

CIS1101 – DATA STRUCTURES WITH JAVA OBJECTS

ASSIGNMENT

DOCUMENTATION

|  |  |  |
| --- | --- | --- |
| **NAME AND SURNAME OF AUTHORS** | **ID.NUMBER** | **DATA STRUCTURE DONE** |
| **MATTHEW FORMOSA** | **279095M** | **BINARY SEARCH TREE** |
| **ANDREA SANT** | **404995M** | **CIRCULAR QUEUE** |

Contents

[**1** **Specification** 1](#_Toc377657583)

[1.1 *AnyClass* class 1](#_Toc377657584)

[1.2 Classes inheriting from the abstract *AnyClass* class 1](#_Toc377657585)

[1.3 Linear and Binary Nodes 2](#_Toc377657586)

[**1.3.1** **Linear Node – Class Node** 2](#_Toc377657587)

[**1.3.2** **Binary Node – class BNode** 2](#_Toc377657588)

[1.4 Heterogeneous Circular Queue - Class *CQueue* 2](#_Toc377657589)

[1.5 Generic Binary Search Tree(BST) – Class *BinSearchTree* 3](#_Toc377657590)

[**2** **Testing Approaches and Results** 4](#_Toc377657591)

[2.1 Object Class: CQueue 4](#_Toc377657592)

[**2.1.1** **Method: public boolean put(AnyClass iObject)** 4](#_Toc377657593)

[**2.1.2** **Method: public AnyClass editObject(String key)** 5](#_Toc377657594)

[**2.1.3** **Method: public void listAll()** 6](#_Toc377657595)

[2.2 Object Class: BinSearchTree 7](#_Toc377657596)

[**2.2.1** **Method: public void populateFromQueue(CQueue que)** 7](#_Toc377657597)

[**2.2.2** **Method: public void listInOrder()** 8](#_Toc377657598)

[**2.2.3** **Method: public AnyClass search(String key)** 9](#_Toc377657599)

[2.3 Object Class: GCQueue 10](#_Toc377657600)

[**2.3.1** **Method: public boolean put(T iObject)** 10](#_Toc377657601)

[**2.3.2** **Method: public T editObject(T key)** 11](#_Toc377657602)

[**2.3.3** **Method: public void listAll()** 12](#_Toc377657603)

[2.4 Object Class: GBinSearchTree 13](#_Toc377657604)

[**2.4.1** **Method: public void populateFromQueue(GCQueue<T> que)** 13](#_Toc377657605)

[**2.4.2** **Method: public void listInOrder()** 14](#_Toc377657606)

[**2.4.3** **Method: public T search(T key)** 15](#_Toc377657607)

[**3** **List of Figures** 16](#_Toc377657608)

1. **Specification**
   1. *AnyClass* class

Both circular queue and binary search tree data structures have to store class *AnyClass* as an abstract class stored in the package *dataobjects*. The class has a single data field *int seqNo* with the corresponding setter and getter methods.

Apart from the constructor the class is to have three polymorphic (virtual) methods:

* Polymorphic method String *getData()* to return data of the object in form of the string. In case of the *AnyClass* it returns value of the *seqNo*
* Polymorphic method String *getKey()* returning a *key* value in form of the *String* type. The *key* will be used for searching of the object. In case of the *AnyClass* it returns value of the **seqNo** as a *String*
* Polymorphic method *void edit()* to edit object data by entering the new values from the *key*board. In case of the *AnyClass* it is “empty” as the only data field *seqNo* is not supposed to be edited.

Any actual object to be stored must extend from the *AnyClass* class.

* 1. Classes inheriting from the abstract *AnyClass* class

Both below specified dynamic data structures (see items 4 and 5 below) will store mixture of objects of the classes *Employee* and *PartTimer* developed during practical sessions.

The *key* used for operations of searching and editing will be **surname.**

Editable value will only be value of **pay.**

Both classes have to override the polymorphic methods from the class *AnyClass.*

* 1. Linear and Binary Nodes
     1. **Linear Node – Class Node**

Used by the circular queue. Its data part is object of the class *AnyClass*. Apart from the constructor it has no methods. The class Node is to be placed in the package *linearnodes.*

* + 1. **Binary Node – class BNode**

Used by the Binary Search Tree. Its data part is object of the class *AnyClass*. Apart from the constructor it has no methods. The class BNode is to be placed in the package *binarynodes*.

* 1. Heterogeneous Circular Queue - Class *CQueue*

The *CQueue* class is to reside in the package *linearstructures* and has to import both the *dataobjects* and *linearnodes* packages. The queue is a circular queue offering the following operations:

1. Construction of a CQueue object with the size of 20 nodes
2. Placing new object to the queue returning success only if the queue was not full: method *Boolean put(AnyClass newObj)*
3. Serving the front object by removing it from the queue and returning object of *AnyClass* or null if the queue was empty: method *AnyClass* s*erve()*
4. Listing all objects currently on the queue while showing data of each object: method *void listAll()*
5. Editing of a particular object identified by the *key*: method *AnyClass editObject* *(String key)*

The method traverses the queue to find the object to be edited. If found then returns a reference to the object just edited otherwise null.

* 1. Generic Binary Search Tree(BST) – Class *BinSearchTree*

Is to reside in the package *non\_linearstructures* and has to import both the *dataobjects* and *binarynodes* packages. The Binary Search Tree is a tree structure offering the following operations:

1. Adding new object to the tree keeping it sorted in order of a the *key* (with its value returned by the *getKey()* method): method void insert (*AnyClass newObj*)
2. Searching for the object by the *key* value returning reference to the object if found otherwise null: method *AnyClass* *search (String key)*
3. Listing of all objects on the tree in ascending order of the *key* while showing data of each object: method *void listInOrder()*
4. Population of a BST from the queue: The method traverses the queue (implemented in item 4 above) and populates the BST by binary nodes containing objects stored in the queue nodes by calling the queue’s *serve()* method. This process empties the current contents of the queue. Note that this method actually sorts the objects in ascending order of the *key* values: method *void populateFromQueue(CQueue que)* where *CQueue* is a class name of the circular queue.
5. **Testing Approaches and Results**
   1. Object Class: CQueue
      1. **Method: public boolean put(AnyClass iObject)**

Approach: Once the user clicks on the “Add Record To CQueue” button, (s)he will be prompted to choose between creating a new employee or part-timer (Figures 1, 2 & 3). This will be repeated 21 times to ensure the validity of this method (Figures 4, 5 & 8). A respective method (*public void listAll()*) to output the contents of the Circular Queue data structure will be used to show that all data is entered and stored as expected (Figures 6 & 7).

Expected Result: For up to 20 workers a message stating that the worker was successfully added to the Circular Queue data structure should be displayed. In the case that the user tries to enter more than 20 workers, a message stating that the Circular Queue data structure is full should be displayed.

Actual Result: The first 20 workers were added successfully, yet any other attempt to populate the Circular Queue data structure with another worker was unsuccessful.

Pass/Fail: PASS

* + 1. **Method: public AnyClass editObject(String key)**

Approach: Once the user clicks on the “Edit Record In CQueue” button, (s)he will be prompted to enter the surname of the worker whose pay rate is to be edited (Figures 9 & 18). If the worker is found, the user will be asked to confirm that the found worker’s pay rate is to be edited (Figures 10 & Figure 11). The user is then prompted to enter the new pay rate (Figures 12, 15 & 17). This will be repeated three times. First, a valid pay rate will be entered. After which, an invalid pay rate will be entered. Finally, a non-numeric input will be entered.

Expected Result: If the worker is found and a valid pay rate is entered, a message stating that the worker’s pay rate was successfully edited followed by the updated list of workers within the Circular Queue data structure should be displayed. On the other hand, if the worker is found and an invalid pay rate or a non-numeric input is entered, an error message should be displayed. Otherwise, if the worker is not found, an error message informing the user that the object was not found should be displayed.

Actual Result: In the case of a valid search and a valid pay rate entered, a message stating that the worker’s pay rate was successfully edited (Figure 13) followed by the updated list of workers within the Circular Queue data structure (Figure 14) was displayed. An error message was displayed in the case of a valid search yet an invalid pay rate or non-numeric input entered (Figure 16). Finally, an error message in the case of an invalid search was displayed (Figure 19).

Pass/Fail: PASS

* + 1. **Method: public void listAll()**

Approach: The user clicks on the “List Records From CQueue” button before adding any worker to the Circular Queue data structure. Once two workers are added, the user clicks on the button once again (Figures 21 & 22).

Expected Result: If the user clicks on the “List Records From CQueue” button prior to adding any workers to the Circular Queue data structure, then an error message stating that the Circular Queue data structure is empty should be displayed. After adding workers to the Circular Queue data structure, clicking the button should return a list of the workers in the order in which they were entered.

Actual Result: An error stating that the Circular Queue data structure is empty was displayed when the button was clicked before actually adding any workers (Figure 20). Otherwise, an updated list of workers in the order in which they were added was displayed (Figure 23).

Pass/Fail: PASS

* 1. Object Class: BinSearchTree
     1. **Method: public void populateFromQueue(CQueue que)**

Approach: The user clicks on the “Populate BST From CQueue” button prior to adding any worker to the Circular Queue data structure. Once three workers are added (Figures 25, 26 & 27), the user clicks on the button once again. A respective method (*public void listInOrder()*) to output the contents of the Binary Search Tree data structure will be used to show that all data is entered and stored as expected (Figure 29).

Expected Result: If the user clicks on the “Populate BST From CQueue” button prior to adding any workers to the Circular Queue data structure, then an error message stating that the Circular Queue data structure is empty should be displayed. Otherwise, the workers should be successfully populated in the Binary Search Tree data structure from the Circular Queue data.

Actual Result: An error stating that the Circular Queue data structure is empty was displayed when the button was clicked before actually adding any workers (Figure 24). After three workers were added and the button was clicked again, a message stating that the Binary Search Tree data structure was successfully populated from the Circular Queue data structure (Figure 28).

Pass/Fail: PASS

* + 1. **Method: public void listInOrder()**

Approach: The user clicks on the “List Records From BST” button before populating the Binary Search Tree data structure from the Circular Queue data structure. Once the Binary Search Tree data structure is populated with three workers (Figures 31, 32 & 33), the user clicks on the button once again.

Expected Result: If the user clicks on the “List Records From BST” button prior to populating the Binary Search Tree data structure from the Circular Queue data structure, then an error message stating that the Binary Search Tree data structure is empty should be displayed. After the Binary Search Tree data structure is populated, clicking the button should return a list of the workers in ascending order of their surname.

Actual Result: An error stating that the Binary Search Tree data structure is empty was displayed when the button was clicked before actually populating it (Figure 30). Otherwise, an updated list of workers in ascending order of their surname was displayed (Figure 34).

Pass/Fail: PASS

* + 1. **Method: public AnyClass search(String key)**

Approach: The user clicks on the “Search Record(s) In BST” button once the Binary Search Tree data structure is populated with three workers (Figures 35, 36 & 37), and is then prompted to enter the surname of the worker to be searched (Figures 38, 40 & 43).

Expected Result: If a match is found for the given search, then the details of the worker should be displayed. Furthermore, if there are other matches for the given search, the user should be informed and asked whether or not (s)he would like to see the whole list of matches. Otherwise, if the search fails, the user should be informed that the worker was not found.

Actual Result: In the case that the search was successful, the details of the worker were displayed (Figure 39 & 42). Furthermore, the user was informed when there were other matches for a given search and was prompted to choose whether or not to view the whole list of matches or not (Figure 41). Otherwise, when the search failed, an error was displayed stating that the worker was not found (Figure 44).

Pass/Fail: PASS

* 1. Object Class: GCQueue
     1. **Method: public boolean put(T iObject)**

Approach: Once the user clicks on the “Add Record To GCQueue” button, ten objects containing random data will be added to the Generic Circular Queue data structure. This will be repeated three times to ensure the validity of this method. A respective method (*public void listAll()*) to output the contents of the Generic Circular Queue data structure will be used to show that all data is entered and stored as expected (Figure 46).

Expected Result: For up to 20 objects, a message stating that the object was successfully added to the Generic Circular Queue data structure should be displayed. In the case that the user tries to enter more than 20 objects, a message stating that the Generic Circular Queue data structure is full should be displayed.

Actual Result: The first 20 objects were added successfully (Figure 45), yet any other attempt to populate the Generic Circular Queue data structure with another object was unsuccessful (Figure 47).

Pass/Fail: PASS

* + 1. **Method: public T editObject(T key)**

Approach: Once the user clicks on the “Edit Record In GCQueue” button, (s)he will be prompted to enter the key field of the object to be edited (Figures 48, 55 & 57). If the object is found and has editable fields, the user will be asked to confirm that the found object is to be edited (Figure 49). The user is then prompted to enter the new value for the field if the Object has editable fields (Figures 50 & 53).

Expected Result: If the object is found and a valid new value for the respective field is entered, a message stating that the object was successfully edited followed by the updated list of objects within the Generic Circular Queue data structure should be displayed. If an invalid value for the respective field is entered, an error should be displayed. On the other hand, if the object is found and it has no editable fields, then an error should be displayed stating that the object has no editable fields. Otherwise, if the object is not found, an error message informing the user that the object was not found should be displayed.

Actual Result: In the case of a valid search and a valid new value for the respective field was entered, a message stating that the object was successfully edited (Figure 51) followed by the updated list of workers within the Generic Circular Queue data structure (Figure 52) was displayed. When an invalid value for the respective field was entered, an error was displayed (Figure 54). An error message was displayed in the case of a valid search yet the object had no editable fields (Figure 56). Finally, an error message in the case of an invalid search was displayed (Figure 58).

Pass/Fail: PASS

* + 1. **Method: public void listAll()**

Approach: The user clicks on the “List Records From GCQueue” button before adding any object to the Generic Circular Queue data structure. Once an object is added, the user clicks on the button once again.

Expected Result: If the user clicks on the “List Records From GCQueue” button prior to adding any objects to the Generic Circular Queue data structure, then an error message stating that the Generic Circular Queue data structure is empty should be displayed. After adding objects to the Generic Circular Queue data structure, clicking the button should return a list of the objects in the order in which they were entered.

Actual Result: An error stating that the Generic Circular Queue data structure is empty was displayed when the button was clicked before actually adding any objects (Figure 59). Otherwise, an updated list of objects in the order in which they were added was displayed (Figure 60).

Pass/Fail: PASS

* 1. Object Class: GBinSearchTree
     1. **Method: public void populateFromQueue(GCQueue<T> que)**

Approach: The user clicks on the “Populate BST From GCQueue” button prior to adding any objects to the Generic Circular Queue data structure. Once ten objects are added, the user clicks on the button once again. A respective method (*public void listInOrder()*) to output the contents of the Generic Binary Search Tree data structure will be used to show that all data is entered and stored as expected (Figure 63).

Expected Result: If the user clicks on the “Populate BST From GCQueue” button prior to adding any objects to the Generic Circular Queue data structure, then an error message stating that the Generic Circular Queue data structure is empty should be displayed. Otherwise, the objects should be successfully populated in the Generic Binary Search Tree data structure from the Generic Circular Queue data.

Actual Result: An error stating that the Generic Circular Queue data structure is empty was displayed when the button was clicked before actually adding any objects (Figure 61). After ten objects were added and the button was clicked again, a message stating that the Generic Binary Search Tree data structure was successfully populated from the Generic Circular Queue data structure (Figure 62).

Pass/Fail: PASS

* + 1. **Method: public void listInOrder()**

Approach: The user clicks on the “List Records From GBST” button before populating the Generic Binary Search Tree data structure from the Generic Circular Queue data structure. Once the Generic Binary Search Tree data structure is populated with ten objects, the user clicks on the button once again.

Expected Result: If the user clicks on the “List Records From GBST” button prior to populating the Generic Binary Search Tree data structure from the Generic Circular Queue data structure, then an error message stating that the Generic Binary Search Tree data structure is empty should be displayed. After the Generic Binary Search Tree data structure is populated, clicking the button should return a list of the objects in ascending order of their key field.

Actual Result: An error stating that the Generic Binary Search Tree data structure is empty was displayed when the button was clicked before actually populating it (Figure 64). Otherwise, an updated list of objects in ascending order of their key field was displayed (Figure 65).

Pass/Fail: PASS

* + 1. **Method: public T search(T key)**

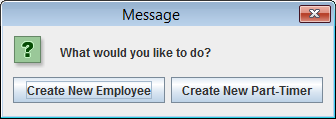
Approach: The user clicks on the “Search Record(s) In GBST” button once the Generic Binary Search Tree data structure is populated with ten objects (Figures 66) and is then prompted to enter the key field of the object to be searched (Figures 67, 69 & 72).

Expected Result: If a match is found for the given search, then the details of the object should be displayed. Furthermore, if there are other matches for the given search, the user should be informed and asked whether or not (s)he would like to see the whole list of matches. Otherwise, if the search fails, the user should be informed that the object was not found.

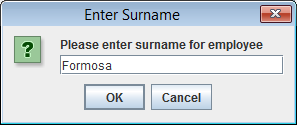
Actual Result: In the case that the search was successful, the details of the object were displayed (Figure 68 & 71). Furthermore, the user was informed when there were other matches for a given search and was prompted to choose whether or not to view the whole list of matches or not (Figure 70). Otherwise, when the search failed, an error was displayed stating that the worker was not found (Figure 73).

Pass/Fail: PASS

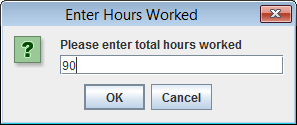
1. **List of Figures**



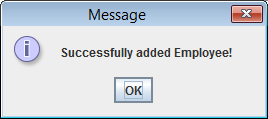
Figure



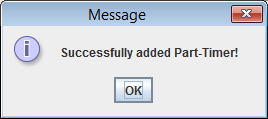
Figure



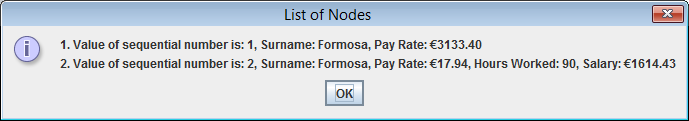
Figure



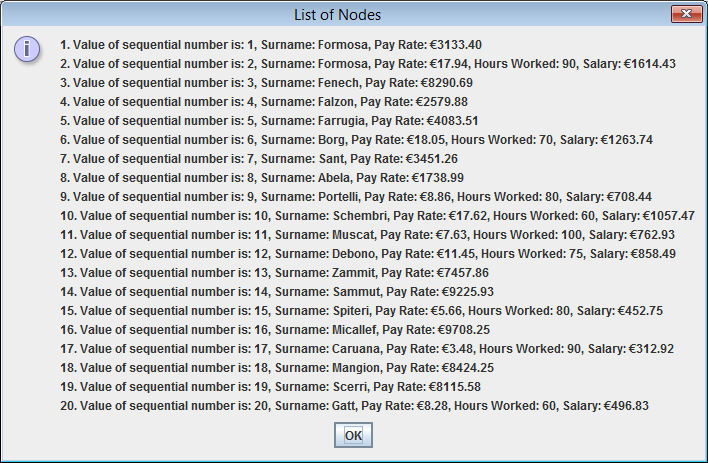
Figure



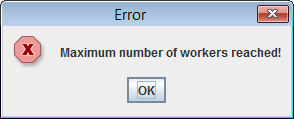
Figure



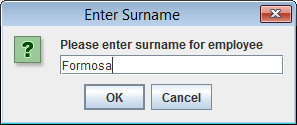
Figure



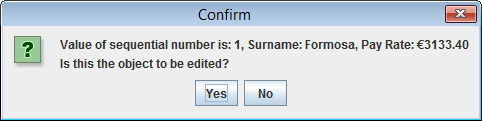
Figure



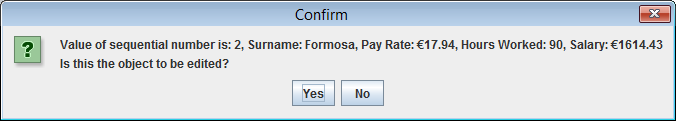
Figure



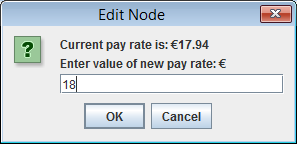
Figure



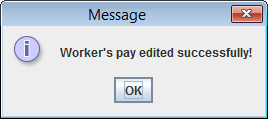
Figure



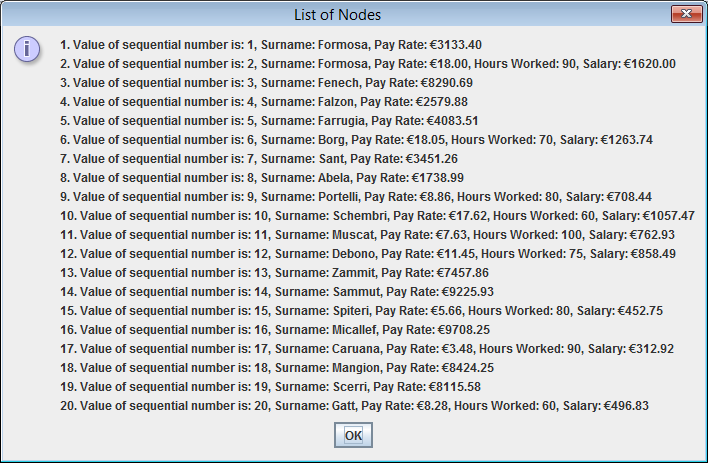
Figure



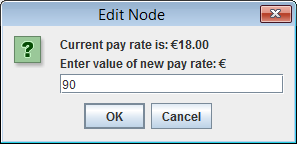
Figure



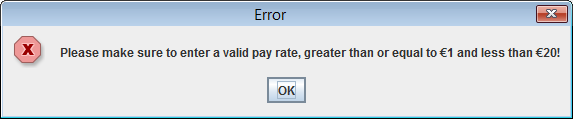
Figure



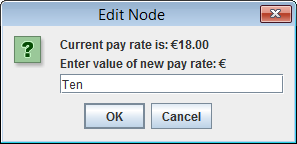
Figure



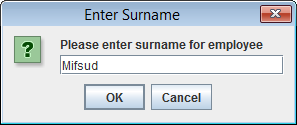
Figure



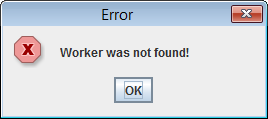
Figure



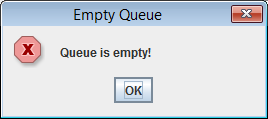
Figure



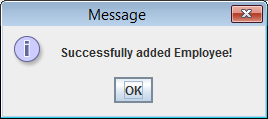
Figure



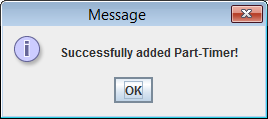
Figure



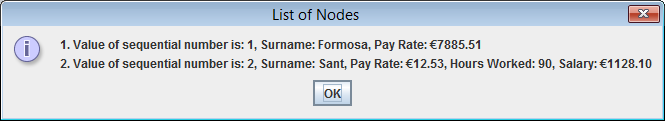
Figure



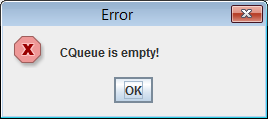
Figure



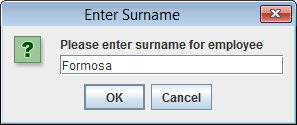
Figure



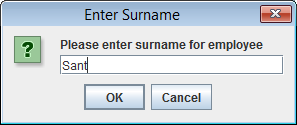
Figure



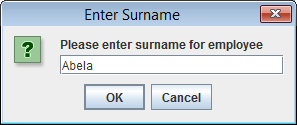
Figure



Figure



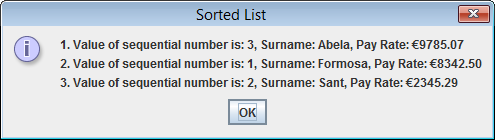
Figure



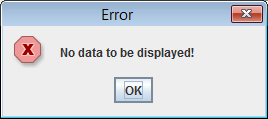
Figure



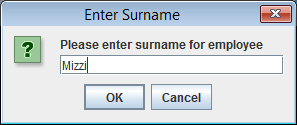
Figure



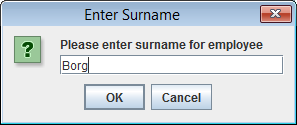
Figure



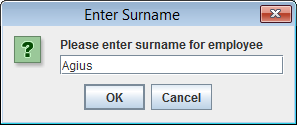
Figure



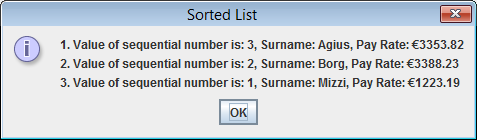
Figure



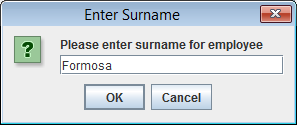
Figure



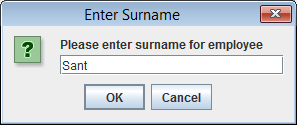
Figure



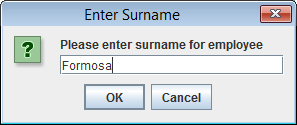
Figure



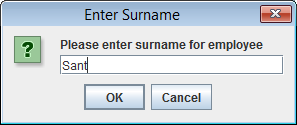
Figure



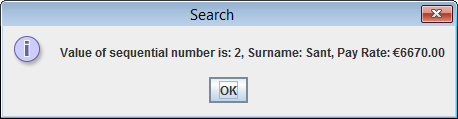
Figure



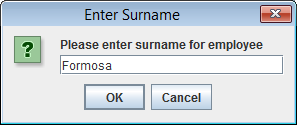
Figure



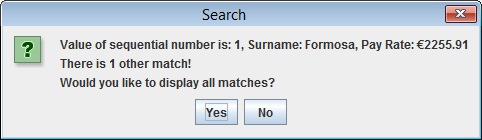
Figure



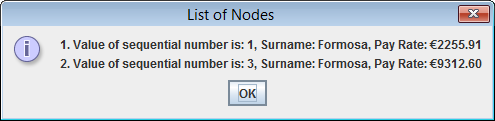
Figure



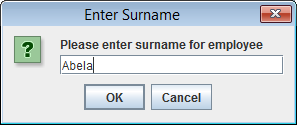
Figure



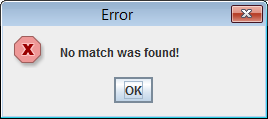
Figure



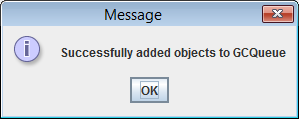
Figure



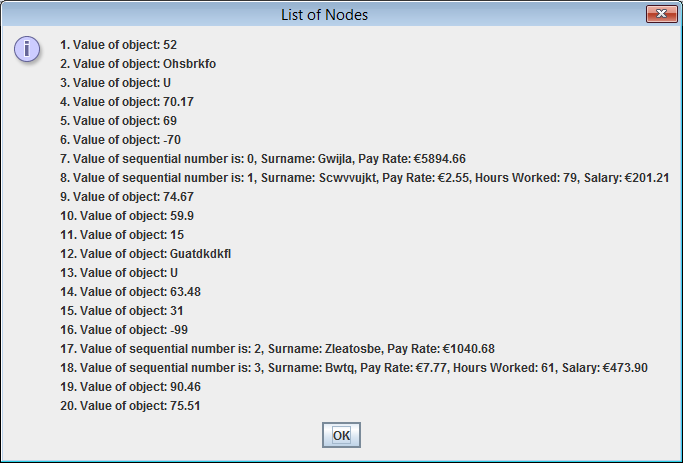
Figure



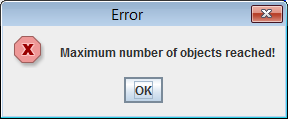
Figure



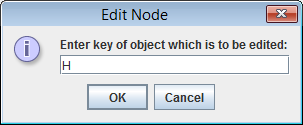
Figure



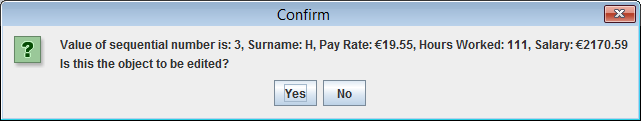
Figure



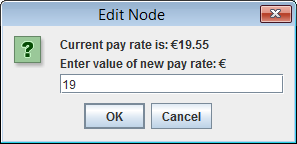
Figure



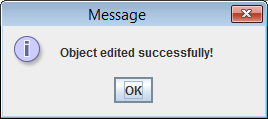
Figure



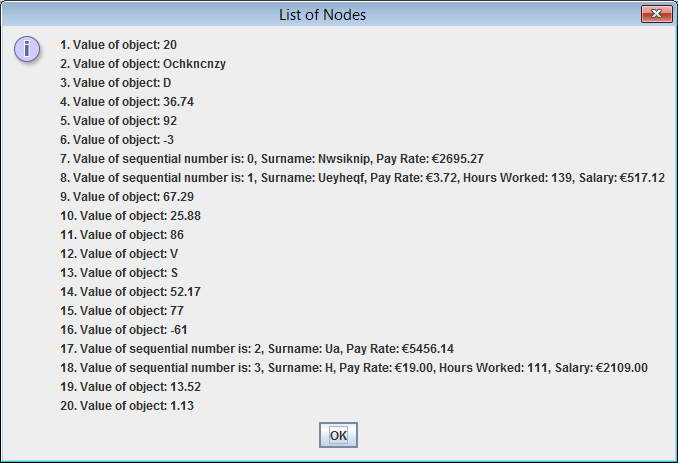
Figure



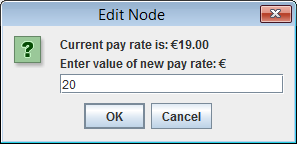
Figure



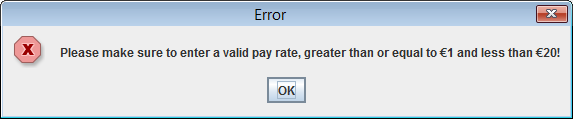
Figure



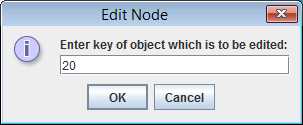
Figure



Figure



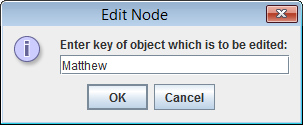
Figure



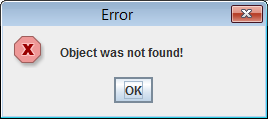
Figure



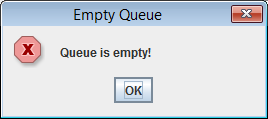
Figure



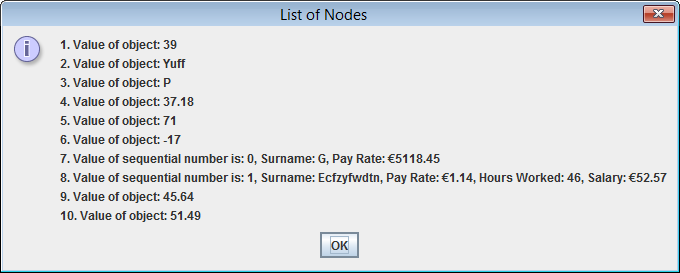
Figure



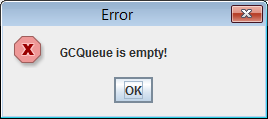
Figure



Figure



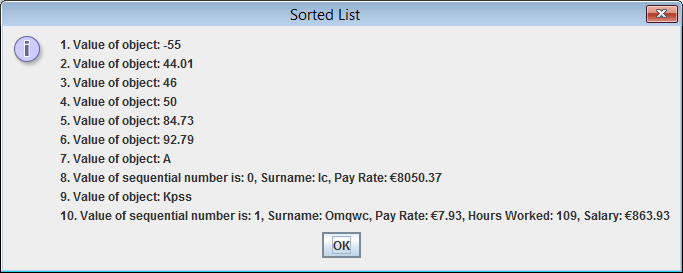
Figure



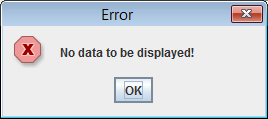
Figure



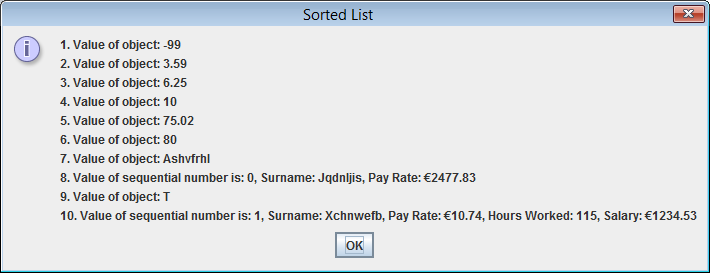
Figure



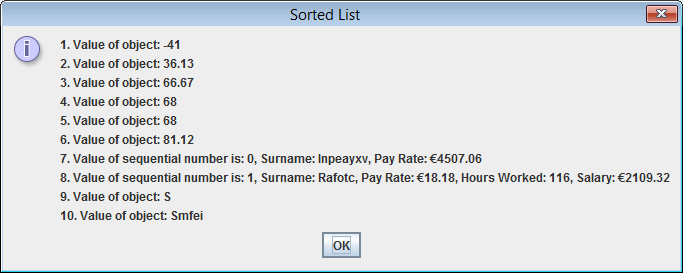
Figure



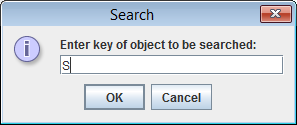
Figure



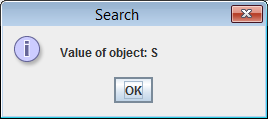
Figure



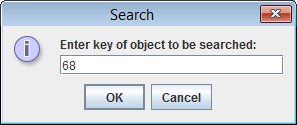
Figure



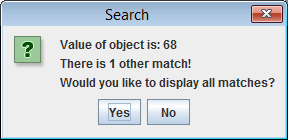
Figure



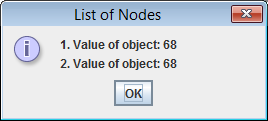
Figure



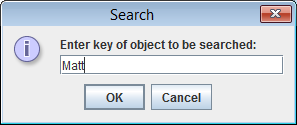
Figure



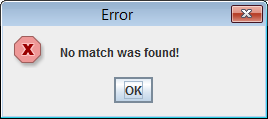
Figure



Figure



Figure



Figure