

OOP Project – 19

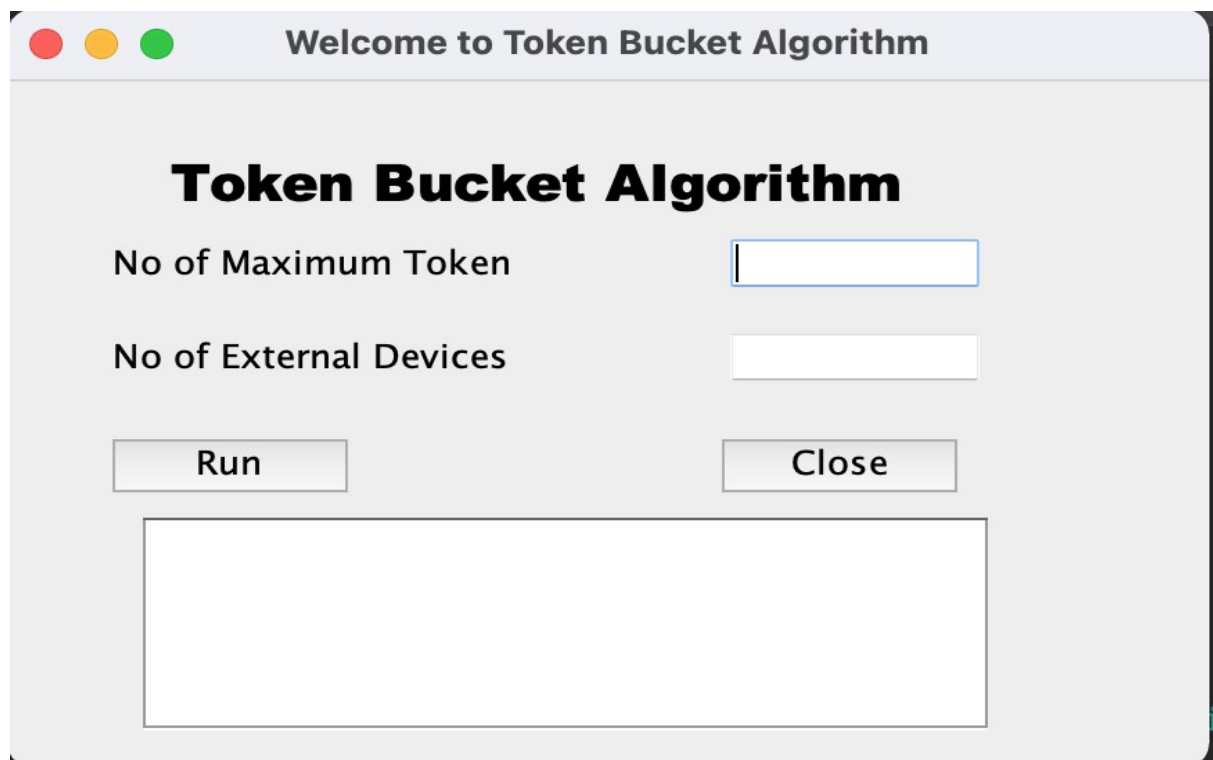
Token Bucket Algorithm

Submitted to:-

Dr. Amit Dua
Dr. Amitesh Singh Rajput

Submitted By:-

Kshitij Gupta(2019A3PS0212P)
Vividh Vivek Raj(2019A8PS0361P)



Welcome to Token Bucket Algorithm

Token Bucket Algorithm

No of Maximum Token

No of External Devices

Drive Link for simulation video :

<https://drive.google.com/drive/folders/187XTk2zn9-VmbP8di6tVVkd89ynpQmnk?usp=sharing>

GitHub Link: <https://github.com/kshitijgupta01/Token-Bucket-Algorithm>

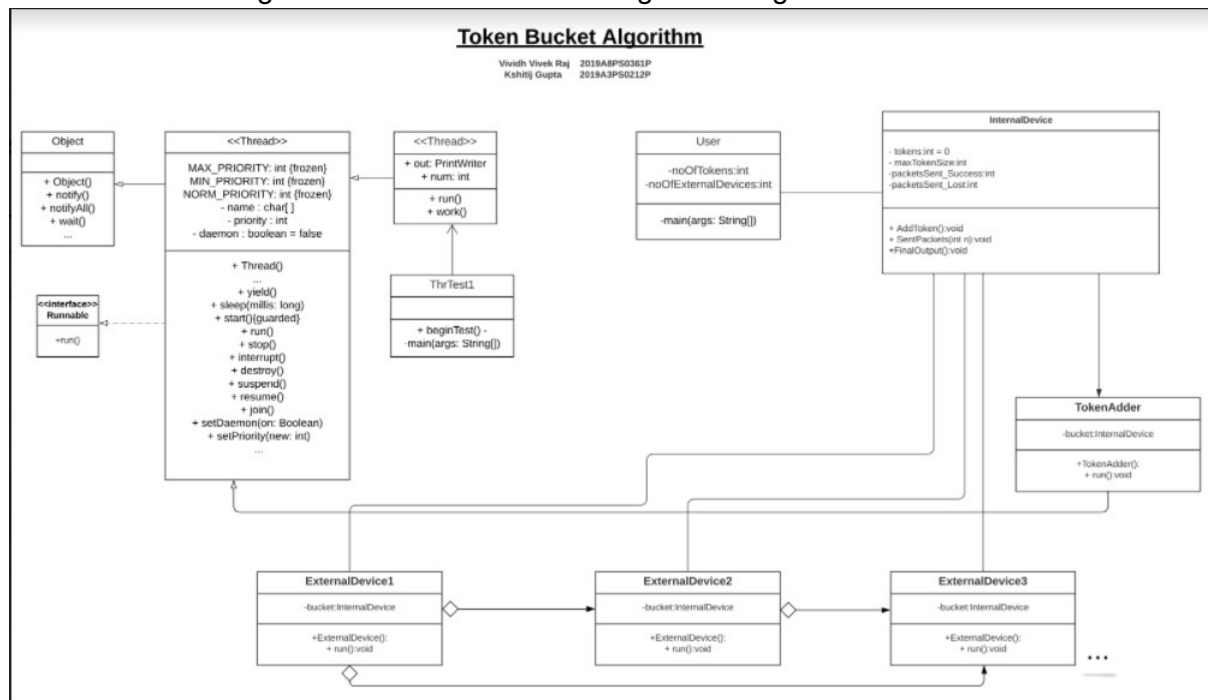
Acknowledgement

We would like to thank Dr. Amit Dua and Dr. Amitesh for this excellent opportunity to pursue this Object Oriented Project on Token Bucket Algorithm under their guidance. We are grateful for having this opportunity.

It is our radiant sentiment to place on record our best regards, deepest sense of gratitude to Dr. Amit Dua and Dr. Amitesh for their guidance.

Without the help and contribution of them, this project would not have been a success.

The UML Class Diagram for the Token Bucket Algorithm is given below:



The GUI Class is used for preparing the GUI for the Token Bucket Algorithm.

```
public class GUIInterface extends JFrame {

    private JPanel contentPane;
    private JTextField t1;
    private JTextField t2;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    GUIInterface frame = new GUIInterface();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

When the **Run** Button of the GUI Class is pressed, the input of

1. Number of external devices to be there that transmits simultaneously.
 2. Number of tokens assigned to intermediate devices
- Is taken.

An object of Internal Device is created to pass as reference in the TokenAdder Class and the External Device Class.

```

JButton btnNewButton = new JButton("Run");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int token = Integer.parseInt(t1.getText());
        int devices = Integer.parseInt(t2.getText());
        InternalDevice b = new InternalDevice(token);
        TokenBucketAlgorithmUser user = new TokenBucketAlgorithmUser(token, devices);

        user.runThreads(b);
    }
});
btnNewButton.setBounds(38, 142, 85, 21);
contentPane.add(btnNewButton);

```

An Object of TokenBucketAlgorithmUser is instantiated to assign the tokens to the Internal Device and start the execution of the Threads .

```

public class TokenBucketAlgorithmUser {
    int n;
    int k;
    public TokenBucketAlgorithmUser(int token, int devices) {
        n = token;
        k = devices;
    }
    public void runThreads(InternalDevice b) {
        Thread tokens = new TokenAdder(b);
        try{
            tokens.start();
            for(int i = 1; i <= k ;i++)
            {
                String str = "Source " + i;
                Thread packets = new ExternalDevice(b);
                packets.setName(str);
                packets.start();
            }
        }
        catch(Exception e1){}
    }
}

```

The Internal Device Class contains the current no of tokens, the Max Size of Token Count and the output variables : .

1. Number of packets with tokens successfully transmitted through an intermediate device.
2. Number of packets lost i.e. returned back to source from intermediate device.

It also contains the method definitions of the AddToken() and SendPacket() which are synchronized and can be accessed by the TokenAdder instance and all the instances of the ExternalDevice class respectively.

The SendPacket() method also prints the transmitting order of the successfully sent tokens and the corresponding source.

```
class InternalDevice{
    public int tokens, maxsize;
    public static int packetsSent_Success = 0;
    public static int packetsSent_Lost = 0;
    int tokenCount = 1;

    InternalDevice(int max){
        tokens = 0;
        maxsize = max;
    }

    synchronized void addToken(int n){
        if(tokens >= maxsize) return;
        tokens += 1;
        // System.out.println("Added a token. Total:" + tokens);
    }

    synchronized void sendPacket(int n,String s){
        // System.out.println("Packet of size " + n + " arrived from " + s);
        if(n > tokens){
            System.out.println("Packet of size : " + n + " Lost from " +s);
            packetsSent_Lost++;
        }
        else{
            tokens -= n;
            for(int i = tokenCount; i <= (tokenCount + n -1) ;i++)
            {
                System.out.println("Token " + i + " from " + s +" transmitted");
                packetsSent_Success++;|
            }
            tokenCount = tokenCount + n ;
        }
    }
}
```

The ExternalDevice is used to send packets of data of various sizes (randomly generated [1,10]) to the InternalDevice to check for the average inflow rate of the traffic.

```

class ExternalDevice extends Thread{
    static boolean flag = true;
    InternalDevice b;
    ExternalDevice(InternalDevice b){
        this.b = b;
    }

    public void run(){
        while(true){
            try{
                Thread.sleep(500 + (int) (Math.random()*3000));
            }
            catch(Exception e){
            }
            if(flag) {
                b.sendPacket(1 + (int) (Math.random()*9),this.getName());
            }
        }
    }
}

```

The TokenAdderClass adds 1 token at a regular interval of time , keeping in mind that the total number of tokens must not exceed the user input.

```

class TokenAdder extends Thread{
    InternalDevice b;
    TokenAdder(InternalDevice b){
        this.b = b;
    }
    public void run(){
        while(true){
            b.addToken(1);
            try{
                Thread.sleep(300);
            } catch(Exception e){}
        }
    }
}

```

When the **Close** Button of the GUI Class is pressed, the threads are halted using a flag and the output of the Internal Device variables is shown on the CapturePane.

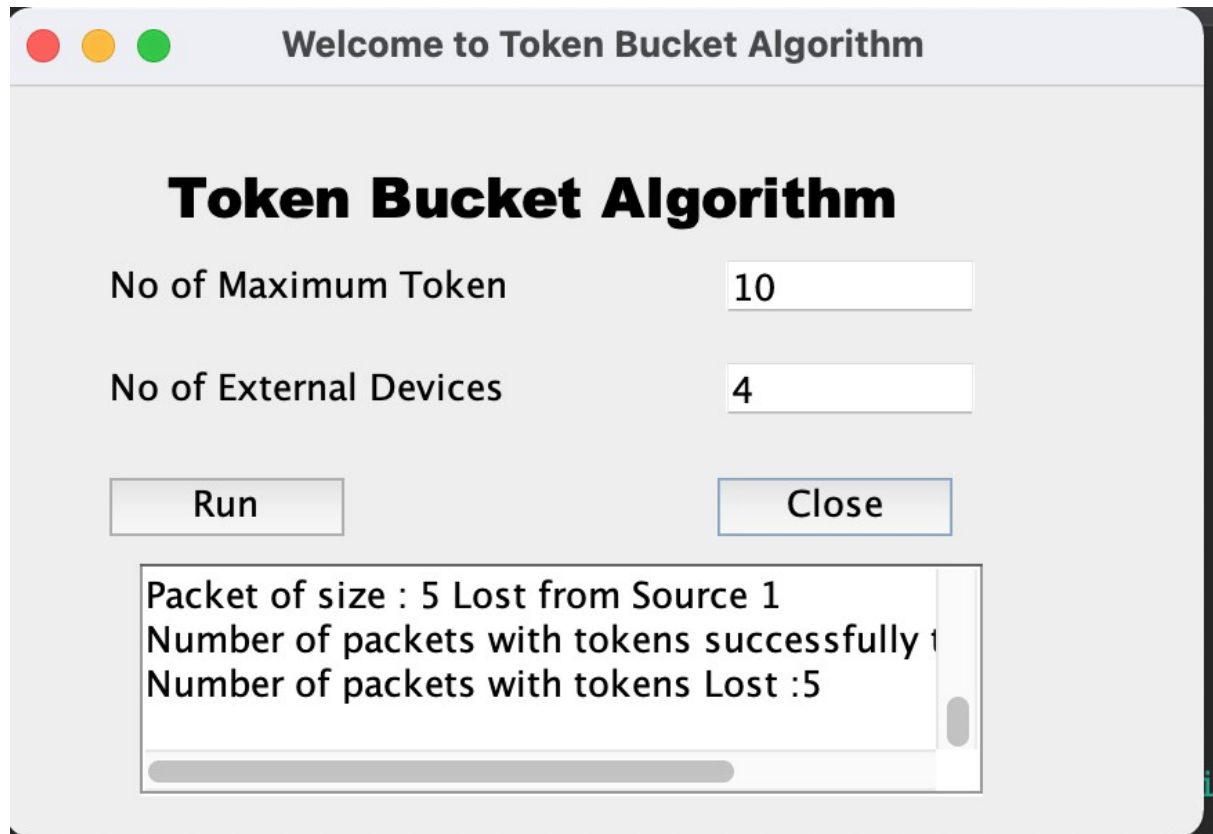
```

JButton btnClose = new JButton("Close");
btnClose.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ExternalDevice.flag = false;
        System.out.println("Number of packets with tokens successfully transmitted :"+ InternalDevice.packetsSent_Success);
        System.out.println("Number of packets with tokens Lost :"+ InternalDevice.packetsSent_Lost);
    }
});
btnClose.setBounds(258, 142, 85, 21);
contentPane.add(btnClose);

JLabel lblNewLabel_1 = new JLabel("Token Bucket Algorithm");
lblNewLabel_1.setFont(new Font("Arial Black", Font.BOLD, 20));
lblNewLabel_1.setBounds(38, 24, 305, 32);
contentPane.add(lblNewLabel_1);

```

Attached below , a test case using the input of max number of tokens = 10 and the input of external devices = 4 ,



Welcome to Token Bucket Algorithm

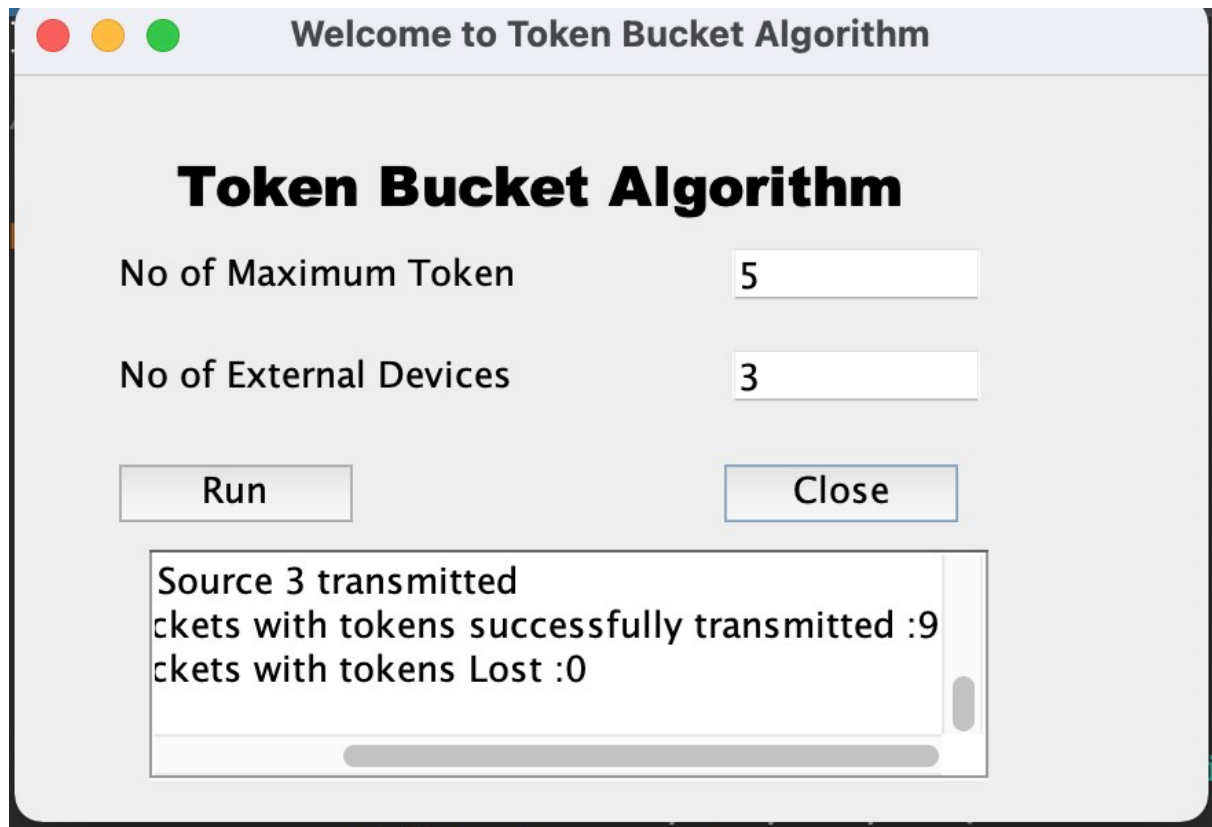
Token Bucket Algorithm

No of Maximum Token

No of External Devices

Packet of size : 5 Lost from Source 1
Number of packets with tokens successfully t
Number of packets with tokens Lost :5

Attached below , a test case using the input of max number of tokens = 5 and the input of external devices = 3 ,



OOP Principles:

Encapsulation :

We were able to use private keywords wherever possible. So, the program shows encapsulation throughout its entirety.

Favouring Composition over Inheritance :

We use several instances of ExternalDevice class, hence implementing reusability of code without explicit Inheritance .