1. ArrayList and Iterator in Java

Inserting elements between existing elements of an ArrayList or Vector is an inefficient operation- all element after the new one must be moved out of the way which could be an expensive operation in a collection.

To define ArrayList

```
ArrayList<String> obj = new ArrayList<String>();
```

- ***Methods of ArrayList class***

a. add(Object o): This method adds an object o to the arraylist.

```
    obj.add("hello");
```

b. add(int index, Object o): It adds the object o to the array list at the given index.

```
      obj.add(2, "bye");
```

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

c. remove(Object o): Removes the object o from the ArrayList

```
      obj.remove("Chaitanya");
```

This statement will remove the string "Chaitanya" from the ArrayList.

d. remove(int index): Removes element from a given index.

```
obj.remove(3);
```

It would remove the element of index 3 (4th element of the list – List starts with o).

e. set(int index, Object o): Used for updating an element. It replaces the element present at the specified index with the object o.

```
obj.set(2, "Tom");
```

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

f. int indexOf(Object o): Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

```
int pos = obj.indexOf("Tom");
```

This would give the index (position) of the string Tom in the list.

g. Object get(int index): It returns the object of list which is present at the specified index.

```
String str= obj.get(2);
```

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string "str". We have stored the returned value in string variable because in our example we have defined the ArrayList is of **String** type. If you are having integer array list then the returned value should be stored in an integer variable.

h. int size(): It gives the size of the ArrayList – Number of elements of the list.

```
int numberofitems = obj.size();
```

i. boolean contains(Object o): It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

```
obj.contains("Steve");
```

It would return true if the string "Steve" is present in the list else we would get false.

j. clear(): It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

```
obj.clear();
```

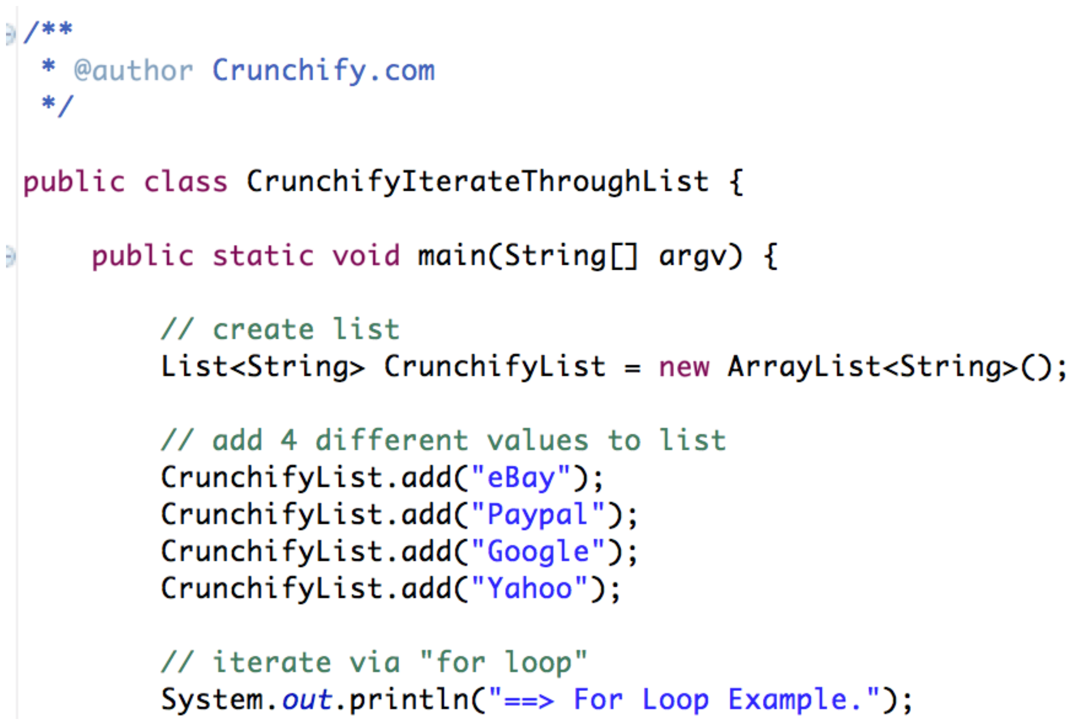
Examples

```
package com.tutorialspoint;
import java.util.ArrayList;


public class ArrayListDemo {
   public static void main(String[] args) {


   // create an empty arraylist with an initial capacity
   ArrayList<Integer> arrlist = new ArrayList<Integer>(5);


   // use add() method to add elements in the list
   arrlist.add(15);
```

```
  arrlist.add(20);
  arrlist.add(25);
  arrlist.add(22);

  // let us print all the elements available in list
  for (Integer number : arrlist) {
  System.out.println("Number = " + number);
  }

  // inserting elment 55 at 3rd position
  arrlist.set(2,55);

  // let us print all the elements available in list
  System.out.println("Printing new list:");
  for (Integer number : arrlist) {
  System.out.println("Number = " + number);
  }
  }
}
```

Let us compile and run the above program, this will produce the following result:

```
Number = 15
Number = 20
Number = 25
Number = 22
Printing new list:
Number = 15
Number = 20
Number = 55
Number = 22
```



```java
/**
 * @author Crunchify.com
 */

public class CrunchifyIterateThroughList {

    public static void main(String[] argv) {

        // create list
        List<String> CrunchifyList = new ArrayList<String>();

        // add 4 different values to list
        CrunchifyList.add("eBay");
        CrunchifyList.add("Paypal");
        CrunchifyList.add("Google");
        CrunchifyList.add("Yahoo");

        // iterate via "for loop"
        System.out.println("==> For Loop Example.");
```

How to iterate through Java List? This tutorial demonstrates the use of ArrayList, Iterator and a List.

There are 5 ways you can iterate through List.

1. For Loop
2. Advanced For Loop
3. Iterator
4. While Loop
5. Collections’s stream() util (Java8)

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

public class CollectionTest
{
    private static final String[] colors =
            { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
    private static final String[] removeColors =
           { "RED", "WHITE", "BLUE" };

 // create ArrayList, add Colors to it and manipulate it
   public CollectionTest()
     {
        List< String > list = new ArrayList< String >( );
        List< String > removeList = new ArrayList< String >( );

       // add elements in colors array to list
        for ( String color : colors )
            list.add( color );

       // add elements in removeColors to removeList
       for ( String color : removeColors )
            removeList.add( color );
       System.out.println( "ArrayList: " );

      // output list contents
      for ( int count = 0; count < list.size(); count++ )
          System.out.printf( "%s ", list.get( count ) );

      // remove colors contained in removeList
      removeColors( list, removeList );
      System.out.println( "\n\nArrayList after calling removeColors: " );



      // output list contents
      for ( String color : list )
         System.out.printf( "%s ", color );
 } // end CollectionTest constructor
// remove colors specified in collection2 from collection1
private void removeColors(Collection< String > collection1, Collection< String >  collection2 )
 {
      Iterator< String > iterator = collection1.iterator();

    // loop while collection has items
    while ( iterator.hasNext() )
         if ( collection2.contains( iterator.next() ) )
                iterator.remove(); // remove current Color

} // end method removeColors

  public static void main( String args[] )
{
  new CollectionTest();
} // end main

} // end class CollectionTest
```

Java Example:

You need JDK 8 to run below program as `point-5` above uses `stream()` util.

void `java.util.stream.Stream.forEach`(Consumer<? super String> action) performs an action for each element of this stream.

```
package crunchify.com.tutorial;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class CrunchifyIterateThroughList {

        public static void main(String[] argv) {

                // create list
                List<String> CrunchifyList = new ArrayList<String>();
```

```
		// add 4 different values to list
		CrunchifyList.add("eBay");
		CrunchifyList.add("Paypal");
		CrunchifyList.add("Google");
		CrunchifyList.add("Yahoo");

		// iterate via "for loop"
		System.out.println("==> For Loop Example.");
		for (int i = 0; i < CrunchifyList.size(); i++) {
			System.out.println(CrunchifyList.get(i));
		}

		// iterate via "New way to loop"
		System.out.println("\n==> Advance For Loop Example..");
		for (String temp : CrunchifyList) {
			System.out.println(temp);
		}

		// iterate via "iterator loop"
		System.out.println("\n==> Iterator Example...");
		Iterator<String> CrunchifyIterator = CrunchifyList.iterator();
		while (CrunchifyIterator.hasNext()) {
			System.out.println(CrunchifyIterator.next());
		}

		// iterate via "while loop"
		System.out.println("\n==> While Loop Example....");
		int i = 0;
		while (i < CrunchifyList.size()) {
			System.out.println(CrunchifyList.get(i));
			i++;
		}

		// collection stream() util: Returns a sequential Stream with this collection as its
source
		System.out.println("\n==> collection stream() util....");
		CrunchifyList.forEach((temp) -> {
			System.out.println(temp);
		});
	}
```

```
}
```

Output:

```
==> For Loop Example.
eBay
Paypal
Google
Yahoo

==> Advance For Loop Example..
eBay
Paypal
Google
Yahoo

==> Iterator Example...
eBay
Paypal
Google
Yahoo

==> While Loop Example....
eBay
Paypal
```

```
22 Google

23 Yahoo

24

25 ==> collection stream() util....

26 eBay

27 Paypal

28 Google

29 Yahoo
```

Hi Aravinthan - You could easily convert List to Set and Set to List.

List to set:

```
Set<string> set = new HashSet<string>(list);
```

Set to List:

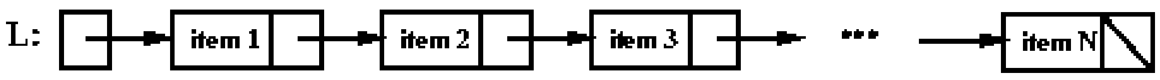
```
List<string> list = new ArrayList<string>(set);
```

Iterate through list or Set:

```
for (String temp : set/list){
        System.out.println(temp);
       }
```

2. LinkedList and Iterator in Java

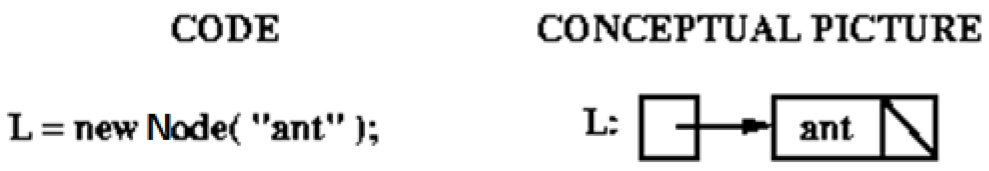
Here's a conceptual picture of a linked list containing N items, pointed to by a variable named L:



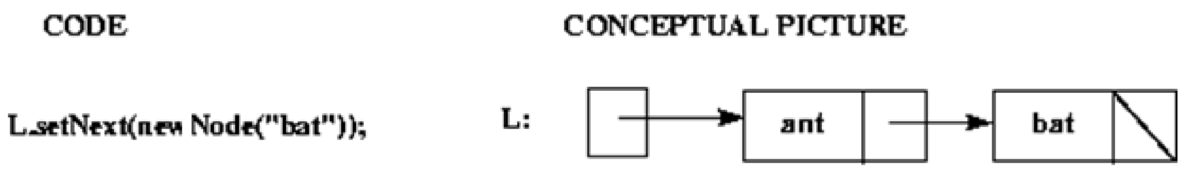
Note that a linked list consists of one or more nodes. Each node contains some data (in this example, item 1, item 2, etc) and a pointer. For each node other than the last one, the pointer points to the next node in the list. For the last node, the pointer is null (indicated in the example using a diagonal line). To implement linked lists in Java, we will define a *Listnode* class, to be used to represent the individual nodes of the list.

```
public class Node {
// reference to the next node in the chain, or null if
there isn't one.
          Node next;
// data carried by this node. could be of any type you
need.
          Object data;

// Node constructor
          public Node(Object dataValue) {
               next = null;
               data = dataValue;
          }

          // another Node constructor if we want to specify
the node to point to.
          @SuppressWarnings("unused")
          public Node(Object dataValue, Node nextValue)
       {
               next = nextValue;
               data = dataValue;
          }

          // these methods should be self-explanatory
          public Object getData() {
               return data;
          }

          @SuppressWarnings("unused")
          public void setData(Object dataValue) {
               data = dataValue;
```

```
        }

        public Node getNext() {
            return next;
        }

        public void setNext(Node nextValue) {
            next = nextValue;
        }

    }
```



```
L = new Node( "ant" );
```



```
L.setNext(new Node("bat"));
```

3. *Linked List Operations*

```
public class linkListClass {

        private static int counter;
        private Node head;

        // Default constructor
        public linkListClass() {

        }

        // appends the specified element to the end of this list.
        public void add(Object data) {

                // Initialize Node only incase of 1st element
                if (head == null) {
                        head = new Node(data);
```

```
		}

		Node Temp = new Node(data);
		Node Current = head;

		// Let's check for NPE before iterate over crunchifyCurrent
		if (Current != null) {

// starting at the head node, crawl to the end of the list and then add element after last node
			while (Current.getNext() != null) {
				Current = Current.getNext();
			}

// the last node's "next" reference set to our new node
			Current.setNext(Temp);
		}

		// increment the number of elements variable
		incrementCounter();
	}

	private static int getCounter() {
		return counter;
	}

	private static void incrementCounter() {
		counter++;
	}

	private void decrementCounter() {
		counter--;
	}

	// inserts the specified element at the specified position in this list
	public void add(Object data, int index) {
		Node Temp = new Node(data);
		Node Current = head;

		// Let's check for NPE before iterate over crunchifyCurrent
		if (Current != null) {
			// crawl to the requested index or the last element in the list, whichever comes
first

			for (int i = 1; i < index && Current.getNext() != null; i++) {
				Current = Current.getNext();
			}
		}

		// set the new node's next-node reference to this node's next-node reference
		Current.setNext(Current.getNext());

		// now set this node's next-node reference to the new node
		Current.setNext(Temp);
		// increment the number of elements variable
		incrementCounter();
	}

	public Object get(int index)
	// returns the element at the specified position in this list.
	{
		// index must be 1 or higher
		if (index <= 0)
			return null;
		Node Current = null;
		if (head != null) {
			Current = head.getNext();
			for (int i = 0; i < index; i++) {
				if (Current.getNext() == null)
					return null;

				Current = Current.getNext();
			}
			return Current.getData();
		}
		return Current;

	}

	// removes the element at the specified position in this list.
	public boolean remove(int index) {

		// if the index is out of range, exit
		if (index < 1 || index > size())
			return false;

		Node Current = head;
		if (head != null) {
			for (int i = 0; i < index; i++) {
				if (Current.getNext() == null)
					return false;

				Current = Current.getNext();
			}
			Current.setNext(Current.getNext().getNext());

			// decrement the number of elements variable
			decrementCounter();
			return true;

		}
		return false;
	}

	// returns the number of elements in this list.
	public int size() {
		return getCounter();
	}

	public String toString() {
		String output = "";

		if (head != null) {
			Node Current = head.getNext();
			while (Current != null) {
				output += "[" + Current.getData().toString() + "]";
				Current = Current.getNext();
			}

		}
		return output;
	}

}
```

Operations of Linked List

```
public class linkListClass {

	private static int counter;
	private Node head;

	// Default constructor
	public linkListClass() {

	}

	// appends the specified element to the end of this list.
	public void add(Object data) {

		// Initialize Node only incase of 1st element
		if (head == null) {
			head = new Node(data);
		}

		Node Temp = new Node(data);
		Node Current = head;

		// Let's check for NPE before iterate over crunchifyCurrent
		if (Current != null) {

			// starting at the head node, crawl to the end of the list and then add element
after last node
			while (Current.getNext() != null) {
				Current = Current.getNext();
			}

			// the last node's "next" reference set to our new node
```

```
				Current.setNext(Temp);
			}

			// increment the number of elements variable
			incrementCounter();
		}

		private static int getCounter() {
			return counter;
		}

		private static void incrementCounter() {
			counter++;
		}

		private void decrementCounter() {
			counter--;
		}

		// inserts the specified element at the specified position in this list
		public void add(Object data, int index) {
			Node Temp = new Node(data);
			Node Current = head;

			// Let's check for NPE before iterate over crunchifyCurrent
			if (Current != null) {
				// crawl to the requested index or the last element in the list, whichever comes
first
				for (int i = 1; i < index && Current.getNext() != null; i++) {
					Current = Current.getNext();
				}
			}

			// set the new node's next-node reference to this node's next-node reference
			Temp.setNext(Current.getNext());

			// now set this node's next-node reference to the new node
			Current.setNext(Temp);

			// increment the number of elements variable
			incrementCounter();
		}

		public Object get(int index)
		// returns the element at the specified position in this list.
		{
			// index must be 1 or higher
			if (index <= 0)
				return null;
			Node Current = null;
			if (head != null) {
				Current = head.getNext();
```

```
                for (int i = 0; i < index; i++) {
                        if (Current.getNext() == null)
                                return null;

                        Current = Current.getNext();
                }
                return Current.getData();
        }
        return Current;

}

// removes the element at the specified position in this list.
public boolean remove(int index) {

        // if the index is out of range, exit
        if (index < 1 || index > size())
                return false;

        Node Current = head;
        if (head != null) {
                for (int i = 0; i < index; i++) {
                        if (Current.getNext() == null)
                                return false;

                        Current = Current.getNext();
                }
                Current.setNext(Current.getNext().getNext());

                // decrement the number of elements variable
                decrementCounter();
                return true;

        }
        return false;
}

// returns the number of elements in this list.
public int size() {
        return getCounter();
}

public String toString() {
        String output = "";

        if (head != null) {
                Node Current = head.getNext();
                while (Current != null) {
                        output += "[" + Current.getData().toString() + "]";
                        Current = Current.getNext();
                }
                }
                return output;
        }

}
```