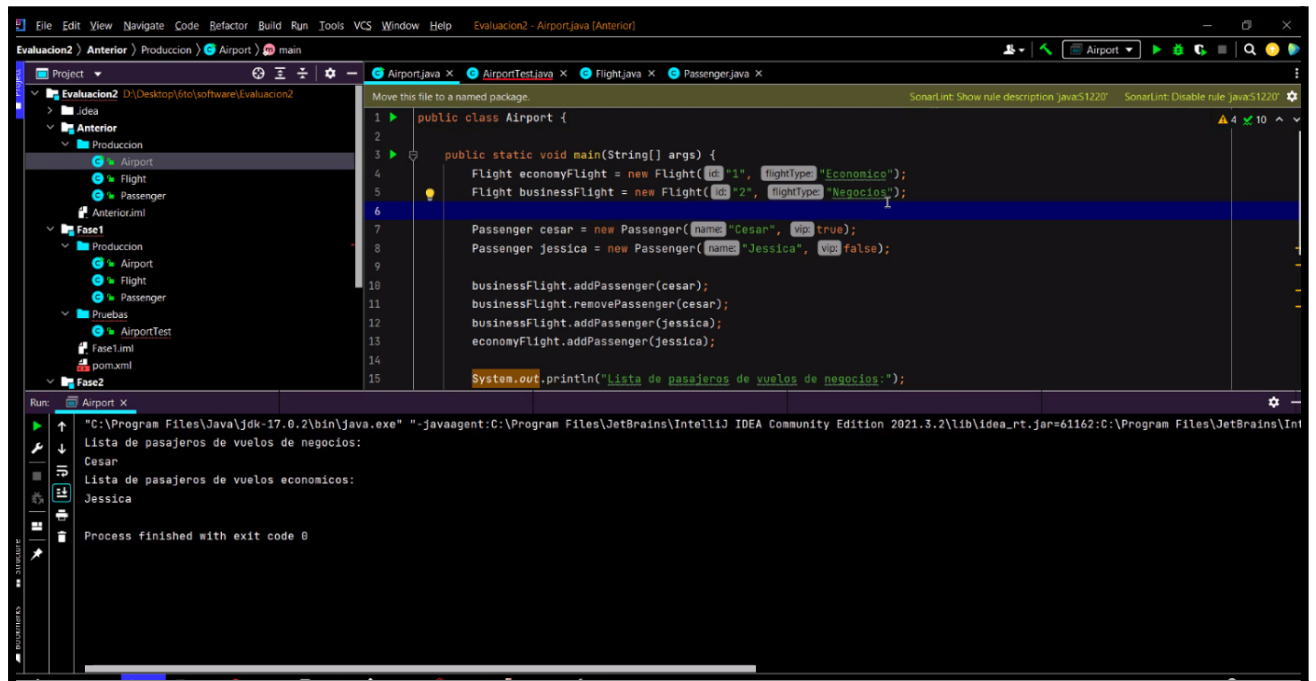


## Ejecuta el programa y presenta los resultados y explica que sucede.

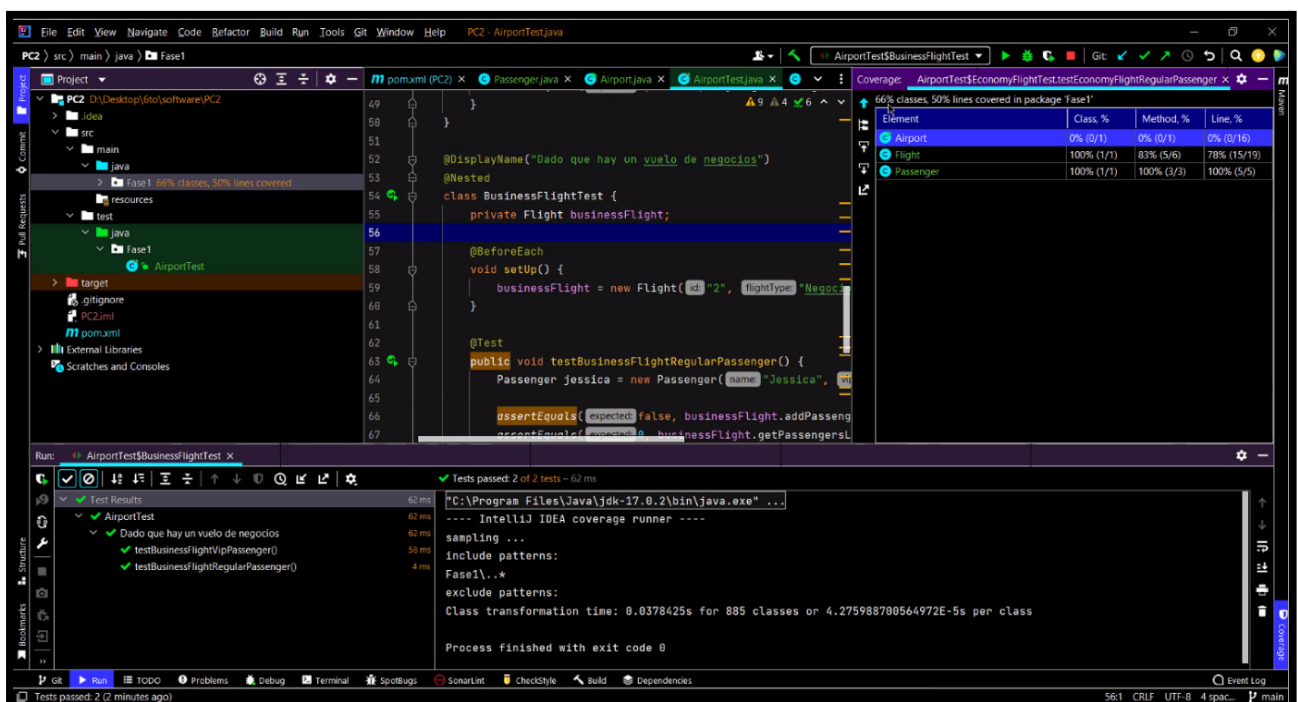
Se inicializa 2 objetos de la clase Flight y 2 objetos de la clase Passenger, según la instancia de los objetos de la clase creada, se agrega los pasajeros de forma indiscriminada y de acuerdo con el valor del atributo de los pasajeros vip se ejecutan los métodos de la clase Flight.



## 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 esté 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), en lo que respecta a las pruebas unitarias tiene una cobertura del 100%

Project: PC2 D:\Desktop\6to\software\PC2

main

- java
  - Fase1 66% classes, 50% lines covered
    - Airport 0% methods, 0% lines covered
    - Flight 83% methods, 78% lines covered
    - Passenger 100% methods, 100% lines covered
  - Fase2 100% classes, 100% lines covered
    - BusinessFlight 100% methods, 100% lines covered
    - EconomyFlight 100% methods, 100% lines covered
    - Flight 100% methods, 100% lines covered
    - Passenger 100% methods, 100% lines covered
- resources
- test
  - java
    - Fase1
      - AirportTest
    - Fase2
      - AirportTest

Run: AirportTest (1) x

Test Results

- ✓ AirportTest 43 ms
  - ✓ Dado que hay un vuelo negocios 43 ms
  - ✓ Dado que hay un vuelo economico 40 ms
    - ✓ testEconomyFlightRegularPassenger() 3 ms
    - ✓ testEconomyFlightVipPassenger() 1 ms

Tests passed: 4 of 4 tests - 43 ms

IntelliJ IDEA coverage runner

---- 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

| 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) |

## 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.

## 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.

## 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.**

#### **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.**