Table of Contents

1. Introduction:

A Process used to decide which program is going to take place in execution and which process on hold is called CPU Scheduling Algorithm. Based on designed algorithms, it is made sure that CPU always has some process to execute and it's not free. CPU scheduler is responsible for selecting processes.
There are 6 types of scheduling algorithms:

I. First come First Serve (FCFS)
II. Shortest Job First (SJF)
III. Shortest Remaining Job First (SRTF)
IV. Round Robin (RR)
V. Priority Scheduling (PS)
VI. Multilevel queue Scheduling (MQ)

The algorithms used in project are FCFS, SJF, RR, PS.

IDE used: *Net Beans (version 12.2)*

2. Scope:

The Application helps in arranging series of processes in accordance with the algorithms. It help in creating efficient environment for CPU, due to which processes are signed accordingly to and CPU does not remain idle. All algorithms implemented have their own unique efficient ways of dealing with the process execution.

User can select algorithm and processes will be assigned in that manner to CPU from ready queue.

3. Hardware / Software Requirements:

3.1 Tools:

Hardware Requirements: A Windows Machine.

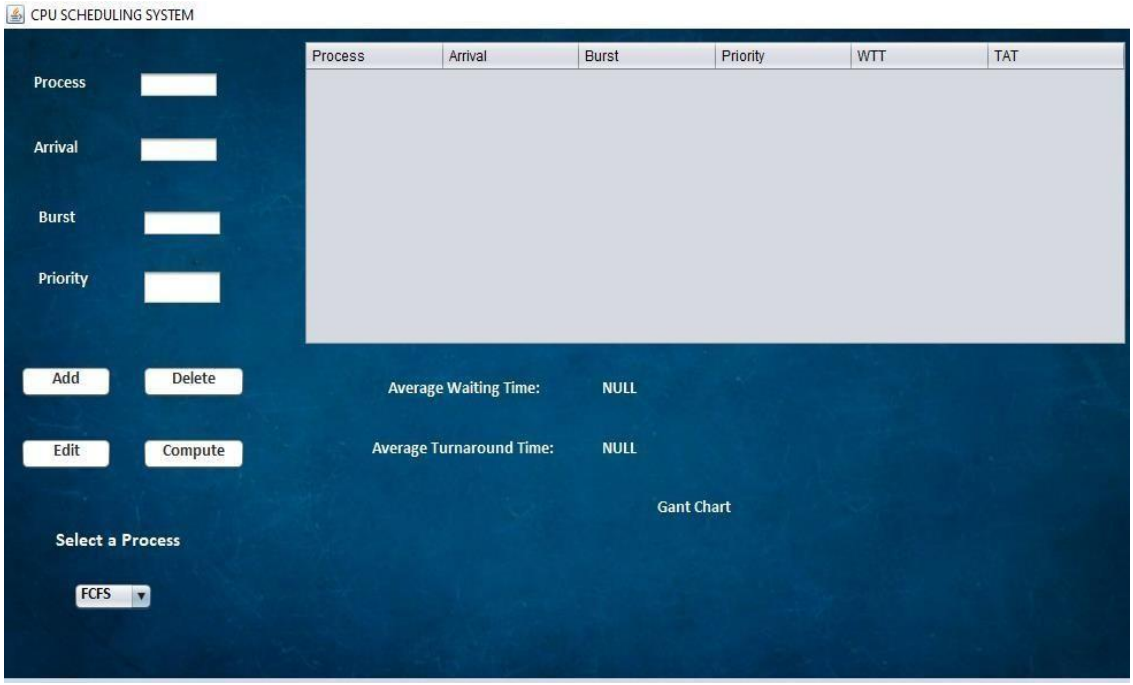
Software Requirements: Windows OS.

3.2 Libraries, IDE:

IDE: net beans (version 12.2)

Libraries javax.swing.JOptionPane, javax.swing.table.DefaultTableModel, java.util.Collections, java.util.List, java.util.ArrayList, javax.swing.JPanel

4. Module Description:



Following window is what appears after the execution of Program. The left side shows some labels and text boxes in which we enter Process name, Arrival time, Burst time and Priority (if required).

Towards the bottom of the form we see buttons that are used to add, delete, edit and execute process. Along with that is attached drop down menu which helps user select which algorithm user wants to execute.

The right side of the form contains a grid which displays data in tabular format after user enters the data. And once the user computes the values respected algorithm is applied to the processes.

Finally the portion with the heading "Gant Chart" shows graphical representation of the scheduling algorithm that is executed.

- **Execution of First Come First Serve Algorithm**:

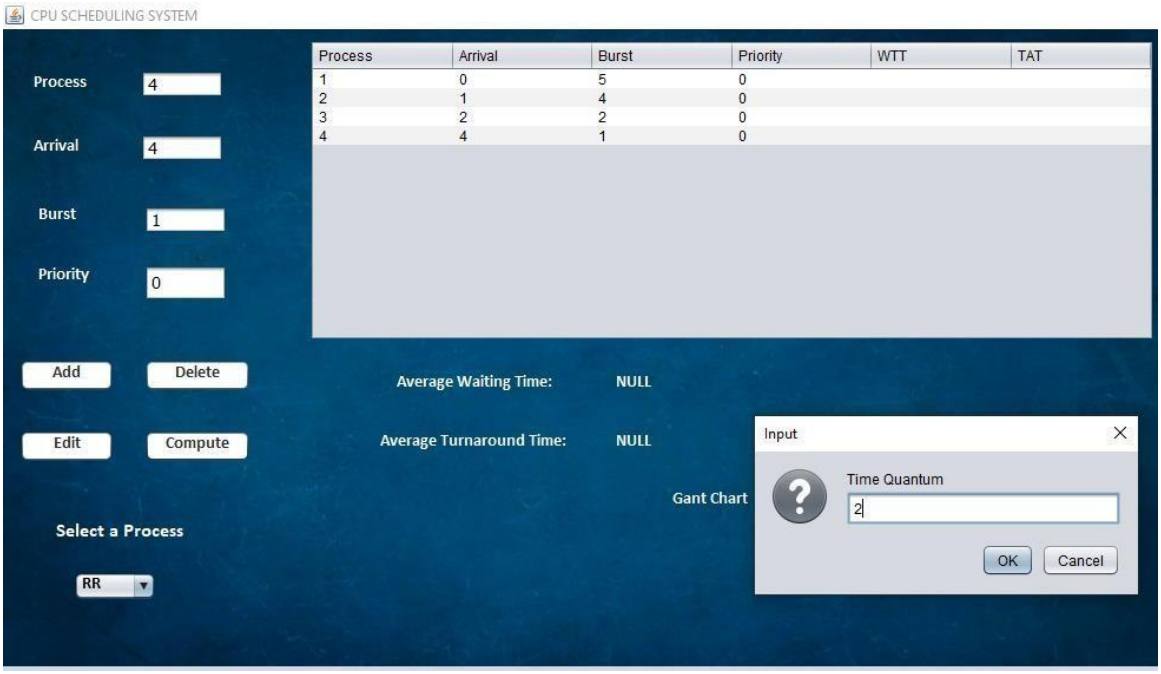
This algorithm works on the basis of arrival time and is non preemptive scheduling algorithm. The output is shown below.

- **Execution of Shortest Job First Algorithm**:

  This algorithm works on the basis of burst time. It is a non-preemptive scheduling algorithm. The output is shown below.

- **Execution of Round Robin Algorithm:**

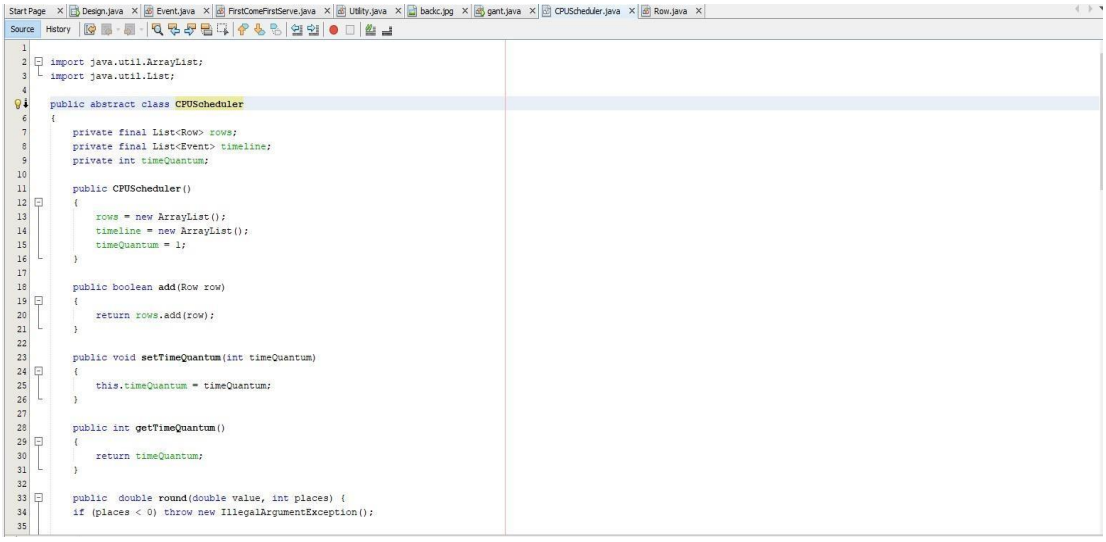
  This algorithm works on the basis of time quantum and is preemptive scheduling algorithm. The output is shown below.



- **Execution of Priority Scheduling Algorithm:**

This algorithm works on the basis of priority and is non-preemptive scheduling algorithm.The output is shown below.

5. Screen Shots:



```
import java.util.ArrayList;
import java.util.List;

public abstract class CPUScheduler
{
    private final List<Row> rows;
    private final List<Event> timeline;
    private int timeQuantum;

    public CPUScheduler()
    {
        rows = new ArrayList();
        timeline = new ArrayList();
        timeQuantum = 1;
    }

    public boolean add(Row row)
    {
        return rows.add(row);
    }

    public void setTimeQuantum(int timeQuantum)
    {
        this.timeQuantum = timeQuantum;
    }

    public int getTimeQuantum()
    {
        return timeQuantum;
    }

    public  double round(double value, int places) {
    if (places < 0) throw new IllegalArgumentException();
```

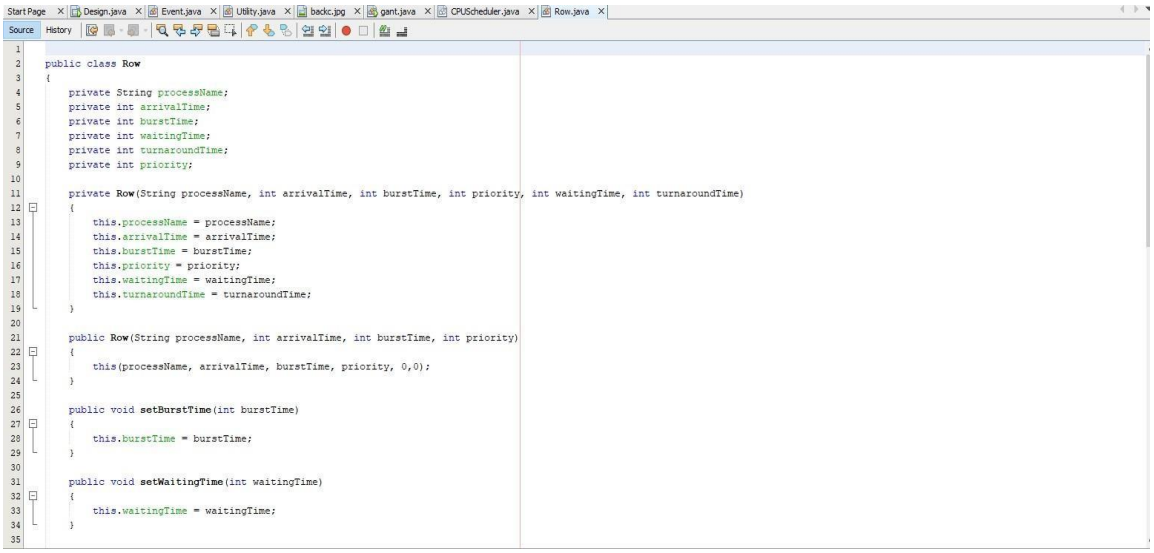
```
    long factor = (long) Math.pow(10, places);
    value = value * factor;
    long tmp = Math.round(value);
    return (double) tmp / factor;
}


public double getAverageWaitingTime()
{
    double avg = 0.0;

    for (Row row : rows)
    {
        avg += row.getWaitingTime();
    }
   // return  avg / rows.size();
   return round(avg / rows.size(),2);
}

public double getAverageTurnAroundTime()
{
    double avg = 0.0;

    for (Row row : rows)
    {
        avg += row.getTurnaroundTime();
    }

    return round(avg / rows.size(),2);
   // return avg / rows.size();
}

public Event getEvent(Row row)
{
    for (Event event : timeline)
```

```
        for (Event event : timeline)
        {
            if (row.getProcessName().equals(event.getProcessName()))
            {
                return event;
            }
        }

        return null;
    }

    public Row getRow(String process)
    {
        for (Row row : rows)
        {
            if (row.getProcessName().equals(process))
            {
                return row;
            }
        }

        return null;
    }

    public List<Row> getRows()
    {
        return rows;
    }

    public List<Event> getTimeline()
    {
        return timeline;
    }

    //hr page me istamal kr sakian yahan pr define nhi howawa

    public abstract void process();
}
```

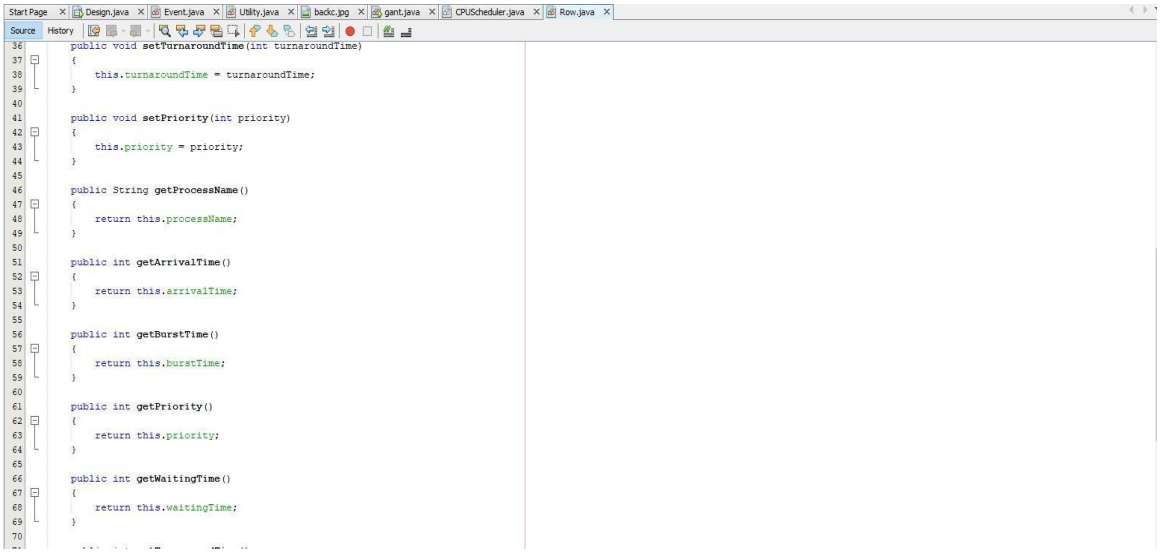
Start Page | Design.java | Event.java | Utility.java | backc.jpg | gant.java | CPUScheduler.java

Source | History

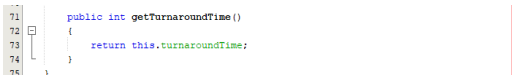
```
// rows event me save ho rhi hain (ik algo perform ho jaye tou wo rows event me aa k set ho jatin hain)
public class Event
{
    private final String processName;
    private final int startTime;
    private int finishTime;

    public Event(String processName, int startTime, int finishTime)
    {
        this.processName = processName;
        this.startTime = startTime;
        this.finishTime = finishTime;
    }

    public String getProcessName()
    {
        return processName;
    }

    public int getStartTime()
    {
        return startTime;
    }

    public int getFinishTime()
    {
        return finishTime;
    }

    public void setFinishTime(int finishTime)
    {
        this.finishTime = finishTime;
    }
}
```



```
Start Page | Design.java | Event.java | Utility.java | backc.jpg | gant.java | CPUScheduler.java | Row.java
Source | History

1
2   public class Row
3   {
4       private String processName;
5       private int arrivalTime;
6       private int burstTime;
7       private int waitingTime;
8       private int turnaroundTime;
9       private int priority;
10
11      private Row(String processName, int arrivalTime, int burstTime, int priority, int waitingTime, int turnaroundTime)
12      {
13          this.processName = processName;
14          this.arrivalTime = arrivalTime;
15          this.burstTime = burstTime;
16          this.priority = priority;
17          this.waitingTime = waitingTime;
18          this.turnaroundTime = turnaroundTime;
19      }
20
21      public Row(String processName, int arrivalTime, int burstTime, int priority)
22      {
23          this(processName, arrivalTime, burstTime, priority, 0,0);
24      }
25
26      public void setBurstTime(int burstTime)
27      {
28          this.burstTime = burstTime;
29      }
30
31      public void setWaitingTime(int waitingTime)
32      {
33          this.waitingTime = waitingTime;
34      }
35
```

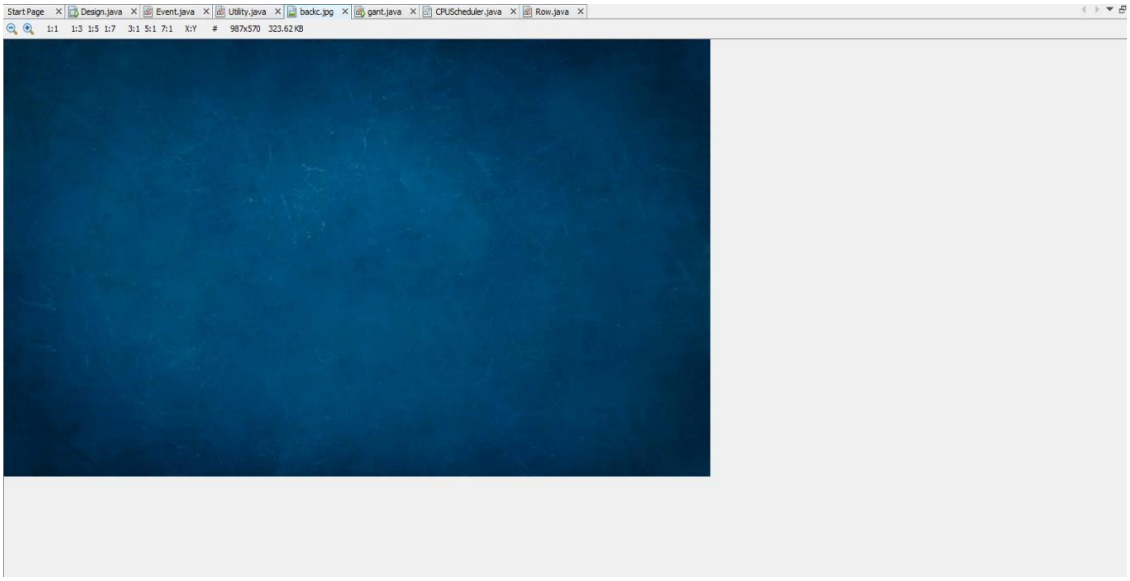


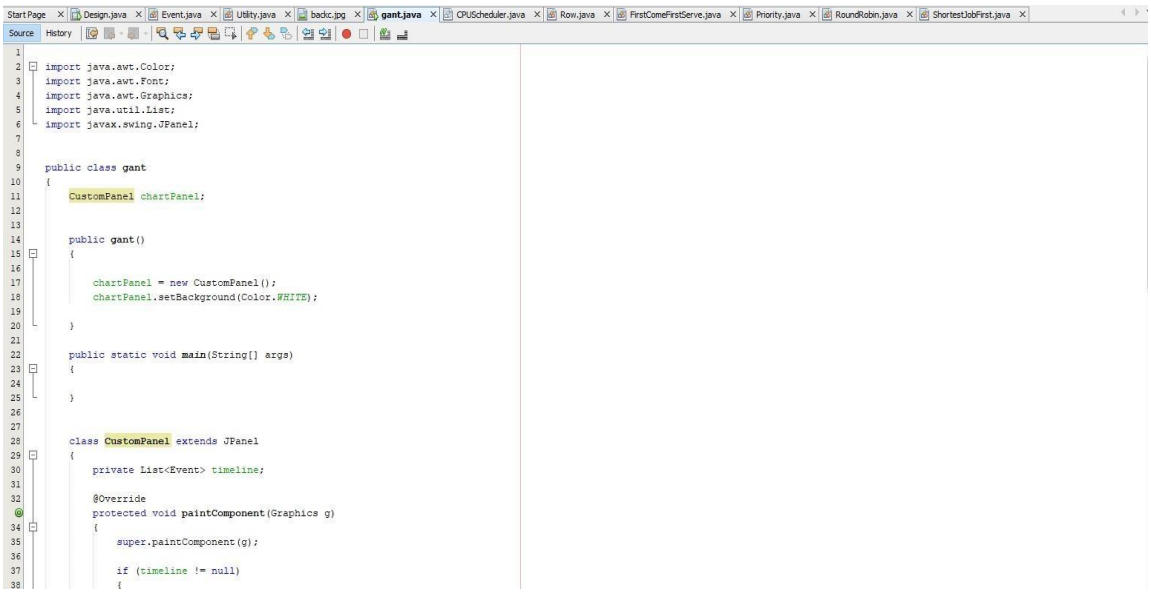
```
Start Page | Design.java | Event.java | Utility.java | backc.jpg | gant.java | CPUScheduler.java | Row.java
Source | History

36      public void setTurnaroundTime(int turnaroundTime)
37      {
38          this.turnaroundTime = turnaroundTime;
39      }
40
41      public void setPriority(int priority)
42      {
43          this.priority = priority;
44      }
45
46      public String getProcessName()
47      {
48          return this.processName;
49      }
50
51      public int getArrivalTime()
52      {
53          return this.arrivalTime;
54      }
55
56      public int getBurstTime()
57      {
58          return this.burstTime;
59      }
60
61      public int getPriority()
62      {
63          return this.priority;
64      }
65
66      public int getWaitingTime()
67      {
68          return this.waitingTime;
69      }
70
```



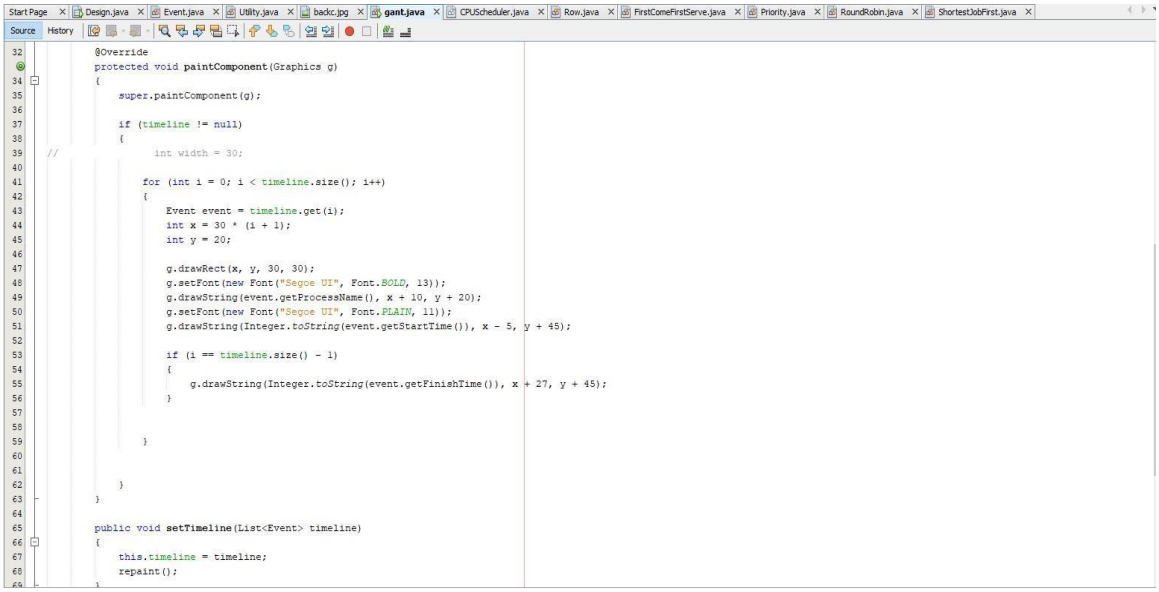
```
71      public int getTurnaroundTime()
72      {
73          return this.turnaroundTime;
74      }
```

```java
import java.util.ArrayList;
import java.util.List;

public class Utility
{
    public static List<Row> deepCopy(List<Row> oldList)
    {
        List<Row> newList = new ArrayList();

        for (Row row : oldList)
        {
            newList.add(new Row(row.getProcessName(), row.getArrivalTime(), row.getBurstTime(),row.getPriority()));
        }

        return newList;
    }
}
```

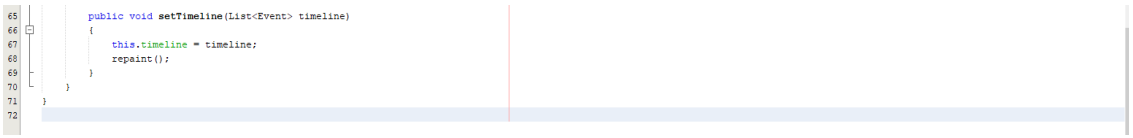




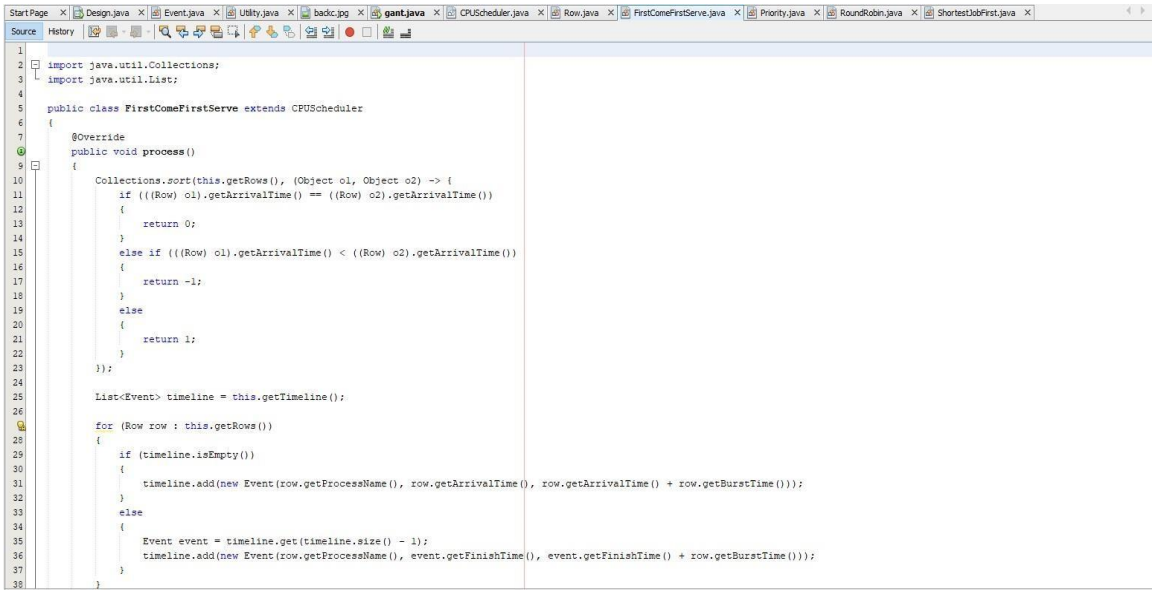
```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.util.List;
import javax.swing.JPanel;


public class gant
{
    CustomPanel chartPanel;


    public gant()
    {

        chartPanel = new CustomPanel();
        chartPanel.setBackground(Color.WHITE);

    }

    public static void main(String[] args)
    {

    }


    class CustomPanel extends JPanel
    {
        private List<Event> timeline;

        @Override
        protected void paintComponent(Graphics g)
        {
            super.paintComponent(g);

            if (timeline != null)
            {
```



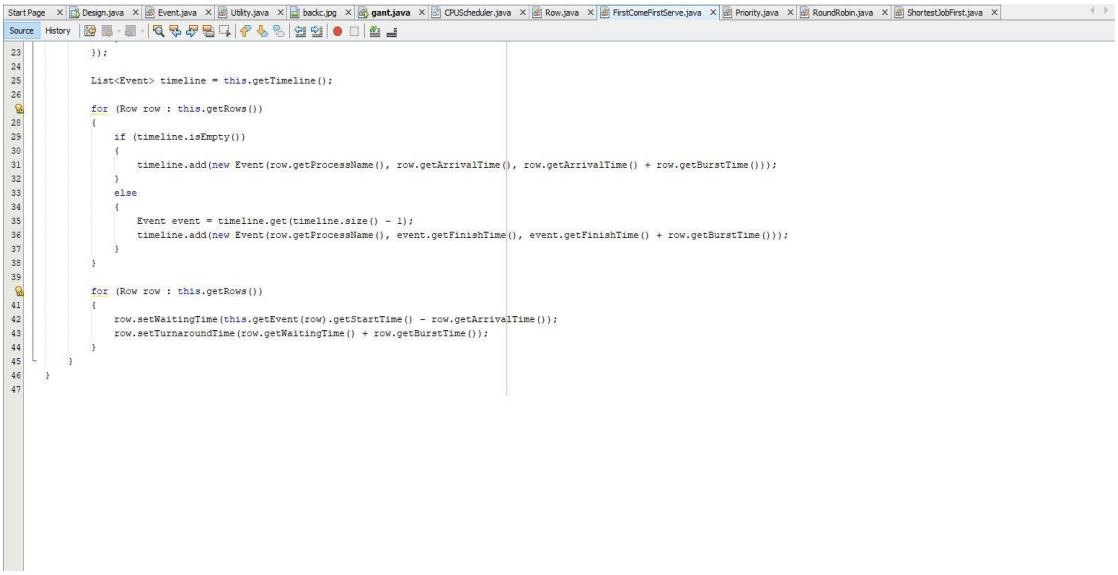
```
        @Override
        protected void paintComponent(Graphics g)
        {
            super.paintComponent(g);

            if (timeline != null)
            {
//                int width = 30;

                for (int i = 0; i < timeline.size(); i++)
                {
                    Event event = timeline.get(i);
                    int x = 30 * (i + 1);
                    int y = 20;

                    g.drawRect(x, y, 30, 30);
                    g.setFont(new Font("Segoe UI", Font.BOLD, 13));
                    g.drawString(event.getProcessName(), x + 10, y + 20);
                    g.setFont(new Font("Segoe UI", Font.PLAIN, 11));
                    g.drawString(Integer.toString(event.getStartTime()), x - 5, y + 45);

                    if (i == timeline.size() - 1)
                    {
                        g.drawString(Integer.toString(event.getFinishTime()), x + 27, y + 45);
                    }


                }


            }
        }

        public void setTimeline(List<Event> timeline)
        {
            this.timeline = timeline;
            repaint();
        }
```



```
        public void setTimeline(List<Event> timeline)
        {
            this.timeline = timeline;
            repaint();
        }
    }
}
```



```java
import java.util.Collections;
import java.util.List;

public class FirstComeFirstServe extends CPUScheduler
{
    @Override
    public void process()
    {
        Collections.sort(this.getRows(), (Object o1, Object o2) -> {
            if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
            {
                return 0;
            }
            else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
            {
                return -1;
            }
            else
            {
                return 1;
            }
        });

        List<Event> timeline = this.getTimeline();

        for (Row row : this.getRows())
        {
            if (timeline.isEmpty())
            {
                timeline.add(new Event(row.getProcessName(), row.getArrivalTime(), row.getArrivalTime() + row.getBurstTime()));
            }
            else
            {
                Event event = timeline.get(timeline.size() - 1);
                timeline.add(new Event(row.getProcessName(), event.getFinishTime(), event.getFinishTime() + row.getBurstTime()));
            }
        }
```

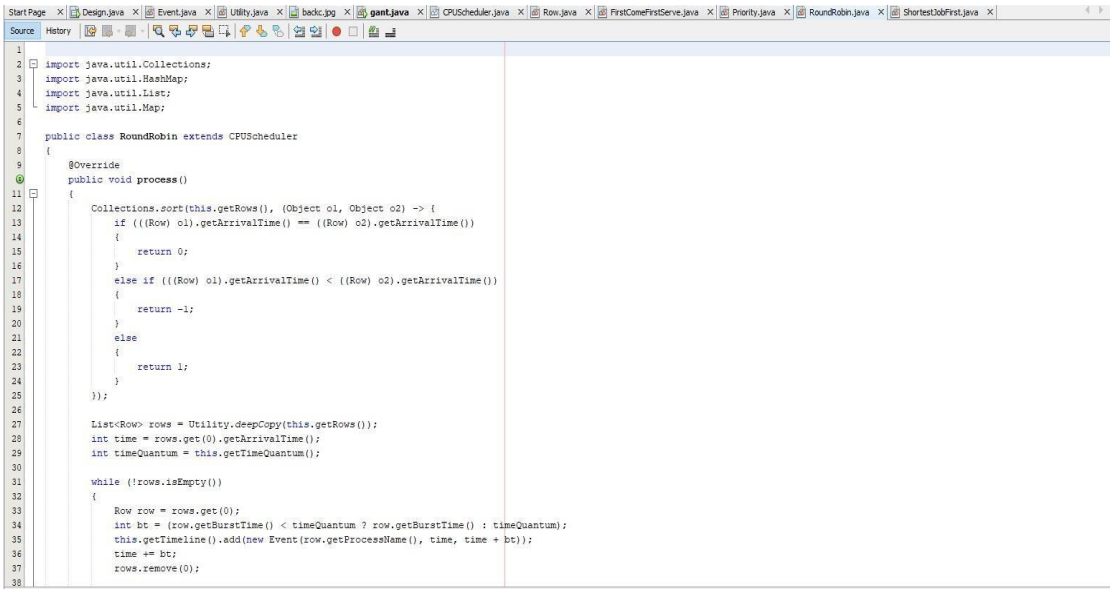


```java
        });

        List<Event> timeline = this.getTimeline();

        for (Row row : this.getRows())
        {
            if (timeline.isEmpty())
            {
                timeline.add(new Event(row.getProcessName(), row.getArrivalTime(), row.getArrivalTime() + row.getBurstTime()));
            }
            else
            {
                Event event = timeline.get(timeline.size() - 1);
                timeline.add(new Event(row.getProcessName(), event.getFinishTime(), event.getFinishTime() + row.getBurstTime()));
            }
        }

        for (Row row : this.getRows())
        {
            row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());
            row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
        }
    }
}
```

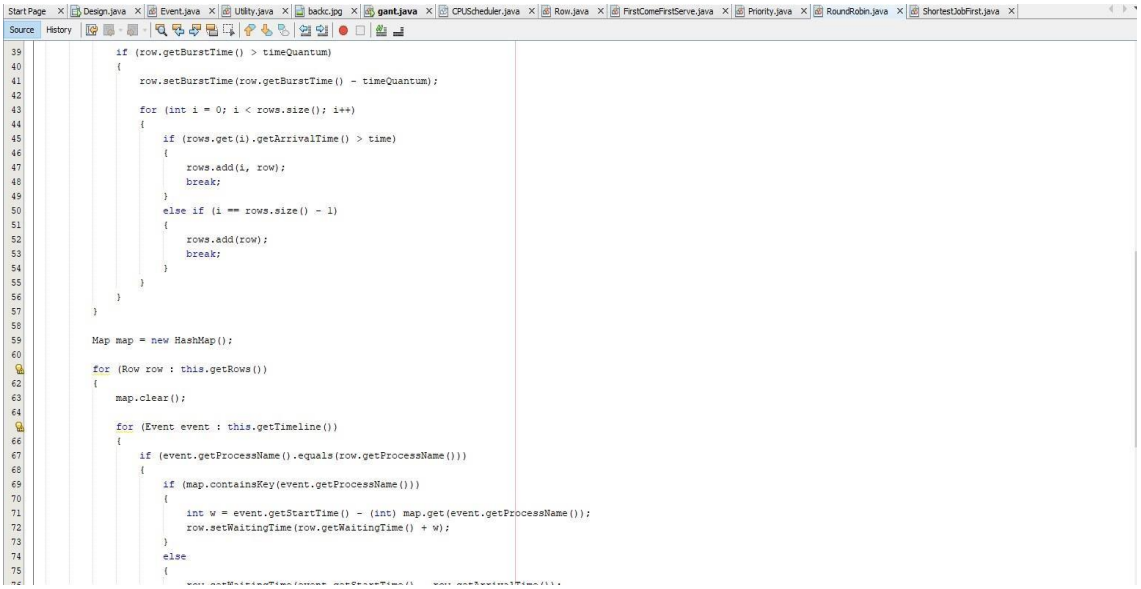
Start Page | Design.java | Event.java | Utility.java | backc.jpg | gant.java | CPUScheduler.java | Row.java | FirstComeFirstServe.java | Priority.java | RoundRobin.java | ShortestJobFirst.java

Source | History

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ShortestJobFirst extends CPUScheduler
{
    @Override
    // to sort process according to arrival time
    public void process()
    {
        // Current row jo aye usko sort karain
        Collections.sort(this.getRows(), (Object o1, Object o2) -> {
            if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
            {
                return 0;
            }
            else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
            {
                return -1;
            }
            else
            {
                return 1;
            }
        });

        List<Row> rows = Utility.deepCopy(this.getRows());
        int time = rows.get(0).getArrivalTime();

        while (!rows.isEmpty())
        {
            List<Row> availableRows = new ArrayList();

            for (Row row : rows)
            {
                if (row.getArrivalTime() <= time)
                {
                    availableRows.add(row);
                }
            }

            Collections.sort(availableRows, (Object o1, Object o2) -> {
                if (((Row) o1).getBurstTime() == ((Row) o2).getBurstTime())
                {
                    return 0;
                }
                else if (((Row) o1).getBurstTime() < ((Row) o2).getBurstTime())
                {
                    return -1;
                }
                else
                {
                    return 1;
                }
            });
            // demag chl giya ha

            Row row = availableRows.get(0);
            this.getTimeline().add(new Event(row.getProcessName(), time, time + row.getBurstTime()));
            time += row.getBurstTime();

            for (int i = 0; i < rows.size(); i++)
            {
                if (rows.get(i).getProcessName().equals(row.getProcessName()))
                {
                    rows.remove(i);
                    break;
                }
            }
        }

        for (Row row : this.getRows())
        {
            row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());
            row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
```

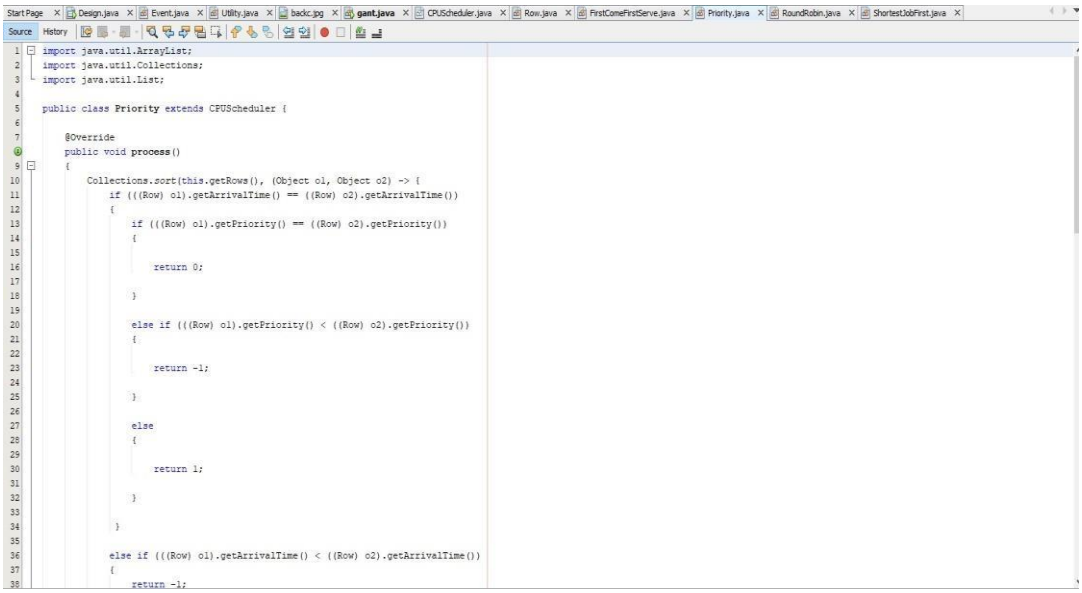


```
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class RoundRobin extends CPUScheduler
{
    @Override
    public void process()
    {
        Collections.sort(this.getRows(), (Object o1, Object o2) -> {
            if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
            {
                return 0;
            }
            else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
            {
                return -1;
            }
            else
            {
                return 1;
            }
        });

        List<Row> rows = Utility.deepCopy(this.getRows());
        int time = rows.get(0).getArrivalTime();
        int timeQuantum = this.getTimeQuantum();

        while (!rows.isEmpty())
        {
            Row row = rows.get(0);
            int bt = (row.getBurstTime() < timeQuantum ? row.getBurstTime() : timeQuantum);
            this.getTimeline().add(new Event(row.getProcessName(), time, time + bt));
            time += bt;
            rows.remove(0);
```

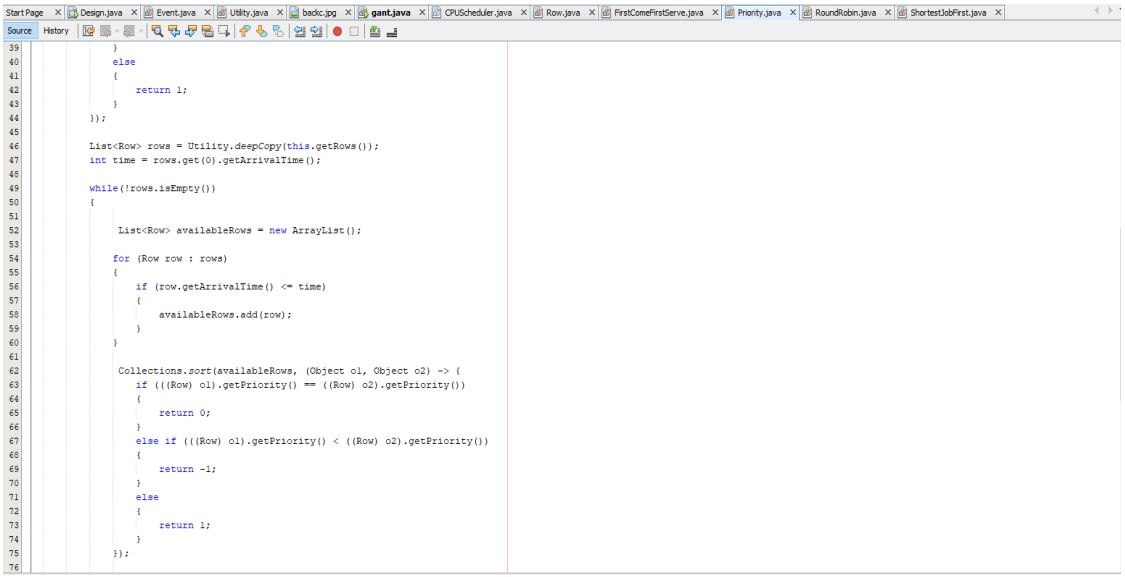


```
            if (row.getBurstTime() > timeQuantum)
            {
                row.setBurstTime(row.getBurstTime() - timeQuantum);

                for (int i = 0; i < rows.size(); i++)
                {
                    if (rows.get(i).getArrivalTime() > time)
                    {
                        rows.add(i, row);
                        break;
                    }
                    else if (i == rows.size() - 1)
                    {
                        rows.add(row);
                        break;
                    }
                }
            }
        }

        Map map = new HashMap();

        for (Row row : this.getRows())
        {
            map.clear();

            for (Event event : this.getTimeline())
            {
                if (event.getProcessName().equals(row.getProcessName()))
                {
                    if (map.containsKey(event.getProcessName()))
                    {
                        int w = event.getStartTime() - (int) map.get(event.getProcessName());
                        row.setWaitingTime(row.getWaitingTime() + w);
                    }
                    else
                    {
```

```
                        row.setWaitingTime(event.getStartTime() - row.getArrivalTime());
                    }

                    map.put(event.getProcessName(), event.getFinishTime());
                }
            }

            row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
        }
    }
}
```

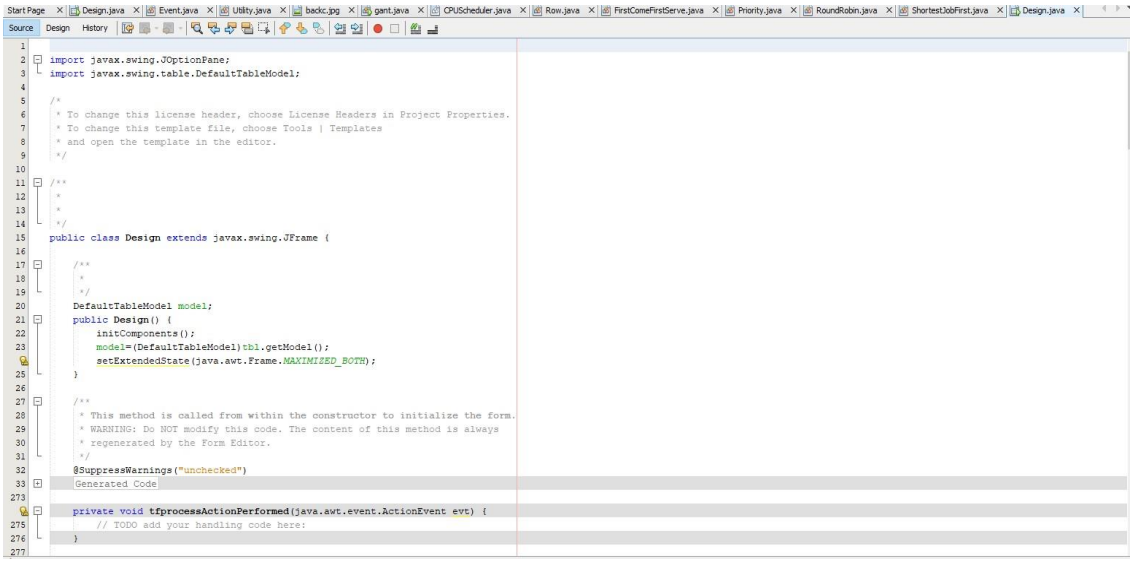


```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Priority extends CPUScheduler {

    @Override
    public void process()
    {
        Collections.sort(this.getRows(), (Object o1, Object o2) -> {
            if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())
            {
                if (((Row) o1).getPriority() == ((Row) o2).getPriority())
                {

                    return 0;

                }

                else if (((Row) o1).getPriority() < ((Row) o2).getPriority())
                {

                    return -1;

                }

                else
                {

                    return 1;

                }

            }

            else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())
            {
                return -1;
```

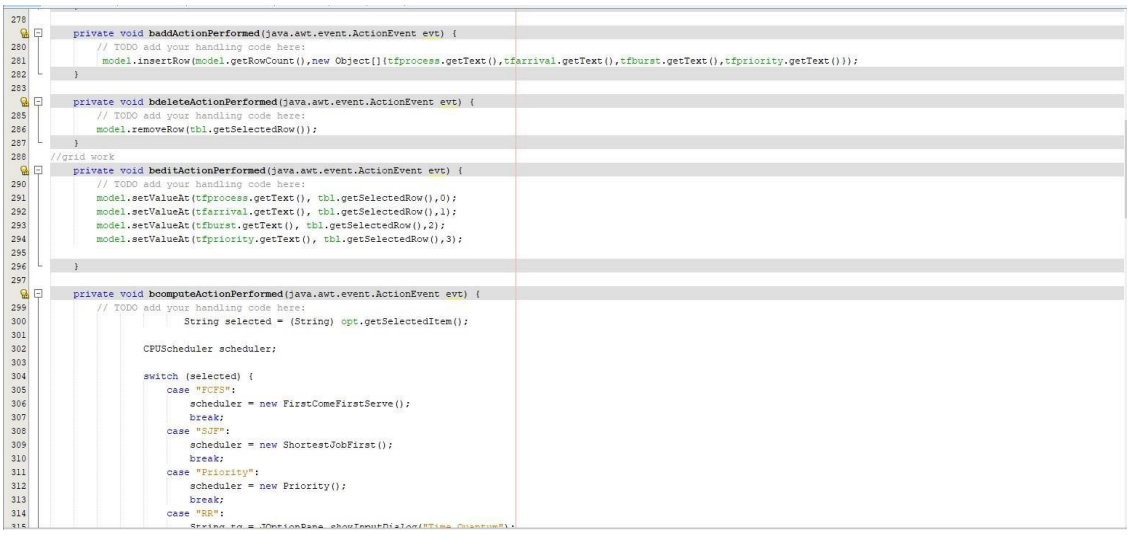


```java
            }
            else
            {
                return 1;
            }
        });

        List<Row> rows = Utility.deepCopy(this.getRows());
        int time = rows.get(0).getArrivalTime();

        while(!rows.isEmpty())
        {

            List<Row> availableRows = new ArrayList();

            for (Row row : rows)
            {
                if (row.getArrivalTime() <= time)
                {
                    availableRows.add(row);
                }
            }

            Collections.sort(availableRows, (Object o1, Object o2) -> {
                if (((Row) o1).getPriority() == ((Row) o2).getPriority())
                {
                    return 0;
                }
                else if (((Row) o1).getPriority() < ((Row) o2).getPriority())
                {
                    return -1;
                }
                else
                {
                    return 1;
                }
            });
```

```java
                }
            });

            Row row = availableRows.get(0);
            this.getTimeline().add(new Event(row.getProcessName(),time, time + row.getBurstTime()));
            time += row.getBurstTime();

            for(int i = 0; i < rows.size(); i++)
            {
                if (rows.get(i).getProcessName().equals(row.getProcessName()))
                {
                    rows.remove(i);
                    break;
                }
            }
        }

        for (Row row: this.getRows())
        {
            row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());
            row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());
        }
    }
}
```



```
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 *
 */
public class Design extends javax.swing.JFrame {

    /**
     *
     */
    DefaultTableModel model;
    public Design() {
        initComponents();
        model=(DefaultTableModel)tbl.getModel();
        setExtendedState(java.awt.Frame.MAXIMIZED_BOTH);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void tfprocessActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```



```
    private void baddActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
         model.insertRow(model.getRowCount(),new Object[]{tfprocess.getText(),tfarrival.getText(),tfburst.getText(),tfpriority.getText()});
    }

    private void bdeleteActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        model.removeRow(tbl.getSelectedRow());
    }
//grid work
    private void beditActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        model.setValueAt(tfprocess.getText(), tbl.getSelectedRow(),0);
        model.setValueAt(tfarrival.getText(), tbl.getSelectedRow(),1);
        model.setValueAt(tfburst.getText(), tbl.getSelectedRow(),2);
        model.setValueAt(tfpriority.getText(), tbl.getSelectedRow(),3);

    }

    private void bcomputeActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
                String selected = (String) opt.getSelectedItem();

            CPUScheduler scheduler;

            switch (selected) {
                case "FCFS":
                    scheduler = new FirstComeFirstServe();
                    break;
                case "SJF":
                    scheduler = new ShortestJobFirst();
                    break;
                case "Priority":
                    scheduler = new Priority();
                    break;
                case "RR":
                    String tq = JOptionPane.showInputDialog("Time Quantum");
```

```
                    if (tq == null) {
                        return;
                    }
                    scheduler = new RoundRobin();
                    scheduler.setTimeQuantum(Integer.parseInt(tq));
                    break;
                default:
                    return;
            }

            for (int i = 0; i < model.getRowCount(); i++)
            {
                String process = (String) model.getValueAt(i, 0);
                int at = Integer.parseInt((String) model.getValueAt(i, 1));
                int bt = Integer.parseInt((String) model.getValueAt(i, 2));
                int priority = Integer.parseInt((String) model.getValueAt(i, 3));
               // in a form of int
                scheduler.add(new Row(process, at, bt,priority));
            }

            scheduler.process();

            for (int i = 0; i < model.getRowCount(); i++)
            {
                String process = (String) model.getValueAt(i, 0);
                Row row = scheduler.getRow(process);
                model.setValueAt(row.getWaitingTime(), i, 4);
                model.setValueAt(row.getTurnaroundTime(), i, 5);
            }

            wtresult.setText(Double.toString(scheduler.getAverageWaitingTime()));
            tatresult.setText(Double.toString(scheduler.getAverageTurnAroundTime()));


            gantt.chartPanel.setTimeline(scheduler.getTimeline());
```

```
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JComboBox<String> opt;
    private javax.swing.JScrollPane scroll;
    private javax.swing.JLabel tatresult;
    private javax.swing.JTable tbl;
    private javax.swing.JTextField tfarrival;
    private javax.swing.JTextField tfburst;
    private javax.swing.JTextField tfpriority;
    private javax.swing.JTextField tfprocess;
    private javax.swing.JLabel wtresult;
    // End of variables declaration
}
```

```
private void tblMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    tfprocess.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 0)));
    tfarrival.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 1)));
    tfburst.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(), 2)));
    tfpriority.setText(String.valueOf(model.getValueAt(tbl.getSelectedRow(),3)));

}

private void optActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            new Design().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton badd;
private javax.swing.JButton bcompute;
private javax.swing.JButton bdelete;
private javax.swing.JButton bedit;
private gant gant1;
private javax.swing.JComboBox<String> jComboBox1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
```