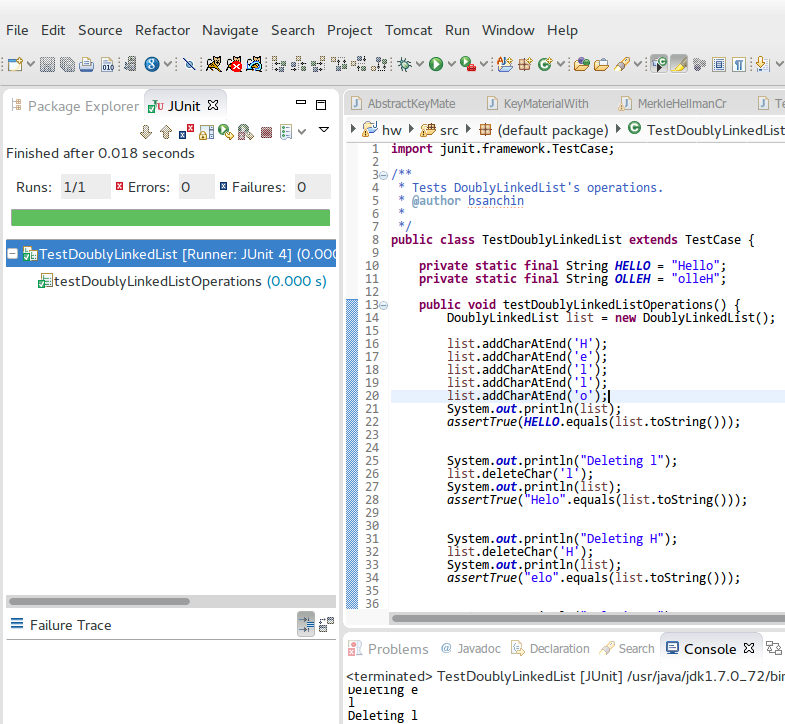
# Part 1. Doubly Linked Lists

DoubleNode and DoublyLinkedList classes are utility classes. If we were to use them in production, we should remove the static main method from them. CMU’s CERT team has published “Secure Coding Guidelines for Java” that [states](https://www.securecoding.cert.org/confluence/display/java/ENV06-J.+Production+code+must+not+contain+debugging+entry+points) “Production code must not contain debugging entry points”. I agree with this guideline and therefore moved static main methods from the utility classes to a dedicated Junit test case.



*Picture 1. TestDoublyLinkedList in action.*

Additional test cases that cover addition and removal after reversals:

|  |
| --- |
| // Some additional test cases below.  list.addCharAtFront('!');  assertTrue(("!" + OLLEH).equals(list.toString()));  list.addCharAtEnd('!');  assertTrue(("!" + OLLEH + "!").equals(list.toString()));  list.reverse();  assertTrue(("!" + HELLO + "!").equals(list.toString()));  list.deleteChar('!');  assertTrue((HELLO + "!").equals(list.toString()));  list.deleteChar('!');  assertTrue(HELLO.equals(list.toString())); |

The source code for the test case:

|  |
| --- |
| import junit.framework.TestCase;  /\*\*  \* Tests DoublyLinkedList's operations.  \* @author bsanchin  \*  \*/  public class TestDoublyLinkedList extends TestCase {    private static final String HELLO = "Hello";  private static final String OLLEH = "olleH";    public void testDoublyLinkedListOperations() {  DoublyLinkedList list = new DoublyLinkedList();    list.addCharAtEnd('H');  list.addCharAtEnd('e');  list.addCharAtEnd('l');  list.addCharAtEnd('l');  list.addCharAtEnd('o');  System.out.println(list);  assertTrue(HELLO.equals(list.toString()));      System.out.println("Deleting l");  list.deleteChar('l');  System.out.println(list);  assertTrue("Helo".equals(list.toString()));      System.out.println("Deleting H");  list.deleteChar('H');  System.out.println(list);  assertTrue("elo".equals(list.toString()));      System.out.println("Deleting o");  list.deleteChar('o');  System.out.println(list);  assertTrue("el".equals(list.toString()));    System.out.println("Deleting e");  list.deleteChar('e');  System.out.println(list);  assertTrue("l".equals(list.toString()));    System.out.println("Deleting l");  list.deleteChar('l');  System.out.println(list);  assertEquals(0, list.countNodes());    list.addCharAtFront('o');  list.addCharAtFront('l');  list.addCharAtFront('l');  list.addCharAtFront('e');  list.addCharAtFront('H');  System.out.println(list);  System.out.println(list.countNodes());  assertTrue(HELLO.equals(list.toString()));    System.out.println("Popping everything");  while(!list.isEmpty()){  System.out.println(list.removeCharFromFront());  }  assertEquals(0, list.countNodes());    list.addCharAtFront('o');  list.addCharAtFront('l');  list.addCharAtFront('l');  list.addCharAtFront('e');  list.addCharAtFront('H');  assertTrue(HELLO.equals(list.toString()));      System.out.println("Popping everything from the end");  while(!list.isEmpty()){  System.out.println(list.removeCharAtEnd());  assertNotSame(HELLO, list.toString());  }  System.out.println(list.countNodes());  assertEquals(0, list.countNodes());    list.addCharAtEnd('o');  list.addCharAtEnd('l');  list.addCharAtEnd('l');  list.addCharAtEnd('e');  list.addCharAtEnd('H');  assertTrue(OLLEH.equals(list.toString()));    list.reverse();  System.out.println(list);  assertTrue(HELLO.equals(list.toString()));    list.reverse();  System.out.println(list);  assertTrue(OLLEH.equals(list.toString()));    // Some additional test cases below.  list.addCharAtFront('!');  assertTrue(("!" + OLLEH).equals(list.toString()));  list.addCharAtEnd('!');  assertTrue(("!" + OLLEH + "!").equals(list.toString()));  list.reverse();  assertTrue(("!" + HELLO + "!").equals(list.toString()));  list.deleteChar('!');  assertTrue((HELLO + "!").equals(list.toString()));  list.deleteChar('!');  assertTrue(HELLO.equals(list.toString()));    }      } |

Program output:

|  |
| --- |
| Hello  Deleting l  Helo  Deleting H  elo  Deleting o  el  Deleting e  l  Deleting l  Hello  5  Popping everything  H  e  l  l  o  Popping everything from the end  o  l  l  e  H  0  Hello  olleH |

# Part 2. Merkle-Hellman Knapsack Cryptosystem

I tried to make a good use abstraction when implementing the crypto system. In my opinion any crypto tool has basically 3 interface methods:

* A method for configuring key material;
* An encryption method;
* And a decryption method.

|  |
| --- |
|  |

*Picture 2. Class diagram of crypto system implementation.*

As you see above picture, we have 2 variations of Crypto system implementation. They are exactly the same except the usage of container/list types they are using.

There are also 2 variations on KeyMaterial implementation:

|  |
| --- |
|  |

*Picture 3. Class diagram of KeyMaterial implementation.*

Finally, the implementation of containers:

* Container type using singly linked lists;
* And another container type using only arrays.

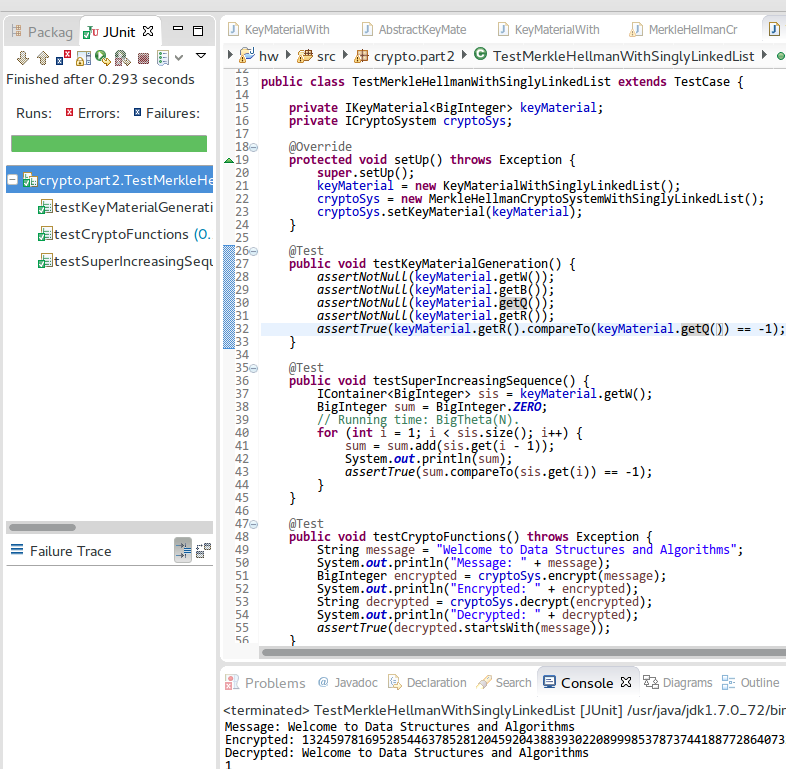
The IContainer interface extends [java.lang.Iterable](http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html) interface. In addition to Iterable’s methods, it also has methods for:

* Adding items to container;
* Getting item at given index;
* Getting size of the container;
* And checking if container is empty or not.

|  |
| --- |
|  |

*Picture 4. Class diagram of container implementation.*

It is easier to make changes/refactoring when we have automated tests (with good coverage). I wrote test cases for both crypto system implementations. Later, it helped a lot when I changed the structure of the classes/methods/implementation for some improvements.



*Picture 5. Automated tests in action.*

Below is a screenshot of actual program in action:

