PRT 452 SOFTWARE ENGINEERING: PROCESS AND TOOLS

ASSIGNMENT-1

Submitted by

Vokerla Rahul Rao(s300437)

Master of Information Technology (Software Engineering)

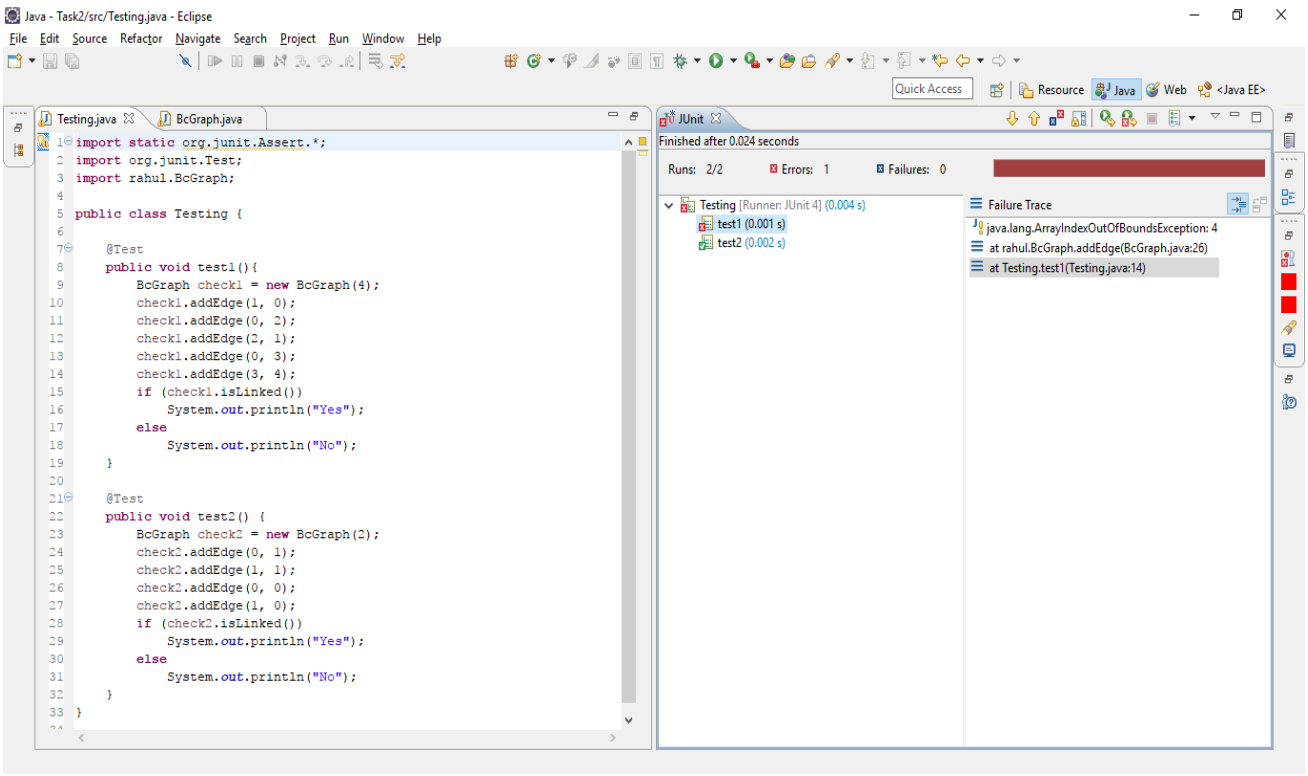
Charles Darwin University

Answer 1:

Created a GitHub repository named as “452”, uploaded programming code and assignment PDF file.

Answer 2:

Junit Test:



Test Fail code:

```
import static org.junit.Assert.*;
import org.junit.Test;
import rahul.BcGraph;

public class Testing {

	@Test
	public void test1(){
		BcGraph check1 = new BcGraph(4);
	  check1.addEdge(1, 0);
	  check1.addEdge(0, 2);
	  check1.addEdge(2, 1);
	  check1.addEdge(0, 3);
	  check1.addEdge(3, 4);
	  if (check1.isLinked())
```

```
            System.out.println("Yes");
        else
            System.out.println("No");
    }
Output: Graph is not connected.
```

Test Pass Code:

```
    @Test
    public void test2() {
          BcGraph check2 = new BcGraph(2);
        check2.addEdge(0, 1);
        check2.addEdge(1, 1);
        check2.addEdge(0, 0);
        check2.addEdge(1, 0);
        if (check2.isLinked())
            System.out.println("Yes");
        else
            System.out.println("No");
    }
```

} (GeeksforGeeks, 2017)

Output: Graph is connected.

Program code:

```
package rahul;
import java.util.*;
import java.util.LinkedList;

public class BcGraph{
 private int V;   // No. of vertices

 // Lists for Adjacency List
 private LinkedList<Integer> adj[];

 int time = 0;
 public static final int NIL = -1;

 public BcGraph(int v)
 {
     V = v;
     adj = new LinkedList[v];
     for (int i=0; i<v; ++i)
         adj[i] = new LinkedList();
 }

 //Adding an edge into the graph
 public void addEdge(int v, int w)
 {
     adj[v].add(w);  //Undirected Graph.
     adj[w].add(v);
 }


 public boolean isLinkedUtil(int u, boolean visited[], int disc[],int low[],
                  int parent[])
```

```
{

    // Count of children
    int children = 0;

    // Visited Node
    visited[u] = true;

    disc[u] = low[u] = ++time;

    // Adjacent Vertices
    Iterator<Integer> i = adj[u].iterator();
    while (i.hasNext())
    {
        int v = i.next();  // v current adjacent of u

        if (!visited[v])
        {
            children++;
            parent[v] = u;

            if (isLinkedUtil(v, visited, disc, low, parent))
                return true;

            // Check connection to an ancestors of u
            low[u]  = Math.min(low[u], low[v]);

            if (parent[u] == NIL && children > 1)
                return true;

            // Low value of its child is more than discovery value of u.
            if (parent[u] != NIL && low[v] >= disc[u])
                return true;
        }

        // Update low value of u for parent function calls.
        else if (v != parent[u])
            low[u]  = Math.min(low[u], disc[v]);
    }
    return false;
}

// Returns true if graph is Connected, else False
public boolean isLinked()
{
    // Mark vertices not visited
    boolean marked[] = new boolean[V];
    int dic[] = new int[V];
    int low[] = new int[V];
    int par[] = new int[V];

    for (int i = 0; i < V; i++)
    {
        par[i] = NIL;
        marked[i] = false;
    }
```

```
        if (isLinkedUtil(0, marked, dic, low, par) == true)
            return false;

        // Graph connected or not.
        for (int i = 0; i < V; i++)
            if (marked[i] == false)
                return false;

        return true;
    }
}
```
(GeeksforGeeks, 2017)

Answer 3:

- **Speculative generality:**

There are some unused class, method, objects or parameter in the code. These fall under speculative generality bad smell. This problem can be solved by removing the unused Class, method, objects or parameter from the code (Sourcemaking.com, 2017).

Following is one the method which found in the apache common IO package which is not used. It is tested and removed from the code.

[IO-235] Tests - remove unused YellOnFlushAndCloseOutputStream from CopyUtilsTest

```
public void testCopy_inputStreamToOutputStream() throws Exception {
    InputStream in = new ByteArrayInputStream(inData);
    in = new YellOnCloseInputStream(in);

    ByteArrayOutputStream baout = new ByteArrayOutputStream();
    OutputStream out = new YellOnFlushAndCloseOutputStream(baout, false, true);

    int count = CopyUtils.copy(in, out);

    assertEquals("Not all bytes were read", 0, in.available());
    assertEquals("Sizes differ", inData.length, baout.size());
    assertTrue("Content differs", Arrays.equals(inData, baout.toByteArray()));
    assertEquals(inData.length, count);
} (Svn.apache.org, 2017)
```

- **Switch Statements:**

Switch statements are bad smells. Whenever a new condition is added, we have to find all the switch cases in the code and modify them. Polymorphism can be used to replace the conditional statements (Sourcemaking.com, 2017).

Following is one the switch case which found in the apache common Logging package.

```
@Override
	public void close() {
		if (sb.length() > 0) {
			logMessage();
		}}
	private void logMessage() {
		if (messagePrefix != null) {
			sb.insert(0, messagePrefix);
		}
		String message = sb.toString();

		switch (logLevel) {
		case TRACE:
			logger.trace(message);
			break;
		case DEBUG:
			logger.debug(message);
			break;
		case INFO:
			logger.info(message);
			break;
		case WARN:
			logger.warn(message);
			break;
		case ERROR:
			logger.error(message);
			break;
		case FATAL:
			logger.fatal(message);
			break;
		}
		sb.setLength(0);
	}
}
```
(Issues.apache.org, 2017)

- **Data Clumps:**
  Some parts of the code contain group of variables together and can be extracted as its own class. These data groups are formed due to poor program structure. Extract lass can be used to solve data clumps which allows the fields to move to their own class. Sometimes the resolving the data clumps may increase the code length but it makes code understandable and organized (Refactoring.guru, 2017).

- **Middle Man:**
  A class delegating it work to other class instead of performing by itself leads to middle man code smell. Here that class remains as empty shell doing no task other than just delegating its own task to other classes. Middle man code smell may also occur due to removing message chains (Refactoring.guru, 2017).

- **Message chains:**

  Message chains occur when a user request an object, that object requests another object and so on. Hide delegate is the technique use to solve the message chains (Refactoring.guru, 2017).

Appendices:

GitHub repository link: https://github.com/Vokerla/452

References:

1. Svn.apache.org. (2017). *[Apache-SVN] Contents of /commons/proper/io/trunk/src/test/org/apache/commons/io/CopyUtilsTest.java*. [online] Available at: http://svn.apache.org/viewvc/commons/proper/io/trunk/src/test/org/apache/commons/io/CopyUtilsTest.java?view=markup&pathrev=982430 [Accessed 25 Aug. 2017].
2. GeeksforGeeks. (2017). *Biconnected graph - GeeksforGeeks*. [online] Available at: http://www.geeksforgeeks.org/biconnectivity-in-a-graph/ [Accessed 25 Aug. 2017].
3. Sourcemaking.com. (2017). *Design Patterns and Refactoring*. [online] Available at: https://sourcemaking.com/refactoring/smells/speculative-generality [Accessed 25 Aug. 2017].
4. Issues.apache.org. (2017). *Cite a Website - Cite This For Me*. [online] Available at: https://issues.apache.org/jira/secure/attachment/12582595/LoggingOutputStream.java [Accessed 27 Aug. 2017].
5. Sourcemaking.com. (2017). *Design Patterns and Refactoring*. [online] Available at: https://sourcemaking.com/refactoring/smells/switch-statements [Accessed 27 Aug. 2017].
6. Refactoring.guru. (2017). *Data Clumps*. [online] Available at: https://refactoring.guru/smells/data-clumps [Accessed 27 Aug. 2017].
7. Refactoring.guru. (2017). Middle Man. [Online] Available at: https://refactoring.guru/smells/middle-man [Accessed 27 Aug.2017].