|  |  |  |
| --- | --- | --- |
| Class being tested: StudentCollection | Author(s): ZachRich | test scenarios for Project 03 |

| 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 | A collection | n/a | Collection isn’t null  Capacity = 2  Spaces remaining = 2 |
|  |  | Add student to collection | n/a | Instance isn’t null  Capacity = 2  Spaces remaining = 1 |
|  |  |  |  |  |
| getSpacesRemaining() | returns number of available storage spaces |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| reset() | sets collection to an empty state | Create collection size 2 | n/a | isEmpty = true  student count = 0 |
|  |  | Add student | n/a | isEmpty = true  getStudentcount = 0 |
|  |  |  |  |  |
| isFull() | determines if there is any remaining storage capacity | Create collection size 1 | n/a | isEmpty = true  isFull = false |
|  |  | Add student | n/a | isFull = true |
|  |  |  |  |  |
| isEmpty() | determines if the collection is empty | Create collection size 1 | n/a | isEmpty = true  isFull = false |
|  |  | Add Student | n/a | isEmpty = false  isFull = true |
|  |  |  |  |  |
| getStudentCount() | returns number of elements in collection |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| addStudent() | inserts a student object into the collection | create collection of size 4; create student1 with a known SID value (SID1) (note – replace "SID1" with a valid value during actual testing) | student1 instance | no exception thrown; getStudentCount() == 1;  isEmpty() == false; isFull== false; getSpacesRemaining() == 3; retrieveStudentBySID(student1.getSID()) retrieves non-null handle *and* handle.getSID() == student1.getSID() |
|  | " | create collection of size 4; create student1 with a known SID value (SID1); add student1 to collection; create student2 with SID2 (different from SID1) | student2 instance | no exception thrown; getStudentCount() == 2;  isEmpty() == false; isFull() == false; getSpacesRemaining() == 2; retrieveStudentBySID(student2.getSID()) retrieves non-null handle *and* handle.getSID() == student2.getSID(); retrieveStudentBySID(student1.getSID()) retrieves non-null handle *and* handle.getSID() == student1.getSID() ("paranoia check") |
|  | " | create collection of size 4; create student1 with a known SID value (SID1); add student1 to collection; create student2 with SID2 (different from SID1); add student2 to collection; create student3 with SID3 (different from SID1 and SID2) | student3 instance | no exception thrown; getStudentCount() == 3;  isEmpty() == false; isFull() == true; getSpacesRemaining() == 1; retrieveStudentBySID(student3.getSID()) retrieves non-null handle *and* handle.getSID() == student3.getSID() |
|  | " | create collection of size 4; create student1, student2, student3, student4 (unique SIDs); add student1, student2, student3 to collection | student4 instance | no exception thrown; getStudentCount() == 4;  isEmpty() == false; isFull() == true; getSpacesRemaining() == 0; retrieveStudentBySID(student4.getSID()) retrieves non-null handle *and* handle.getSID() == student4.getSID() |
|  | " | create collection of size 4; create student1, student2, student3, student4 (unique SIDs); add student1, student2, student3, student4 to collection; create student5 with SID5 (different from SID1, SID2, SID3, SID4) | student5 instance | exception is thrown; getStudentCount() == 4;  isEmpty() == false; isFull() == true; getSpacesRemaining() == 0; retrieveStudentBySID(student5.getSID()) retrieves null handle |
|  | " | create collection of size 3, create student1 with a known SID value (SID1), add student1 to collection | null | exception is thrown; getStudentCount() == 1;  isEmpty() == false; isFull== false; getSpacesRemaining() == 2; retrieveStudentBySID(student1.getSID()) retrieves non-null handle *and* handle.getSID() == student1.getSID() |
|  |  |  |  |  |
| 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 StudentCollection of size 4 and insert all four students; create a checklist collection (array, ArrayList, List, whatever) of size 4 and insert all 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;  invoke createIterator via the **reference** collection | 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  invoke createIterator via the **reference** collection | n/a | returned handle non-null;  handle.remove() throws UnsupportedOperationException |
|  | " | create collection of size 5  invoke createIterator via the **reference** collection | 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. | | |
|  | filename cannot be null | none | null | throws IllegalArgumentException |
|  | filename cannot be an empty string | none | “” (empty string) | throws IllegalArgumentException |
|  | 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. | | |
|  | filename cannot be null | none | null | throws IllegalArgumentException |
|  | filename cannot be an empty string | none | “” (empty string) | throws IllegalArgumentException |
|  | filename must exist | none | valid filename that does not exist | throws StudentCollectionException |
|  | 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(); |
|  |  |  |  |  |
| 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 |
|  |  |  |  |  |