**Homework Assignment 5**

Due date: October 16th, 11:55pm EST

**Problem 1**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **operation** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **return value** |
| queue.insert(15) | 15 |  |  |  |  |  |  |  | none |
| queue.insert(3) | 15 | 3 |  |  |  |  |  |  | none |
| queue.insert(-15) | 15 | 3 | -15 |  |  |  |  |  | none |
| queue.insert(35) | 15 | 3 | -15 | 35 |  |  |  |  | none |
| queue.remove() |  | 3 | -15 | 35 |  |  |  |  | 15 |
| queue.remove() |  |  | -15 | 35 |  |  |  |  | 3 |
| queue.remove() |  |  |  | 35 |  |  |  |  | -15 |
| queue.insert(13) | 13 |  |  | 35 |  |  |  |  | none |
| queue.remove() | 13 |  |  |  |  |  |  |  | 35 |
| queue.remove() |  |  |  |  |  |  |  |  | 13 |
| queue.remove() |  |  |  |  |  |  |  |  | null |
| queue.insert(3) |  | 3 |  |  |  |  |  |  | none |

**Problem 2**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **operation** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **return value** |
| stack.push('c') | c |  |  |  |  |  |  |  | none |
| stack.push( new Character('s') ) | c | s |  |  |  |  |  |  | none |
| stack.pop(); | c |  |  |  |  |  |  |  | s |
| char p = 's';  stack.push( p ); | c | s |  |  |  |  |  |  | none |
| stack.push( p ); | c | s | s |  |  |  |  |  | none |
| stack.push ( new Character( '1' ) ); | c | s | s | 1 |  |  |  |  | none |
| stack.peek(); | c | s | s | 1 |  |  |  |  | 1 |
| stack.pop(); | c | s | s |  |  |  |  |  | 1 |
| stack.push( '%' ); | c | s | s | % |  |  |  |  | none |
| stack.peek(); | c | s | s | % |  |  |  |  | % |
| stack.push('A'); | c | s | s | % | A |  |  |  | none |
| stack.push('X'); | c | s | s | % | A | X |  |  | none |
| stack.pop(); | c | s | s | % | A |  |  |  | X |
| stack.pop(); | c | s | s | % |  |  |  |  | A |

**Problem 3**

|  |
| --- |
| **private** <E> **void** orderedInsert (E item)  {  // Please note: This code is based off of the code shown before in the GenericLinkedList Class  // Validation check: Checks to see that the node's data is not null  **if** (item != **null**)  {  // Creates a new node that will be inserted, with data initialized to item, and  // newNode.getNext() initialized to null for now  GenericNode<E> newNode = **new** GenericNode<E>(item, **null**);    // Corner Case: Checks for an empty list  **if** (head == **null**)  head = newNode;  **else**  {  // Creates the current reference (starts at the first node)  GenericNode<E> current = head;    // Advances the current reference until the data value is more than  // the newNode’s data value  **while** (current.getNext() != **null** && current.getData().compareTo(newNode.getData()) < 0)  {current = current.getNext();} // Ends the advancement    // Sets the newNode's next node to the current nextNode  newNode.setNext(current.getNext());    // Set's the current node's next node to the newNode  current.setNext(newNode);  } // End of the insertion  } // End of the overall null checking  } // End of orderedInsert method |