Pregunta 1 (3 puntos) Si ejecutamos las pruebas con cobertura desde IntelliJ IDEA, ¿cuales son los resultados que se muestran?, ¿Por qué crees que la cobertura del código no es del 100%? Puedes revisar https://www.jetbrains.com/help/idea/code-coverage.html y https://en.wikipedia.org/wiki/Code_coverage para responder la pregunta. **(MIGUEL)**

-Al ejecutar una prueba con cobertura en JUnit en IntelliJ IDEA, se muestra una ventana donde podemos una tabla con 3 columnas , donde la primera columna nos muestra el porcentaje del código que es de la clase, la según nos muestra el porcentaje de líneas de código que conforman los métodos de la clase, y la tercera columna nos muestra el porcentaje de líneas de código que fueron evaluadas en la prueba, es decir , el porcentaje de cobertura del código. Además, en la parte izquierda, donde se muestran todas las clases creadas, se pueden per los porcentajes de código de la clase y el porcentaje de cobertura.

-Si un código al ser sometido a una prueba no muestra un 100% de cobertura, significa que no todo lo que las líneas de código son revisadas en la prueba unitaria, esto también nos quiere decir, que no es totalmente seguro que el código no presente fallas, ya que faltaría realizar pruebas que involucre las líneas de código faltantes.

Pregunta 2 (1 punto) ¿ Por qué John tiene la necesidad de refactorizar la aplicación? **(HANS, MIGUEL Y RENZO)**
John tuvo que refactorizar la clase Flight y Airport para sustituirlos por la clase abstracta Flight y las clase BusinessFlight y EconomyFlight , las cuales heredan de Flight, esto lo hizo con el fin de poder mejorar la organización de su código y ser más específico al momento de escribir los métodos de cada tipo de vuelo y poder emplearlos individualmente en lugar de usar un solo código para representar ambos tipos de vuelos, además de crear pruebas individuales para los métodos de cada tipo de vuelo.

Pregunta 3 (3 puntos) La refactorización y los cambios de la API se propagan a las pruebas. Reescribe el archivo Airport Test de la carpeta **Fase 3**. **(RENZO)**

Pregunta 4 (2 puntos): ¿En qué consiste está regla relacionada a la refactorización?. Evita utilizar y copiar respuestas de internet. Explica cómo se relaciona al problema dado en la evaluación. **(HANS, MIGUEL Y RENZO)**
Para el TDD tendríamos que plantear pruebas acordes a la funcionalidad que necesitemos agregar (antes de agregar el código de la misma), al tener las pruebas con un error ya que no contamos con la funcionalidad desarrollada pasamos a hacer el código de la misma para que la prueba planteada en el inicio pueda pasarse. Al tener la prueba en verde (sin error) es donde tendremos que aplicar la refactorización, si nuestro nuevo código pasa las pruebas sin errores ya se habría terminado el trabajo, si no se da este caso debemos solucionar los errores causados por una mala refactorización.

Pregunta 5 (2 puntos): Escribe el diseño inicial de la clase llamada PremiumFlight y agrega a la **Fase 4** en la carpeta producción.**(HANS)**

Pregunta 6 (3 puntos): 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. **(RENZO)**

Pregunta 7 (3 puntos): Agrega la lógica comercial solo para pasajeros VIP en la clase

PremiumFlight. Guarda ese archivo en la carpeta Producción de la **Fase 5**.**(HANS)**

Pregunta 8 (3 puntos) 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.**(MIGUEL)**