Pregunta 1

Antes:
Se crearon las clases Airport, Flight y Passenger usando el código provisto. La clase Flight define los métodos addPassenger y removePassenger para añadir y remover pasajeros, donde se implementa que para tipos de vuelo Economico y de Negocios se puede añadir tano pasajeros regulares como VIP pero solo se puede remover pasajeros regulares.
En la clase Airport se define el método main. En este método se crean dos vuelos: uno Economico y uno de Negocios y dos pasajeros: uno regular y uno VIP. En el vuelo de negocios se llama a los métodos para añadir y eliminar al cliente VIP y añadir al pasajero regular. En el vuelo económico se llama al método para añadir al cliente regular. Finalmente se imprimen los pasajeros que quedan en ambos vuelos y se observa que se cumple la lógica implementada ya que solo el pasajero VIP es añadido al vuelo de negocios y no es eliminado mientras que el pasajero regular es añadido al vuelo económico.

Fase 1:
En esta fase se crea el archivo AirportTest para realizar las pruebas. Se crean dos pruebas:
- En la primera prueba se crea un vuelo económico y se crean dos usuarios: uno regular y uno VIP. Se llama a los métodos correspondientes para añadir y eliminar a ambos usuarios de ambos vuelos y se comprueba que ambos son añadidos pero solo se elimina el usuario regular.
- En la segunda prueba se crea un vuelo de negocios y se crean dos usuarios: uno regular y uno VIP. Se llama a los métodos correspondientes para añadir y eliminar a ambos usuarios de ambos vuelos y se comprueba que solo el usuario VIP es añadido al vuelo y además no es removido.
Las pruebas pasan. La cobertura indica un 80% de cobertura de línea y de rama para la clase Flight y 83% y 100% de cobertura de línea y de rama para la clase Passenger.

Fase 2:
La clase Flight se hace abstracta, se elimina el campo flightType y se modifica el constructor. Los campos id y passengers se cambian a acceso protected. Los métodos addPassenger y removePassenger se vuelven abstractos. Se crean las clases EconomyFlight y BusinessFlight que extienden Flight y se sobreescribern los métodos addPassenger y removePassenger. En el método main de Airport se cambia el llamado al constructor de Flight por los constructores de EconomyFlight y BusinessFlight. Se comprueba que al ejecutar el método main el resultado es el mismo luego de la refactorizacion.
En las pruebas se procede igualmente a cambiar el llamado al constructor de Flight por los constructores de EconomyFlight y BusinessFlight. Las pruebas pasan. La cobertura es de 100% para todos los tipos de cobertura para las nuevas clases EconomyFlight y BusinessFlight. La clases Flight y Passenger tienen ambas un 66% de cobertura de método, 83% de cobertura de línea y 100% de cobertura de rama. La clases creadas BusinessFlight y EconomyFlight tienen 100% de cobertura.
Se observa que la refactorización mejora la cobertura de línea y de rama de la clase Flight y además mejora la calidad del código ya que ahora se entiende más fácilmente y también es más fácil de extender su funcionalidad, ya que si se quiere añadir otro tipo de vuelo solo es necesario crear una clase que extienda la clase Flight sin modificar el código existente, cumpliendo por lo tanto el principio Solid Abierto/Cerrado.

Fase 3:
Se reemplaza el código de la clase AirportTest por el código provisto refactorizado. Este código contiene las pruebas para las clases EconomyFlight y BusinessFlight. Se ejecutan las pruebas y se observa que se obtiene una cobertura del 100% de las clases Flight, Passenger, BusinessFlight y EconomyFlight. Por lo tanto este conjunto de pruebas es más completo que el anterior ya que realiza una mayor cobertura del código.

Fase 4:
Se crea la clase PremiumFlight, la cual extiende Flight, se sobreescriben sus métodos pero no implementan la lógica, solo retornan false. Se modifica la clase AirportTest para agregar las pruebas a la clase PremiumFlight en la cual se definen las pruebas para pasajeros regulares y VIP para vuelos premium. Se observa que las pruebas pasan para pasajeros regulares pero dan error para pasajeros VIP. Por lo que solo es necesario agregar la lógica comercial para pasajeros VIP, con lo cual las pruebas pasan.

Fase 5:
En esta fase se modifica la estructura de la lista de pasajeros a conjunto. En vez de utilizar la interfaz List se usa la interfaz Set y en vez de utilizar la clase ArrayList se usa la clase HashSet. Para el método getPassengerList, se usa el método unmodifiableSet en vez de unmodifiableList. También se modifica en las pruebas la parte que verifica que un pasajero se haya añadido al vuelo, utilizando contains en vez de get y assertTrue en vez de assertEquals. Se comprueba que las pruebas pasan. Luego se modifica las pruebas agregando código para añadir al mismo pasajero dos veces y comprobando que el método retorna false y el cantidad de pasajeros no se modifica. La cobertura de código es la misma que en la fase anterior.