DISEÑO DE TEST

HASHTABLE TEST

Primero que todo definimos dos setUp para hacer uso de ellas en diferentes funcionalidades de la clase hashTable, en el setUp1 única y exclusivamente definimos el hashTable con un size de 10, por otro lado, el setUp2 aparte de la definición del hashTable con un size de 5 también insertamos 3 pacientes con diferentes datos en el hastTable

Test de insertElement:

insertElement1() : Utilizamos el setUp1 además de crear un paciente e insertarlo en hashTable, primeramente buscamos al paciente que acabamos de meter colocando su propia key y viendo si efectivamente se insertó en el hashTable haciendo uso de la funcionalidad search(), por lo tanto el resultado esperado nos arrojaría un objeto de tipo Paciente que tiene que ser igual al que nosotros creamos.

inserElement2() : Utilizamos nuevamente el setUp1 y creamos un paciente diferente al anterior y lo insertamos en hashTable, utilizamos el método search() de la clase hashTable y le pasamos como parámetro el mismo id del paciente que acabamos de crear, con el objetivo de verificar si el paciente fue insertado, por lo tanto el valor que esperamos es el mismo paciente que ingresamos.

insertElement3() : Nuevamente hacemos uso del setUp1, volvemos a crear e insertar un paciente diferentes a los dos anteriores, buscamos con su id en el método search() y nos tiene que devolver el mismo pacientes que acabamos de insertar.

Finalmente vemos que los paciente si fueron ingresados en los diferentes tests ya que todos fueron aceptados.

Test de searchElement:

searchElement1(), searchElement2(), searchElement() : En los tres test hacemos uso del setUp2 y para verificar que los pacientes si fueron ingresados en el setUp2 hacemos uso del método search(), donde le pasaremos algun id de un paciente que ingresamos, esto con el objetivo de que nos devuelva el mismo paciente que ingresamos en el setUp2.

En los tres test le pasamos un id diferente y nos devuelve un paciente diferente dándonos a entender que la funcionalidad search() si funciona y solo busca el paciente con ese identificador.

Test de deleteElement:

deleteElement1() : En este primer test hacemos uso del setUp2, seguidamente y ya con pacientes en hashTable eliminamos un paciente con el metodo delete() y para verificar que si lo eliminamos utilizaremos el metodo search() lo buscaremos y el resultado que esperamos obtener claramente es null.

deleteElement2() : En este segundo test llamaremos al setUp2 para que cargue algunos pacientes en hashTable, luego volvemos hacer exactamente los mismo pasos del primer test, primero eliminamos un paciente con el método delete() de la clase hashTable y luego lo trataremos de buscar con el método search() pero claramente el resultado que nos debería de arrojar es null ya que no encuentra a ningún paciente con ese identificador en hashTable.

deleteElement3() : Por último volvemos a cargar los paciente del setUp2 y acto seguido eliminamos un paciente con el método delete(), por lo tanto cuando hagamos uso del método search() y busquemos el id de ese paciente que acabamos de borrar lo que nos tiene que retornar es un null dándonos a entender que el fue exitosamente borrado de hashTable.

Test de hashKey:

hashKey1(), hashKey2(), hashKey3() : Estos tests funcionan llamando al método hashKey() y pasandole como parametro una identificación aleatoria, este nos tendra que devolver un índice que se ve influencia por la misma funcionalidad del método hashKey. En el primer caso hashKey1() utilizamos el identificador “1234432234” que nos devuelve el índice 1, en el segundo caso hashKey2() utilizamos el identificador “1987654321” que nos devuelve el índice 0, y por último en el test hashKey3() utilizamos el identificador “1912567490” y nos arrojo el índice 4.

STACK TEST

Para el test de la estructura de datos STACK haremos uso de tres escenarios diferentes, los cuales llamaremos: setUp1: Este escenario únicamente inicializará el Stack que estara vacio, setUp2: Este escenario cuenta con 4 pacientes registrados, dad uno con prioridades diferentes y setUp3: Contará únicamente con un paciente registrado. Probaremos los métodos de pop,peek,push,isEmpty y search.

peekTest1(){} En este Test usamos el escenario setUp3, agregamos un paciente con una prioridad mayor a la del paciente ya registrado, y nos debe devolver el paciente que acabamos de registrar

peekTest2(){} En este Test usamos el escenario setUp1, dado que no hay pacientes en este escenario esperamos que al usar el método peel nos devuelva un nulo.

peekTest3(){} E n este Test usamos el escenario setUp2, agregamos un paciente con una mayor prioridad a los otros 4 ya registrados, usamos el método peek y este nos debería devolver el recién ingresado

pushTest1(){}En este Test usamos el escenario setUp1, dado que en este escenario no hay pacientes registrados usamos el metodo push para agregar uno, posteriormente usamos el metodo peek y nos deveeria devolver el recien ingresado.

pushTest2(){} En este Test usamos el escenario setUp1 nuevamente, agregamos 2 nuevos pacientes , el ingresado en el Tes1 y uno con mayor prioridad al anterior, después, utilizamos el método peek que nos retorna el último ingresado dado que tiene mayor prioridad

.

pushTest3(){} En este Test usamos el escenario setUp3, usamos el método push para agregar uno con mayor prioridad al ya registrado por el escenario, y al realizar el método peek nos retorna el ultimo ingresado , mostrando que el método push funciona correctamente

popTest1(){} En este Test usamos el escenario setUp1, al estar vacio el Stack, si realizamos el método pop este nos devuelve nulo

popTest2(){} Para este Test usamos el escenario setUp2, con el método push, agregamos un paciente con mayor prioridad a los ya registrados, y si usamos el método pop, nos devolverá el último ingresado.

popTest3(){} Para este Test usamos el escenario setUp3, con el método push, agregamos un nuevo paciente al ya registrado en el escenario, pero con una prioridad mayor, si usamos el método pop nos retorna el último ingresado gracias a su prioridad

isEmptyTest1(){} En este Test usamos el escenario setUp1, dado que en este escenario no hay ningun paciente registrado, si usamos el método isEmpty, nos retorna true.

isEmptyTest2(){} En este Test usamos el escenario setUp2, usamos el método isEmpty, pero al tener pacientes ya registrados nos debe retornar false

isEmptyTest3(){} En este Test usamos el escenario setUp3, nuevamente probamos con el metodo is Empty y nos debería retornar false

searchTest1(){} En este Test usamos el escenario setUp1, y con el metodo search buscamos un paciente recién creado pero no lo ingresamos,, al escenario estar vacío, nos debe retornar false

searchTest2(){} En este Test usamos el escenario setUp1 nuevamente, pero en este caso si agregamso un nuevo paciente, los buscamos con el método search y nos devolverá true

searchTest3(){} En este test usamos el escenario setUp2, buscamos un paciente que no esté entre los ya registrados por el escenario, y este nos devuelve false

Con estos tests probamos que todos los métodos pertenecientes a la estructura de datos STACK funcionan correctamente

PRIORITYQUEUE TEST

Este test de la clase priorityqueue involucra tres escenarios diferentes los cuales son: setUp1 solo la definición del priorityQueue, setUp2 involucra la definición y la inserción de 3 pacientes diferentes, y por ultimo en el setUp3 aparte de la definicion y tambien agregamos 3 pacientes claramente diferentes al setUp anterior.

insertElement1() : Hacemos uso del setUp1 y le agregamos un paciente, luego para validar que si fue insertado utilizamos la funcionalidad extractMax() que nos devuelve el de mayor prioridad pero como solo hemos agregado uno nos tendria que devolver el mismo paciente que creamos aqui.

insertElement2() : Hacemos uso del setUp1 y hacemos la inserción de un paciente diferente al anterior, para verificar que si fue agregado extraemos el de mayor prioridad pero como solo creamos ese paciente entonces nos lo tiene que devolver.

insertElement3() : Hacemos usos del setUp1 y añadimos un paciente diferentes a los dos anteriores, para concluir que si fue añadido extraemos el paciente que tenga más prioridad pero como solo agregamos ese paciente nos lo tiene que devolver.

extractMax1() : Utilizamos el setUp3 que ya viene con 3 pacientes agregados, lo único que hacemos es utilizar el método extractMax() que nos buscará el paciente de mayor prioridad.

extractMax2() : Utilizamos el setUp2 que ya viene con 3 pacientes agregados, lo único que hacemos es utilizar el método extractMax() que nos buscará el paciente de mayor prioridad en este caso como fue un setUp diferente al anterior nos arrojaría un paciente diferente.

extractMax3() : Utilizamos el setUp1 hacemos la creación e inserción de un paciente y utilizamos el método extractMax() que nos debería devolver el mismo paciente que creamos ya que solo hemos añadido ese.

contPatient1() : Cargaremos los paciente del setUp3 y para validar que el método contPatient() de la clase priorityQueue funcione nos tendría que devolver 3 ya que el setUp3 cuenta con 3 pacientes.

contPatient2() : Cargaremos los paciente del setUp2 y para validar que el método contPatient() de la clase priorityQueue funcione nos tendría que devolver 3 ya que el setUp3 cuenta con 3 pacientes.

contPatient3() : Utilizaremos el setUp1 donde solo está definido el priorityQueue lo que haremos en la creación e inserción de un solo paciente por lo que el método contPatient() nos tendría que devolver 1.

showPatient1() : Usaremos el setUp2(), acto seguido utilizaremos el método showPatients() de la clase priorityQueue, lo que esperamos que suceda es que devuelva una cadena de caracteres con todos los pacientes que están el setUp2.

showPatient2() : Usaremos el setUp3(), acto seguido utilizaremos el método showPatients() de la clase priorityQueue, lo que esperamos que suceda es que devuelva una cadena de caracteres con todos los pacientes que están el setUp3.

showPatient3() : Usaremos el setUp1(), y como este se encuentra vacío creamos e insertamos un paciente y con el método showPatients() solo nos tendrá que mostrar en la priorityQueue ese paciente.

increaseKey1(), increaseKey2(), increaseKey3() : Usamos el setUp1() y creamos e insertamos un paciente, luego utilizaremos el método increaseKey() de la clase priorityQueue le pasaremos un nuevo valor de prioridad, por último imprimimos con el método showPatients() y lo que podemos observar es que la prioridad del único paciente que agregamos cambio por el que metimos en increaKey().

deleteElment1() : Empleamos el setUp1, creamos e insertamos un paciente con el método insertElement() de la clase priorityQueue, posteriormente lo borraremos con la función deleteElement() y para verificar que si fue borrado imprimimos los paciente en la queue, en este caso nos tendría que devolver una cadena vacía ya que solo insertamos uno y lo borramos.

deleteElment2() : Empleamos el setUp1, creamos e insertamos dos pacientes con el método insertElement() de la clase priorityQueue, posteriormente borramos uno de ellos con la función deleteElement() y para verificar que si fue borrado imprimimos los paciente en la queue, en este caso nos tendría que devolver los datos de un solo paciente y obviamente del que no fue borrado.

deleteElment2() : Empleamos el setUp1, creamos e insertamos tres pacientes con el método insertElement() de la clase priorityQueue, posteriormente borramos uno de ellos con la función deleteElement() y para verificar que si fue borrado imprimimos los paciente en la queue, en este caso nos tendría que devolver los datos de dos pacientes y obviamente de los que no fueron borrados.