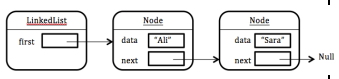
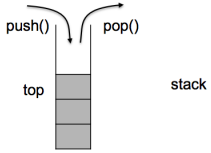
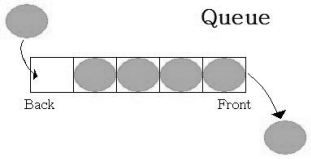
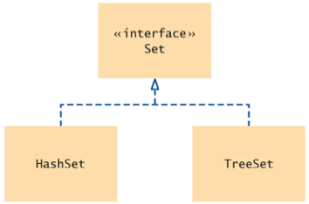
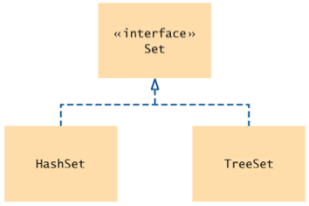
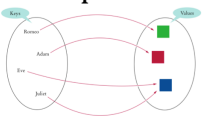
Data Structure – Java

	Concept	Library	Declaration	Insert	Remove	Update/Get	Note
Linked List	consists of a number of nodes, each of which has a reference to the next node 	java.util. LinkedList java.util. ListIterator	-LinkedList: ***LinkedList*** **<**DataType**>** llname ***= new LinkedList*** **<**DataType**>*();*** -Iterator: ***ListIterator*** **<**DataType**>** itr = llname. ***listIterator();***	-At beginning: llname.***addFirst***(..); -At end: llname.***addLast***(..); -After specific position based on iterator: iter.***add(..);***	-From beginning: llname.***removeFirst*** (..); -From end: llname.***removeLast*** (..); -From specific position based on iterator: iter.***next( );*** iter.***add(..);***	We can get the element using the returning value of: iter.next(); e.g.: `String x = iter.next();` So, we can update node's data through x e.g.: `x.data = "Ali";`	DoubleLinked List is similar to LinkedList except that each node has another reference to the previous node. So, we use iter. ***previous();***

Related topics:

- Inserting implementation (steps to re-order links between nodes)
- Removing implementation (steps to re-order links between nodes)
- Efficiency – Big-O notation
- Methods of `ListIterator` interface

	Concept	Library	Declaration	Insert	Remove	Update/Get	Note
Stack	Collection of items with "last in, first out" LIFO retrieval 	java.util. Stack	`Stack <DataType> s = new Stack <DataType> ();`	Only at the end: `s.push(..);`	Only from the end: `s.pop();`	We can get the top element only without removing it using: `s.peek();`	Elements pushed to stack are poped in reverse order
Queue	Collection of items with "first in, first out" FIFO retrieval 	java.util. Queue java.util. LinkedList	`Queue <DataType> q = new LinkedList <DataType> ();`	Only at the end (tail): `q.add(..);`	Only from the beginning (head): `q.remove();`	We can get the head element only without removing it using: `q.peek();`	Elements added to the queue are removed in the same order

	Concept	Library	Declaration	Insert	Remove	Update/Get	Note
Set	Unordered collection of distinct elements 	java.util. Set java.util. HashSet java.util. TreeSet	If order is not important: `Set <DataType> s = new HashSet <DataType>();` If order is important: `Set <DataType> s = new TreeSet <DataType>();`	s.***add(..);*** if element is exist, skip adding	s.***remove(..);***	Use an iterator to visit all elements in a set : ***Iterator***<DataType> iter = s.***iterator( );*** iter.***next( );*** e.g.: `String x = iter.next();`	No duplication
Map	Function from one set, the key set, to another set, the value set. Every key in a map has a unique value. A value may be associated with several keys. 	java.util. Map java.util. HashMap java.util. TreeMap	If order is not important: `Map <KeyDataType, ValueDataType> m = new HashMap <KeyDataType, ValueDataType>();` If order is important: `Map <KeyDataType, ValueDataType> m = new TreeMap <KeyDataType, ValueDataType>();`	m.***put(..);*** if key is exist, update value	m.***remove(..);***	To update a value, use the same inserting method with the same key but new value.	No duplication 

Related topics:

- Other methods of Set and Map
- How to Display Set\Map elements