Vaishnavi Manthena and Sanjana Pendharkar (Group 19)

Dr. Chang

Project 1: Test Design Document

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Purpose of test case | Input data | Expected output |
| 1 | This test case should ensure that the add function can do the basic operation of add, which involves including another GroceryItem item within the shopping bag by increasing the bag’s size and storing the data of this item and keeping the bag’s capacity at its starting value of 5. After adding the items, this case would be tested by printing the size, capacity and all the Grocery Items. | "pineapples", 2.99, true  "olives", 2.12, false  "red\_bell\_peppers", 4.53, true  "pizza\_dough", 12.10, false | There are 4 items in your bag.  This is the capacity of your bag: 5.  pineapples: $2.99 : is taxable  olives: $2.12 : tax free  red\_bell\_peppers: $4.53 : is taxable  pizza\_dough: $12.10 : tax free |
| 2 | This test case should ensure that the add function correctly adds to additional items and correctly grows and updates the capacity of our grocery bag. In this case, since we have 6 GroceryItem items in our shopping bag, our capacity should increase to 10 and our size should be 6. After adding the items and printing them out using our toString() method, this case would be tested by printing the size, capacity and all the GroceryItem items. | "sauce", 2.92, false  "cheese", 4.58, true | There are 6 items in your bag.  This is the capacity of your bag: 10.  pineapples: $2.99 : is taxable  olives: $2.12 : tax free  red\_bell\_peppers: $4.53 : is taxable  pizza\_dough: $12.10 : tax free  sauce: $2.92 : tax free  cheese: $4.58 : is taxable  3 |
| 3 | This test case should ensure that given a ShoppingBag, the SaleTax method should correctly calculate the sales tax by considering only the taxable items. To implement this case, I calculated the correct sales tax by multiplying 0.06625 with the sum of the prices of taxable items. Then I compared this value with the value returned by the SalesTax method | Correct sales tax = 0.80  If shoppingbag.salesTax() = correct sales tax:  “Correct sales tax: 0.80!”  Else:  “Incorrect sales tax. The sales tax should be 0.80” | “Correct sales tax: 0.80!” |
| 4 | This test case is used to check the remove() method, which deletes a GroceryItem item from the shopping bag after checking if it is included within the shopping bag.   * Case 1: GroceryItem item belongs to the shopping bag, and can be removed. The size of the bag decreases by 1, and the capacity does not change. * Case 2: We test the same GroceryItem item to check if it can be removed. In this case, our remove method should not be able to find the index of the item and return false. | Say we initialize a GroceryItem item6, which is in the shopping bag.  Case 1:  If shoppingbag.remove(item6):  “Test passed: remove method returned true because item is in bag.”  Case 2:  If shoppingbag.remove(item6) == false:  “Test passed: remove method returned false because item is no longer in bag.” | * Case 1 returns “Test passed: remove method returned true because item is in bag.” * Case 2 returns “Test passed: remove method returned false because item is no longer in bag.” |
| 5 | This test case is used to check if the remove() method can operate properly if given an item that is not originally included in the shopping bag. In this case, the function should return false, as the new item that does not belong to the shopping bag does not exist in the shopping bag, and therefore, cannot be removed from the bag itself. | Say we initialize a GroceryItem notincluded, which is not in the shopping bag.  If shoppingbag.remove(notincluded) == false:  “Test passed: remove method returned false because item not in bag.” | Returns:  “Test passed: remove method returned false because item not in bag.” |
| 6 | These test cases are designed to check if the remove() method works for items that are at various points within the array. All of the items we are testing for these methods have already been added to our shopping bag.   * Case 1: GroceryItem item is the last element of the bag. * Case 2: GroceryItem item is a middle element in the bag. * Case 3: GroceryItem item is the first element in the bag. | Case 1: Item6 is the last item in the bag.  if shoppingbag.remove(item6):  “Test passed: remove method returned true because item is in bag.”  Case 2: Item2 is a middle item in the bag.  If shoppingbag.remove(item2):  “Test passed: remove method returned true because item is in bag.”  Case 3: Item1 is the first item in the bag.  If shoppingbag.remove(item1):  “Test passed: remove method returned true because item is in bag.” | Returns:   * Case 1: “Test passed: remove method returned true because item is in bag.” * Case 2: “Test passed: remove method returned true because item is in bag.” * Case 3: “Test passed: remove method returned true because item is in bag.” |
| 7 | This test case is designed to ensure that our grow method, when called, keeps the size of the bag constant and adds 5 to the capacity of the bag. | Currently the size of the shopping bag is 3 and the capacity is 10.  shoppingbag.grow();  Print out the following:   * “There are *shoppingbag.size* items in your bag.” * “This is the capacity of your shoppingbag : *shoppingbag.capacity*.” | Returns:   * “There are *3* items in your bag.” * “This is the capacity of your shoppingbag : *15*.” |
| 8 | This case would ensure that after all the removing and adding manipulations done before, the ShoppingBag has the right elements in the right order. Before printing, we are also displaying the size to make sure we have the correct number of elements | No input. We will simply print out the size and string versions of grocery items in the bag. | There are 3 items in your bag.  pizza\_dough: $12.10 : tax free  sauce: $2.92 : tax free  red\_bell\_peppers: $4.53 : is taxable |