Test Scenarios for Lab 001

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| method being tested | Rule / requirement being tested | Setup work required | Test value supplied to method being tested | Expected result of executing method |
| createStudentCollection() | Constructor initializes instance of StudentCollection with default capacity | None | None | * getCapacity returns the default capacity * instanceof StudentCollection |
| " | Constructor initializes instance of StudentCollection with provided capacity value | Define a capacity value | (int)capacity | * instanceof StudentCollection * getCapacity returns capacity |
| " | Capacity cannot be negative | Define a negative capacity value | (int)capacity | * throws StudentCollectionException() |
| addStudent() | If collection is not full, Student instance is added to collection | Instantiate new StudentCollection with capacity 10;  Create valid Student instance | Student instance | * isEmpty() returns False * isFull == False * getStudentCount == 1 * getSpacesRemaining == 9 * retrieveStudentBySID() is instance of Student |
| “ | If collection is full, StudentCollectionException is thrown | Instantiate StudentCollection of capacity 10;  Create 10 valid instances of Student  Add the 10 students | 10 Instances of Student | * isFull() == True * isEmpty() = False * getStudentCount() == 10 * getSpacesRemaining == 0 |
| createIterator() | Creates an iterator object. Contractual obligations of hasNext() and next() have been implemented. | Create a StudentCollection instance using the no argument constructor | n/a | * Returned object is non-null and is of Iterator type * hasNext() returns false * next() throws NoSuchElementException |
| " | “ | Create a StudentCollection with capacity of 4.  Instantiate a Student object.  Add Student object to StudentCollection | n/a | * hasNext() returns true * next() returns Student object * hasNext() returns false 3 times. * next() throws NoSuchElementException |
| " | " | Create a StudentCollection with capacity of 4.  Instantiate 4 Student objects.  Add Student objects to StudentCollection | n/a | * hasNext() returns true * next() returns first Student object * hasNext() returns true * next() returns second Student object * hasNext() returns true * next() returns third Student object * hasNext() returns true * next() returns fourth Student object |
| getCapacity() | Return the storage capacity of a StudentCollection object with default capacity | Instantiate a StudentCollection object with default capacity | n/a | * isEmpty() returns true * getStudentsRemaining() == DEFAULT\_CAPACITY |
| " | Return the storage capacity of a StudentCollection object with declared capacity | Instantiate a StudentCollection object with a pre-determined capacity  determine capacity | capacity | * isEmpty() returns true * getStudentsRemaining() == capacity |
| getSpacesRemaining() | Return the number of unoccupied storage units in a StudentCollection object of default capacity | Instantiate a StudentCollection object with default capacity | n/a | * isEmpty() returns true. * getSpacesRemaining() == getCapacity() |
| “ | Return the number of unoccupied storage units in a StudentCollection object of default capacity | Instantiate a Student object;  Instantiate a StudentCollection object with default capacity.  Add Student object to StudentCollection | StudentObject | * getSpacesRemaining() == (getCapacity() - getStudentCount()) |
| getStudentCount() | Return the number of students contained in the StudentCollection | Instantiate a StudentCollection object using the no argument constructor | n/a | * getStudentCount == 0 |
| " | “ | Instantiate a StudentCollection object using the no argument constructor  Create an instance of Student  Add student to collection | n/a | * getStudentCount() == 1 |
| isEmpty() | Returns true if the collection contains no elements | Create an instance of StudentCollection using the no-argument constructor | n/a | * isEmpty() == true |
| " | Returns false if collection contains elements | Create an instance of StudentCollection using the no-argument constructor  Create an instance of Student  Add student instance to the collection | n/a | * isEmpty() == false |
| isFull() | Returns true if collection holds maximum Student capacity | Create an instance of StudentCollection of size 5. | n/a | * isFull() == false |
| " |  | Create an instance of StudentCollection of size 5.  Create 4 instances of Student  Add Student objects to the collection | n/a | * isFull() == false |
| " | “ | Create an instance of StudentCollection of size 5.  Create 5 instances of Student  Add Student objects to the collection | n/a | * isFull() == true |
| removeStudentBySID(String sidKey) | If the collection contains a Student instance with the provided sid key, the Student is removed from the collection and used as the return valuie | Create an instance of StudentCollection of size 5  Create an instance of Student with a valid sid key and add it to the collection | The Student instance’s sid key String | * The return value is of Student type * Returned Student’s sid key String is equal to the original Student instance’s sid String * isEmpty() == true |
| “ | “ | Create an instance of StudentCollection of size 5  Create two instances of Student with two valid sid keys  Add both instances to the collection | The sid key String of one of the two Student instances | * The return value is of Student type * Returned Student’s sid key String is equal to that of the selected Student instance * GetStudentCount == 1 |
| “ | If sid String does not exist within the collection, an exception is thrown | Create an instance of StudentCollection of size 5  Create two instances of Student with two valid sid keys  Add both instances to the collection | Sid key that is different from that of the Student instances within the collection | * StudentCollectionException is thrown |
| reset() | Removes all instances of Student within the collection | Create an instance of StudentCollection of size 5  Create 5 instances of Student and add them to the collection | n/a | * getStudentCount() == 0 * isEmpty() == true |
| retrieveStudentBySID(String sidKey) | Method returns instance of Student with the given sid key String. | Create an instance of StudentCollection using the no argument constructor  Create an instance of Student with a valid sid key String and add it to the collection | Student instance’s sid key String | * Return value is not null and is of type Student * Returned instance sid key String is equal to the one which was provided * getStudentCount == 1 |
| “ | If the collection does not contain a Student instance with the provided sid key String, throw an exception | Create an instance of StudentCollection using the no argument constructor.  Create an instance of Student with a valid sid key String and add it to the collection | Any valid sid key String | * StudentCollectionException is thrown |
| " | “ | Create an instance of StudentCollection using the no argument constructor.  Create an Instance of Student with a valid sid key String and add it to the collection | Any valid sid key that is NOT that of the newly added instance of Student | * StudentCollectionException is thrown |
| toString() | Method returns a String representation of all Student objects in the collection | Create an instance of StudentCollection with capacity 5  Create 5 instances of Student and add them to the collection | n/a | * Return String contains the String representation of the 5 provided Student instances |
| " | “ | Create an instance of StudentCollection with capacity 5 | n/a | * Returns an empty String |

(You can insert a block of rows into the table by highlighting the *n* rows of the table just *above* the desired insertion point, then right-clicking and choosing Insert → Insert rows *below*; to delete a block of rows, highlight the block, then right-click and choose Delete Cells → Delete Entire Row.)

Note: it’s generally considered “best practice” that “Expected result of executing unit” be quite specific when designing test scenarios *for black box testing*. The test scenarios are being designed without knowledge of the underlying implementation, and the level of detail provided for “expected results” will be most influential and useful when test scenarios are designed when they *should* be designed (***before*** implementation begins, not *after* implementation has been *completed*…).