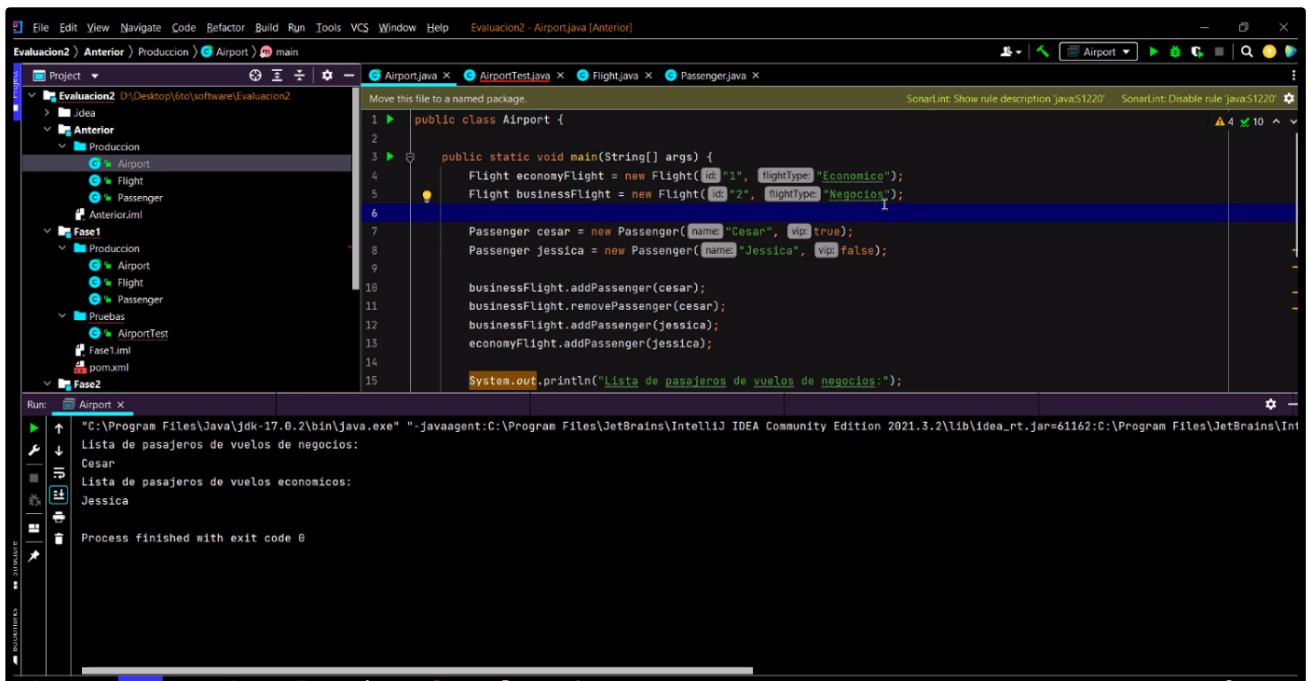


## Ejecuta el programa y presenta los resultados y explica que sucede.

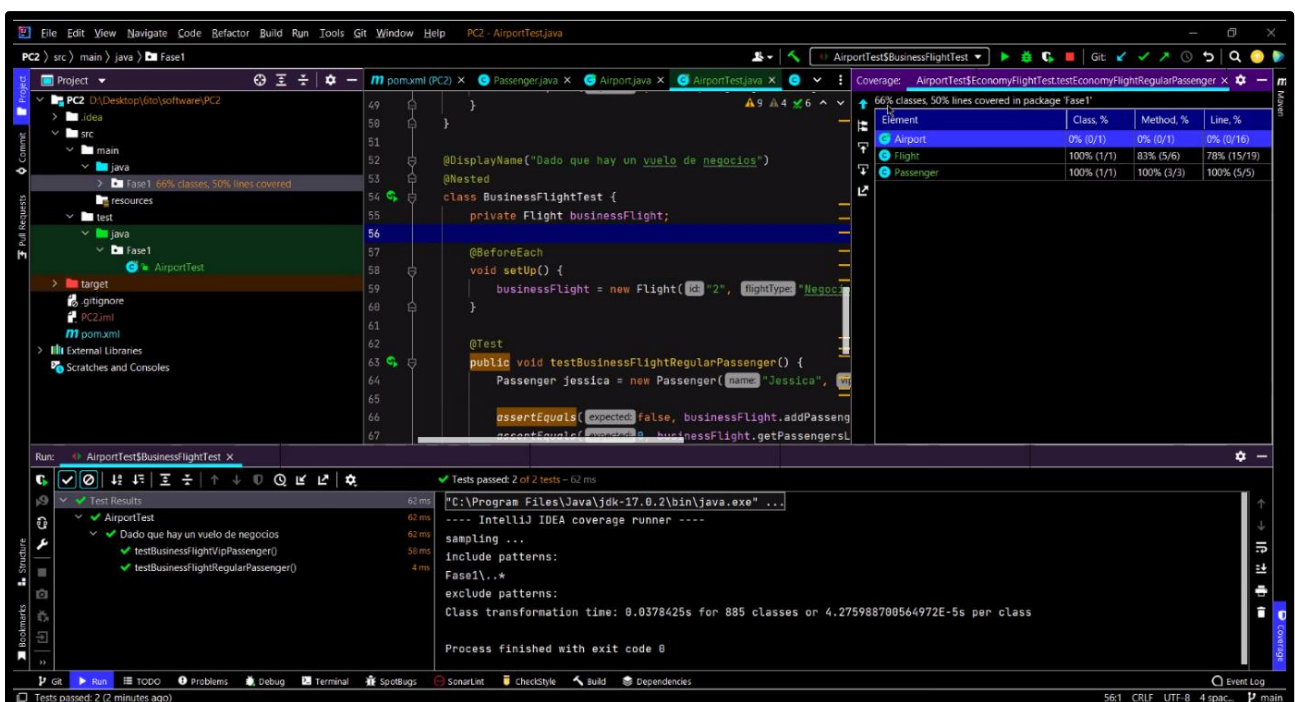
Se inicializa 2 objetos de la clase Flight y 2 objetos de la clase Passenger, según la instancia de los objetos de la clase creada, se agrega los pasajeros de forma indiscriminada y de acuerdo con el valor del atributo de los pasajeros vip se ejecutan los métodos de la clase Flight.



## Pregunta 1

¿Cuáles son los resultados que se muestran? ¿Por qué crees que la cobertura del código no es del 100%?

Se muestra que el código de cobertura tiene un 66%, porque se está testeando solamente dos de las tres clases que están codificadas.



## Pregunta 2

### ¿Por qué John tiene la necesidad de refactorizar la aplicación?

Porque existe la clase Airport en la cual no se hace uso de las pruebas unitarias para asegurar que la clase este bien escrita.

### Revisa la Fase 2 de la evaluación y realiza la ejecución del programa y analiza los resultados.

En la fase 2, se implementó una clase abstracta Flight donde se ve la inicialización de un ArrayList de tipo Passengers y las firmas de los métodos a utilizar, luego se extendió esta clase para crear 2 clases nuevas (BusinessFlight y EconomyFlight

The screenshot displays the IntelliJ IDEA IDE interface. The main editor shows the `AirportTest.java` file with the following code:

```
public class AirportTest {
    @DisplayName("Dado que hay un vuelo economico")
    @Nested
    class EconomyFlightTest {
        private Flight economyFlight;

        @BeforeEach
        void setUp() { economyFlight = new EconomyFlight("1"); }

        @Test
        public void testEconomyFlightRegularPassenger() {
            Passenger jessica = new Passenger("Jessica", false);

            assertEquals("expected: '1', economyFlight.getId());
            assertEquals("expected: true, economyFlight.addPassenger(jessica);", true, economyFlight.addPassenger(jessica));
            assertEquals("expected: 1, economyFlight.getPassengers().size()", 1, economyFlight.getPassengers().size());
            assertEquals("expected: 'Jessica', economyFlight.getPassengers().get(0)", "Jessica", economyFlight.getPassengers().get(0));
        }
    }
}
```

The right sidebar shows the coverage report for the package 'Fase2':

Element	Class, %	Method, %	Line, %
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)

The bottom panel shows the test results for the `AirportTest` class:

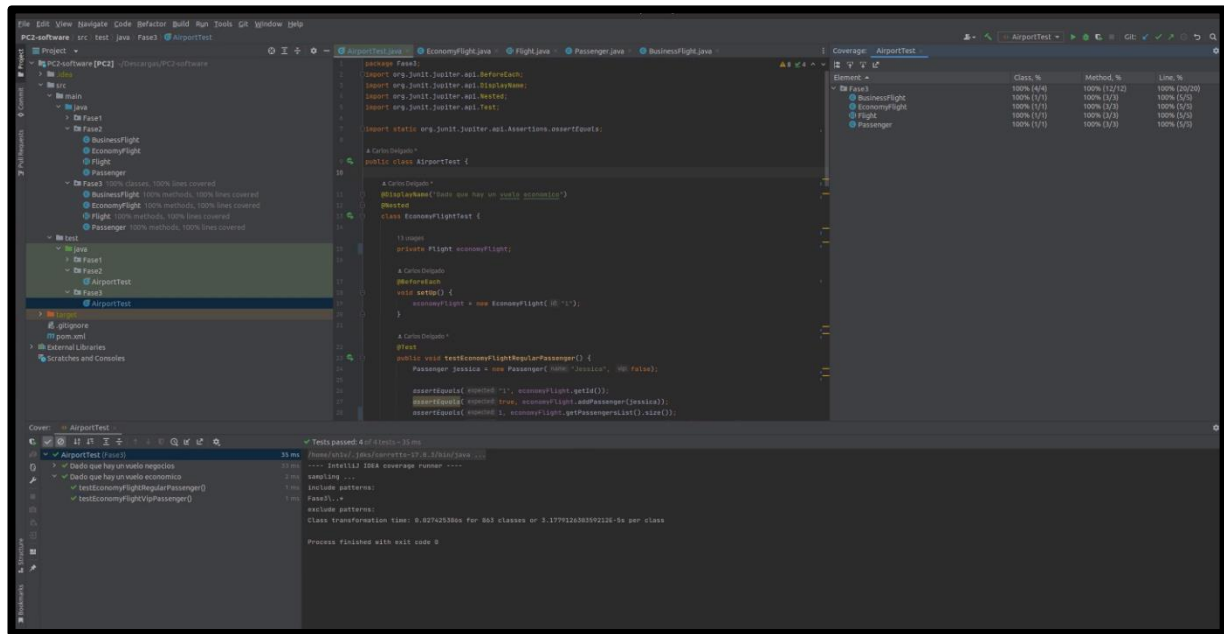
```
Run: AirportTest (1) X
Tests passed: 4 of 4 tests - 43 ms
--- IntelliJ IDEA coverage runner ---
sampling ...
include patterns:
Fase2...
exclude patterns:
Class transformation time: 0.0261117s for 887 classes or 2.9438218714768886E-5s per class
Process finished with exit code 0
```

## Pregunta 3

La refactorización y los cambios de la API se propagan a las pruebas.  
Reescribe el archivo Airport Test de la carpeta Fase 3.

¿Cuál es la cobertura del código?

100%



¿La refactorización de la aplicación TDD ayudó tanto a mejorar la calidad del código?

Sí, porque se fue más específico con los nombres de los métodos.

## Pregunta 4

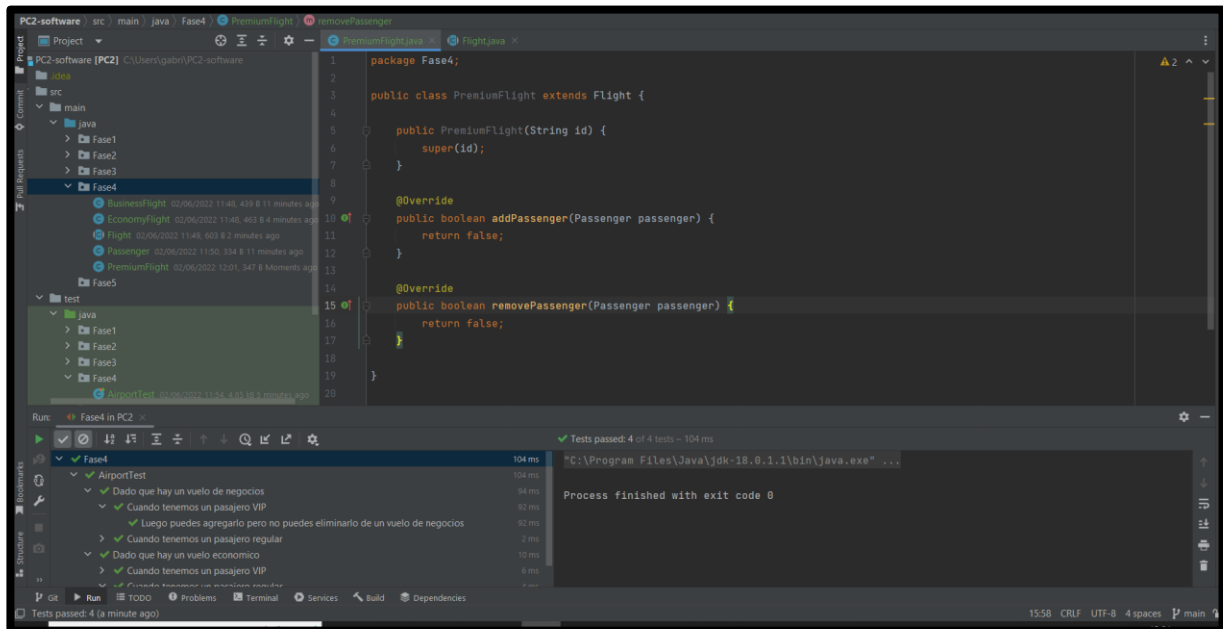
¿En qué consiste esta regla relacionada a la refactorización? Evita utilizar y copiar respuestas de internet. Explica cómo se relaciona al problema dado en la evaluación.

La Regla de Tres aplicada en la programación especifica que cuando se tiene una pieza de código la cual se repite 3 veces, se tiene que refactorizar para así facilitar la visualización y mantenimiento del código.

En este caso se crea una nueva clase Premium Flight con el objetivo de evitar repetición.

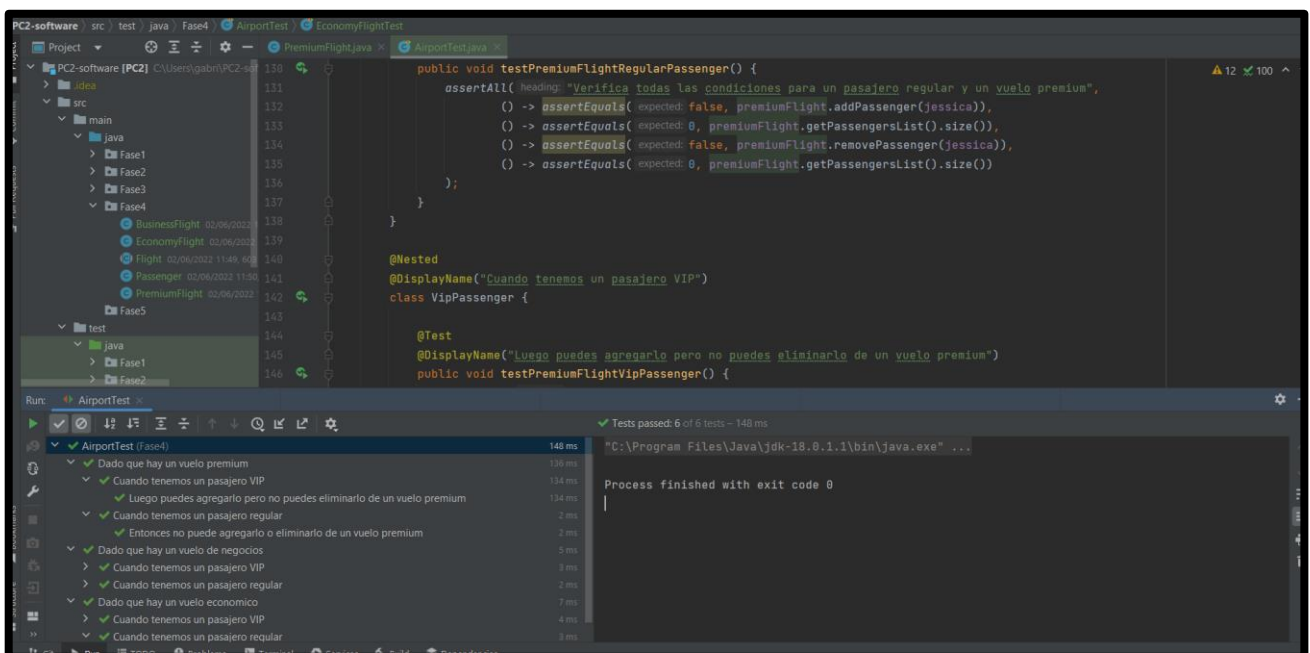
## Pregunta 5

Escribe el diseño inicial de la clase llamada PremiumFlight y agrega a la Fase 4 en la carpeta producción.



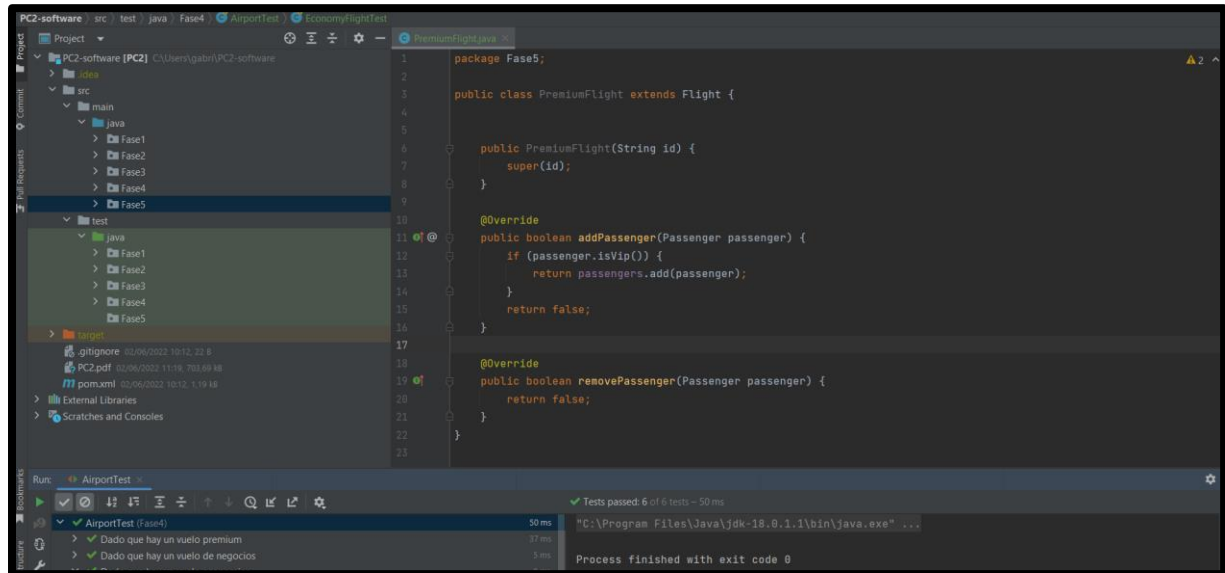
## Pregunta 6

Ayuda a John e implementa las pruebas de acuerdo con la lógica comercial de vuelos premium de las figuras anteriores. Adjunta tu código en la parte que se indica en el código de la Fase 4. Después de escribir las pruebas, John las ejecuta.



## Pregunta 7

Agrega la lógica comercial solo para pasajeros VIP en la clase PremiumFlight. Guarda ese archivo en la carpeta Producción de la Fase 5.



The screenshot shows an IDE with the following components:

- Project Explorer:** Shows a project structure with folders for 'src' (containing 'main' and 'test') and 'target'. The 'test' folder is expanded, showing 'Fase1' through 'Fase5'.
- Editor:** Displays the 'PremiumFlight.java' file. The code is as follows:

```
1 package Fase5;
2
3 public class PremiumFlight extends Flight {
4
5     public PremiumFlight(String id) {
6         super(id);
7     }
8
9     @Override
10    public boolean addPassenger(Passenger passenger) {
11        if (passenger.isVip()) {
12            return passengers.add(passenger);
13        }
14        return false;
15    }
16
17    @Override
18    public boolean removePassenger(Passenger passenger) {
19        return false;
20    }
21 }
22
23
```
- Run Console:** Shows the execution of 'AirportTest'. The output is:

```
✓ Tests passed: 6 of 6 tests - 50 ms
✓ AirportTest (Fase4) 50 ms
  > Dado que hay un vuelo premium 37 ms
  > Dado que hay un vuelo de negocios 5 ms
Process finished with exit code 0
```

## Pregunta 8

Ayuda a John a crear una nueva prueba para verificar que un pasajero solo se puede agregar una vez a un vuelo. La ejecución de las pruebas ahora es exitosa, con una cobertura de código del 100 %. John ha implementado esta nueva característica en estilo TDD.