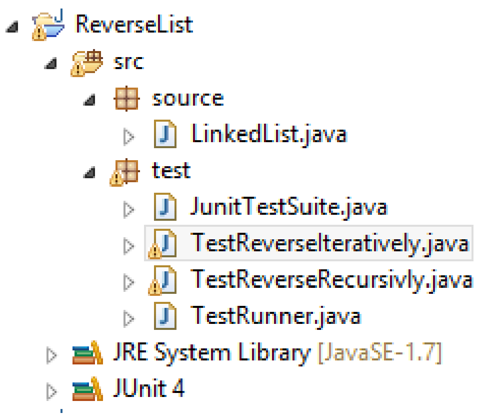
To run the test execute **TestRunner.java**


ReverseList
src
source
LinkedList.java
test
JunitTestSuite.java
TestReverseIteratively.java
TestReverseRecursivly.java
TestRunner.java
JRE System Library [JavaSE-1.7]
JUnit 4

Implementation of a simple singly-linked list, and two functions to reverse the order of the list Using Java and Junit 4.0.

1. An iterative reverse.
2. A recursive reverse.
3. A full suite of automated tests.

LinkedList.java

STRECTURE OF LINKEDLIS

Constructor

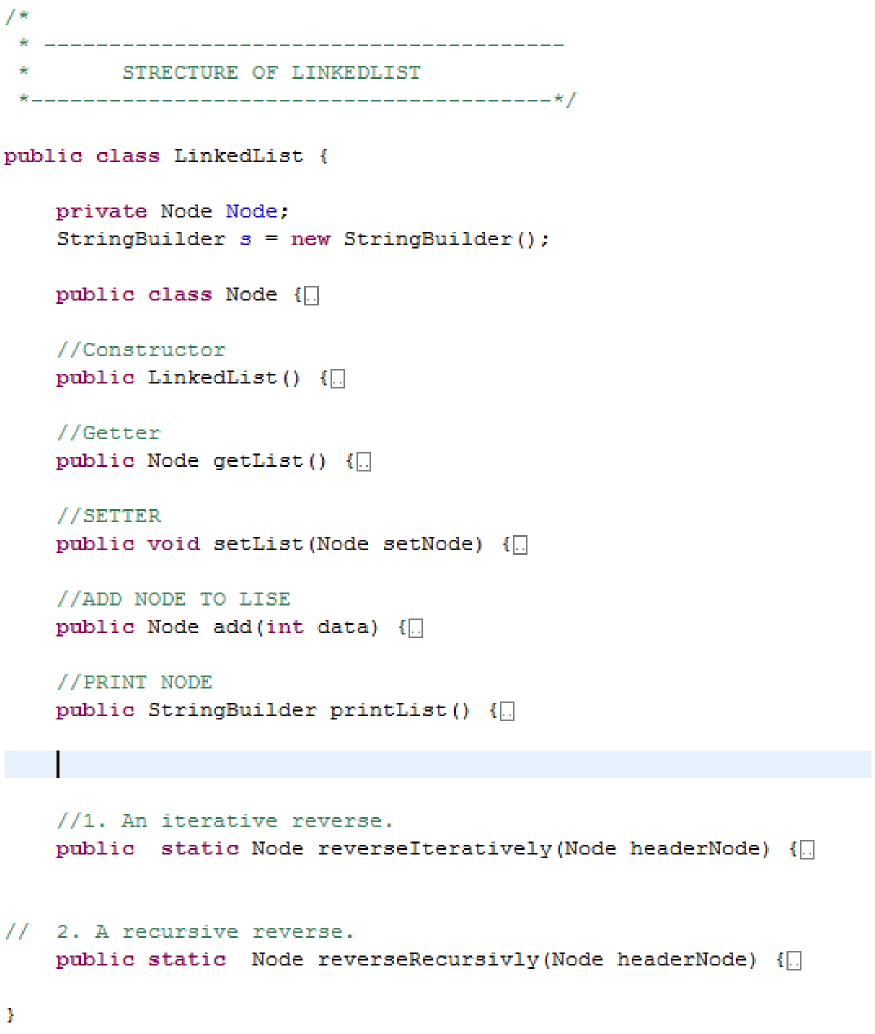
LinkedList()

Getter and Setter

ListNode getList()
void setList(ListNode setListNode)

Helpers methods

ListNode add(int data)
StringBuilder printList()



```
/*
 * ------------------------------------------
 *       STRECTURE OF LINKEDLIST
 *------------------------------------------*/

public class LinkedList {

    private Node Node;
    StringBuilder s = new StringBuilder();

    public class Node {

    //Constructor
    public LinkedList() {

    //Getter
    public Node getList() {

    //SETTER
    public void setList(Node setNode) {

    //ADD NODE TO LISE
    public Node add(int data) {

    //PRINT NODE
    public StringBuilder printList() {



    //1. An iterative reverse.
    public  static Node reverseIteratively(Node headerNode) {


//  2. A recursive reverse.
    public static  Node reverseRecursivly(Node headerNode) {

}
```

Additional Functions

Reverse Functions:

An iterative reverse.
A recursive reverse.

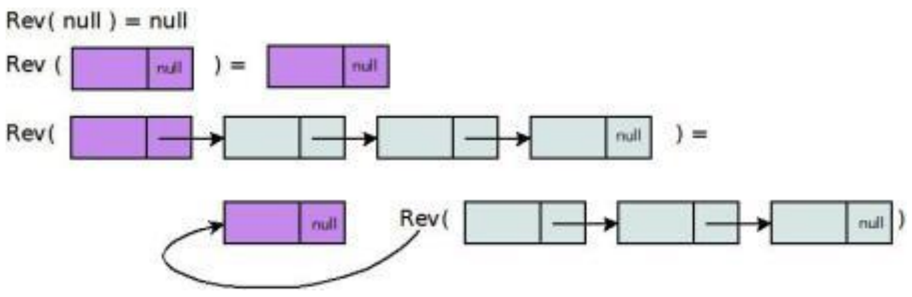
1. Reverse a Single Linked List: Recursive Procedure

The following are the sequence of steps :

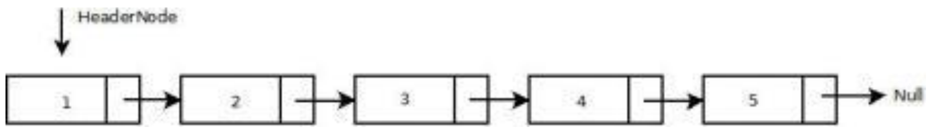
- If the list is empty, then the reverse of the list is also empty
- If the list has one element, then the reverse of the list is the element itself

If the list has n elements, then the reverse of the complete list is reverse of the list starting from second node followed by the first node element. This step is recursive step

The above mentioned steps can be described pictorially as shown below:

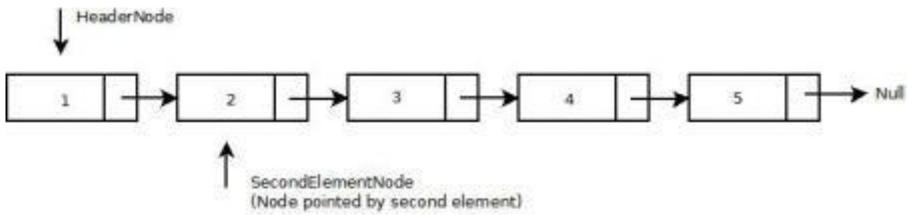


Consider the following linked list that needs to be reversed:

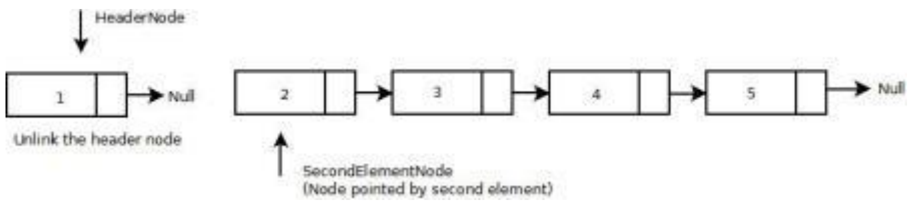


Example:

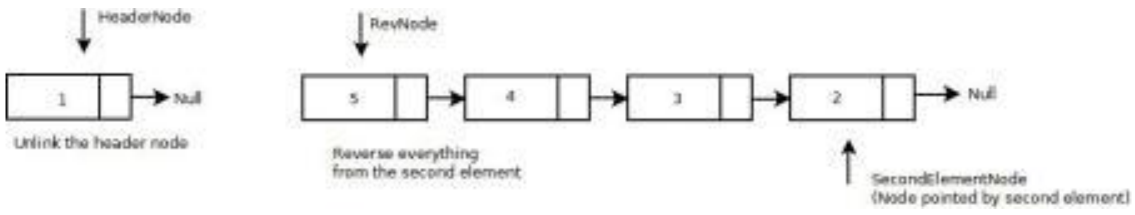
Take a pointer called SecondElementNode, which points to the second element of the list. Here the SecondElementNode points to 2.



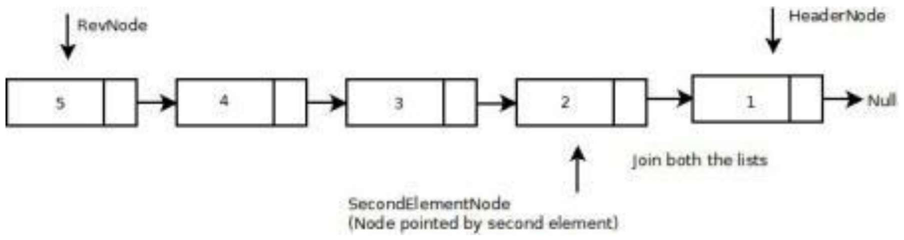
Now we need to unlink the node pointed by HeaderNode. This step is to avoid cycle.



Reverse the list pointed by SecondElementNode recursively.


HeaderNode
1
Null
Unlink the header node
RevNode
5
4
3
2
Null
Reverse everything from the second element
SecondElementNode
(Node pointed by second element)

Now we have to append unlinked HeaderNode to the reversed list.



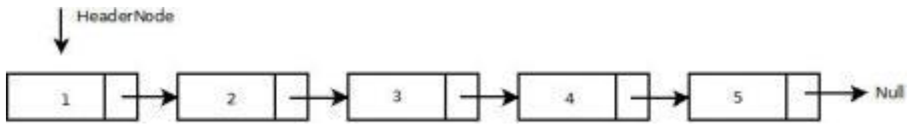
```
public Node reverseRecursivly(Node headerNode) {
            // Reverse of a empty list or null list is null
            if (headerNode == null) {
                  return null;
            }

            // Reverse of a single element list is the list with that
            element
            if (headerNode.next == null) {
                  return headerNode;
            }

            // Reverse of n element list is reverse of the second element
            followed
            // by first element

            // Get the list node pointed by second element
            Node secondElementNode = headerNode.next;

            // Unlink the first element
            headerNode.next = null;

            // Reverse everything from the second element
            Node revNode = reverseRecursivly(secondElementNode);

            // Now we join both the lists
            secondElementNode.next = headerNode;

            return revNode;
      }
```

2. Reverse a Single Linked List: Iterative Procedure

The following are the sequence of steps:

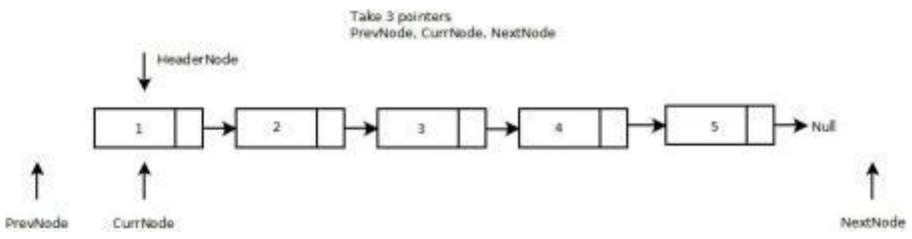
- Initially take three pointers: PrevNode, CurrNode, NextNode
- Let CurrNode point to HeaderNode of the list. And let PrevNode and NextNode points to null
- Now iterate through the linked list until CurrNode is null
- In the loop, we need to change NextNode to PrevNode, PrevNode to CurrNode and CurrNode to NextNode

Consider the following linked list that needs to be reversed:



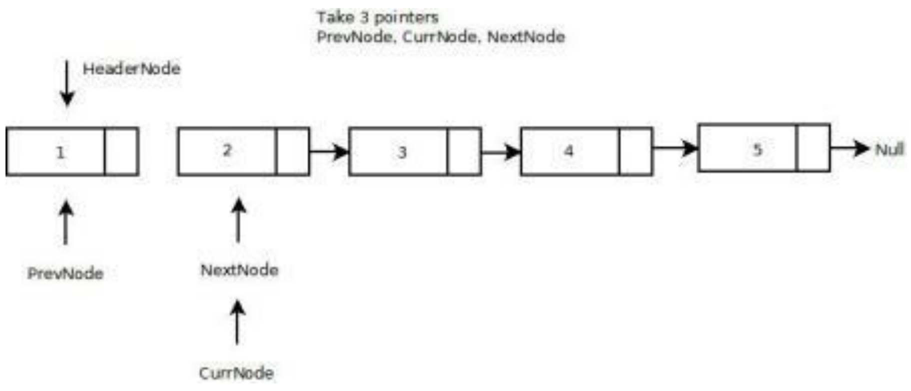
<u>Example:</u>

Taking 3 pointers: PrevNode, CurrNode and NextNode where CurrNode pointing to HeaderNode



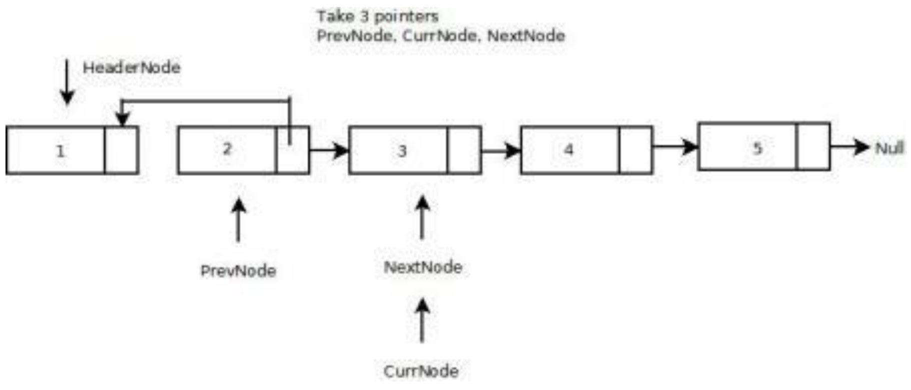
After First Iteration:

After the first iteration of the loop, PrevNode points to the node containing element 1 and CurrNode & NextNode points to the node containing element 2. And the node pointed by PrevNode gets unlinked.

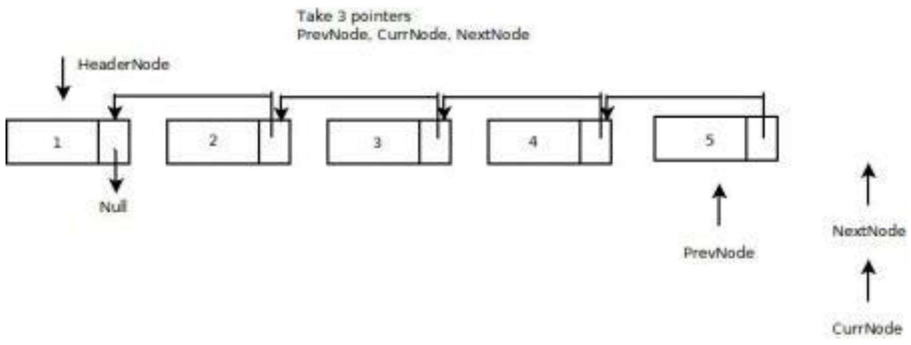


After Second Iteration:

After the second iteration of the loop, PrevNode Points to the node containing element 2 and CurrNode & NextNode point to the node containing element 3. And the CurrNode next would be pointing to PrevNode.

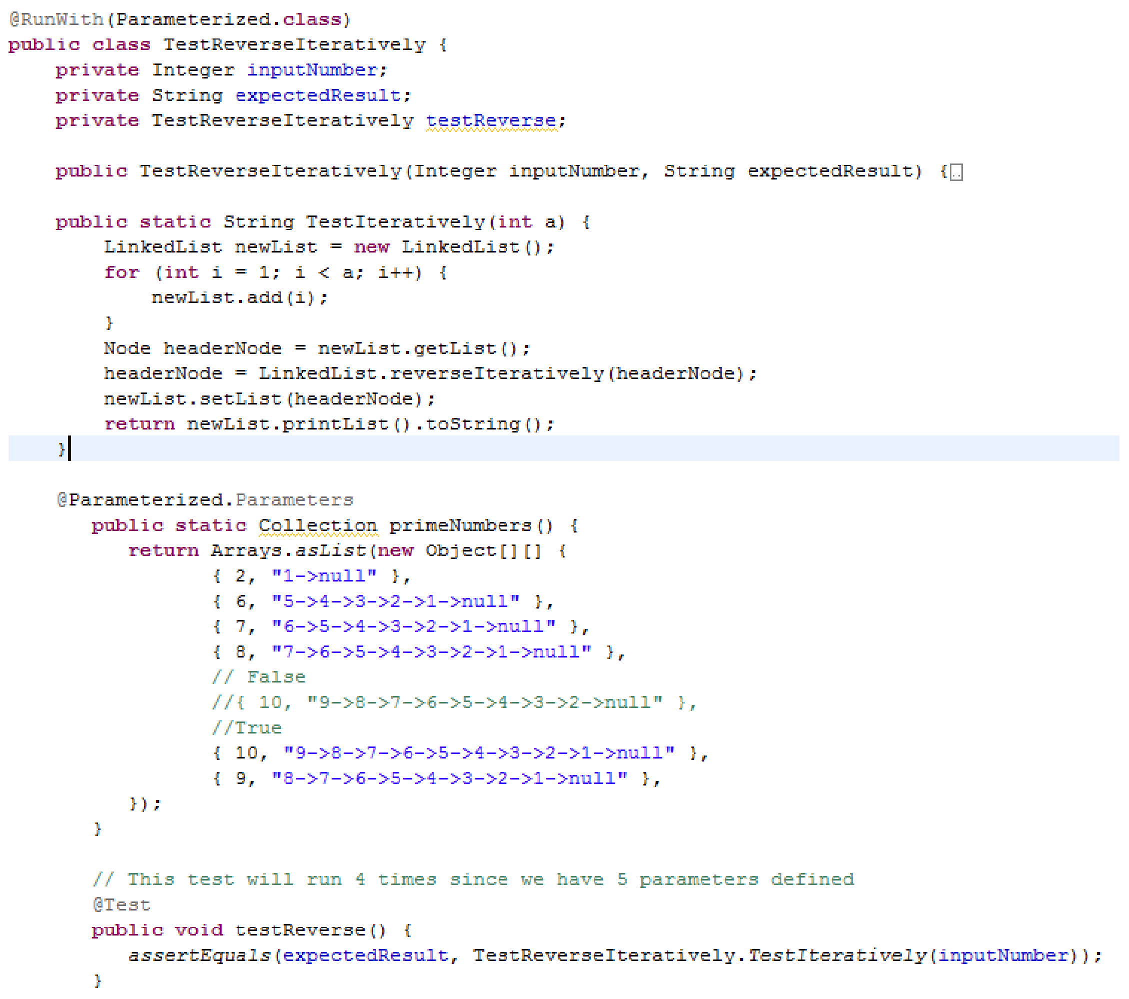


By the end of the iteration, PrevNode contains the reverse of the complete list.



```
public Node reverseIteratively(Node headerNode) {
      Node prevNode = null;
      Node currNode = headerNode;
      Node nextNode = null;

      while (currNode != null) {
            nextNode = currNode.next;
            currNode.next = prevNode;
            prevNode = currNode;
            currNode = nextNode;
      }

      return prevNode;
}
```

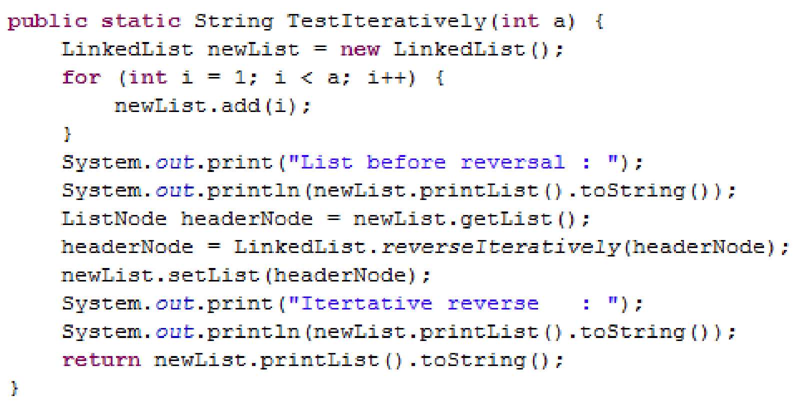
TestReverseIteratively.java



```java
@RunWith(Parameterized.class)
public class TestReverseIteratively {
    private Integer inputNumber;
    private String expectedResult;
    private TestReverseIteratively testReverse;

    public TestReverseIteratively(Integer inputNumber, String expectedResult) {

    public static String TestIteratively(int a) {
        LinkedList newList = new LinkedList();
        for (int i = 1; i < a; i++) {
            newList.add(i);
        }
        Node headerNode = newList.getList();
        headerNode = LinkedList.reverseIteratively(headerNode);
        newList.setList(headerNode);
        return newList.printList().toString();
    }

    @Parameterized.Parameters
       public static Collection primeNumbers() {
          return Arrays.asList(new Object[][] {
                 { 2, "1->null" },
                 { 6, "5->4->3->2->1->null" },
                 { 7, "6->5->4->3->2->1->null" },
                 { 8, "7->6->5->4->3->2->1->null" },
                 // False
                 //{ 10, "9->8->7->6->5->4->3->2->null" },
                 //True
                 { 10, "9->8->7->6->5->4->3->2->1->null" },
                 { 9, "8->7->6->5->4->3->2->1->null" },
          });
       }

       // This test will run 4 times since we have 5 parameters defined
       @Test
       public void testReverse() {
          assertEquals(expectedResult, TestReverseIteratively.TestIteratively(inputNumber));
       }
```

Int InputNumber: the length of the list

Example:



```java
public static String TestIteratively(int a) {
    LinkedList newList = new LinkedList();
    for (int i = 1; i < a; i++) {
        newList.add(i);
    }
    System.out.print("List before reversal : ");
    System.out.println(newList.printList().toString());
    ListNode headerNode = newList.getList();
    headerNode = LinkedList.reverseIteratively(headerNode);
    newList.setList(headerNode);
    System.out.print("Itertative reverse   : ");
    System.out.println(newList.printList().toString());
    return newList.printList().toString();
}
```

The **inputNumber it's** : **a**

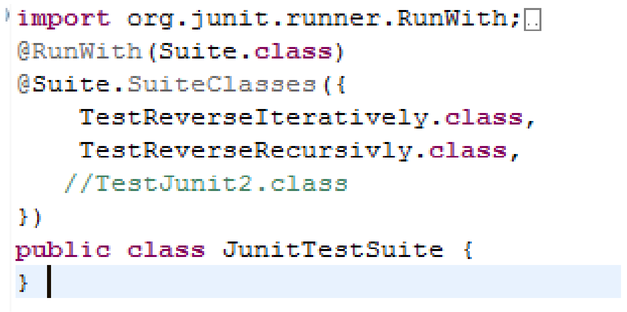
String ExpectedResult: **the correct form of the reverse list.**

The **ExpectedResult it’s** : **"5->4->3->2->1->null"** **etc. …**

TestReverseRecursivly.java

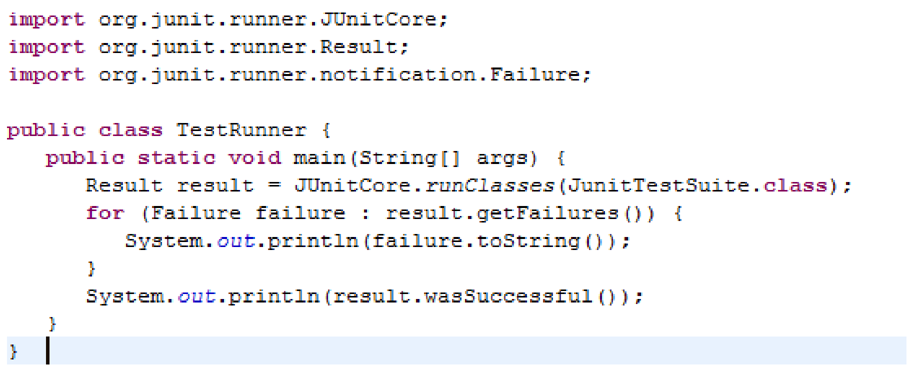
It’s the same for **TestReverseRecursivly.java**

JunitTestSuite.java



```java
import org.junit.runner.RunWith;
@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestReverseIteratively.class,
    TestReverseRecursivly.class,
   //TestJunit2.class
})
public class JunitTestSuite {
}
```

TestRunner.java



```java
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(JunitTestSuite.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

References:

JUnit 4.0

http://www.mkyong.com/tutorials/junit-tutorials/

http://www.vogella.com/articles/JUnit/article.html#junit_intro

http://www.tutorialspoint.com/junit/junit_suite_test.htm