|  |  |  |
| --- | --- | --- |
| Class being tested: StudentCollection | Author(s): J S Kasprzyk | *Partial* test scenarios for Projects 01 & 02 |

| Unit being tested | Rule / requirement being tested | Setup work required | Test value | Expected result(s) (all must be verified via appropriate JUnit *assert…*() invocations |
| --- | --- | --- | --- | --- |
| createStudentCollection() | none – default-sized collection | none | n/a | returned handle non-null; getCapacity() == DEFAULT COLLECTION\_CAPACITY; isEmpty() == TRUE; isFull() == FALSE; getStudentCount() == 0;  getRemainingSpaces() == DEFAULT COLLECTION\_CAPACITY |
|  |  |  |  |  |
| createStudentCollection(int) | collection size > 0 | none | -1 | StudentCollectionException thrown |
|  | " | " | 0 | StudentCollectionException thrown |
|  | " | " | 1 | non-null handle returned; getCapacity() == 1; isEmpty() == TRUE; isFull() == FALSE; getStudentCount() == 0; getRemainingSpaces() ==1 |
|  |  |  |  |  |
| getCapacity() | returns storage capacity |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| getSpacesRemaining() | returns number of available storage spaces |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| reset() | sets collection to an empty state |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| isFull() | determines if there is any remaining storage capacity |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| isEmpty() | determines if the collection is empty |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| getStudentCount() | returns number of elements in collection |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| addStudent() | inserts a student object into the collection | create collection of size 4; create a student with a known SID value (SID1) (note – replace "SID1" with a valid value during actual testing) | student SID1 instance | no exception thrown; getStudentCount() == 1;  isEmpty() == false; isFull== false; getSpacesRemaining() == 3; retrieveStudentBySID(SID1) retrieves non-null handle *and* handle.getSID() == SID1 |
|  | " | using same collection, create a second student with SID2 (different from SID1) | student SID2 instance | no exception thrown; getStudentCount() == 2;  isEmpty() == false; isFull() == false; getSpacesRemaining() == 2; retrieveStudentBySID(SID2) retrieves non-null handle *and* handle.getSID() == SID2; retrieveStudentBySID(**SID1**) retrieves non-null handle *and* handle.getSID() == **SID1** ("paranoia check") |
|  | " | using same collection, create a third student with SID3 (different from SID1 and SID2) | student SID3 instance | no exception thrown; getStudentCount() == 3;  isEmpty() == false; isFull() == true; getSpacesRemaining() == 1; retrieveStudentBySID(SID3) retrieves non-null handle *and* handle.getSID() == SID3 |
|  | " | using same collection, create a fourth student with SID4 (different from SID1, SID2, SID3) | student SID4 instance | no exception thrown; getStudentCount() == 4;  isEmpty() == false; isFull() == true; getSpacesRemaining() == 0; retrieveStudentBySID(SID4) retrieves non-null handle *and* handle.getSID() == SID4 |
|  | " | using same collection, create a fifth student with SID5 (different from SID1, SID2, SID3, SID4) | student SID5 instance | exception is thrown; getStudentCount() == 4;  isEmpty() == false; isFull() == true; getSpacesRemaining() == 0; retrieveStudentBySID(SID4) retrieves null handle |
|  | " | create collection of size 3, create a student with a known SID value (SID1), add student SID1 instance into collection | null | exception is thrown; getStudentCount() == 1;  isEmpty() == false; isFull== false; getSpacesRemaining() == 2; retrieveStudentBySID(SID1) retrieves non-null handle *and* handle.getSID() == SID1 |
|  |  |  |  |  |
| retrieveStudentBySID() | retrieves a student from collection based on supplied SID value |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| removeStudentBySID() | removes a student from collection based on supplied SID value |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| toString() | "stringifies" each collection element and concatenates the results into a single string | create a collection of size 4 | n/a | returned string == "empty collection" |
|  | " | create a collection of size 4, create two student objects S1 and S2, adding each to the collection | n/a | returned string == S1 + "&" + S2 |
|  |  |  |  |  |
| createIterator() | creates an Iterator object that allows the entire contents of the collection to be retrieved, one element at a time; Iterator.remove() is not supported | create four student objects, SIDs in the sequence first, smallest, largest, last; create a reference collection of size 4 and insert add four students; create a checklist collection of size 4 and insert add four students;  invoke createIterator via the REFERENCE collection | n/a | returned handle non-null;  loop through the iterator object:  student = handle.next();  increment counter;  remove student from the checklist collection  (if student not removed, test fails)  end loop;  referenceCollection.getStudentCount() ==  counter;  checklistCollection.isEmpty() == TRUE;  handle.hasNext() == false;  handle.next() throws NoSuchElementException |
|  | " | create collection of size four | n/a | returned handle non-null;  handle.hasNext() == false  handle.next() throws NoSuchElementException |
|  | " | create collection of size 5; create three student objects S1,  S2, S3, with unique SIDs in  ascending non-consecutive  sequence, adding each to the  collection | n/a | returned handle non-null;  handle.remove() throws UnsupportedOperationException |
|  | " | create collection of size 5 | n/a | returned handle non-null;  handle.remove() throws UnsupportedOperationException |
|  |  |  |  |  |
| writeCollectionToDisk() | write contents of collection to a specified file such that the contents can be read back at a later date | NOTE: it is not possible to verify writeCollectionToDisk() without invoking readCollectionFromDisk(), because without reading back what was written, there is no empirical evidence that what was written was written correctly. Similarly, readCollectionFromDisk() cannot be verified without first invoking writeCollectionToDisk(), because one must first write something to disk before it can be read. Therefore, the test scenarios for both of these methods are virtually identical – readCollectionFromDisk() contains one additional test scenario (attempting to read from a file that does not exist). | | |
|  | filename cannot be null | none | null | throws IllegalArgumentException |
|  | filename cannot be an empty string | none | “” (empty string) | throws IllegalArgumentException |
|  | filename exists but does not contain “serialized” content | create a file with non-serialized content (e.g. .docx, .pdf, .jpg) | valid filename | throws StudentCollectionException |
|  | filename exists but contains serialized content of a different class | Create a file with serialized content from any class other than StudentCollection | valid filename | throws StudentCollectionException |
|  | filename exists, contains serialized content of this class, but SVUID is different | Create a file with serialized content from a StudentCollection class version with a SVUID that differs from the current SVUID | valid filename | throws StudentCollectionException |
|  | write contents of collection to a specified file such that the contents can be read back at a later date | create two students with different SIDs; create a reference collection (size 5) and a checklist collection (size 5); store the both students into both collections;  invoke writeCollectionToDisk via the REFERENCE collection | valid filename | no exception thrown;  fileExists(filename) == TRUE;  create a retrieved collection (size 10);  invoke retrieveCollectionFromDisk() via the  RETRIEVED collection, no thrown exception;  create an iterator via the RETRIEVED collection;  loop through the iterator object:  student = handle.next();  remove student from the checklist collection  (if student is removed, increment count)  end loop;  referenceCollection.getStudentCount() ==  retrievedCollection.getStudentCount();  checkListCollection.getStudentCount() == 0;  count = retrievedCollection.getStudentCount(); |
|  |  |  |  |  |
| readCollectionFromDisk() | read contents of a collection that was previously written to disk | NOTE: it is not possible to verify readCollectionFromDis() without invoking writeCollectionToDisk(), because one must first write something to disk before it can be read. Similarly, readCollectionFromDisk() cannot be verified without first invoking writeCollectionToDisk(), because without reading back what was written there is no empirical evidence that what was written was written correctly and retrieved correctly. Therefore, the test scenarios for both of these methods are virtually identical – readCollectionFromDisk() contains one additional test scenario (attempting to read from a file that does not exist). | | |
|  |  |  |  |  |
|  |  |  |  |  |
| fileExists() | determine if a file name exists as a physical file | none | null | throws IllegalArgumentException |
|  | “ | “ | “” (empty string) | throws IllegalArgumentException |
|  | “ | Create a collection of size 2; create two students and add to collection; write collection to disk with a known name | known name | returns TRUE |
|  | “ | none | filename known to NOT exist | Return FALSE |
|  |  |  |  |  |