PA05

Generated by Doxygen 1.8.11

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 File Index

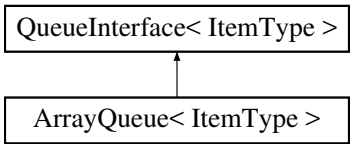
3.1 File List

Here is a list of all documented files with brief descriptions:

4 Class Documentation

4.1 ArrayQueue< ItemType > Class Template Reference

Inheritance diagram for ArrayQueue< ItemType >:



Public Member Functions

- bool isEmpty () const
- bool enqueue (const ItemType &newEntry)
- bool dequeue ()
- ItemType peekFront () const throw (PrecondViolatedExcept)

4.1.1 Member Function Documentation

4.1.1.1 template<class ItemType > bool ArrayQueue< ItemType >::dequeue ( ) [virtual]

Removes the front of this queue.

Postcondition

If the operation was successful, the front of the queue has been removed.

Returns

True if the removal is successful or false if not.

Implements QueueInterface< ItemType >.

4.1.1.2 template<class ItemType > bool ArrayQueue< ItemType >::enqueue (const ItemType & *newEntry*) [virtual]

Adds a new entry to the back of this queue.

Postcondition

If the operation was successful, newEntry is at the back of the queue.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if the addition is successful or false if not.

Implements QueueInterface< ItemType >.

4.1.1.3 template<class ItemType > bool ArrayQueue< ItemType >::isEmpty ( ) const [virtual]

Sees whether this queue is empty.

Returns

True if the queue is empty, or false if not.

Implements QueueInterface< ItemType >.

4.1.1.4 template<class ItemType > ItemType ArrayQueue< ItemType >::peekFront ( ) const throw PrecondViolatedExcept) [virtual]

Exceptions

PrecondViolatedExcept	if queue is empty.

Implements QueueInterface< ItemType >.

The documentation for this class was generated from the following files:

- ArrayQueue.h
- ArrayQueue.cpp

4.2 Event Class Reference

Public Member Functions

- bool **operator==** (const Event &rightSide) const
- bool **operator!=** (const Event &rightSide) const
- bool **operator>** (const Event &rightSide) const
- bool **operator<** (const Event &rightSide) const
- bool **operator=** (const Event &rightSide)

Public Attributes

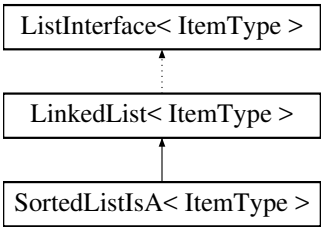
- bool **arrival**
- int **time**
- int **length**

The documentation for this class was generated from the following files:

- Event.h
- Event.cpp

4.3 LinkedList< ItemType > Class Template Reference

Inheritance diagram for LinkedList< ItemType >:



Public Member Functions

- LinkedList ()
- LinkedList (const LinkedList< ItemType > &aList)
- virtual ~LinkedList ()
- bool isEmpty () const
- int getLength () const
- bool insert (int newPosition, const ItemType &newEntry)
- bool remove (int position)
- void clear ()
- ItemType getEntry (int position) const throw (PrecondViolatedExcept)
- void replace (int position, const ItemType &newEntry) throw (PrecondViolatedExcept)

4.3.1 Constructor & Destructor Documentation

4.3.1.1 template<class ItemType > LinkedList< ItemType >::LinkedList ( )

default constructor

4.3.1.2 template<class ItemType > LinkedList< ItemType >::LinkedList (const LinkedList< ItemType > & *aList*)

copy constructor

4.3.1.3 template<class ItemType > LinkedList< ItemType >::~LinkedList ( ) [virtual]

destructor

4.3.2 Member Function Documentation

4.3.2.1 template<class ItemType > void LinkedList< ItemType >::clear ( ) [virtual]

removes all items from the list.

Precondition

None.

Postcondition

List contains no entries.

Implements ListInterface< ItemType >.

4.3.2.2 template<class ItemType > ItemType LinkedList< ItemType >::getEntry (int *position*) const throw PrecondViolatedExcept) [virtual]

Gets the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The desired entry has been returned.

Parameters

position	The list position of the desired entry.

Returns

The entry at the given position.

Exceptions

PrecondViolatedExcept	if position < 1 or position > getLength().

Implements ListInterface< ItemType >.

4.3.2.3 template<class ItemType > int LinkedList< ItemType >::getLength ( ) const [virtual]

checks how many items are in the list

Returns

the integer value of how many items are contained in the list.

Implements ListInterface< ItemType >.

4.3.2.4 template<class ItemType > bool LinkedList< ItemType >::insert (int *newPosition*, const ItemType & *newEntry*) [virtual]

inserts an entry into the list at a given position

Precondition

None.

Postcondition

if the position is valid and insertion is possible a new entry is entered into the list.

Parameters

newPosition	the position in the list at which to insert the new entry.
newEntry	the new item to be placed in the list.

Returns

True if the item was successfully placed in the list.

Implements ListInterface< ItemType >.

Reimplemented in SortedListIsA< ItemType >.

4.3.2.5 template<class ItemType > bool LinkedList< ItemType >::isEmpty ( ) const [virtual]

checks if the list contains any items

Returns

returns true if the list is empty.

Implements ListInterface< ItemType >.

4.3.2.6 template<class ItemType > bool LinkedList< ItemType >::remove (int *position*) [virtual]

removes the entry at the specified position.

Precondition

None.

Postcondition

if the position is valid the item is removed from the list and the list is renumbered.

Parameters

position	the position in the list which contains the item to be removed.

Returns

True if the item was removed succesfully otherwise returns false.

Implements ListInterface< ItemType >.

4.3.2.7 template<class ItemType > void LinkedList< ItemType >::replace (int *position*, const ItemType & *newEntry*) throw PrecondViolatedExcept) [virtual]

Replaces the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The entry at the given position is newEntry.

Parameters

position	The list position of the entry to replace.
newEntry	The replacement entry.

Exceptions

PrecondViolatedExcept	if position < 1 or position > getLength().

Implements ListInterface< ItemType >.

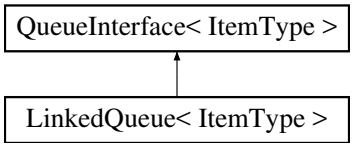
Reimplemented in SortedListIsA< ItemType >.

The documentation for this class was generated from the following files:

- LinkedList.h
- LinkedList.cpp

4.4 LinkedQueue< ItemType > Class Template Reference

Inheritance diagram for LinkedQueue< ItemType >:



Public Member Functions

- **LinkedQueue** (const LinkedQueue &aQueue)
- bool isEmpty () const
- bool enqueue (const ItemType &newEntry)
- bool dequeue ()
- ItemType peekFront () const throw (PrecondViolatedExcept)

4.4.1 Member Function Documentation

4.4.1.1 template<class ItemType > bool LinkedQueue< ItemType >::dequeue ( ) [virtual]

Removes the front of this queue.

Postcondition

If the operation was successful, the front of the queue has been removed.

Returns

True if the removal is successful or false if not.

Implements QueueInterface< ItemType >.

4.4.1.2 template<class ItemType > bool LinkedQueue< ItemType >::enqueue (const ItemType & *newEntry*) [virtual]

Adds a new entry to the back of this queue.

Postcondition

If the operation was successful, newEntry is at the back of the queue.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if the addition is successful or false if not.

Implements QueueInterface< ItemType >.

4.4.1.3 template<class ItemType > bool LinkedQueue< ItemType >::isEmpty ( ) const [virtual]

Sees whether this queue is empty.

Returns

True if the queue is empty, or false if not.

Implements QueueInterface< ItemType >.

4.4.1.4 template<class ItemType > ItemType LinkedQueue< ItemType >::peekFront ( ) const throw PrecondViolatedExcept) [virtual]

Exceptions

PrecondViolatedExcept	if the queue is empty

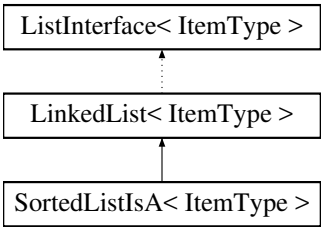
Implements QueueInterface< ItemType >.

The documentation for this class was generated from the following files:

- LinkedQueue.h
- LinkedQueue.cpp

4.5 ListInterface< ItemType > Class Template Reference

Inheritance diagram for ListInterface< ItemType >:



Public Member Functions

- virtual bool isEmpty () const =0
- virtual int getLength () const =0
- virtual bool insert (int newPosition, const ItemType &newEntry)=0
- virtual bool remove (int position)=0
- virtual void clear ()=0
- virtual ItemType getEntry (int position) const =0
- virtual void replace (int position, const ItemType &newEntry)=0

4.5.1 Member Function Documentation

4.5.1.1 template<class ItemType > virtual void ListInterface< ItemType >::clear ( ) [pure virtual]

Removes all entries from this list.

Postcondition

List contains no entries and the count of items is 0.

Implemented in LinkedList< ItemType >.

4.5.1.2 template<class ItemType > virtual ItemType ListInterface< ItemType >::getEntry (int *position*) const [pure virtual]

Gets the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The desired entry has been returned.

Parameters

position	The list position of the desired entry.

Returns

The entry at the given position.

Implemented in LinkedList< ItemType >.

4.5.1.3 template<class ItemType > virtual int ListInterface< ItemType >::getLength ( ) const [pure virtual]

Gets the current number of entries in this list.

Returns

The integer number of entries currently in the list.

Implemented in LinkedList< ItemType >.

4.5.1.4 template<class ItemType > virtual bool ListInterface< ItemType >::insert (int *newPosition*, const ItemType & *newEntry*) [pure virtual]

Inserts an entry into this list at a given position.

Precondition

None.

Postcondition

If 1 <= position <= getLength() + 1 and the insertion is successful, newEntry is at the given position in the list, other entries are renumbered accordingly, and the returned value is true.

Parameters

newPosition	The list position at which to insert newEntry.
newEntry	The entry to insert into the list.

Returns

True if insertion is successful, or false if not.

Implemented in LinkedList< ItemType >, and SortedListIsA< ItemType >.

4.5.1.5 template<class ItemType > virtual bool ListInterface< ItemType >::isEmpty ( ) const [pure virtual]

Sees whether this list is empty.

Returns

True if the list is empty; otherwise returns false.

Implemented in LinkedList< ItemType >.

4.5.1.6 template<class ItemType > virtual bool ListInterface< ItemType >::remove (int *position*) [pure virtual]

Removes the entry at a given position from this list.

Precondition

None.

Postcondition

If 1 <= position <= getLength() and the removal is successful, the entry at the given position in the list is removed, other items are renumbered accordingly, and the returned value is true.

Parameters

position	The list position of the entry to remove.

Returns

True if removal is successful, or false if not.

Implemented in LinkedList< ItemType >.

4.5.1.7 template<class ItemType > virtual void ListInterface< ItemType >::replace (int *position*, const ItemType & *newEntry*) [pure virtual]

Replaces the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The entry at the given position is newEntry.

Parameters

position	The list position of the entry to replace.
newEntry	The replacement entry.

Implemented in LinkedList< ItemType >, and SortedListIsA< ItemType >.

The documentation for this class was generated from the following file:

- ListInterface.h

4.6 Node< ItemType > Class Template Reference

Public Member Functions

- Node ()
- Node (const ItemType &anItem)
- Node (const ItemType &anItem, Node< ItemType > *nextNodePtr)
- void setItem (const ItemType &anItem)
- void setNext (Node< ItemType > *nextNodePtr)
- ItemType getItem () const
- Node< ItemType > * getNext () const

4.6.1 Constructor & Destructor Documentation

4.6.1.1 template<class ItemType > Node< ItemType >::Node ( )

default constructor

4.6.1.2 template<class ItemType > Node< ItemType >::Node (const ItemType & *anItem*)

constructor with item value

4.6.1.3 template<class ItemType > Node< ItemType >::Node (const ItemType & *anItem*, Node< ItemType > * *nextNodePtr*)

constructor with item value and next pointer

4.6.2 Member Function Documentation

4.6.2.1 template<class ItemType > ItemType Node< ItemType >::getItem ( ) const

returns the item stored in the node

4.6.2.2 template<class ItemType > Node< ItemType > * Node< ItemType >::getNext ( ) const

returns the next node

4.6.2.3 template<class ItemType > void Node< ItemType >::setItem (const ItemType & *anItem*)

sets the item value of the node

4.6.2.4 template<class ItemType > void Node< ItemType >::setNext (Node< ItemType > * *nextNodePtr*)

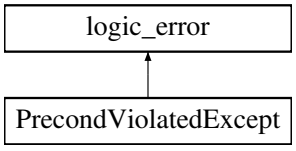
sets the pointer to the next node

The documentation for this class was generated from the following files:

- Node.h
- Node.cpp

4.7 PrecondViolatedExcept Class Reference

Inheritance diagram for PrecondViolatedExcept:



Public Member Functions

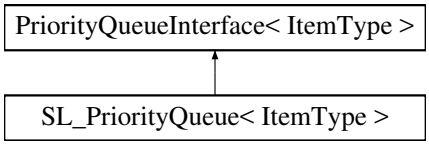
- **PrecondViolatedExcept** (const std::string &message="")

The documentation for this class was generated from the following files:

- PrecondViolatedExcept.h
- PrecondViolatedExcept.cpp

4.8 PriorityQueueInterface< ItemType > Class Template Reference

Inheritance diagram for PriorityQueueInterface< ItemType >:



Public Member Functions

- virtual bool isEmpty () const =0
- virtual bool enqueue (const ItemType &newEntry)=0
- virtual bool dequeue ()=0
- virtual ItemType peekFront () const =0
- virtual ~PriorityQueueInterface ()

4.8.1 Constructor & Destructor Documentation

4.8.1.1 template<class ItemType > virtual PriorityQueueInterface< ItemType >::~PriorityQueueInterface ( ) [inline], [virtual]

Destroys this queue and frees its memory.

4.8.2 Member Function Documentation

4.8.2.1 template<class ItemType > virtual bool PriorityQueueInterface< ItemType >::dequeue ( ) [pure virtual]

Removes the front of this queue.

Postcondition

If the operation was successful, the front of the queue has been removed.

Returns

True if the removal is successful or false if not.

Implemented in SL_PriorityQueue< ItemType >.

4.8.2.2 template<class ItemType > virtual bool PriorityQueueInterface< ItemType >::enqueue (const ItemType & *newEntry*) [pure virtual]

Adds a new entry to the back of this queue.

Postcondition

If the operation was successful, newEntry is at the back of the queue.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if the addition is successful or false if not.

Implemented in SL_PriorityQueue< ItemType >.

4.8.2.3 template<class ItemType > virtual bool PriorityQueueInterface< ItemType >::isEmpty ( ) const [pure virtual]

Sees whether this queue is empty.

Returns

True if the queue is empty, or false if not.

Implemented in SL_PriorityQueue< ItemType >.

4.8.2.4 template<class ItemType > virtual ItemType PriorityQueueInterface< ItemType >::peekFront ( ) const [pure virtual]

Returns the front of this queue.

Precondition

The queue is not empty.

Postcondition

The front of the queue has been returned, and the queue is unchanged.

Returns

The front of the queue.

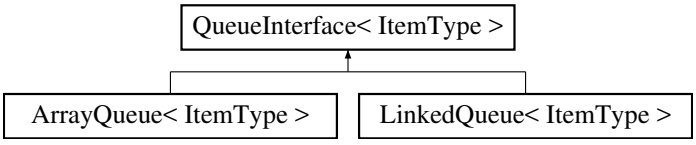
Implemented in SL_PriorityQueue< ItemType >.

The documentation for this class was generated from the following file:

- PriorityQueueInterface.h

4.9 QueueInterface< ItemType > Class Template Reference

Inheritance diagram for QueueInterface< ItemType >:



Public Member Functions

- virtual bool isEmpty () const =0
- virtual bool enqueue (const ItemType &newEntry)=0
- virtual bool dequeue ()=0
- virtual ItemType peekFront () const =0
- virtual ~QueueInterface ()

4.9.1 Constructor & Destructor Documentation

4.9.1.1 template<class ItemType > virtual QueueInterface< ItemType >::~QueueInterface ( ) [inline], [virtual]

Destroys this queue and frees its memory.

4.9.2 Member Function Documentation

4.9.2.1 template<class ItemType > virtual bool QueueInterface< ItemType >::dequeue ( ) [pure virtual]

Removes the front of this queue.

Postcondition

If the operation was successful, the front of the queue has been removed.

Returns

True if the removal is successful or false if not.

Implemented in LinkedQueue< ItemType >, and ArrayQueue< ItemType >.

4.9.2.2 template<class ItemType > virtual bool QueueInterface< ItemType >::enqueue (const ItemType & *newEntry*) [pure virtual]

Adds a new entry to the back of this queue.

Postcondition

If the operation was successful, newEntry is at the back of the queue.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if the addition is successful or false if not.

Implemented in LinkedQueue< ItemType >, and ArrayQueue< ItemType >.

4.9.2.3 template<class ItemType > virtual bool QueueInterface< ItemType >::isEmpty ( ) const [pure virtual]

Sees whether this queue is empty.

Returns

True if the queue is empty, or false if not.

Implemented in LinkedQueue< ItemType >, and ArrayQueue< ItemType >.

4.9.2.4 template<class ItemType > virtual ItemType QueueInterface< ItemType >::peekFront ( ) const [pure virtual]

Returns the front of this queue.

Precondition

The queue is not empty.

Postcondition

The front of the queue has been returned, and the queue is unchanged.

Returns

The front of the queue.

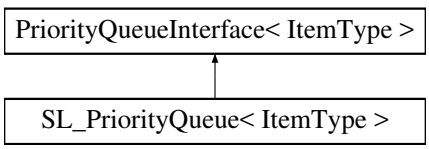
Implemented in LinkedQueue< ItemType >, and ArrayQueue< ItemType >.

The documentation for this class was generated from the following file:

- QueueInterface.h

4.10 SL_PriorityQueue< ItemType > Class Template Reference

Inheritance diagram for SL_PriorityQueue< ItemType >:



Public Member Functions

- **SL_PriorityQueue** (const SL_PriorityQueue &pq)
- bool isEmpty () const
- bool enqueue (const ItemType &newEntry)
- bool dequeue ()
- ItemType peekFront () const throw (PrecondViolatedExcept)

4.10.1 Member Function Documentation

4.10.1.1 template<class ItemType > bool SL_PriorityQueue< ItemType >::dequeue ( ) [virtual]

Removes the front of this queue.

Postcondition

If the operation was successful, the front of the queue has been removed.

Returns

True if the removal is successful or false if not.

Implements PriorityQueueInterface< ItemType >.

4.10.1.2 template<class ItemType > bool SL_PriorityQueue< ItemType >::enqueue (const ItemType & *newEntry*) [virtual]

Adds a new entry to the back of this queue.

Postcondition

If the operation was successful, newEntry is at the back of the queue.

Parameters

newEntry	The object to be added as a new entry.

Returns

True if the addition is successful or false if not.

Implements PriorityQueueInterface< ItemType >.

4.10.1.3 template<class ItemType > bool SL_PriorityQueue< ItemType >::isEmpty ( ) const [virtual]

Sees whether this queue is empty.

Returns

True if the queue is empty, or false if not.

Implements PriorityQueueInterface< ItemType >.

4.10.1.4 template<class ItemType > ItemType SL_PriorityQueue< ItemType >::peekFront ( ) const throw PrecondViolatedExcept) [virtual]

Exceptions

PrecondViolatedExcept	if priority queue is empty.

Implements PriorityQueueInterface< ItemType >.

The documentation for this class was generated from the following files:

- SL_PriorityQueue.h
- SL_PriorityQueue.cpp

4.11 SortedListInterface< ItemType > Class Template Reference

Public Member Functions

- virtual bool insertSorted (const ItemType &newEntry)=0
- virtual bool removeSorted (const ItemType &anEntry)=0
- virtual int getPosition (const ItemType &anEntry) const =0
- virtual bool isEmpty () const =0
- virtual int getLength () const =0
- virtual bool remove (int position)=0
- virtual void clear ()=0
- virtual ItemType getEntry (int position) const =0
- virtual ~SortedListInterface ()

4.11.1 Constructor & Destructor Documentation

4.11.1.1 template<class ItemType > virtual SortedListInterface< ItemType >::~SortedListInterface ( ) [inline], [virtual]

Destroys this sorted list and frees its assigned memory.

4.11.2 Member Function Documentation

4.11.2.1 template<class ItemType > virtual void SortedListInterface< ItemType >::clear ( ) [pure virtual]

Removes all entries from this list.

4.11.2.2 template<class ItemType > virtual ItemType SortedListInterface< ItemType >::getEntry (int *position*) const [pure virtual]

Gets the entry at the given position in this list.

4.11.2.3 template<class ItemType > virtual int SortedListInterface< ItemType >::getLength ( ) const [pure virtual]

Gets the current number of entries in this list.

4.11.2.4 template<class ItemType > virtual int SortedListInterface< ItemType >::getPosition (const ItemType & *anEntry*) const [pure virtual]

Gets the position of the first or only occurrence of the given entry in this sorted list. In case the entry is not in the list, determines where it should be if it were added to the list.

Precondition

None.

Postcondition

The position where the given entry is or belongs is returned. The sorted list is unchanged.

Parameters

anEntry	The entry to locate.

Returns

Either the position of the given entry, if it occurs in the sorted list, or the position where the entry would occur, but as a negative integer.

4.11.2.5 template<class ItemType > virtual bool SortedListInterface< ItemType >::insertSorted (const ItemType & *newEntry*) [pure virtual]

Inserts an entry into this sorted list in its proper order so that the list remains sorted.

Precondition

None.

Postcondition

newEntry is in the list, and the list is sorted.

Parameters

newEntry	The entry to insert into the sorted list.

Returns

True if insertion is successful, or false if not.

4.11.2.6 template<class ItemType > virtual bool SortedListInterface< ItemType >::isEmpty ( ) const [pure virtual]

Sees whether this list is empty.

4.11.2.7 template<class ItemType > virtual bool SortedListInterface< ItemType >::remove (int *position*) [pure virtual]

Removes the entry at a given position from this list.

4.11.2.8 template<class ItemType > virtual bool SortedListInterface< ItemType >::removeSorted (const ItemType & *anEntry*) [pure virtual]

Removes the first or only occurrence of the given entry from this sorted list.

Precondition

None.

Postcondition

If the removal is successful, the first occurrence of the given entry is no longer in the sorted list, and the returned value is true. Otherwise, the sorted list is unchanged and the returned value is false.

Parameters

anEntry	The entry to remove.

Returns

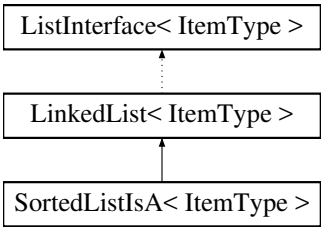
True if removal is successful, or false if not.

The documentation for this class was generated from the following file:

- SortedListInterface.h

4.12 SortedListIsA< ItemType > Class Template Reference

Inheritance diagram for SortedListIsA< ItemType >:



Public Member Functions

- **SortedListIsA** (const SortedListIsA< ItemType > &sList)
- bool **insertSorted** (const ItemType &newEntry)
- bool **removeSorted** (int position)
- int **getPosition** (const ItemType &anEntry) const
- bool insert (int newPosition, const ItemType &newEntry) override
- void replace (int position, const ItemType &newEntry) override throw (PrecondViolatedExcept)

4.12.1 Member Function Documentation

4.12.1.1 template<class ItemType > bool SortedListIsA< ItemType >::insert (int *newPosition,* const ItemType & *newEntry*) [override], [virtual]

inserts an entry into the list at a given position

Precondition

None.

Postcondition

if the position is valid and insertion is possible a new entry is entered into the list.

Parameters

newPosition	the position in the list at which to insert the new entry.
newEntry	the new item to be placed in the list.

Returns

True if the item was successfully placed in the list.

Reimplemented from LinkedList< ItemType >.

4.12.1.2 template<class ItemType > void SortedListIsA< ItemType >::replace (int *position,* const ItemType & *newEntry*) throw PrecondViolatedExcept) [override], [virtual]

Replaces the entry at the given position in this list.

Precondition

1 <= position <= getLength().

Postcondition

The entry at the given position is newEntry.

Parameters

position	The list position of the entry to replace.
newEntry	The replacement entry.

Exceptions

PrecondViolatedExcept	if position < 1 or position > getLength().

Reimplemented from LinkedList< ItemType >.

The documentation for this class was generated from the following files:

- SortedListIsA.h
- SortedListIsA.cpp

5 File Documentation

5.1 ArrayQueue.cpp File Reference

```
#include "ArrayQueue.h"
```

5.1.1 Detailed Description

ADT queue: Circular array-based implementation.

5.2 ArrayQueue.h File Reference

```
#include "QueueInterface.h"
#include "PrecondViolatedExcept.h"
#include "ArrayQueue.cpp"
```

Classes

- class ArrayQueue< ItemType >

5.2.1 Detailed Description

ADT queue: Circular array-based implementation.

5.3 LinkedList.cpp File Reference

```
#include "LinkedList.h"
#include "Node.h"
#include "PrecondViolatedExcept.h"
```

5.3.1 Detailed Description

ADT list: Link-based implementation.

5.4 LinkedList.h File Reference

Header file for a linked list.

```
#include "ListInterface.h"
#include "Node.h"
#include "PrecondViolatedExcept.h"
#include "LinkedList.cpp"
```

Classes

- class LinkedList< ItemType >

5.4.1 Detailed Description

Header file for a linked list.

ADT list: Link-based implementation. Listing 9-2.

establishes functions for list Created by Frank M. Carrano and Timothy M. Henry. Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

5.5 LinkedQueue.h File Reference

```
#include "QueueInterface.h"
#include "Node.h"
#include "PrecondViolatedExcept.h"
#include <memory>
#include "LinkedQueue.cpp"
```

Classes

- class LinkedQueue< ItemType >

5.5.1 Detailed Description

ADT queue: Link-based implementation.

5.6 ListInterface.h File Reference

Interface file for the List ADT.

Classes

- class ListInterface< ItemType >

5.6.1 Detailed Description

Interface file for the List ADT.

Author

Rory Pierce

Specifies the implementation contract of the List ADT

Version

0.10

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

5.7 Node.cpp File Reference

```
#include "Node.h"
#include <iostream>
```

5.7.1 Detailed Description

Listing 4-2

5.8 Node.h File Reference

```
#include "Node.cpp"
```

Classes

- class Node< ItemType >

5.8.1 Detailed Description

Listing 4-1

5.9 PrecondViolatedExcept.cpp File Reference

```
#include "PrecondViolatedExcept.h"
```

5.9.1 Detailed Description

Listing 7-6.

5.10 PrecondViolatedExcept.h File Reference

```
#include <stdexcept>
#include <string>
```

Classes

- class PrecondViolatedExcept

5.10.1 Detailed Description

Listing 7-5.

5.11 PriorityQueueInterface.h File Reference

Classes

- class PriorityQueueInterface< ItemType >

5.12 QueueInterface.h File Reference

Classes

- class QueueInterface< ItemType >

5.13 SL_PriorityQueue.h File Reference

```
#include "PriorityQueueInterface.h"
#include "SortedListIsA.h"
#include "PrecondViolatedExcept.h"
#include <memory>
#include "SL_PriorityQueue.cpp"
```

Classes

- class SL_PriorityQueue< ItemType >

5.13.1 Detailed Description

ADT priority queue: ADT sorted list implementation.

5.14 SortedListInterface.h File Reference

Classes

- class SortedListInterface< ItemType >

5.14.1 Detailed Description

Interface for the ADT sorted list

5.15 SortedListIsA.h File Reference

```
#include <memory>
#include "LinkedList.h"
#include "Node.h"
#include "PrecondViolatedExcept.h"
#include "SortedListIsA.cpp"
```

Classes

- class SortedListIsA< ItemType >

5.15.1 Detailed Description

ADT sorted list using ADT list.

Index