**Diseño de Pruebas unitarias.**

Especificación de Escenarios de prueba:

|  |  |  |
| --- | --- | --- |
| Nombre: | Clase: | Especificación: |
| Stage1 | Date | Vacío |
| Stage2 | Flight | Vacío |
| Stage3 | Airport | Vacío |
| Stage4 | Airport | Una instancia de Airport ya ha sido creada. |
| Stage5 | LinkedFlightList | Vacío |
| Stage6 | Airport |  |
| Stage7 | Airport | El aeropuerto ha sido instanciado y 150 vuelos completamente al azar han sido generados. |

Diseño de pruebas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 1: | Objetivo: Probar que el programa crea correctamente instancias de las diferentes clases del modelo. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| DateTest | Date() | Stage1 |  | Se crea un objeto de la clase Date con datos al azar |
| DateTest | Date(int,int,int,int,int) | Stage1 | d = 12  m = 6  y = 2019  h = 14  m = 16 | Se crea un objeto de la clase Date con los datos enviados por parámetro.  getDate() = “2019-06-12”  getTime() = “02:16 PM”  toString() = “2019-06-12-02:16 PM” |
| DateTest | Date(String) | Stage1 | s=“2019-06-12 02:16 PM” | Se traduce de la cadena el objeto date adecuado. |
| FlightTest | Flight(String, String, String, int) | Stage2 | al = “Copa Airlines”  fn = “C80192”  d = “Paris”  bg = 2 | Se crea un objeto de la clase Flight con los datos enviados por parámetro. |
| AirportTest | Airport() | Stage3 |  | Se crea el objeto Airport. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 2: | Objetivo: Probar que el aeropuerto se llena sin mayor problema con datos al azar y que cada vuelo tiene un identificador único. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| AirportTest | generateFlights(int) | stage4 | 100 | Ninguna de las 100 entradas es creada con errores, todos los vuelos tienen un identificador diferente. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 3: | Objetivo: Probar que los métodos para añadir elementos a la lista funcionan de manera correcta. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| LinkedFlightTest | LinkedFlightList() | stage5 |  | Una instancia de la clase LinkedFlightList es creada |
| LinkedFlightTest | add(Flight) | stage5 | Un vuelo completamente generado al azar | La lista debe ser no nula, y el primer elemento debe estar enlazado consigo mismo.  isEmpty() = false. |
| LinkedFlightTest | add(Flight) | stage5 | Cinco vuelos, todos generados completamente al azar. | La lista debe ser no nula, no vacía, de tamaño 5 y cada elemento de esta debe estar conectado con sus siguientes. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 4: | Objetivo: Probar que los métodos para conocer información de la lista funcionan de manera correcta. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| LinkedFlightTest | isEmpty() | stage5 |  | True |
| LinkedFlightTest | get(int) | stage6 | 1 | El método retorna el segundo elemento de la lista |
| LinkedFlightTest | get(int) | stage6 | 3 | El método retorna el último elemento de la lista |
| LinkedFlightTest | set(int, Flight) | stage6 | 3, un vuelo generado al azar | El elemento antes ubicado en la cuarta posición debe ahora ser diferente al que ya había. Además, el ahora ubicado ahí debe ser el generado al azar. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 5: | Objetivo: Probar que el método de ordenamiento implementado en la clase LinkedFlightList funciona de manera efectiva. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| LinkedFlightTest | sort(Comparator) | stage6 | Un elemento Comparator que compare utilizando el número de puerta de abordaje. | Cada elemento de la lista finalmente ordenada tiene asignada una puerta de abordaje menor que su elemento siguiente, exceptuando el último elemento de la lista. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 6: | Objetivo: Probar que el método para seccionar una lista funciona correctamente. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| LinkedFlightTest | subList(int,int) | stage6 | initIndex = 0  finalIndex = 2 | Se devuelve una lista con los elementos desde el primero hasta el segundo de la lista original. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 7: | Objetivo: Probar que los métodos de búsqueda funcionan correctamente. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| AirportTest | search(Sortings, String) | stage6 | Sortings.AIRLINE, “Avianca” | Se retorna el tercer elemento de la lista de aviones. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.AIRLINE, “Ecolines” | Se entrega un elemento nulo. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.DESTINATION, “Bogota” | Se retorna el cuarto elemento de la lista de aviones. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.DESTINATION, “Ciudad de Mexico” | Se entrega un elemento nulo. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.FLIGHT\_NUMBER, “C80192” | Se retorna el segundo elemento de la lista de aviones. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.FLIGHT\_NUMBER, “D80192” | Se entrega un elemento nulo. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.BOARDING\_GATE, 4 | Se retorna el primer elemento de la lista de aviones. |
| AirportTest | search(Sortings, String) | stage6 | Sortings.BOARDING\_GATE, 5 | Se entrega un elemento nulo. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba No 8: | Objetivo: Probar que los métodos de ordenamiento funcionan correctamente. | | | |
| Clase | **Método** | **Escenario** | **Entradas** | **Salidas** |
| AirportTest | sortTime() | stage7 |  | Cada elemento exceptuando el último está en una fecha y hora anterior al siguiente en la lista. |
| AirportTest | sortAirline() | stage7 |  | Cada elemento exceptuando el último tiene asignada una aerolínea lexicográficamente menor al siguiente de la lista. |
| AirportTest | sortDestination() | stage7 |  | Cada elemento exceptuando el último tiene una ciudad de destino lexicográficamente menor a su siguiente en la lista. |
| AirportTest | sortFN() | stage7 |  | Cada elemento tiene un número de vuelo lexicográficamente menor a su siguiente en la lista, exceptuando el último. |
| AirportTest | sortBG() | stage7 |  | Cada elemento de la lista tiene asignado un número de puerta de abordaje menor al de su siguiente en la lista, exceptuando al último elemento de esta. |