GEBZE TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

CSE 343 – SOFTWARE ENGINEERING
ONLINE SEARCH ENGINE
Version 1.00

GROUP – 1

FURKAN MUSTAFA AKSOY

EMRULLAH GENÇOĞLU

DENİZ BABAT

MUHAMMED CANER BAKAR

BEKİRCAN AĞAOĞLU

EMRAH KORKMAZ

RIDVAN PORTAKAL

INSTRUCTOR
ASSISTANT PROFESSOR URAZ CENGİZ TÜRKER

12 DECEMBER 2016
GEBZE / KOCAELİ

~~*MODULES*~~

DATABASE:

Rıdvan PORTAKAL
Emrullah GENÇOĞLU

SOFTWARE:

Caner BAKAR
Bekircan AĞAOĞLU
Emrah KORKMAZ
Furkan Mustafa AKSOY
Deniz BABAT

INTERFACE:

Furkan Mustafa AKSOY
Bekircan AĞAOĞLU
Emrah KORKMAZ

Table Of Content

Validation

Validation demonstrates that a software or systems product is fit for purpose. That is, it satisfies all the customer's stated and implied needs.

Component Testing

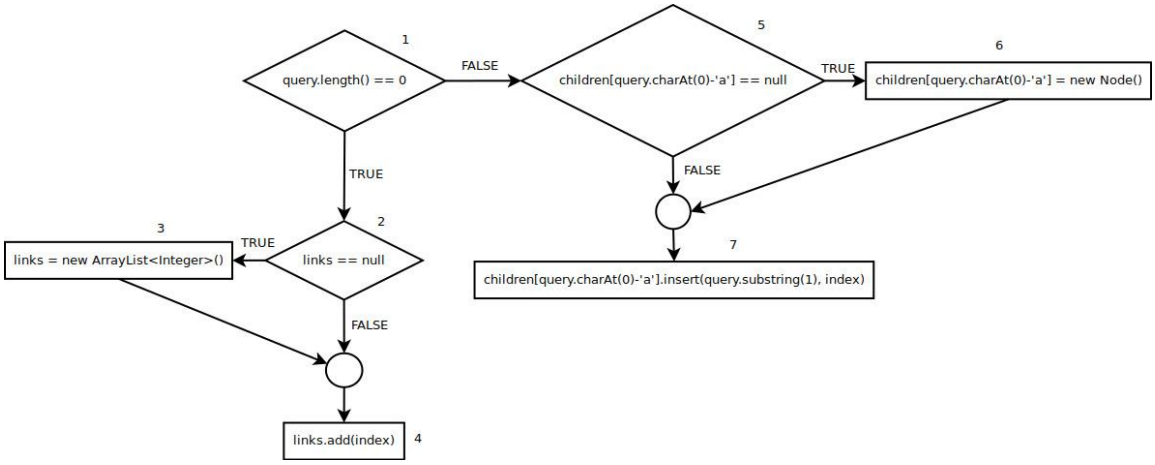
Individual components are tested independently (Unit Testing). Components may be functions or objects or coherent groupings of these entities.

Control Flow

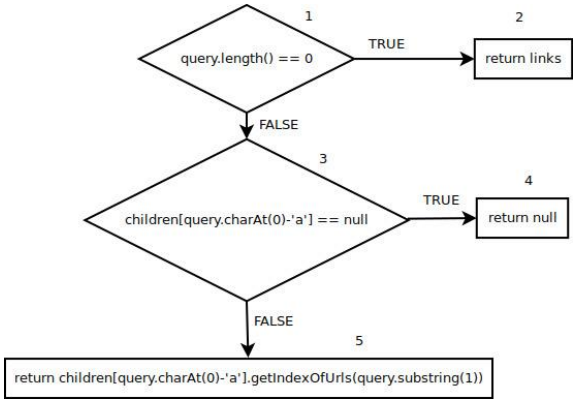
Control Flow testing is a structural testing strategy that uses the program's control flow as a model. It is testing technique that comes white box testing.

Node: It represents one or more procedural statements.
Edges or links: They represent the flow of control in a program.
Decision node: A node with more than one arrow leaving is called a decision node.
Junction node: A node with more than one arrow entering it is called a junction.
Regions: Areas bounded by edges and nodes are called regions.

public void insert(String query, Integer index):



public ArrayList<Integer> getIndexOfUrls(String query):



Unit Test Code

```
package search.engine.cse343;
	import java.util.ArrayList;
	import static org.junit.Assert.*;
	public class UrlTreeTest {
		@org.junit.Test
		public void insert() throws Exception {

			UrlTree tree = new UrlTree();
			UrlTree.Node root = (UrlTree.Node)tree.getRoot();

			//1-2-3-4
			assertNull(root.getLinks()); //2
			tree.insert("", 0); //1
			assertTrue(root.getLinks().size() == 1 && root.getLinks().get(0).equals(0)); //4

			//1-2-4
			assertNotNull(root.getLinks()); //2
			tree.insert("", 0); //1
			assertTrue(root.getLinks().size() == 2
				&& root.getLinks().get(0).equals(0)
				&& root.getLinks().get(1).equals(0)); //4

			//1-5-6-7
			assertNull(root.getChildren()[0]); //5
			tree.insert("a", 0); //1
			assertNotNull(root.getChildren()[0]); //6
			assertTrue(((UrlTree.Node)root.getChildren()[0]).getLinks().size() == 1
				&&((UrlTree.Node)root.getChildren()[0]).getLinks().get(0).equals(0)); //7

			//1-5-7
			assertNotNull(root.getChildren()[0]); //5
			tree.insert("a", 0); //1
			assertTrue(((UrlTree.Node)root.getChildren()[0]).getLinks().size() == 2
				&& ((UrlTree.Node)root.getChildren()[0]).getLinks().get(0).equals(0)
				&& ((UrlTree.Node)root.getChildren()[0]).getLinks().get(1).equals(0)); //7
		}
		@org.junit.Test
		public void getIndexOfUrls() throws Exception {

			// I tested insert method above, so I'm using it for testing getIndexOfUrls method.
			UrlTree tree = new UrlTree();
			UrlTree.Node root = (UrlTree.Node)tree.getRoot();

			//1-2
			assertNull(root.getIndexOfUrls(""));

			root.insert("", 0);
			assertTrue(root.getIndexOfUrls("").size() == 1
			&& root.getIndexOfUrls("").get(0).equals(0));

			//1-3-4
			assertNull(root.getChildren()[0]); //3
			assertNull(root.getIndexOfUrls("a")); //1

			//1-3-5
			root.insert("a",0);
			assertNotNull(root.getChildren()[0]); //3
			ArrayList<Integer> instance = root.getIndexOfUrls("a"); //1
			assertTrue(instance.size() == 1 && instance.get(0).equals(0)); //5
		}
	}
```

Paths

A path through a program is a sequence of statements that starts at an entry, junction or decision ends at another, junction, decision or exit.

Statement Coverage: It is assumed that if all the statements of the module are executed once, every bug will be notified.

Decision Coverage: This criterion states that one must write enough test cases such that each decision has a true and false outcome at least once.

Condition Coverage: In this case, one writes enough test cases such that each condition in a decision takes on all possible outcomes at least once.

public void insert(String query, Integer index):

Path: 1->2->3->4:
Input: insert("", 0)
Expected output : [0] (root)
Output : [0] (root)

Path: 1->2->4
(After first insertion, "links" isn't null anymore)
Input: insert("", 0)
Expected output : [0, 0] (root)
Output : [0, 0] (root)

Path: 1->5->6->7
Input: insert("a", 0)
Expected output : [0] (child)
Output : [0] (child)

Path: 1->5->7
Input: insert("a", 0)
Expected output : [0, 0] (child)
Output : [0, 0] (child)

I tested insert method above, so I'm using it for testing getIndexOfUrls method

public ArrayList<Integer> getIndexOfUrls(String query)

Path: 1->2
Input: getIndexOfUrls("")
Expected output: null
Output: null
insert("", 0);
Input: getIndexOfUrls("")
Expected output: [0]
Output: [0]

Path: 1->3->4
Input: getIndexOfUrls("a") (here, child is null)
Expected output: null
Output: null

Path: 1->3->5
insert("a", 0);
Input: getIndexOfUrls("a")
Expected output: [0]
Output: [0]

Cyclomatic Complexity

Cyclomatic complexity measures the number of independent path through a program's source code. It is computed using CFG.

Formulas Of Cyclomatic Complexity:

- M = (Edges) – (Nodes) + 2(Components)
- M = (Decision Node) + 1
- M = (Enclosed Region) + 1

public void insert(String query, Integer Index):
It has 3 decision nodes. Cyclomatic Complexity for this method is M = 4.

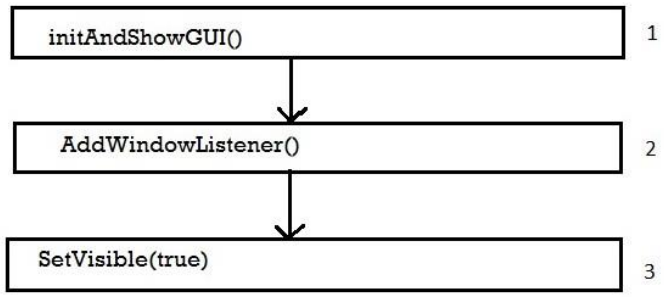
public ListIterator<Integer> getIndexOfUrls(String query):
It has 2 decision nodes. Cyclomatic Complexity for this method is M = 3.

System Testing

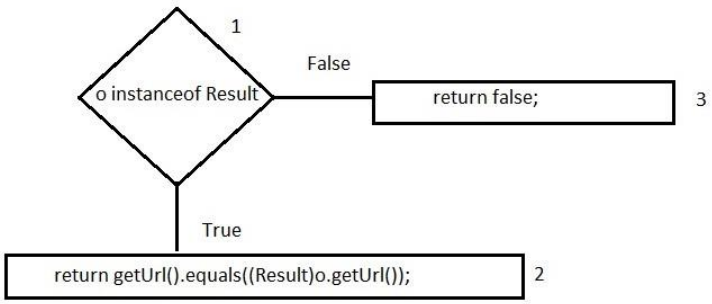
Testing of the system as a whole. Testing of emergent properties is particularly important.

Control Flow

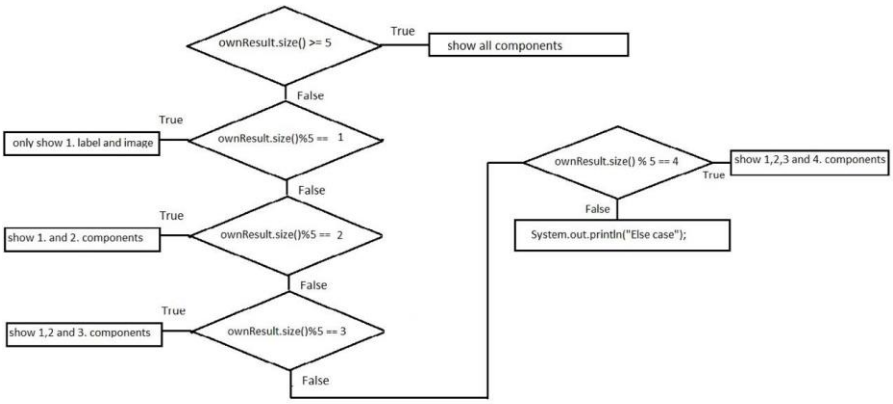
public WebFrame(String searchSite):



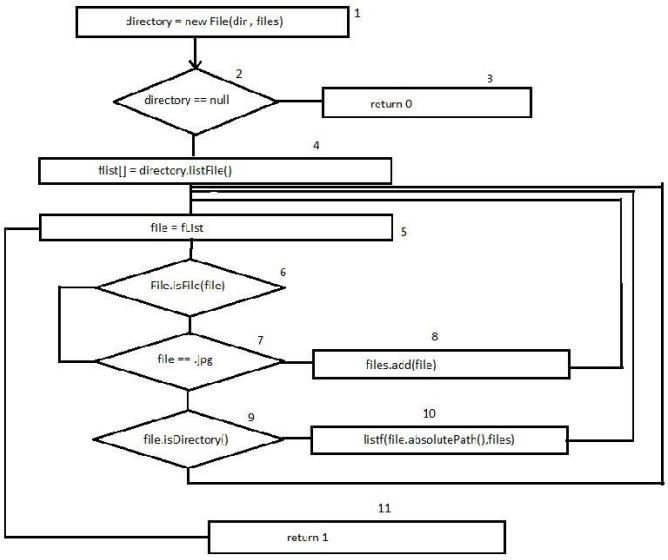
public boolean equals(Object o):



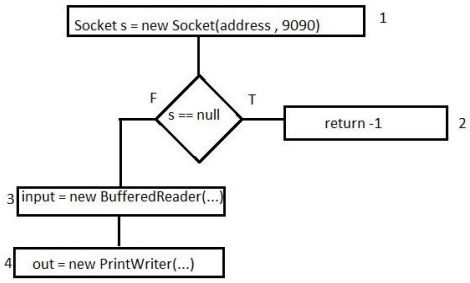
public void checkFirstPage():



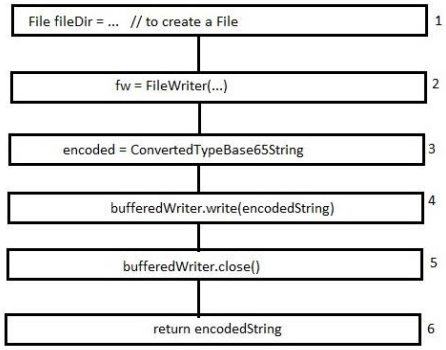
public int listf(String directoryName, ArrayList<File> files):



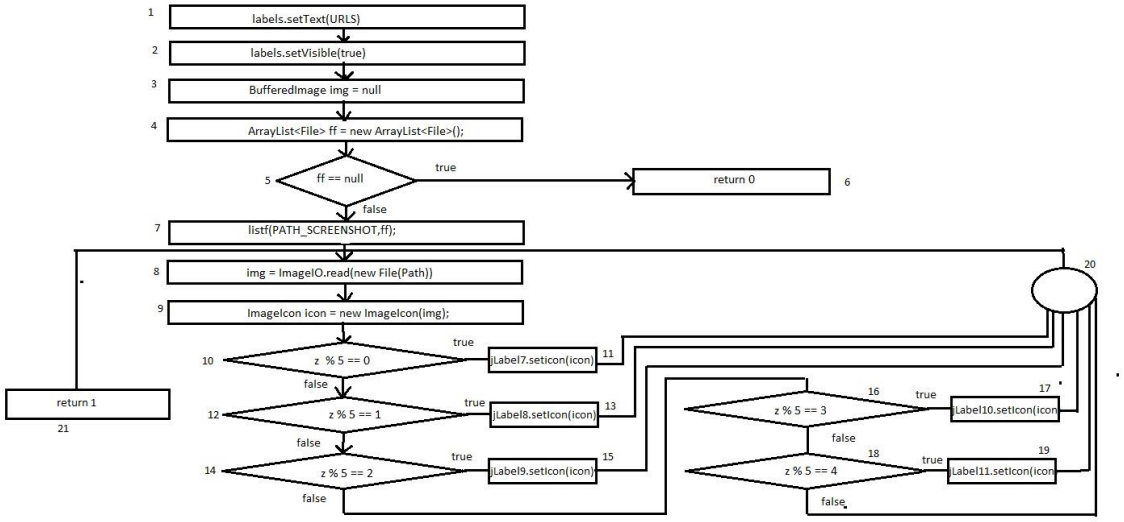
public static int sendRequest(String address):



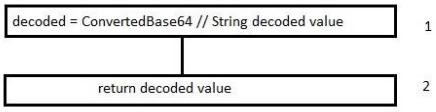
public String toEncode(String str):



public int checkinterPages(int pageInt):



public String toDecode(String str):



Unit Test Code

```
import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.net.URL;
import java.util.ArrayList;
import javax.swing.JFrame;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
```

```
public class NewJFrameTest {

   public NewJFrameTest() {
    }
    @BeforeClass
    public static void setUpClass() {
    }
    @AfterClass
    public static void tearDownClass() {
    }
    @Before
    public void setUp() {
    }
    @After
    public void tearDown() {
    }

    /**
     * Test of sendRequest method, of class NewJFrame.
     */
    @org.junit.Test
    public void testSendRequest() throws Exception {

        System.out.println("sendRequest");
        String address = "127.0.0.1";
        int result=NewJFrame.sendRequest(address);
        int expResult = 0;

        assertEquals(expResult,result);
    }

    /**
     * Test of toEncode method, of class NewJFrame.
     */
    @org.junit.Test
    public void testToEncode() throws Exception {

        System.out.println("toEncode");
        String str = "dmo";
        NewJFrame instance = new NewJFrame();
        String expResult = "ZG1v";
        String result = instance.toEncode(str);

        assertEquals(expResult, result);
    }

     /**
     * Test of checkinterPages method, of class NewJFrame.
     */
    @org.junit.Test
    public void testCheckinterPages() {
        try {
            System.out.println("checkinterPages");
            int pageInt = 0;
            NewJFrame instance = new NewJFrame();

        } catch (IOException ex) {

            Logger.getLogger(NewJFrameTest.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

```
    /**
     * Test of toDecode method, of class NewJFrame.
     */
    @org.junit.Test
    public void testToDecode() throws IOException {

        System.out.println("toDecode");
        String str = "ZG1v";
        NewJFrame instance = new NewJFrame();
        String expResult = "dmo";
        String result = instance.toDecode(str);

        assertEquals(expResult, result);
    }

    /**
     * Test of listf method, of class NewJFrame.
     */
    @org.junit.Test
    public void testListf() throws IOException {

        System.out.println("listf");
        String directoryName = "filesendtestfolder1";
        ArrayList<File> ff=new ArrayList<File>();
        NewJFrame instance = new NewJFrame();
        int result = instance.listf(directoryName, ff);
        int expectedResult=1;

        assertEquals(result,expectedResult);
    }
}
```

Paths

public WebFrame(String searchSite):

Path: 1->2->3:
Input: WebFrame("http://www.google.com.tr")
Expected output : to showing Google Page
Output : showed Google Page

public boolean equals(Object o):

Path: 1->2:
Input: resultA.equals(resultB)
Expected output : true
Output : true

Path: 1->3:
Input: resultA.equals(null)
Expected output : false
Output : false

public void checkFirstPage():

Path: 1->2:
Input: no_input
Expected output : show all components
Output : showed all components

Path: 1->2->3->4:
Input: no_input
Expected output : show 1. object components
Output : showed 1. object components

Path: 1->2->3->4->5->6:
Input: no_input
Expected output : show 1 and 2. object components
Output : showed 1 and 2. object components

Path: 1->2->3->4->5->6->7->8:
Input: no_input
Expected output : show 1,2 and 3. object components
Output : showed 1,2 and 3. object components

Path: 1->2->3->4->5->6->7->8->9->10:
Input: no_input
Expected output : show 1,2,3 and 4. object components
Output : showed 1,2,3 and 4. object components

Path: 1->2->3->4->5->6->7->8->9->11:
Input: no_input
Expected output : print("Else Case")
Output : printed "Else Case"

public int listf(String directoryName, ArrayList<File> files):

Path: 1->2->3:
Input: directoryName, fileArrayList
Expected output : 0
Output : 0

Path: 1->2->3->4->5->11:
Input: directoryName, fileArrayList
Expected output : 1
Output : 1

Path: 1->2->3->4->5->6->7->8->5->11:
Input: directoryName, fileArrayList
Expected output : 1
Output : 1

Path: 1->2->3->4->5->6->7->9->10->5->11:
Input: directoryName, fileArrayList
Expected output : 1
Output : 1

Path: 1->2->3->4->5->6->7->9->5->11:
Input: directoryName, fileArrayList
Expected output : 1
Output : 1

public static int sendRequest(String address):

Path: 1->2:
Input: "127.10.10.1", 9090
Expected output : -1
Output : -1

Path: 1->3->5:
Input: "127.0.0.1", 9090
Expected output : working input and output values
Output : worked input and output values

public String toEncode(String str):

Path: 1->2->3->4->5->6:
Input: str
Expected output : return to encoded String of str
Output : returned encoded String of str

public int checkinterPages(int pageInt):

Path 1->2->3->4->5->6
Input : no_input
Expected Output: to return 0
Output: returned 0

Path 1->2->3->4->5->7->21
Input : no_input
Expected Output: to return 1
Output: returned 1

Path 1->2->3->4->5->6->-7->8->9->10->11->20->21
Input : no_ input
Expected Output: to return 1
Output: returned 1

Path 1->2->3->4->5->6->7->8->9->10->12->13->20->21
Input : no_input
Expected Output: to return 1
Output: returned 1

Path 1->2->3->4->5->6->7->8->9->10->12->14->15->20->21
Input : no_input
Expected Output: to return 1
Output: returned 1

Path 1->2->3->4->5->6->7->8->9->10->12->14->16->17->20->21
Input : no_input
Expected Output: to return 1
Output: returned 1

Path 1->2->3->4->5->6->7->8->9->10->12->14->16->18->19->20->21
Input : no_input
Expected Output: to return 1
Output: returned 1

Path 1->2->3->4->5->6->7->8->9->10->12->14->16->18->20->21
Input : no_input
Expected Output: to return 1
Output: returned 1

public String toDecode(String str):

Path: 1->2:
Input: “Test String”
Expected output : decoded str value
Output : Zq2E

Cyclomatic Complexity

public WebFrame(String searchSite):
It has no decision node. Cyclomatic Complexity for this method is M = 1.

public boolean equals(Object o):
It has 1 decision nodes. Cyclomatic Complexity for this method is M = 2.

public void checkFirstPage():
It has 5 decision nodes. Cyclomatic Complexity for this method is M = 6.

public int listf(String directoryName, ArrayList<File> files):
It has 4 decision nodes. Cyclomatic Complexity for this method is M = 5.

public static int sendRequest(String address):
It has 1 decision nodes. Cyclomatic Complexity for this method is M = 2.

public String toEncode(String str):
It has no decision node. Cyclomatic Complexity for this method is M = 1.

public int checkinterPages(int pageInt):
It has 6 decision nodes. Cyclomatic Complexity for this method is M = 7.

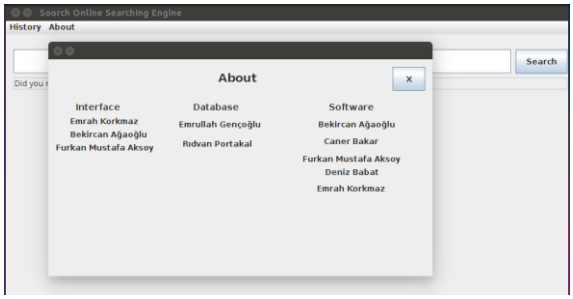
public String toDecode(String str):
It has no decision node. Cyclomatic Complexity for this method is M = 1.

Acceptance Testing

Testing with customer data to check that the system meets the customer's needs.

About

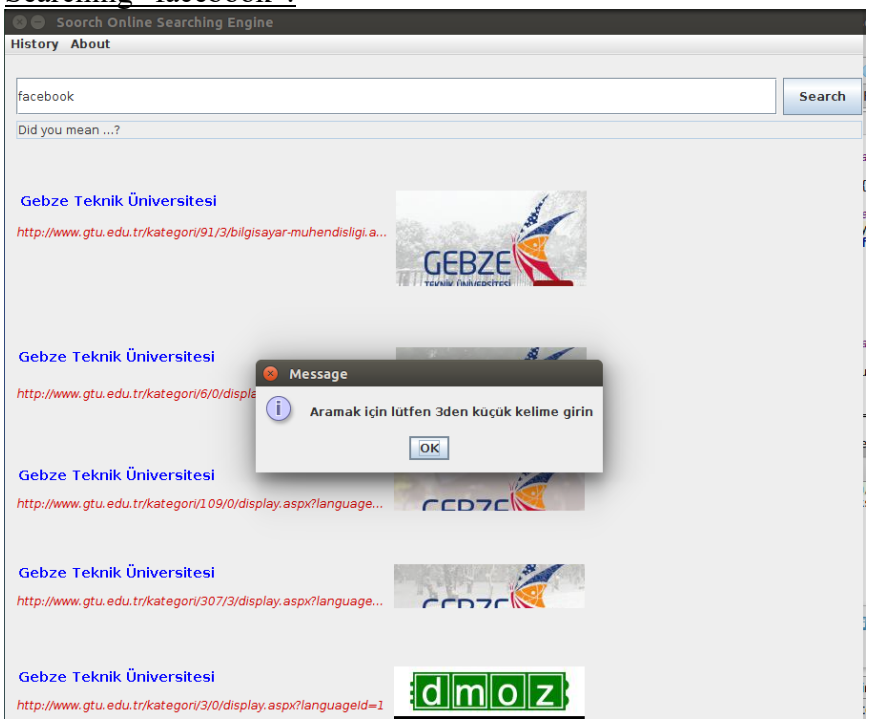
Internet Search Engine Project modules and member information shows to user.



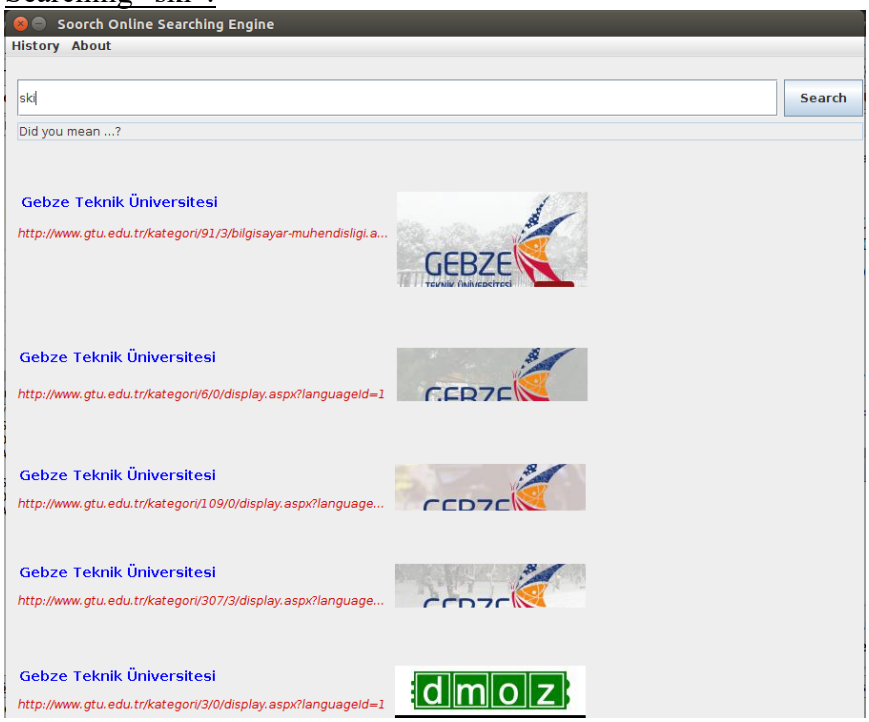
Search

Checking giving string in database (forest tree in server). Result shows page to page. Every page has maximum 5 links and sites screen shots. String that for searching must be 3 characters.

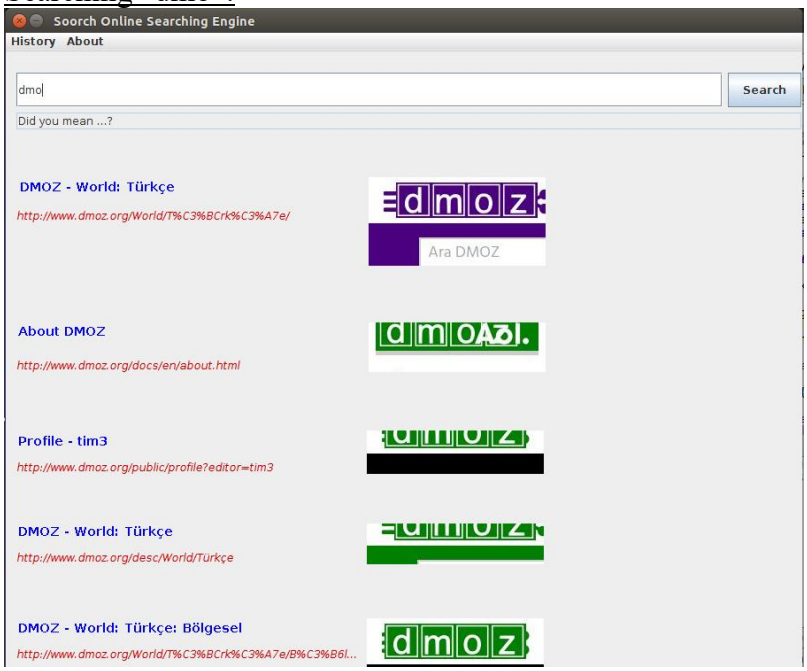
Searching “facebook”:



Searching “ski”:



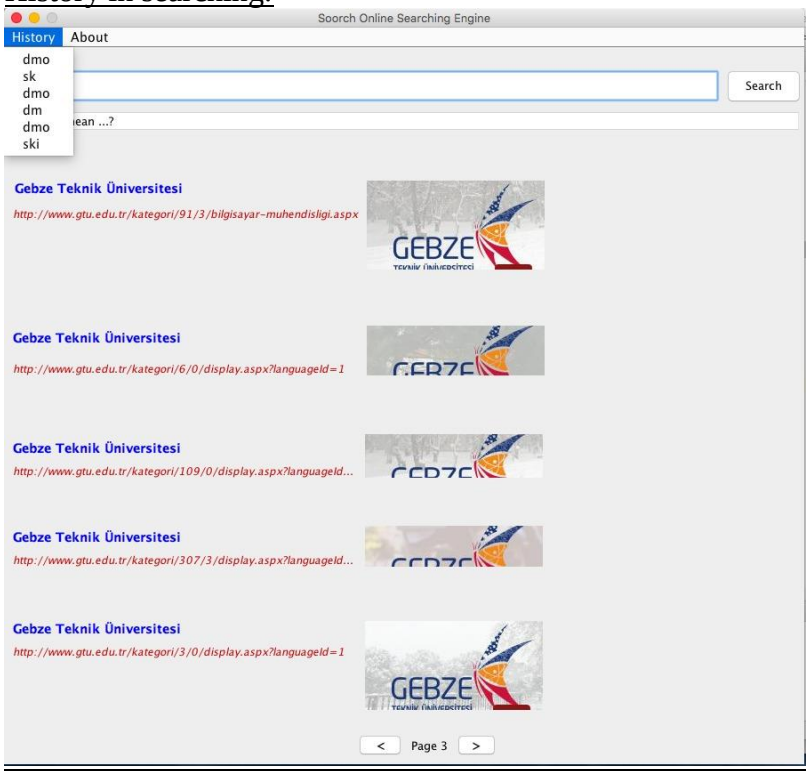
Searching “dmo”:



History

Searched words keep in history log file with encryption. History log file supports saving history when program closing so user can see previous movement and user can click that every time.

History in searching:



History in program start: