

# Validate UML

Name: Asala Ahmed Saleh ID : 6916

Name: Nadine Ayman Hassan ID : 7130

Name: Jera Gamal Lamey ID : 6914

- **Classes:**

- 1-LinkValidation:

- Link Validation includes 3 functions:

- Validate Single URL function which checks whether the link is valid or not .

```
public static boolean validateSingleURL(String link) throws IOException {  
    boolean valid = false;  
    try {  
        Document doc = (Document) Jsoup.connect(link).get();  
        valid = true;  
    } catch (HttpStatusException ex) {  
        valid = false;  
        System.err.println("PAGE NOT FOUND");  
    } catch (IOException ex) {  
        valid = false;  
        System.err.println("Invalid Link : " + link);  
    } catch (Exception ex) {  
    }  
    return valid;  
}
```

- Link Validation function which takes the threads number as an input from the user so that the executer could take it as an input.

```
public LinkValidation(int threadsno) {  
    System.out.println("Number Of Threads : " + threadsno);  
    es = Executors.newFixedThreadPool(threadsno);  
}
```

- And validate URL function which extracts the links included in the link sent to the function according to the maximum depth given and checks whether each extracted link is valid or not by calling validate Single URL function.

```
public static void validateURL(String link, int currentDepth, int totalDepth) throws IOException, InterruptedException {
    Threads t1 = null;
    if (validateSingleURL(link)) {
        System.out.println("Valid Link : " + link );
        count++;
        if (currentDepth == totalDepth) {
            return;
        }
        Document doc = Jsoup.connect(link).get();
        Elements e = doc.select("a[href]"); // extract only links
        URL u = new URL(link);
        System.out.println("Protocol: " + u.getProtocol());
        System.out.println("Host: " + u.getHost());
        String baseLINK = u.getProtocol() + "://" + u.getHost();
        System.out.println("BASE URL : " + baseLINK);
        for (int i = 0; i < e.size(); i++) {
            String x = e.get(i).attr("href");
            e.get(i).text();
            System.out.println("Text: " + e.get(i).text());
            if (!x.startsWith("http")) {
                x = baseLINK + x;
            }

            t1 = new Threads(x, currentDepth + 1, totalDepth);
            es.execute(t1);
        }
    } else {
        counter++;
    }
}
```

## 2-Threads:

It includes the link, current depth and maximum depth as private data and includes a constructor that takes the link, current depth and maximum depth.

```
public class Threads extends Thread {  
  
    private String link;  
    private int depth;  
    private int maxDepth;  
  
    public Threads(String link, int depth, int maxDepth) {  
        this.link = link;  
        this.depth = depth;  
        this.maxDepth = maxDepth;  
    }  
  
    @Override  
    public void run() {  
        try {  
            try {  
                LinkValidation.validateURL(link, depth, maxDepth);  
            } catch (InterruptedException ex) {  
                Logger.getLogger(Threads.class.getName()).log(Level.SEVERE, null, ex);  
            }  
        } catch (IOException ex) {  
            Logger.getLogger(Threads.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
}
```

### 3-Main Frame:

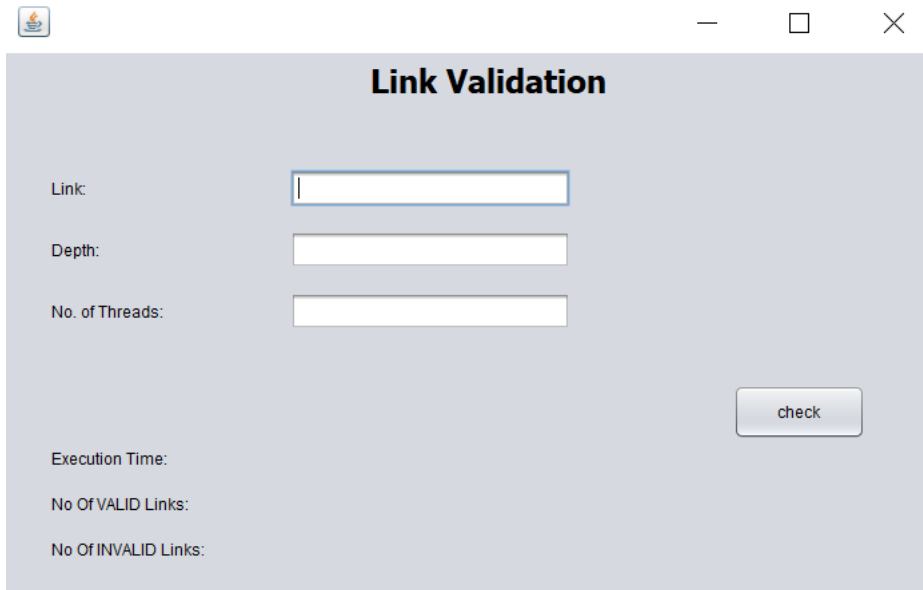
It includes the gui so that it could take the link, current depth, maximum depth as an input from the user, includes the function that calculates the execution time and print the execution time, number of valid links and number of invalid links in the gui.

```
String link = jTextField1.getText();
String totalDepth = jTextField2.getText();
String threads = jTextField3.getText();
int currentDepth = 0;
double start = currentTimeMillis();
int dep = 0;
int k = 0;
int threadno = 0;
try {
    threadno = Integer.parseInt(threads);
    dep = Integer.parseInt(totalDepth);
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(null, "Invalid Input");
}
LinkValidation E = new LinkValidation(threadno);
try {
    LinkValidation.validateURL(link, currentDepth, dep);
} catch (InterruptedException ex) {
    Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
}
while (((ThreadPoolExecutor) LinkValidation.es).getActiveCount() > 0) {
}
LinkValidation.es.shutdown();

double end = currentTimeMillis();
double time = (end - start) / 1000;
jLabel5.setText("Execution Time:" + time);
jLabel6.setText("Number Of VALID Links:" + LinkValidation.count);
jLabel7.setText("Number Of INVALID Links:" + LinkValidation.counter);
```

- **Sample Run:**

First of all the gui frame appears



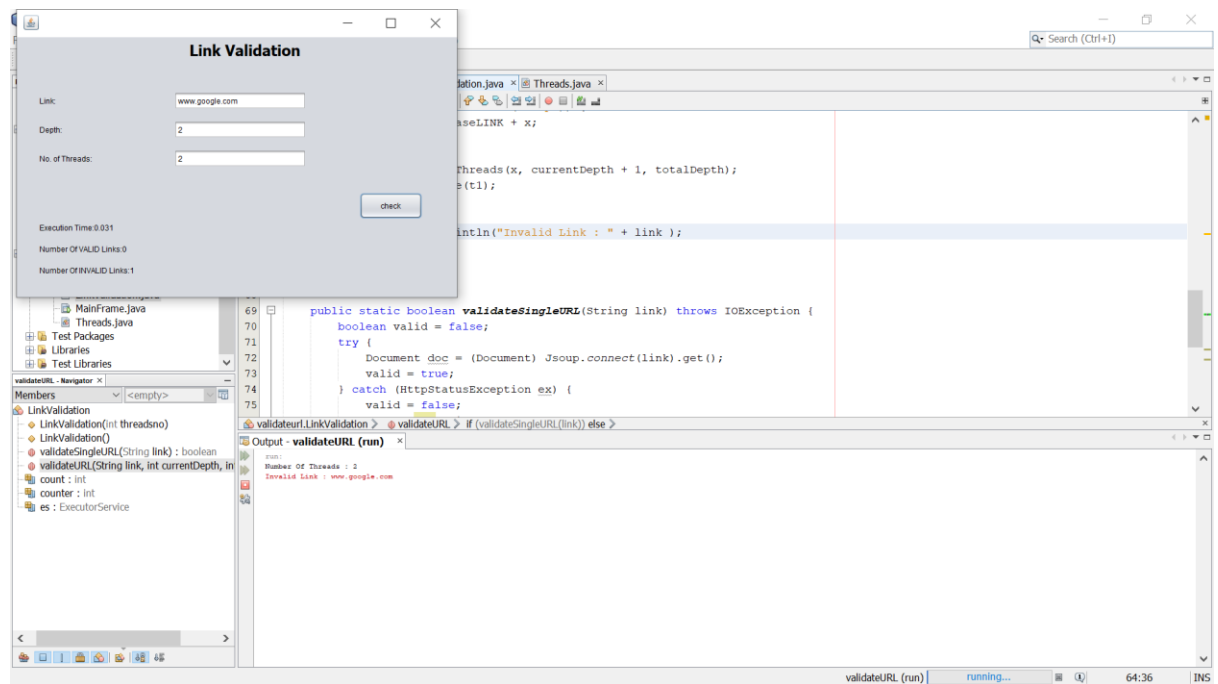
The screenshot shows a window titled "Link Validation". It contains three input fields: "Link:", "Depth:", and "No. of Threads:". To the right of these fields is a "check" button. Below the input fields, there are three labels: "Execution Time:", "No Of VALID Links:", and "No Of INVALID Links:". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button.

If the depth or number of threads is not an integer a message appears to the user that the input is invalid.

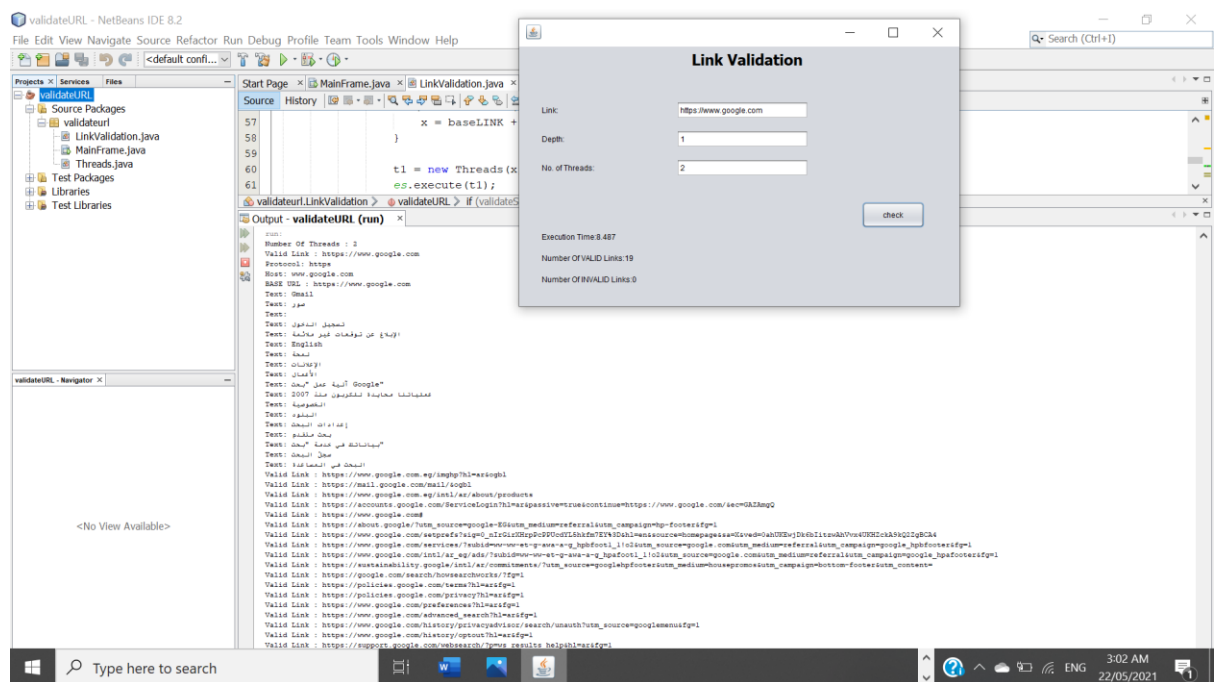


The screenshot shows the same "Link Validation" window, but with an error message displayed. The "Link:" field contains "https://www.google.com", the "Depth:" field contains "as", and the "No. of Threads:" field contains "1". A "Message" dialog box is overlaid on the window, displaying an information icon and the text "Invalid Input". The "check" button is visible to the right of the dialog box. The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button.

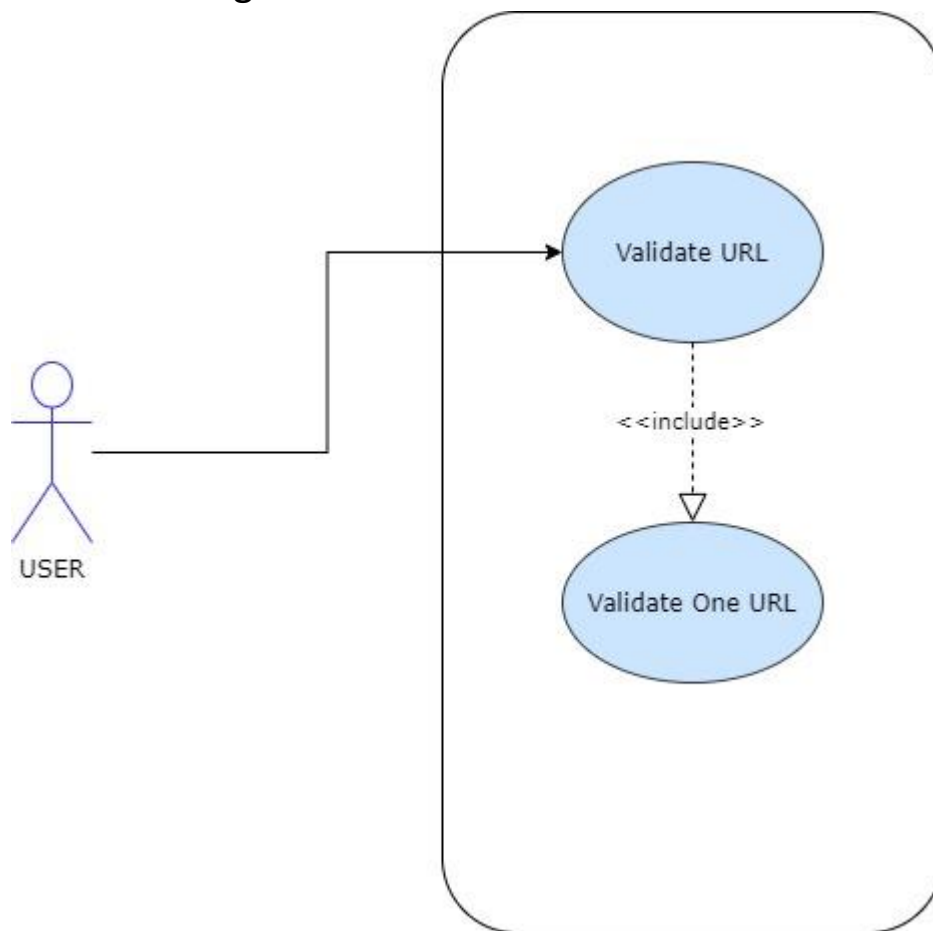
If the link is invalid



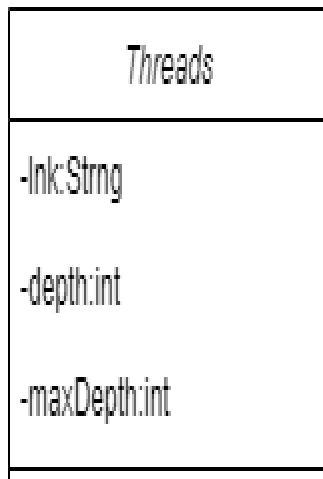
If the link is valid



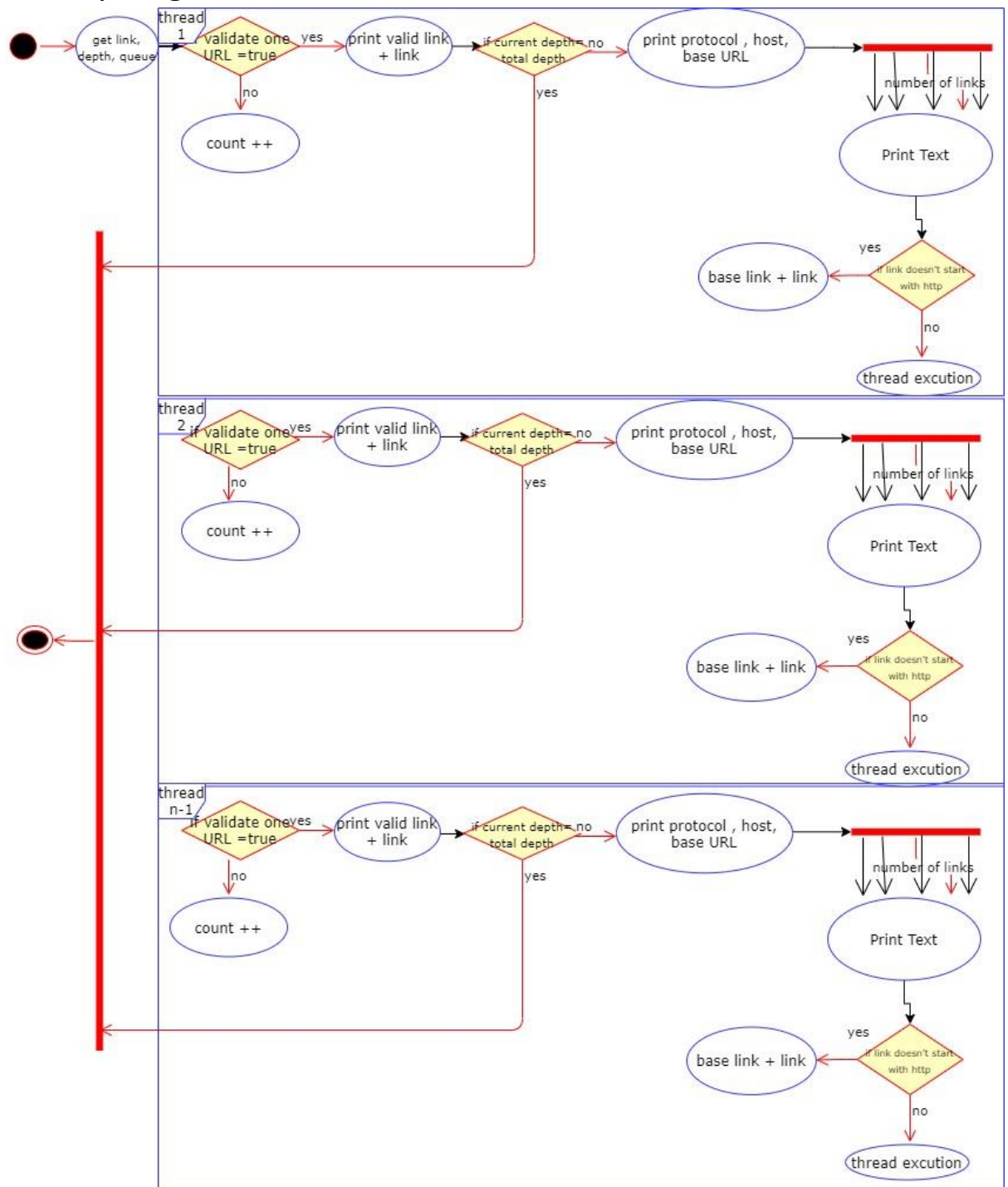
## Use Case Diagram:



## Class Diagram



## Activity Diagram:





## Sequential Diagram:

